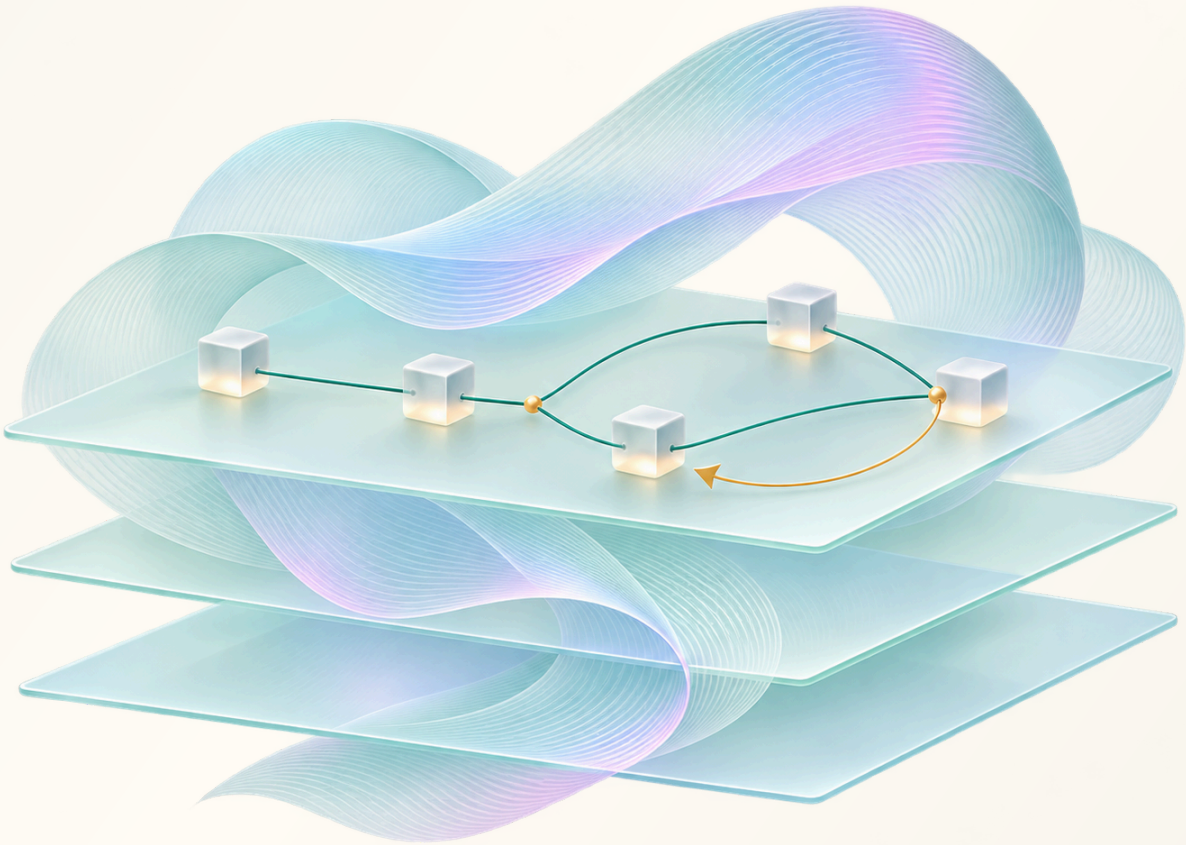
1id1.net

DIGITAL EDITIONS / EN

Cloud Services and AI PaaS



CLOUD · AGENT PAAS · ADVANCED AI

Engineering AI-Native Applications with Azure and China's Leading Cloud Platforms

Wuming Pan

1id1:core:20261122-0002@v1

1id1.net

Cloud Services and AI PaaS

Engineering AI-Native Applications with Azure and China's Leading Cloud Platforms

Wuming Pan

English edition

Copyright and edition

Cloud Services and AI PaaS

Author	Wuming Pan
Language and scope	English · Six parts · Twenty chapters · 126 figures
Publishing platform	1id1.net
Edition and date	First edition · 22 November 2026
Book object PID	1id1:core:20261122-0002
Edition citation PID	1id1:core:20261122-0002@v1
Companion edition	Simplified Chinese edition 1id1:core:20261122-0001@v1
Publication mark	1id1.net Orange-Red Publication Mark 1id1:core:20261122-0003@v1
Cover artwork	Floating Glass Platform and Mathematical Ribbon 1id1:core:20261122-0004@v1
AI use disclosure	The author used ChatGPT Pro to assist with writing this book.

© 2026 Wuming Pan
Product names and trademarks belong to their respective owners.

Preface

Advances in model capability, inference efficiency, and agent execution bring AI application design into contact with multiple layers of infrastructure. The quality, latency, and cost of a task depend on how inference, context, data access, and execution environments work together. Tasks that continue across services also connect application state and identity with changes to cloud resources. Cloud Services and AI PaaS examines these relationships: how cloud platforms participate in the architecture of AI systems, and how applications establish their own operating behavior using platform capabilities.

Platform abstractions influence how applications are built and how engineering work is divided. Data services, inference interfaces, and execution mechanisms together shape an application's interaction patterns, workflows, and autonomous execution. The extent to which inference is managed affects a team's access to scheduling and capacity decisions. An agent runtime's treatment of state shapes the options for task recovery, migration, and version changes. Throughout the book, we examine what the platform takes on, what remains under application control, and how the two sides fit together in a working architecture.

Resource-level thinking provides the book's analytical thread. It grounds architectural choices in resource identity, dependencies, and lifecycle, and connects configuration and change in the control plane with execution paths and consequences in the data plane. This perspective applies to compute, networking, and storage, and extends to model deployments, retrieval indexes, and agent platforms. It allows these components to be considered together while preserving their distinct management and execution boundaries.

The infrastructure chapters therefore contribute directly to the discussion of AI architecture. Network paths and data placement influence how services cooperate. Identity and isolation mechanisms constrain access to data and tools, while persistent state and delivery mechanisms affect how tasks continue over time. Reliability, observability, and cost must be assessed in relation to these mechanisms. As cloud platforms encapsulate implementation details, architectural judgment increasingly concerns the relationships between services and the scope of the guarantees they provide.

Developments in AI infrastructure make these relationships particularly consequential. Memory management and request scheduling within inference systems affect the tradeoffs among capacity, throughput, and latency. Platform choices about deployment and billing then influence how those tradeoffs appear in an application's costs. The book examines how these implementation constraints enter application architecture through service interfaces, deployment models, and billing arrangements. That connection informs its treatment of service selection, capacity planning, and system design.

AI PaaS and Agent PaaS can be examined as part of this evolution in platform abstraction. This book uses Agent PaaS to discuss how agent runtimes, session and task state, tool integration, evaluation, and operational management are organized as platform capabilities. Particular attention goes to where each kind of state resides, which boundaries execution crosses, what the platform can recover, and which business consequences the application must handle. Continuing a conversation, recovering a task, and compensating for an external operation each require mechanisms suited to the problem. These distinctions matter when deciding how much execution responsibility to place with a platform.

When cloud capabilities become agent tools, the platform becomes part of the agent's action space. Tool interfaces must express the target and scope of an operation. The execution system must enforce authorization and account for the repeated effects that retries can produce. It should be possible to connect an action proposed by a model with the call actually issued and the resulting change to a resource. This creates a concrete engineering task across the control plane, tool design, and agent runtime: connecting the model's choice of action to a system with explicit permissions and execution semantics.

Evaluation, observability, and FinOps run through these choices. Changes to a model, prompt, retrieval index, or tool implementation may alter the execution path of the same task. Deciding whether to release a change requires connecting task quality with execution records and examining the associated latency, resource consumption, and expenditure. This brings model performance, system behavior, and operating cost into the same engineering decision. It also provides a basis for revisiting platform choices as workloads change and evidence accumulates in operation.

I use Azure as the main reference so that resource models, identity, and platform services can be examined continuously within one implementation. Alibaba Cloud, Tencent Cloud, and Volcengine provide the comparative perspective. The same engineering objective can lead to different architectures when product boundaries, service combinations, and operating mechanisms differ. The purpose of comparison is to establish how those differences affect application design and which judgments can be carried from one platform to another.

The book begins with resource organization and governance, proceeds through application platforms, containers, Kubernetes, and continuous delivery, and develops the discussion of observability, cost management, and AI platforms. It then examines retrieval, agent platforms, and the exposure of cloud capabilities as tools. This sequence provides a continuous reference for relationships across system layers. The experiments in individual chapters test mechanisms and their boundaries. Readers can move between chapters according to the architectural questions raised by their own systems.

Models and platforms will continue to evolve, and systems need room to accommodate that evolution. I hope this book contributes an analytical approach to design discussions, helping us explain how a new model or platform changes an application's composition, execution, and resource requirements, and how an existing system can accommodate those changes. The book is organized around the relationship between model capabilities, the guarantees a platform provides, and the behavior of an application in operation.

Contents

Preface ii

PART I

Cloud Services as Programmable Control Planes viii

01 Cloud Services as a Programmable Control Plane 1

02 Resource-Level Thinking: Control Plane, Data Plane, and Lifecycle 24

03 Multicloud Architecture: Shared Responsibility, Platform DNA, and Engineering Intent 48

PART II

Foundational Cloud Resource Control Planes 72

04 Resource Organization, Tags, and Infrastructure as Code 73

05 Identity and Least Privilege 109

06 Compute, Elasticity, and Reliability 143

07 Networking, Private Connectivity, and Exposure Control 178

08 Storage, Data Lakes, and Event-Driven Data Paths 215

PART III

Application and Cloud-Native Control Planes 248

09 Application PaaS, Functions, Events, and API Management 249

10 Containers, Registries, and Software Supply Chains 291

11 Managed Kubernetes and Application Definition 330

12 Continuous Delivery, GitOps, and Change Governance 376

Contents, continued

PART IV

Operations and Value Control Planes 412

13 Observability as a Managed Feedback Loop 413

14 FinOps as a Cloud Control Plane 447

15 Security Posture, Compliance, and Organizational Governance 482

PART V

AI PaaS Control Planes 517

16 From Data+AI Platforms and ML PaaS to AI PaaS 518

17 From Model APIs to AI PaaS Resources 543

18 RAG Is Not a Vector Database 576

19 Agent Platforms as Governed Cloud Resources 608

PART VI

From Cloud Capabilities to Governed Agent Tools 647

20 Turning Cloud Capabilities into Agent Tools 648

Subject index 679

Cloud Services as Programmable Control Planes

What happens between a deployment request and a change to a cloud resource? Follow the request through the control plane, inspect one resource's identity and lifecycle, then carry the same engineering objective across providers and compare who is responsible for the result.

01 Cloud Services as a Programmable Control Plane

02 Resource-Level Thinking: Control Plane, Data Plane, and Lifecycle

03 Multicloud Architecture: Shared Responsibility, Platform DNA, and Engineering Intent

Part I / Chapter 1

Cloud Services as a Programmable Control Plane

Why Industrial Software Engineering Begins with Managed Resources

Cloud computing becomes an engineering discipline when infrastructure, platform, data, and AI capabilities are exposed as addressable resources with identity, policy, lifecycle, telemetry, and cost.

In This Chapter

- Why the remote-server metaphor is too small for modern cloud engineering
- How the essential characteristics of cloud computing become control-plane obligations
- How control-plane operations differ from data-plane operations
- Why a deployable application is a resource graph rather than a server
- How cloud services productize operational responsibilities across IaaS, PaaS, and AI PaaS
- Why Azure provides a coherent reference control plane for teaching and comparison
- How to translate engineering intent across Alibaba Cloud, Tencent Cloud, and Volcengine without forcing false equivalence
- How to inspect a resource graph and trace a lifecycle event in an independent capability check

Ask a new cloud user what the cloud is and the answer is often some version of "someone else's computer." The phrase is memorable because it punctures marketing language. It is also an incomplete engineering model. It explains where the hardware is and who owns it, but it says almost nothing about how a cloud capability is requested, authorized, connected, observed, changed, charged, or retired.

A virtual machine rented by the hour is useful. A virtual machine represented as a typed resource, created through an API, attached to a network policy, assigned an identity, queried in a resource graph, protected by a lock, measured by telemetry, attributed to a cost center, and deleted through a recorded lifecycle operation is something more. It is part of an industrial software engineering system.

That system is the cloud control plane. The control plane is the set of management APIs, resource models, identity checks, policies, lifecycle operations, inventory services, activity records, quotas, and cost scopes through which cloud capabilities become governable. The portal is one client of this system. Command-line tools, SDKs, infrastructure-as-code engines, deployment pipelines, and future agents are other clients.

This distinction matters even more for AI-native applications. A model endpoint by itself is a callable capability. A model deployment with a resource identity, quota, network boundary, monitoring, evaluation records, policy, and cost scope belongs to an AI PaaS control plane. The same progression applies to knowledge bases, retrieval systems, tools, and agents.

This chapter establishes the book's governing idea: cloud services should be understood as a programmable control plane for software engineering. Later chapters formalize the resource-level contract, examine the foundational and application control planes, and extend the model to data, models, knowledge, and agent platforms.

The chapter thesis

Cloud value does not begin with remote hardware. It begins when operational responsibility is exposed as a programmable, governable, and economically visible resource lifecycle.

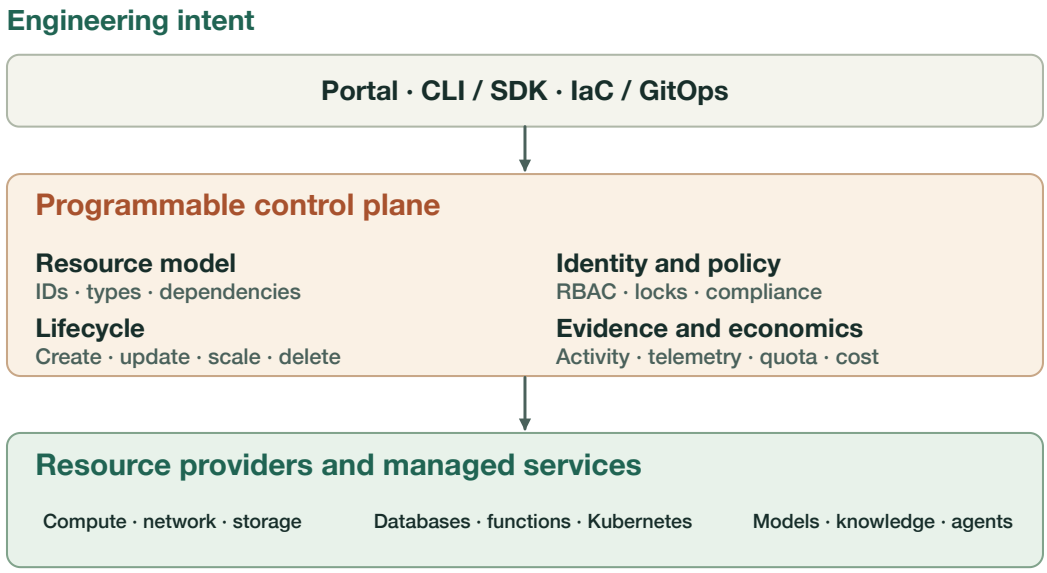


Figure 1-1 Engineering intent enters the cloud through a programmable control plane, not through a product catalog alone.

1.1 The Server-Rental Story Is Too Small

The server-rental model is a useful historical entry point. It explains why a team can obtain compute without purchasing racks, negotiating datacenter space, or installing hypervisors. It also explains elasticity at a basic level: acquire capacity when needed and release it when demand falls. But the model becomes misleading when it suggests that the cloud is merely a cheaper or faster datacenter.

Consider a small web API. If the architecture is described as "an application on a VM," the sentence hides almost every production decision. Which account owns the VM? Which region and failure domain contain it? Which image and size were chosen? Which network interface, subnet, route, security rule, public address, and DNS record make it reachable? Which identity lets the application access storage or secrets? Which policy forbids a public endpoint? Which monitor detects failure? Which backup or rebuild path exists? Which tag assigns cost to a team? Which operation record shows who changed the instance last night?

The VM is therefore not the system. It is one node in a graph of managed resources and responsibilities. The engineering value of the cloud appears in four surrounding systems:

1. A resource system. Capabilities have types, identifiers, properties, dependencies, scopes, and lifecycle states.

2. An authority system. Humans, applications, pipelines, and services receive permissions over explicit scopes rather than informal machine access.

3. An operating system for change. Provisioning, scaling, configuration, policy evaluation, audit, deletion, and recovery are mediated by APIs and controllers.

4. An economic system. Usage is metered, attributed, budgeted, forecast, optimized, and connected to technical ownership.

Industrial-grade engineering is not defined only by workload size. A small system can be industrial-grade if its changes are reproducible, its authority is explainable, its failure boundaries are known, its evidence is inspectable, and its cost is attributable. A large system can remain artisanal if those properties depend on undocumented operator knowledge.

The control plane is not the portal

A portal is a graphical client. The same resource operation should be expressible through an API, CLI, SDK, or declarative deployment. Treat screenshots as learning aids, not as the definition of the platform.

1.2 From Cloud Characteristics to Engineering Consequences

The NIST definition of cloud computing remains useful because its five essential characteristics describe more than commercial packaging. They imply a particular operating model: consumers can provision on demand; capabilities are available over networks; providers pool resources; capacity expands and contracts; and usage is measured. NIST also separates IaaS, PaaS, and SaaS as service models and public, private, community, and hybrid arrangements as deployment models. [1] [2]

For an engineer, each characteristic creates a control-plane consequence. On-demand self-service requires a programmable request path. Resource pooling requires isolation, placement, quota, and scheduling. Rapid elasticity requires desired-state policies and health signals. Measured service requires metering and attribution, not merely an invoice at month end. Broad network access requires explicit endpoint and trust-boundary design.

Table 1-1 Cloud characteristics translated into control-plane obligations.

Cloud characteristic	Engineering consequence	Evidence in a mature control plane
On-demand self-service	A consumer can request, change, and release capabilities without a manual provisioning ticket.	Management API, declarative template, deployment record, lifecycle status

Cloud characteristic	Engineering consequence	Evidence in a mature control plane
Broad network access	Reachability must be designed for public, private, hybrid, and service-to-service paths.	Endpoints, network interfaces, routes, DNS, firewall or security-group policy
Resource pooling	Shared capacity must preserve isolation, placement, quota, and ownership boundaries.	Region/zone, tenancy scope, quota, scheduler decision, resource identifier
Rapid elasticity	Capacity becomes a policy-controlled state rather than a one-time purchase.	Scale rule, desired capacity, health signal, activity history, limit
Measured service	Technical usage must become transparent to both engineering and financial owners.	Meter, usage record, tag, budget, cost allocation, forecast, anomaly

This translation is important because not every hosted environment delivers the full cloud operating model. A virtualized server obtained through a multiday ticket, with fixed capacity, no programmatic lifecycle, and no usage transparency may be remote infrastructure, but it provides only part of the cloud experience. NIST SP 500-322 makes the same distinction operationally: a computing capability should be evaluated against the essential characteristics rather than accepted as cloud merely because it is off premises. [2]

The strongest cloud services make the five characteristics composable. A team can create storage on demand, reach it through a private endpoint, place it under an account and policy boundary, scale capacity independently of a VM, and attribute usage to a product. The service is valuable not only because it stores bytes, but because it exposes storage as an operable contract.

1.3 The Control Plane Is the Engineering Surface

Cloud operations divide into two broad categories. The control plane manages a resource: create it, configure it, authorize access, attach policy, inspect status, change capacity, or delete it. The data plane uses the capability exposed by a resource instance: log in to a VM, upload a blob, execute a database query, send an event, invoke a model, or retrieve knowledge. Azure Resource Manager handles Azure control-plane operations, while data-plane requests normally go to a resource-specific endpoint. [4] [5]

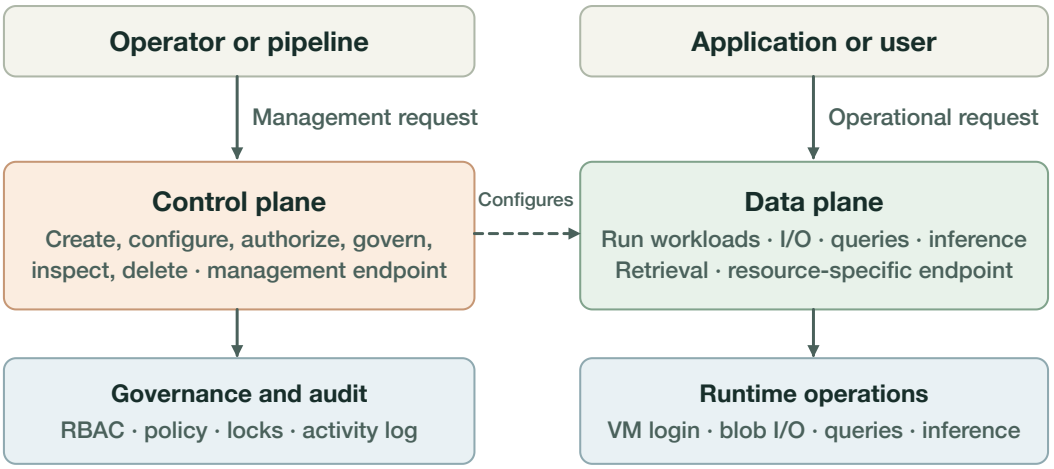


Figure 1-2 Control-plane and data-plane requests have different endpoints, permissions, evidence, and failure modes.

Table 1-2 The same cloud service exposes different management and operational paths.

Resource	Typical control-plane operation	Typical data-plane operation
Virtual machine	Create the VM, select size, attach identity, stop or delete the resource.	Connect to the guest OS and run an application.
Object storage	Create the account or bucket, set redundancy, networking, encryption, or lifecycle policy.	Read, write, list, or delete objects.
Managed database	Create an instance, select tier, configure backup, private access, or failover.	Issue SQL or database-native queries and transactions.
Managed model	Create a model deployment, assign quota, identity, network, monitoring, or policy.	Send inference requests and receive model output.
Knowledge resource	Define sources, index schema, ingestion, authorization, versions, and evaluation.	Submit retrieval requests and receive grounding evidence.

The distinction prevents several common security and reliability errors. A management lock that prevents deletion of a storage account does not, by itself, prevent a principal with data-plane permission from deleting blobs. A user who can query a database does not necessarily need permission to resize or delete the database resource. A model caller does not

automatically need permission to change the model deployment. Control-plane and data-plane authorization must therefore be designed separately, even when the same identity system participates in both. [5]

The planes also fail differently. A temporary control-plane disruption may prevent a new resource from being created while an existing data plane continues serving traffic. Conversely, a healthy management API does not prove that an application endpoint, database, or model inference path is healthy. Operational evidence must identify which plane is being observed.

1.3.1 One management system, several clients

A portal click, a CLI command, an SDK call, and an infrastructure-as-code deployment should be understood as different clients of a management system. They differ in ergonomics, repeatability, reviewability, and automation, but they ultimately request control-plane operations from resource providers.

Table 1-3 Management clients and their engineering strengths.

Client	Best use	Primary limitation
Portal	Discovery, learning, one-off inspection, visual diagnostics.	Manual intent is difficult to review, reproduce, test, and diff.
CLI	Interactive automation, investigation, scripting, reproducible commands.	Long scripts can become imperative and order-dependent.
SDK / REST	Product integration and custom control-plane applications.	The application must own retries, state, authorization, and API-version discipline.
IaC	Declarative creation, review, versioning, environment replication, and controlled deletion.	Not every day-two operation maps cleanly to a static declaration.
GitOps / deployment controller	Continuous reconciliation of declared application or platform state.	Requires a clear source of truth and careful separation of generated and human-owned state.

ClickOps is not inherently wrong. It is often the fastest way to discover a resource model. The problem begins when the portal becomes the only record of intent. A production change should leave durable evidence: a reviewed declaration or command, an authenticated management request, an activity event, a resulting resource state, and a cost or quota consequence.

1.3.2 Resource providers turn APIs into product boundaries

A cloud control plane is not a single monolithic program. It routes requests to resource providers. In Azure, a provider namespace such as Microsoft.Compute, Microsoft.Storage, Microsoft.Network, Microsoft.Web, or Microsoft.ManagedIdentity defines resource types and management operations for a service domain. Azure Resource Manager authenticates and authorizes the control-plane request and then passes it to the provider that implements the operation. [4] [6]

This provider model explains why the cloud feels both unified and heterogeneous. The platform can apply common identity, policy, tags, activity records, and deployment semantics while each service preserves a domain-specific schema and lifecycle. A VM, a virtual network, a search service, and a model deployment do not have identical properties, but they participate in a common management fabric.

1.4 An Application Is a Resource Graph, Not a Server

A resource inventory is useful; a resource graph is more powerful. An inventory answers what exists. A graph also captures containment, attachment, dependency, exposure, ownership, and evidence relationships. The graph changes the unit of reasoning from an isolated product to an operable system.

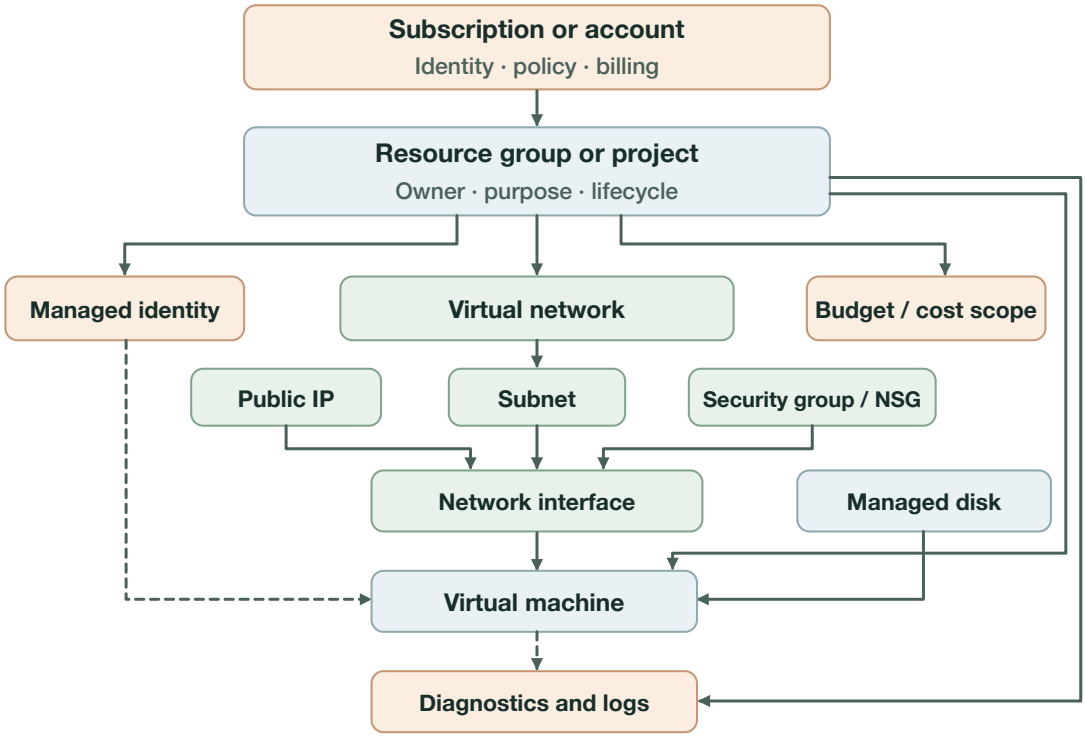


Figure 1-3 Even a minimal VM-based application is a graph of compute, network, identity, observability, and cost resources.

The graph supports questions that a server list cannot answer:

- Which public address, route, security rule, or load balancer exposes the workload?
- Which identity can read a secret, storage object, model endpoint, or database?
- Which resources share one lifecycle and should be deployed or deleted together?
- Which owner, environment, application, and cost center should receive the spend?
- Which activity event explains an unexpected change?
- Which dependent resources remain after the main workload is removed?

Azure Resource Graph extends Azure Resource Management by supporting resource exploration across subscriptions at scale. It can query resource types and properties, join related information, and provide a governance-oriented view that is difficult to reconstruct by visiting individual service consoles. Azure Activity Log records subscription-level management events and can be routed to Log Analytics, Event Hubs, or Storage for longer analysis and retention. [7] [8]

Table 1-4 Graph question, engineering risk, and control-plane evidence.

Question	Risk if unanswered	Useful control-plane evidence
What owns this resource?	Orphaned resources, unclear authority, unallocated cost.	Subscription/account, resource group/project, tags, creator, role assignments
What can reach it?	Unexpected public exposure or lateral movement.	VNet/VPC, subnet, endpoint, route, security group, DNS, flow log
What does it depend on?	Partial deletion, hidden blast radius, failed recovery.	Resource IDs, attachments, deployment stack, dependency graph
Who changed it?	Slow incident response and weak accountability.	Activity/audit event, caller identity, request ID, timestamp, source
Can it be recreated?	Configuration drift and manual disaster recovery.	IaC definition, image/version, parameters, deployment history
What does it cost?	Optimization without ownership or business context.	Meter, tag, billing scope, budget, usage export, unit-cost calculation

PaaS does not remove the resource graph; it compacts it. A managed web application may hide guest operating systems and patching, but it still has a hosting plan or capacity setting, identity, endpoint, configuration, certificates, network integration, diagnostic settings, deployment history, quota, and cost. A serverless function hides idle servers but adds triggers, bindings, concurrency limits, permissions, logs, and event sources. Managed abstraction reduces what the customer operates, not what the system must model.

1.5 Cloud Services Productize Responsibilities

The most consequential cloud abstraction is not virtualization. It is the productization of operational responsibility. Tasks that once depended on local procedures become service contracts with APIs, platform automation, measurable limits, and evidence. The customer still owns architectural decisions, application behavior, data governance, and the configuration choices that remain on the customer side of the service boundary. The provider owns more of the underlying operation as the service model moves upward.

Table 1-5 Responsibility shifts across service models.

Engineering concern	Traditional / self-managed	IaaS	Application PaaS	AI PaaS
Physical facility and hardware	Customer	Provider	Provider	Provider

Engineering concern	Traditional / self-managed	IaaS	Application PaaS	AI PaaS
Virtualization and base capacity	Customer	Provider	Provider	Provider
Guest OS and runtime patching	Customer	Customer	Mostly provider	Provider for managed runtime
Application code and configuration	Customer	Customer	Customer	Customer
Data quality and authorization	Customer	Customer	Customer	Customer
Scaling mechanism	Customer-built	Customer configures resources	Platform capability with customer policy	Platform capability with model/agent policy
Operational telemetry	Customer-built	Cloud + customer instrumentation	Managed integration + application signals	Model, retrieval, agent, tool, and application signals
Metering and quota	Customer accounting	Cloud resource meters	Service and capacity meters	Tokens, throughput, retrieval, evaluation, tool, and runtime meters

The table should not be read as a promise that higher-level services eliminate engineering. They change its center of gravity. In IaaS, teams spend more effort on operating systems, middleware, and instance topology. In PaaS, they spend more effort on service configuration, application behavior, data contracts, identity, policy, deployment, observability, and cost. In AI PaaS, they must additionally govern models, prompts, knowledge, evaluation, tool authority, safety, and nondeterministic behavior.

A managed service therefore purchases an operating contract, not merely a feature. The contract includes what the provider automates, what the customer can configure, which limits apply, which events and metrics are exposed, how the resource is upgraded, how data is protected, how failure is handled, how the service is billed, and how it is retired. Good architecture chooses these contracts deliberately.

Managed does not mean responsibility-free

A provider can automate backups, replicas, patching, or model hosting. The customer must still choose the right service boundary, configure it correctly, validate recovery, control access, interpret telemetry, and decide whether the cost and risk are acceptable.

1.5.1 Five engineering effects of productized responsibility

1. Standardization. A resource type gives teams a common vocabulary and repeatable properties instead of local server folklore.

2. Automation. Lifecycle operations become API calls and desired-state declarations that pipelines and controllers can execute.

3. Evidence. Activity records, health states, metrics, logs, and cost data make change and operation inspectable.

4. Economies of operation. Providers can operate large fleets, distribute fixes, and expose specialized capabilities that would be expensive to reproduce locally.

5. Reversibility. A resource can be versioned, replaced, scaled, locked, recovered, or deleted through a defined lifecycle rather than an undocumented sequence of manual actions.

1.6 Azure as a Coherent Reference Control Plane

This book uses Azure as the primary implementation spine because its management hierarchy and cross-cutting services make the control-plane model unusually explicit. Azure Resource Manager provides the deployment and management layer for creating, updating, deleting, organizing, and controlling access to resources. Declarative ARM templates and Bicep files state the intended resources rather than a sequence of machine-building commands. [4]

At a high level, the hierarchy is tenant and management groups, subscriptions, resource groups, resource providers, and resources. A subscription is an authorization, quota, and billing boundary. A resource group is a lifecycle-oriented container for related resources. A resource provider supplies one or more resource types and management operations. The resource itself has a globally structured identifier within the Azure environment.

```
/subscriptions/{subscriptionId}
/resourceGroups/{resourceGroupName}
/providers/Microsoft.Storage/storageAccounts/{accountName}
```

The identifier is more than a path. It is an anchor for authorization scope, policy evaluation, activity records, deployment history, locks, inventory queries, and cost attribution. A well-designed cloud system can explain every important resource in terms of that anchor.

Table 1-6 Azure services that make the provider control plane visible.

Control-plane concern	Azure anchor	Engineering purpose
Resource lifecycle	Azure Resource Manager and resource providers	Create, update, inspect, and delete typed resources.
Organization	Management groups, subscriptions, resource groups, tags	Express ownership, environment, delegation, and cost scope.
Authorization	Microsoft Entra identities and Azure RBAC	Grant management actions over explicit scopes.
Guardrails	Azure Policy and management locks	Audit or deny noncompliant state and reduce accidental change.
Inventory	Azure Resource Graph	Query resources and properties across scopes.
Change evidence	Azure Activity Log	Identify caller, operation, time, result, and affected resource.
Economics	Cost Management, budgets, meters, quotas	Connect usage and spend to technical and organizational ownership.

This coherence is pedagogically valuable. The same questions recur whether the resource is a VM, a storage account, a Kubernetes cluster, a search service, or an AI platform object: Where is it addressed? Who can manage it? Which policy applies? Which network boundary contains it? How is it observed? What does it cost? What happens when it is changed or deleted?

Azure is not used because every other cloud should imitate it or because every capability has an exact equivalent. It is used as a stable reference vocabulary. The comparison platforms are then evaluated by engineering intent and evidence, not by superficial product-name matching.

1.7 Translating the Control Plane Across China's Leading Cloud Platforms

A multi-cloud comparison fails when it becomes a four-column synonym list. Equivalent names do not guarantee equivalent scope, lifecycle, authorization granularity, network behavior, observability, or billing. Useful translation begins with a management question: what owns the resource, how is it grouped, who can act on it, where is the action recorded, how can it be queried, and how is cost attributed?

Alibaba Cloud Resource Management currently brings together Resource Directory, Resource Group, Resource Sharing, Tag, and Resource Center. Resource Directory organizes accounts, Resource Group organizes resources within one account, tags add cross-cutting classification, and Resource Center provides cross-account, cross-product, and cross-region resource views. ROS defines resources and dependencies in templates and automates deployment. ActionTrail records management events and, when configured, data events. [10] [11] [12]

Tencent Cloud uses CAM for identities, roles, and permission policies, Tencent Cloud Organization for multi-account structures, projects and tags for resource organization, and CloudAudit for management activity evidence. Tag-based authorization and cost allocation make tags part of the governance model rather than decorative metadata. [13] [14] [15]

Volcengine documentation exposes project-based resource management, IAM authorization, tags, Terraform and Pulumi integration, and CloudTrail log collection. The exact coverage varies by service, which is itself an important comparison result: resource-level management should be verified per capability rather than assumed from a global brand promise. [16] [17] [18]

Table 1-7 Translating common control-plane intent across the four reference platforms.

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Organize accounts	Tenant / management groups / subscriptions	Resource Directory / folders / members	Tencent Cloud Organization / departments / members	Account and organization capabilities; verify current scope
Group resources	Resource groups and tags	Resource Groups and tags	Projects and tags	Projects and tags
Authorize management	Microsoft Entra ID + Azure RBAC	RAM and resource-directory control policies	CAM users, roles, policies, permission boundaries	IAM users, roles, and policies
Declare infrastructure	ARM / Bicep / Terraform	ROS / Terraform	Terraform provider and service APIs	Terraform / Pulumi / service APIs
Query inventory	Azure Resource Graph	Resource Center and advanced search	Project/tag/service inventories; organization views	Project/tag/service inventories
Trace changes	Azure Activity Log	ActionTrail	CloudAudit	CloudTrail logs and service audit records
Attribute cost	Subscriptions, resource groups, tags, budgets	Accounts, resource groups, tags, billing data	Accounts, projects/tags, allocation labels, billing center	Accounts, projects/tags, billing and quota views

The table deliberately contains cautious language. A comparison should document non-equivalence. One provider may offer organization-wide inventory as a first-class service while another exposes it through several consoles or APIs. One service may support resource-level authorization while another stops at operation or service scope. One audit product may record management events by default while data events require explicit configuration. These are engineering differences, not documentation inconveniences.

Chapter 3 develops this comparison method further through shared responsibility, deployment location, operational boundary, degree of integration, and platform DNA. For now, the essential discipline is simple: translate an engineering intent, then inspect the actual resource and policy model that satisfies it.

1.8 Control Planes Compose

A large cloud system rarely has only one control plane. Control planes compose in layers. The provider control plane creates and governs cloud resources. A managed-service control plane governs domain objects inside a service. An application control plane governs business configuration, tenants, workflows, and feature policy. The operational data plane then executes requests under the state established by those control planes.

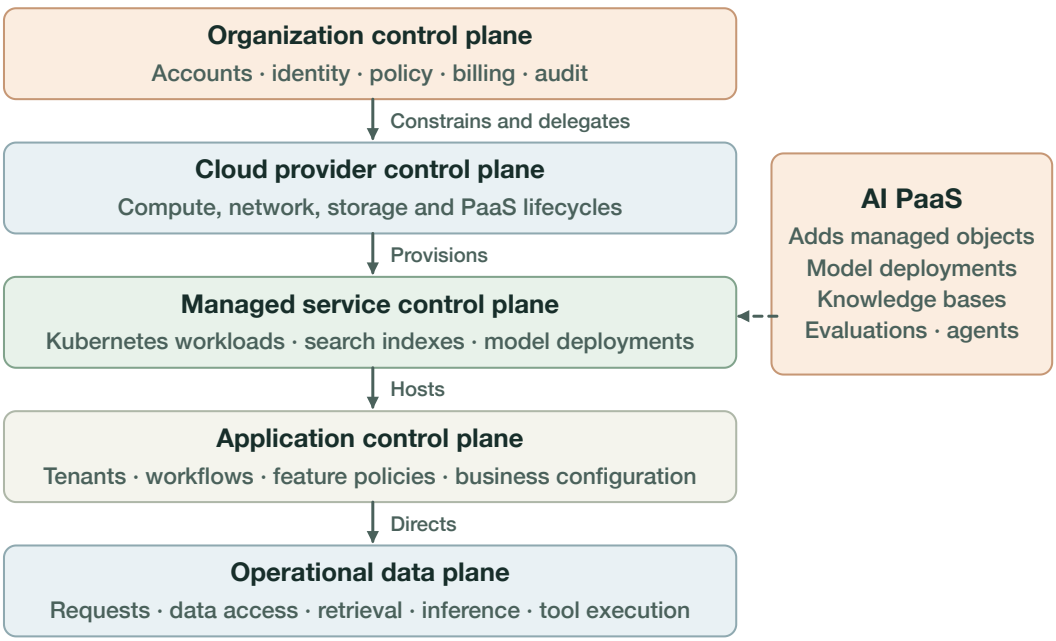


Figure 1-4 Provider, managed-service, and application control planes compose; they do not replace one another.

A managed Kubernetes example makes the layers concrete. Azure Resource Manager creates an AKS resource, attaches identity and networking, selects node pools, and records management operations. The Kubernetes API is a service control plane for Deployments, Services, Secrets, and other desired-state objects. The application running on the cluster may expose yet another control plane for tenants, workflows, or business configuration. Confusing these layers leads to misplaced permissions and incomplete audit trails.

The same composition appears in AI PaaS. The cloud control plane creates a Foundry or model resource, applies RBAC, network, policy, and cost scope. The AI platform control plane manages model deployments, knowledge objects, evaluations, agents, tools, and runtime configuration. The application controls user sessions, business permissions, approval workflows, and product behavior. Inference, retrieval, and tool execution occur in the data plane.

1.8.1 AI PaaS is the control-plane extension of cloud engineering

A direct model API can be valuable, but a callable endpoint is not yet an AI engineering platform. AI PaaS emerges when model and agent capabilities participate in the same disciplines as other cloud resources: addressability, identity, network boundaries, lifecycle, policy, telemetry, quota, evaluation, cost, and retirement.

Table 1-8 Cloud control-plane disciplines extended to AI PaaS.

Control-plane discipline	Infrastructure / PaaS example	AI PaaS example
Resource identity	VM, database, function, Kubernetes cluster	Model deployment, knowledge base, evaluation, agent
Lifecycle	Create, update, scale, patch, fail over, delete	Deploy model, revise prompt/tool policy, version knowledge, promote or retire agent
Authorization	Who can manage or use a resource	Who can deploy models, retrieve knowledge, run evaluations, or invoke tools
Networking	Public/private endpoints, peering, firewall	Private inference, private retrieval, connector and tool boundaries
Observability	Health, latency, errors, saturation, logs	Tokens, latency, grounding, tool traces, evaluation, safety and quality signals
Economics	Instance hours, storage, requests, transfer	Tokens, reserved throughput, retrieval, vectorization, evaluation, tool and runtime cost

Control-plane discipline	Infrastructure / PaaS example	AI PaaS example
Governance	Policy, lock, configuration compliance	Approved models, safety policy, evaluation gates, data and tool restrictions

Microsoft Foundry currently illustrates this direction by grouping agents, models, and tools with enterprise controls such as RBAC, networking, policy, tracing, monitoring, and evaluation. The product surface will evolve, but the architectural signal is durable: AI capabilities are moving from library objects and isolated API calls into cloud resource systems. [9]

This book follows that signal. The first four parts build the control-plane foundation for resources, applications, delivery, operations, and cost. The fifth part applies the same questions to data+AI platforms, managed model resources, RAG, and agent platforms. The final part shows how a verified cloud capability can later be exposed as a governed Agent tool without giving an Agent unrestricted platform authority.

1.9 Capability Check C1: Inspect a Resource Graph and Trace a Lifecycle Event

This capability check is intentionally small. It does not deploy an application. It proves the chapter's central claim by creating one low-cost resource, observing it through the management plane, and collecting lifecycle evidence. The exercise is independent of every later lab.

Table 1-9 Capability Check C1 at a glance.

Item	Definition
Capability goal	Show that a cloud resource has an address, type, scope, properties, tags, lifecycle state, audit event, and cleanup path.
Azure resources	One resource group and one Standard_LRS storage account with no application data.
Permission	Contributor on the lab resource group or a sandbox subscription; permission to read Activity Log.
Cost boundary	Negligible when no data is stored and the resource group is deleted immediately after verification.
Evidence	Resource ID, resource type, location, tags, provisioning state, creation event, caller, timestamp, and deletion verification.

Item	Definition
Safety boundary	Use a sandbox subscription. Do not run the commands in a production resource group.

1.9.1 Prerequisites

- Azure CLI installed and authenticated with az login.
- A sandbox subscription selected with az account set --subscription <id-or-name>.
- Permission to create a resource group and storage account, and to read subscription activity events.
- The Azure Resource Graph CLI extension. Install it with az extension add --name resource-graph if needed.

1.9.2 Create one scoped resource graph

Choose a region that is available in your subscription. The commands below create a unique resource group and storage account, then apply tags that express purpose and lifecycle. Replace eastus if another region is more appropriate.

```
suffix=$(openssl rand -hex 4)
rg=cp-lab-$suffix
location=eastus
storage=cplab${suffix}

az group create \
  --name "$rg" \
  --location "$location" \
  --tags purpose=book-capability-check lifecycle=ephemeral owner=reader

az storage account create \
  --name "$storage" \
  --resource-group "$rg" \
  --location "$location" \
  --sku Standard_LRS \
  --kind StorageV2 \
  --https-only true \
  --min-tls-version TLS1_2 \
  --tags purpose=book-capability-check lifecycle=ephemeral owner=reader
```

The command sequence is still imperative, but the resulting resources are not anonymous side effects. They have provider-defined types, resource IDs, properties, tags, and lifecycle state. Later labs use IaC to make the declaration reviewable and repeatable; this check stays focused on control-plane visibility.

1.9.3 Inspect the addressable resource

```
az resource list \
  --resource-group "$rg" \
  --query "[].{name:name,type:type,id:id,location:location,tags:tags}" \
  --output jsonc
```

Record the storage account resource ID. Confirm that it includes the subscription, resource group, provider namespace, resource type, and resource name. Then query the graph at subscription scope:

```
az graph query -q "Resources
| where resourceGroup =~ '$rg'
| project id, name, type, location, tags,
provisioningState=tostring(properties.provisioningState)" \
  --output table
```

The result should show one storage account. Resource Graph may not expose every service property, and ingestion can be slightly delayed. Those limitations are part of the control-plane contract: an inventory query is a management projection, not the data plane itself.

1.9.4 Trace the creation event

```
az monitor activity-log list \
  --resource-group "$rg" \
  --offset 2h \
  --max-events 30 \
  --query "[].
{time:eventTimestamp,caller:caller,operation:operationName.localizedValue,status
:status.localizedValue,resource:resourceId,correlation:correlationId}" \
  --output table
```

Find the successful create or update operation for the storage account. Capture the caller, operation name, timestamp, resource ID, status, and correlation ID. The correlation ID is especially useful when one deployment produces several provider operations.

The activity event is not the complete story. It proves that a management operation occurred; it does not prove that an application can read or write objects, that network policy is correct, or that data-plane authorization is safe. Those require separate evidence.

1.9.5 Verify the resource contract

Table 1-10 Evidence checklist for the capability check.

Question	Expected evidence
Can the resource be addressed?	A full Azure resource ID.
Can its type be named?	Microsoft.Storage/storageAccounts.
Can its scope be explained?	A subscription and resource group created for this check.
Can its intent be attributed?	purpose, lifecycle, and owner tags.
Can current state be inspected?	Resource properties and provisioning state from the management plane.
Can the change be traced?	A successful Activity Log event with caller, time, operation, resource ID, and correlation ID.
Can it be removed safely?	Deletion of the lab resource group and verification that the group no longer exists.

1.9.6 Clean up

```
az group delete --name "$rg" --yes

az group exists --name "$rg"
```

The final command should return false. Deleting the resource group demonstrates a lifecycle boundary: the temporary resources were intentionally grouped so that one reviewed operation could retire them together. The Activity Log remains available according to the platform's retention and export configuration even after the resources are gone.

1.9.7 Multi-cloud translation

Table 1-11 Repeating the same capability check by engineering intent.

Intent	Alibaba Cloud	Tencent Cloud	Volcengine
Create an isolated scope	Create or select a Resource Group.	Use a sandbox account/project and tags.	Create or select a project and tags.
Create one low-cost resource	Create an empty OSS bucket or another ephemeral resource supported by the group.	Create an empty COS bucket or another ephemeral resource.	Create an empty TOS bucket or another ephemeral resource.

Intent	Alibaba Cloud	Tencent Cloud	Volcengine
Inspect inventory	Use Resource Center or the service/resource-group view.	Use project/tag filters and the relevant service inventory.	Use project/tag filters and the relevant service inventory.
Trace management change	Query ActionTrail management events.	Query CloudAudit event history.	Query CloudTrail or the service audit record.
Delete and verify	Release the resource and empty/delete the resource group where supported.	Delete the resource and confirm it no longer appears.	Delete the resource and confirm it no longer appears.

The purpose is not to reproduce identical console steps. The purpose is to collect the same classes of evidence: address, scope, type, intent metadata, lifecycle state, audit event, and cleanup result. Any missing class of evidence is a meaningful platform or service limitation to document.

CloudDesk reuse note

A future CloudDesk FinOps Agent could consume this capability through two read-only tools: cloud.resource.inventory, which returns resources with provider-neutral identity and ownership fields; and cloud.resource.lifecycle.explain, which correlates an audit event with the affected resource and human-readable change. This chapter does not implement those tools; it establishes the cloud evidence they must preserve.

```
cloud.resource.inventory(scope, filters)
  -> providerResourceId, resourceType, region, ownership, lifecycleState,
costScope
evidence: inventoryTimestamp, sourceApi, requestCorrelationId
```

1.10 Principles to Carry Forward

The rest of the book applies the following principles repeatedly:

1. Cloud is an operating model, not a location. Remote hardware becomes cloud engineering only when provisioning, pooling, elasticity, networking, and measurement are exposed through an operable service model.

2. The control plane is the primary engineering surface. Portal, CLI, SDK, IaC, policy, inventory, audit, and cost tools should converge on addressable resources and explicit lifecycle operations.

3. Control-plane and data-plane authority are different. Creating a service and using a service require separate endpoints, permissions, evidence, and threat models.

4. Applications are resource graphs. Compute is connected to network, identity, storage, observability, policy, and cost resources whose dependencies determine blast radius and recoverability.

5. Managed services productize responsibility. They reduce customer-operated layers while increasing the importance of configuration, data, identity, policy, evidence, and service-boundary judgment.

6. Multi-cloud comparison begins with intent. Compare ownership, scope, lifecycle, authorization, audit, inventory, and cost before comparing product names.

7. Control planes compose. Provider, service, and application control planes should have explicit boundaries rather than one ambiguous administrator surface.

8. AI PaaS extends the same discipline. Models, knowledge, evaluation, tools, and agents become production resources when their lifecycle, authority, evidence, and cost are governed.

The next chapter turns this broad thesis into a precise resource-level checklist. It asks what makes a capability a governable cloud resource rather than a library call, an SDK object, or an opaque remote service. That checklist will then be used to examine every infrastructure, platform, data, AI, and agent capability in the book.

References and Official Documentation

Product and documentation state checked August 24, 2026. Volatile portal paths, API versions, quotas, and regional availability should be reverified before final manuscript freeze.

[1] NIST SP 800-145. The NIST Definition of Cloud Computing. Five essential characteristics, three service models, and four deployment models. https://csrc.nist.gov/pubs/sp/800/145/final

[2] NIST SP 500-322. Evaluation of Cloud Computing Services Based on NIST SP 800-145. Operational guidance for deciding whether a capability meets the NIST cloud model. https://www.nist.gov/publications/evaluation-cloud-computing-services-based-nist-sp-800-145

[3] NIST SP 500-292. NIST Cloud Computing Reference Architecture. Reference roles, activities, and architectural components for cloud systems. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-292.pdf

[4] Microsoft Learn. What is Azure Resource Manager? Azure deployment and management layer, resource providers, declarative syntax, tags, access control, and lifecycle management. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/overview

[5] Microsoft Learn. Control plane and data plane operations. Management endpoints, operational endpoints, authorization, locks, and plane-specific behavior. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/control-plane-and-data-plane

[6] Microsoft Learn. Azure resource providers and types. Provider namespaces, resource types, and management operations. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/resource-providers-and-types

[7] Microsoft Learn. Overview of Azure Resource Graph. Cross-subscription resource exploration and governance-oriented queries. https://learn.microsoft.com/en-us/azure/governance/resource-graph/overview

[8] Microsoft Learn. Activity Log in Azure Monitor. Subscription-level management events and export destinations. https://learn.microsoft.com/en-us/azure/azure-monitor/platform/activity-log

[9] Microsoft Learn. What is Microsoft Foundry? Unified management of agents, models, and tools with enterprise controls. https://learn.microsoft.com/en-us/azure/foundry/what-is-foundry

[10] Alibaba Cloud Documentation. What is Resource Management? Resource Directory, Resource Group, Resource Sharing, Tag, and Resource Center. https://www.alibabacloud.com/help/en/resource-management/product-overview/what-is-resource-management

[11] Alibaba Cloud Documentation. Resource Orchestration Service. Template-based resource definition, dependency expression, and automated deployment. https://www.alibabacloud.com/help/en/ros/

[12] Alibaba Cloud Documentation. What is ActionTrail? Management and data events, delivery, retention, change tracking, and audit use cases. https://www.alibabacloud.com/help/en/actiontrail/product-overview/what-is-actiontrail

[13] Tencent Cloud Documentation. CAM Overview. Identity and access management for users, groups, roles, policies, and cloud-resource permissions. https://www.tencentcloud.com/document/product/598/72360

[14] Tencent Cloud Documentation. Tag application scenarios. Resource classification, tag-based access control, and cost allocation. https://www.tencentcloud.com/document/product/651/41690

[15] Tencent Cloud. CloudAudit product overview. Account activity, API usage, resource-change tracking, compliance, and troubleshooting. https://www.tencentcloud.com/product

[16] Volcengine Documentation. Resource project management. Project-based grouping and IAM authorization over groups of resources. https://www.volcengine.com/docs/6257/170353?lang=en

[17] Volcengine Documentation. Terraform. Infrastructure-as-code support and provider configuration guidance. https://www.volcengine.com/docs/6706/101989?lang=en

[18] Volcengine Documentation. Collect CloudTrail logs. Collection of cloud operation audit logs into Torch Log Service. https://www.volcengine.com/docs/6470/2201696?lang=en

Part I / Chapter 2

Resource-Level Thinking: Control Plane, Data Plane, and Lifecycle

A Formal Contract for Governed Cloud Capabilities

A cloud capability becomes governable when it has a durable identity, a management contract, an operational endpoint, observable state, accountable evidence, and explicit change and retirement semantics.

In This Chapter

- Why resource-level is an architectural test rather than a marketing label
- How a durable resource identity differs from a client object, session, endpoint, or process
- How control-plane and data-plane contracts separate authority, availability, and evidence
- Why desired state, provisioning state, and operational state must be observed independently
- How resource graphs express dependencies without forcing every component into one lifecycle boundary
- How create, configure, operate, change, recover, detach, retain, delete, and purge form a managed lifecycle
- Why an update may mean in-place mutation, restart, rolling change, redeployment, replacement, or migration
- How asynchronous operations, retries, correlation identifiers, and convergence affect automation
- How the same resource-level checklist applies to IaaS, application PaaS, AI PaaS, knowledge, and agent platforms
- How to perform a concise resource-level review before accepting a cloud capability as governable

Chapter 1 established the cloud as a programmable control plane. This chapter makes that claim precise. The word resource is used casually in technology: a CPU is a resource, a file handle is a resource, a database connection is a resource, and a software library may expose a Resource class. Those meanings are legitimate, but they are too broad for cloud architecture. In this book, resource-level has a narrower and more demanding meaning.

A resource-level cloud capability has a durable management identity that exists independently of one client process. It can be created or adopted, inspected, authorized, configured, connected, observed, costed, changed, and retired through a management contract. It also exposes—or governs access to—a data-plane capability that an application, operator, or agent actually uses. The resource therefore participates in an estate: it has ownership, policy, dependencies, evidence, and economic consequences.

This definition is intentionally stricter than "available through an API." A remote API may accept requests while offering no customer-visible lifecycle, no scoped management identity, no policy attachment point, no inventory record, and no reliable retirement semantics. Conversely, a highly managed service may expose only a small data-plane API while possessing a rich cloud control plane behind it. Resource-level thinking asks what can be governed, not how many methods a developer can call.

The distinction is particularly important for AI-native systems. A model client object is not a model deployment. A vector collection is not automatically a governed knowledge resource. A prompt-and-tool loop is not automatically an agent resource. Before a capability can be trusted as part of industrial software engineering, its management identity, authority, state, evidence, cost, and end-of-life behavior must be explicit.

The chapter thesis

A cloud resource is not defined merely by being remote or callable. It is defined by a durable identity and a governable lifecycle that joins management intent, operational use, evidence, economics, and retirement.

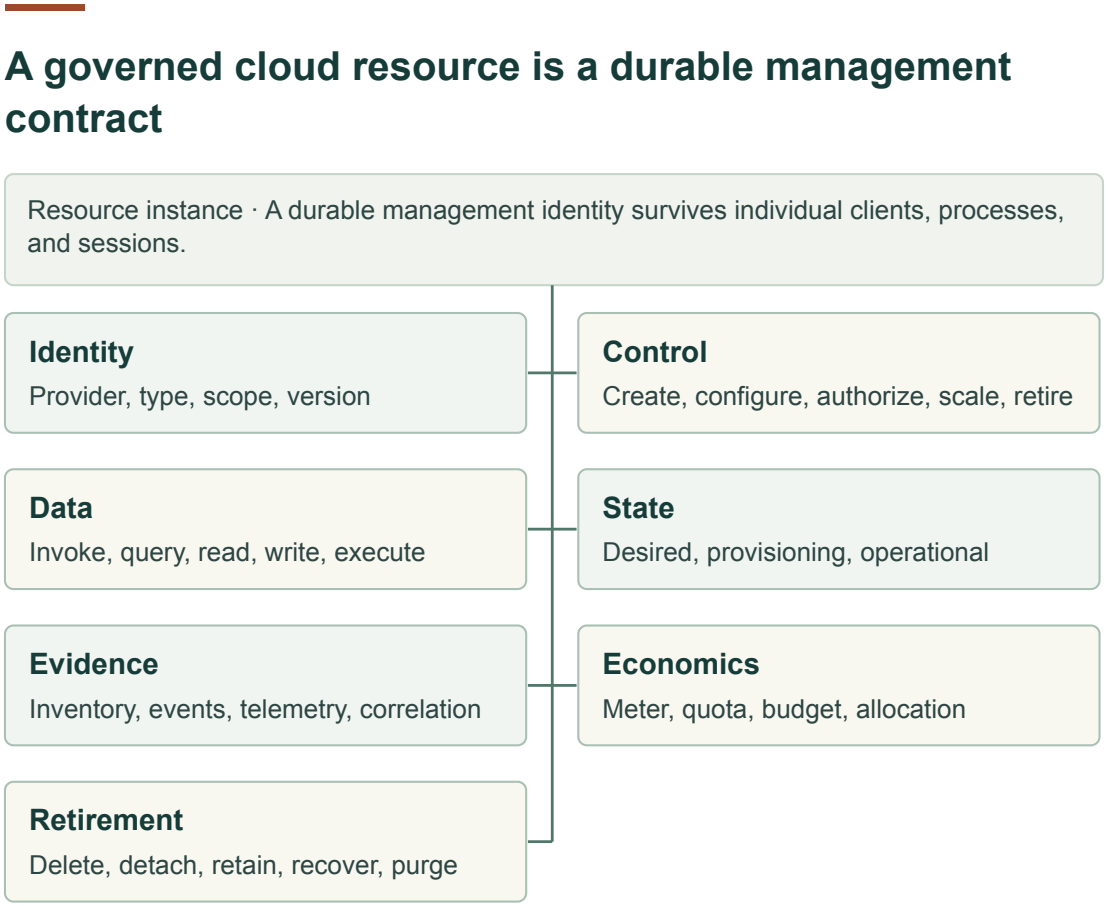


Figure 2-1 A governed resource combines seven obligations around one durable resource instance.

2.1 Resource-Level Thinking Is a Test, Not a Label

Resource-level is a diagnostic test used throughout this book. It does not mean that every property must be visible in one portal page, nor does it imply that all services provide equal granularity. It asks whether the capability participates in a durable cloud management system strongly enough to support repeatable engineering and accountable operation.

The test begins with independence. If deleting a local object or closing a network connection causes the capability to disappear, the object is probably a client-side handle or session rather than the cloud resource itself. If the capability persists, has a provider-defined type and identifier, can be rediscovered through inventory, and accepts lifecycle operations through a management endpoint, it is a candidate resource. The capability becomes governed when identity, policy, network, telemetry, cost, and retirement can be attached to that durable object.

NIST defined cloud computing in terms of on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. Its reference architecture places a resource abstraction and control layer between physical resources and the service layer. Resource-level thinking translates those provider-side abstractions into the consumer-side contract that software teams can manage. [1] [2] [3]

Surface	What persists?	What can be governed?	Resource-level verdict
Library object	Only in the client process	Local configuration and code	Not a cloud resource
Remote API call	Provider service exists, but the caller may have no named instance	Request authentication and provider-wide terms	Callable capability; resource status is unproven
Managed endpoint	A deployment or endpoint persists	Identity, capacity, network, telemetry, and version may be configurable	Often resource-level, subject to lifecycle verification
Governed cloud resource	Named instance with durable provider identity	Lifecycle, policy, authority, evidence, cost, dependencies, and retirement	Meets the book's resource-level test

Resource-level is not the same as fully managed

An IaaS virtual machine is resource-level even though the customer manages its guest operating system. A fully hosted API may be highly managed by the provider yet expose no customer-specific resource instance. Service responsibility and resource addressability are related, but they are not identical.

2.2 The Resource Contract

The resource contract is the minimum set of questions that a cloud architect should be able to answer before treating a capability as part of a governed system. The questions are deliberately cross-cutting. A product page may explain features, but the contract asks how the capability fits into an organization, an application, an evidence chain, and an end-of-life process.

The contract can be summarized as follows:

```
Governed resource = identity + control contract + data contract
                  + state model + evidence + economics + retirement
```

None of these terms is sufficient alone. A resource ID without an operational endpoint describes something that can be managed but not necessarily used. A data endpoint without a management identity describes a utility rather than a customer-governed instance.

Telemetry without lifecycle context cannot reliably explain which version or configuration produced an outcome. Cost without ownership cannot guide action. Deletion without retention semantics is not a safe retirement design.

Contract dimension	Question	Minimum evidence
Identity	What exactly is this instance, who provides it, and where is it scoped?	Provider namespace, resource type, resource ID, parent scope, region or placement
Control contract	How is desired configuration created, changed, authorized, and retired?	Management endpoint, API version, permissions, policy result, operation record
Data contract	How is the capability used?	Service endpoint, protocol, caller identity, request/transaction evidence
State	What was requested, what is being provisioned, and what is operational now?	Desired properties, provisioning status, health/readiness/quality signals
Evidence	Can management changes and operational outcomes be reconstructed?	Inventory, activity/audit events, correlation IDs, logs, metrics, traces
Economics	What is metered, limited, allocated, budgeted, and optimized?	Meter, SKU, quota, tags, cost scope, budget and usage records
Retirement	What happens when management ends?	Delete/detach behavior, retention, soft delete, purge, credential revocation

2.3 Identity and Addressability

A resource begins with addressability. The management system must be able to distinguish one instance from another and route an operation to the provider responsible for it. In Azure, a resource ID encodes subscription, resource group, provider namespace, resource type, and resource name. Resource providers define the types and operations implemented under namespaces such as Microsoft.Compute, Microsoft.Storage, or Microsoft.MachineLearningServices. [4] [6]

The exact syntax differs across clouds, but the architectural components are stable: provider, type, name, parent scope, location or placement, and an API contract. A display name is not enough. Names may be reused or mutable, while a provider-generated ID is the durable reference used in role assignments, policy evaluation, dependencies, audit events, billing records, and automation output.

The management identity should also be distinguished from service endpoints. A storage resource can have one management ID and several data endpoints. A model deployment may have one control-plane identity while exposing regional or private inference endpoints. An endpoint can be rotated, disabled, or fronted by a gateway without changing the managed resource that owns it. Automation that stores only endpoint URLs loses lifecycle and governance context.

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}
/providers/{providerNamespace}/{resourceType}/{resourceName}
```

Identity element	Purpose	Common error
Provider and type	Routes management operations and defines the schema	Assuming similarly named products have equivalent resource contracts
Resource ID	Provides a durable address for policy, RBAC, evidence, and automation	Using a portal URL, DNS name, or display name as the canonical identity
Parent scope	Defines ownership, inheritance, policy, and cost context	Treating a resource as independent of subscription/account/project hierarchy
API version	Selects the management schema and operation behavior	Letting generated examples silently select preview or obsolete behavior
Location or placement	Constrains data residency, capacity, latency, and service availability	Assuming the management location is the same as the data location
Tags and metadata	Add business classification without changing provider type	Using tags as a substitute for durable resource identity

2.4 Control Plane and Data Plane Are Separate Contracts

Chapter 1 introduced the control-plane/data-plane distinction. Resource-level thinking adds an important refinement: the two planes are separate contracts attached to the same managed capability. The control plane answers how the resource is defined, authorized, connected, scaled, observed, and retired. The data plane answers how the resource performs its service: executing code, storing objects, answering queries, publishing messages, serving inference, retrieving knowledge, or running an agent.

Azure documents the distinction explicitly. Control-plane requests are routed through Azure Resource Manager, while data-plane requests normally go to service-specific endpoints. A management lock applies to Resource Manager operations, not automatically to

operations inside the service. Similarly, a principal may be able to manage a Key Vault resource without being able to read secrets, or use stored data without permission to resize or delete the containing resource. [4] [5] [14]

Separate contracts imply separate threat models, identities, availability assumptions, telemetry, and SLOs. The control plane can report that creation succeeded while the service endpoint remains unreachable because of DNS, network, application configuration, data readiness, or authorization. Conversely, an already provisioned service can continue serving requests during a temporary management-plane impairment. A design that reports only one plane cannot establish the health of the other.

One resource exposes two different operational contracts

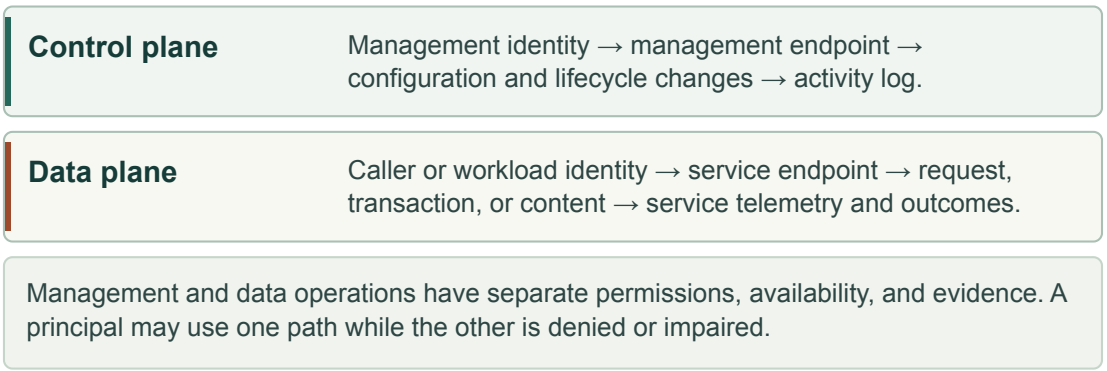


Figure 2-2 The control plane manages the resource; the data plane delivers its capability through a different contract.

Resource	Control-plane operation	Data-plane operation	Why the distinction matters
Virtual machine	Create, size, attach identity, configure networking, delete	Run guest processes, read/write disks, accept application traffic	VM management success does not prove guest readiness
Object storage	Create account/bucket, set redundancy, network and lifecycle policy	Put, get, list, copy, or delete objects	A resource lock does not automatically protect every object
Managed database	Provision engine, set tier, backup, network, and maintenance policy	Execute queries and transactions	Database users need not be database-resource administrators

Resource	Control-plane operation	Data-plane operation	Why the distinction matters
Model deployment	Choose model/version/capacity/network/safety policy	Submit inference requests	A caller should not automatically change deployment configuration
Knowledge resource	Create schema, ingestion, access, version and alias policy	Retrieve grounded evidence	Index health and retrieval quality require different evidence
Agent resource	Version instructions, tools, knowledge, policy, evaluation, release	Start runs, receive events, invoke tools	Runtime authority must be narrower than platform administration

2.5 Desired State, Provisioning State, and Operational State

The word state is dangerously overloaded. A declarative template describes desired state: the properties the owner intends the platform to establish. A resource provider reports provisioning state: the progress or terminal result of a management operation. The service exposes operational state: whether the capability is ready, reachable, healthy, current, and meeting its functional and nonfunctional objectives.

These states are related but not interchangeable. A desired replica count of five is an instruction, not an observation. A provisioningState of Succeeded indicates that the management operation reached a terminal success state; it does not prove that an application passed its health probe or that a newly built index contains the expected documents. Azure Resource Manager operations can be asynchronous, and clients may need to poll an operation URL until the provider reports Succeeded, Failed, or Canceled. [7] [8] [9]

The separation becomes more important as services become more managed. A platform may successfully accept a model deployment while capacity allocation continues. A search service may accept an indexer configuration while ingestion later fails on a document. An agent definition may be published while one tool connection is unavailable. The control plane proves management progress; operational evidence proves service behavior.

Desired, provisioning, and operational state answer different questions

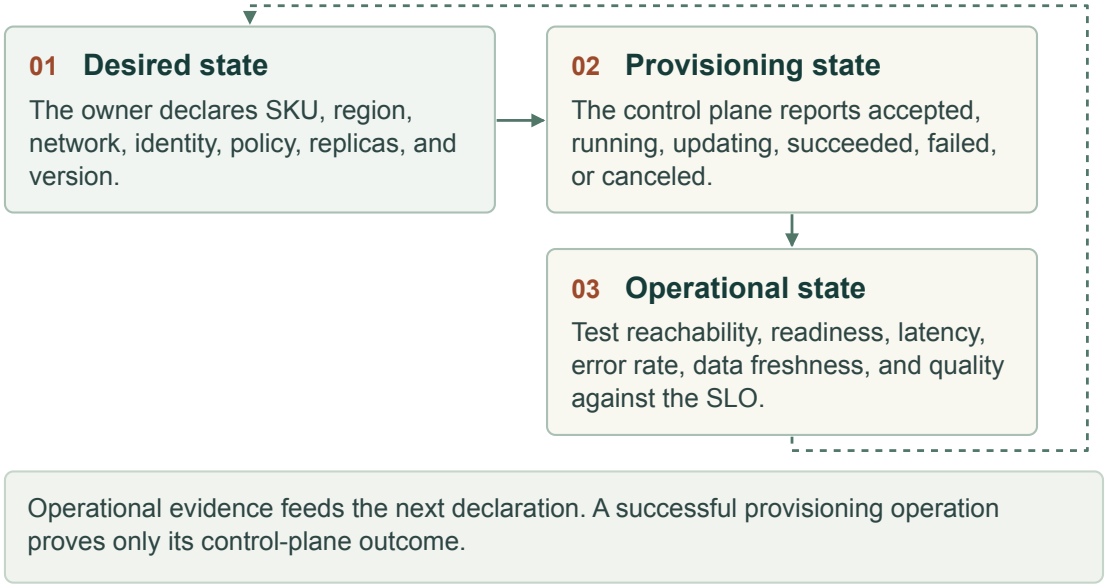


Figure 2-3 Desired state, provisioning state, and operational state form three evidence channels.

State channel	Owned by	Typical questions	Evidence
Desired state	Architecture and delivery process	What should exist? What version and configuration are intended?	Template, parameters, policy, approved plan, Git commit
Provisioning state	Cloud resource provider and management client	Was the operation accepted? Is it still running? Did it succeed or fail?	Operation URL, provisioningState, status code, error, correlation ID
Operational state	Service runtime and application owners	Is the capability reachable, ready, correct, performant, and within SLO?	Health/readiness, logs, metrics, traces, quality and freshness tests

Succeeded is not the same as ready

A provider can finish the management operation before the underlying application, data set, route, certificate, model, or index is ready for production use. Automation should wait for the evidence that matches the dependency, not merely for a generic terminal state.

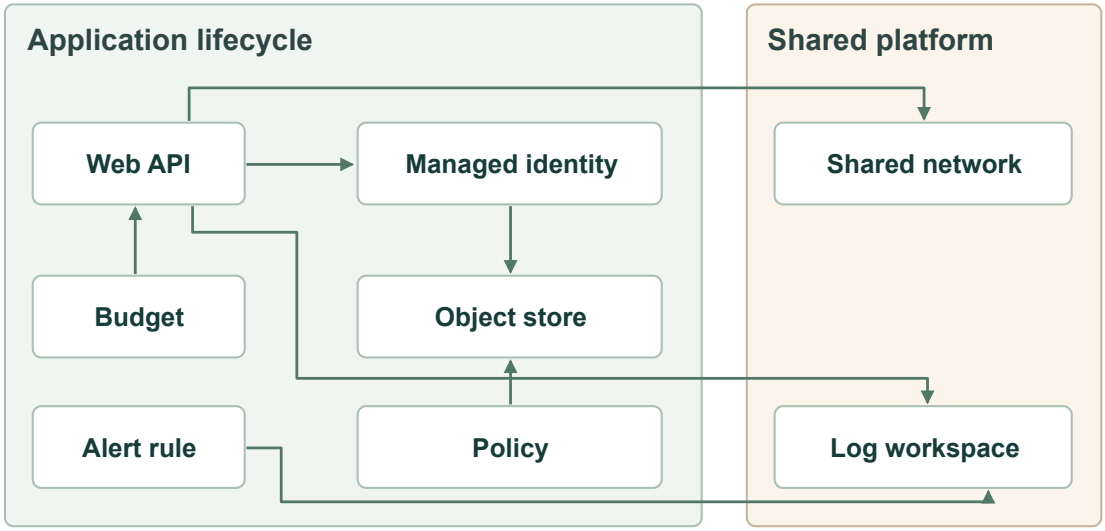
2.6 Resource Graphs and Lifecycle Boundaries

A production solution is not one resource. It is a graph of typed resources connected by dependencies and governance relationships. A web application may depend on a network, identity, secret store, database, object store, model deployment, monitoring destination, budget, and policy assignment. Infrastructure as code makes some dependencies explicit; other relationships arise through IDs, endpoints, role assignments, DNS, tags, or runtime configuration.

Bicep and ARM templates use references and dependsOn relationships to determine deployment order. Bicep can infer many dependencies when one symbolic resource reference is used by another. These deployment dependencies are useful, but the full operational graph is larger. A budget does not have to be created before a web app to affect its financial governance. A log workspace may outlive the application whose telemetry it stores. A shared network may serve dozens of applications. [10] [11]

Resource groups are useful lifecycle and management scopes, but co-location alone does not prove that every resource should be created, changed, and deleted together. Azure recommends that resources in one resource group share a lifecycle; the same principle applies across providers even when the grouping primitive has a different name. Shared resources should have their own ownership and retirement path instead of being hidden inside one application deployment. [4]

A solution is a resource graph with more than one lifecycle boundary



The Web API uses the managed identity to access the object store. Budget constrains the API and policy governs the store. The API connects to the shared network and emits logs to the workspace. The alert rule queries that workspace.

Resources with different change, recovery, and retirement schedules belong to separate lifecycle boundaries.

Figure 2-4 A solution is a resource graph whose application and shared-platform components have different lifecycle boundaries.

Relationship	Meaning	Lifecycle implication
Owns	One resource is the durable parent of a child resource	Deleting or moving the parent may constrain or remove the child
Depends on	Creation or operation requires another resource	Order, readiness, rollback, and failure propagation must be explicit
Attaches to	A resource is connected but not owned, such as a disk or private endpoint	Detachment may be valid without deletion
Authorizes	Identity or policy grants authority over another resource	Role assignments and credentials must be revoked during retirement
Routes to	DNS, gateway, load balancer, or event rule directs traffic	Cutover and rollback can occur independently of resource replacement

Relationship	Meaning	Lifecycle implication
Observes	Telemetry destination, alert, or evaluation monitors another resource	Evidence often must outlive the observed workload
Allocates cost to	Tags, account/project scope, or billing rule assigns expenditure	Cost records must remain meaningful after resource deletion

2.7 The Managed Lifecycle Extends Beyond CRUD

Create, read, update, and delete are useful API verbs, but they are not a sufficient lifecycle model. Industrial cloud engineering also needs validation, policy evaluation, dependency resolution, readiness, operation, scaling, recovery, version transition, retention, evidence preservation, and retirement. Treating lifecycle as CRUD encourages automation that declares success too early and deletion that ends too abruptly.

A lifecycle begins before provisioning. A request must be authenticated and authorized; policy, quota, location, naming, dependencies, and available capacity may be evaluated. The provider may accept the operation and perform long-running work asynchronously. After provisioning, the resource must be connected to identities, networks, telemetry, data, and cost scopes. Readiness evidence must then establish that the capability can be used safely.

Operation is not a passive phase. The resource consumes quota and budget, emits signals, receives policy changes, experiences incidents, and may drift from intended configuration. Change can be in-place or replacement-based. Recovery may restore a protected state or create a replacement resource. Retirement must decide which objects, data, credentials, endpoints, evidence, and financial records are deleted, detached, retained, recoverable, or purged.

The managed lifecycle extends beyond CRUD

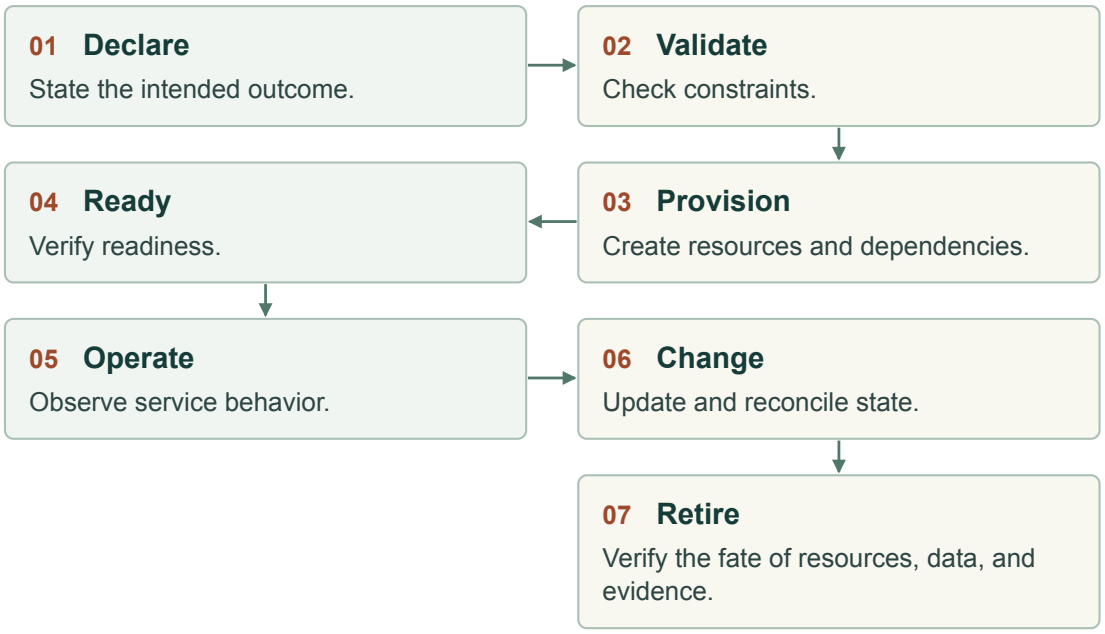


Figure 2-5 A managed resource lifecycle includes validation, readiness, operation, recovery, and several retirement outcomes.

Lifecycle stage	Control-plane responsibility	Acceptance evidence
Declare	Capture intended type, properties, dependencies, ownership, and policy	Template, parameters, change record, approved plan
Validate	Authenticate, authorize, evaluate policy, quota, name, region, and schema	Validation result, deny reason, quota/capacity check
Provision	Allocate provider capacity and establish the resource instance	Operation status, resource ID, provisioning state, error details
Connect and configure	Attach identity, network, secrets, data sources, telemetry, and budgets	Dependency graph, role assignments, endpoint and diagnostic settings

Lifecycle stage	Control-plane responsibility	Acceptance evidence
Ready	Prove that the data path and functional dependencies work	Health/readiness test, sample transaction, retrieval or inference test
Operate	Observe use, reliability, security, quality, and cost	Logs, metrics, traces, evaluations, billing and audit events
Change and recover	Apply controlled mutation, rollout, replacement, rollback, restore, or failover	Plan/diff, version, correlation, before/after evidence
Retire	Remove authority and decide delete, detach, retain, recover, or purge outcomes	Deletion/retention records, revoked access, residual inventory and cost check

2.8 Change Semantics: Update Does Not Mean One Thing

The word update hides several materially different operations. Some properties are mutable in place. Some require a process restart or a rolling restart of instances. Some create a new revision while retaining the old revision for rollback. Some cause the provider to replace the resource behind the same logical name. Others are immutable and require an explicit migration to a new resource identity.

Automation must therefore reason about change semantics, not merely send PATCH or PUT. A safe change plan identifies the blast radius, downtime, data movement, endpoint continuity, identity continuity, cost overlap, rollback path, and evidence required to declare success. Preview and what-if capabilities help, but they can only report the semantics understood by the provider and deployment engine. Azure deployment stacks and what-if results distinguish create, modify, no-change, delete, and detach outcomes for managed resources. [12] [13]

Change semantics also explain why stable indirection is valuable. DNS names, aliases, deployment slots, revisions, gateway routes, index aliases, model deployment names, and tool versions allow clients to depend on a stable contract while the implementation changes behind it. The indirection itself must be managed as a resource and included in the evidence chain.

Change form	What changes?	Primary risk	Useful control
In-place mutation	A property on the existing instance	Unexpected side effects or unsupported combinations	Validation, ETag/concurrency check, configuration diff

Change form	What changes?	Primary risk	Useful control
Restart	Runtime process or nodes restart with the same resource identity	Temporary unavailability and lost in-memory state	Readiness probe, maintenance window, retry and drain
Rolling change	Instances or replicas change incrementally	Mixed versions and partial capacity	Max surge/unavailable, health gate, progressive verification
Revision or slot	A new immutable version is created alongside the old	Traffic split and state compatibility	Alias/route, canary, rollback and versioned telemetry
Replacement	A new resource instance supersedes the old	Endpoint, identity, data, and cost discontinuity	Migration plan, parallel run, cutover and retirement checklist
Cross-resource migration	Data or workload moves to a different type, region, account, or provider	Semantic incompatibility and incomplete evidence	Export/import contract, reconciliation, dual-read/write where necessary

2.9 Deletion Is an Architectural Operation

Deletion is frequently treated as housekeeping. At resource level, it is a first-class architectural operation because it changes authority, reachability, retained data, dependencies, evidence, and cost. A control-plane delete request may remove only the management object, may cascade to child resources, may be blocked by a lock or dependency, or may place data into a recoverable state. The data plane may also support deletion of application data independently of deletion of the containing resource.

Azure management locks illustrate the plane boundary. A CannotDelete or ReadOnly lock constrains Resource Manager operations but does not automatically prevent data-plane mutation. Deployment stacks add an explicit actionOnUnmanage decision: resources removed from the managed declaration can be deleted or detached. Detachment preserves the resource but ends the stack's ownership. These are different outcomes and must not be hidden behind one “destroy” button. [12] [14]

Retirement should remove more than the primary resource. Role assignments, service credentials, private endpoints, DNS records, event subscriptions, diagnostic settings, dashboards, budgets, automation schedules, registry artifacts, and external references may remain. Some evidence and billing records should remain intentionally. A retirement verification must therefore query residual inventory, authority, traffic, telemetry, data retention, and continuing cost.

Outcome	Resource result	Data/evidence result	When it is appropriate
Delete	Management object is removed	May be removed, retained, or separately protected	The capability and its dependencies are no longer required
Detach / unmanage	Resource remains but leaves one deployment or ownership boundary	Data remains with the resource	Ownership or lifecycle moves to another stack/team
Soft delete	Resource or data is hidden but recoverable for a period	Retained until recovery window expires	Protection against accidental or malicious deletion
Retain	Selected resource, data, or evidence intentionally remains	Retention follows legal, operational, or audit policy	State must outlive the workload that produced it
Purge	Recoverable state is permanently removed	Recovery becomes impossible	Retention period and approval conditions have been satisfied
Replace	Old resource is retired after a new one takes responsibility	Data/evidence is migrated or reconciled	Immutable change, region move, redesign, or provider transition

A clean portal is not a clean retirement

A resource disappearing from one console does not prove that child resources, access grants, data copies, DNS records, event subscriptions, logs, commitments, or charges have ended. Retirement requires positive evidence of the intended residual state.

2.10 Asynchrony, Retries, Idempotence, and Convergence

Control-plane automation operates over distributed systems. Requests cross management gateways, policy engines, resource providers, regional capacity managers, and service backends. A successful HTTP response may mean that the request was accepted rather than completed. Clients must distinguish request delivery, operation acceptance, resource creation, dependency readiness, and operational success.

Azure long-running operations return headers or operation URLs that clients poll, together with provisioning state and terminal outcomes. Provider implementations vary, so SDK pollers are usually safer than hand-written sleep loops. The operation record and correlation identifier should be retained as evidence. [7] [8] [15] [16]

Retries create another risk. A client may time out after the provider accepted a create request. Blindly repeating a non-idempotent operation can create duplicates or conflicting work. Providers may support deterministic names, ETags, idempotent PUT semantics, or client tokens. Alibaba Cloud ROS explicitly recommends ClientToken for retries that might otherwise duplicate operations. [20]

Convergence is also not instantaneous. Inventory systems, policy views, billing exports, DNS, telemetry pipelines, and cross-region replicas can lag behind the authoritative service. A robust workflow defines which source is authoritative for each decision, how long it will wait, what stale evidence looks like, and which actions are safe to repeat. “Eventually visible” is an engineering condition, not an excuse to ignore verification.

Automation hazard	Failure mode	Required design response
Accepted but incomplete	Workflow treats HTTP acceptance as resource readiness	Poll the operation and then perform service-specific readiness tests
Lost response	Client retries after provider already accepted the operation	Use deterministic identity, idempotency token, or read-before-retry reconciliation
Partial failure	Some graph nodes exist while others failed	Record created IDs, inspect dependency state, choose rollback or resume semantics
Concurrent change	Two actors overwrite each other's intended properties	Use ETag/version checks, locks where appropriate, and serialized ownership
Stale inventory	Resource appears absent, present, or noncompliant after state already changed	Define authoritative source, bounded wait, and reconciliation loop
Ready-but-not-healthy	Management reports success while service behavior fails	Separate provisioning evidence from operational SLI and synthetic test
Delete blocked	Lock, dependency, retention, or provider behavior prevents cleanup	Surface the blocker; never silently declare retirement complete

2.11 The Same Contract Applies Across Service Models

Resource-level thinking is deliberately independent of IaaS, PaaS, or AI terminology. The provider may manage more of the underlying stack as the service model rises, but the consumer still needs an addressable contract for the part that remains configurable and accountable. The resource identity moves upward: from machine and disk, to application and database, to model deployment, knowledge base, evaluation, and agent.

This movement is one reason AI PaaS should not be described as "an API plus a portal." The engineering gain appears when model, knowledge, safety, evaluation, tracing, and agent configuration become typed resources with independent authority, lifecycle, evidence, and cost. The resource boundary gives platform teams something they can provision, review, release, audit, optimize, and retire.

Service layer	Representative resource	Data-plane capability	Resource-level management questions
IaaS	Virtual machine, disk, network, load balancer	Run processes, store blocks, route packets	Size, placement, identity, network, availability, replacement, backup, cost
Application PaaS	Web app, function, database, message topic	Serve requests, execute functions, query data, deliver messages	Runtime/version, scaling, connection, trigger, deployment, health, retention
Cloud-native platform	Cluster, namespace, workload, ingress, persistent volume	Schedule containers and expose services	Controller ownership, desired state, policy, workload identity, rollout, drift
AI PaaS	Model deployment, knowledge base, evaluation, safety policy	Infer, retrieve, evaluate, filter	Model/version, capacity, data boundary, quality, latency, token cost, retirement
Agent platform	Agent definition/version, tool connection, runtime, trace	Plan, retrieve, invoke tools, produce outcomes	Authority, memory, tool risk, evaluation gate, audit, cost, rollback, retirement

2.12 Translating the Contract Across Clouds

The contract remains stable even when cloud product names and resource granularity differ. Azure makes the model especially visible through Azure Resource Manager, provider namespaces, resource IDs, Resource Graph, Activity Log, Bicep, and deployment stacks. Alibaba Cloud combines account and resource organization with Resource Management, resource types and stacks in ROS, ActionTrail, and service-specific APIs. Tencent Cloud uses project and tag organization, CAM authorization, service APIs, and CloudAudit. Volcengine exposes project grouping, IAM, infrastructure-as-code providers, service resource APIs, and CloudTrail/TLS-based audit collection. [4] [15] [16] [17] [18] [19] [21] [22] [23] [24] [25] [26] [27]

Cross-cloud comparison should not force false equivalence. One provider may expose a feature as a first-class resource while another places the same behavior inside a larger application object. One may provide a detachable child resource while another requires replacement of the parent. One may expose operational state through a managed metric while another requires an application probe. The comparison should record the difference instead of hiding it behind a generic product category.

Engineering intent	Azure anchor	Alibaba Cloud anchor	Tencent Cloud anchor	Volcengine anchor
Name and type resources	ARM resource ID + provider/type	Resource ID/ARN + product/ROS type	Resource identifiers + product API type	Resource identifiers + service API type
Declare a graph	Bicep/ARM/Terraform; deployment stacks	ROS/Terraform	Terraform and product APIs/templates	Terraform/Pulumi and product APIs
Track management state	Provisioning state and long-running operation status	Stack/resource status and service operation state	Service API task/state fields	Service API task/state fields
Inspect estate	Resource Graph	Resource Center / Resource Management	Projects, tags, service inventories	Projects, tags, service inventories
Audit lifecycle events	Activity Log	ActionTrail	CloudAudit	CloudTrail logs delivered to TLS
End management	Delete, lock, stack delete/detach, service retention	Stack delete and service retention rules	Product-specific delete/retention behavior	Product-specific delete/retention behavior

2.13 The Resource-Level Review

The following review is used by every later chapter and lab. It is intentionally compact. A team should be able to complete it before selecting a cloud service, exposing it as an agent tool, or declaring an implementation production-ready.

A negative answer does not always disqualify the service. It identifies where the customer must add a wrapper, external evidence, process control, or a stronger isolation boundary. The important result is explicit responsibility.

Review question	Expected answer or evidence	Reject or escalate when…
1. What is the durable identity?	Provider, type, resource ID, parent scope, location	Only a client object, URL, or secret identifies the capability
2. What is the management endpoint?	Documented control-plane API/SDK/IaC contract and version	Lifecycle depends on manual provider intervention

Review question	Expected answer or evidence	Reject or escalate when…
3. What is the data endpoint?	Protocol, caller identity, network path, request evidence	Management and use share an undifferentiated super-credential
4. What desired state can be declared?	Typed properties, defaults, immutable fields, dependencies	Critical configuration exists only in undocumented portal state
5. How is provisioning observed?	Long-running operation, status, terminal error and correlation ID	Automation must infer completion from elapsed time
6. How is readiness proven?	Health, sample transaction, quality/freshness or synthetic test	Provisioning success is treated as production readiness
7. What constrains authority and configuration?	RBAC/IAM, policy, network, quota, locks, approvals	The only safe role is account-wide administrator
8. What is the dependency graph?	Owned, attached, routed, authorized, observed, and shared dependencies	Deletion or replacement effects are unknown
9. What does change mean?	In-place, restart, rollout, revision, replacement, or migration semantics	An update can silently replace identity or lose state
10. What evidence survives?	Inventory, activity/audit events, telemetry, version and cost records	Incidents or changes cannot be reconstructed
11. What is metered and limited?	SKU, meter, quota, capacity, budget, allocation scope	Cost or capacity has no accountable owner
12. How does it retire?	Delete/detach/retain/recover/purge, credential and endpoint cleanup	A clean console is the only proof of cleanup

Resource-Level Review Exercise

Choose one capability that will appear later in the book—for example a virtual machine, object store, managed web app, model deployment, search index, knowledge base, or agent definition. Complete the twelve-question review using only official documentation and evidence visible in your account.

Do not score the capability with a single number. Produce a one-page decision record with three sections: verified resource properties; provider- or service-specific gaps; and customer controls required to close those gaps. Save the resource ID, management API version, one successful management event, one readiness test, and the documented retirement behavior. This is a design review, not a full lab, and should not create a long-lived or expensive environment.

Decision rule

A capability can enter a governed architecture when its resource identity and lifecycle are verified, its gaps have explicit owners, and its operational use can be correlated with management and cost evidence.

CloudDesk Reuse: Resource Description and Lifecycle Assessment

The CloudDesk FinOps Agent described in the companion implementation work can reuse this chapter as a capability contract. The Agent should never infer lifecycle or authority from a product name alone. It should retrieve the provider resource identity, inspect current management and operational state, traverse relevant dependencies, and produce evidence before recommending or executing a change.

```
capability: cloud.resource.lifecycle.assess
inputs:
  provider: azure | alibaba | tencent | volcengine
  resource_id: string
  requested_operation: inspect | change | replace | retire
outputs:
  resource_type: string
  parent_scope: string
  control_plane_state: object
  operational_evidence: object
  dependencies: array
  change_semantics: in_place | restart | rollout | replace | migrate | unknown
  retirement_semantics: delete | detach | retain | recover | purge | unknown
  evidence_refs: array
risk_controls:
  - read_only_by_default
  - require_plan_for_mutation
  - require_approval_when_identity_or_data_boundary_changes
  - verify_residual_inventory_and_cost_after_retirement
```

The contract does not implement a provider adapter. It defines what every adapter must return. That separation allows the Agent to compare clouds without pretending their APIs or lifecycle details are identical.

Chapter Summary

- Resource-level is a governance test: a capability must have a durable identity and a management lifecycle independent of one client process.

- A governed resource joins identity, control-plane and data-plane contracts, state, evidence, economics, dependencies, and retirement.
- Desired state, provisioning state, and operational state answer different questions and require different evidence.
- A solution is a graph of resources, and shared dependencies often require lifecycle boundaries separate from the application that uses them.
- Lifecycle extends beyond CRUD to validation, readiness, operation, controlled change, recovery, evidence preservation, and retirement.
- Update may mean mutation, restart, rollout, revision, replacement, or migration; automation must know which semantics apply.
- Delete, detach, soft delete, retain, recover, and purge are different architectural outcomes.
- Asynchronous operations, retries, stale inventory, and convergence require correlation, idempotence, reconciliation, and explicit readiness tests.
- The same resource contract applies to IaaS, application PaaS, cloud-native platforms, AI PaaS, knowledge resources, and agent platforms.
- Cross-cloud comparison should preserve differences in resource granularity and lifecycle rather than force product-name equivalence.

References and Official Documentation

Product and documentation state checked August 25, 2026. Provider schemas, API versions, resource types, lifecycle states, retention behavior, and regional availability should be reverified before final manuscript freeze.

[1] NIST SP 800-145. The NIST Definition of Cloud Computing. On-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. https://csrc.nist.gov/pubs/sp/800/145/final

[2] NIST SP 500-292. NIST Cloud Computing Reference Architecture. Resource abstraction and control layer, service layer, cloud service management, actors, and orchestration. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-292.pdf

[3] NIST SP 500-322. Evaluation of Cloud Computing Services Based on NIST SP 800-145. Methodology for testing whether a capability meets the cloud model. https://www.nist.gov/publications/evaluation-cloud-computing-services-based-nist-sp-800-145

[4] Microsoft Learn. What is Azure Resource Manager? Deployment and management layer, resource groups, declarative management, identity, tags, and lifecycle guidance. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/overview

[5] Microsoft Learn. Control plane and data plane operations. Plane-specific endpoints, authorization, locks, and operational behavior. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/control-plane-and-data-plane

[6] Microsoft Learn. Azure resource providers and types. Provider namespaces, resource types, registration, and operations. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/resource-providers-and-types

[7] Microsoft Learn. Use the Azure SDK for Go for control plane operations. Defines the control plane as the APIs controlling creation, update, configuration, and deletion. https://learn.microsoft.com/en-us/azure/developer/go/control-plane

[8] Microsoft Learn. Status of asynchronous operations. Azure-AsyncOperation, Location, Retry-After, provisioningState, and terminal outcomes. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/async-operations

[9] Microsoft Learn. Azure REST API reference. REST endpoints, HTTP operations, authentication, requests, and responses for Azure resources. https://learn.microsoft.com/en-us/rest/api/azure/

[10] Microsoft Learn. Set resource dependencies in Bicep. Implicit and explicit dependencies and dependency visualization. https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/resource-dependencies

[11] Microsoft Learn. Define the order for deploying resources in ARM templates. dependsOn and deployment sequencing. https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/resource-dependency

[12] Microsoft Learn. Create and deploy Azure deployment stacks in Bicep. Cohesive resource management and delete/detach actionOnUnmanage semantics. https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/deployment-stacks

[13] Microsoft Learn. Preview deployment stack changes with what-if. Create, modify, no-change, delete, and detach outcomes. https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/deployment-stacks-what-if

[14] Microsoft Learn. Lock your Azure resources to protect your infrastructure. CannotDelete and ReadOnly management locks and scope behavior. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/lock-resources

[15] Microsoft Learn. Overview of Azure Resource Graph. Cross-subscription inventory and governance queries. https://learn.microsoft.com/en-us/azure/governance/resource-graph/overview

[16] Microsoft Learn. Azure Activity Log. Subscription-level control-plane events, correlation, routing, retention, and analysis. https://learn.microsoft.com/en-us/azure/azure-monitor/platform/activity-log

[17] Alibaba Cloud Documentation. What is Resource Management? Resource Directory, Resource Group, Resource Sharing, Tag, and Resource Center. https://www.alibabacloud.com/help/en/resource-management/product-overview/what-is-resource-management

[18] Alibaba Cloud Documentation. ROS resource type index. Standard resource and data-source types across Alibaba Cloud products. https://www.alibabacloud.com/help/en/ros/developer-reference/list-of-resource-types-by-service

[19] Alibaba Cloud Documentation. Deploy a LNMP environment by using Ansible ali_ros_stack. Stack creation and deletion of the resources created by the template. https://www.alibabacloud.com/help/en/ros/use-cases/use-ansible-ali-ros-stack-to-deploy-a-lnmp-environment

[20] Alibaba Cloud Documentation. ROS SDK FAQ. ClientToken guidance to prevent duplicate operations during retries. https://www.alibabacloud.com/help/en/ros/support/what-do-i-do-if-the-errcode-sdk-serverunreachable-error-is-reported-when-i-use-sdks

[21] Tencent Cloud Documentation. Project Management. Group cloud products and services into project scopes for management. https://www.tencentcloud.com/document/product/378/34726

[22] Tencent Cloud Documentation. Resource Tag. Resource categorization, tag-based authorization, querying, and cost allocation. https://www.tencentcloud.com/document/product/1131/44513

[23] Tencent Cloud Documentation. CAM Overview. Users, groups, roles, policies, and resource permissions. https://www.tencentcloud.com/document/product/598/72360

[24] Tencent Cloud. CloudAudit product and documentation. Account activity, API usage, resource-change tracking, compliance, and troubleshooting. https://www.tencentcloud.com/product

[25] Volcengine Documentation. Resource project management. Project grouping and IAM authorization over resource collections. https://www.volcengine.com/docs/6257/170353?lang=en

[26] Volcengine Documentation. Terraform. Infrastructure-as-code provider setup and resource management. https://www.volcengine.com/docs/6706/101989?lang=en

[27] Volcengine Documentation. Collect CloudTrail logs. Cloud operation audit logs delivered into Torch Log Service. https://www.volcengine.com/docs/6470/2201696?lang=en

Part I / Chapter 3

Multicloud Architecture: Shared Responsibility, Platform DNA, and Engineering Intent

Translating Architecture Without Pretending Clouds Are Identical

Multicloud architecture is the disciplined translation of engineering intent across different responsibility boundaries, resource models, delivery locations, integration patterns, and economic contracts.

In This Chapter

- Why multicloud is an architecture and translation problem rather than a product-comparison exercise
- How shared responsibility moves as services become more managed, without eliminating customer accountability
- Why responsibility must include reliability, identity, data, cost, evidence, and AI outcomes—not only security
- How a four-dimensional model separates service layer, operating boundary, delivery location, and integration degree
- How platform DNA can explain path-dependent product emphasis without becoming technological determinism
- Why Azure is used as the primary teaching reference and how Alibaba Cloud, Tencent Cloud, and Volcengine expose different strengths
- How to translate engineering intent through resource, identity, network, lifecycle, evidence, economics, and non-equivalence
- Why functional, operational, governance, economic, and organizational equivalence must be assessed separately
- Which multicloud patterns are purposeful and which anti-patterns merely duplicate complexity

- How to produce a one-page engineering-intent translation record for a cloud capability

A multicloud architecture is often introduced as a list: two public clouds, perhaps one private cloud, and a collection of products that appear to correspond. That list describes purchasing and placement, but it does not yet describe architecture. Architecture begins when a team can state why a capability is placed where it is, which properties must remain invariant, which responsibilities move to the provider, which differences are accepted, and how the resulting system will be governed as one engineering estate.

The central difficulty is not that clouds use different names. It is that they package responsibility differently. A managed database, Kubernetes service, model endpoint, search system, or agent platform may provide similar application functionality in two clouds while exposing different resource boundaries, identity models, private-network options, upgrade behavior, evidence, limits, and cost allocation. A name-to-name mapping can therefore be correct at the feature level and dangerously wrong at the operating-contract level.

This chapter uses three ideas to make comparison more rigorous. Shared responsibility identifies which obligations remain with the customer for a specific service configuration. The four-dimensional multicloud model separates service layer, operating boundary, delivery location, and integration degree. Platform DNA provides a cautious editorial lens for understanding why providers repeatedly emphasize certain integration patterns. The final translation unit is engineering intent: a provider-neutral statement of desired outcome, constraints, evidence, and acceptable trade-offs.

Azure is the primary reference platform in this book because its management hierarchy and cross-service conventions make control-plane concepts unusually visible for teaching. Alibaba Cloud, Tencent Cloud, and Volcengine are not treated as secondary copies. They are first-class comparison points whose different industrial histories and product portfolios reveal where cloud abstractions are genuinely shared and where apparent equivalence breaks down.

The chapter thesis

Multicloud architecture is not the simultaneous use of several provider catalogs. It is the disciplined translation of engineering intent across different responsibility boundaries, resource contracts, operating assumptions, delivery locations, and integration patterns.

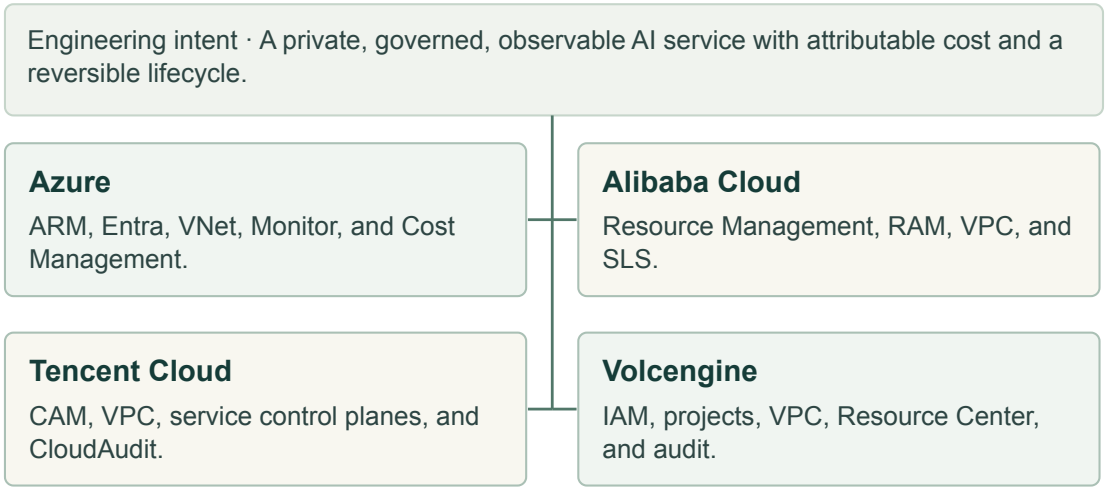


Figure 3-1 One engineering intent may require four different provider-specific resource graphs.

3.1 Multicloud Is a Translation Problem

The term multicloud is useful only after the architecture has a purpose. An organization may use several clouds because of acquisition history, regional availability, regulation, commercial commitments, customer proximity, specialized data or AI services, resilience objectives, or a deliberate avoidance of one-provider concentration. Those motivations are not interchangeable. An architecture designed for regulatory placement is evaluated differently from one designed for price arbitrage or access to a specialized model.

Hybrid cloud and multicloud also describe different aspects of a system. Hybrid emphasizes connection across deployment environments such as provider regions, customer premises, dedicated environments, or edge locations. Multicloud emphasizes the use or governance of services from more than one provider. A system can be hybrid without being multicloud, multicloud without deep interconnection, or both. The important question is not which label applies, but which resource contracts, data paths, identities, and responsibilities cross the boundaries.

A portable application artifact does not create a portable operating model. A container may run on several managed Kubernetes services, yet the cluster identity, node lifecycle, CNI integration, load-balancer behavior, storage classes, upgrade channels, audit logs, and cost meters remain provider-specific. An OpenAI-compatible inference request may work

against several model platforms while deployment capacity, content safety, data residency, rate limits, observability, and retirement policy differ. Portability must therefore be decomposed rather than asserted.

```
Multicloud decision = engineering intent
                    + responsibility boundary
                    + resource contract
                    + platform context
                    + verified non-equivalence
```

Question	Weak multicloud answer	Architectural answer
Why more than one cloud?	To avoid lock-in	A named business, regulatory, reliability, capability, or economic objective with measurable acceptance criteria
What is portable?	The application	Specific artifacts, interfaces, schemas, policies, and operating procedures, each with stated limits
What changes by provider?	Only product names	Responsibility, resource granularity, identity, networking, lifecycle, evidence, quota, and cost contract
How is success verified?	The deployment completed	The service meets functional, security, SLO, evidence, cost, and retirement requirements
What is the exit strategy?	Terraform can recreate it	Data, identity, state, DNS, secrets, evidence, commitments, and dependent resources have documented transition paths

Portability has dimensions

Source-code portability, artifact portability, API compatibility, data portability, operational portability, governance portability, and commercial portability are different claims. A design should say which claim it makes.

3.2 Shared Responsibility Is a Moving Boundary

Shared responsibility is frequently presented as a security diagram in which the provider protects the cloud and the customer protects what is placed in the cloud. That formulation is a useful beginning, but it is too narrow for architecture. The same shifting boundary applies to reliability, patching, scaling, backup, recovery, network configuration, identity, data governance, observability, cost control, and—on AI platforms—model selection, grounding, evaluation, tool authority, and acceptable outcomes.

The boundary moves upward as the provider manages more of the stack. With IaaS, the provider operates facilities, hardware, and virtualization while the customer typically controls the guest operating system, middleware, workload, identities, network rules, and data. Application PaaS transfers much of the operating-system and runtime burden to the provider, but application configuration, dependencies, data, authorization, release policy, and SLOs remain customer concerns. AI PaaS can manage model serving, capacity abstractions, safety filters, retrieval infrastructure, evaluation services, and agent runtimes, yet the customer still owns use-case fitness, data rights, access boundaries, tool permissions, evaluation criteria, and business consequences. Microsoft, Alibaba Cloud, Tencent Cloud, and Volcengine each document service-specific forms of this boundary. [2] [3] [4] [9] [10] [15] [16] [17] [23] [24] [25]

Responsibility also depends on configuration. A managed database with public access, customer-managed keys, cross-region replicas, and custom backup policy creates a different customer obligation from the same service with private access, provider-managed encryption, and default retention. A managed Kubernetes service may operate the control plane while leaving nodes, add-ons, network policy, workloads, secrets, and data protection to the customer. The service name cannot substitute for a configuration-specific responsibility review.

Shared responsibility moves upward as the service abstraction rises

	IaaS	Application PaaS	AI PaaS	SaaS / agentic SaaS
Physical facilities	Provider	Provider	Provider	Provider
Hardware and virtualization	Provider	Provider	Provider	Provider
Operating system and runtime	Customer	Provider	Provider	Provider
Platform middleware	Customer	Provider	Provider	Provider
Application logic	Customer	Customer	Customer	Provider
Identity and configuration	Customer	Customer	Customer	Customer
Data, policy, outcomes	Customer	Customer	Customer	Customer

The boundary changes by service and configuration. Responsibility for business intent, data classification, identity, and acceptable outcomes never disappears.

Figure 3-2 The responsibility boundary moves upward with service abstraction, but customer accountability for intent and outcomes remains.

Responsibility domain	IaaS emphasis	Application PaaS emphasis	AI PaaS / agent-platform emphasis
Runtime and patching	Customer operates guest OS and most middleware	Provider operates OS/runtime; customer controls supported versions, dependencies, and app configuration	Provider operates inference or agent runtime; customer controls deployment, prompts, tools, knowledge, policies, and release
Reliability	Customer designs instance, zone, backup, and recovery topology	Provider supplies platform availability mechanisms; customer defines SLO, scaling, dependency, data, and recovery requirements	Provider supplies managed serving and platform recovery; customer validates quality, fallback, rate-limit, tool, knowledge, and business continuity behavior

Responsibility domain	IaaS emphasis	Application PaaS emphasis	AI PaaS / agent-platform emphasis
Identity and network	Customer configures workload identity, firewall, routes, and exposure	Provider integrates identity/network features; customer selects scopes, private access, ingress, and callers	Customer separates human, workload, agent, retrieval, and tool identities and constrains each data path
Data and compliance	Customer classifies, encrypts, backs up, and governs data	Provider manages storage mechanisms; customer owns data rights, retention, schema, and access policy	Customer owns training/grounding rights, retrieval authorization, prompt/output handling, memory, residency, and regulatory interpretation
Evidence and cost	Customer enables telemetry, audit, tags, budgets, and optimization	Provider emits platform telemetry; customer correlates it with app outcomes and cost allocation	Customer correlates model, retrieval, tool, safety, evaluation, token, capacity, and business-outcome evidence

Managed does not mean unowned

A provider can automate an operation without accepting the customer's responsibility for choosing the correct policy, validating the result, meeting a business SLO, or proving compliance. Automation changes who performs work; it does not automatically transfer accountability.

3.3 Responsibility Extends Beyond Security

Security is the most familiar shared-responsibility domain because unauthorized access has immediate consequences. Production architecture requires at least five additional responsibility ledgers. A reliability ledger states who designs redundancy, detects failure, invokes recovery, and tests the result. A data ledger states who controls classification, residency, lineage, backup, deletion, and legal basis. An operations ledger states who patches, upgrades, observes, responds, and retires. An economics ledger states who allocates cost, manages commitment, responds to anomalies, and proves savings. An AI ledger states who selects models and knowledge, evaluates behavior, constrains tools, handles harmful output, and accepts business outcomes.

These ledgers reveal gaps that a generic provider/customer diagram hides. For example, a provider may guarantee the availability of a managed search service while the customer remains responsible for index freshness, document-level authorization, retrieval quality, alias rollout, and the correctness of citations. A model endpoint may satisfy an in-

frastructure SLA while the application fails because a model version was retired, quota was exhausted, retrieval returned unauthorized evidence, or a tool executed beyond its intended scope.

A multicloud design should create the ledgers before choosing equivalent products. The exercise forces the team to identify obligations that must be implemented once at an organizational layer, obligations that can be delegated to provider-native controls, and obligations that remain application-specific in every cloud.

Ledger	Questions that must have named owners	Typical evidence
Security	Who grants authority, protects secrets, constrains networks, detects abuse, and revokes access?	Role assignments, policy results, firewall rules, key history, audit events
Reliability	Who defines SLOs, redundancy, backup, recovery, failover, capacity, and tests?	Architecture decision, restore tests, failover records, health and burn-rate signals
Data	Who classifies, locates, retains, deletes, and authorizes data and derived knowledge?	Catalog, lineage, retention policy, access filter, deletion and purge evidence
Operations	Who upgrades, changes, observes, responds, rolls back, and retires?	Release records, telemetry, incidents, change approvals, retirement packet
Economics	Who allocates, forecasts, budgets, commits, optimizes, and validates value?	Tags, billing export, allocation rules, budgets, commitments, realized-savings evidence
AI behavior	Who approves models, knowledge, prompts, tools, evaluations, safety boundaries, and human review?	Evaluation suite, safety results, tool policy, trace, approval record, outcome review

3.4 A Four-Dimensional Model for Cloud Placement

The usual public/private/hybrid/multicloud vocabulary mixes several decisions into one label. This book uses a four-dimensional model so that those decisions can be made independently. The first dimension is service layer: whether the consumer receives infrastructure, a managed application platform, software, or an AI-specific platform capability. The second is operating boundary: whether the consumer self-manages the capability, shares operations with the provider, or consumes a fully managed service. The third is delivery location: where the service and data execute. The fourth is integration degree: how deeply identity, network, billing, policy, monitoring, development, and support are integrated with the surrounding cloud.

The model avoids an important category error. A service delivered inside a customer facility can still be provider-operated. A database physically placed in another cloud can retain the original provider's operating model. A container platform can run on premises while being governed from a public-cloud control plane. Conversely, two services in the same provider region can remain poorly integrated if they use separate identities, billing scopes, policies, and operational evidence.

Each workload can therefore be represented as a point—or a trajectory—in four dimensions. Migration may change only delivery location while preserving service layer and operating boundary. Modernization may move from self-managed IaaS to managed PaaS without changing region. Multicloud integration may increase identity and billing integration while leaving separate providers responsible for their respective services.

Four dimensions of multicloud placement

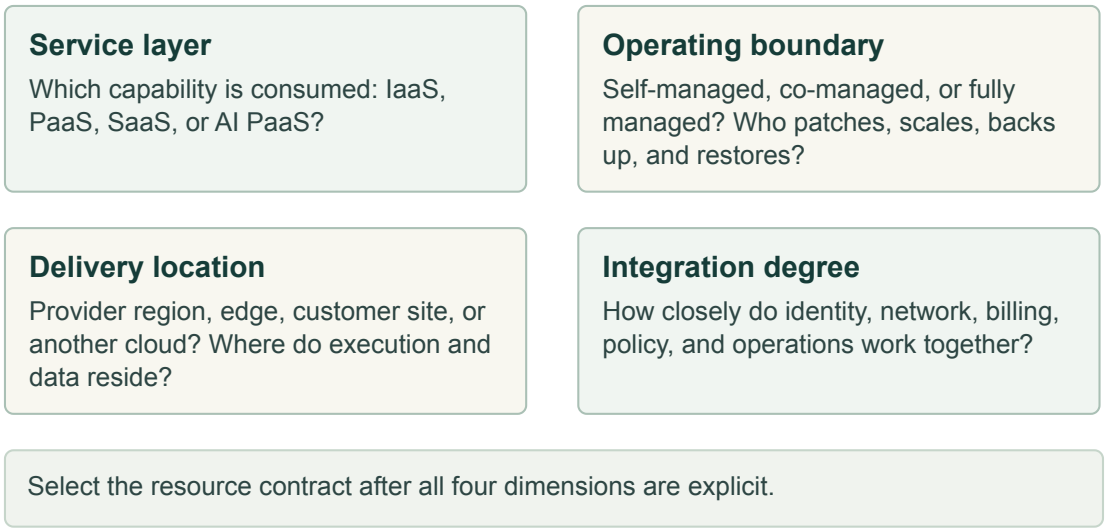


Figure 3-3 The four-dimensional model separates service layer, operating boundary, delivery location, and integration degree.

Dimension	Representative choices	Primary design question	Common mistake
Service layer	IaaS, application PaaS, SaaS, AI PaaS	What capability and abstraction is consumed?	Treating service layer as a maturity ranking rather than a responsibility choice
Operating boundary	Self-managed, co-managed, fully managed	Who patches, scales, backs up, restores, upgrades, and supports?	Assuming "managed" transfers every operational duty

Dimension	Representative choices	Primary design question	Common mistake
Delivery location	Provider region, another cloud, customer site, sovereign environment, edge	Where do processing, data, keys, and failure domains reside?	Equating physical location with operator or control plane
Integration degree	Identity, network, billing, policy, telemetry, developer workflow, support	How much should separate services behave as one cloud estate?	Calling connected services integrated when evidence and governance remain fragmented

Use the dimensions independently

Do not infer operating boundary from delivery location, or integration degree from provider branding. Write all four values into the architecture decision record.

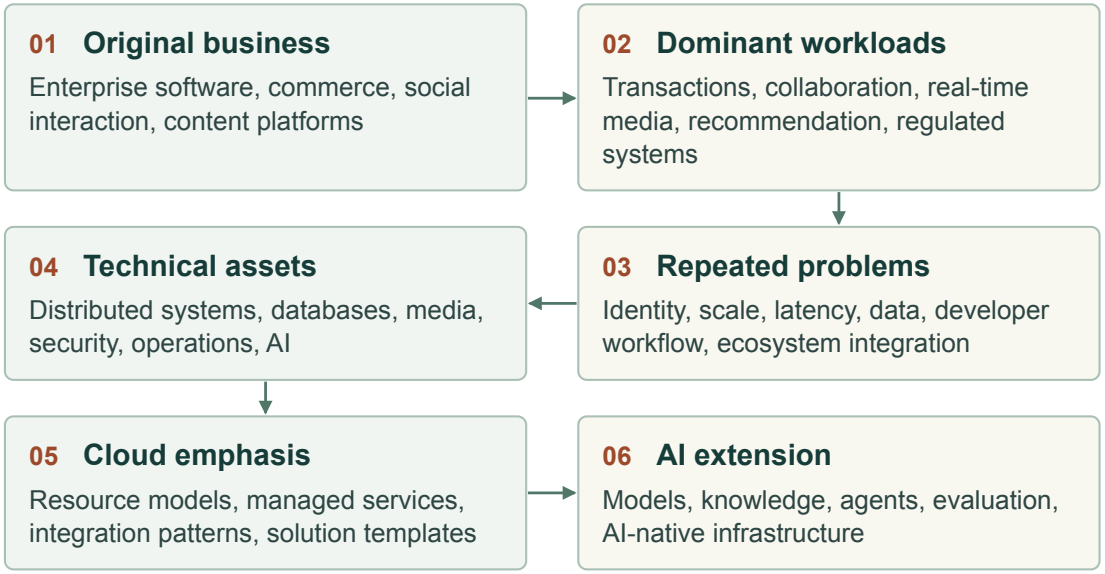
3.5 Platform DNA: Path Dependence Without Determinism

Cloud platforms are not designed on a blank sheet. Providers carry technical assets, operating habits, customer relationships, developer ecosystems, and business incentives from the industries in which they first achieved scale. This book calls that accumulated path dependence platform DNA. The term is an editorial lens, not a vendor classification and not a claim that a provider can serve only its original market.

The useful causal chain is: original business, dominant workloads, repeated engineering problems, reusable technical assets, cloud product emphasis, and AI extension. Enterprise software produces repeated pressure for identity, manageability, backward compatibility, hybrid operation, and organizational governance. Commerce produces pressure for transaction scale, merchants, search, data processing, elasticity, promotion peaks, and ecosystem integration. Social, gaming, and real-time media produce pressure for connection, low latency, content delivery, interactive services, and developer distribution. Content and recommendation platforms produce pressure for data pipelines, experimentation, ranking, media processing, and AI-native infrastructure.

DNA becomes dangerous when it is used as proof. A provider's origin cannot establish a current SLA, security property, price advantage, regional availability, or technical superiority. Acquisitions, regulation, open-source participation, customer mix, and strategic investment change portfolios. The correct use is to form a hypothesis about likely emphasis, then verify the present resource contract in official documentation and in a controlled implementation.

Platform DNA is a path-dependent chain, not a deterministic label



Evidence test

Use DNA to form a hypothesis about product emphasis. Confirm architecture, limits, control-plane behavior, and operational evidence in current official documentation.

Acquisitions, regulation, customer mix, open-source ecosystems, and strategic investment can change the trajectory. DNA explains path dependence; it does not predict every product decision.

Figure 3-4 Platform DNA links original business pressures to recurring cloud and AI platform emphases, but every conclusion requires current evidence.

Safe use of platform DNA	Unsafe use
Explain why certain product integrations, documentation patterns, or solution templates are especially visible	Claim that a provider is inherently best for every workload associated with its original business
Generate questions for architecture review and comparative testing	Replace service documentation, benchmark data, contractual terms, or regional verification
Interpret portfolio coherence and likely organizational fit	Treat historical origin as an immutable technical limitation

Safe use of platform DNA	Unsafe use
Identify migration friction caused by surrounding ecosystems and operating habits	Infer compliance, sovereignty, availability, or security from brand reputation
Describe path dependence while acknowledging convergence and strategic change	Use cultural stereotypes or marketing slogans as architecture evidence

3.6 Four Clouds, Four Visible Emphases

The following profiles are deliberately concise. They describe visible emphases that help explain product coherence and teaching value; they are not rankings. Every implementation decision still uses the engineering-intent translation described later in the chapter.

3.6.1 Azure: Enterprise Software, Developer Platforms, and Hybrid Consistency

Microsoft entered cloud computing with decades of enterprise software, operating-system, database, productivity, identity, and developer-tool experience. Its current reporting still places Azure, GitHub, SQL Server, Windows Server, Visual Studio, virtual desktop, and enterprise services within a broad Intelligent Cloud business, while Microsoft 365 and related products reinforce identity and organizational integration. [8]

That history is visible in Azure Resource Manager, management groups, subscriptions, resource groups, provider namespaces, Entra identity, Azure Policy, Monitor, Cost Management, hybrid services, and a consistent emphasis on enterprise governance. Azure is used as the primary teaching reference in this book because these cross-service conventions make resource hierarchy, responsibility, identity, policy, evidence, and cost relatively easy to expose as one coherent control-plane model. This is an editorial and pedagogical judgment, not a claim of universal technical superiority. [5] [6] [7]

3.6.2 Alibaba Cloud: Commerce Scale, Transactions, Data, and Ecosystem Operations

Alibaba describes commerce and cloud computing as core businesses. Its commerce ecosystem created sustained pressure for large transaction volumes, merchant operations, search, logistics coordination, promotion peaks, data processing, and partner integration. Its cloud portfolio extends those assets through elastic compute, storage, networking, security, big-data, and AI services. [12]

This path is visible in the breadth of Alibaba Cloud's infrastructure, data, resource-management, orchestration, and industry-platform services. Resource Directory, Resource Groups, Resource Center, tags, RAM, ROS, ActionTrail, and managed data and AI platforms provide the control-plane anchors needed for enterprise comparison. The architectural lesson is not that commerce workloads must use Alibaba Cloud; it is that commerce-scale operating experience helps explain why data, transaction, elasticity, and ecosystem patterns are prominent. [9] [10] [11]

3.6.3 Tencent Cloud: Connection, Interaction, Games, and Real-Time Media

Tencent's business portfolio is strongly associated with social networks, communication, games, video, Mini Programs, content, and digital services. Those workloads create repeated demands for low-latency interaction, real-time audio and video, content delivery, identity at large user scale, ecosystem distribution, and integration with consumer and enterprise applications. Tencent's 2025 reporting continues to describe games, social networks, Video Accounts, Mini Programs, cloud, and AI as major business areas. [18]

Tencent Cloud's platform should therefore be examined carefully where connection, interactive media, games, application ecosystems, and real-time services matter. Its control-plane mapping uses CAM for authority, projects and tags for organization, service-specific resource APIs, CloudAudit for management evidence, and provider-specific network and data services. Again, the DNA lens proposes where to look; the current service contract determines the decision. [13] [14] [15] [16] [17]

3.6.4 Volcengine: Content, Recommendation, Data, Media, and AI-Native Infrastructure

Volcengine is ByteDance's cloud and AI platform. ByteDance's public product portfolio includes global and Chinese content, creation, collaboration, and information platforms such as TikTok, Douyin, CapCut, Lark, Feishu, and Toutiao. Those systems create sustained pressure for recommendation, experimentation, data processing, media pipelines, content understanding, growth analytics, and large-scale AI. [26] [27]

The resulting emphasis is visible in Volcengine's data and AI positioning, multimedia capabilities, Doubao model services, Ark, AgentKit, and cloud-native AI infrastructure. Its control-plane anchors include projects, IAM, Resource Center, tags, service APIs, and audit services. The platform is particularly valuable in this book because it demonstrates how an AI-native narrative can grow from an internet content and recommendation lineage rather than from an enterprise software or commerce lineage. [19] [20] [21] [22] [23] [24] [25]

Provider	Historic pressure visible in the portfolio	Control-plane emphasis to inspect	Do not infer without evidence
Azure	Enterprise software, databases, productivity, identity, developer tools, hybrid estates	Hierarchical resource governance, identity, policy, monitoring, cost, enterprise and hybrid integration	Best fit, lowest cost, or strongest service for every enterprise workload
Alibaba Cloud	Commerce, transactions, merchants, logistics, search, data, promotion-scale elasticity	Elastic infrastructure, data platforms, resource organization, orchestration, industry and AI services	Automatic superiority for every commerce or data workload
Tencent Cloud	Social connection, communication, games, video, Mini Programs, interactive services	Identity and ecosystem integration, real-time media, connectivity, service APIs, content and gaming capabilities	Universal low latency or direct equivalence with Tencent consumer platforms
Volcengine	Content, recommendation, experimentation, media processing, growth analytics, AI	Data/AI platforms, multimedia, model and agent services, projects, IAM, resource and audit controls	That “AI-native” removes the need to verify governance, lifecycle, isolation, or cost

3.7 Engineering Intent Is the Translation Unit

A product is not the correct unit of multicloud translation. The correct unit is engineering intent: a provider-neutral statement of the result to achieve, the constraints to preserve, the evidence to collect, and the trade-offs the organization is willing to accept. “Use Azure AI Search” is a product decision. “Provide private, document-authorized, hybrid retrieval with versioned ingestion, citations, latency objectives, auditable queries, and tenant-level cost allocation” is an engineering intent.

Intent must be more precise than a feature request. It should name business outcome, data classification, availability and latency targets, authority model, placement constraints, change and retirement requirements, evidence, cost owner, and portability expectation. Those invariants are then translated into each provider’s resource model. The translation may select a single managed resource, a composition of resources, or a customer-built control that compensates for a missing provider feature.

The final record must also state non-equivalence. If one provider supports document-level authorization natively while another requires filtering in application code, the difference is architectural. If one model platform exposes dedicated throughput as a deployment re-

source while another manages capacity through account quota, the difference affects isolation, evidence, cost, and operations. A professional comparison preserves those differences instead of flattening them into a checkbox.

Engineering-intent translation preserves invariants and records non-equivalence

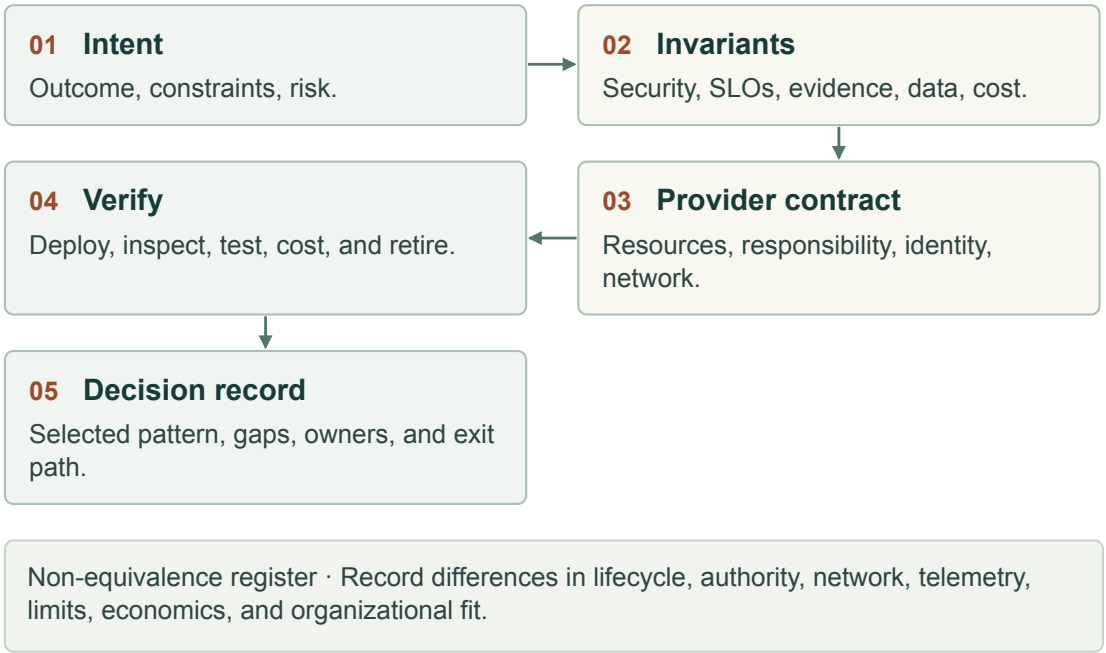


Figure 3-5 Engineering-intent translation preserves invariants, verifies provider contracts, and records non-equivalence.

Translation lens	Question	Evidence to collect
Resource model	What named resources, child objects, scopes, and dependencies realize the intent?	Resource IDs, types, hierarchy, API/IaC schemas, dependency graph
Responsibility	Which duties are provider, customer, or shared for this exact configuration?	Service responsibility documentation, operating procedure, RACI
Identity	Which human, workload, agent, retrieval, and tool principals exist, and at what scope?	Role definitions, assignments, tokens/credentials, denied-operation test

Translation lens	Question	Evidence to collect
Network and placement	Where do control and data paths run, and where do data, keys, and failure domains reside?	Topology, private endpoint/DNS, route, region, residency, exposure test
Lifecycle	How are resources created, updated, upgraded, rolled back, recovered, and retired?	IaC, operation states, change plan, restore and cleanup evidence
Evidence and SLO	Can changes, requests, quality, reliability, and incidents be reconstructed?	Audit events, logs, metrics, traces, evaluations, correlation IDs
Economics	What is metered, limited, committed, budgeted, allocated, and optimized?	SKU/meter, quota, billing export, tags, budget, unit-cost model
Non-equivalence	Which invariant is weaker, externalized, costlier, or operationally different?	Gap register, customer compensating control, accepted risk, owner

3.8 Equivalence Is Multidimensional

Two services are rarely simply equivalent or not equivalent. Equivalence is a relation defined against a purpose. Object stores may be functionally equivalent for basic PUT and GET operations while differing in identity integration, event delivery, lifecycle policy, immutability, replication, consistency behavior, private networking, inventory, and cost. Managed Kubernetes services may run the same Deployment manifest while differing in cluster upgrades, node responsibility, workload identity, networking, ingress, storage, policy, and observability.

AI services make the distinction more visible. An OpenAI-compatible API can reduce application migration effort, but it does not establish governance equivalence. Model versions, content safety, capacity, quota, private access, data treatment, evaluation, tracing, regional availability, retirement notice, and billing may differ. API compatibility is valuable, but it is one layer of the resource contract.

Equivalence dimension	Meaning	Example of a mismatch
Interface	Calls, schemas, protocols, or artifacts can be reused	OpenAI-compatible request works, but streaming events or tool schema differ
Functional	The service produces the required application capability	Both retrieve documents, but only one supports required hybrid ranking or authorization

Equivalence dimension	Meaning	Example of a mismatch
Operational	Teams can deploy, observe, recover, upgrade, and retire with comparable effort	Same Kubernetes API, different node, upgrade, ingress, and backup responsibility
Governance	Identity, network, policy, audit, residency, and evidence satisfy the same controls	One service supports private endpoint and scoped roles; another requires public access or broad credentials
Economic	Metering, quota, commitments, allocation, and unit cost behave acceptably	Equivalent throughput requires dedicated capacity in one cloud and variable pay-per-request in another
Organizational	The platform fits skills, support, procurement, regulatory, and ecosystem constraints	A technically strong service cannot be operated or purchased under the organization's model

```
Engineering equivalence is always relative to a declared intent and evidence
set.
It is not a permanent property of two product names.
```

3.9 Purposeful Multicloud Patterns

Multicloud becomes valuable when the placement pattern follows a named objective. The architecture should minimize the number of boundaries that must be crossed at runtime. Cross-cloud data transfer, identity federation, distributed tracing, incident coordination, consistency, and egress cost are real system components, not footnotes. A design that duplicates every layer in every provider often doubles operational surface without providing independent failure protection or meaningful optionality.

The patterns below are not mutually exclusive. A global organization may use provider-native platforms for regional systems, a portable container baseline for selected workloads, a federated governance layer for inventory and policy, and a specialized AI service from another provider. The point is to choose each pattern intentionally rather than to apply one portability doctrine to the entire estate.

Pattern	Use when	Primary benefit	Primary cost or risk
Provider-native platform by domain	A provider offers a materially better managed capability or ecosystem fit	Maximum use of managed services and local integration	Greater migration effort and provider-specific skills
Portable workload baseline	A workload genuinely benefits from common artifacts and interfaces	Repeatable build and deployment across environments	Lowest-common-denominator design if portability is overextended

Pattern	Use when	Primary benefit	Primary cost or risk
Primary plus recovery provider	A second provider addresses a quantified concentration or continuity risk	Independent recovery path for selected business capabilities	Data replication, identity, runbook, testing, and dormant-cost complexity
Regional or regulatory placement	Data, customers, licenses, or regulation require different locations/providers	Local compliance, latency, or market access	Fragmented operations and evidence unless governance is federated
Specialized service composition	A model, data, media, security, or industry service is uniquely valuable	Access to differentiated capability without migrating the whole estate	Cross-cloud data path, egress, latency, identity, and contractual coupling
Federated control and evidence	Resources remain provider-native but need common inventory, policy intent, FinOps, and incident context	Organization-level visibility without pretending one provider API owns all resources	Normalization can hide provider-specific semantics if source evidence is discarded

3.10 Failure Modes in Multicloud Design

The most common multicloud failures begin as simplifications. Teams match product names, draw symmetric boxes, assume common APIs create common operations, or select the lowest shared feature set. The resulting architecture looks portable in a diagram but accumulates hidden provider adapters, unowned responsibilities, fragmented evidence, and inconsistent security controls.

A second class of failure occurs when the platform becomes the abstraction goal. A custom layer is built to hide every provider difference, but the layer must then reproduce identity, networking, policy, observability, quota, cost, failure behavior, and release semantics. The organization has created a new cloud control plane without staffing it as a product. An abstraction is justified only when it removes repeated complexity while preserving the evidence needed to operate provider-native resources safely.

Anti-pattern	Why it fails	Correction
Product-name matching	Similar labels hide different scopes, lifecycle, responsibility, and evidence	Translate a declared engineering intent through the eight lenses
Symmetric architecture diagrams	Visual symmetry implies false operating equivalence	Show provider-specific resource graphs and mark asymmetry explicitly

Anti-pattern	Why it fails	Correction
Lowest common denominator	Managed capabilities are discarded to preserve hypothetical portability	Choose portability only for artifacts and interfaces that have a real transition objective
API compatibility as proof	Compatible requests do not prove identity, safety, capacity, telemetry, or retirement equivalence	Assess the complete resource and operating contract
One control plane to hide all differences	The abstraction layer inherits every provider edge case and becomes an underfunded platform product	Federate evidence and intent; keep provider-native semantics visible
Unowned shared responsibility	Each team assumes the provider or another team covers the gap	Maintain service- and configuration-specific responsibility ledgers
Data path afterthought	Egress, latency, authorization, residency, and consistency dominate runtime behavior	Design cross-cloud data movement before selecting services
Platform DNA as stereotype	History is mistaken for current technical proof	Use DNA only to generate questions; verify current contracts and evidence

A useful asymmetry

A good multicloud architecture is allowed to be intentionally asymmetric. Different providers may play different roles because the engineering intent, responsibility boundary, or platform fit differs. Symmetry is not a design virtue by itself.

3.11 Capability Check C2: Translate Resource Inventory and Ownership

This capability check produces a one-page engineering-intent translation rather than a full deployment. The chosen intent is deliberately foundational: discover cloud resources within an organizational scope, associate each resource with an accountable owner and cost center, and preserve evidence of the management hierarchy. The same capability will later support governance, FinOps, security review, and CloudDesk resource inventory.

The exercise should be completed with official documentation and one account in each provider that is available to the reader. A four-provider implementation is not required. Complete Azure as the reference, then translate at least one Chinese cloud deeply enough to identify a real non-equivalence. The remaining providers may be documented at the resource-contract level.

Engineering-intent field	Required statement for this check
Outcome	Produce an inventory of resources in a defined organizational scope with provider ID, type, location, owner, cost center, and management parent
Invariants	Read-only discovery; no customer data-plane reads; least privilege; source resource IDs retained; unknown ownership reported rather than guessed
Evidence	Scope definition, identity/role, inventory query or export, tag/metadata result, one management audit event, and timestamp
Economics	Inventory operation should be free or negligible; no long-lived compute; note billing/export latency
Retirement	Remove temporary identities, exports, scripts, and test tags; preserve only the decision record and sanitized evidence
Non-equivalence focus	Hierarchy, cross-account coverage, tag enforcement, inventory latency, audit retention, and cost-allocation schema

Provider realization	Resource and organization anchors	Authority and evidence	Known translation caution
Azure reference	Management group → subscription → resource group → ARM resource; Resource Graph; tags	Entra principal + scoped RBAC; Activity Log; Cost Management scope	Resource Graph, policy, billing, and Activity Log have different freshness and retention semantics
Alibaba Cloud	Resource Directory/account → resource group → service resource; Resource Center; tags	RAM identity/policy; ActionTrail; billing/cost allocation services	Resource coverage and cross-account aggregation vary by service and organizational setup
Tencent Cloud	Organization/account/project or service scope → resource; projects and tags; service inventories	CAM identity/policy; CloudAudit; billing data	Project, tag, and service-resource boundaries do not necessarily match Azure resource groups
Volcengine	Account/project → service resource; Resource Center; tags	IAM identity/policy; CloudTrail/audit services; billing data	Project support, Resource Center coverage, and audit integration must be verified per service

```
intent_id: resource.inventory-and-ownership
scope:
  provider: azure
  management_scope: /subscriptions/<subscription-id>
principals:
```

```
  discovery_identity: <read-only-principal-id>
invariants:
  - no data-plane content access
  - preserve provider resource IDs
  - report missing owner or costCenter as a finding
translation:
  resource_model: management-group/subscription/resource-group/resource
  inventory: Azure Resource Graph
  ownership_metadata: tags.owner + tags.costCenter
  management_evidence: Azure Activity Log
non_equivalence:
  - tag policy and billing allocation are separate control-plane contracts
  - inventory and cost data have different freshness windows
verdict: acceptable-with-documented-gaps
```

Verification requires more than a screenshot of a resource list. Save the exact scope, caller identity, query or export definition, timestamp, and at least one resource ID. Confirm that the caller cannot read a protected data-plane object merely because it can discover the resource. Record resources with missing owner or cost-center metadata as findings. Finally, document which provider-native control would prevent or remediate the gap.

The result should fit on one page. It is not a product matrix. It is a decision record showing how one intent is realized, which obligations remain with the customer, and where provider differences affect governance or future automation.

Acceptance criterion

The translation passes when a reviewer can trace the intent to named provider resources and scopes, reproduce the evidence, identify at least one meaningful non-equivalence, and see who owns the compensating control.

CloudDesk Reuse: Provider-Intent Translation

CloudDesk FinOps Agent can reuse the chapter as a read-only translation capability. The Agent should not choose a product by matching names. It should accept an engineering intent and invariants, inspect provider resource contracts, return provider-specific realizations, and preserve a non-equivalence register. Any later execution capability must remain separate and require its own authority, plan, approval, and evidence.

```
capability: cloud.capability.translate
risk_level: R0
mode: read_only
inputs:
  intent_id: string
```

```
  provider_targets: array
  required_invariants: array
  workload_context:
    data_classification: string
    regions: array
    slo: object
    cost_owner: string
outputs:
  provider_realizations: array
  responsibility_matrix: object
  resource_contracts: array
  evidence_sources: array
  non_equivalence_register: array
  recommended_pattern: string
  unresolved_questions: array
execution_constraints:
  - preserve provider-native identifiers and terminology
  - distinguish documented facts from architectural inference
  - never infer equivalence from product name or API compatibility
  - require human review before any provider-selection or change action
```

CloudDesk use	Value of this chapter	Required guardrail
FinOps inventory	Translate account, project, resource, tag, billing, and evidence scopes across providers	Do not normalize away provider-native IDs or billing semantics
Architecture recommendation	Compare operating boundary, placement, integration, and non-equivalence	Return options and evidence; do not claim a universally best cloud
Tool selection	Match an intent to governed capability cards rather than raw APIs	Identity, network, lifecycle, risk, cost, and evidence must be explicit
Migration planning	Separate portable artifacts from provider-specific operating contracts	Include data, identity, commitments, telemetry, and retirement in the plan

Chapter Summary

- Multicloud architecture begins with a purpose and an engineering intent, not with a list of provider products.
- Shared responsibility moves as service abstraction rises, but accountability for identity, data, policy, business intent, cost, and acceptable outcomes remains with the customer.
- Responsibility should be recorded across security, reliability, data, operations, economics, and AI behavior.

- The four-dimensional model separates service layer, operating boundary, delivery location, and integration degree.
- Platform DNA is a cautious path-dependence lens: original businesses create repeated workloads and technical assets that influence cloud and AI platform emphasis.
- Azure provides the primary teaching reference; Alibaba Cloud, Tencent Cloud, and Volcengine expose equally important but different industrial and engineering trajectories.
- Engineering intent is translated through resource model, responsibility, identity, network, lifecycle, evidence, economics, and non-equivalence.
- Interface, functional, operational, governance, economic, and organizational equivalence are separate claims.
- Purposeful multicloud architecture is often asymmetric and minimizes unnecessary cross-cloud runtime boundaries.
- A one-page translation record is more useful than a product checklist because it preserves provider-specific contracts and assigns owners to gaps.

References and Official Documentation

Product names, documentation state, and cited provider descriptions were checked August 25, 2026. Cloud products, preview status, regional availability, quotas, prices, and responsibility boundaries change; verify current official documentation before implementation.

[1] National Institute of Standards and Technology. The NIST Definition of Cloud Computing, SP 800-145. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[2] Microsoft. Shared responsibility in the cloud. https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility

[3] Microsoft. Shared responsibility for reliability. https://learn.microsoft.com/en-us/azure/reliability/concept-shared-responsibility

[4] Microsoft. Shared responsibility for AI workloads. https://learn.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility-ai

[5] Microsoft. Cloud Adoption Framework overview. https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/overview

[6] Microsoft. Unified operations for hybrid and multicloud. https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/scenarios/hybrid/strategy

[7] Microsoft. Azure Resource Manager overview. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/overview

[8] Microsoft. 2025 Annual Report. https://www.microsoft.com/investor/reports/ar25/index.html

[9] Alibaba Cloud. ECS shared responsibility model. https://www.alibabacloud.com/help/en/ecs/user-guide/ecs-shared-responsibility-model

[10] Alibaba Cloud. ACK shared responsibility model. https://www.alibabacloud.com/help/en/ack/ack-managed-and-ack-dedicated/security-and-compliance/shared-responsibility-model

[11] Alibaba Cloud. Resource Management overview. https://www.alibabacloud.com/help/en/resource-management/

[12] Alibaba Group. Fiscal Year 2024 Annual Report. https://static.alibabagroup.com/reports/fy2024/ar/ebook/EN/index.html

[13] Tencent Cloud. Cloud Access Management overview and resource description. https://www.tencentcloud.com/document/product/598/10583 and https://cloud.tencent.com/document/product/598/10606

[14] Tencent Cloud. Tag. https://cloud.tencent.com/document/product/651

[15] Tencent Cloud. Data Processing and Security Agreement. https://www.tencentcloud.com/document/product/301/17347

[16] Tencent Cloud. TRTC security and responsibility sharing. https://www.tencentcloud.com/document/product/647/53568

[17] Tencent Cloud. Cloud Dedicated Cloud Center shared responsibility. https://www.tencentcloud.com/document/product/1226/61856

[18] Tencent. 2025 interim and results materials. https://static.www.tencent.com/uploads/2025/08/13/56af6c5c98acdc7bd757a7bd208d8189.pdf

[19] Volcengine. Identity and Access Management. https://www.volcengine.com/docs/6257/64959

[20] Volcengine. Resource Center. https://www.volcengine.com/docs/6649/1109337

[21] Volcengine. Projects. https://www.volcengine.com/docs/6649/94337

[22] Volcengine. Tags. https://www.volcengine.com/docs/6649/189377

[23] Volcengine. Cloud service shared responsibility terms. https://www.volcengine.com/docs/6256/79758

[24] Volcengine. VKE shared responsibility. https://www.volcengine.com/docs/6460/2179469

[25] Volcengine. AgentKit shared responsibility. https://docs.volcengine.com/docs/86681/2605800?lang=zh

[26] ByteDance. Company and product portfolio. https://www.bytedance.com/en/

[27] Volcengine. Cloud and AI platform overview. https://www.volcengine.com/

Foundational Cloud Resource Control Planes

Before an application goes live, where will its resources live, who can change them, and how will its data recover? Resource organization and identity establish the boundaries. Compute, networking, and storage turn them into a dependency graph that can be tested.

04 Resource Organization, Tags, and Infrastructure as Code

05 Identity and Least Privilege

06 Compute, Elasticity, and Reliability

07 Networking, Private Connectivity, and Exposure Control

08 Storage, Data Lakes, and Event-Driven Data Paths

Part II / Chapter 4

Resource Organization, Tags, and Infrastructure as Code

How Scope, Metadata, and Declarative State Make Cloud Estates Attributable, Reproducible, Reviewable, and Disposable

Cloud resources do not become manageable merely because they appear in a portal. They become governable when they live in explicit scopes, carry controlled metadata, can be queried as an inventory, and are produced from reviewable desired state with a verified retirement path.

In This Chapter

- Why resource organization is a control-plane contract rather than an exercise in portal tidiness
- How organizations, accounts, subscriptions, resource groups, projects, and individual resources create different policy, billing, and lifecycle boundaries
- Why provider resource IDs, human-readable names, and tags must not be treated as interchangeable identifiers
- How to design a tag vocabulary for ownership, environment, cost allocation, data classification, criticality, and lifecycle
- Why an inventory must be queryable across scopes and linked to activity, compliance, and cost evidence
- How Infrastructure as Code turns cloud configuration into versioned desired state, planned change, and deterministic retirement
- When native declarations such as Bicep or ROS templates differ from provider-based workflows such as Terraform
- How state, drift, import, refactoring, modules, provider versions, and secrets affect IaC safety
- How to create an evidence-producing change loop from validate and plan through apply, verify, and destroy

- How two independent Azure labs establish resource attribution and a complete IaC lifecycle, with concise mappings to Alibaba Cloud, Tencent Cloud, and Volcengine
- How CloudDesk FinOps Agent can later reuse organization, tag-audit, deployment, and diff capabilities without being implemented in this chapter

A cloud estate usually becomes confusing before it becomes large. A few experiments are created under personal names. A shared database is placed in a resource group that belongs to no application. The production API has an owner tag, but its storage and network resources do not. A manually created resource survives after the project ends because nobody knows whether it is safe to delete. A later migration introduces Terraform, yet the existing resources remain outside its state. The immediate symptom is clutter; the architectural problem is that ownership, lifecycle, evidence, and cost were never made explicit.

Resource organization is therefore not a cosmetic prelude to “real” architecture. It is the first discipline of a programmable cloud control plane. Scopes determine where identities receive authority, where policy is inherited, where quotas and budgets are evaluated, where changes can be isolated, and where deletion has a bounded blast radius. Names help people navigate. Tags attach mutable business meaning. Resource IDs provide authoritative addresses. Inventory services make the resulting graph queryable. Infrastructure as Code then places desired state, dependency relationships, and lifecycle operations into a reviewable engineering workflow.

The course material behind this book described the transition from ClickOps to IaC in practical terms: portal creation is a one-time configuration action, whereas IaC is versionable, auditable, reproducible, and disposable. This chapter develops that observation into a resource-level contract. It does not teach every Bicep or Terraform language feature. Instead, it explains how organization and declaration work together so that a cloud estate can answer four questions at any time: What exists? Who owns it? What state should it be in? How can it be changed or removed safely?

Azure supplies the complete implementation path through resource groups, tags, Azure Resource Graph, Cost Management, Bicep and Resource Manager, deployment history, What-If, deployment stacks, and Terraform. Alibaba Cloud, Tencent Cloud, and Volcengine are compared by engineering intent. Their organization constructs, tag systems, inventory services, orchestration products, and Terraform providers differ in product boundary, but they solve the same recurring problem: turn a growing set of provider resources into an attributable and controlled system.

The chapter thesis

Resource organization and Infrastructure as Code are the first control-plane disciplines. Organization gives every resource a scope, owner, purpose, and cost context. IaC gives that resource a reviewable desired state, change history, dependency graph, and retirement path.

Resource organization is a control-plane graph, not folder hygiene

Authority hierarchy	Organization (policy, identity, billing) → account or subscription (boundary, quota, invoice) → resource group or project (ownership, lifecycle, delegation) → resource (ID, type, region, state).
Cross-cutting views	Tags describe business metadata. Inventory queries the resource graph. Policy checks state and remediation. Cost views allocate spend.

Scope provides authority and blast-radius boundaries; metadata and inventory provide cross-cutting views.

Figure 4-1 Resource organization combines hierarchical authority with cross-cutting metadata, inventory, policy, and cost views.

4.1 Organization Is a Control-Plane Contract, Not Folder Hygiene

A folder tree answers where an item is displayed. A control-plane scope answers which identities can act, which policies apply, which costs are aggregated, which quotas constrain the workload, which evidence is collected, and which resources can be changed together. These questions may coincide in a portal hierarchy, but they are not the same question. An attractive folder layout can still hide weak ownership and unsafe lifecycle boundaries.

Resource organization begins by identifying management units. The organization or tenant level coordinates multiple accounts. An account or subscription is usually a strong boundary for billing, quotas, service enablement, and administrative delegation. A resource group or project is a smaller operational scope that can collect resources for a workload,

environment, team, or lifecycle. The resource remains the provider-addressable object on which data-plane and control-plane operations ultimately act. Tags and queries then create views that cut across this hierarchy.

Good organization makes common operations local. A platform team should be able to grant a workload team access without granting unrelated access. A policy should target the intended environments without enumerating thousands of resource IDs. A cost view should attribute most direct spend without a spreadsheet maintained by hand. A deployment should create and remove a coherent set of resources. An incident responder should know which owner and service are affected. When these operations require ad hoc interpretation, the organization model is too weak.

Question	Control-plane answer	Failure when absent
What is the management boundary?	Organization, account/subscription, resource group/project, resource ID	Permissions and policies apply too broadly or unpredictably
Who owns the resource?	Accountability tags, workload catalog, assigned group or project	Incidents and costs have no responsible team
Why does it exist?	Purpose, workload, environment, data class, and lifecycle metadata	Orphaned resources appear legitimate
What state should it have?	Versioned IaC, parameters, policy, and provider versions	Portal state becomes the only specification
How is change approved?	Plan/What-If, policy checks, review, deployment identity, activity evidence	Configuration changes cannot be reconstructed
How is it retired?	Destroy plan, dependency review, retention rule, cleanup verification	Billable and security-sensitive remnants survive

Design rule

Choose scopes so that authority, lifecycle, and cost are understandable. Do not use a resource group merely because it exists, and do not use tags to imitate a boundary that should have been implemented with an account, subscription, project, policy, or dedicated resource.

4.2 The Scope Hierarchy: Organization, Account, Resource Group, and Resource

Every major cloud exposes more than one level of resource organization, but the names and semantics differ. Azure uses management groups, subscriptions, resource groups, and resources. Alibaba Cloud combines Resource Directory, accounts, resource groups, and tags. Tencent Cloud Organization coordinates member accounts and departments, while project and tag capabilities organize resources inside accounts. Volcengine provides projects, tags, a resource center, and multi-account resource-management services. Similar words do not guarantee equivalent lifecycle or authorization behavior.

An organization-level hierarchy should express durable corporate structure and control delegation. It is a poor place for short-lived application deployments. An account or subscription is a stronger scale unit: it can separate production from experimentation, isolate billing or quotas, and limit the effect of a compromised administrator. A resource group or project is better suited to a workload or environment whose resources can be operated and retired together. Individual resources then retain service-specific lifecycle and data-plane behavior.

The right boundary depends on the consequence of failure. If a policy mistake must never reach production, separate production at the account or subscription level. If two resources must be deployed, observed, and deleted together, place them in the same deployment boundary unless a shared-service architecture requires otherwise. If one database is shared by ten applications, do not pretend it belongs entirely to one application resource group; model it as a shared platform resource with explicit consumers and allocation rules.

Scope	Primary purpose	Typical decisions	Poor use
Organization / tenant hierarchy	Multi-account governance and delegated administration	Enterprise policy, identity center, account creation, consolidated billing	Per-deployment lifecycle
Account / subscription	Strong administrative, billing, quota, and blast-radius boundary	Production separation, legal entity, region strategy, platform ownership	One account for every small component
Resource group / project	Operational grouping and delegated workload management	Application/environment ownership, policy scope, cost scope, cleanup boundary	A single miscellaneous group for the whole company
Resource	Provider-addressable managed object	Configuration, data-plane access, SLO, retention, replacement	Using the name as the only durable identity

Scope	Primary purpose	Typical decisions	Poor use
Tag / label view	Cross-cutting classification and allocation	Owner, workload, environment, cost center, data class, lifecycle	Treating metadata as a hard isolation boundary

Grouping pattern	When it works	Trade-off
One resource group per workload and environment	The resources share owners, release cadence, evidence, and retirement date	Shared services need separate groups and explicit relationships
One subscription/account per environment class	Production requires stronger policy, quota, identity, and billing separation	More cross-scope networking, identity, and operational coordination
Shared platform resource group	Network hubs, registries, observability, or data services serve many workloads	Cost allocation and change governance must identify consumers
Ephemeral lab group	All resources are intentionally temporary and must be deleted together	Useful only if cleanup is verified and no durable data is placed inside
One resource per group	Rarely justified for an exceptional policy or deletion boundary	Creates administrative noise and weakens the meaning of grouping

4.3 Names, Resource IDs, and Tags Serve Different Semantics

Cloud engineers often overload names. A resource name is expected to identify the provider object, describe the workload, encode the environment, reveal the owner, communicate the region, and support cost allocation. It cannot do all of these reliably. Names are constrained by service-specific character sets, lengths, global uniqueness rules, and immutability rules. Some names are visible in data-plane endpoints. Others can be changed without changing the provider identity. A naming convention should support navigation, but it must not become the only source of business meaning.

The provider resource ID is the authoritative address. It connects API operations, activity logs, policy state, configuration history, cost records, and deployment outputs. A resource can be renamed in a catalog while retaining its provider ID, or replaced with a new object that reuses a human-readable name. Evidence systems should therefore preserve IDs, account/subscription scope, resource type, region, and provider, not only display names.

Tags are mutable key-value metadata. They are valuable because they can classify resources across type and region, drive inventory filters, support cost allocation, and feed automation. Their flexibility is also their weakness. Uncontrolled spelling, case, abbreviations, personal names, and free-form text create a taxonomy that cannot be queried consistently. Tags are stored as plain text and may appear in cost exports, deployment history, commands, and logs. They must never contain credentials, personal data, or confidential business text.

Identity, names, and tags answer different questions

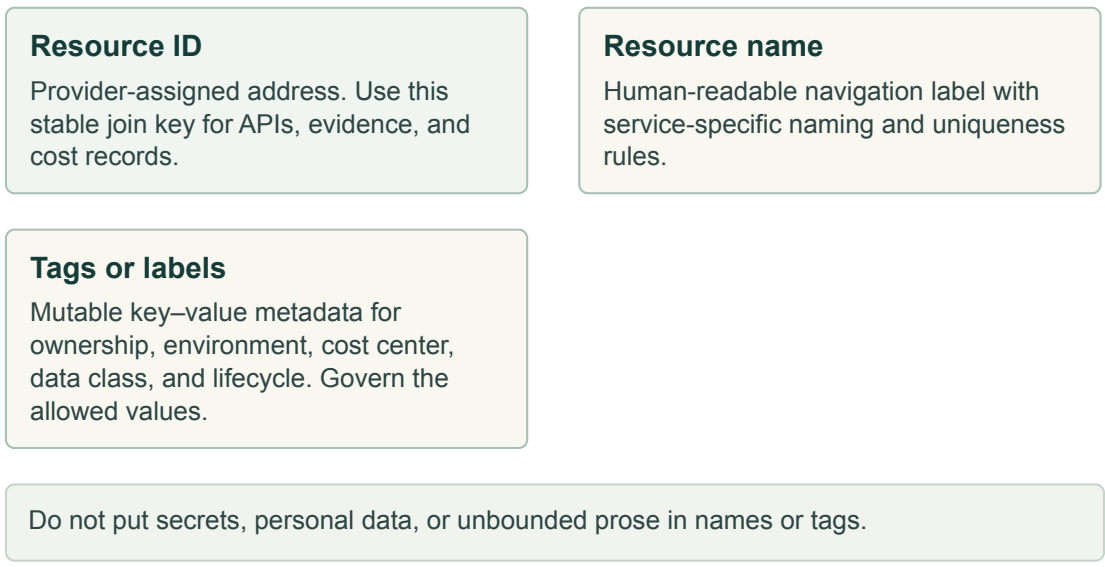


Figure 4-2 Resource ID, name, and tags have distinct roles and should be joined rather than substituted for one another.

Mechanism	Best use	Stability	Security note
Resource ID	Authoritative API, evidence, and billing join key	Durable for the lifetime of the resource	May reveal account, scope, type, and path; still not a secret
Resource name	Human navigation and operational recognition	Varies by service; rename support is inconsistent	Do not encode personal or confidential information
Tag key/value	Mutable business classification and automation inputs	Expected to change as ownership and lifecycle change	Plain-text metadata; do not store secrets

Mechanism	Best use	Stability	Security note
Catalog entry / CMDB ID	Enterprise workload, service, or product identity across providers	Should outlive individual provider resources	Must preserve provider IDs and evidence links
IaC address	Logical address inside a module and state	Stable only if refactoring is managed	Changing an address can cause replacement unless moved/import semantics are used

Naming principle

Put only the attributes needed for human recognition into a name. Put mutable business classification into controlled tags. Put authoritative identity into provider resource IDs and a service catalog. A name should not be parsed as a database record.

4.4 Designing a Tagging and Cost-Allocation Strategy

A tag strategy is a small data model. Its keys, value domains, owners, enforcement modes, and consumers must be defined before tags are applied. The strategy should start from decisions that teams need to make: who responds to an incident, which product receives the cost, whether a resource is production, which data handling rules apply, when the resource should be reviewed or deleted, and whether automation may modify it. A tag with no consumer is merely decorative metadata.

The most useful tags fall into a few categories. Accountability tags identify the owner and cost center. Workload tags connect a resource to an application or product. Operational tags record environment, criticality, lifecycle, and service level. Governance tags classify data and compliance context. Automation tags indicate whether IaC, a platform team, or another controller manages the resource. Each key should have a controlled vocabulary and a documented system of record.

Cost allocation depends on more than attaching tags. Some charges do not map directly to a resource, some services do not emit all resource tags into usage records, and shared platforms require an allocation rule. Tags generally apply to usage after they are added; they do not rewrite historical billing records. Azure Cost Management can group by resource group or tag and can apply supported tag inheritance to usage records. Similar allocation-tag capabilities exist in the Chinese cloud platforms. A mature design therefore combines account hierarchy, resource scope, tags, billing exports, and derived allocation metadata.

FinOps guidance treats allocation as the process of assigning cost and usage to responsible owners, products, or cost centers by using accounts, tags, labels, and other metadata. That definition is important: tagging is an input to allocation, not allocation itself. A tag compliance rate should be measured by cost coverage as well as resource count. One untagged shared cluster may matter more than hundreds of tagged test objects.

Category	Recommended key	Example values	Primary consumer
Accountability	owner	team-platform; team-search; course-lab	Incident response, access review, orphan detection
Financial	costCenter	cc-4102; research-ai; training	Cost analysis, showback, chargeback
Workload	workload	catalog-api; finops-agent; course-rag	Service catalog, dependency mapping, cost/unit economics
Environment	environment	dev; test; staging; prod; lab	Policy, deployment workflow, alert routing
Purpose	purpose	web-api; knowledge-index; backup; experiment	Inventory explanation and optimization
Criticality	criticality	tier-0; tier-1; tier-2; noncritical	SLO, backup, recovery, change approval
Data governance	dataClass	public; internal; confidential; restricted	Security, retention, residency, review
Lifecycle	lifecycle	ephemeral; persistent; review-2026q4; retire	Cleanup, retention, technical debt
Automation	managedBy	bicep; terraform; ros; platform-team	Drift handling and change routing

Tag anti-pattern	Why it fails	Better rule
Owner is a personal email address	People leave and personal data spreads into exports and logs	Use a durable team or service ownership ID
Environment values include prod, production, PRD, and live	Queries and policy assignments miss resources	Use a controlled enumeration
The resource name is parsed to derive cost center	Renaming and service naming limits break allocation	Store cost center explicitly and validate it
All tags are optional	Coverage decays and orphan resources accumulate	Define a small required set and measure compliance

Tag anti-pattern	Why it fails	Better rule
Hundreds of speculative tags	High maintenance cost with no consumers	Create a tag only when a process reads it
Sensitive values in tags	Tags appear in many operational and billing surfaces	Use a secret store or protected catalog
Tags are treated as authorization by default	Metadata alone does not enforce access	Use IAM/ABAC only where explicitly supported and tested
Tags are applied once and never reviewed	Ownership, environment, and lifecycle become stale	Run periodic reconciliation against systems of record

```
# Example organization tag contract (conceptual)
required:
  owner:       service-catalog.team-id
  workload:    service-catalog.workload-id
  environment: [dev, test, staging, prod, lab]
  costCenter:  finance.cost-center-id
  lifecycle:   [ephemeral, persistent, review, retire]
optional:
  dataClass:   [public, internal, confidential, restricted]
  criticality: [tier-0, tier-1, tier-2, noncritical]
  managedBy:   [bicep, terraform, ros, platform-team]
prohibited:
  - passwords, tokens, connection strings
  - personal or regulated data
  - free-form incident notes
```

Allocation metric

Measure both resource coverage and cost coverage. A useful cost-weighted metric is: compliant direct cost divided by total directly attributable cost. Shared and unallocated charges require a separate allocation policy rather than a fictitious tag.

4.5 Inventory Must Be Queryable

A resource list in a portal is not yet an inventory. An inventory is a queryable control-plane view that can answer questions across resource types, regions, accounts, and tags. It should return provider IDs, types, locations, owners, lifecycle metadata, and management state. It should also reveal gaps: resources without owners, production objects in unauthorized regions, objects that claim to be Terraform-managed but are absent from the expected state, and resources whose deletion date has passed.

Azure Resource Graph extends Resource Manager with cross-scope queries over resource metadata. Comparable resource-center, configuration, and tag-search services exist in Alibaba Cloud, Tencent Cloud, and Volcengine. These systems are optimized for exploration and governance. They are not substitutes for the activity log, configuration timeline, telemetry, or billing ledger. Inventory answers what is visible now; activity evidence answers who changed it; cost data answers what was charged; observability answers how it behaved.

A useful inventory query preserves resource IDs and projects normalized metadata. It does not merely count objects. It should be exportable as an evidence packet and repeatable in automation. The same query can support a book lab, a governance review, or a read-only Agent tool, provided the scope and permissions are explicit.

```
Resources
| where resourceGroup =~ 'csaip-ch04-lab-rg'
| project
    id,
    name,
    type,
    location,
    owner = tostring(tags.owner),
    environment = tostring(tags.environment),
    purpose = tostring(tags.purpose),
    costCenter = tostring(tags.costCenter),
    lifecycle = tostring(tags.lifecycle),
    managedBy = tostring(tags.managedBy)
| order by type asc
```

Evidence source	Question it answers	Do not infer
Resource inventory / graph	What objects and metadata are currently visible at this scope?	Who made every change or whether the service is healthy
Activity / audit log	Which principal performed a control-plane operation and when?	The full data-plane effect or business reason
Configuration timeline / IaC state	How did properties and declared state change?	That the running service meets an SLO
Metrics, logs, and traces	How did the workload behave during an interval?	Authoritative billing or ownership
Cost and usage data	Which charges were recorded and how can they be allocated?	Real-time spend or complete tag coverage

Evidence source	Question it answers	Do not infer
Service catalog / CMDB	Which product, owner, dependency, and business context should apply?	That provider configuration actually conforms

4.6 Infrastructure as Code Is a Managed Change Contract

Infrastructure as Code is frequently reduced to “writing YAML or HCL instead of clicking.” That definition misses the engineering value. IaC is a managed change contract among declared intent, provider schemas, an execution identity, current infrastructure state, a dependency graph, and evidence of the proposed and actual change. The source file matters because it can be versioned and reviewed, but the file alone is not the system.

Declarative IaC describes the desired result and lets an engine derive an operation graph. An imperative script describes a sequence of commands. Both can be automated, and both can fail. Declarative tools are valuable because they can compare desired state with known or observed state, construct dependencies from references, preview change, and converge repeated executions toward the same result. Idempotence is an objective, not a guarantee: provider defaults, asynchronous APIs, non-deterministic values, external controllers, and manual changes can still produce surprises.

IaC creates several durable engineering artifacts. The configuration records intended structure. The plan or What-If records a predicted change. The apply or deployment record identifies the execution and outcome. Outputs expose resource IDs and endpoints to downstream systems. State or deployment metadata connects logical addresses to provider objects. Verification records test the actual result. A destroy plan and cleanup proof close the lifecycle. Together these artifacts are much stronger than a screenshot of a successful portal blade.

Infrastructure as Code is a managed feedback loop

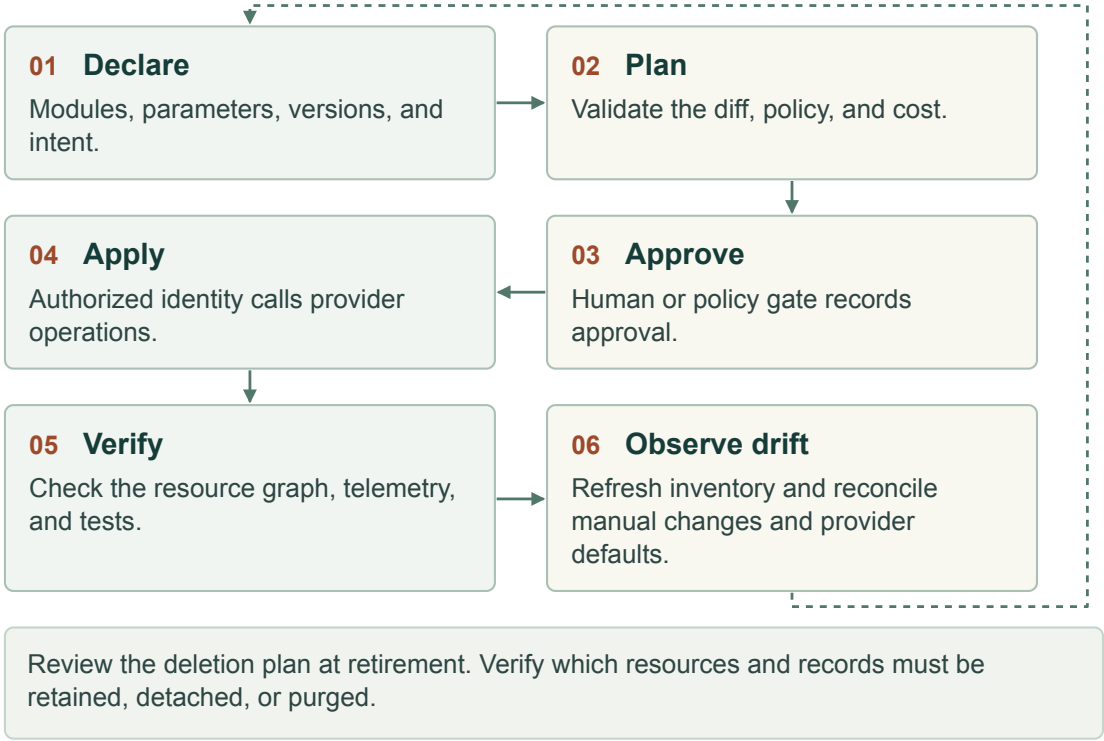


Figure 4-3 IaC becomes a control-plane feedback loop when declaration, authorization, evidence, verification, drift handling, and retirement are connected.

Property	Engineering meaning	Evidence
Versionable	Desired state changes can be reviewed and associated with a commit	Repository history and release record
Reproducible	The same inputs can create equivalent environments within documented provider constraints	Parameters, provider lock, deployment output, verification
Reviewable	The proposed resource graph and destructive operations are visible before execution	Plan or What-If plus policy results
Auditable	The executing identity, time, scope, and result are recorded	Deployment history, activity log, pipeline record
Composable	Modules expose typed inputs and outputs and preserve dependency relationships	Module interface and resource graph

Property	Engineering meaning	Evidence
Drift-aware	Differences between declared, known, and actual state can be detected and classified	Refresh, inventory query, drift report
Disposable	Temporary infrastructure has an explicit and verified destruction path	Destroy plan, apply record, zero-resource verification

4.7 Native and Portable IaC

Cloud-native declaration languages are closely aligned with one provider control plane. Azure Resource Manager templates and Bicep expose Azure resource types and deployment scopes directly. Bicep compiles to ARM JSON and uses Resource Manager for deployment. Alibaba Cloud Resource Orchestration Service accepts ROS templates and also provides managed Terraform execution. Native systems often expose new platform features early and produce provider-native deployment history, but they are intentionally not portable to another cloud.

Terraform provides a shared configuration language, plan/apply workflow, module system, and state model across providers. The provider plugin translates Terraform resource operations into each cloud API. This gives teams a portable engineering workflow, not portable infrastructure semantics. An Azure resource group, an Alibaba Cloud resource group, a Tencent Cloud project, and a Volcengine project are not made equivalent by being declared in HCL. The module must still encode provider-specific identity, network, lifecycle, and evidence contracts.

The choice is not ideological. Use native IaC when the architecture is provider-specific, when first-class support for platform features matters, or when the provider deployment model should be the primary evidence source. Use Terraform when a team needs a consistent workflow across many providers or SaaS systems, when mature modules already exist, or when a separate state and planning model fits the operating process. Many organizations use both: native declarations for landing zones or strongly provider-specific services, and Terraform for cross-provider platform composition.

Azure deployment stacks add lifecycle semantics beyond a conventional deployment record. A stack manages a group of resources as a unit and can delete or detach resources removed from the declaration according to its unmanage behavior. This matters because incremental template deployment does not delete every resource omitted from a later template, and complete deployment mode is being phased out in favor of deployment stacks for deletion scenarios. Deletion semantics must be selected deliberately rather than assumed from the word “declarative.”

Native and portable IaC expose different trade-offs

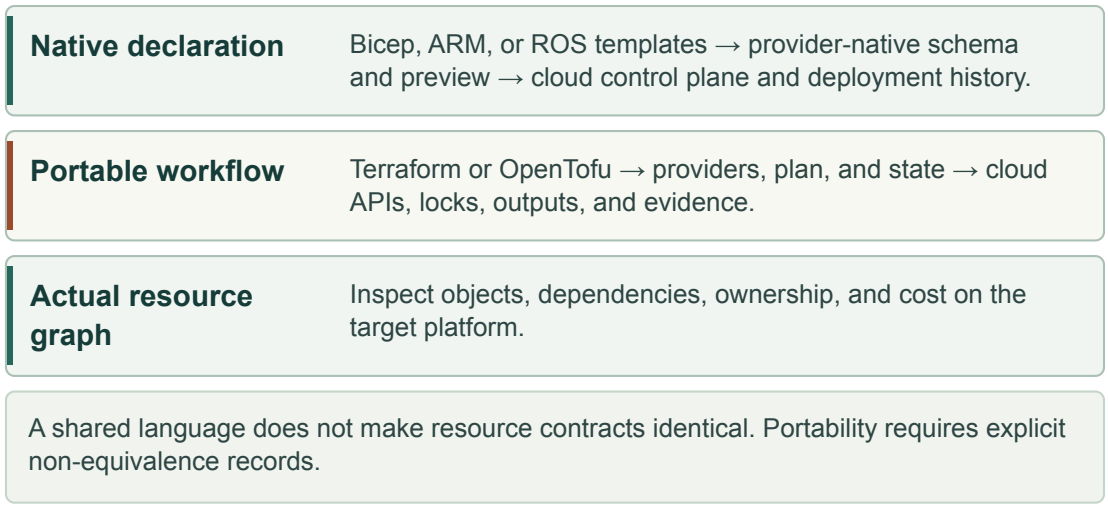


Figure 4-4 Native IaC and provider-based IaC share control-plane goals but place state, schema, and lifecycle responsibility in different locations.

Dimension	Bicep / native template	Terraform / provider workflow
Primary schema	Provider-native resource types and API versions	Terraform provider resource schema
State model	Actual resource state remains in the cloud control plane; deployments record operations	A state file maps logical addresses to provider objects and metadata
Change preview	ARM What-If or native stack preview	terraform plan, optionally stored as an apply artifact
Feature timing	Often exposes provider features early	Depends on provider release and schema implementation
Cross-provider workflow	Not a goal	Common language and workflow, provider-specific resources
Deletion model	Incremental deployment, deployment stacks, or explicit deletion	Configuration/state comparison and destroy plan
Operational risk	Template/API-version drift and provider deployment semantics	State security, locking, provider upgrades, address refactoring
Best fit	Deep provider integration and native control-plane governance	Consistent orchestration across multiple providers and services

4.8 State, Drift, Import, and Refactoring

Terraform state is not a backup of the cloud. It is the mapping Terraform uses between configuration addresses and remote objects, together with metadata and cached attributes needed to plan efficiently. Losing or exposing state can break lifecycle control or reveal sensitive values. Team use therefore requires a protected remote backend, access control, encryption, versioning, and locking. The state must not be committed to a normal source repository.

Bicep does not require the author to maintain an equivalent state file. Resource Manager observes current Azure resources and deployment metadata when it calculates a What-If or performs a deployment. This removes one class of state operations but not drift. A portal edit, policy remediation, provider default, or external controller can still change actual state. The difference is where the mapping and comparison responsibility resides.

Drift is any material difference among declared state, tool-known state, and actual provider state. Not all drift is malicious or accidental. Incident responders may make emergency changes. Autoscaling and managed services legitimately change operational state. A lifecycle policy may be managed by another controller. The response should classify drift: reconcile it from code, import or record it, delegate the property to another controller, or explicitly ignore it. Blindly forcing every difference back to configuration can cause an outage.

Existing resources can be brought under IaC, but import is not automatic architecture. The resource must be declared, imported or referenced, compared, and normalized without destructive replacement. Logical addresses may also change as modules are refactored. Terraform moved blocks and provider-native import or existing-resource references help preserve identity. A plan that proposes replacement after a refactor is a warning that address semantics were changed without a migration record.

Condition	Preferred response	Evidence to retain
Unauthorized portal edit	Revert through IaC and restrict direct write permissions	Activity event, plan, approval, apply, verification
Emergency incident change	Stabilize first, then reconcile code/state through reviewed follow-up	Incident record, resource diff, final configuration
Managed-service or autoscaler mutation	Exclude operational fields from IaC ownership where appropriate	Controller contract and ignored-property rationale
Resource created outside IaC	Decide to import, reference, replace, or retire	Ownership decision and non-destructive plan

Condition	Preferred response	Evidence to retain
Module address changed	Use explicit move/refactor semantics	Moved mapping and plan showing no replacement
Resource removed from declaration	Review retain, detach, delete, and purge requirements	Destroy/unmanage plan and data-retention approval

State security

Treat an IaC state store as sensitive operational data. It can contain identifiers, topology, computed attributes, and values returned by providers. “The secret is not in the .tf file” does not prove that the secret is absent from state.

4.9 Modules, Parameters, Environments, and Secrets

Modules should follow ownership and lifecycle boundaries, not vendor catalog categories. A network module may be reusable, but a complete workload module should connect network, identity, storage, diagnostics, policy, and cost metadata required by that workload. A “one module for every resource type” design merely hides the provider schema behind another thin layer. A giant module for the entire enterprise creates the opposite problem: every change has an enormous plan and blast radius.

Environment differences should be expressed through explicit parameters and stronger scope boundaries where needed. Production and development should not differ only because one variable says environment=prod. Production may require a separate subscription or account, different identities, private endpoints, stronger policy, distinct state, protected approvals, and a longer retention path. Parameters configure a contract; they do not create isolation by themselves.

Provider and module versions are part of desired state. Pin compatible ranges, commit dependency locks where the tool supports them, and review upgrades as infrastructure changes. Provider upgrades can alter defaults, validation, diff behavior, and resource replacement. A plan after a version upgrade is therefore not clerical output; it is the migration assessment.

Secrets require a separate path. Source repositories, variable files, plan artifacts, state files, deployment history, tags, and command logs can all leak values. Prefer workload identity and references to managed secret stores. When a provider requires a secret as an input, classify where the value appears, restrict the execution environment, mark outputs as sensitive, and verify the state/backend protections. Redaction in console output is not a complete secret-management strategy.

```
cloud-infrastructure/
├── README.md
├── modules/
│   ├── workload-foundation/
│   ├── private-storage/
│   └── observability-baseline/
├── environments/
│   ├── dev/
│   │   ├── main.tf
│   │   └── dev.tfvars.example
│   └── prod/
│       ├── main.tf
│       └── prod.tfvars.example
├── policies/
├── tests/
├── evidence/
└── .gitignore
```

Repository artifact	Purpose	Publishing rule
Configuration and modules	Desired state and reusable contracts	Version, review, lint, test
Example variable files	Document required inputs without credentials	Commit examples; exclude real secrets and state
Provider lock file	Make provider selection reproducible	Commit and review upgrades
Plan artifact	Record the proposed operation graph	Store for the approved run; treat as potentially sensitive
State backend	Map configuration addresses to managed objects	Encrypt, lock, restrict, version; never publish
Evidence packet	Connect plan, apply, inventory, activity, and verification	Retain according to change and audit policy
Cleanup script or destroy plan	Close the lifecycle of ephemeral environments	Test as part of acceptance, not after the book is published

4.10 The IaC Change Loop: Plan, Approve, Apply, Verify, Retire

A mature IaC workflow separates prediction from authority. Formatting, static validation, and module tests establish that the declaration is internally coherent. A plan or What-If predicts provider operations. Policy checks determine whether the proposed state is allowed. Cost analysis identifies material economic effects. Human or automated approval

authorizes a specific plan. The apply identity performs the change with least privilege. Verification then queries actual resources and tests expected behavior. None of these stages should be silently collapsed into "run the pipeline."

A plan is time-bound evidence, not a promise. Another actor can change the resource after the plan is generated. Provider capacity can change. A policy can be assigned. A secret can expire. For high-risk changes, apply the saved plan or re-plan immediately before execution and reject material differences. The evidence packet should identify configuration commit, provider versions, plan digest, approver, execution identity, deployment ID, resource IDs, and verification result.

Retirement belongs in the same workflow. Preview destructive operations, classify retained data and shared dependencies, obtain the required approval, execute deletion or detach semantics, and verify that billable or exposed resources no longer exist. "The destroy command returned success" is not enough if a public IP, disk, snapshot, DNS record, key, deployment identity, or retained storage object remains. Ephemeral infrastructure is only ephemeral when its cleanup path is tested.

Stage	Minimum control	Evidence
Format and validate	Syntax, schema, module input, provider compatibility	Validation output
Plan / What-If	Explicit scope and predicted create/update/replace/delete operations	Saved plan or rendered change report
Policy and security	Allowed regions, tags, exposure, identity, encryption, high-risk replacements	Policy results and exceptions
Cost review	Estimate material recurring, usage, egress, and commitment effects	Cost delta or documented limitation
Approval	Named approver for a specific plan digest and environment	Approval record
Apply	Least-privilege non-human identity and serialized state access	Deployment/run ID and activity evidence
Verify	Query actual resource graph and run functional/security checks	Inventory and test output
Retire	Destroy/unmanage preview, data-retention decision, residual-resource check	Cleanup and zero-remainder evidence

4.11 Translating Organization and IaC Across Clouds

Multicloud comparison should preserve engineering intent rather than force identical hierarchy. Azure resource groups, Alibaba Cloud resource groups, Tencent Cloud projects, and Volcengine projects all support grouping, but their relationships to accounts, policy, cost, and service coverage differ. The same caution applies to tag inheritance, resource inventory, and native orchestration. The correct translation records both the nearest implementation and the non-equivalence that affects operations.

Terraform can provide a common repository and plan/state workflow across these providers, but each provider still requires its own resource types, authentication, region identifiers, lifecycle behavior, and feature coverage. Native orchestration remains relevant: Azure Resource Manager and Bicep, Alibaba Cloud ROS, and Volcengine Infra Manager expose provider-specific control-plane capabilities. Tencent Cloud commonly uses its Terraform provider and cloud service APIs. The architecture should select a primary source of desired state for each resource, not let native templates, Terraform, and portal edits compete without ownership rules.

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Multi-account hierarchy	Management groups and subscriptions	Resource Directory and member accounts	Tencent Cloud Organization departments and member accounts	Cloud management / multi-account resource services
Workload grouping	Resource group	Resource group	Project and product-specific grouping constructs	Project
Cross-cutting metadata	Resource tags and tag policy through governance tooling	Tags and tag policies, including supported inheritance/remediation	Tags, tag policy, and cost-allocation tags	Tags and project metadata
Inventory	Azure Resource Graph	Resource Center / Cloud Config / tag search	Resource Center / Config / tag search	Resource Center / tag search
Native IaC	ARM templates, Bicep, deployment stacks	ROS templates and managed Terraform in ROS	Service APIs and provider tooling; Terraform is a common path	Infra Manager and service-specific templates where supported
Portable IaC	HashiCorp AzureRM provider	Alibaba Cloud provider	Tencent Cloud provider	Volcengine provider

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Cost attribution	Subscription/resource group scope, tags, Cost Management inheritance/allocation	Account/resource group/tags and cost center	Projects, allocation tags, billing and FinOps tools	Projects, tags, billing and cost-management services
Non-equivalence to record	Resource-group lifecycle and deployment-stack behavior	Resource Directory, account, group, and tag-policy mode distinctions	Project coverage and service-specific resource grouping	Project coverage, tag support, and Infra Manager/provider maturity

Multicloud rule

Standardize the metadata vocabulary and change evidence before standardizing templates. A shared owner, workload, environment, cost-center, and lifecycle contract is often more portable than a forced abstraction over every provider resource type.

4.12 Lab 1: Resource Organization and Cost Attribution

This lab creates a dedicated Azure resource group and a low-cost storage account, applies the same controlled metadata to both, queries the resources through Azure Resource Graph, prepares a cost-attribution view, and proves cleanup. It is independent of every other lab in the book. The objective is not to store data; it is to demonstrate that scope, ownership, inventory, and cost context can be verified from the control plane.

Lab 1 evidence: organization and attribution must be queryable

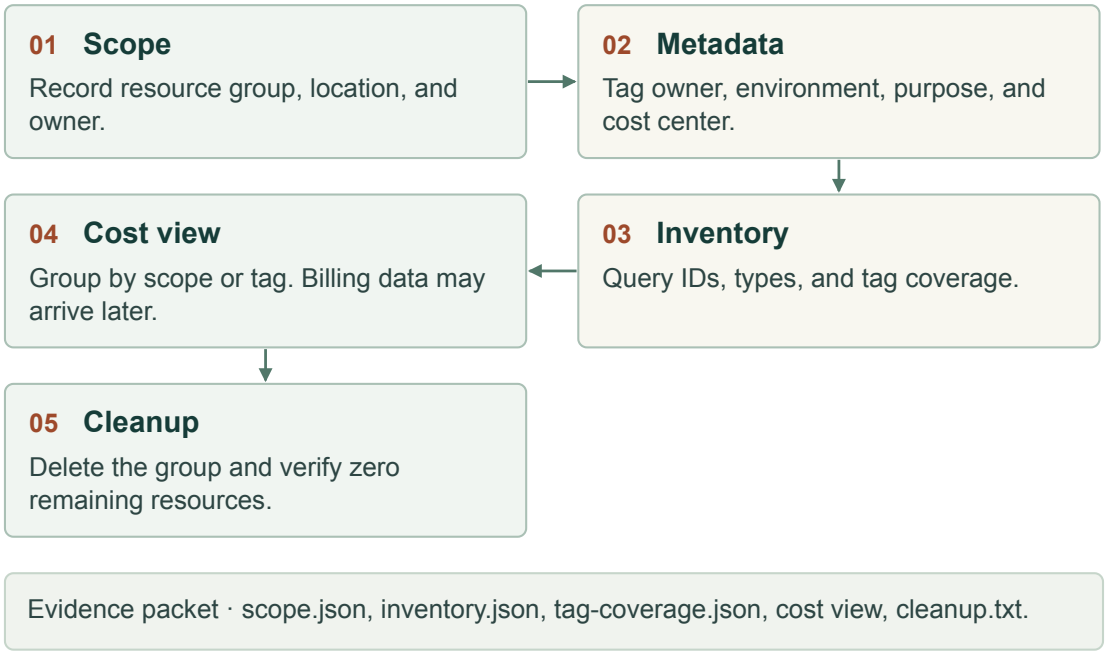


Figure 4-5 Lab 1 is accepted only when organization and attribution are represented by a repeatable evidence packet and cleanup proof.

Item	Requirement
Permissions	Permission to create and delete a resource group and storage account in one Azure subscription; read access to Resource Graph and Cost Management for the chosen scope
Tools	Azure CLI; Resource Graph extension is installed automatically by current CLI versions when az graph is first used
Cost boundary	One Standard_LRS StorageV2 account with no uploaded data. Delete the resource group when the evidence packet is complete
Metadata contract	owner, workload, environment, purpose, costCenter, lifecycle, managedBy
Evidence directory	Create a local evidence/ directory. Do not place credentials, tokens, or connection strings in it
Timing caveat	Cost and usage data is not real time. Tag-based cost results can appear only after usage is emitted and processed; tags are not retroactive

```
# 1. Choose a subscription and lab variables
az login
az account set --subscription "<subscription-id-or-name>"

LOCATION="eastasia"          # choose an allowed region
SUFFIX="$(openssl rand -hex 3)"
RG="csaip-ch04-${SUFFIX}-rg"
STORAGE="csaip${SUFFIX}sa"
OWNER="team-cloud-course"

mkdir -p evidence

# 2. Create a dedicated scope with controlled metadata
az group create   --name "$RG"   --location "$LOCATION"   --tags owner="$OWNER"
workload="book-ch04"          environment="lab"          purpose="resource-
organization"          costCenter="training"          lifecycle="ephemeral"
managedBy="cli"   --output json > evidence/01-resource-group.json

# 3. Create one low-cost resource with the same metadata
az storage account create   --name "$STORAGE"   --resource-group "$RG"   --
location "$LOCATION"   --sku Standard_LRS   --kind StorageV2   --min-tls-version
TLS1_2   --allow-blob-public-access false   --tags owner="$OWNER"
workload="book-ch04"          environment="lab"          purpose="resource-
organization"          costCenter="training"          lifecycle="ephemeral"
managedBy="cli"   --output json > evidence/02-storage-account.json
```

```
# 4. Query the resource group itself
az graph query -q "
ResourceContainers
| where type =~ 'microsoft.resources/subscriptions/resourcegroups'
| where name =~ '$RG'
| project id, name, type, location, tags
" -o json > evidence/03-scope-query.json

# 5. Query all resources and normalize the required tags
az graph query -q "
Resources
| where resourceGroup =~ '$RG'
| project id, name, type, location,
    owner=tostring(tags.owner),
    workload=tostring(tags.workload),
    environment=tostring(tags.environment),
    purpose=tostring(tags.purpose),
    costCenter=tostring(tags.costCenter),
    lifecycle=tostring(tags.lifecycle),
    managedBy=tostring(tags.managedBy)
| order by type asc
```

```
" -o json > evidence/04-inventory.json

# 6. The following query must return an empty result set
az graph query -q "
Resources
| where resourceGroup =~ '$RG'
| where isempty(tostring(tags.owner))
    or isempty(tostring(tags.workload))
    or isempty(tostring(tags.environment))
    or isempty(tostring(tags.costCenter))
    or isempty(tostring(tags.lifecycle))
| project id, name, type, tags
" -o json > evidence/05-tag-gaps.json
```

Open Cost Management at the resource-group scope. Save a Cost Analysis view filtered to the lab resource group. After usage data appears, group by the costCenter or workload tag and export the table or capture the saved view. If no charge has appeared yet, record the time, scope, filters, and expected tag dimensions instead of claiming real-time attribution. Cost Management data is refreshed as providers emit usage, and newly applied tags do not rewrite historical records.

Review the evidence files. The authoritative acceptance condition is not that the portal displays a green check. It is that the scope query returns the resource group, the inventory returns the storage account and required metadata, the tag-gap query returns no resources, and the cost view is scoped or grouped using the same allocation contract.

```
# 7. Cleanup is part of the lab
az group delete --name "$RG" --yes
az group wait --name "$RG" --deleted

# 8. Prove that the scope no longer exists
az group exists --name "$RG" | tee evidence/06-group-exists-after-delete.txt

# Optional: prove that no resource remains visible under the group name
az graph query -q "
Resources
| where resourceGroup =~ '$RG'
| project id, name, type
" -o json > evidence/07-resources-after-delete.json
```

Acceptance evidence	Pass condition
01-resource-group.json	Resource group ID, region, and required tags are present

Acceptance evidence	Pass condition
04-inventory.json	Every lab resource has provider ID, type, location, and normalized ownership/cost metadata
05-tag-gaps.json	The result contains zero resources
Cost view or export	The resource-group scope and intended tag grouping are recorded; latency is disclosed
06-group-exists-after-delete.txt	The value is false
07-resources-after-delete.json	The result contains zero resources after control-plane propagation

Chinese-cloud translation

Repeat the intent rather than the commands: create an isolated group/project, apply the same controlled tags, query the provider resource center, open the cost scope or allocation-tag view, and delete the group/project resources. Record where the provider cannot group, inherit, or allocate tags in the same way.

4.13 Lab 2: IaC Lifecycle - Create, Verify, Change, and Destroy

This lab uses Terraform with the AzureRM provider because the state and plan lifecycle are visible and the same workflow can later be translated to other providers. Bicep remains the Azure-native reference in the chapter. The lab creates a resource group and storage account, saves a plan, applies it, makes one controlled in-place change, verifies the actual resource graph, previews destruction, destroys the stack, and proves that no managed resource remains.

The lab deliberately avoids passwords, data-plane keys, and uploaded objects. Even so, the Terraform state is operationally sensitive and must not be committed. Use a disposable directory. In a team environment, replace local state with an encrypted and locked remote backend.

Lab 2: create, change, and destroy a deployment

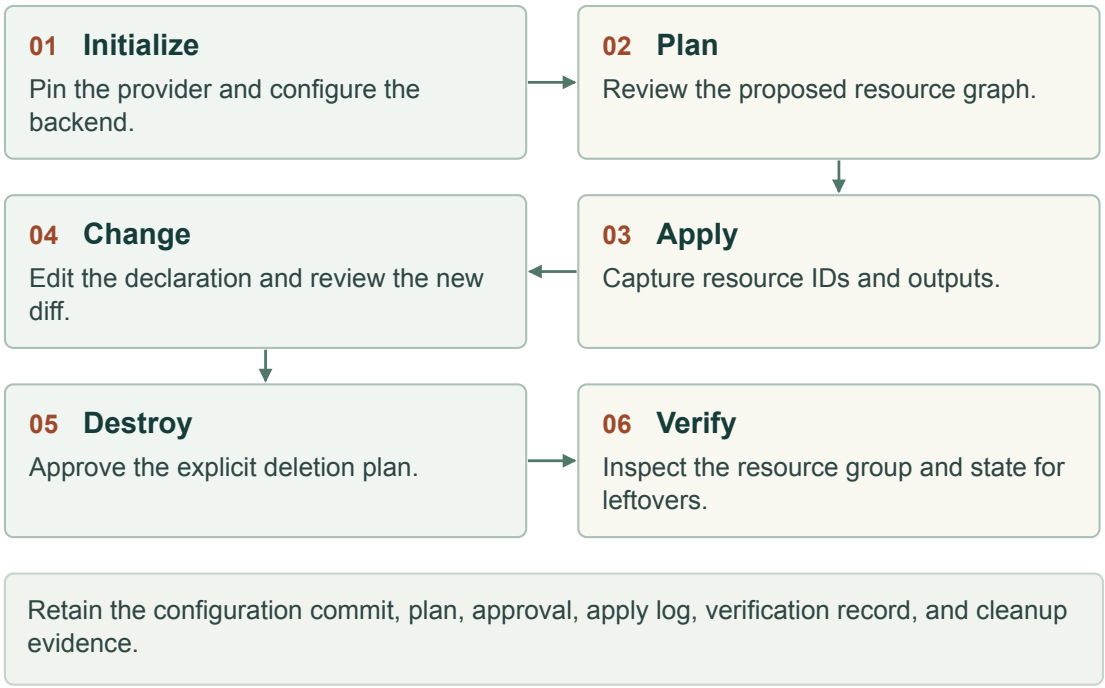


Figure 4-6 Lab 2 treats destruction and zero-remainder verification as acceptance criteria, not optional cleanup.

File	Responsibility
versions.tf	Terraform and provider constraints
main.tf	Provider configuration, tags, resource group, storage account, and outputs
variables.tf	Subscription, region, suffix, owner, cost center, and controlled access tier
terraform.tfvars	Local non-secret inputs; do not commit
.gitignore	Exclude state, plans, crash logs, and real variable files
evidence/	Store human-readable plans, outputs, inventory queries, and cleanup proof

```
# versions.tf
terraform {
  required_version = ">= 1.8.0"
```

```
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = ">= 5.0, < 6.0"
    }
  }
}

provider "azurerm" {
  features {}
  subscription_id = var.subscription_id
}
```

```
# variables.tf
variable "subscription_id" { type = string }
variable "location" {
  type    = string
  default = "eastasia"
}
variable "suffix" {
  type = string
  validation {
    condition     = can(regex("^[a-z0-9]{6}$", var.suffix))
    error_message = "suffix must contain exactly six lowercase letters or
digits."
  }
}
variable "owner"       { type = string }
variable "cost_center" { type = string }
variable "access_tier" {
  type    = string
  default = "Hot"
  validation {
    condition     = contains(["Hot", "Cool"], var.access_tier)
    error_message = "access_tier must be Hot or Cool."
  }
}
```

```
# main.tf
locals {
  common_tags = {
    owner       = var.owner
    workload    = "book-ch04-iac"
    environment = "lab"
    purpose     = "iac-lifecycle"
    costCenter  = var.cost_center
```

```
    lifecycle   = "ephemeral"
    managedBy   = "terraform"
  }
}

resource "azurerm_resource_group" "lab" {
  name     = "csaip-ch04-${var.suffix}-rg"
  location = var.location
  tags     = local.common_tags
}

resource "azurerm_storage_account" "lab" {
  name                             = "csaip${var.suffix}sa"
  resource_group_name              = azurerm_resource_group.lab.name
  location                         = azurerm_resource_group.lab.location
  account_tier                     = "Standard"
  account_replication_type         = "LRS"
  account_kind                     = "StorageV2"
  access_tier                      = var.access_tier
  min_tls_version                  = "TLS1_2"
  allow_nested_items_to_be_public = false
  tags                             = local.common_tags
}

output "resource_group_name" {
  value = azurerm_resource_group.lab.name
}
output "storage_account_id" {
  value = azurerm_storage_account.lab.id
}
```

```
# terraform.tfvars.example
subscription_id = "00000000-0000-0000-0000-000000000000"
location        = "eastasia"
suffix          = "a1b2c3"
owner           = "team-cloud-course"
cost_center     = "training"
access_tier     = "Hot"

# .gitignore
.terraform/
*.tfstate
*.tfstate.*
*.tfplan
crash.log
terraform.tfvars
evidence/*.json
```

```
# 1. Initialize, validate, and create a saved plan
mkdir -p evidence
cp terraform.tfvars.example terraform.tfvars
# Edit subscription_id and choose a unique six-character suffix.

az login
terraform init
terraform fmt -check -recursive
terraform validate
terraform plan -out=create.tfplan
terraform show -no-color create.tfplan   | tee evidence/01-create-plan.txt

# 2. Apply exactly the reviewed plan
terraform apply create.tfplan   | tee evidence/02-create-apply.txt
terraform output -json > evidence/03-outputs.json
RG="$(terraform output -raw resource_group_name)"

# 3. Verify actual control-plane state
az graph query -q "
Resources
| where resourceGroup =~ '$RG'
| project id, name, type, location, tags
" -o json > evidence/04-inventory-after-create.json
```

Make one controlled change by setting access_tier = "Cool" in terraform.tfvars. Run a new plan and inspect whether the provider proposes an in-place update or a replacement. The expected result for this configuration is normally an in-place storage-account update, but the plan is the authority for the installed provider version. Do not proceed if the plan contains an unexplained replacement or deletion.

```
# 4. Plan and apply the controlled change
terraform plan -out=change.tfplan
terraform show -no-color change.tfplan   | tee evidence/05-change-plan.txt
terraform apply change.tfplan   | tee evidence/06-change-apply.txt

# 5. Preview destruction before authorizing it
terraform plan -destroy -out=destroy.tfplan
terraform show -no-color destroy.tfplan   | tee evidence/07-destroy-plan.txt

# 6. Apply the reviewed destroy plan
terraform apply destroy.tfplan   | tee evidence/08-destroy-apply.txt

# 7. Verify zero remainder
az group exists --name "$RG"   | tee evidence/09-group-exists-after-destroy.txt
terraform state list   | tee evidence/10-state-list-after-destroy.txt
```

Acceptance check	Required result
Create plan	Only the intended resource group and storage account are created
Create apply	Outputs contain the provider resource IDs and the inventory query finds the resources
Change plan	The plan explains the access-tier change and contains no unexplained replacement or deletion
Destroy plan	Every managed resource is visible as a planned deletion
Destroy apply	The apply completes without hidden retained child resources
Control-plane verification	az group exists returns false
State verification	terraform state list returns no managed addresses

Provider-version note

The provider constraint in a printed book is a tested range, not a permanent recommendation. Re-run the lab against the current stable Terraform and provider release before publication and in the companion repository. Commit the provider lock file generated by terraform init when the lab is placed under version control.

Multicloud translation

Retain the workflow and replace the provider contract: declare the nearest group/project and one low-cost resource; pin the provider; plan; apply; query the provider inventory; make one controlled change; preview destroy; destroy; and prove zero remainder. Record any resource whose provider lifecycle cannot be expressed safely.

4.14 CloudDesk Reuse: Organization, Tag Audit, Deployment, and Diff

CloudDesk FinOps Agent can later reuse the capabilities established in this chapter, but this chapter does not implement the agent. The read-only capabilities are straightforward: inventory a scope, normalize ownership metadata, calculate tag coverage, and explain drift or a proposed plan. Mutating capabilities require stronger controls because creating a scope or applying IaC can incur cost, change exposure, and delete data.

A safe Agent tool should never receive a generic Terraform shell. The tool broker should expose a bounded capability with an approved module, typed inputs, a fixed scope, a plan-only default, a plan digest, a risk classification, and separate approval for apply or destroy. The result must contain provider resource IDs and evidence links rather than an unstructured success message.

```
capability: cloud.deploy.stack
risk_level: R3_mutating_reversible_with_review
inputs:
  provider: azure
  module_id: approved/private-storage-v2
  target_scope: /subscriptions/.../resourceGroups/...
  parameters:
    owner: team-platform
    workload: finops-agent
    environment: dev
    costCenter: cc-4102
mode_default: plan_only
controls:
  - validate target scope and required tags
  - pin module and provider versions
  - return plan digest and destructive-operation summary
  - require approval bound to the plan digest before apply
  - use a least-privilege deployment identity
  - verify resource IDs, tags, policy state, and cost scope
  - require a separate destroy approval
outputs:
  plan_digest: sha256:...
  resource_ids: []
  evidence_links: []
  residual_resources: []
```

Capability	Purpose	Risk and minimum control
cloud.resource.organize	Create or move resources into an approved scope and apply required metadata	R2/R3; approved target scope, policy validation, preview, rollback limits
cloud.cost.tagging.audit	Measure missing, invalid, stale, and cost-unallocated tags	R0 read-only; Resource Graph/resource-center and billing read permissions
cloud.deploy.stack	Plan and apply an approved Bicep/Terraform module	R3; plan-only default, digest-bound approval, least-privilege execution, verification
cloud.resource.diff	Compare declared, tool-known, and actual provider state	R0/R1; read-only inventory and state metadata, no automatic remediation
cloud.resource.retire	Preview and execute bounded retirement with retention decisions	R4 for destructive data paths; separate approval, backup/retention checks, zero-remainder proof

4.15 Key Takeaways

- Resource organization is a control-plane design decision that shapes authority, policy inheritance, billing, blast radius, evidence, and lifecycle.
- Organization hierarchy, account/subscription, resource group/project, resource ID, name, and tag each solve a different problem and must not be collapsed into one naming convention.
- Tags are a governed metadata model. Required keys need controlled values, systems of record, consumers, compliance measurement, and periodic reconciliation.
- Cost allocation combines hierarchy, tags, provider billing data, inheritance where supported, and explicit rules for shared or unallocated charges.
- A queryable inventory connects resource IDs, ownership, type, location, lifecycle, and management metadata; activity logs, telemetry, and billing answer different questions.
- IaC is not merely configuration text. It is a contract among desired state, provider schema, execution identity, state or deployment metadata, dependency graph, evidence, verification, and retirement.
- Native IaC and Terraform provide different state and lifecycle models. Terraform standardizes a workflow, not the semantics of resources across clouds.
- State, plan artifacts, variable files, and deployment history can contain sensitive operational data and require explicit protection.
- Drift must be classified before it is reconciled. Some differences should be reverted, some imported, some delegated to another controller, and some documented as emergency changes.
- Destroy and zero-remainder verification are acceptance criteria for ephemeral infrastructure, not optional housekeeping.
- The two labs are independent: one proves organization and cost attribution; the other proves create-change-destroy lifecycle from declared state.
- CloudDesk may reuse bounded organization, tag-audit, deployment, diff, and retirement capabilities, but generic shell or unrestricted Terraform access is not a safe Agent tool.

References and Official Documentation

[1] National Institute of Standards and Technology. The NIST Definition of Cloud Computing. NIST SP 800-145, 2011. https://doi.org/10.6028/NIST.SP.800-145

[2] Microsoft. Resource organization design area overview. Cloud Adoption Framework. Updated January 9, 2026. https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/landing-zone/design-area/resource-org

[3] Microsoft. Organize your Azure resources effectively. Cloud Adoption Framework. Updated October 3, 2025. https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-setup-guide/organize-resources

[4] Microsoft. Define your naming convention. Cloud Adoption Framework. Updated June 25, 2026. https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-best-practices/resource-naming

[5] Microsoft. Define your tagging strategy. Cloud Adoption Framework. Updated July 1, 2025. https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-best-practices/resource-tagging

[6] Microsoft. Use tags to organize your Azure resources and management hierarchy. Microsoft Learn. Updated September 15, 2025. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/tag-resources

[7] Microsoft. Tag support for Azure resources. Microsoft Learn. https://learn.microsoft.-com/en-us/azure/azure-resource-manager/management/tag-support

[8] Microsoft. Overview of Azure Resource Graph. Microsoft Learn. https://learn.microsoft.com/en-us/azure/governance/resource-graph/overview

[9] Microsoft. Azure Resource Graph sample queries for Azure Resource Manager. Updated February 27, 2026. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/resource-graph-samples

[10] Microsoft. Quickstart: Run Resource Graph query using Azure CLI. https://learn.microsoft.com/en-us/azure/governance/resource-graph/first-query-azurecli

[11] Microsoft. Introduction to cost allocation. Microsoft Cost Management. Updated June 27, 2025. https://learn.microsoft.com/en-us/azure/cost-management-billing/costs/cost-allocation-introduction

[12] Microsoft. Group and allocate costs using tag inheritance. Updated June 27, 2025. https://learn.microsoft.com/en-us/azure/cost-management-billing/costs/enable-tag-inheritance

[13] Microsoft. Common cost analysis uses in Cost Management. Updated July 1, 2025. https://learn.microsoft.com/en-us/azure/cost-management-billing/costs/cost-analysis-common-uses

[14] Microsoft. Understand Cost Management data. Microsoft Learn. https://learn.microsoft.com/en-us/azure/cost-management-billing/costs/understand-cost-mgt-data

[15] Microsoft. Azure Resource Manager templates overview. Updated June 26, 2026. https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/overview

[16] Microsoft. Bicep file structure and syntax. Microsoft Learn. https://learn.microsoft.-com/en-us/azure/azure-resource-manager/bicep/file

[17] Microsoft. Deploy Bicep files with the Azure CLI. Updated May 27, 2026. https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/deploy-cli

[18] Microsoft. Bicep What-If: Preview changes before deployment. Updated March 3, 2026. https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/deploy-what-if

[19] Microsoft. Create and deploy Azure deployment stacks in Bicep. Updated June 24, 2026. https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/deployment-stacks

[20] Microsoft. Azure Resource Manager deployment modes. Updated June 26, 2026. https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/deployment-modes

[21] HashiCorp. Terraform State. https://developer.hashicorp.com/terraform/language/state

[22] HashiCorp. Purpose of Terraform State. https://developer.hashicorp.com/terraform/language/state/purpose

[23] HashiCorp. terraform plan command reference. https://developer.hashicorp.com/terraform/cli/commands/plan

[24] HashiCorp. terraform apply command reference. https://developer.hashicorp.com/terraform/cli/commands/apply

[25] HashiCorp. terraform destroy command reference. https://developer.hashicorp.com/terraform/cli/commands/destroy

[26] HashiCorp. Manage resource drift. https://developer.hashicorp.com/terraform/tutorials/state/resource-drift

[27] HashiCorp. Resource lifecycle meta-argument. https://developer.hashicorp.com/terraform/language/meta-arguments/lifecycle

[28] HashiCorp. Use modules in your configuration. https://developer.hashicorp.com/terraform/language/modules/configuration

[29] HashiCorp. Refactor modules.
https://developer.hashicorp.com/terraform/language/modules/develop/refactoring

[30] FinOps Foundation. Allocation capability. https://www.finops.org/framework/capabilities/allocation/

[31] FinOps Foundation. Cloud Cost Allocation Guide. https://www.finops.org/wg/cloud-cost-allocation/

[32] FinOps Foundation. How to Measure Tagging Policy Compliance. Updated March 17, 2026. https://www.finops.org/wg/how-to-measure-tagging-policy-compliance/

[33] FOCUS. FinOps Open Cost and Usage Specification 1.4.
https://focus.finops.org/docs/specification/v1-4/

[34] Alibaba Cloud. Compare models for resource organization and management.
https://www.alibabacloud.com/help/en/resource-management/product-overview/differences-and-relationships-among-the-resource-directory-resource-group-and-tag-services

[35] Alibaba Cloud. Overview of tag policies. Updated June 3, 2026. https://www.alibabacloud.com/help/en/resource-management/tag/user-guide/overview

[36] Alibaba Cloud. Enable automatic tag inheritance from a resource group. Updated June 3, 2026. https://www.alibabacloud.com/help/en/resource-management/tag/user-guide/enable-automatic-tag-inheritance-from-a-resource-group

[37] Alibaba Cloud. Resource Orchestration Service documentation. https://www.alibabacloud.com/help/en/ros/

[38] Alibaba Cloud. Create a Terraform template in ROS. Updated June 2, 2026.
https://www.alibabacloud.com/help/en/ros/user-guide/create-a-terraform-template

[39] Tencent Cloud. Tencent Cloud Organization documentation. https://www.tencentcloud.com/document/product/1031

[40] Tencent Cloud. Project Management.
https://www.tencentcloud.com/document/product/378/34726

[41] Tencent Cloud. Tagging.
https://www.tencentcloud.com/document/product/1172/52380

[42] Tencent Cloud. Tag Policy Overview.
https://www.tencentcloud.com/document/product/1031/55959

[43] Tencent Cloud. Cost Allocation Tags. Updated May 15, 2026. https://www.tencentcloud.com/document/product/555/32276

[44] Tencent Cloud. Managing existing resources with Terraform. https://www.tencent-cloud.com/document/product/1172/52393

[45] Volcengine. Project resource management. https://www.volcengine.com/docs/6649/94337

[46] Volcengine. Manage tags and resources. Updated February 11, 2025. https://www.volcengine.com/docs/6649/189381

[47] Volcengine. Resource Center overview. Updated July 16, 2025. https://www.volcengine.com/docs/6649/1109337

[48] Volcengine. Terraform concepts and provider documentation. Updated March 17, 2026. https://www.volcengine.com/docs/6706/2275245

[49] Volcengine. Infra Manager product introduction. Updated August 11, 2026. https://docs.volcengine.com/docs/88540?lang=zh

[50] Volcengine. Create an Infra Manager resource stack. Updated July 9, 2026. https://www.volcengine.com/docs/88540/2549851

Draft boundary note

This chapter establishes resource organization and IaC lifecycle discipline. It is not a complete Bicep or Terraform language tutorial. Compute, networking, storage, application PaaS, Kubernetes, GitOps, observability, FinOps, and AI PaaS apply this organization model in later chapters. Custom serverless Agent runtimes and tenant-isolation implementation belong to the separate book, Serverless AI Agent Development.

Part II / Chapter 5

Identity and Least Privilege

How Humans, Workloads, and Future Agents Receive Narrowly Scoped Authority

Identity is the primary control-plane boundary. Programs should receive identities and short-lived authorization rather than embedded credentials, and every permission should be explainable in terms of task, action, resource scope, condition, and lifetime.

In This Chapter

- Why identity is the primary security boundary of the cloud control plane
- How authentication, authorization, credentials, delegation, and audit evidence differ
- Why least privilege is an authority envelope rather than a role name
- How human identities, workload identities, managed identities, and federated identities should be separated
- How Azure RBAC combines a principal, role definition, scope, and optional condition
- Why control-plane permission does not automatically grant data-plane permission
- How Key Vault, temporary tokens, workload identity federation, and Privileged Identity Management reduce standing secrets and standing privilege
- How to divide identity across deployment, ingestion, retrieval, operations, and Agent tool execution
- How RAM, CAM, and Volcengine IAM translate the same engineering intent
- How to create and verify a read-only service identity without passing a client secret to the workload

A deployment pipeline is granted Owner at subscription scope because it must create one resource group. The web application uses the same service principal because the credential is already available. The ingestion job, retrieval API, administrative dashboard, and an experimental Agent all reuse that credential. The secret is copied into several CI variables and application settings. The system works. It also has no defensible authority boundary.

If the secret leaks, the attacker does not gain the permission needed by one task. The attacker gains the accumulated authority of every shortcut: resource creation, deletion, role assignment, data access, secret retrieval, and perhaps access across environments. If the Agent is prompt-injected, a model output can become a request issued under the same broad identity. The failure is not that authentication was absent. The failure is that authentication proved control of an identity whose authority was never designed.

This chapter treats identity as a cloud resource system rather than a login feature. The central question is not merely, “Can this caller obtain a token?” It is, “Which principal may perform which action on which resource, under which conditions, for how long, and what evidence will prove the decision?” That question applies to administrators, developers, pipelines, virtual machines, functions, Kubernetes workloads, AI ingestion processes, retrieval applications, and future Agent tool executors.

The chapter thesis

Least privilege is not achieved by selecting a role whose name sounds safe. It is achieved when the principal, permitted actions, resource scope, conditions, credential path, and lifetime are all no broader than the verified task requires.

Identity is the access decision path

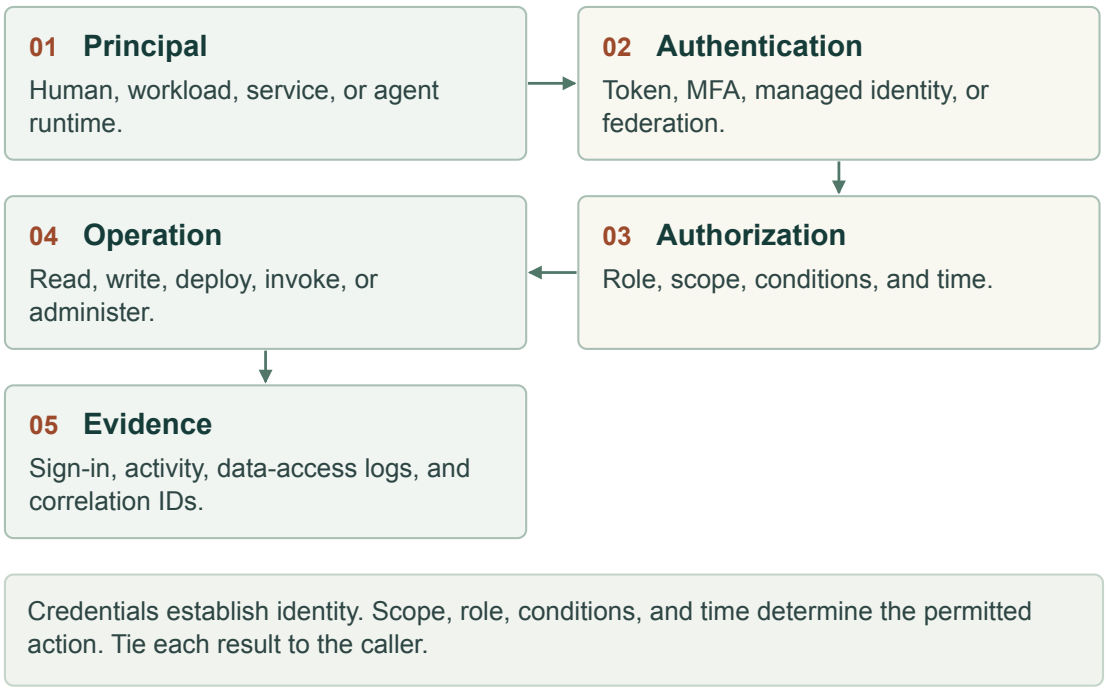


Figure 5-1 Identity connects authentication, authorization, operation, and accountable evidence.

5.1 Identity Is the Primary Control-Plane Boundary

Chapter 1 described the cloud as a programmable control plane. Every meaningful management request must arrive from a principal. The portal signs in a human. A CLI or infrastructure pipeline obtains a token. A managed application calls an API. A virtual machine, function, container, or Kubernetes pod accesses a downstream resource. The control plane evaluates the caller before it accepts a lifecycle operation such as create, update, assign, scale, or delete.

Network isolation, encryption, policy, and monitoring remain essential, but identity decides who may cross those boundaries and change their configuration. A private endpoint protects a service from public reachability, yet an overprivileged principal inside the trusted network can still delete or reconfigure the resource. Encryption protects data at rest, yet a principal with permission to retrieve plaintext can legitimately decrypt it. Policy can block a noncompliant resource state, yet the people allowed to exempt or modify policy remain an identity question.

For this reason, identity should be designed before application wiring. The dependency graph should state which workload speaks to which resource and under which identity. The deployment identity should be distinct from the runtime identity. A read path should not silently carry a write permission. A monitoring identity should not be able to mutate the workload it observes. A tool executor should not be able to grant itself additional roles.

Table 5-1 Identity vocabulary used throughout the book.

Term	Durable meaning	Engineering question
Principal	The human, group, workload, service, role, or managed identity that requests access.	Who or what is acting?
Credential	Proof material used to obtain or present an authentication result: password, key, certificate, external token, or platform-managed credential.	How does the principal prove control without exposing reusable secrets?
Token	A time-limited assertion issued after authentication and presented to a resource or authorization service.	Which audience, claims, lifetime, and delegated authority does the token carry?
Authentication	Verification that the claimant controls the asserted identity.	Is the caller the principal it claims to be?
Authorization	Decision that an authenticated principal may perform an action on an eligible resource.	May this principal perform this action here and now?

Term	Durable meaning	Engineering question
Role definition / policy	A named or structured collection of permitted and excluded operations, often including resource and condition syntax.	What actions are described?
Role assignment / binding	Association of a principal with a role or policy at a scope.	Where does the permission apply, and how is it inherited?
Delegation	Authority to assign, assume, or pass authority to another principal.	Can this identity create more authority than the task requires?
Audit evidence	Sign-in, authorization, activity, data-access, and role-change records.	Can the decision and its effect be reconstructed?

5.1.1 Authentication is not authorization

Authentication answers an identity question. Authorization answers a permission question. OAuth 2.0 is an authorization framework for limited access; OpenID Connect adds an identity layer for authenticating an end user. In cloud engineering, those protocols may contribute to a sign-in or token exchange, but the resource still evaluates what the resulting principal may do. A valid token can be correctly authenticated and correctly denied.

This distinction is operationally useful. A 401-style failure commonly indicates missing or unacceptable authentication. A 403-style failure commonly indicates that the caller is known but not authorized. The exact status and error contract vary by service, but the diagnostic sequence is stable: identify the principal, inspect the token audience and claims, determine the requested action and resource, enumerate effective assignments and conditions, then check policy or data-plane controls.

5.1.2 Credentials are not identities

A client secret is not a service principal. An AccessKey pair is not a RAM user. A password is not a person. Credentials are replaceable proof mechanisms associated with an identity. Treating the credential as the identity leads to shared keys, copied secrets, unclear ownership, and missing lifecycle. A secure design gives the principal a name and purpose, then selects the least risky authentication mechanism available for that environment.

The best credential is often one the application cannot read or export. Azure managed identities are special service principals whose credentials are managed by the platform; applications request short-lived Microsoft Entra tokens without storing a secret. Workload identity federation allows an external workload, such as a GitHub Actions job or Kubernetes service account, to exchange a trusted OIDC token for a cloud access token

without maintaining a cloud client secret. Alibaba Cloud RAM roles, Tencent Cloud CAM roles, and Volcengine IAM roles similarly use temporary credentials after role assumption, although the exact workload integration and service coverage differ. [5] [6] [14] [17] [20]

5.2 Least Privilege Is an Authority Envelope

NIST defines least privilege as restricting users or processes to the minimum system authorizations and resources needed to perform their function. That definition contains two ideas that cloud designs often separate incorrectly: authorization must be minimal, and the eligible resources must also be minimal. A read-only role over an entire organization can expose more information than a narrowly scoped operator role over one resource. A powerful role activated for five minutes after approval can create less standing risk than a moderate role assigned permanently to every engineer.

A useful cloud formulation is:

```
effective authority = principal × actions × resource scope × conditions × time
```

The expression is conceptual, not arithmetic, but the multiplicative intuition matters. Broadening any dimension enlarges the blast radius. Least privilege is reached when every dimension has an explicit reason and a negative test.

Five boundaries of least privilege

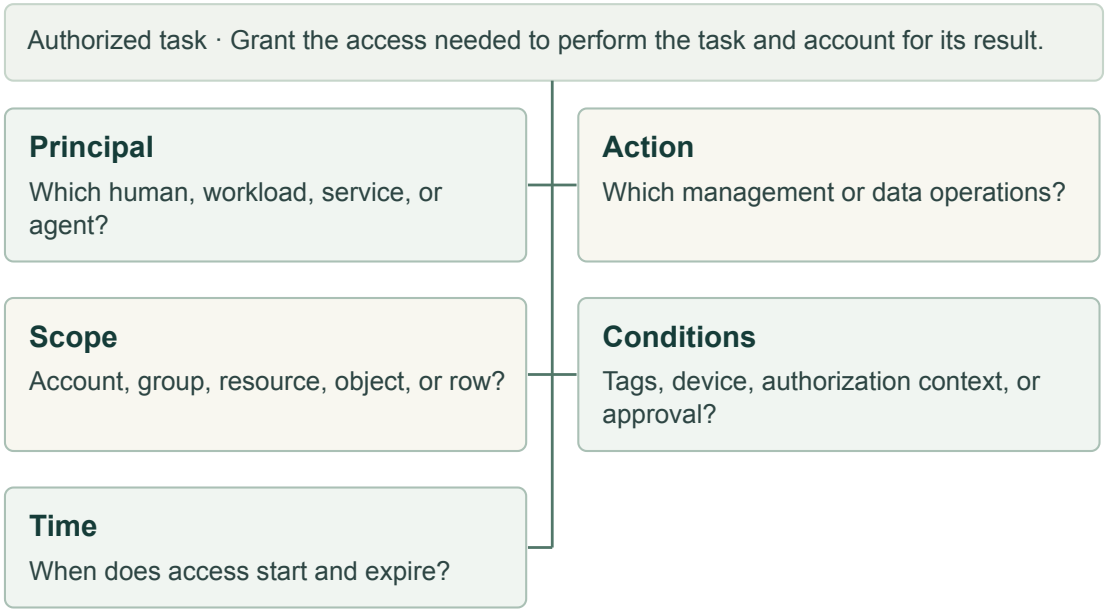


Figure 5-2 Least privilege bounds principal, actions, scope, conditions, and time around one authorized task.

5.2.1 Principal: one purpose, one lifecycle

Use a distinct principal for a distinct operational purpose. A deployment pipeline, application runtime, ingestion worker, retrieval API, monitoring collector, and emergency administrator have different lifecycles and different evidence requirements. Reusing one identity hides these differences. When a role appears in an audit event, the operator should be able to infer the purpose from the identity name and ownership metadata.

Direct assignments to individuals can be appropriate for exceptional tasks, but team permissions should normally be managed through groups or governed roles. Group membership provides a lifecycle boundary for joiners, movers, and leavers. Workload identities should have an owner, environment, application, and expiry or review metadata even when the platform does not require those fields.

5.2.2 Actions: separate job functions and data access

A role definition should express the operations needed by a job function. Avoid starting with Owner, Administrator, or a wildcard and subtracting a few obvious dangers. Begin with the task and enumerate the provider operations. Prefer a built-in job-function role

when it fits. Create a custom role only when the built-in alternatives are materially broader, and preserve the analysis that justified every action.

Management and data operations require separate attention. Creating a storage account is a management action. Reading blobs is a data action. Creating a Key Vault is a management action. Reading a secret value is a data action. Creating a model deployment is a management action. Sending an inference request is a data action. A permission review that inspects only management actions can approve a workload that cannot run, or miss an identity that can read sensitive data.

5.2.3 Scope: inheritance is convenient and dangerous

Cloud permissions usually inherit downward through an organization or resource hierarchy. A role at a management group, organization, account, subscription, project, or resource group can apply to many descendants. Inheritance simplifies administration, but it also turns a locally reasonable role into broad standing access. The scope should be the narrowest stable boundary that contains the resources required by the task.

Resource-level assignments are not always the best operational answer. Thousands of one-off assignments can become unreviewable. The design objective is narrow and governable: prefer a dedicated resource group, project, workspace, vault, or account boundary when it expresses a coherent application or environment. The boundary should support delegation, cost attribution, policy, and deletion—not only RBAC.

5.2.4 Conditions and attributes: filter authority, do not replace it

Attribute-based conditions can narrow a role assignment by evaluating resource, request, environment, or principal attributes. Azure ABAC extends selected Azure RBAC scenarios with role-assignment conditions. Other providers expose condition syntax in their policy languages. Examples include restricting access to objects with a project tag, limiting a role to a source network, or constraining which roles an administrator is allowed to assign.

Conditions should not compensate for an unnecessarily broad principal or role. They add another authorization dependency that must be tested, versioned, and audited. A security review should state what happens when an attribute is missing, changed, or malformed, and should include a negative test proving that an ineligible resource is denied.

5.2.5 Time: remove standing privilege

Human administration often needs short bursts of elevated authority rather than a permanent assignment. Microsoft Entra Privileged Identity Management can make a user eligible for an Azure or directory role, require approval, multifactor authentication, and justifica-

tion, and automatically remove the activated role after a time window. Comparable just-in-time designs can be implemented through provider role assumption, temporary sessions, and approval workflows. [8]

For workloads, time is expressed through token lifetime and identity lifecycle. Temporary tokens reduce the value of a captured credential, but they do not compensate for a role that is too powerful. An automatically refreshed token can create continuous effective authority. Review both the session duration and the standing role assignment that allows the session to be created.

Least privilege is not "read-only everywhere"
The correct question is not whether an identity can write. It is whether it can perform exactly the verified task. A deployment identity must write selected resource types. A retrieval identity may need data-plane read but no management write. A cost collector may read usage records but not business data. Least privilege is task-shaped authority.

5.3 Human, Workload, Service, and Agent Identities

Table 5-2 Identity classes have different authentication and lifecycle needs.

Identity class	Typical authentication	Primary controls	Common failure
Human workforce	Passwordless or multifactor sign-in, federation, Conditional Access.	Groups, job-function roles, PIM, access reviews, separation of duties.	Permanent administrator roles and personal accounts reused by automation.
Application or service principal	Certificate, federated token, or secret when unavoidable.	Named owner, narrow role, credential rotation, sign-in monitoring.	Client secret copied into repositories, pipelines, and application settings.
Managed identity	Platform-managed credential and short-lived token acquisition.	Role assignment, resource attachment, lifecycle ownership, activity and sign-in logs.	One shared identity attached to unrelated workloads or left after the workload is retired.
Federated workload identity	External OIDC/SAML assertion exchanged for a cloud token.	Exact issuer, subject, audience, repository/service-account binding, short session.	Broad trust pattern that lets an unintended workflow or namespace assume the identity.
Cloud service role	Provider service assumes a service-linked or trusted role.	Trust policy, permission policy, service ownership, change audit.	Granting a service role to a human or copying its policy to unrelated identities.

Identity class	Typical authentication	Primary controls	Common failure
Future Agent runtime	Workload identity plus constrained tool-executor identities.	Tool allowlist, approval, request binding, task scope, evidence, no self-delegation.	Giving the language model a broad cloud credential or role-assignment authority.

5.3.1 Human identity: reduce permanent administration

Human operators need a directory identity with strong authentication, but authentication strength does not reduce authorization breadth. Use groups for recurring job functions, separate production administration from everyday development, and make privileged roles eligible or time-bound when the platform and licensing permit. Emergency or break-glass identities should be few, independently monitored, excluded from normal workflows, and tested under a documented incident procedure.

Separation of duties matters most around authority creation. The person who writes an infrastructure template should not automatically be the only person who can approve and deploy it to production. The identity that can assign roles, modify trust policies, or retrieve signing keys deserves a higher review threshold than a general resource contributor. A role-assignment administrator can often create more risk than the workload administrator because delegation changes who may act later.

5.3.2 Workload identity: eliminate embedded credentials

A workload should authenticate as itself rather than impersonating a developer. The identity should survive or disappear according to an intentional lifecycle. A system-assigned managed identity is coupled to one Azure resource and disappears with it. A user-assigned managed identity is an independent Azure resource that can be preauthorized and attached to several compute resources. Microsoft currently recommends user-assigned managed identities for Microsoft service scenarios that need independent lifecycle, reuse, or preauthorization. [5]

For workloads outside Azure, workload identity federation establishes trust in an external issuer. The cloud identity trusts a specific issuer, subject, and audience, then exchanges the external token for a Microsoft Entra access token. The design removes the cloud secret but creates a trust-policy obligation. A repository pattern such as repo:organization/* is easier to configure than an exact environment or branch subject and may be dangerously broad. Federation is credential-free, not policy-free.

5.3.3 Service identity: trust and permission are separate

A cloud service may need authority to act on behalf of a customer resource: a managed database writes backups, an indexer reads a data source, a function reads a secret, or an AI service calls an embedding deployment. The trust policy answers who may assume the role or use the identity. The permission policy answers what the assumed identity may do. Mixing those questions creates roles that can be assumed by too many principals or that perform too many actions.

Service-linked roles and provider-managed roles reduce setup work, but they should still appear in inventory and change review. Their policies can evolve with the service. Do not attach a service-linked policy to an unrelated human or workload identity merely because the permissions appear useful.

5.3.4 Agent identity: the model is not the principal

An Agent system may plan actions in natural language, but the language model should not be treated as the security principal. The principal is the runtime component that obtains a token and invokes a tool. The runtime must validate the requested action, resource, parameters, approval state, and identity before execution. Model output is untrusted input to that decision.

This book stops at the managed cloud capability boundary. The companion volume implements the CloudDesk tenant system, response binding, cross-tenant isolation, desktop approval, and custom tool runtime. The durable principle belongs here: an Agent should receive a governed tool contract, not a general cloud credential.

5.4 Azure as the Reference Identity Model

Azure provides a useful teaching model because directory identities, Azure resource roles, managed identities, policy, secret management, activity evidence, and AI PaaS resources participate in a coherent control-plane vocabulary. The object model is not simple, but its distinctions are productive.

Azure reference model: directory principals meet resource scopes

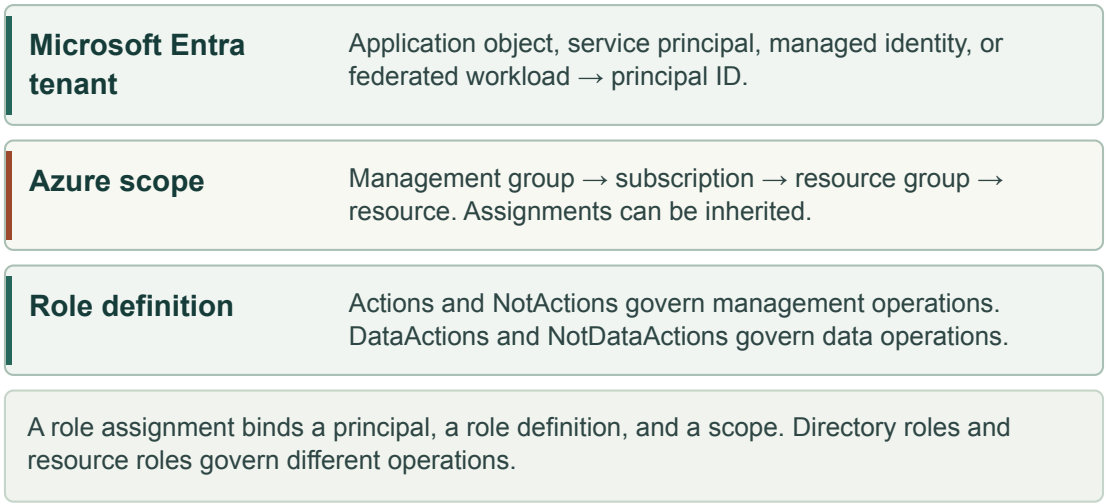


Figure 5-3 Microsoft Entra principals are bound to Azure resource roles at explicit scopes.

5.4.1 Application object, service principal, and managed identity

An application registration creates an application object that describes an application in its home Microsoft Entra tenant. A service principal is the local identity of that application in a tenant—the object to which roles, consent, and policy can be applied. A multitenant application can therefore have one application object and a service principal in every tenant that accepts it. [4]

A managed identity is a special type of service principal managed for an Azure resource or created as a standalone user-assigned resource. It is not an application user account. Its credential is not exported to the developer. The workload uses the Azure identity libraries, Microsoft Authentication Library, or an integrated connector to request a token from the platform environment. [5]

These objects answer different lifecycle questions. The application object represents the software definition. The service principal represents the tenant-local security identity. The managed identity represents a credential-free workload identity tied to Azure resource operation. Naming them all “the app” makes authorization reviews ambiguous.

5.4.2 Azure role assignment = principal + role + scope (+ condition)

Azure RBAC grants access through a role assignment. The assignment binds a security principal—user, group, service principal, or managed identity—to a role definition at a scope. Scope can be a management group, subscription, resource group, or individual resource, and permissions assigned at a parent scope are inherited by child scopes. Optional conditions can further narrow selected assignments. [2] [3] [7]

Table 5-3 The components of an Azure role assignment.

Component	Question	Review evidence
Security principal	Which object will receive authority?	Object ID, principal type, owner, purpose, lifecycle.
Role definition	Which Actions, DataActions, and exclusions are described?	Built-in or custom role ID, versioned JSON, provider operations.
Scope	Which resources inherit the assignment?	Full Azure resource ID, parent hierarchy, child inventory.
Condition	Which attributes or delegated role operations narrow the assignment?	Condition expression, version, test cases, applicable actions.
Assignment state	Is access active, eligible, time-bound, inherited, or permanent?	PIM record, start/end time, assignment source, reviewer.
Evidence	What proves use, denial, change, and removal?	Sign-in logs, Activity Log, data logs, correlation ID, access review.

5.4.3 Control-plane and data-plane roles

Azure role definitions separate management Actions and NotActions from data-plane DataActions and NotDataActions. This prevents a management wildcard from silently becoming permission to read every future data type. A Reader assignment lets a principal inspect Azure resource configuration at its scope; it does not automatically let the principal list blobs, read Key Vault secret values, query a database, retrieve search documents, or invoke every model endpoint. [3] [10]

Table 5-4 Management authority and data authority are separate.

Resource	Management-plane task	Data-plane task	Least-privilege implication
Storage	Read account properties, networking, redundancy, and lifecycle settings.	List, read, write, or delete blobs.	Reader is not Storage Blob Data Reader; review both roles.

Resource	Management-plane task	Data-plane task	Least-privilege implication
Key Vault	Create or configure the vault, networking, diagnostics, and role assignments.	Read a secret value, sign with a key, or retrieve a certificate.	Key Vault Contributor manages the vault but does not read secret values.
Azure AI Search	Create the service, configure networking and diagnostics.	Index documents or issue search/retrieval queries.	Separate deployment, ingestion, and retrieval identities.
Foundry/model resource	Create the account/project or model deployment and configure quota.	Invoke inference, evaluate, or use Agent/knowledge data-plane APIs.	A resource contributor should not be the production inference caller.
Kubernetes service	Create the managed cluster and attach identity/network.	Create Kubernetes objects or access application data.	Azure RBAC and Kubernetes authorization are composed control planes.

Key Vault provides a particularly clear example. Control-plane access manages the vault resource; data-plane access manages keys, secrets, and certificates. Microsoft recommends Azure RBAC for Key Vault authorization and provides data roles such as Key Vault Secrets User. A Key Vault Contributor can manage the vault control plane but cannot read secret contents. Legacy access-policy configurations require special caution because a sufficiently privileged control-plane user may be able to grant itself data access. [10]

5.4.4 Managed identity and workload identity federation

Managed identity is the default choice for a workload hosted on a supported Azure resource. The source resource receives the identity; the target resource receives an RBAC assignment for that identity; the workload requests a short-lived token at runtime. No client secret is placed in application configuration. System-assigned identity is convenient for one resource. User-assigned identity supports preauthorization, reuse, and independent lifecycle. [5]

Workload identity federation is the default direction for external build systems, Kubernetes workloads, and cross-cloud compute. The external identity provider issues a signed token. Microsoft Entra checks the configured issuer, subject, and audience and exchanges it for an access token. This removes secret storage and rotation but still requires exact trust, token audience, role scope, and revocation design. [6]

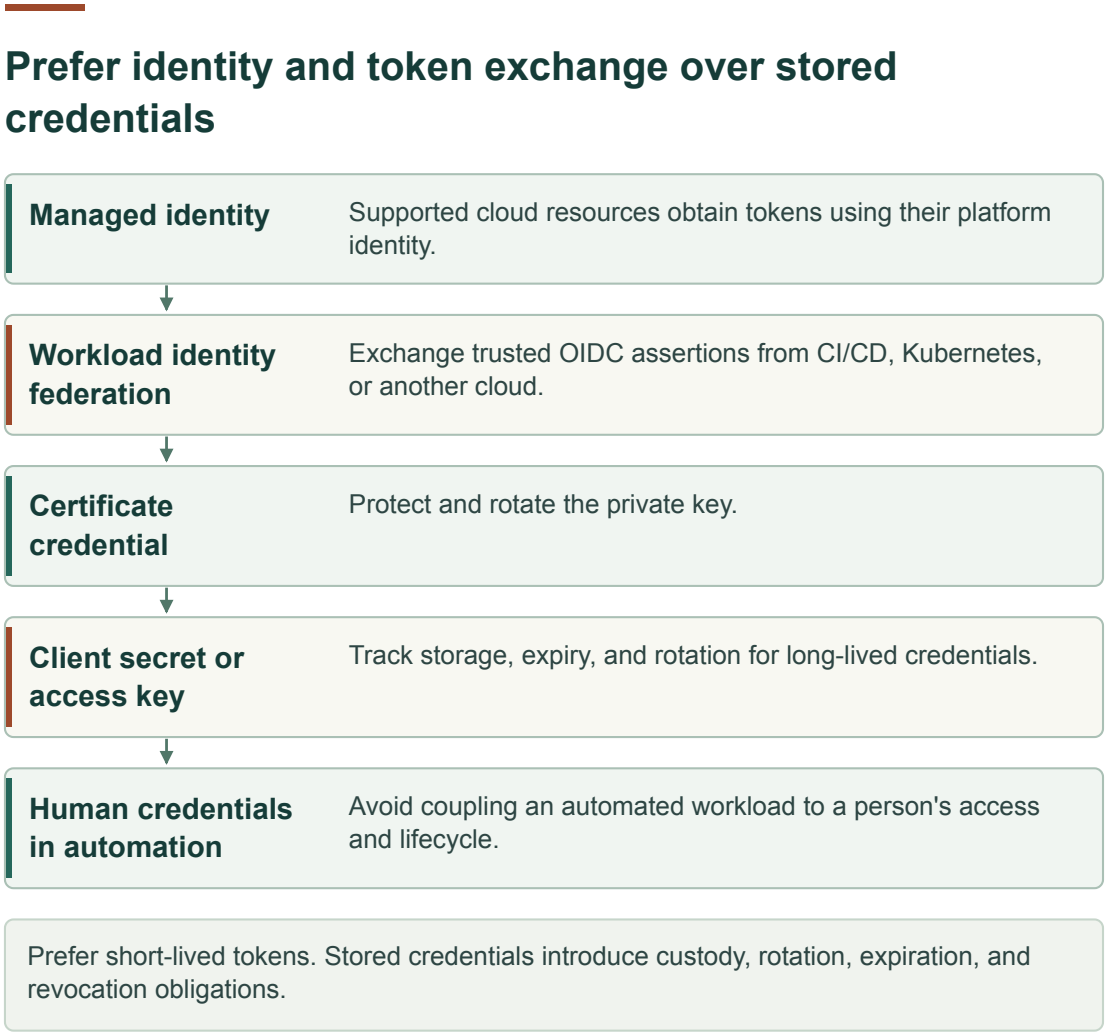


Figure 5-4 Prefer platform-managed identity or token federation; stored credentials are fallbacks with increasing operational risk.

5.4.5 Human privileged access and PIM

Permanent Owner or User Access Administrator assignments at subscription scope create standing privilege over both resources and future role assignments. Azure RBAC guidance recommends removing unnecessary privileged assignments, preferring job-function roles, and narrowing privileged scope. Microsoft Entra Privileged Identity Management adds eligible and time-bound assignments, approval, MFA, justification, notification, access reviews, and audit history. [8] [9]

PIM is not a substitute for role design. Activating Owner for thirty minutes is still broader than activating a narrowly defined production-deployer role. Use PIM to reduce standing time, and use RBAC to reduce actions and scope. High-risk operations such as role assign-

ment, policy exemption, key recovery, and production break-glass should have a distinct activation and evidence path.

5.4.6 RBAC and Policy answer different questions

Azure RBAC evaluates whether a principal may perform an action at a scope. Azure Policy evaluates whether the resulting resource state complies with an assigned rule. An engineer may be authorized to create a storage account, while Policy denies the request because public network access is enabled or required tags are absent. Conversely, Policy does not grant the engineer permission to make the request. The two systems compose: RBAC controls actors and actions; Policy controls acceptable state. [11]

This distinction prevents a common anti-pattern: granting a broad role and expecting policy to make it least privilege. Policy may block some noncompliant outcomes, but the principal may still read sensitive configuration, perform compliant but destructive actions, or operate on resources outside the policy coverage. Least privilege starts with identity and authorization, then adds guardrails.

5.5 Secrets Are Not an Identity Strategy

Some systems still require passwords, API keys, database credentials, signing material, or third-party tokens. A secret manager reduces the risk of storing and distributing those values, but it does not by itself create a least-privilege identity system. The application needs an identity to retrieve the secret. That first authentication step is the secret-zero problem.

Use managed identity or federation to reach the secret store when possible. Give the workload permission to read only the required secret or vault boundary. Separate secret administration from secret consumption. Record retrieval, rotation, and version changes. Prevent secret contents from entering logs, traces, prompts, build output, or error messages. Rotate on a schedule and immediately after suspected exposure.

Table 5-5 Secret management responsibilities.

Responsibility	Control-plane decision	Evidence
Storage	Which vault or secret manager stores the value, in which region and environment?	Resource ID, encryption configuration, private endpoint, tags.
Retrieval identity	Which workload may obtain the value?	Managed/federated principal and data-plane role assignment.
Administration	Who may create, rotate, recover, or delete the secret?	Separate administrator role, PIM activation, change event.

Responsibility	Control-plane decision	Evidence
Version and rotation	When does the value change and how do dependents adopt it?	Version ID, rotation job, staged rollout, rollback record.
Exposure prevention	Where must the plaintext never appear?	Repository scan, pipeline masking, log redaction, prompt filtering.
Retirement	What proves that the secret and all references are gone?	Disabled version, dependency inventory, deletion/recovery evidence.

Secrets remain necessary—but they should not be the default cloud credential
A secret manager is the correct home for a secret. It is not a reason to create one. Prefer managed identity or workload federation for cloud authentication; use secret storage for credentials that cannot yet be eliminated.

5.6 Designing Least Privilege as a Repeatable Process

Least privilege fails when permissions are assigned by intuition. The same design method should be used for a human administrator, deployment pipeline, function, Kubernetes workload, RAG ingestion job, or Agent tool executor.

5.6.1 Start from a task contract

Write the task in observable terms: “List resource metadata in resource group X and emit a cost-attribution report,” not “needs Azure access.” Identify the input, output, failure behavior, environment, expected frequency, and whether the task changes state. If the task cannot be explained without the word “everything,” the boundary is not ready for authorization.

5.6.2 Enumerate positive and negative operations

List the management and data operations that must succeed. Then list the adjacent operations that must fail. A retrieval application may query a knowledge base and invoke one model deployment; it should fail to rebuild the index, modify the model deployment, assign roles, or enumerate unrelated storage. Negative requirements become executable authorization tests.

5.6.3 Choose the principal and credential path

Select an identity whose lifecycle matches the task. Use a group for a workforce function, a managed identity for Azure-hosted compute, or a federated identity for an external pipeline or Kubernetes service account. Avoid sharing a principal across environments. Name

the owner and retirement condition. Decide how tokens are obtained without placing reusable material in code or configuration.

5.6.4 Select the narrowest governable scope

The narrowest resource is not always the most maintainable boundary. Prefer an application resource group, project, workspace, vault, or dedicated resource when it provides coherent ownership and cleanup. If several resources must share one workload identity, document the grouping and ensure that unrelated future resources cannot be added silently under the inherited scope.

5.6.5 Prefer built-in roles, then measure the gap

A built-in role is easier to recognize and receives provider maintenance, but it may be broader than the task. Compare its actions with the task contract. If the gap is unacceptable, define a custom role from observed provider operations rather than from a guessed wildcard. Version the role definition as code. Review newly added provider operations before extending a wildcard custom role.

5.6.6 Add time, condition, and delegation boundaries

Human privilege should be eligible or time-bound where possible. Workload sessions should use temporary tokens. Attribute conditions can restrict object tags, repositories, containers, or role delegation. Treat roleAssignments/write, AssumeRole, trust-policy updates, and secret-administration permissions as delegation capabilities. These actions deserve a separate identity and approval path.

5.6.7 Prove both allow and deny paths

A permission design is not verified when the expected operation succeeds. It is verified when required operations succeed, forbidden operations fail, the failure is attributable to authorization rather than an unrelated error, and the evidence identifies the principal, scope, operation, and correlation ID. Repeat tests after role changes and at publication or deployment freeze because service actions and built-in roles evolve.

5.6.8 Review use and remove authority

Role assignments accumulate because creation is part of delivery while removal is treated as cleanup. Make review and retirement first-class lifecycle operations. Inventory unused principals, stale credentials, broad assignments, inherited privileges, unowned service identities, and identities attached to no active resource. Remove the role assignment before deleting an identity so that evidence remains understandable, then verify that all references are gone.

```
identity_design:
  principal: id-cost-reader-prod
  purpose: "Read Azure resource metadata for cost attribution"
  required_actions:
    - Microsoft.Resources/subscriptions/resourceGroups/read
    - "resource provider read operations within rg-finops-prod"
  prohibited_actions:
    - Microsoft.Resources/subscriptions/resourceGroups/write
    - Microsoft.Authorization/roleAssignments/write
    - "all business-data DataActions"
  scope: /subscriptions/<sub>/resourceGroups/rg-finops-prod
  credential: user-assigned managed identity
  lifetime: assignment reviewed every 90 days
  evidence:
    - role assignment ID
    - successful inventory query
    - failed write attempt
    - Activity Log correlation ID
```

5.7 Identity Boundaries for AI PaaS and Agentic Applications

AI-native applications extend, rather than replace, the identity control plane. Model endpoints, knowledge sources, vector indexes, content-safety services, evaluation data, tool connections, storage, telemetry, and cost APIs all expose distinct operations. Collapsing them into one “AI app identity” creates a broad and opaque principal.

AI-native systems need distinct identities

Deployment
Manage model, network, and search resources without automatic access to customer data.

Ingestion
Read approved sources, transform content, and write the index.

Retrieval
Query authorized knowledge and return citations.

Operations
Read telemetry, health, and audit. Grant configuration changes separately.

Tool execution
Invoke approved actions with a dedicated, narrowly scoped identity.

Each service authorizes the requested scope and records the result. Model tool selection grants no permissions.

Figure 5-5 Deployment, ingestion, retrieval, operations, and tool execution should use separate authorities.

Table 5-6 A minimum identity split for an AI-native application.

Identity	Allow	Explicitly exclude
AI resource deployer	Create/update approved model, search, network, diagnostic, and policy resources through IaC.	Production inference, business-data reads, secret-value reads, unrestricted role assignment.
Knowledge ingestion worker	Read approved source locations, call one embedding deployment, write one target index, report ingestion telemetry.	Change model deployments, query unrelated sources, update access policy, answer user queries.
Retrieval application	Use approved retrieval or knowledge-base endpoints, invoke approved model deployment, emit citations and trace context.	Index schema changes, source browsing outside retrieval, secret administration, role assignment.
Evaluator	Read designated test sets and captured outputs, submit evaluation jobs, write evaluation results.	Production mutation, unrestricted prompt-log access, deployment or policy change.

Identity	Allow	Explicitly exclude
Operator / SRE	Read health, logs, metrics, traces, quota, cost, and deployment status; initiate approved rollback.	Business data by default, model-key export, arbitrary tool execution.
Agent tool executor	Execute one validated tool action on the approved resource scope with task and approval binding.	General cloud shell, permission administration, self-delegation, unrelated tenant or project resources.

5.7.1 Model access is a data-plane permission

The identity that creates or configures a model resource should not automatically be the identity that invokes production inference. Separate management roles from data-plane roles. Apply quota and cost attribution to the calling identity or application scope when supported. A direct API key may be convenient for a prototype, but a resource-level AI PaaS design should prefer directory authentication, private networking, rotation-free workload identity, and auditable role assignments where the service supports them.

5.7.2 Retrieval authorization must be deterministic

The retrieval identity may query a shared knowledge resource, but document eligibility must be enforced by trusted authorization context and filter policy. The language model must not construct its own tenant, workspace, or ACL predicate. Chapter 18 develops the knowledge-resource lifecycle; the identity rule is simple: the authenticated application context creates the authorization filter, and negative tests prove that ineligible chunks never enter model-visible context.

5.7.3 Tool authority must be narrower than Agent intent

An Agent can request “optimize cloud cost,” but that intent is not a permission. The tool contract should expose a narrow operation such as list idle VMs in resource group X, create a dry-run recommendation, or stop a specifically approved VM. The executor identity should hold only the provider actions required by that contract. High-risk actions should require approval, parameter validation, idempotency, and evidence. The executor should never possess role-assignment authority unless the explicit product purpose is governed access administration.

5.8 Translating Identity Across Azure and China’s Leading Cloud Platforms

The four platforms expose the same fundamental identity problem but not identical resource hierarchies, policy grammars, role integrations, or managed-identity coverage. Translate the engineering intent—named principal, temporary authentication, least-privi-

lege policy, scope, trust, secret management, and evidence—then verify the actual service contract.

Table 5-7 Identity and least-privilege control-plane translation.

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Workforce identities	Microsoft Entra users/groups; Azure roles; PIM for eligible/time-bound access.	RAM users/groups; enterprise or resource-directory identity features; role assumption for temporary access.	CAM sub-users/groups and organization identity-management features; CAM roles for account access.	IAM users/groups and organization/account controls; verify workforce SSO coverage for the target region.
Workload identity	Managed identities; service principals; workload identity federation.	RAM roles, ECS instance RAM roles, Function Compute roles, OIDC/RRSA scenarios, STS tokens.	CAM roles, STS temporary credentials, OIDC role integration for selected TKE and service scenarios.	IAM roles and AssumeRole temporary credentials; service-specific attachment mechanisms vary.
Authorization model	Role definition + role assignment + Azure scope; selected ABAC conditions.	System/custom RAM policies, resource ARN, condition, trust policy, resource group/account scope.	Preset/custom CAM policies, QCS resource syntax, conditions, roles and trust policies; API granularity varies by service.	System/custom IAM policies attached to users, groups, or roles; TRN resource syntax and conditions.
Temporary credential	Microsoft Entra access token; managed identity token; federated exchange.	STS token with temporary AccessKey pair and security token after role assumption.	STS token with temporary SecretId/SecretKey and session token after role assumption.	Temporary AK/SK and token after role assumption where supported.
Secret management	Azure Key Vault with RBAC, managed identity, private access, rotation integrations.	KMS Credential Manager / secret features; RAM-role access; supported credential rotation.	Secrets Manager (SSM) integrated with CAM, CloudAudit, rotation, and Kubernetes scenarios.	KMS and service credential features; verify secret-manager and workload integration for the selected service.
Evidence	Entra sign-in logs, Activity Log, resource/data logs, access reviews.	ActionTrail, RAM/STS records, service logs, KMS records.	CloudAudit, CAM/STS records, SSM audit and service logs.	CloudTrail/audit and IAM/service records according to product support; verify data-event coverage.

5.8.1 Alibaba Cloud RAM

Alibaba Cloud RAM separates users, groups, and roles. A RAM role has no password or permanent AccessKey pair. A trusted principal assumes the role and receives a temporary STS token containing a temporary AccessKey pair and security token. Current RAM guidance recommends STS tokens for programmatic access because the credential expires automatically and limits exposure compared with a long-lived AccessKey pair. [14] [15]

For workloads on Alibaba Cloud, instance RAM roles and Function Compute roles allow the platform to provide temporary credentials without embedding an AccessKey. KMS Credential Manager can store credentials that cannot be eliminated and supports controlled retrieval and selected rotation scenarios. The permission review must inspect both the role trust policy and the attached permission policies. [16]

5.8.2 Tencent Cloud CAM

Tencent Cloud CAM provides users, groups, roles, and policies. Roles do not require a permanent key; STS APIs issue temporary credentials whose operations, resources, conditions, and validity are limited by policy and session. Tencent documentation recommends least privilege for sub-accounts and separates human sub-accounts from programmatic identities. [17] [18]

Secrets Manager (SSM) centralizes sensitive values, integrates with CAM for access control and CloudAudit for evidence, and supports secret lifecycle and rotation scenarios. Current TKE guidance also describes OIDC and CAM-role integration for password-free workload access in selected scenarios. Exact resource-level authorization granularity differs by service and API; a policy that names a resource does not make an operation resource-scoped if the service supports only operation-level authorization. [19]

5.8.3 Volcengine IAM

Volcengine IAM exposes users, user groups, roles, and policies. A policy attached to an identity grants the described permissions; roles use trust relationships and temporary credentials obtained through role assumption. Volcengine publishes IAM policy elements and TRN resource formats, and its access-key guidance recommends separate workload users with minimum permissions when keys remain necessary. [20] [21]

Service-by-service identity integration and resource granularity must be verified. Do not infer that every workload receives a platform-managed identity simply because the global IAM service supports roles. Record whether the specific compute, function, container, database, model, or Agent service can assume a role without a stored key, what token lifetime applies, and which audit service records the call.

5.9 Operating the Identity Control Plane

Identity is not finished after a role assignment is created. It must be observed, reviewed, and retired. A production identity program should answer four operational questions: Which principals exist? Which authority is effective? Which authority is used? Which authority is no longer justified?

Table 5-8 Identity operating signals and review questions.

Signal	Question	Example action
Privileged assignment inventory	Who has Owner, access-administration, policy-exemption, key-recovery, or equivalent authority?	Move to narrower roles, scopes, and PIM eligibility.
Unused permission and unused identity	Has the principal signed in or used the role during the review window?	Remove stale role assignment; disable and retire identity after dependency check.
Credential age and type	Does the workload still depend on a long-lived secret or AccessKey?	Migrate to managed identity, federation, or STS; rotate remaining credentials.
Failed authorization	Are deny events expected negative tests, broken deployments, or attack signals?	Correlate principal, action, scope, token audience, and change history.
Role and trust changes	Who changed a role definition, trust policy, group membership, or assignment?	Require review, alert on privileged changes, retain correlation IDs.
Data-plane access	Can a management reader also read business data through another role or key?	Enumerate DataActions, service-native ACLs, and shared keys.
Orphaned principals	Is a managed identity or service principal still attached to active resources and owned by a team?	Remove attachments, assignments, credentials, and the principal in order.
Break-glass readiness	Can the emergency identity authenticate, and will its use trigger immediate evidence and response?	Test under procedure; protect credentials; alert on every use.

5.9.1 Identity SLOs and policy objectives

Identity governance can be expressed as measurable objectives. Examples include: 100% of production workloads use managed or federated identity where supported; 0 long-lived cloud credentials in source repositories; 100% of privileged human roles are eligible or

time-bound; 100% of role assignments have owner and purpose metadata; authorization-denial tests run on every policy change; and orphaned identities are removed within a defined retirement window.

These objectives should not become vanity metrics. “No secrets” is meaningless if one broad managed identity serves every workload. “All admins use PIM” is incomplete if they activate Owner instead of a job-function role. Pair identity metrics with scope, action, and evidence quality.

5.10 Lab 3: Create and Verify a Read-Only Service Identity

This independent lab proves three ideas. First, the workload can authenticate without a client secret. Second, a Reader role assignment at resource-group scope permits management-plane inventory. Third, the same identity is denied when it attempts a management write. The lab uses a user-assigned managed identity attached to a short-lived Azure Container Instance running the Azure CLI image. [12] [13]

Lab 3: prove read-only authority with a credential-free workload

Prepare	Admin creates a temporary resource group, user-assigned managed identity, and container group. Assign Reader at the test scope.
Allow	Sign in from the container using the attached identity. List resources successfully.
Deny	Attempt to update resource-group tags with the same identity. Capture the denied management write.

Evidence: role assignment, container logs, successful list output, failed write output, Activity Log event, and verified cleanup.

Figure 5-6 The lab attaches a read-only managed identity to a temporary container and verifies both allow and deny paths.

Table 5-9 Lab 3 at a glance.

Item	Value
Primary capability	Workload identity, Azure RBAC scope, least-privilege verification.
Azure resources	One resource group, one user-assigned managed identity, one role assignment, one temporary container group.
Cost class	Low. Managed identity and RBAC have no direct charge; Container Instances runs for only a few minutes.
Required evidence	Identity resource ID and principal ID; Reader assignment; successful resource list; denied resource-group update; container logs; cleanup result.
Required privilege for setup	Permission to create the resources and Microsoft.Authorization/roleAssignments/write at the lab scope, such as an appropriate access-administration role.
Cleanup boundary	Delete the dedicated resource group and verify it no longer exists.

5.10.1 Prerequisites and safety boundary

- Azure CLI installed, authenticated with az login, and pointed to a sandbox subscription.
- Permission to create a resource group, user-assigned managed identity, role assignment, and Azure Container Instance.
- A region that supports Azure Container Instances and user-assigned managed identity.
- A shell with openssl for generating a short suffix, or replace the suffix command with another unique value.
- No production subscription. The lab intentionally generates a failed authorization event.
- Authorization changes can take time to propagate. If the first container attempt cannot read the resource group, wait briefly and rerun the container step rather than broadening the role.

The lab uses only the Azure management plane. Reader does not grant blob, database, secret, search-document, or model-inference access. Section 5.10.8 provides an optional extension for proving the control/data-plane distinction.

5.10.2 Set variables and create the isolated scope

```
suffix=$(openssl rand -hex 3)
rg="id-lab-$suffix"
location="eastus"
identity="id-reader-$suffix"
```

```
runner="id-runner-$suffix"
subscription_id=$(az account show --query id --output tsv)

az group create \
  --name "$rg" \
  --location "$location" \
  --tags purpose=identity-lab lifecycle=ephemeral owner=reader

rg_id="/subscriptions/$subscription_id/resourceGroups/$rg"
```

Use a region appropriate to the subscription. The dedicated resource group is both the authorization scope and the cleanup boundary. Its tags provide ownership and lifecycle intent before any workload is created.

5.10.3 Create a user-assigned managed identity

```
az identity create \
  --resource-group "$rg" \
  --name "$identity"

identity_id=$(az identity show \
  --resource-group "$rg" \
  --name "$identity" \
  --query id --output tsv)

principal_id=$(az identity show \
  --resource-group "$rg" \
  --name "$identity" \
  --query principalId --output tsv)

client_id=$(az identity show \
  --resource-group "$rg" \
  --name "$identity" \
  --query clientId --output tsv)

printf "identity_id=%s\nprincipal_id=%s\nclient_id=%s\n" \
  "$identity_id" "$principal_id" "$client_id"
```

Record all three identifiers. The Azure resource ID addresses the managed-identity resource. The principal ID is the Microsoft Entra object used in the RBAC assignment. The client ID selects the user-assigned identity when the container requests a token. Confusing these identifiers is a common cause of authentication and role-assignment errors.

5.10.4 Assign Reader at resource-group scope

```
assignment_id=$(az role assignment create \
  --assignee-object-id "$principal_id" \
  --assignee-principal-type ServicePrincipal \
  --role Reader \
  --scope "$rg_id" \
  --query id --output tsv)

echo "$assignment_id"

az role assignment list \
  --assignee-object-id "$principal_id" \
  --scope "$rg_id" \
  --include-inherited \
  --query "[].{role:roleDefinitionName,scope:scope,principal:principalId}" \
  --output table
```

The principal type is ServicePrincipal because a managed identity is represented by a special service principal in Microsoft Entra ID. The assignment is intentionally made at the dedicated resource group, not at subscription scope. Reader allows management-plane read operations and excludes management writes.

5.10.5 Run the workload without a client secret

The container receives the user-assigned managed identity as a resource attachment. Environment variables carry only nonsecret identifiers. The command signs in with the managed identity, lists resources, and then attempts to update the resource-group tags. The expected write failure is converted into a successful lab exit so the container can preserve its logs.

```
command="/bin/sh -c '
  set -eu;
  az login --identity --client-id \$CLIENT_ID --output none;
  az account set --subscription \$SUBSCRIPTION_ID;
  echo ==== READ_TEST ====;
  az resource list --resource-group \$RESOURCE_GROUP \
    --query "[].{name:name,type:type}" --output table;
  echo ==== WRITE_TEST ====;
  if az group update --name \$RESOURCE_GROUP \
       --set tags.writeTest=unexpected; then
    echo UNEXPECTED_WRITE_SUCCEEDED;
    exit 42;
  else
    echo EXPECTED_WRITE_DENIED;
  fi
```

```
'"

az container create \
  --resource-group "$rg" \
  --name "$runner" \
  --image mcr.microsoft.com/azure-cli \
  --assign-identity "$identity_id" \
  --restart-policy Never \
  --os-type Linux \
  --cpu 1 \
  --memory 1 \
  --environment-variables \
      CLIENT_ID="$client_id" \
      SUBSCRIPTION_ID="$subscription_id" \
      RESOURCE_GROUP="$rg" \
  --command-line "$command"
```

The Azure CLI image is convenient for a control-plane lab; an application would normally use Azure.Identity or MSAL rather than run az login. The security property is the same: the workload receives a token through the attached managed identity, and no client secret is provided to the container.

5.10.6 Collect the allow and deny evidence

```
for attempt in $(seq 1 60); do
  state=$(az container show \
    --resource-group "$rg" \
    --name "$runner" \
    --query "containers[0].instanceView.currentState.state" \
    --output tsv)
  [ "$state" = "Terminated" ] && break
  sleep 5
done

[ "$state" = "Terminated" ] || {
  echo "Container did not terminate within five minutes." >&2
  exit 1
}

az container logs \
  --resource-group "$rg" \
  --name "$runner"
```

The logs should show a table of resources under READ_TEST and EXPECTED_WRITE_DENIED under WRITE_TEST. The list normally includes the user-assigned identity and the container group. The exact AuthorizationFailed text can vary, but

it should identify the managed-identity principal and the denied resource-group write action or operation.

Inspect recent failed activity events:

```
az monitor activity-log list \
  --resource-group "$rg" \
  --offset 2h \
  --max-events 50 \
  --query "[?status.value=='Failed'].
{time:eventTimestamp,caller:caller,operation:operationName.localizedValue,status
:status.localizedValue,resource:resourceId,correlation:correlationId}" \
  --output table
```

Activity-log availability and event detail can lag or differ by provider operation. The primary proof is the workload log plus the role assignment. When a failed event is present, retain the caller, operation, resource ID, timestamp, and correlation ID as authorization evidence.

5.10.7 Verify the least-privilege contract

Table 5-10 Lab evidence checklist.

Check	Expected result	Failure interpretation
Managed identity exists	Resource type is Microsoft.ManagedIdentity/userAssignedIdentities; IDs are recorded.	The principal cannot be addressed or attached.
Role assignment scope	Reader is assigned to the managed-identity principal at exactly the lab resource group.	A parent-scope assignment may be broader; a child scope may be insufficient.
Credential path	The container uses az login --identity --client-id; no password, certificate, or secret is supplied.	A copied credential defeats the lab objective.
Positive authorization	The identity lists management-plane resources in the lab group.	Check role propagation, token audience, subscription, and identity attachment.
Negative authorization	Resource-group tag update fails with authorization denial.	A successful write indicates an inherited or additional role; investigate effective access.
Accountability	Container log and, when available, Activity Log identify operation and principal.	Without evidence the allow/deny decision is difficult to audit.

Check	Expected result	Failure interpretation
Cleanup	The resource group is deleted and az group exists returns false.	Residual compute or identity can retain cost or authority.

5.10.8 Optional extension: prove that Reader is not data read

Create a storage account and blob container as the administrator. Keep the managed identity assigned only Reader at the resource-group scope. From the container, an authenticated command can read the storage-account resource properties, but az storage container list --auth-mode login should be denied because Reader has no blob DataActions. Assign Storage Blob Data Reader at the storage-account scope, wait for propagation, and rerun the data-plane command. It should then succeed without granting any management write. Remove the data role after the test.

This extension demonstrates why permission reviews must enumerate both Actions and DataActions. It also shows a safe troubleshooting rule: when a data request fails, add the specific data role at the narrow resource scope instead of upgrading the workload to Contributor.

5.10.9 Clean up and verify retirement

```
az group delete \
  --name "$rg" \
  --yes \
  --no-wait

az group wait --name "$rg" --deleted
az group exists --name "$rg"
```

The final command should return false. Because the managed identity, role assignment, and container group are contained in the dedicated resource group, the group provides one reviewed retirement operation. In a production system, remove assignments and detach shared user-assigned identities before deleting the identity resource, and preserve audit evidence according to policy.

5.10.10 Multi-cloud translation

Table 5-11 Repeating Lab 3 by engineering intent.

Provider	Credential-free or temporary identity path	Read-only test	Denied test
Azure	Attach a user-assigned managed identity to supported compute and obtain a Microsoft Entra token.	List resources at a dedicated resource-group scope with Reader.	Attempt a management write such as tag update.
Alibaba Cloud	Attach an instance or Function Compute RAM role, or assume a RAM role for an STS token.	Use a read-only RAM policy scoped to a dedicated resource group or resource ARN where supported.	Attempt create/update/delete outside the allowed actions.
Tencent Cloud	Assume a CAM role for STS credentials or use supported OIDC workload integration.	Use a read-only CAM policy over a dedicated project/resource scope.	Attempt an API operation excluded by the policy.
Volcengine	Assume an IAM role for temporary credentials or use the specific service identity integration when available.	Attach a read-only IAM policy at the narrowest supported TRN/project scope.	Attempt a write and capture the IAM/audit denial.

The result should be compared by evidence class, not by console similarity: named principal, temporary authentication, policy or role, exact scope, successful read, failed write, audit event, and verified cleanup. A missing managed-workload path or resource-level authorization boundary is a meaningful provider or service difference to document.

5.11 Handoff to CloudDesk FinOps Agent

CloudDesk FinOps Agent will later need identities that read cost and resource metadata, identities that produce recommendations, and narrowly scoped executors for approved optimization actions. This chapter does not implement CloudDesk's Tenant Registry, cross-tenant isolation, response binding, desktop approval, or custom Serverless runtime. Those belong to Serverless AI Agent Development.

The reusable output is a set of governed capability contracts:

```
identity.leastPrivilege.create(task, principalType, resourceScope, lifetime)
  -> principalId, roleOrPolicyId, assignmentId, credentialPath
  evidence: positiveTest, negativeTest, assignmentScope, reviewer
```

```
identity.permission.audit(principalId, scope)
  -> effectiveManagementActions, effectiveDataActions, inheritedAssignments,
conditions
  evidence: sourceAssignments, lastUsed, privilegedFindings, recommendedRemovals

secret.reference.resolve(secretRef, workloadIdentity)
  -> versionedSecretValue available only to the authorized workload
  evidence: vaultId, secretVersion, retrievalPrincipal, correlationId
```

A future Agent may request one of these capabilities, but it must not receive the permission-administration identity itself. The CloudDesk control plane should create or review authority through an approved workflow, while tool execution uses the resulting narrow identity.

5.12 Key Takeaways

Identity is the primary cloud control-plane boundary because every management and data operation arrives from a principal.

Authentication, authorization, credentials, tokens, delegation, and evidence are distinct engineering objects.

Least privilege is an envelope across principal, actions, scope, conditions, and time—not a safe-sounding role name.

Human administrators, deployment pipelines, ingestion workers, retrieval applications, operators, and Agent tool executors should not share one identity.

Azure RBAC binds a principal to a role definition at a scope; permissions inherit downward and selected conditions can narrow an assignment.

Management Actions and data-plane DataActions must be reviewed separately. Reader is not universal data read.

Managed identities and workload identity federation remove stored cloud credentials but do not remove the need for narrow roles and exact trust policy.

Secret managers protect unavoidable secrets; they should not become the default authentication strategy for cloud workloads.

A least-privilege design is verified only when required actions succeed, forbidden actions fail, and both results produce accountable evidence.

Multicloud identity comparison should translate principal, trust, temporary credential, policy, scope, secret, and audit intent while documenting non-equivalence.

References and Official Documentation

Sources were verified on August 24, 2026. Built-in role definitions, supported identity integrations, token limits, product names, and service-specific authorization granularity should be rechecked at manuscript freeze.

[1] NIST Computer Security Resource Center. Least Privilege glossary and NIST SP 800-53 Rev. 5, control AC-6.

[2] Microsoft Learn. What is Azure role-based access control (Azure RBAC)? Security principals, role definitions, assignments, and scope.

[3] Microsoft Learn. Understand Azure role definitions. Actions, NotActions, DataActions, and NotDataActions.

[4] Microsoft Learn. Application and service principal objects in Microsoft Entra ID.

[5] Microsoft Learn. Managed identities for Azure resources. System-assigned and user-assigned identity lifecycle and token acquisition.

[6] Microsoft Learn. Workload identity federation concepts. External OIDC trust and secretless token exchange.

[7] Microsoft Learn. Understand scope for Azure RBAC and What is Azure ABAC? Scope inheritance and role-assignment conditions.

[8] Microsoft Learn. What is Privileged Identity Management? Just-in-time, time-bound, approved, and audited privileged access.

[9] Microsoft Learn. Best practices for Azure RBAC. Remove unnecessary privilege and use narrow scopes.

[10] Microsoft Learn. Grant permission to applications to access an Azure key vault using Azure RBAC. Control-plane and data-plane separation.

[11] Microsoft Learn. Overview of Azure Policy. Relationship between RBAC authorization and compliant resource state.

[12] Microsoft Learn. How to use managed identities with Azure Container Instances.

[13] Microsoft Learn. Sign into Azure using a managed identity and Azure CLI.

[14] Alibaba Cloud Documentation. What is RAM? RAM users, groups, roles, policies, and STS tokens.

[15] Alibaba Cloud Documentation. Best practices for identity and access control. Prioritize temporary credentials and least privilege.

[16] Alibaba Cloud Documentation. KMS Credential Manager and secure KMS access using RAM roles.

[17] Tencent Cloud Documentation. CAM role and STS temporary credential concepts; AssumeRole API.

[18] Tencent Cloud Documentation. Security Practical Tutorial and access-key recommendations for least privilege and identity separation.

[19] Tencent Cloud Documentation. Secrets Manager overview, CAM integration, CloudAudit, and rotation use cases.

[20] Volcengine Documentation. IAM basic concepts and role management. Users, groups, roles, trust, and temporary credentials.

[21] Volcengine Documentation. Policy overview, resource/TRN syntax, and Access Key practice recommendations.

[22] IETF RFC 6749. The OAuth 2.0 Authorization Framework.

[23] OpenID Foundation. OpenID Connect Core 1.0 incorporating errata set 2.

Part II / Chapter 6

Compute, Elasticity, and Reliability

How Workload Shape, Failure Domains, and Capacity Contracts Turn Virtual Machines into Replaceable Cloud Resources

A cloud virtual machine is not merely a remote server with a power button. It is a governed resource graph connected to an image, disks, interfaces, placement choices, health signals, scaling rules, quotas, recovery mechanisms, and billing commitments. This chapter develops the control-plane reasoning required to select, scale, replace, protect, and cost compute without confusing an instance with the service it supports.

In This Chapter

- Why a compute instance is a managed resource contract and dependency graph rather than a CPU-and-memory purchase
- How to describe workload shape before selecting an instance family or size
- How hard constraints, benchmarking evidence, regional availability, and lifecycle status narrow the candidate set
- Why elasticity is a feedback controller with delays, health semantics, cooldown, and termination policy
- How vertical scaling, horizontal scaling, scheduled scaling, and interruptible capacity solve different problems
- Why reliability claims must name the failure domain they address
- When to replace, repair, recover, restore, or fail over a compute resource
- How quota, stock, placement, image availability, and specialized hardware create capacity risk independently of price
- How pay-as-you-go, Spot capacity, reservations, savings plans, and capacity assurance form different contracts
- How telemetry and business context support safe right-sizing decisions
- How Azure, Alibaba Cloud, Tencent Cloud, and Volcengine express the same engineering intents with different resource models

- How Lab 5 produces a compute dependency graph, runtime proof, reliability statement, and monthly cost estimate without depending on any other lab

A portal makes compute selection look deceptively small. Choose a region, an image, an instance size, a disk, and perhaps a public address; then click Create. Yet every selection binds the workload to a physical architecture, a fault domain, a network path, an operational model, a capacity pool, and a billing contract. The virtual machine resource may be created in minutes, but the consequences of an unsuitable selection can persist for years through operating-system images, data placement, monitoring assumptions, reserved commitments, and recovery procedures.

The course material behind this book described a “real VM dependency” that included the subscription or account, resource group or project, region and zone, virtual network, subnet, security policy, image, size, addresses, keys, disks, monitoring, backup, and tags. It also framed IaaS as an engineering loop: plan, configure, secure, observe, protect, recover, and optimize. This chapter retains that practical viewpoint but converts it into a reusable compute control-plane model.

The central claim is simple: compute selection is a decision about workload shape, failure domain, elasticity, operational responsibility, and cost - not merely a decision about vCPU and memory. A service is not reliable because one virtual machine is powerful. It becomes reliable when the architecture can detect failure, maintain or recreate capacity, preserve the right state, and continue to satisfy a defined service objective. Similarly, compute is not elastic because an instance can be resized. It is elastic when a governed controller can add, remove, or replace capacity fast enough to preserve the intended outcome without creating oscillation, corruption, or uncontrolled cost.

Virtual machines remain the clearest reference object because their dependencies are visible. The same reasoning later applies to Kubernetes node pools, machine-learning training clusters, GPU inference pools, hosted runners, and Agent tool workers. In each case, the platform does not eliminate physical scarcity. It exposes scarcity through a programmable resource contract.

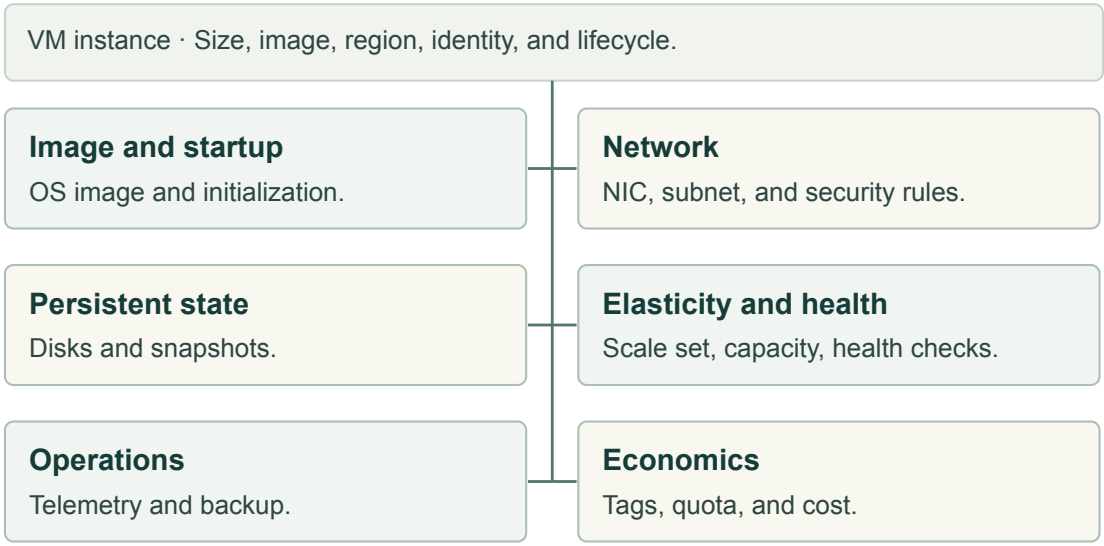


Figure 6-1 A virtual machine is only the center of a larger compute resource graph. Reliability, security, observability, and cost attach to the graph, not to the guest operating system alone.

6.1 Compute Is a Managed Resource Contract

A physical server is primarily a piece of equipment. A cloud compute resource is a named contract through which the control plane allocates equipment, records desired state, exposes lifecycle operations, attaches policy, and emits evidence. The guest operating system and applications run in the data plane, but the instance itself exists because the provider maintains control-plane state for it.

That distinction explains why a VM can be stopped, deallocated, resized, reimaged, redeployed, moved into a scale group, backed up, protected by a lock, tagged for cost allocation, or deleted while its disks follow a different lifecycle. It also explains why an operating-system shutdown is not necessarily equivalent to releasing the underlying compute allocation. Control-plane state and guest state can diverge.

Contract element	Question the architecture must answer	Typical evidence
Identity and scope	What is the durable resource ID, parent scope, region, zone, owner, and workload context?	Resource inventory, tags, activity history

Contract element	Question the architecture must answer	Typical evidence
Hardware profile	What architecture, vCPU, memory, accelerator, local storage, and network limits are requested?	Instance type/size and effective configuration
Software origin	Which image, generation, bootstrap declaration, extensions, and patch policy create the node?	Image reference, image digest/version, deployment record
Dependencies	Which disks, NICs, subnets, security controls, load balancers, identities, and secrets are attached?	Resource graph and dependency IDs
Placement and failure domain	Where can correlated failures occur, and how is capacity distributed?	Zone, fault-domain, host-group, scale-group configuration
Lifecycle	How is the instance created, started, stopped, updated, replaced, protected, and retired?	Control-plane operations and provisioning state
Health and elasticity	Who declares an instance healthy, ready, removable, or replaceable?	Health probes, autoscale policy, lifecycle hooks
Economics	What is metered, what remains charged while stopped, and which commitments or discounts apply?	Rate assumptions, billing state, utilization and commitment data

The resource graph matters because different nodes fail and retire independently. Deleting a VM may or may not delete its NIC and disks, depending on the provider and the configured delete semantics. Replacing an instance can preserve a data disk but invalidate a local temporary disk. Moving a workload to another zone can change available sizes, network paths, and data-transfer costs. A reliability or cleanup review that considers only the VM ID is therefore incomplete.

Control-plane rule. Never describe a compute resource only by its friendly name. Operational evidence should include the provider resource ID, region, zone, instance type, image reference, dependent resource IDs, billing mode, and current lifecycle state.

6.2 Start with Workload Shape, Not an Instance Catalog

Instance catalogs are organized around provider hardware generations and marketing families. Workloads are organized around bottlenecks, latency objectives, failure tolerance, and duty cycles. The selection process should therefore begin with workload evidence and only later translate that evidence into provider-specific families.

A useful workload profile records more than average utilization. It captures the shape of demand: whether CPU is steady or bursty; whether memory working sets fit predictably; whether storage access is sequential or random; whether network packet rate matters more than aggregate bandwidth; whether an accelerator is continuously occupied; and whether the workload can checkpoint, restart, or move. The profile also records the consequence of under-provisioning. A batch job that finishes two hours later and an interactive API that violates a 200-millisecond latency objective are not equivalent failures.

Dimension	Questions	Selection consequence
CPU	Sustained or bursty? Single-threaded or parallel? Vector instructions? Full physical cores required?	General-purpose, burstable, or compute-optimized family; core count and generation
Memory	Working-set size? Cache sensitivity? NUMA behavior? Growth pattern?	Memory ratio, memory-optimized family, scale-up limit
Storage I/O	Random or sequential? IOPS, throughput, latency, queue depth? Durable or temporary?	Disk class, number of disks, local NVMe, striping, data placement
Network	Bandwidth, packet rate, east-west traffic, latency, accelerated networking, RDMA?	NIC limits, instance family, placement, HPC fabric
Accelerator	GPU/FPGA/other accelerator type, memory, interconnect, driver and framework compatibility?	Specialized family, image, quota, topology, regional stock
Duty cycle	Continuous baseline, daily schedule, short burst, batch window, event-driven demand?	Pay-as-you-go, schedule, scale group, commitment, interruptible capacity
State	Can the process restart? Where is authoritative state? What must survive replacement?	Stateless pool, attached durable disk, replication, checkpointing
Reliability	What failure domain must the service survive, and what is the recovery objective?	Replica count, zone distribution, backup and failover design
Compliance	Required architecture, confidential-compute features, tenancy, locality, or approved images?	Hard exclusion of otherwise attractive families

The common family labels - general purpose, compute optimized, memory optimized, storage optimized, GPU accelerated, and high-performance computing - are useful only at this level of abstraction. The exact family names, processor generations, accelerator models,

and availability change frequently. They belong in a current selection record, not in a permanent architectural assumption.

Burstable instances deserve special care. They can be excellent for low-baseline services with occasional peaks, development environments, jump hosts, and low-traffic control services. They are poor choices when sustained CPU demand exceeds the baseline or when a performance objective cannot tolerate credit depletion. The apparent hourly price must be evaluated together with the credit model and the workload duty cycle.

Specialized AI and HPC workloads add topology constraints. GPU count is not enough. Accelerator memory, host memory, CPU-to-GPU ratio, local storage, driver image, collective-communication performance, RDMA or InfiniBand availability, quota, and stock can dominate the design. For these workloads, the smallest unit of selection may be a multi-accelerator node or a tightly placed cluster rather than an individual virtual machine.

Select compute by eliminating invalid choices before optimizing price

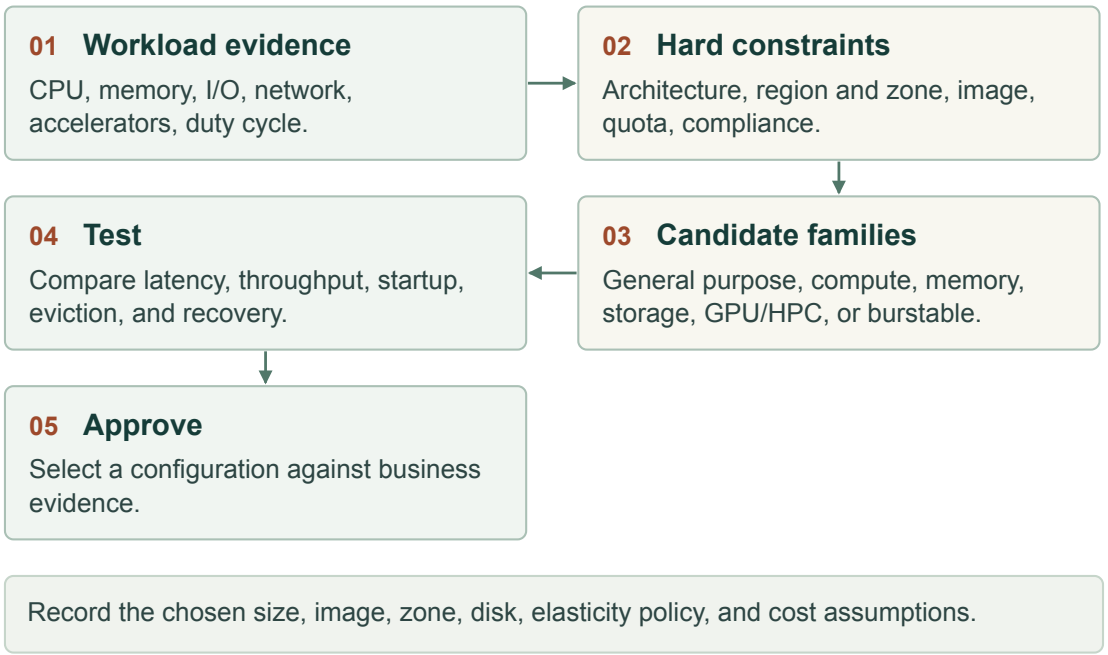


Figure 6-2 A defensible compute selection eliminates candidates that violate hard constraints before comparing price or benchmark results.

6.3 Instance Selection Is a Constraint-Satisfaction Problem

Let S be the set of instance types advertised for a provider. The architecture should first construct a feasible set F by applying hard constraints. A simplified expression is:

```
F = { s in S | architecture(s) = requiredArchitecture
             and regionAvailable(s)
             and quotaAvailable(s)
             and memory(s) >= requiredMemory
             and network(s) >= requiredNetwork
             and acceleratorCompatible(s)
             and policyApproved(s) }
```

Only after F has been established should the team rank candidates using soft criteria such as estimated cost, startup time, energy efficiency, benchmark throughput, operational familiarity, or commitment coverage. A weighted score can support discussion, but the weights must be explicit:

```
score(s) = w1 * normalizedCost(s)
         + w2 * normalizedLatency(s)
         + w3 * normalizedRecoveryTime(s)
         + w4 * normalizedOperationalRisk(s)
```

This is not an optimization algorithm that eliminates engineering judgment. It is a way to prevent an attractive price from hiding a violated constraint. A candidate that lacks the required image, zone stock, accelerator quota, network feature, or compliance status is not "second best"; it is infeasible.

Decision record field	Why it matters
Workload and service objective	Connects the compute choice to an outcome rather than a catalog item.
Hard constraints	Documents why certain architectures, regions, families, or billing modes were excluded.
Candidate families and generations	Prevents accidental dependence on one aging or retiring SKU.
Benchmark method and dataset	Makes performance claims reproducible and distinguishes synthetic from representative load.
Failure and recovery test	Shows whether restart, replacement, checkpoint, or zone loss is acceptable.

Decision record field	Why it matters
Cost assumptions	Records hours, utilization, disks, licenses, network, monitoring, backup, and commitment coverage.
Exit path	States how the workload can move when the size, image, region, or generation is retired.

Benchmarking should be representative, bounded, and comparative. It should include startup and recovery where those affect the service. A compute-optimized instance that completes a job faster can cost less per result even when its hourly rate is higher. Conversely, a large instance can hide a concurrency bottleneck that would be better addressed through horizontal partitioning. The metric of interest is often cost per successful outcome, not cost per hour.

The instance lifecycle also belongs in the decision. Providers introduce new generations and retire old ones. Images and extensions age. A long-lived workload needs an upgrade path that can be rehearsed before the provider forces migration. Recording a family rather than a single size, and separating image construction from instance creation, reduces the cost of that change.

6.4 Elasticity Is a Feedback Loop

Elasticity is the ability to change allocated capacity in response to demand or policy. Scaling is the action; elasticity is the managed relationship among demand, measurement, decision, provisioning, readiness, and removal. A system that can add instances but cannot make them ready before the backlog violates its SLO is technically scalable and operationally inelastic.

Vertical scaling changes the capacity of an instance. Horizontal scaling changes the number of instances. Vertical scaling is often straightforward for stateful or license-bound software, but it may require stopping or replacing the VM and it eventually reaches a family limit. Horizontal scaling usually offers better fault tolerance and incremental growth, but it requires partitionable work, externalized state, load distribution, and safe instance removal.

Elasticity mode	Trigger	Strength	Primary risk
Manual vertical	Operator changes size	Simple, useful for known step changes	Downtime, resize availability, over-provisioning
Manual horizontal	Operator changes desired count	Explicit and easy to reason about	Slow response and human dependency

Elasticity mode	Trigger	Strength	Primary risk
Scheduled	Calendar or known business event	Predictable and cheap to operate	Forecast error and exceptional demand
Metric-based	CPU, memory, latency, queue length, custom metric	Responds to measured demand	Lag, noise, oscillation, wrong signal
Event-based	Messages, jobs, partitions, requests, external events	Directly reflects queued work	Poison work, duplicate events, drain semantics
Predictive	Forecast model estimates future demand	Can provision before demand arrives	Model error, governance, unexplained action

A simple controller might estimate a target instance count as:

```
target = clamp(minInstances,
               ceil(observedDemand / effectiveCapacityPerReadyInstance),
               maxInstances)
```

Real controllers require more. They account for measurement windows, boot time, initialization, health probes, connection draining, cooldown, minimum residency, failure replacement, and provider API delays. They use hysteresis so that a workload near a threshold does not repeatedly scale out and in. They also distinguish desired count from ready count. During a zone shortage or image failure, desired capacity can remain unmet even while the scaling policy is correct.

Elasticity is a feedback controller, not a synonym for “more instances”

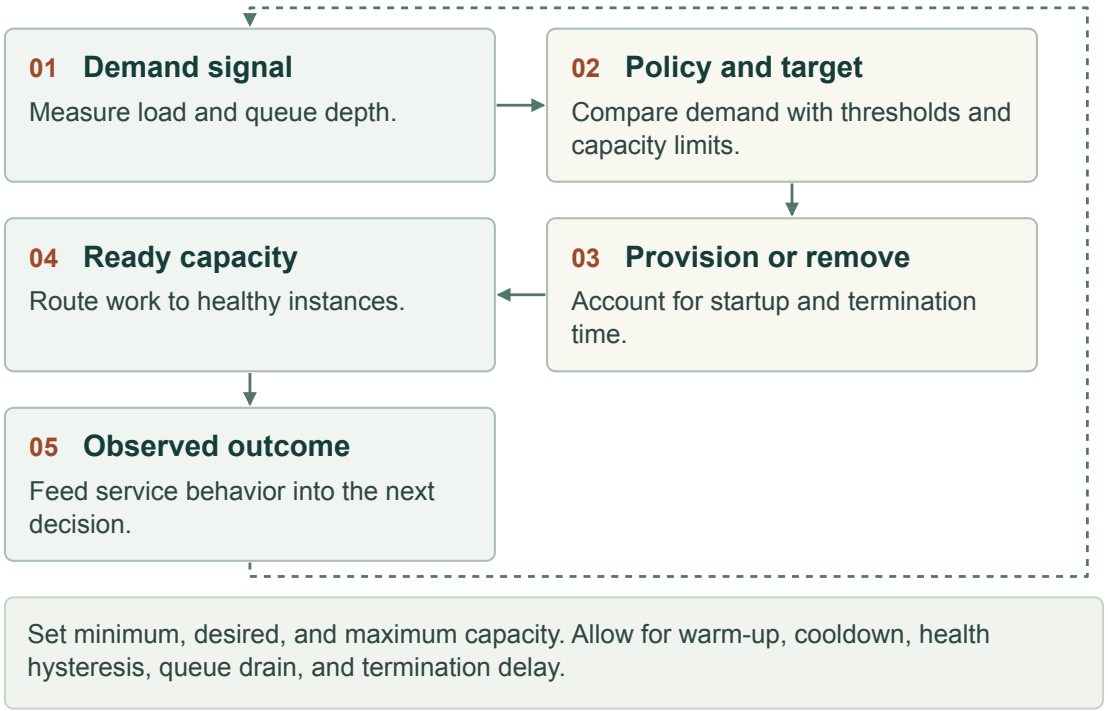


Figure 6-3 Elasticity is a delayed feedback loop. Stable operation requires explicit readiness, warm-up, cooldown, health, and termination semantics.

Horizontal elasticity depends on an instance contract that makes nodes replaceable. Bootstrap must be idempotent. Configuration should be externally versioned. Health probes should reflect whether the instance can serve work, not merely whether the process exists. Stateful connections and queued work need a drain procedure. Data must live in a system whose durability and replication model survives instance replacement.

Autoscaling does not remove capacity planning. Minimum capacity establishes the reliability and latency baseline. Maximum capacity constrains cost and provider quota exposure. Default or desired capacity determines the starting point after deployment. A scaling group is therefore simultaneously a reliability object, an economic guardrail, and a change controller.

Elasticity anti-pattern. Scaling on average CPU alone can be dangerously misleading. A service can have low average CPU while a single partition, dependency, lock, disk queue, memory limit, or tail-latency path is saturated.

6.5 Reliability Must Name the Failure Domain

“Highly available” is not a complete statement. The architecture must state which failures it is intended to tolerate: a process crash, a VM failure, host maintenance, a rack or fault-domain outage, a zone outage, a regional outage, or the loss of a shared dependency such as identity, DNS, storage, or the deployment control plane.

Availability sets and equivalent fault-domain constructs reduce correlated host or rack failures inside a region. Availability zones address a larger failure domain by placing replicas in physically separate datacenter groups within the same region. A scale group can distribute instances across zones or fault domains, but a scale group by itself does not make an application zone-resilient. The load balancer, state store, image source, identity path, DNS, monitoring, and deployment process must also survive the named failure.

Failure event	Compute control	Additional dependency that must be checked
Process crash	Service supervisor, restart policy, health probe	Startup idempotence and local corruption
VM or host loss	Multiple instances, replacement controller	Load balancing, externalized state, readiness
Rack/fault-domain event	Platform spread or availability set	Shared network and storage paths
Zone outage	Replicas across zones, zone-capable scale group	Zone-redundant data, load balancer, DNS, capacity
Regional outage	Cross-region deployment and failover	Replication lag, identity, traffic steering, runbook
Capacity shortage	Multiple sizes/zones, reserved capacity, fallback policy	Quota, image availability, deployment automation
Operator or deployment error	Versioned image, staged rollout, rollback	Independent evidence and approval path
Shared control-plane dependency	Preplanned degraded mode and recovery path	Alternative identity, DNS, secrets, and operations access

For independent identical replicas with individual availability a, a simplified model gives service availability of 1 - (1 - a)^n when any one of n replicas can serve the request. The formula is useful for intuition and dangerous as a guarantee. Cloud replicas are rarely independent. They can share a zone, load balancer, storage account, certificate, identity provider, deployment error, or data corruption event. Reliability engineering is largely the work of finding and breaking these hidden correlations.

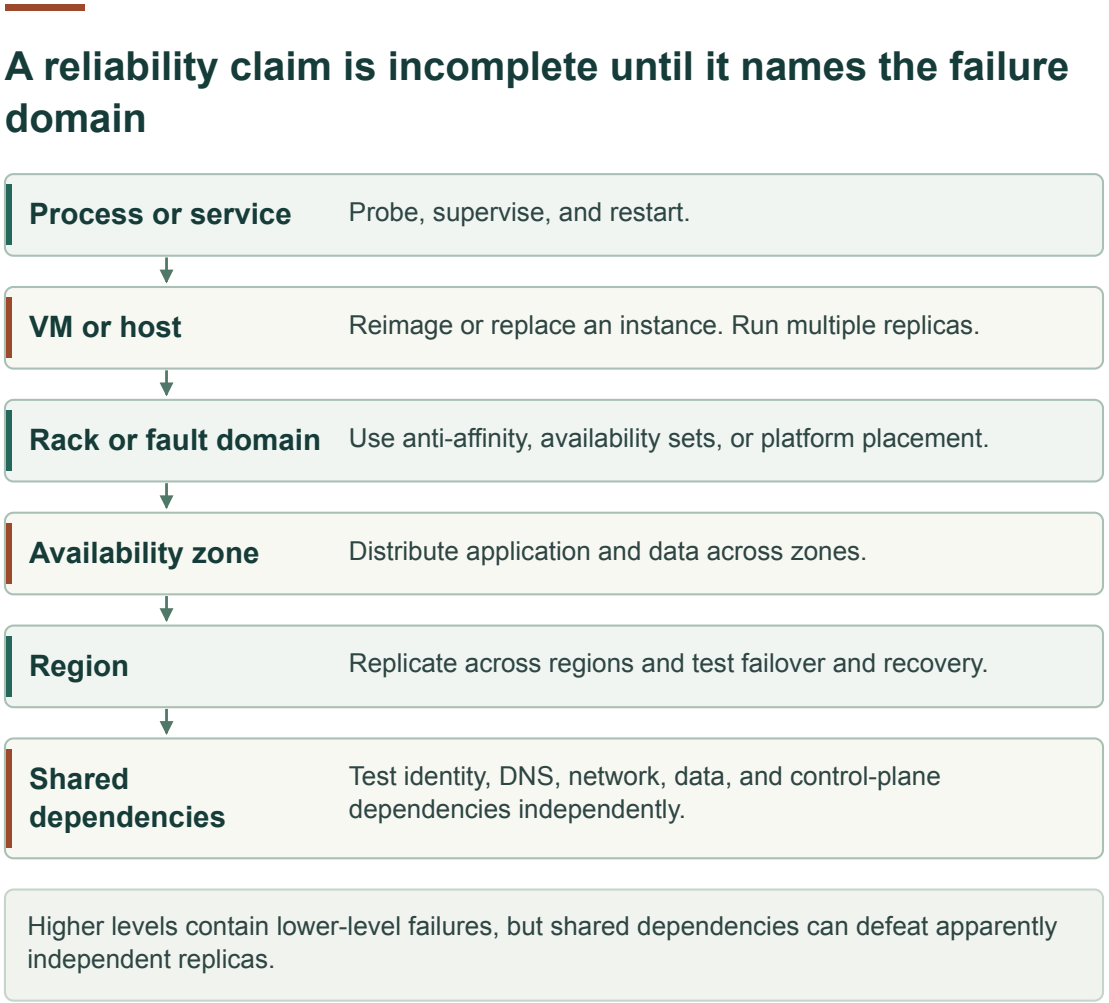


Figure 6-4 Reliability controls must be matched to the failure domain. Adding replicas at one level does not automatically protect higher levels or shared dependencies.

The design should also distinguish availability from durability. Two stateless web VMs across zones may continue serving while one zone is unavailable, but they do not protect the authoritative database. Three copies of a disk do not guarantee application-level consistency. Backup protects recoverable history; replication supports continuity; neither replaces the other.

6.6 Replace, Repair, and Recover Preserve Different State

Cloud-native guidance often prefers replacement over repair. The preference is valuable because replacement exercises the declared image and bootstrap path, reduces configuration drift, and fits automated scale groups. It is not a universal rule. The correct response

depends on which state must be preserved and how quickly service must return.

Artifact or mechanism	What it preserves	What it does not prove
Image	Operating-system and software baseline	Current data, runtime state, or successful bootstrap
IaC declaration	Desired resource graph and configuration parameters	Guest data, application correctness, or provider capacity
Snapshot	Point-in-time block-storage state	Application consistency unless coordinated
Backup	Recoverable protected history with retention policy	Immediate availability or current state
Replication	A current or near-current copy for continuity	Protection from logical corruption without history
Scale group model	Repeatable instance configuration and desired count	Stateful service quorum or data safety
Runbook and recovery test	Operational ability to execute recovery	Future success unless retested and evidence remains current

Replacement is appropriate when authoritative state lives elsewhere, initialization is deterministic, and the service can tolerate the replacement interval. Repair is appropriate when local state, licensed configuration, forensic evidence, or a long rebuild time makes replacement more disruptive. Recovery is required when the service or data has already crossed an availability or integrity boundary and must be restored, failed over, or rebuilt from protected state.

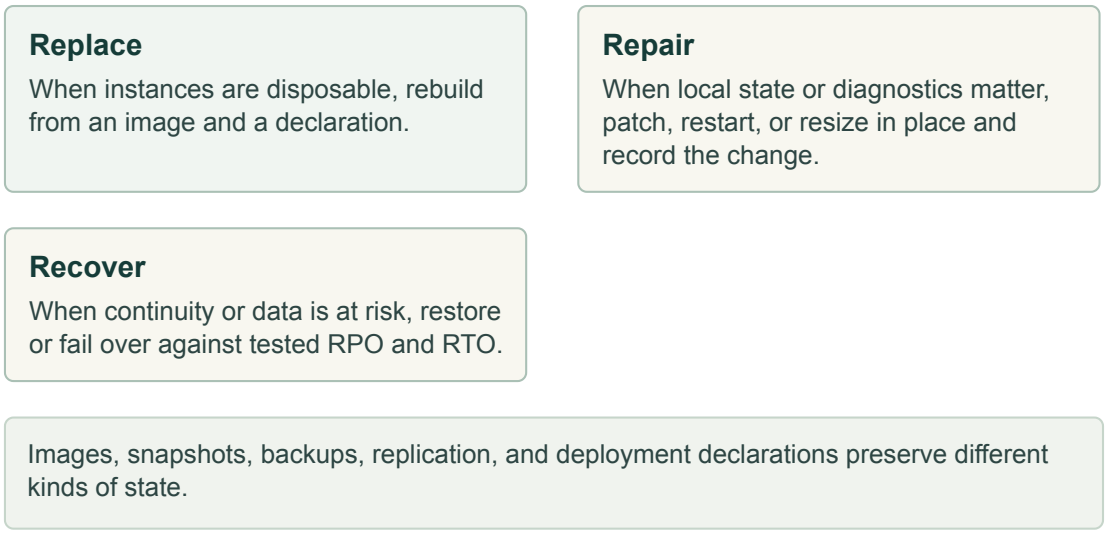


Figure 6-5 Replace, repair, and recover are different operations because they preserve different state and satisfy different recovery objectives.

A reliable replaceable-instance design uses a versioned image, declarative configuration, workload identity, externalized secrets, health checks, graceful termination, and a tested provisioning path. It also records which resources should be deleted with the instance and which should be retained. Without those semantics, "replaceable" can mean "easy to lose data."

Backup and disaster recovery should be tied to RPO and RTO. The recovery point objective states how much data loss is tolerable. The recovery time objective states how long restoration may take. A daily backup can be excellent protection against accidental deletion and still be unsuitable for a service requiring minutes of data loss and minutes of downtime. High availability, backup, and disaster recovery solve different temporal problems.

Reliability rule. Do not count a backup as a replica, a replica as a backup, or a snapshot as a tested recovery procedure.

6.7 Capacity, Quota, and Placement Are Separate Constraints

A valid deployment can fail even when the template, identity, and budget are correct. Compute depends on physical stock. The requested size might not exist in the selected zone, the subscription quota might be insufficient, the image might not support the archi-

tecture, or a specialized accelerator might be temporarily unavailable. These are capacity failures, not syntax errors.

Quota is permission to request up to a limit. Capacity is the provider's ability to fulfill the request at a specific time and placement. Reservation and savings commitments are primarily billing contracts. Capacity reservations, elasticity assurances, dedicated hosts, or provider-specific private capacity pools are different mechanisms intended to improve fulfillment certainty. The names and guarantees vary, so the design must record the actual contract rather than infer it from the word “reserved.”

Constraint	Typical symptom	Mitigation
Subscription or project quota	Deployment rejected before allocation	Request quota early; monitor headroom; use policy-approved fallback size
Regional or zonal stock	Allocation failure or scale group below desired count	Use multiple zones/sizes; capacity assurance; staggered rollout
Image availability	Chosen image or generation unavailable in target region/architecture	Replicate or publish approved images; validate before rollout
Specialized hardware scarcity	GPU/HPC capacity unavailable or long wait	Reserve capacity; broaden topology; checkpoint jobs; queue work
Placement constraint	Anti-affinity, proximity, dedicated host, or network feature cannot be satisfied	Model placement as a hard constraint and test realistic scale
Provisioning time	Capacity exists but is not ready before demand arrives	Warm pool, scheduled scale-out, smaller images, prevalidated bootstrap

Capacity headroom should be expressed at the level that matters. If C_ready is ready service capacity and C_required is demand at the service objective, then headroom H = C_ready - C_required. A positive H in one zone is not regional resilience. A positive vCPU quota does not prove that the required accelerator is available. Capacity dashboards need to separate quota, desired capacity, allocated capacity, ready capacity, and service throughput.

Scale groups can improve fulfillment by allowing multiple zones, size priorities, or a mixture of on-demand and interruptible instances. That flexibility is valuable only if the application profile supports the alternatives. Mixing sizes can change per-instance capacity and termination behavior. Mixing architectures or accelerator generations can change images, drivers, numerical behavior, and performance. Fallbacks should therefore be explicit approved profiles, not arbitrary substitutions.

6.8 Build a Capacity Portfolio, Not a Single Billing Choice

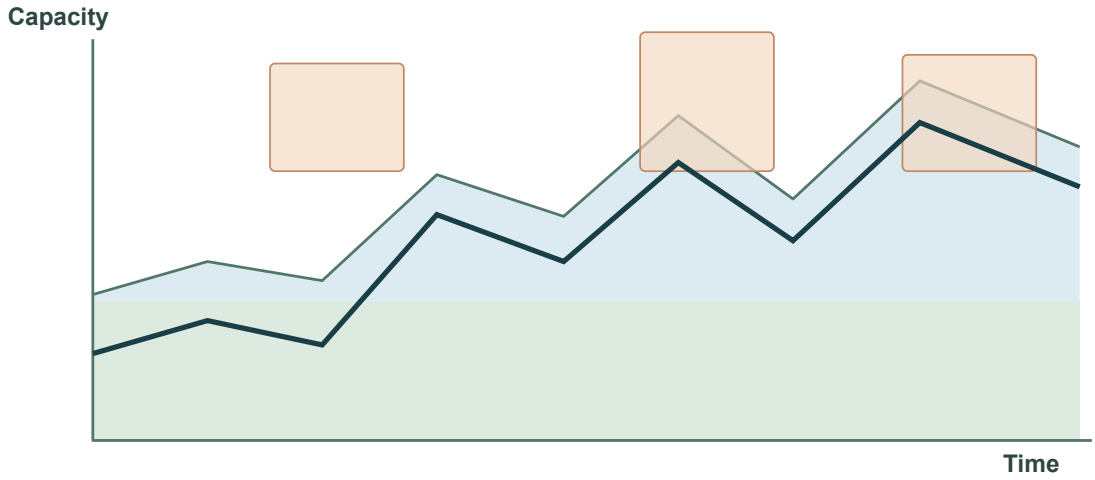
Compute economics is a portfolio problem. Different portions of demand have different certainty and interruption tolerance. A stable baseline may justify a commitment. Variable but essential demand needs on-demand elasticity. Fault-tolerant batch or stateless burst capacity may use Spot or other interruptible markets. Specialized or regulated workloads may require dedicated capacity even when it is more expensive.

Capacity contract	Best fit	What it does not guarantee
Pay-as-you-go / on-demand	Uncertain, short-lived, evolving, or burst demand	Lowest price, future stock, or automatic elasticity
Spot / interruptible	Checkpointable batch, disposable workers, tolerant distributed jobs	Availability, continuity, or stable capacity
Reservation	Predictable matching usage where a specific commitment yields a billing discount	A running instance, high availability, or capacity unless explicitly included
Savings plan / spend commitment	Evolving compute usage needing broader discount flexibility	Unlimited discount coverage or capacity assurance
Capacity reservation / assurance	Known placement and scale where allocation risk matters	Application readiness, data resilience, or low utilization cost
Dedicated host / isolated hardware	Licensing, compliance, tenancy, or placement constraints	Elastic efficiency or automatic service reliability

Interruptible capacity should be treated as a failure mode that happens during normal operation. Workers must checkpoint or make work idempotent. The queue must redeliver safely. The scale group must replace capacity. The service should remain within its SLO when a realistic fraction of Spot instances disappear. Provider eviction notices are useful optimization signals, not reliability guarantees.

Commitments should follow measured baseline demand, not optimistic forecasts. A reservation that matches the wrong region, family, or operating system can be underutilized. A savings plan can be more flexible but still creates a spend obligation. The FinOps control plane described in Chapter 14 should track coverage, utilization, expiration, and the relationship between commitments and approved workload profiles.

A production capacity portfolio separates baseline, burst, and interruption tolerance



Committed or reserved	Baseline capacity
On demand	Elastic capacity
Spot or interruptible	Work that tolerates eviction and retry

The dark line shows illustrative demand. Compare reliable service costs, including interruptions and recovery.

Figure 6-6 Baseline commitments, on-demand elasticity, and interruptible bursts solve different economic and reliability problems.

6.9 Observability Turns Right-Sizing into an Evidence-Based Change

Right-sizing is often presented as a simple response to low CPU utilization. Safe right-sizing is a change decision supported by multiple signals and a representative observation window. It must account for memory, disk, network, tail latency, batch deadlines, seasonal peaks, recovery time, and the workload's position in a larger service graph.

Evidence	Questions before a size change
CPU distribution	Are averages hiding short saturation, single-thread limits, or burst-credit exhaustion?
Memory working set	Is memory stable, growing, reclaimable, or close to an out-of-memory boundary?

Evidence	Questions before a size change
Disk and local storage	Are IOPS, throughput, latency, or queue depth tied to the VM size or disk class?
Network	Will the smaller size reduce bandwidth, packet rate, NIC count, or accelerated-network support?
Application SLI	What happens to p95/p99 latency, error rate, throughput, and job completion time?
Elasticity	Would fewer larger instances or more smaller instances improve recovery and cost?
Failure behavior	Does the candidate size exist in every required zone and recover fast enough?
Economic context	Are license, disk, data transfer, commitments, and operational effort included?

A useful efficiency metric relates resource cost to the delivered outcome, for example cost per successful request, cost per completed training epoch, or cost per document processed. Utilization is an input, not the objective. Very high utilization can reduce resilience and increase latency; very low utilization can be intentional standby capacity or a symptom of oversized resources. The target is enough headroom to meet the SLO at an acceptable unit cost.

Provider recommendation systems such as Azure Advisor can identify underutilized VMs and scale sets by analyzing telemetry. Recommendations are evidence, not authority. They should be reviewed against business cycles, failure-domain requirements, capacity availability, and rollback readiness. Automated execution should begin with read-only inventory and recommendation capabilities, followed by a reviewed change plan.

Compute state also affects billing. A guest operating-system shutdown may leave the resource allocated. A deallocated VM can stop compute charges while disks, addresses, backup, monitoring, and other dependencies continue to incur cost. The evidence packet should therefore record both operational state and billing state.

6.10 Azure Reference Control Plane

Azure exposes compute through standard Virtual Machine resources and Virtual Machine Scale Sets. A VM resource refers to a hardware profile, storage profile, operating-system profile, network interfaces, identity, zones, diagnostics, and extensions. The attached disks and NICs remain separately addressable resources. This makes Azure Resource Graph well suited to inventorying the dependency graph.

Virtual Machine Scale Sets add a group-level desired state, instance orchestration, scaling, health, upgrade, and placement semantics. At the time of writing, Microsoft recommends Flexible orchestration for most new scale-set deployments. Flexible mode manages standard VM resources and can support mixed sizes and combinations of Spot and on-demand capacity; Uniform mode emphasizes a homogeneous scale-set model. This recommendation and the feature matrix are volatile facts and should be rechecked before publication or implementation.

Azure object	Control-plane role	Reliability or elasticity significance
Microsoft.Compute/virtualMachines	Individual compute instance with hardware, image, disk, NIC, identity, and lifecycle state	Basic replacement unit; not a service-level HA design by itself
Microsoft.Compute/virtualMachineScaleSets	Group desired state, orchestration, scaling, health, placement, and upgrades	Maintains or changes capacity and can spread instances across failure domains
Microsoft.Compute/disks	Durable OS or data block-storage resource	Independent data lifecycle; snapshot/backup considerations
Microsoft.Network/networkInterfaces	Data-plane attachment to subnet, IP configuration, and security controls	Dependency that must follow replacement and placement design
Availability zone / availability set	Placement and correlated-failure control	Defines which infrastructure failures replicas are intended to tolerate
Azure Monitor / VM Insights / alerts	Telemetry, health, diagnostics, and decision signals	Supports failure detection, scaling, and right-sizing evidence
Azure Backup / Recovery Services vault	Protected recovery points and restore workflow	Recovery capability, not an availability replica
Azure Advisor / Cost Management	Recommendations, rates, commitments, and cost evidence	Supports right-sizing and economic portfolio decisions

Availability zones are the preferred mechanism when the region and workload support them and zone-level resilience is required. Availability sets remain useful in regions or scenarios without zone support, but they address a smaller failure domain. Zone-spanning scale sets can distribute instances across zones; the application and its data still need zone-aware design.

Azure Spot VMs expose unused capacity at a discount and can be evicted when Azure needs the capacity. They have no availability guarantee and require interruption-tolerant workloads. Azure Reservations and savings plans are billing constructs with different specificity and flexibility. Neither should be treated as proof of high availability or as a substitute for capacity planning.

Azure Backup provides isolated recovery points for VM data, while VM images and scale-set models provide reproducibility. The operations team should test both replacement and restore. Being able to redeploy an empty VM does not prove that the service data can be recovered; being able to restore a disk does not prove that the current application version will start correctly.

6.11 Multicloud Translation by Engineering Intent

Azure, Alibaba Cloud, Tencent Cloud, and Volcengine all expose virtual-machine resources, images, disks, zones, autoscaling groups, monitoring, and multiple billing modes. The shared engineering intent is strong; the operational equivalence is not automatic. Instance families, quota models, capacity assurance, Spot interruption behavior, shutdown billing, image catalogs, scale-group policies, and zone coverage differ.

Provider	Compute and dependencies	Elasticity and failure domains	Economic and capacity controls
Azure	Virtual Machines, images, managed disks, NICs, VM extensions	VM Scale Sets, availability zones, availability sets, health and autoscale policies	Pay-as-you-go, Spot VMs, reservations, savings plans, capacity reservation, Advisor
Alibaba Cloud	ECS instances, images, Block Storage, ENIs, security groups	Auto Scaling (ESS), multi-zone placement, auto provisioning, elasticity assurance	Subscription, pay-as-you-go, Spot, reserved instances, savings plans, capacity-oriented assurances
Tencent Cloud	CVM instances, images, cloud disks, ENIs, security groups	Auto Scaling groups and launch configurations, availability zones, CLB integration	Yearly/monthly, pay-as-you-go, Spot, reserved-instance discount, quota and stock constraints
Volcengine	ECS instances, images, disks, network interfaces, projects and tags	Auto Scaling groups, rules, cooldown, lifecycle hooks, regional and zonal placement	Pay-as-you-go and Spot-oriented capacity, provider-specific scaling and inventory behavior

A multicloud compute profile should therefore record intent before provider syntax: required architecture, resource ratios, accelerator type, disk and network performance, failure domain, scaling mode, interruption tolerance, recovery objective, and economic contract. Each provider adapter then maps the profile to approved instance families and reports non-equivalence explicitly.

For example, “two general-purpose instances” is not a portable reliability design. The two sizes may have different CPU generations, network baselines, disk limits, availability-zone coverage, startup times, or Spot behavior. Likewise, a provider's savings product may ap-

ply broadly by spend while another provider's reservation matches a narrower set of attributes. The translation must preserve the engineering intent, not the product label.

6.12 Lab 5 - Build a Compute Resource Dependency Graph and Monthly Cost Estimate

This independent lab creates one small Linux VM in Azure without a public IP address, proves that the instance is running through a control-plane command, queries its dependency graph, and produces a documented monthly retail-cost estimate. The lab does not claim high availability. Its reliability outcome is an explicit statement of which failure domains are and are not addressed.

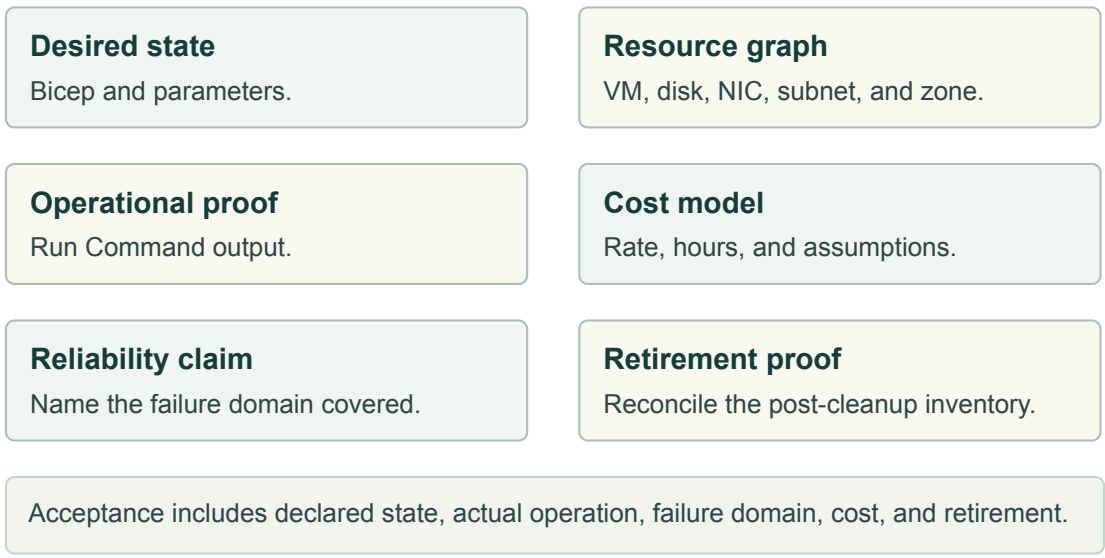


Figure 6-7 The lab is complete only when declared state, runtime proof, dependency inventory, cost assumptions, reliability scope, and cleanup evidence are collected.

6.12.1 Capability Goal and Acceptance Criteria

The capability goal is to understand a VM as a governed resource graph and to separate runtime success, reliability, and cost evidence. The lab is successful when all of the following are true:

- The VM and its network and disk dependencies are created from declared state in an isolated resource group.

- The VM has no public IP address and no inbound Internet rule; no SSH session is required.
- Azure Run Command proves that the guest operating system started and can execute a bounded script.
- Azure Resource Graph returns the VM size, zone if present, OS disk ID, NIC ID, subnet ID, private IP address, and tags.
- A monthly estimate lists compute hours, disk charges, and every explicit exclusion or uncertainty.
- The reliability statement correctly says that a single VM is not zone-resilient or service-level highly available.
- The resource group is deleted and a final query proves that no lab resources remain.

6.12.2 Prerequisites and Cost Boundary

Item	Requirement
Azure access	Permission to create a resource group, VNet, subnet, NSG, NIC, managed disk, and VM, and to run Resource Graph queries.
Tools	Azure CLI, Bicep, a shell, and Python 3 or a spreadsheet for the estimate.
SSH key	A public key is supplied to the VM, but the lab does not create a public route for SSH.
Region and size	Choose a region and a low-cost Linux general-purpose or burstable size currently available to the subscription. The example uses Standard_B1s only as a replaceable default.
Cost class	Low if the VM is deleted promptly. Compute, OS disk, and any retained dependencies can continue to incur charges until deletion.
Out of scope	Guest hardening, application benchmarking, multi-zone architecture, backup, and public ingress.

Volatile fact. Instance availability, quota, images, and retail prices vary by region and time. Treat the example size and API versions as lab defaults to verify, not permanent book facts.

6.12.3 Declare the Minimal Resource Graph

Use a new directory and save the following as main.bicep. The template intentionally omits a public IP and inbound security rules. It enables managed boot diagnostics and requests deletion of the NIC and OS disk with the VM. The full lab repository should pin tested API versions and expose the image reference as parameters.

```
param location string = resourceGroup().location
param prefix string
param vmSize string = 'Standard_B1s'
param adminUsername string = 'azureuser'
@secure()
param sshPublicKey string

var tags = {
  workload: 'compute-lab'
  environment: 'lab'
  owner: 'book-reader'
  costCenter: 'learning'
  expiresOn: '2026-08-26'
}

resource vnet 'Microsoft.Network/virtualNetworks@2024-05-01' = {
  name: '${prefix}-vnet'
  location: location
  tags: tags
  properties: {
    addressSpace: { addressPrefixes: [ '10.42.0.0/16' ] }
  }
}

resource subnet 'Microsoft.Network/virtualNetworks/subnets@2024-05-01' = {
  parent: vnet
  name: 'compute'
  properties: { addressPrefix: '10.42.1.0/24' }
}

resource nsg 'Microsoft.Network/networkSecurityGroups@2024-05-01' = {
  name: '${prefix}-nsg'
  location: location
  tags: tags
  properties: { securityRules: [] }
}

resource nic 'Microsoft.Network/networkInterfaces@2024-05-01' = {
  name: '${prefix}-nic'
  location: location
  tags: tags
  properties: {
    networkSecurityGroup: { id: nsg.id }
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
```

```
          subnet: { id: subnet.id }
        }
      }
    ]
  }
}

resource vm 'Microsoft.Compute/virtualMachines@2024-11-01' = {
  name: '${prefix}-vm'
  location: location
  tags: tags
  properties: {
    hardwareProfile: { vmSize: vmSize }
    osProfile: {
      computerName: '${prefix}-vm'
      adminUsername: adminUsername
      linuxConfiguration: {
        disablePasswordAuthentication: true
        ssh: { publicKeys: [ { path:
'/home/${adminUsername}/.ssh/authorized_keys', keyData: sshPublicKey } ] }
      }
    }
    storageProfile: {
      imageReference: {
        publisher: 'Canonical'
        offer: 'ubuntu-24_04-lts'
        sku: 'server'
        version: 'latest'
      }
      osDisk: {
        name: '${prefix}-osdisk'
        createOption: 'FromImage'
        deleteOption: 'Delete'
        managedDisk: { storageAccountType: 'Standard_LRS' }
      }
    }
    networkProfile: {
      networkInterfaces: [ { id: nic.id, properties: { deleteOption: 'Delete' }
} ]
    }
    diagnosticsProfile: { bootDiagnostics: { enabled: true } }
  }
}

output vmId string = vm.id
output nicId string = nic.id
output osDiskName string = vm.properties.storageProfile.osDisk.name
```

Verify the image and size before deployment. The following commands are examples; select a current image and a size that is available in your region and quota:

```
LOCATION=eastus2
VM_SIZE=Standard_B1s

az vm image list \
  --location "$LOCATION" \
  --publisher Canonical \
  --offer ubuntu-24_04-lts \
  --all --output table

az vm list-skus \
  --location "$LOCATION" \
  --size "$VM_SIZE" \
  --resource-type virtualMachines \
  --all --output table
```

6.12.4 Deploy and Prove Runtime Readiness

```
RG=book-ch06-compute
LOCATION=eastus2
PREFIX=ch06demo
SSH_KEY_FILE="$HOME/.ssh/id_ed25519.pub"

az group create --name "$RG" --location "$LOCATION" \
  --tags workload=compute-lab environment=lab owner=book-reader
costCenter=learning

az deployment group create \
  --resource-group "$RG" \
  --template-file main.bicep \
  --parameters prefix="$PREFIX" vmSize=Standard_B1s \
               sshPublicKey="$(cat "$SSH_KEY_FILE")"

az vm run-command invoke \
  --resource-group "$RG" \
  --name "${PREFIX}-vm" \
  --command-id RunShellScript \
  --scripts 'uname -a; uptime; echo compute-lab-ready' \
  --output json > runtime-proof.json
```

Run Command is a bounded control-plane operation and avoids creating a public management path solely for the experiment. It is not a substitute for a production access design. Save the complete JSON response because it proves that deployment success progressed to guest execution.

6.12.5 Query the Dependency Graph

Install the Resource Graph extension if needed, then run a query that joins the VM with its disk and NIC. Save both JSON and CSV forms. Resource Graph schemas can evolve, so inspect the returned properties when adapting the query.

```
az extension add --name resource-graph --upgrade

QUERY=$(cat <<'KQL'
let vm = Resources
| where type =~ 'microsoft.compute/virtualmachines'
| where resourceGroup =~ 'book-ch06-compute'
| extend vmSize=tostring(properties.hardwareProfile.vmSize),
         osDiskId=tostring(properties.storageProfile.osDisk.managedDisk.id),
         nicId=tostring(properties.networkProfile.networkInterfaces[0].id),
         zone=tostring(zones[0])
| project vmId=id, vmName=name, location, zone, vmSize, osDiskId, nicId, tags;
let disks = Resources
| where type =~ 'microsoft.compute/disks'
| project osDiskId=id, diskName=name,
          diskSku=tostring(sku.name), diskSizeGB=toint(properties.diskSizeGB);
let nics = Resources
| where type =~ 'microsoft.network/networkinterfaces'
| project nicId=id, nicName=name,

privateIp=tostring(properties.ipConfigurations[0].properties.privateIPAddress),

subnetId=tostring(properties.ipConfigurations[0].properties.subnet.id);
vm
| join kind=leftouter disks on osDiskId
| join kind=leftouter nics on nicId
| project vmName, location, zone, vmSize, diskName, diskSku,
          diskSizeGB, nicName, privateIp, subnetId, tags
KQL
)

az graph query --graph-query "$QUERY" --output json > compute-graph.json
az graph query --graph-query "$QUERY" --output table
```

The result should make the dependency graph visible. The VM name alone is insufficient evidence; the output should contain the effective size, location, zone if any, managed-disk properties, NIC, private address, subnet, and tags. If the zone field is empty, state that the VM is regional or nonzonal rather than inventing zone resilience.

6.12.6 Produce the Monthly Retail-Cost Estimate

Use 730 hours as a transparent monthly planning convention unless the workload schedule specifies otherwise. Retrieve the current pay-as-you-go Linux compute meter from the Azure Retail Prices API or verify it in the Azure Pricing Calculator. The API can return multiple meters, including Windows, Spot, low-priority, or secondary records; inspect the candidates rather than accepting the first response.

```
REGION=eastus2
SKU=Standard_B1s

curl -sG 'https://prices.azure.com/api/retail/prices' \
  --data-urlencode "api-version=2023-01-01-preview" \
  --data-urlencode "\$filter=serviceName eq 'Virtual Machines' and armRegionName
eq '${REGION}' and armSkuName eq '${SKU}' and priceType eq 'Consumption'" \
  > retail-price-candidates.json

python - <<'PY'
import json
with open('retail-price-candidates.json') as f:
    data=json.load(f)
for x in data.get('Items', []):
    print(x.get('productName'), x.get('skuName'), x.get('meterName'),
          x.get('unitOfMeasure'), x.get('retailPrice'), x.get('currencyCode'))
PY
```

Create a worksheet rather than a single number. The estimate should show assumptions and exclusions:

Line item	Quantity assumption	Rate source	Monthly estimate
Linux VM compute	1 instance x 730 active hours	Current reviewed retail meter or pricing calculator	hourly rate x 730
Managed OS disk	Provisioned disk class and size	Current managed-disk price	monthly disk rate
Public IP	None in this lab	Resource graph proves absence	0
Outbound data transfer	Assume no material egress for control-plane verification	Explicit assumption; not measured	0 for lab estimate
Monitoring and logs	Managed boot diagnostics only; no Log Analytics workspace created	Explicit scope	0 or current included behavior

Line item	Quantity assumption	Rate source	Monthly estimate
Backup	Not enabled	Explicit scope	0
Support, tax, negotiated discount	Excluded	Retail estimate limitation	not included

The total is a planning estimate, not an invoice forecast. It should include the date, currency, region, operating system, meter name, hours, disk class, and any offer-specific assumptions. If the resource will run only during a class or test window, calculate a scheduled estimate as well as the 730-hour upper bound.

6.12.7 State the Reliability Boundary

Record the following conclusion in reliability.md:

```
The lab deploys one VM in one region with no explicit multi-zone replica.
It can demonstrate instance creation and guest readiness, but it does not
survive VM, host, zone, or regional loss as a service. Managed disk durability
and boot diagnostics do not make the application highly available. A
production design would require multiple ready instances, load distribution,
zone-aware placement where required, externalized or replicated state, and a
tested recovery path.
```

This statement is an important deliverable. It prevents a successful deployment from being mislabeled as a reliable architecture.

6.12.8 Clean Up and Prove Zero Remainder

```
az group delete --name "$RG" --yes --no-wait
az group wait --name "$RG" --deleted

az graph query \
  --graph-query "Resources | where resourceGroup =~ '$RG' | project id, type,
name" \
  --output json > cleanup-proof.json
```

The final query should return no resources. Review the subscription for any resource created outside the resource group, and retain the deletion operation ID or activity evidence. Zero-remainder verification is part of the acceptance criteria because abandoned disks, snapshots, public addresses, backup vault items, or monitoring destinations can outlive the VM and continue to incur cost.

6.12.9 Evidence Packet

Artifact	Purpose
main.bicep and parameter record	Declared desired state and reproducibility
deployment-output.json	Control-plane creation evidence and resource IDs
runtime-proof.json	Guest startup and bounded command execution
compute-graph.json / CSV	VM, disk, NIC, subnet, zone, and tag dependency inventory
retail-price-candidates.json	Dated source data for the selected retail meter
monthly-estimate.csv	Transparent quantities, rates, exclusions, and total
reliability.md	Named failure domain and explicit non-guarantees
cleanup-proof.json	Evidence that the lab graph no longer exists

6.12.10 Common Failure Modes

Failure	Likely cause	Correct response
SKU not available	Region/zone stock, quota, subscription restriction, or retired/unsupported size	Query current SKUs, inspect restrictions, choose an approved fallback, and record the change.
Image not found	Offer/SKU changed or unavailable in region or architecture	List current images and update the parameterized reference.
Run Command fails	Guest agent not ready, extension failure, or provisioning still in progress	Inspect instance view and boot diagnostics; do not open public SSH as the first workaround.
Resource Graph join is empty	Property path or API projection changed, or resource creation is not yet indexed	Query each resource type separately and inspect actual properties.
Price API returns many rows	Multiple operating systems, meters, Spot or savings records, or duplicate tiers	Review product, meter, unit, region, and price type; cross-check with calculator.
Cleanup leaves charges	Dependent resource retained outside group or soft-deleted/protected item persists	Inventory by tags and resource IDs; follow service-specific purge or retention procedures.

6.12.11 Chinese-Cloud Translation

Repeat the intent, not the Azure syntax. On Alibaba Cloud, create a small ECS instance with its image, system disk, ENI/VSwitch, security group, project/resource group context, and tags; query the ECS and related resources; use the price calculator or current billing data; then release the instance and verify dependent resources. On Tencent Cloud, use CVM, cloud disks, VPC/subnet, security groups, project and tags, and record the billing mode and shutdown behavior. On Volcengine, use ECS, the relevant VPC and disk resources, project and tag context, and its Auto Scaling or instance APIs for inventory. In every provider, state the actual failure domain and prove cleanup.

6.13 CloudDesk Reuse: Inventory, Right-Sizing, and Failure-Domain Explanation

CloudDesk FinOps Agent can later reuse the chapter's read-only reasoning without receiving unrestricted compute administration. The first capability inventories the full dependency graph. The second produces a right-sizing recommendation from approved evidence. The third explains what a proposed architecture can and cannot survive.

```
capability: compute.vm.inventory
risk_level: R0_read_only
inputs:
  provider: azure
  scope_id: /subscriptions/.../resourceGroups/...
outputs:
  instances:
    - resource_id
    - size
    - image
    - zone
    - disk_ids
    - nic_ids
    - billing_state
    - tags
  evidence_time
```

```
capability: compute.vm.rightsize.recommend
risk_level: R1_read_only_analysis
inputs:
  instance_id: /subscriptions/.../virtualMachines/...
  observation_window: 30d
  service_objective_id: api-prod-slo
  approved_profiles: [general-small-v2, general-medium-v2]
outputs:
```

```
  current_profile
  candidate_profile
  evidence_summary
  expected_unit_cost_change
  capacity_and_zone_constraints
  rollback_requirements
  recommendation_confidence
```

```
capability: compute.failureDomain.explain
risk_level: R0_read_only
inputs:
  workload_scope: /subscriptions/.../resourceGroups/...
outputs:
  protected_failure_domains
  unprotected_failure_domains
  shared_dependencies
  recovery_assumptions
  evidence_links
```

A future mutating capability such as compute.vm.resize should be separate. It should require an approved plan, maintenance or rolling-update semantics, a concurrency check, capacity validation, rollback or replacement strategy, and post-change SLO verification. The Agent should never be given a generic shell or unrestricted cloud CLI merely because a right-sizing recommendation exists.

6.14 Key Takeaways

- A cloud compute resource is a named control-plane contract linked to images, disks, network interfaces, placement, health, lifecycle, evidence, and economics.
- Workload shape - not provider catalog order - should determine the candidate instance families.
- Hard constraints create the feasible set; benchmark, recovery, and cost evidence rank feasible candidates.
- Elasticity is a delayed feedback loop. Desired capacity, allocated capacity, ready capacity, and delivered service capacity are different states.
- Vertical, horizontal, scheduled, metric-based, event-based, and predictive scaling solve different problems and create different failure modes.
- A reliability claim must name the failure domain. Replicas can still share common dependencies and common-mode failures.
- Replacement, repair, recovery, restore, and failover preserve different state and must not be used as synonyms.

- Quota, provider stock, placement, image support, and specialized-hardware availability are independent capacity constraints.
- Spot capacity, reservations, savings plans, and capacity assurance are different contracts. Price discounts do not create high availability.
- Right-sizing requires representative telemetry, application SLOs, failure-domain constraints, and a verified rollback path.
- Azure VM and VM Scale Set resources provide the reference model; Alibaba Cloud ECS/ESS, Tencent Cloud CVM/AS, and Volcengine ECS/AS implement comparable intent with non-equivalent details.
- Lab 5 proves that a running VM, a dependency graph, a cost estimate, a reliability claim, and cleanup evidence are separate deliverables.
- CloudDesk may reuse bounded inventory, recommendation, and explanation capabilities, but compute mutation requires stronger approval and verification controls.

References and Official Documentation

[1] National Institute of Standards and Technology. The NIST Definition of Cloud Computing. NIST SP 800-145, 2011. https://doi.org/10.6028/NIST.SP.800-145

[2] Microsoft. Sizes for virtual machines in Azure. Microsoft Learn. Updated March 9, 2026. https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/overview

[3] Microsoft. Availability options for Azure Virtual Machines. Microsoft Learn. Updated December 5, 2025. https://learn.microsoft.com/en-us/azure/virtual-machines/availability

[4] Microsoft. What are Azure Availability Zones? Microsoft Learn. Updated June 25, 2026. https://learn.microsoft.com/en-us/azure/reliability/availability-zones-overview

[5] Microsoft. Availability sets overview. Microsoft Learn. Updated April 6, 2026. https://learn.microsoft.com/en-us/azure/virtual-machines/availability-set-overview

[6] Microsoft. Reliability in Azure Virtual Machines. Microsoft Learn. Updated January 22, 2026. https://learn.microsoft.com/en-us/azure/reliability/reliability-virtual-machines

[7] Microsoft. Azure Virtual Machine Scale Sets overview. Microsoft Learn. Updated May 19, 2026. https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/overview

[8] Microsoft. Orchestration modes for Virtual Machine Scale Sets in Azure. Microsoft Learn. Updated May 19, 2026. https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-orchestration-modes

[9] Microsoft. Create an Azure scale set that uses availability zones. Microsoft Learn. Updated May 26, 2026. https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/virtual-machine-scale-sets-use-availability-zones

[10] Microsoft. Architecture best practices for Azure Virtual Machines and scale sets. Azure Well-Architected Framework. https://learn.microsoft.com/en-us/azure/well-architected/service-guides/virtual-machines

[11] Microsoft. Azure Virtual Machines baseline architecture. Azure Architecture Center. https://learn.microsoft.com/en-us/azure/architecture/virtual-machines/baseline

[12] Microsoft. About Azure Spot Virtual Machines. Microsoft Learn. Updated February 6, 2026. https://learn.microsoft.com/en-us/azure/virtual-machines/spot-vms

[13] Microsoft. Build workloads with Azure Spot Virtual Machines. Azure Architecture Center. https://learn.microsoft.com/en-us/azure/architecture/guide/spot/spot-eviction

[14] Microsoft. Create a scale set that uses Azure Spot Virtual Machines. Microsoft Learn. Updated May 19, 2026. https://learn.microsoft.com/en-us/azure/virtual-machine-scale-sets/use-spot

[15] Microsoft. Decide between a savings plan and a reservation. Microsoft Learn. Updated March 14, 2026. https://learn.microsoft.com/en-us/azure/cost-management-billing/savings-plan/decide-between-savings-plan-reservation

[16] Microsoft. What are Azure Reservations? Microsoft Learn. Updated July 17, 2026. https://learn.microsoft.com/en-us/azure/cost-management-billing/reservations/save-compute-costs-reservations

[17] Microsoft. Azure Retail Prices REST API overview. Microsoft Learn. https://learn.microsoft.com/en-us/rest/api/cost-management/retail-prices/azure-retail-prices

[18] Microsoft. Azure Resource Graph sample queries for Azure Virtual Machines. Updated August 22, 2024. https://learn.microsoft.com/en-us/azure/virtual-machines/resource-graph-samples

[19] Microsoft. Explore your Azure resources with Resource Graph. Microsoft Learn. Updated June 3, 2024. https://learn.microsoft.com/en-us/azure/governance/resource-graph/concepts/explore-resources

[20] Microsoft. Optimize virtual machines or scale sets with Azure Advisor cost recommendations. Updated February 18, 2026. https://learn.microsoft.com/en-us/azure/advisor/advisor-cost-recommendations

[21] Microsoft. Cost recommendations - Azure Advisor. https://learn.microsoft.com/en-us/azure/advisor/advisor-reference-cost-recommendations

[22] Microsoft. Reliability recommendations - Azure Advisor. Updated August 11, 2026. https://learn.microsoft.com/en-us/azure/advisor/advisor-reference-reliability-recommendations

[23] Microsoft. About Azure VM backup. Updated January 8, 2026. https://learn.microsoft.com/en-us/azure/backup/backup-azure-vms-introduction

[24] Microsoft. Back up an Azure VM from VM settings. Updated September 24, 2025. https://learn.microsoft.com/en-us/azure/backup/backup-azure-vms-first-look-arm

[25] Microsoft. States and billing status for Azure Virtual Machines. Microsoft Learn. https://learn.microsoft.com/en-us/azure/virtual-machines/states-billing

[26] Microsoft. Retired Azure VM size series. Updated July 16, 2026. https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/lifecycle/retired-sizes-list

[27] Alibaba Cloud. What is Elastic Compute Service (ECS)? Updated April 15, 2026. https://www.alibabacloud.com/help/en/ecs/user-guide/what-is-ecs

[28] Alibaba Cloud. Features and specifications of ECS instance families. Updated June 25, 2026. https://www.alibabacloud.com/help/en/ecs/user-guide/overview-of-instance-families

[29] Alibaba Cloud. Benefits of Elastic Compute Service. Updated May 27, 2025. https://www.alibabacloud.com/help/en/ecs/benefits

[30] Alibaba Cloud. Create an ECS instance by using the wizard. Updated January 29, 2026. https://www.alibabacloud.com/help/en/ecs/user-guide/create-an-instance-by-using-the-wizard

[31] Alibaba Cloud. Auto Provisioning overview. https://www.alibabacloud.com/help/doc-detail/120020.htm

[32] Alibaba Cloud. Elasticity Assurance overview. Updated June 20, 2026. https://www.alibabacloud.com/help/en/ecs/user-guide/overview-of-elasticity-assurance

[33] Alibaba Cloud. Auto Scaling API overview. Updated March 13, 2026. https://www.alibabacloud.com/help/en/auto-scaling/developer-reference/api-ess-2022-02-22-overview

[34] Tencent Cloud. CVM Overview. Updated November 7, 2025. https://www.tencentcloud.com/document/product/213/495

[35] Tencent Cloud. CVM Model Selection. https://www.tencentcloud.com/document/product/213/41951

[36] Tencent Cloud. Regions and Zones. Updated February 12, 2026. https://www.tencentcloud.com/document/product/213/6091

[37] Tencent Cloud. Billing Modes for CVM. Updated May 19, 2026. https://www.tencentcloud.com/document/product/213/2180

[38] Tencent Cloud. Creating a Scaling Group. https://www.tencentcloud.com/document/product/377/8551

[39] Tencent Cloud. Creating a Launch Configuration. https://www.tencentcloud.com/document/product/377/3579

[40] Volcengine. Auto Scaling - Region and availability zone. https://www.volcengine.com/docs/6617/87001?lang=en

[41] Volcengine. CreateScalingGroup - Creates a scaling group. Updated June 26, 2026. https://docs.volcengine.com/docs/6617/109812?lang=en

[42] Volcengine. Scaling group lifecycle. https://www.volcengine.com/docs/6617/87000

[43] Volcengine. Withdraw spot instance. Updated June 18, 2026. https://docs.volcengine.com/docs/6396/155896?lang=en

[44] Volcengine. Auto Scaling SDK overview. https://www.volcengine.com/docs/6617/114423

[45] Google. Site Reliability Engineering: How Google Runs Production Systems. O'Reilly Media, 2016. https://sre.google/sre-book/table-of-contents/

[46] FinOps Foundation. FinOps Framework - Optimize Cloud Usage and Cost capability. https://www.finops.org/framework/capabilities/

Part II / Chapter 7

Networking, Private Connectivity, and Exposure Control

How Addressing, Routing, DNS, Endpoint Policy, and Evidence Turn Cloud Reachability into a Governed Resource Contract

A cloud network is not a collection of subnets around an application. It is a managed reachability system: names resolve to addresses, routes select paths, policies permit or deny flows, endpoints expose services, and evidence proves whether the intended boundary actually exists. Private connectivity is therefore not achieved by drawing a VNet around a resource. It is achieved when public paths are deliberately removed or constrained, private paths are resolvable and authorized, and the resulting state is observable, testable, and reversible.

In This Chapter

- Why cloud networking should be modeled as governed reachability rather than as a diagram of address ranges
- How addressing, routing, segmentation, name resolution, endpoint state, and service policy compose a network path
- Why network reachability and data authorization are different decisions
- How to describe ingress, egress, east-west, managed-service, and management-plane exposure separately
- How VNet/VPC, subnets, route tables, security groups, NAT, load balancers, firewalls, and edge services divide responsibility
- Why private endpoints, interface endpoints, and gateway endpoints are not interchangeable with service endpoints or public firewall rules
- How private DNS becomes part of the security and availability boundary

- When peering, transit routing, VPN, direct connection, and global edge services solve different connectivity problems
- How effective routes, effective security rules, connection tests, flow logs, and activity logs create network evidence
- Why the absence of a public IP does not prove the absence of public egress or public service exposure
- How Azure, Alibaba Cloud, Tencent Cloud, and Volcengine express the same network engineering intents with non-equivalent products
- How Lab 6 proves that one managed service is reachable through a private endpoint while its public path is unavailable
- How CloudDesk FinOps Agent can later reuse bounded network inventory and exposure-audit capabilities without receiving unrestricted network administration

Cloud networking is often introduced through a familiar sequence: create a virtual network, divide it into subnets, attach security groups, and place a load balancer in front of the application. That sequence is useful, but it can create a false sense of completion. A network diagram expresses intended topology. It does not prove that the intended path resolves through the expected DNS context, follows the expected route, passes the expected policy, reaches the expected endpoint, and is denied from every path that was meant to be closed.

The control-plane view changes the question. Instead of asking only, “Can these two resources communicate?”, it asks: Who declared the path? Which resource owns the address? Which route selected the next hop? Which rule permitted the flow? Which service front door accepted it? Which evidence proves the result? What does the path cost? Which failure domains or shared dependencies can remove it? And how is the path retired without leaving a public endpoint, stale DNS record, permissive security rule, or orphaned gateway behind?

This chapter develops a resource-level model of cloud networking. Azure provides the main implementation language because its VNet, route, network security group, Private Link, Private DNS, Network Watcher, and managed-service firewall objects make the boundaries explicit. Alibaba Cloud, Tencent Cloud, and Volcengine are compared by engineering intent rather than by forcing product-name equivalence. The chapter culminates in an independent lab that uses the same Azure Blob Storage URL for two tests: the request must fail from a public client and succeed from a private virtual machine through a private endpoint. The lab is complete only when DNS, endpoint state, activity evidence, and cleanup have also been verified.



Figure 7-1 A successful data-plane operation requires both a valid network path and valid identity and authorization. The decisions are related, but they are not substitutes for one another.

7.1 Networking Is Governed Reachability

A cloud network is a set of managed claims about reachability. A subnet claims that addresses belong to a particular routing domain. A route table claims that traffic for a destination should use a particular next hop. A security group claims that selected flows are permitted or denied. A load balancer claims that a frontend address maps to healthy backends. A private endpoint claims that a managed service can be reached through a private IP address in a customer-controlled network. A private DNS zone claims that a service name should resolve differently inside a trusted network than it does outside that network.

These claims are independently configured and independently fallible. A route can be correct while DNS points to the wrong address. DNS can resolve to the private endpoint while a security group blocks the port. The network path can be open while the managed service still rejects the request because public access is disabled, the endpoint connection is not approved, or the caller lacks data-plane authorization. Conversely, a caller can hold a valid token but have no route to the endpoint. Reliable operation therefore requires a compositional model rather than a single “connected” flag.

```
networkReachable(source, name, protocol, port) =
    nameResolves(name)
    and routeExists(source, resolvedAddress(name))
    and segmentPolicyAllows(source, resolvedAddress(name), protocol, port)
    and endpointIsReady(resolvedAddress(name))
    and serviceFrontDoorAllows(source)
```

```
dataOperationAllowed =
    networkReachable
    and identityAuthenticated
    and dataAuthorizationAllows
```

This is not meant to be executed as literal code. It is an architectural checklist that prevents categories from collapsing into one another. “Private” describes the path by which the service is reached; it does not automatically describe the caller, the encryption contract, the authorization policy, the data classification, or the service-level objective.

Control-plane rule. Treat every required path as a positive assertion and every forbidden path as a negative assertion. “The application can reach the database privately” and “the database cannot be reached through its public endpoint” are two separate acceptance criteria.

Decision	Control-plane object	Evidence
Name to address	Public DNS, private DNS zone, resolver, record or alias	Query result from the intended client context; zone link and record inventory
Address to next hop	System route, route table, peering, transit route, gateway advertisement	Effective routes, next-hop test, route propagation state
Flow permitted	NSG/security group, ACL, firewall, service policy	Effective rules, denied/allowed connection test, flow log
Endpoint accepted	Public IP, load-balancer frontend, private endpoint, endpoint service	Provisioning and connection state; frontend/listener configuration
Managed service admitted	Public network access, service firewall, network security perimeter, endpoint policy	Service network settings and failed public-path request
Data operation authorized	Managed identity, role assignment, resource policy, token scope	Authentication log, authorization decision, successful bounded operation

7.2 The Network Is a Resource Graph and a Policy Graph

Virtual networking is implemented through a graph of resources with different lifecycle boundaries. The VNet or VPC owns an address space. Subnets allocate portions of that address space. Network interfaces receive private addresses. Route tables and security groups attach to subnets or interfaces. Public IP addresses, NAT gateways, load balancers, firewalls, private endpoints, DNS zones, VPN gateways, direct-connect gateways, and transit routers add new path decisions and new dependencies.

The graph is also a policy graph. An NSG attached to a subnet can apply to every current and future interface in that subnet. A user-defined route can redirect all default traffic to a firewall appliance. A private DNS zone linked to several virtual networks can change service resolution across multiple workloads. A peering can make an address space reachable without making the relationship transitive. A public IP can expose a frontend while the backends remain private. A network change therefore has a blast radius that is not visible from the changed resource alone.

The cloud network is a resource graph, a policy graph, and an evidence graph

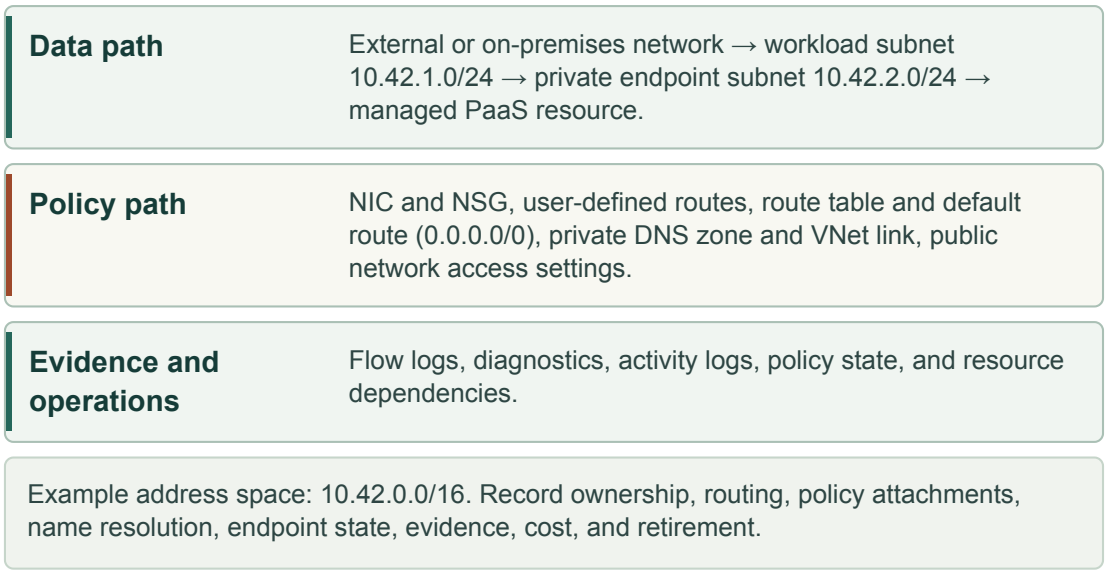


Figure 7-2 A cloud network is simultaneously a resource graph, a policy graph, and an evidence graph. Each node has its own identity, lifecycle, cost, and failure modes.

A resource-level network decision record should name at least the following:

Field	Question
Trust zones	Which sources and destinations belong to the same trust zone, and which crossings require explicit inspection or approval?
Address ownership	Who allocates each CIDR block and prevents overlap across environments, regions, tenants, and on-premises networks?
Ingress paths	Which public or private frontends accept incoming traffic, on which protocols and ports, and for which applications?

Field	Question
Egress paths	Which subnets can initiate public or cross-network traffic, through which NAT, firewall, proxy, or gateway?
East-west paths	Which workloads, subnets, VNets, VPCs, and clusters may communicate laterally?
Managed-service access	Which PaaS services use public endpoints, service endpoints, private endpoints, or service-specific private access?
DNS context	Which resolver and zone determine the address for each service name from each client location?
Inspection and evidence	Where are effective rules, route state, flow records, health, and control-plane changes observed?
Economics	Which paths incur gateway, endpoint, processing, cross-zone, cross-region, or internet-transfer charges?
Retirement	What must be deleted or detached, and in what order, to remove the path completely?

7.3 Addressing, Segmentation, and Name Resolution

Address planning is a long-lived architectural decision disguised as a setup form. A VNet or VPC address space must not only fit today's instances; it must coexist with peered networks, on-premises ranges, managed-service integrations, Kubernetes pod and service ranges, acquisition networks, disaster-recovery regions, and future tenant partitions. Overlap does not merely inconvenience routing. It can make a desired connection ambiguous or impossible without translation, proxies, or redesign.

Segmentation should follow trust and operational boundaries rather than arbitrary numerical symmetry. A three-tier application does not automatically require three subnets, and three subnets do not automatically create three security zones. A segment becomes meaningful when it has a named purpose, address ownership, route policy, security policy, lifecycle owner, evidence, and capacity plan. Subnets used for private endpoints, delegated services, gateways, firewalls, or Kubernetes nodes may also have service-specific constraints. Those constraints should be recorded before address ranges are consumed.

Cloud security groups are usually stateful, but their attachment and evaluation models differ across providers. Network ACLs may be stateless or subnet-oriented. Firewall policies may inspect application names or threat intelligence, while security groups usually operate

on addresses, service tags, protocols, and ports. Because these controls answer different questions, adding more controls does not automatically create defense in depth. It can instead create contradictory policy and ambiguous ownership.

DNS is part of the path, not a convenience layer. Private connectivity often depends on split-horizon behavior: the application continues to use the service's normal fully qualified domain name, but trusted resolvers return the private endpoint address. External resolvers continue to return the public service path. This preserves application configuration while changing the network destination according to client context. It also means that resolver availability, zone links, forwarding rules, record lifecycle, and caching behavior become part of service reliability.

Control-plane rule. Do not replace a managed-service FQDN with a hard-coded private IP. The endpoint address and service records are controlled resources. Preserve the provider-supported name and control the resolution context.

7.4 Routing, Transit, and Explicit Egress

Routes select paths; they do not grant trust. Cloud platforms typically create system routes for local subnet reachability, the broader VNet or VPC address space, and default destinations. User-defined routes can override selected system paths and direct traffic toward a virtual appliance, VPN gateway, direct-connect gateway, or other next hop. Dynamic routes learned through BGP can add or withdraw prefixes as hybrid topology changes. The effective route seen by a network interface is therefore the product of system routes, user-defined routes, propagated routes, peering, and service-specific behavior.

Peering provides direct connectivity between two virtual networks, but direct does not mean transitive. If spoke A peers with a hub and spoke B peers with the same hub, A and B do not automatically communicate through the hub. Transit requires an explicit routing or gateway design. Azure Virtual WAN, Alibaba Cloud CEN with transit routers, Tencent Cloud CCN, and Volcengine CEN or transit-router capabilities productize that responsibility at different scales. The engineering intent is to centralize route propagation and connectivity policy; the exact resource and charging model is provider-specific.

Egress deserves the same design discipline as ingress. A private virtual machine can still initiate internet connections through default outbound access, an instance public IP, a load-balancer outbound rule, a NAT gateway, a firewall, or a proxy. "No public IP" therefore does not prove "no public egress." The architecture should name the explicit egress owner, public source addresses, allowed destinations, inspection point, DNS policy, port capacity, logging path, and failure behavior.

Azure's current guidance makes this shift explicit: new virtual networks default to private subnets, and workloads that require public access should use an explicit outbound method such as NAT Gateway, Azure Firewall, or a proxy. The exact rollout and product behavior are volatile facts, but the stable engineering principle is not: implicit egress creates hidden dependencies, unstable source addresses, and unclear evidence. Explicit egress turns an accidental path into a named resource contract.

Control-plane rule. Volatile platform fact. Azure networking defaults, SKU availability, and outbound-precedence rules evolve. Verify current regional behavior before relying on a lab default or a migration sequence.

Egress mechanism	What it provides	What it does not provide
Instance public IP	Direct address and bidirectional internet path subject to security policy	Centralized source address, centralized inspection, or shared lifecycle
NAT gateway	Subnet-level source translation and predictable outbound IPs	Unsolicited inbound access or application-aware inspection
Firewall / secure web gateway	Centralized policy, inspection, logging, and controlled egress	Automatic application authorization or DNS correctness
Load-balancer outbound rule	Outbound SNAT associated with a load-balancer frontend	General-purpose egress governance or protection from all SNAT exhaustion modes
Proxy	Application-protocol mediation, destination policy, and possibly identity-aware access	Transparent support for every protocol or path
No explicit egress	Reduced public reachability when the platform truly disables implicit access	Proof that every managed dependency and update path still works

7.5 Exposure Has More Than One Direction

Exposure is often reduced to public ingress: whether an application has a public IP address or internet-facing load balancer. That is only one surface. A cloud workload may be exposed through outbound callbacks, lateral routes, peered networks, managed-service public endpoints, DNS rebinding, administrative APIs, third-party connectors, or an agent tool that can create new network resources. The review must therefore distinguish at least five directions.

Exposure is multidirectional

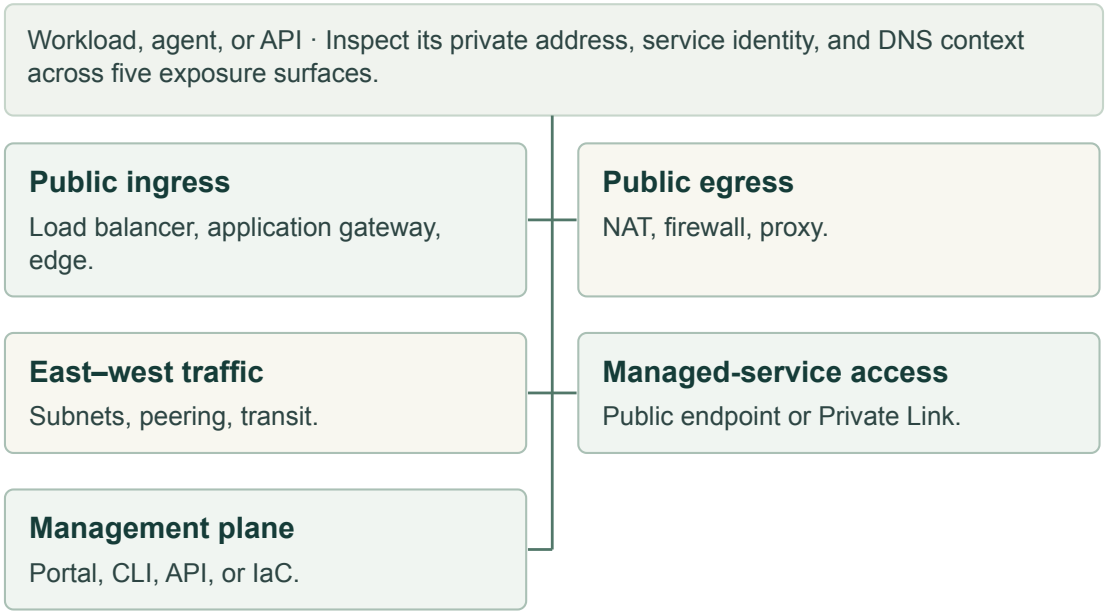


Figure 7-3 Exposure includes public ingress, public egress, east-west reachability, managed-service access, and control-plane authority. Closing one surface does not close the others.

Exposure surface	Typical resources	Required evidence
Public ingress	Public IP, public load balancer, Application Gateway, API gateway, Front Door/CDN, WAF	Listener and frontend inventory; DNS; effective inbound rules; successful and denied probes
Public egress	NAT gateway, firewall, proxy, instance public IP, load-balancer outbound rule	Effective route; source IP proof; destination policy; session/flow evidence
East-west	Subnet route, peering, transit router, service mesh, internal load balancer	Effective routes and security rules from both sides; latency and reachability tests
Managed-service front door	Public endpoint, service firewall, service endpoint, private endpoint, endpoint policy	Public-network setting; endpoint state; private DNS result; data-plane denial/allow evidence
Management plane	Portal, CLI, ARM/API, IaC pipeline, delegated role, agent tool	Role and scope inventory; activity log; policy; change approval and rollback evidence

Exposure control must also distinguish listening from reachable and reachable from authorized. A public frontend can exist without an allowed network security rule. An internal listener can be reachable from a very large peered address space. A managed service can retain a public DNS name while rejecting every public request. A management API can create a public endpoint even if the current data plane is private. The evidence should record each layer instead of assigning one binary "private" label to the entire application.

7.6 Ingress: Layer 4, Layer 7, and the Global Edge

Ingress resources implement different contracts. A Layer 4 load balancer distributes TCP or UDP flows according to frontend addresses, rules, health probes, and backend pools. It is appropriate when the network protocol should pass through with minimal application interpretation. An application gateway or application load balancer understands HTTP attributes such as host names, URL paths, headers, cookies, certificates, and web-application firewall rules. A global edge service adds anycast or DNS-based distribution, CDN behavior, global health evaluation, and edge security.

These services are composable rather than mutually exclusive. A global edge can route to a regional application gateway, which then routes to private backends. An internal Layer 4 load balancer can front a database proxy or service-mesh gateway. Private Link can protect an origin behind a global or regional frontend. The design should make each termination and trust transition explicit: where TLS terminates, where client identity is checked, where WAF policy applies, where source information is preserved, and how health probes differ from real user requests.

High availability is not created by inserting a load balancer in front of one backend. The frontend, health probe, DNS, certificate, backend pool, zone distribution, and application state must all support the intended failure domain. Likewise, an edge service can improve global routing and absorb certain attacks without repairing a broken regional dependency or an application that reports health incorrectly.

Intent	Typical Azure reference	Key design question
Regional TCP/UDP distribution	Standard Load Balancer	Which ports and flows require pass-through Layer 4 balancing?
Regional HTTP/S routing and WAF	Application Gateway v2 / WAF_v2	Where do TLS, host/path routing, and web policy terminate?
Global HTTP/S delivery and edge security	Azure Front Door	How are global health, origin privacy, caching, and failover governed?

Intent	Typical Azure reference	Key design question
DNS-based global routing	Traffic Manager	Is DNS-level endpoint selection sufficient without proxying traffic?
Centralized network inspection	Azure Firewall or network virtual appliance	Which ingress and egress flows require stateful or application-aware inspection?

7.7 Private Connectivity Is an Endpoint Contract

Private connectivity to a managed service is frequently described as "putting the service in the VNet." That wording is usually inaccurate. The service remains operated in the provider's service infrastructure. A private endpoint creates a customer-visible network interface and private address that maps traffic to a specific service resource or subresource over the provider's private network. The customer controls the endpoint's subnet, DNS integration, network policy where supported, and consumer-side lifecycle. The provider controls the managed service and the Private Link implementation.

This model differs from a service endpoint. A service endpoint extends the identity of a subnet to a managed service's public endpoint and allows the service firewall to admit that subnet. The destination remains a public service address, even though traffic may stay on the provider backbone. A private endpoint changes the destination seen by the client to a private address in the VNet or VPC. It can support stronger exfiltration controls and more explicit private DNS, but it adds endpoint, DNS, approval, quota, cost, and lifecycle responsibilities.

Alibaba Cloud also distinguishes gateway endpoints and PrivateLink interface endpoints. A gateway endpoint can add a route for supported cloud services and apply endpoint policy without allocating an interface address. PrivateLink interface endpoints use private network interfaces and security groups and can participate in more complex VPC, CEN, Express Connect, or VPN designs. Tencent Cloud Private Link and Volcengine PrivateLink likewise connect endpoint consumers to supported endpoint services, but supported backends, regions, cross-account behavior, and DNS integration differ.

Private-access pattern	Address seen by client	Primary policy point	Typical strength	Important limitation
Public endpoint + IP firewall	Public IP	Managed-service firewall	Simple restriction for known public sources	Still uses public endpoint; source IP can be unstable

Private-access pattern	Address seen by client	Primary policy point	Typical strength	Important limitation
Service endpoint / VNet rule	Public service IP	Subnet identity plus service firewall	Direct provider-backbone path with simple service restriction	Not equivalent to a private IP or Private Link
Private endpoint / interface endpoint	Private IP in consumer VNet/VPC	Endpoint, security policy, DNS, and service connection state	Strong private path and explicit endpoint lifecycle	Costs, quotas, DNS complexity, service support limits
Gateway endpoint	Provider prefix routed through endpoint	Route table and endpoint policy	Efficient private access for supported object services	Service and routing support is provider-specific
Self-hosted private service	Private IP behind internal load balancer or endpoint service	Consumer and provider VPC policies	Full control over application service publication	Customer owns backend availability and patching

Control-plane rule. A private endpoint does not disable the public endpoint. Configure the managed service's public-network policy separately, and prove the negative path from an external client.

7.8 DNS Is Part of the Security and Availability Boundary

The most robust private-endpoint pattern preserves the service's standard FQDN. Public DNS publishes a canonical chain that identifies the service's private-link domain. A private DNS zone linked to the trusted VNet overrides the final answer with the endpoint's private address. An external resolver follows the public chain and reaches the public service front door, which should reject the request when public access is disabled. An internal resolver returns the private address and the client reaches the private endpoint.

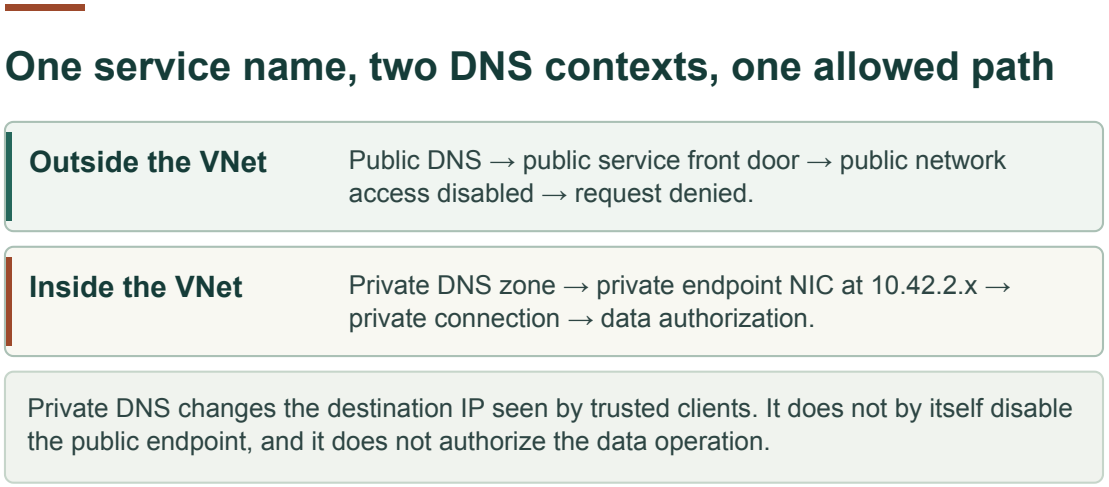


Figure 7-4 Private DNS changes the address returned to trusted clients. Public-network policy still has to close the public path, and data-plane authorization still has to approve the operation.

DNS failure modes are operationally significant. The zone may exist but not be linked to the client VNet. A custom DNS server may not forward the private-link zone. A peered VNet may lack a zone link. A stale record may point to a deleted endpoint. Two independently managed zones with the same name may contain divergent records. A hybrid client may query on-premises DNS that does not forward to the cloud resolver. Because the application still uses a familiar hostname, these failures can look like generic connectivity or TLS errors unless the resolution path is captured explicitly.

DNS resolution and access control are independent. Public DNS resolution can reveal that a globally unique service name exists without revealing its private address or granting access. Conversely, a correct private address can still be blocked by route, NSG, endpoint policy, service firewall, identity, or data authorization. The lab later in this chapter records DNS output together with connection and service evidence so that the failure domain is visible.

7.9 Peering, Transit, Hybrid Connectivity, and Edge

A single VNet or VPC is a regional isolation domain. Larger systems add connectivity domains deliberately. Peering connects address spaces directly and is usually optimized for low-latency private traffic. Transit platforms centralize route propagation, segmentation, shared services, and multi-region connectivity. VPN gateways encrypt traffic over public networks. Direct-connect services establish private circuits and BGP relationships with enterprise or carrier networks. Edge and CDN services move public application entry points closer to users and protect origins.

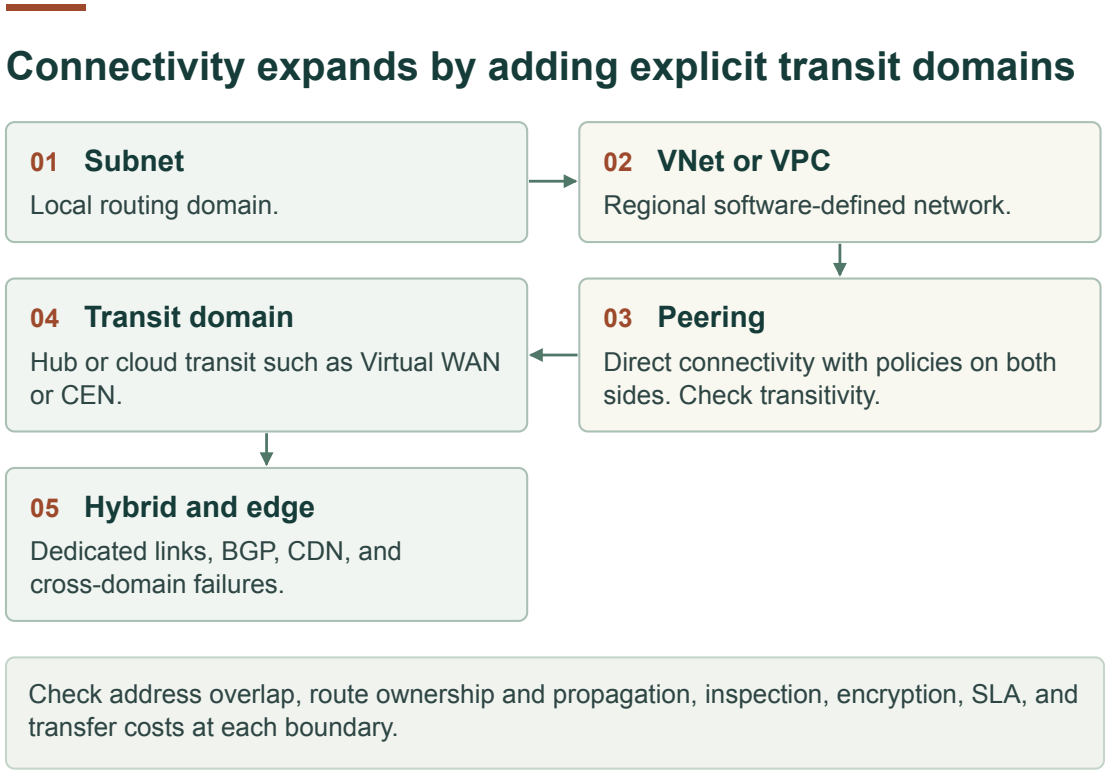


Figure 7-5 Connectivity expands from a subnet to a regional VNet, direct peering, transit domains, and hybrid or global edge services. Each step adds route state, policy, cost, and failure modes.

The decision is not "peering or transit" in isolation. It depends on the number of networks, route scale, need for centralized inspection, cross-region topology, overlapping addresses, shared DNS, organizational ownership, and required failure isolation. A small pair of VNets may use direct peering. A large organization may use hub-and-spoke or a managed transit service. A hybrid environment may terminate VPN or direct connection in a hub and distribute routes to spokes. A global application may combine private transit for service-to-service traffic with public edge services for user traffic.

Connectivity does not automatically imply policy symmetry. Both sides may need security rules. Route propagation can be asymmetric. A firewall may see the forward path but not the return path. Network address translation can hide the original source. Peering and transit charges may apply in both directions. A design record should identify path symmetry, source preservation, maximum tolerated latency, encryption contract, route ownership, BGP advertisements, failover behavior, and diagnostic evidence.

7.10 Network Security Controls Are Layered but Non-Interchangeable

Network security is strongest when each control has a precise responsibility. Security groups and NSGs provide distributed stateful filtering close to subnets or interfaces. Network ACLs provide subnet-oriented filtering in providers that expose them. Firewalls centralize stateful or application-aware inspection. Web application firewalls inspect HTTP/S requests for application attacks. DDoS services protect network or application entry points against volumetric and protocol attacks. Private endpoints remove public network paths to supported services. Identity and authorization decide who may perform the data operation.

These layers should not be used as rhetorical substitutes. A WAF does not protect an SSH listener. A security group does not validate an OAuth token. A private endpoint does not prevent an overprivileged identity from deleting data. A firewall does not fix an incorrect DNS zone. A network security perimeter or service firewall does not create an endpoint in the consumer VNet. The design should state the failure or threat class each control addresses and the evidence used to prove it.

Control	Primary layer	Protects against / governs	Does not prove
NSG / security group	L3/L4 distributed policy	Address, protocol, port, and service-tag reachability	Application identity, content safety, or DNS correctness
Route table	Path selection	Next-hop choice and traffic steering	Trust, authorization, or application health
NAT gateway	Address translation	Predictable outbound identity and SNAT capacity	Inspection or unsolicited inbound access
Firewall	Centralized L3-L7 policy	Stateful filtering, egress policy, threat intelligence, inspection	Business authorization or backend correctness
WAF	HTTP application edge	Common web attacks, bots, request policies	Non-HTTP protocols or internal lateral movement
Private endpoint	Managed-service access path	Private destination address and provider-backbone path	Public endpoint closure or data authorization
DDoS protection	Network/application availability	Volumetric and protocol attacks within service scope	Application correctness or credential abuse

Control	Primary layer	Protects against / governs	Does not prove
Identity and RBAC	Control/data authorization	Who can manage or use the resource	Network path existence

Zero Trust does not mean removing the network. It means the network location is not treated as sufficient trust. Identity, device, workload, data, and risk signals still matter; least-privilege network paths reduce attack surface and exfiltration opportunities while identity and application policy decide the operation. The most useful design is therefore "private by path, explicit by policy, authorized by identity, and evidenced by telemetry."

7.11 Network Evidence: Effective State, Tests, and Flow Records

A network control plane is only trustworthy when intended state can be compared with effective state. The declared route table may not include system or propagated routes. The NSG file in a repository may not be the NSG attached to the active subnet. A private endpoint may be provisioned but awaiting approval. A DNS zone may contain the correct record but be unlinked from the client network. Diagnostic evidence must therefore be collected from the same source, destination, name, protocol, port, and time window as the claim being tested.

Azure Network Watcher exposes tools for IP flow verification, NSG diagnostics, next-hop analysis, effective security rules, connection troubleshooting, packet capture, and VPN troubleshooting. Connection troubleshoot can test a VM or other supported source against an FQDN, URI, IP address, or another resource and report DNS, route, NSG, socket, and latency information. Virtual network flow logs record Layer 4 flows at VNet scope and supersede the older NSG flow-log model. The important architectural point is not the specific tool name; it is that configuration evidence, active test evidence, and observed-flow evidence answer different questions.

Control-plane rule. Volatile platform fact. Azure no longer allows new NSG flow-log creation and plans to retire the feature on September 30, 2027. Use virtual network flow logs for new designs, and verify supported workloads and regional availability.

Exposure control is a managed feedback loop

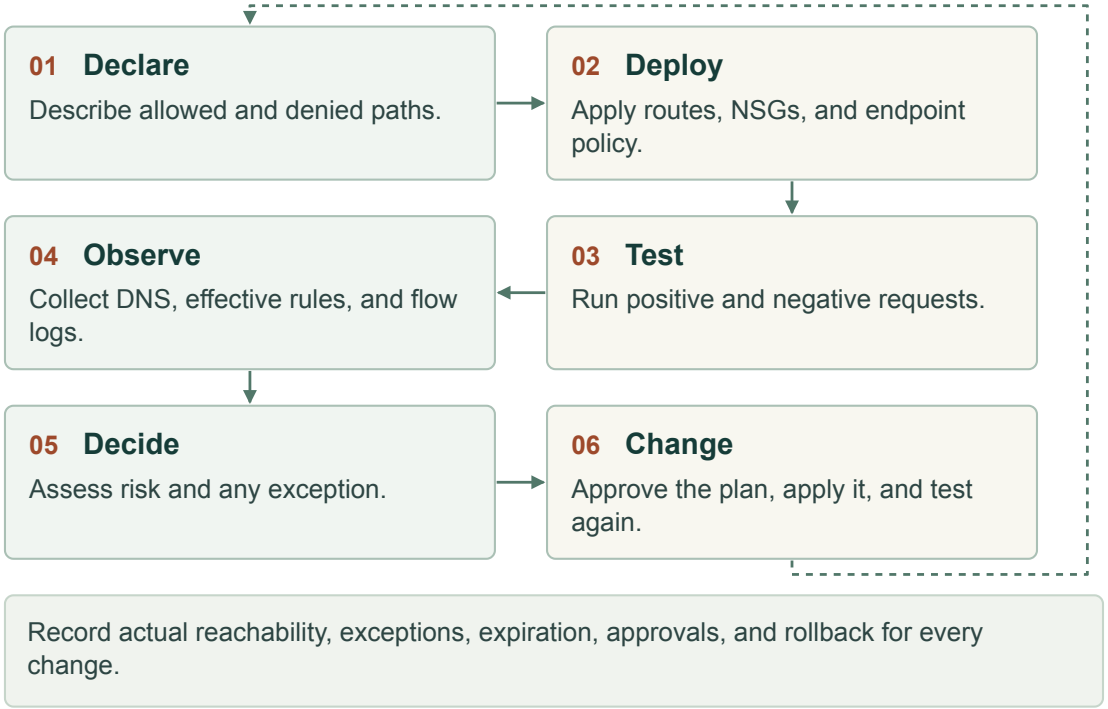


Figure 7-6 Exposure control is a managed feedback loop: declare, deploy, test, observe, decide, and remediate. A topology diagram is only the first step.

Evidence type	Question answered	Typical limitation
Resource inventory	What network resources and attachments currently exist?	Does not prove the active packet path
Effective routes	Which next hop will the source use for a destination?	Does not prove the destination accepts the flow
Effective security rules	Which distributed policy applies to the interface or subnet?	May not include service-front-door or application policy
DNS query from client context	Which address does this client actually receive?	Does not prove the address is reachable or authorized
Connection test	Can this source reach this destination and port now?	A point-in-time probe; may not represent real traffic volume
Flow log / session log	Which flows were observed, allowed, denied, or translated?	Retention, sampling, supported workloads, and cost vary
Activity or audit log	Who changed the control-plane resource and when?	Usually does not contain application data-plane events

Evidence type	Question answered	Typical limitation
Application trace	Did the request reach and complete in the application?	May not identify the exact network control that failed

7.12 Reliability and Economics of Network Paths

Every network path has cost and failure domains. Public egress may incur data-transfer charges, NAT processing, firewall processing, proxy capacity, and destination charges. Private endpoints usually have hourly and data-processing charges. Peering, transit, cross-zone, cross-region, direct-connect, and global-edge services apply different ingress, egress, port, gateway, or bandwidth models. Flow logs and traffic analytics add ingestion, storage, and analysis cost. Network cost therefore belongs in the resource graph rather than as an invoice surprise.

A cheaper path may also have different reliability. A single virtual appliance can become a bottleneck or shared failure domain. A zonal NAT gateway may not satisfy a zone-failure objective. A private endpoint can be healthy while private DNS is unavailable. A VPN can provide encrypted connectivity without the latency or availability characteristics of a private circuit. A direct circuit can fail while route propagation or customer-premises equipment remains the real recovery bottleneck. The reliability statement should name the path, each dependency, the protected failure domain, and the failover mechanism.

Network optimization should be evidence-driven. Remove unused public IPs, idle load balancers, stale peerings, unneeded private endpoints, duplicate flow logs, and expensive cross-region paths only after ownership and dependency are proven. Consolidating egress can reduce cost and improve control, but it can also increase blast radius. Moving a service behind Private Link can reduce public exposure while adding endpoint and DNS cost. The optimization objective is not the smallest network bill; it is the required reachability, security, performance, and recovery outcome at an explainable cost.

7.13 Multicloud Translation by Engineering Intent

Azure, Alibaba Cloud, Tencent Cloud, and Volcengine share the same major network intents: isolate address spaces, segment workloads, select routes, control ingress and egress, connect networks, publish services privately, resolve private names, inspect traffic, and collect evidence. The products are not operationally identical. Supported endpoint services, route propagation, security-group semantics, private DNS integration, cross-region behavior, billing, quotas, and diagnostic tools must be verified for each provider and region.

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Regional private network	Virtual Network (VNet)	VPC + vSwitch	VPC + subnet	VPC + subnet
Distributed packet policy	NSG / ASG	Security group / network ACL	Security group / network ACL	Security group / network ACL
Custom routing and egress	Route table, NAT Gateway, Firewall	Route table, Internet/VPC NAT Gateway, Cloud Firewall	Route table, NAT Gateway, CFW	Route table, NAT Gateway, Cloud Firewall
L4 / L7 ingress	Load Balancer / Application Gateway	NLB, CLB, ALB	CLB / ALB	CLB / ALB
Private service access	Private Link / private endpoint; service endpoint	PrivateLink interface endpoint; gateway endpoint for supported services	Private Link endpoint and endpoint service	PrivateLink endpoint and endpoint service
Network interconnection	VNet peering, Virtual WAN	VPC peering, CEN / transit router	VPC peering, CCN	VPC peering, CEN / transit router
Hybrid connectivity	VPN Gateway, ExpressRoute	VPN Gateway, Express Connect	VPN Connection, Direct Connect	VPN Connection, Direct Connect
Private DNS	Azure Private DNS / Private Resolver	PrivateZone / DNS resolver capabilities	Private DNS	PrivateZone / DNS resolver capabilities
Traffic evidence	Network Watcher, VNet flow logs, Activity Log	VPC flow logs, CloudMonitor, ActionTrail	Flow Logs to CLS, diagnostics, CloudAudit	VPC flow logs, CloudMonitor, CloudTrail

A portable design record should describe the intent first: “workload subnet has no implicit internet egress,” “managed storage is reachable only by private address,” “public HTTP enters through an L7 frontend with WAF,” or “on-premises routes are learned through redundant private circuits.” The provider mapping is then evaluated for functional, operational, governance, economic, and failure-domain equivalence. A product with a similar name may still require a different architecture.

7.14 Lab 6 - Create Private Connectivity and Prove Public Exposure Is Disabled

This independent lab creates an Azure Storage account, a private endpoint for the Blob service, a private DNS zone, and a small Linux VM with no public IP address. The storage account begins with public network access enabled only long enough to upload one test object. The same Bicep deployment is then updated to disable the public endpoint. The read-

er uses one blob URL for two tests: a request from the public shell must fail, while a request executed inside the VNet through Azure Run Command must resolve to the private IP address and return the object.

Lab 6: the same blob URL must fail publicly and succeed privately

Public negative test	External client → public DNS → storage account public front door → GET fails.
Private positive test	VM at 10.42.1.x without a public IP → private DNS → private endpoint NIC at 10.42.2.x → blob read succeeds.
Management evidence	In book-ch07-network, use Run Command for the internal test. Capture the resource graph, A records, access policy, activity log, and cleanup.

Acceptance criterion: identical FQDN, explicit negative test from outside, explicit positive test from inside, and zero-remainder cleanup proof.

Figure 7-7 Lab 6 uses one Blob Storage FQDN to prove two different paths: the public path is denied, while the private path resolves to the endpoint NIC and succeeds.

7.14.1 Capability Goal and Acceptance Criteria

The capability goal is not merely to create a private endpoint. It is to produce a bounded evidence packet proving the intended reachability and the intended absence of reachability. The lab is successful when all of the following are true:

- The VNet, two subnets, NSG, VM, private DNS zone, private endpoint, Blob container, and storage account are created in an isolated resource group from declared state.
- The VM has no public IP address, no inbound management rule, and no implicit default outbound path.
- The private endpoint connection state is approved and targets the Blob subresource.
- From the private VM, the normal storage FQDN resolves to an RFC 1918 address in the private-endpoint subnet.
- The same private VM retrieves the test object successfully through HTTPS.
- From the public shell, the same object URL does not return the object after public network access is disabled.

- The storage network settings, private DNS zone link, endpoint connection, Resource Graph inventory, and Activity Log evidence are saved.
- The resource group is deleted and a final query proves that no lab resources remain.

7.14.2 Prerequisites and Cost Boundary

Requirement	Lab choice
Azure access	A subscription in which the reader can create a resource group, network resources, a small VM, a storage account, a private endpoint, a private DNS zone, and role-neutral Run Command operations.
Local tools	Azure CLI, Bicep support, curl, an SSH public key, and a shell that can save JSON and text evidence.
Region	Choose one region that supports the selected VM size, Storage, Private Link, and Run Command. Verify current quotas and API versions.
Cost class	Low but not free. The VM, private endpoint, storage, and any optional flow logs incur charges. Complete and delete the lab promptly.
Security boundary	Use synthetic content only. The SAS token is short-lived and must not be committed to source control or evidence archives.
Evidence directory	Create evidence/ before deployment and store command outputs, headers, DNS output, Resource Graph results, and cleanup proof.

Control-plane rule. The sample API versions, region, VM size, image, and command output are implementation defaults to verify. They are not permanent facts of the book.

7.14.3 Declare the Private-Access Resource Graph

Save the following as main.bicep. The template accepts publicNetworkAccess as a parameter so that the same declared graph can be deployed first in bootstrap mode and then updated to private-only mode. The storage container is an ARM resource; the test blob is uploaded in the next step. The workload subnet explicitly disables default outbound access, and the VM has no public IP. The private endpoint uses a separate subnet and a Blob private DNS zone.

```
param location string = resourceGroup().location
param prefix string
param adminUsername string = 'azureuser'
@secure()
param sshPublicKey string
@allowed([
  'Enabled'
  'Disabled'
```

```
])
param publicNetworkAccess string = 'Enabled'

var vnetName = '${prefix}-vnet'
var workloadSubnetName = 'workload'
var endpointSubnetName = 'private-endpoints'
var storageName = take(toLower('${prefix}${uniqueString(resourceGroup().id)}'),
24)
var dnsZoneName = 'privatelink.blob.core.windows.net'
var tags = {
  workload: 'network-private-access-lab'
  environment: 'lab'
  owner: 'book-reader'
  costCenter: 'learning'
  expiresOn: 'replace-with-date'
}

resource nsg 'Microsoft.Network/networkSecurityGroups@2024-05-01' = {
  name: '${prefix}-workload-nsg'
  location: location
  tags: tags
  properties: {
    securityRules: [
      {
        name: 'Deny-Internet-Outbound'
        properties: {
          priority: 200
          direction: 'Outbound'
          access: 'Deny'
          protocol: '*'
          sourcePortRange: '*'
          destinationPortRange: '*'
          sourceAddressPrefix: 'VirtualNetwork'
          destinationAddressPrefix: 'Internet'
        }
      }
    ]
  }
}

resource vnet 'Microsoft.Network/virtualNetworks@2024-05-01' = {
  name: vnetName
  location: location
  tags: tags
  properties: {
    addressSpace: { addressPrefixes: [ '10.42.0.0/16' ] }
    subnets: [
      {
```

```
        name: workloadSubnetName
        properties: {
          addressPrefix: '10.42.1.0/24'
          networkSecurityGroup: { id: nsg.id }
          defaultOutboundAccess: false
        }
      }
      {
        name: endpointSubnetName
        properties: {
          addressPrefix: '10.42.2.0/24'
          privateEndpointNetworkPolicies: 'Disabled'
          defaultOutboundAccess: false
        }
      }
    ]
  }
}

resource nic 'Microsoft.Network/networkInterfaces@2024-05-01' = {
  name: '${prefix}-vm-nic'
  location: location
  tags: tags
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          subnet: { id: resourceId('Microsoft.Network/virtualNetworks/subnets',
vnet.name, workloadSubnetName) }
        }
      }
    ]
  }
}

resource vm 'Microsoft.Compute/virtualMachines@2024-07-01' = {
  name: '${prefix}-vm'
  location: location
  tags: tags
  properties: {
    hardwareProfile: { vmSize: 'Standard_B1s' }
    storageProfile: {
      imageReference: {
        publisher: 'Canonical'
        offer: 'ubuntu-24_04-lts'
        sku: 'server'
```

```
        version: 'latest'
      }
      osDisk: { createOption: 'FromImage', deleteOption: 'Delete' }
    }
    osProfile: {
      computerName: '${prefix}-vm'
      adminUsername: adminUsername
      linuxConfiguration: {
        disablePasswordAuthentication: true
        ssh: { publicKeys: [ { path:
'/home/${adminUsername}/.ssh/authorized_keys', keyData: sshPublicKey } ] }
      }
    }
    networkProfile: {
      networkInterfaces: [ { id: nic.id, properties: { deleteOption: 'Delete' }
} ]
    }
    diagnosticsProfile: { bootDiagnostics: { enabled: true } }
  }
}

resource storage 'Microsoft.Storage/storageAccounts@2025-01-01' = {
  name: storageName
  location: location
  tags: tags
  sku: { name: 'Standard_LRS' }
  kind: 'StorageV2'
  properties: {
    allowBlobPublicAccess: false
    allowSharedKeyAccess: true
    minimumTlsVersion: 'TLS1_2'
    publicNetworkAccess: publicNetworkAccess
    networkAcls: {
      bypass: 'None'
      defaultAction: publicNetworkAccess == 'Disabled' ? 'Deny' : 'Allow'
    }
  }
}

resource blobService 'Microsoft.Storage/storageAccounts/blobServices@2023-05-01'
= {
  parent: storage
  name: 'default'
}
resource container
'Microsoft.Storage/storageAccounts/blobServices/containers@2023-05-01' = {
  parent: blobService
  name: 'lab'
```

```
  properties: { publicAccess: 'None' }
}

resource dnsZone 'Microsoft.Network/privateDnsZones@2024-06-01' = {
  name: dnsZoneName
  location: 'global'
  tags: tags
}
resource dnsLink 'Microsoft.Network/privateDnsZones/virtualNetworkLinks@2024-06-
01' = {
  parent: dnsZone
  name: '${prefix}-vnet-link'
  location: 'global'
  properties: {
    registrationEnabled: false
    virtualNetwork: { id: vnet.id }
  }
}

resource endpoint 'Microsoft.Network/privateEndpoints@2024-05-01' = {
  name: '${prefix}-blob-pe'
  location: location
  tags: tags
  properties: {
    subnet: { id: resourceId('Microsoft.Network/virtualNetworks/subnets',
vnet.name, endpointSubnetName) }
    privateLinkServiceConnections: [
      {
        name: '${prefix}-blob-connection'
        properties: {
          privateLinkServiceId: storage.id
          groupIds: [ 'blob' ]
        }
      }
    ]
  }
}
resource dnsZoneGroup
'Microsoft.Network/privateEndpoints/privateDnsZoneGroups@2024-05-01' = {
  parent: endpoint
  name: 'default'
  properties: {
    privateDnsZoneConfigs: [ { name: 'blob', properties: { privateDnsZoneId:
dnsZone.id } } ]
  }
}

output storageName string = storage.name
```

```
output vmId string = vm.id
output storageFqdn string = '${storage.name}.blob.core.windows.net'
output privateEndpointId string = endpoint.id
```

If your environment does not support one of the listed API versions or the defaultOutboundAccess property, select a current supported version and record the change in the lab notes. Do not silently remove the private-subnet intent; replace it with an explicit equivalent and prove the resulting egress state.

7.14.4 Bootstrap the Object, Then Close the Public Path

```
RG=book-ch07-network
LOCATION=eastus2
PREFIX=ch07demo
SSH_KEY_FILE="$HOME/.ssh/id_ed25519.pub"
mkdir -p evidence

az group create --name "$RG" --location "$LOCATION"   --tags workload=network-
private-access-lab environment=lab owner=book-reader costCenter=learning

az deployment group create   --resource-group "$RG"   --template-file main.bicep
--parameters prefix="$PREFIX" sshPublicKey="$(cat "$SSH_KEY_FILE")"
publicNetworkAccess=Enabled   --output json > evidence/01-bootstrap-
deployment.json

STORAGE=$(az deployment group show --resource-group "$RG" --name main   --query
properties.outputs.storageName.value -o tsv)
VM_ID=$(az deployment group show --resource-group "$RG" --name main   --query
properties.outputs.vmId.value -o tsv)
FQDN="$STORAGE.blob.core.windows.net"

echo 'private-path-ok' > probe.txt
KEY=$(az storage account keys list --resource-group "$RG" --account-name
"$STORAGE"   --query '[0].value' -o tsv)
az storage blob upload --account-name "$STORAGE" --account-key "$KEY"   --
container-name lab --name probe.txt --file probe.txt --overwrite true

az deployment group create   --resource-group "$RG"   --template-file main.bicep
--parameters prefix="$PREFIX" sshPublicKey="$(cat "$SSH_KEY_FILE")"
publicNetworkAccess=Disabled   --output json > evidence/02-private-only-
deployment.json

az storage account show --resource-group "$RG" --name "$STORAGE"   --query
'{id:id, publicNetworkAccess:publicNetworkAccess,
defaultAction:networkRuleSet.defaultAction}'   --output json > evidence/03-
storage-network-state.json
```

The upload occurs before public access is disabled so that the reader does not need a second private workstation or a deployment script solely to seed the object. The final deployed state is private-only. The short bootstrap period is part of the evidence and must not be mistaken for the production target state.

7.14.5 Generate a Short-Lived Read-Only SAS

```
EXPIRY=$(date -u -v+30M '+%Y-%m-%dT%H:%MZ' 2>/dev/null || date -u -d '+30
minutes' '+%Y-%m-%dT%H:%MZ')
SAS=$(az storage blob generate-sas   --account-name "$STORAGE"   --account-key
"$KEY"   --container-name lab   --name probe.txt   --permissions r   --expiry
"$EXPIRY"   --https-only   --output tsv)
URL="https://$FQDN/lab/probe.txt?$SAS"

# Do not write URL or SAS into the permanent evidence packet.
unset KEY
```

The SAS is a temporary data-plane credential used only to make the public and private network tests identical. It is not a recommended application-authentication design. A production workload should prefer workload identity and least-privilege data roles where the service supports them. The evidence packet should record the expiry and permission set, but not the token value.

7.14.6 Prove the Public Path Is Unavailable

```
set +e
curl --connect-timeout 10 --max-time 20   --silent --show-error   --dump-header
evidence/04-public-headers.txt   --output evidence/04-public-body.txt   --write-
out '%{http_code}
'   "$URL" | tee evidence/04-public-status.txt
PUBLIC_RC=$?
set -e

echo "curl_exit=$PUBLIC_RC" >> evidence/04-public-status.txt
head -40 evidence/04-public-headers.txt
```

Run this command from a client outside the VNet, such as the local development shell or Cloud Shell. The request must not return the object. Depending on the client path and current Storage behavior, the result may be an HTTP denial, connection rejection, or service-specific network error. Save the status, headers, and body. The acceptance criterion is not one permanent error string; it is that the object contents are unavailable through the public path after the public-network setting is disabled.

7.14.7 Prove Private DNS and Private Data Access

```
az vm run-command invoke   --ids "$VM_ID"   --command-id RunShellScript   --
scripts "set -e
    echo 'DNS:'
    getent ahostsv4 '$FQDN'
    echo 'CONTENT:'
    curl --fail --silent --show-error '$URL'
  "   --output json > evidence/05-private-run-command.json

jq -r '.value[].message' evidence/05-private-run-command.json
```

The DNS output must show an address in the private-endpoint subnet, and the content output must be private-path-ok. This proves that the same FQDN has been resolved in the private DNS context and that the service accepted the connection through Private Link. The SAS proves data authorization only for one object and one short time window; it does not weaken the public-path test because the public client uses the identical credential.

7.14.8 Inspect Endpoint, DNS, Route, and Connection Evidence

```
az network private-endpoint-connection list   --id "$(az storage account show -g
"$RG" -n "$STORAGE" --query id -o tsv)"   --output json > evidence/06-private-
endpoint-connections.json

az network private-dns record-set a list   --resource-group "$RG"   --zone-name
privatelink.blob.core.windows.net   --output json > evidence/07-private-dns-
records.json

az network watcher test-connectivity   --source-resource "$VM_ID"   --dest-
address "$FQDN"   --dest-port 443   --protocol Tcp   --output json >
evidence/08-connection-troubleshoot.json

az monitor activity-log list   --resource-group "$RG"   --offset 2h   --output
json > evidence/09-activity-log.json
```

If the agentless connection-troubleshoot experience or a command parameter differs in the current CLI, use the current supported command and record the adaptation. The evidence should still show the source, destination FQDN, port, reachability result, and any route, DNS, or security issue detected. Virtual network flow logs can be enabled as an advanced extension, but they are not required for this short lab because they add storage, ingestion delay, supported-workload constraints, and cost.

7.14.9 Query the Network Resource Graph

```
az extension add --name resource-graph --upgrade

QUERY=$(cat <<'KQL'
Resources
| where resourceGroup =~ 'book-ch07-network'
| where type in~ (
    'microsoft.network/virtualnetworks',
    'microsoft.network/networksecuritygroups',
    'microsoft.network/networkinterfaces',
    'microsoft.network/privateendpoints',
    'microsoft.network/privatednszones',
    'microsoft.compute/virtualmachines',
    'microsoft.storage/storageaccounts')
| extend publicNetworkAccess=tostring(properties.publicNetworkAccess),
         provisioningState=tostring(properties.provisioningState),
         privateLinkServiceConnections=properties.privateLinkServiceConnections
| project id, type, name, location, publicNetworkAccess, provisioningState,
          privateLinkServiceConnections, tags
| order by type asc, name asc
KQL
)
az graph query --graph-query "$QUERY" --output json > evidence/10-resource-
graph.json
```

The graph should contain the VNet, NSG, NIC, VM, private endpoint, private DNS zone, and storage account. The storage account should report publicNetworkAccess as Disabled. Resource Graph does not replace active testing; it proves the resource and policy inventory that produced the observed path.

7.14.10 Evidence Packet

Artifact	Minimum content
01-bootstrap-deployment.json	Initial deployment result and resource identities.
02-private-only-deployment.json	Idempotent update that disables the public endpoint.
03-storage-network-state.json	PublicNetworkAccess Disabled and default network action Deny.
04-public-*	Public-client status, headers, and body proving the object was not returned.
05-private-run-command.json	Private DNS result and successful retrieval of private-path-ok.

Artifact	Minimum content
06-private-endpoint-connections.json	Approved endpoint connection for the Blob subresource.
07-private-dns-records.json	A record for the storage account in the Blob private-link zone.
08-connection-troubleshoot.json	Source, destination, port, and active connectivity result.
09-activity-log.json	Control-plane creation and network-setting changes.
10-resource-graph.json	Final resource inventory and public-network state.
11-cleanup-proof.json	Zero-result query after resource-group deletion.
lab-notes.md	Region, API versions, deviations, cost boundary, SAS expiry, and observed error behavior.

7.14.11 Common Failure Modes

Symptom	Likely cause	Evidence to inspect
Private VM resolves a public IP	Private DNS zone missing, wrong zone name, no VNet link, or custom DNS lacks forwarding	getent/nslookup result; private DNS zone links and records
Private DNS resolves correctly but curl times out	Route or NSG issue; endpoint not ready; VM guest firewall; service path issue	Connection troubleshoot, effective rules, endpoint state
Public client still retrieves the object	Public network access not disabled, trusted exception remains, or test used a different endpoint	Storage network settings, Activity Log, public request headers
Private curl returns authorization error	SAS expired, wrong blob path, insufficient permission, or clock skew	SAS expiry and permissions; blob inventory; response headers
Run Command fails before script execution	VM agent not ready, unsupported image, or platform communication issue	VM instance view, boot diagnostics, Run Command status
Deployment fails on subnet property	API version or regional feature mismatch	Deployment operation details and current provider documentation
Private endpoint connection remains pending	Service requires manual approval or connection validation failed	Private endpoint connection state and resource activity log

Symptom	Likely cause	Evidence to inspect
Cleanup leaves resources	Dependency deletion still running or an item was created outside the resource group	Resource Graph and subscription-wide search by tags

7.14.12 Clean Up and Prove Zero Remainder

```
unset SAS_URL
az group delete --name "$RG" --yes --no-wait
az group wait --name "$RG" --deleted

az graph query   --graph-query "Resources | where resourceGroup =~ '$RG' |
project id, type, name"   --output json > evidence/11-cleanup-proof.json
```

The final query must return no resources. Also search for resources with the workload tag in case a diagnostic or regional Network Watcher resource was created outside the lab resource group. Retain the cleanup proof and remove the short-lived SAS from shell history or temporary files where practical.

7.14.13 Chinese-Cloud Translation

Repeat the acceptance criteria, not the Azure syntax. On Alibaba Cloud, use a VPC, vSwitches, security groups, PrivateLink interface endpoint or an OSS gateway endpoint where supported, PrivateZone or the documented DNS integration, the service public-access setting, and VPC flow logs or ActionTrail evidence. On Tencent Cloud, use a VPC, subnet, security group, Private Link endpoint and endpoint service or service-specific private access, Private DNS, Flow Logs delivered to CLS, and CloudAudit evidence. On Volcengine, use a VPC, subnet, security group, PrivateLink, PrivateZone or supported private resolution, VPC flow logs, and CloudTrail. In every provider, verify service support, endpoint type, approval behavior, region, DNS automation, pricing, and the control that actually disables public access.

7.15 CloudDesk Reuse: Private-Access Audit and Exposure Detection

CloudDesk FinOps Agent can reuse this chapter's read-only network reasoning without receiving unrestricted rights to create routes, public IPs, peerings, or endpoints. The first capability audits whether a managed service has a valid private path and whether the public path is closed. The second inventories public and implicit exposure across resource, net-

work, DNS, and service layers. A mutating remediation capability should remain separate and require an approved plan, ownership validation, cost estimate, rollback path, and post-change connectivity tests.

```
capability: network.privateAccess.audit
risk_level: R0_read_only
inputs:
  service_resource_id: /subscriptions/.../storageAccounts/...
  consumer_scope_id: /subscriptions/.../virtualNetworks/...
outputs:
  private_endpoint_ids
  endpoint_connection_state
  private_dns_zone
  resolved_private_addresses
  public_network_state
  positive_test_evidence
  negative_test_evidence
  findings
  evidence_refs
```

```
capability: cloud.service.publicExposure.detect
risk_level: R0_read_only
inputs:
  scope_id: /subscriptions/.../resourceGroups/...
outputs:
  public_ip_resources
  internet_facing_frontends
  public_service_endpoints
  implicit_or_explicit_egress_paths
  permissive_network_rules
  stale_dns_or_endpoint_objects
  estimated_network_cost_drivers
  remediation_candidates
  evidence_refs
```

```
capability: network.path.explain
risk_level: R0_read_only
inputs:
  source_resource_id
  destination_fqdn
  protocol: tcp
  port: 443
outputs:
  resolution_chain
  resolved_address
  effective_route
```

```
effective_security_controls
endpoint_state
service_network_policy
authorization_boundary
likely_failure_domain
evidence_refs
```

Control-plane rule. A future network.privateAccess.enforce capability is a high-risk change tool. It can break production access, backups, monitoring, build pipelines, or trusted-service integrations. It should never be merged with the read-only audit capability.

7.16 Key Takeaways

- Cloud networking is a managed set of reachability claims, not merely a topology diagram.
- Name resolution, routing, segment policy, endpoint state, service-front-door policy, identity, and authorization are separate decisions.
- A private IP or private endpoint proves a path choice; it does not grant data permission or automatically close the public endpoint.
- Address planning must account for future peering, transit, Kubernetes ranges, hybrid networks, disaster recovery, and tenant boundaries.
- The absence of a public IP closes only one exposure surface. Ingress, egress, east-west, managed-service, DNS, and management-plane paths must be reviewed separately.
- NAT provides outbound translation, not inbound publication or application-aware inspection.
- Layer 4 load balancing, Layer 7 routing/WAF, global edge delivery, and centralized firewall inspection solve different problems and can be composed.
- Service endpoints, gateway endpoints, and private endpoints are not interchangeable. Their address, route, policy, DNS, cost, and exfiltration properties differ.
- Private DNS is part of the security and reliability boundary. Use the provider-supported FQDN and prove resolution from the real client context.
- Peering is usually direct and nontransitive; large or hybrid topologies require explicit transit, route propagation, and inspection design.
- Configuration inventory, active connection tests, flow records, and control-plane audit logs provide complementary evidence.
- Network paths have explicit costs and failure domains. Optimization must preserve required reachability, security, performance, and recovery outcomes.
- Multicloud portability starts with engineering intent and acceptance criteria, not product-name matching.
- Lab 6 is complete only when the private path succeeds, the public path fails, endpoint and DNS state are recorded, and cleanup proves zero remainder.

- CloudDesk may reuse bounded audit and explanation capabilities, while network mutation remains a separately governed high-risk tool.

References and Official Documentation

[1] National Institute of Standards and Technology. The NIST Definition of Cloud Computing. NIST SP 800-145, 2011. https://doi.org/10.6028/NIST.SP.800-145

[2] National Institute of Standards and Technology. Zero Trust Architecture. NIST SP 800-207, 2020. https://doi.org/10.6028/NIST.SP.800-207

[3] IETF. Address Allocation for Private Internets. RFC 1918, 1996. https://www.rfc-editor.org/rfc/rfc1918

[4] IETF. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632, 2006. https://www.rfc-editor.org/rfc/rfc4632

[5] IETF. A Border Gateway Protocol 4 (BGP-4). RFC 4271, 2006. https://www.rfc-editor.org/rfc/rfc4271

[6] IETF. Virtual eXtensible Local Area Network (VXLAN). RFC 7348, 2014. https://www.rfc-editor.org/rfc/rfc7348

[7] Microsoft. Azure Virtual Network documentation. Microsoft Learn. https://learn.microsoft.com/en-us/azure/virtual-network/

[8] Microsoft. Secure your Azure Virtual Network deployment. Updated August 13, 2026. https://learn.microsoft.com/en-us/azure/virtual-network/secure-virtual-network

[9] Microsoft. Azure Virtual Network FAQ. Updated July 8, 2026. https://learn.microsoft.com/en-us/azure/virtual-network/virtual-networks-faq

[10] Microsoft. Azure virtual network traffic routing. Updated March 31, 2026. https://learn.microsoft.com/en-us/azure/virtual-network/virtual-networks-udr-overview

[11] Microsoft. Azure network security groups overview. Updated July 15, 2025. https://learn.microsoft.com/en-us/azure/virtual-network/network-security-groups-overview

[12] Microsoft. Azure Virtual Network peering. Updated August 12, 2026. https://learn.microsoft.com/en-us/azure/virtual-network/virtual-network-peering-overview

[13] Microsoft. Default outbound access in Azure. Updated January 30, 2026. https://learn.microsoft.com/en-us/azure/virtual-network/ip-services/default-outbound-access

[14] Microsoft. What is Azure NAT Gateway? Updated July 29, 2026. https://learn.microsoft.com/en-us/azure/nat-gateway/nat-overview

[15] Microsoft. What is Azure Load Balancer? Microsoft Learn. https://learn.microsoft.com/en-us/azure/load-balancer/load-balancer-overview

[16] Microsoft. What is Azure Application Gateway? Updated December 16, 2025. https://learn.microsoft.com/en-us/azure/application-gateway/overview

[17] Microsoft. What is Azure Application Gateway v2? Updated April 10, 2026. https://learn.microsoft.com/en-us/azure/application-gateway/overview-v2

[18] Microsoft. What is Azure Front Door? Updated April 24, 2026. https://learn.microsoft.com/en-us/azure/frontdoor/front-door-overview

[19] Microsoft. Azure Private Link documentation. https://learn.microsoft.com/en-us/azure/private-link/

[20] Microsoft. What is a private endpoint? Microsoft Learn. https://learn.microsoft.com/en-us/azure/private-link/private-endpoint-overview

[21] Microsoft. Azure Private Endpoint private DNS zone values. Updated August 11, 2026. https://learn.microsoft.com/en-us/azure/private-link/private-endpoint-dns

[22] Microsoft. Azure Private Endpoint DNS integration scenarios. Updated September 30, 2025. https://learn.microsoft.com/en-us/azure/private-link/private-endpoint-dns-integration

[23] Microsoft. Azure virtual network service endpoints. Updated July 8, 2026. https://learn.microsoft.com/en-us/azure/virtual-network/virtual-network-service-endpoints-overview

[24] Microsoft. Azure Storage network security overview. Updated July 16, 2025. https://learn.microsoft.com/en-us/azure/storage/common/storage-network-security-overview

[25] Microsoft. Set the default public network access rule for an Azure Storage account. Updated August 26, 2025. https://learn.microsoft.com/en-us/azure/storage/common/storage-network-security-set-default-access

[26] Microsoft. Azure Storage firewall rules and network access control. Updated July 6, 2026. https://learn.microsoft.com/en-us/azure/storage/common/storage-network-security

[27] Microsoft. Azure Storage firewall guidelines and limitations. Updated July 16, 2025. https://learn.microsoft.com/en-us/azure/storage/common/storage-network-security-limitations

[28] Microsoft. Azure Network Watcher overview. Updated February 25, 2026. https://learn.microsoft.com/en-us/azure/network-watcher/network-watcher-overview

[29] Microsoft. Connection troubleshoot overview. Updated April 8, 2026. https://learn.microsoft.com/en-us/azure/network-watcher/connection-troubleshoot-overview

[30] Microsoft. Virtual Network Flow Logs overview. Updated June 25, 2026. https://learn.microsoft.com/en-us/azure/network-watcher/vnet-flow-logs-overview

[31] Microsoft. Manage Virtual Network Flow Logs. Updated January 22, 2026. https://learn.microsoft.com/en-us/azure/network-watcher/vnet-flow-logs-manage

[32] Microsoft. Azure VPN Gateway documentation. https://learn.microsoft.com/en-us/azure/vpn-gateway/

[33] Microsoft. Azure ExpressRoute documentation. https://learn.microsoft.com/en-us/azure/expressroute/

[34] Microsoft. Azure Virtual WAN documentation. https://learn.microsoft.com/en-us/azure/virtual-wan/

[35] Microsoft. Azure Firewall documentation. https://learn.microsoft.com/en-us/azure/firewall/

[36] Microsoft. Azure DDoS Protection documentation. https://learn.microsoft.com/en-us/azure/ddos-protection/

[37] Alibaba Cloud. Virtual Private Cloud documentation. https://www.alibabacloud.com/help/en/vpc/

[38] Alibaba Cloud. PrivateLink endpoints - access Alibaba Cloud services privately. Updated May 27, 2026. https://www.alibabacloud.com/help/en/privatelink/create-and-manage-endpoints/

[39] Alibaba Cloud. VPC private access to Alibaba Cloud services. Updated May 27, 2026. https://www.alibabacloud.com/help/en/vpc/vpc-connection-to-cloud-service

[40] Alibaba Cloud. VPC flow logs. Updated June 21, 2026. https://www.alibabacloud.com/help/en/vpc/vpc-flow-logs

[41] Alibaba Cloud. VPC monitoring and logging. Updated April 1, 2026. https://www.alibabacloud.com/help/en/vpc/monitoring-and-logging

[42] Alibaba Cloud. VPC NAT Gateway. Updated May 28, 2026. https://www.alibabacloud.com/help/en/nat-gateway/user-guide/what-is-a-vpc-nat-gateway/

[43] Tencent Cloud. Virtual Private Cloud overview. https://www.tencentcloud.com/document/product/215/535

[44] Tencent Cloud. Tencent Cloud Network overview. https://www.tencentcloud.com/document/product/215/47911

[45] Tencent Cloud. Private Link product overview. https://www.tencentcloud.com/document/product/1216/57917

[46] Tencent Cloud. Cloud Connect Network. https://www.tencentcloud.com/document/product/215/40089

[47] Tencent Cloud. Flow Logs. https://www.tencentcloud.com/document/product/215/35523

[48] Tencent Cloud. Route tables overview. https://www.tencentcloud.com/document/product/215/31810

[49] Volcengine. Virtual Private Cloud documentation and VPC overview. Updated June 18, 2026. https://docs.volcengine.com/docs/6396/733270?lang=en

[50] Volcengine. Network FAQ - PrivateLink and VPC connectivity. Updated June 18, 2026. https://docs.volcengine.com/docs/6396/67801?lang=en

[51] Volcengine. NAT Gateway documentation. https://docs.volcengine.com/docs/6404/106344?lang=en

[52] Volcengine. VPC flow log service documentation. https://www.volcengine.com/docs/6401/2479863

[53] Volcengine. CloudTrail supported products and control-plane scope. Updated August 28, 2025. https://www.volcengine.com/docs/7015/1142171

Part II / Chapter 8

Storage, Data Lakes, and Event-Driven Data Paths

How Namespace, Tiering, Durability, Metadata, Eventing, and Evidence Turn Stored Bytes into Governed Data Products

Cloud storage is not a cheap container for bytes. It is a family of resource contracts that determine how data is named, accessed, retained, replicated, transformed, observed, and retired. A data lake adds shared namespace, metadata, policy, and compute coordination. Event-driven paths turn changes in stored data into reliable work only when notifications, idempotency, and evidence are designed together.

In This Chapter

- Treat object, block, and file storage as different control-plane and data-plane contracts rather than interchangeable capacity.
- Model a storage resource as a graph of namespace, identity, network, lifecycle, protection, event, telemetry, and cost objects.
- Separate durability and replication from backup, point-in-time recovery, and validated restoration procedures.
- Design data lifecycle policies from classification, access evidence, retention obligations, and economic intent.
- Understand a data lake as a governed data estate with zones, catalog, lineage, quality, identity, and compute coordination.
- Build event-driven data paths that tolerate duplicate notifications, delayed delivery, reordering, and partial failure.
- Connect storage and data-lake decisions to AI-native ingestion, retrieval, evaluation, observability, and FinOps.
- Complete an independent Azure lab that creates, verifies, and retires an object-storage lifecycle policy, then translate the engineering intent to Alibaba Cloud, Tencent Cloud, and Volcengine.

Central claim. Object, block, file, and data-lake services are distinct managed contracts. An AI-native data path becomes reliable only when namespace, identity, placement, lifecycle, event semantics, evidence, and economics are designed together.

Modern application diagrams often draw one cylinder labeled “storage.” That symbol is convenient and dangerously incomplete. A database volume, a shared file system, an object corpus, a lakehouse table, a backup repository, and a model artifact may all contain bytes, but they do not expose the same unit of access, latency profile, namespace, concurrency model, retention semantics, failure behavior, or cost surface. Treating them as one category produces architectures that are easy to draw and difficult to operate.

The control-plane view corrects this abstraction error. Instead of asking only where data is stored, it asks which resource owns the namespace, who can change policy, how data-plane access is authorized, what failures replication covers, how old data is tiered or deleted, which events are emitted, what evidence proves a transition occurred, and how cost is attributed. These questions make storage part of software engineering rather than a post-deployment capacity decision.

8.1 Storage Is a Family of Contracts, Not One Service

The first design decision is not “which storage product?” It is “which access and lifecycle contract does the workload require?” The correct answer usually begins with the unit that the application reads and writes.

Table 8-1 Storage contracts should be selected by semantics, not by the generic presence of data.

Contract	Primary unit	Typical access	Strengths	Common misuse
Object storage	Object addressed by key, path-like name, and metadata	HTTPS/REST, SDK, multipart transfer	Massive scale, low administration, tiering, event source, analytics corpus	Mounted as if it were a low-latency POSIX disk or used for chatty small random writes
Block storage	Volume attached to one or more compute nodes under defined access modes	Filesystem, database engine, raw device I/O	Low-latency random I/O, predictable performance tiers, snapshots	Treated as a shared data service without a cluster/filesystem protocol

Contract	Primary unit	Typical access	Strengths	Common misuse
File storage	File and directory in a shared hierarchy	SMB, NFS, filesystem APIs	Lift-and-shift compatibility, concurrent clients, familiar ACLs	Used as an unlimited analytics object store without namespace and metadata planning
Data-lake storage	Object/file plus hierarchical namespace, metadata, governance, and analytics integration	Object APIs, filesystem APIs, query engines, table formats	Large-scale analytics, shared data products, decoupled storage and compute	Assumed to be "a bucket with folders" while catalog, lineage, quality, and access remain unmanaged

Storage is a family of contracts

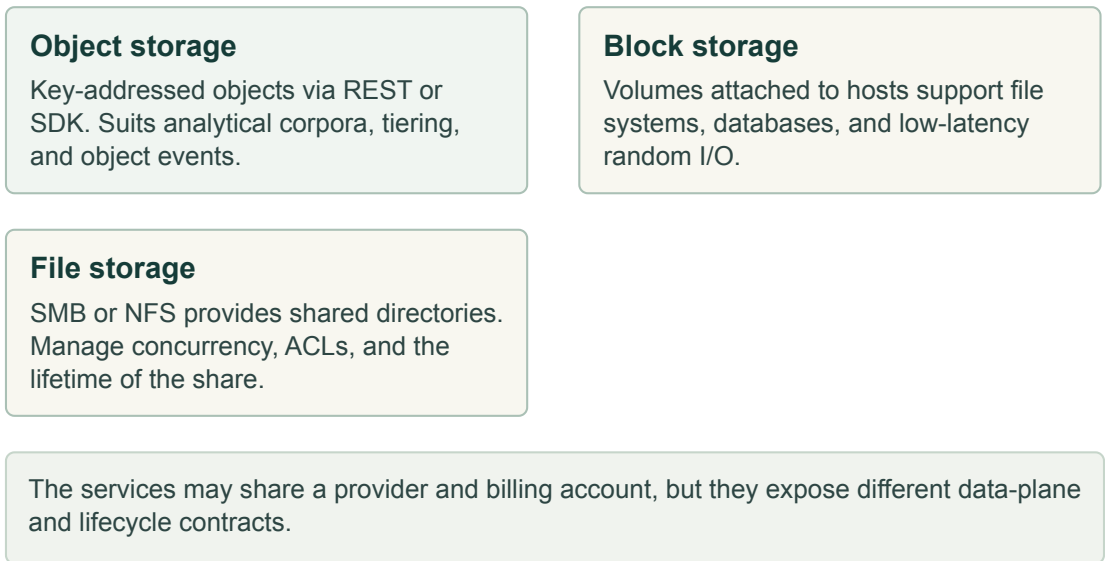


Figure 8-1 Object, block, and file services expose distinct resource and data-plane contracts.

A service boundary also determines what the provider manages. With a managed disk, the provider handles physical media, replication inside the selected redundancy contract, and volume provisioning; the customer still owns the filesystem, database consistency, application backup, and restore validation. With object storage, the provider manages the distributed object service, while the customer owns object naming, identity design, lifecycle filters, metadata quality, event consumers, and recovery semantics. A higher abstraction removes some work, but it never removes the need to define intent.

Design rule. Do not compare storage products until you have written the required namespace, access pattern, consistency, durability, recovery, retention, eventing, security, and cost contracts.

8.2 The Storage Resource Graph

A storage account or bucket is only the root of a larger resource graph. Its data-plane endpoints expose objects, files, queues, or tables. Control-plane objects define encryption, redundancy, network access, private endpoints, diagnostic settings, lifecycle rules, immutable retention, replication, event subscriptions, role assignments, and budget scope. The operational behavior emerges from the graph, not from the account alone.

A storage resource is a managed graph

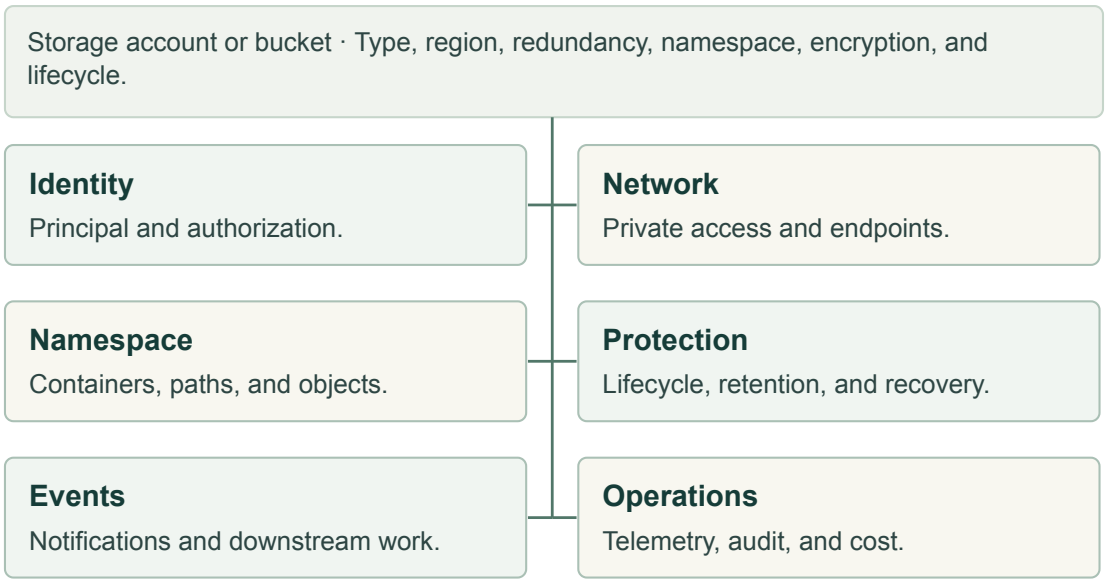


Figure 8-2 The storage service is surrounded by identity, network, lifecycle, event, evidence, and cost resources.

Table 8-2 A storage design is a resource graph with multiple control and evidence surfaces.

Resource concern	Representative objects	Engineering question
Identity and authorization	Role assignment, access policy, service principal, managed identity, ACL	Which principal can manage the resource, and which principal can read or write data?

Resource concern	Representative objects	Engineering question
Network boundary	Firewall, private endpoint, private DNS, service endpoint, routing policy	Can the data plane be reached only through the intended network path?
Namespace	Account/bucket, container, filesystem, directory, object key, share	What is the durable name and ownership boundary for each data product?
Lifecycle and protection	Tier rule, retention rule, soft delete, versioning, immutability, replication	When may data change, move, expire, recover, or be purged?
Eventing	Event subscription, topic, queue, filter, dead-letter destination	How does a committed change become reliable downstream work?
Evidence and economics	Metrics, logs, inventory, access records, cost export, budget	How can operators prove state, attribute cost, and detect drift?

The graph must preserve two separate authority paths. Management-plane rights create accounts, alter replication, change network policy, or install lifecycle rules. Data-plane rights list, read, write, tag, delete, or restore data. Granting one does not automatically grant the other. This separation is essential for least privilege: a deployment identity may create a storage resource but should not necessarily read its contents, while an ingestion worker may write objects without being able to disable immutability or public-access restrictions.

8.3 Namespace and Access Semantics

Namespace is architecture. It determines the durable names on which applications, policies, events, and governance depend. A poor namespace becomes a permanent coupling mechanism because lifecycle filters, access controls, data products, and downstream processors all begin to depend on it.

8.3.1 Flat Object Names and Hierarchical Namespaces

Object stores commonly expose a flat keyspace even when names contain slash characters. Prefixes can imitate directories, but directory rename or delete may still require operations across many objects. A hierarchical namespace makes directories first-class and can support atomic directory operations and filesystem-style authorization. Azure Data Lake Storage adds a hierarchical namespace to Blob Storage so analytics engines can use directory semantics while retaining object-scale economics and tiering.

A hierarchy is not automatically a good data model. It should encode stable dimensions such as source, data product, date partition, sensitivity, or processing state—not temporary team names, usernames, or implementation details. Stable partitions support query pruning and lifecycle policy; unstable paths create expensive rewrites and brittle permissions.

Table 8-3 Namespace elements should reflect stable governance and query dimensions.

Namespace decision	Good candidate	Risky candidate
Top-level ownership	Business data domain or durable product	Current project team or individual user
Processing zone	raw, quarantined, curated, serving	temporary1, final-final, copy
Time partition	event date with explicit granularity	upload time mixed with business time
Tenant or security boundary	Explicit tenant identifier when isolation is designed end to end	Tenant hidden only in filename while authorization is shared
Schema/version	Versioned data product or table metadata	Implicit format guessed by consumers

8.3.2 ACLs, RBAC, and Object-Level Conditions

Authorization can apply at several layers: cloud resource scope, container or share, directory hierarchy, object tag, signed URL, and application tenancy. These layers solve different problems. Cloud RBAC is suitable for role assignment at durable resource scopes. Directory ACLs support filesystem-style delegation. Object tags can drive lifecycle filters or conditional access. Short-lived signed access can delegate a narrow data-plane action without granting broad account authority. None of these should be confused with application-level tenant authorization.

Least-privilege boundary. Do not place long-lived account keys in application configuration when workload identity or short-lived delegated access is available. A storage key often represents an account-wide bypass around finer-grained authorization.

8.4 Performance Is a Workload Contract

Capacity in gigabytes is an incomplete storage requirement. Storage performance is shaped by object size, I/O size, concurrency, read/write ratio, random versus sequential access, metadata operations, locality, network path, and service limits. A system can have sufficient capacity and still fail because it saturates request rate, metadata throughput, disk queue depth, or egress bandwidth.

Table 8-4 Storage performance must be expressed in workload-specific terms.

Dimension	Object storage	Block storage	File storage
Scaling unit	Objects, partitions, request distribution	Volume performance tier and host attachment	Share/filesystem throughput and metadata scale
Best fit	Large immutable/sequential objects, parallel readers, analytics corpus	Database pages, boot/data volumes, low-latency random I/O	Shared application files, user shares, legacy filesystem expectations
Common bottleneck	Hot prefix, many tiny objects, serial upload, egress	Provisioned IOPS/throughput, VM limit, queue depth	Metadata operations, protocol latency, share throughput
Primary optimization	Partition keys, multipart upload, batching, parallelism	Right-size tier, stripe when justified, cache, align host limits	Directory design, client concurrency, caching, suitable protocol/tier

Large object uploads commonly use multipart or block-based transfer so clients can retry parts independently and exploit parallelism. This design is not merely a performance trick. It changes failure recovery, checksum validation, and observability: operators should know whether an incomplete multipart upload exists, which parts committed, and whether the final object represents the expected content.

For analytics, decoupling storage from compute allows multiple engines to read the same corpus. That flexibility makes metadata, partitioning, file size, compression, and table format more important. Thousands of tiny files can convert a cheap object store into an expensive metadata and scheduling problem. The storage bill may remain low while query planning and compute costs rise.

8.5 Durability, Availability, Replication, Backup, and Recovery

Storage reliability vocabulary is often collapsed into the word “backup.” The terms describe different mechanisms and promises. Durability is the probability that committed data is not lost. Availability is whether the service can satisfy requests at a given time. Replication maintains additional copies. Versioning retains prior object states. Soft delete delays irreversible deletion. Backup creates a separately managed recovery asset or restore point. Disaster recovery defines how service is restored after a larger failure. These mechanisms overlap, but none is a complete substitute for the others.

Replication protects availability; recovery protects meaning

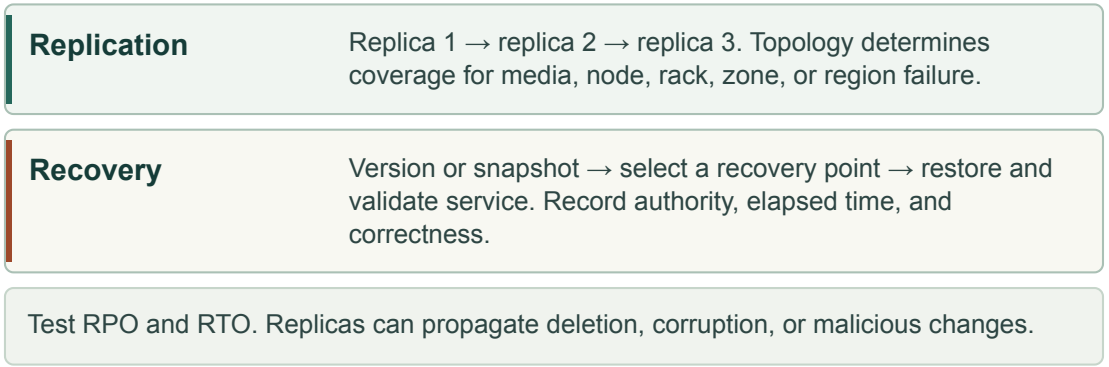


Figure 8-3 Replication protects a failure topology; a recovery procedure restores a meaningful application state.

Table 8-5 Reliability mechanisms answer different failure questions.

Mechanism	Primary protection	Does not by itself prove
Synchronous regional replication	Media, node, rack, or zone failure within the stated topology	Recovery from a valid earlier logical state
Asynchronous cross-region replication	Regional disaster with a nonzero replication lag	Zero data loss, application consistency, or instant failover
Versioning / soft delete	Accidental overwrite or deletion within retention window	Application-consistent recovery across multiple resources
Snapshot	Point-in-time volume or object state	That the snapshot is uncorrupted, bootable, or transactionally consistent
Backup and restore procedure	Defined recovery point, retention, and isolated restore path	That recovery meets RPO/RTO unless it is regularly tested
Immutable retention	Modification or deletion during a defined retention period	Confidentiality, correctness of original data, or legal compliance by itself

A useful reliability record states the protected failure domain, replication mode, expected RPO, recovery procedure, authorization required to initiate recovery, and verification criteria. For a data lake, a recovery test may need to restore not only objects but also catalog metadata, table snapshots, access policies, keys, event subscriptions, and downstream indexes. Recovering bytes without recovering metadata can leave a technically durable but unusable system.

Reliability rule. Replication is a topology; recovery is an operational procedure. Treat RPO and RTO as tested properties of the complete data service, not as inferred properties of a storage SKU.

8.6 Tiering, Retention, Versioning, and Immutability

Lifecycle management translates business value over time into storage actions. The familiar hot–cool–cold–archive progression is only one policy shape. Some data remains hot because it supports active transactions. Some moves quickly to archive because its value is evidentiary rather than operational. Some cannot be deleted until a retention obligation expires. Some should never enter archive because recovery latency would violate an SLO.

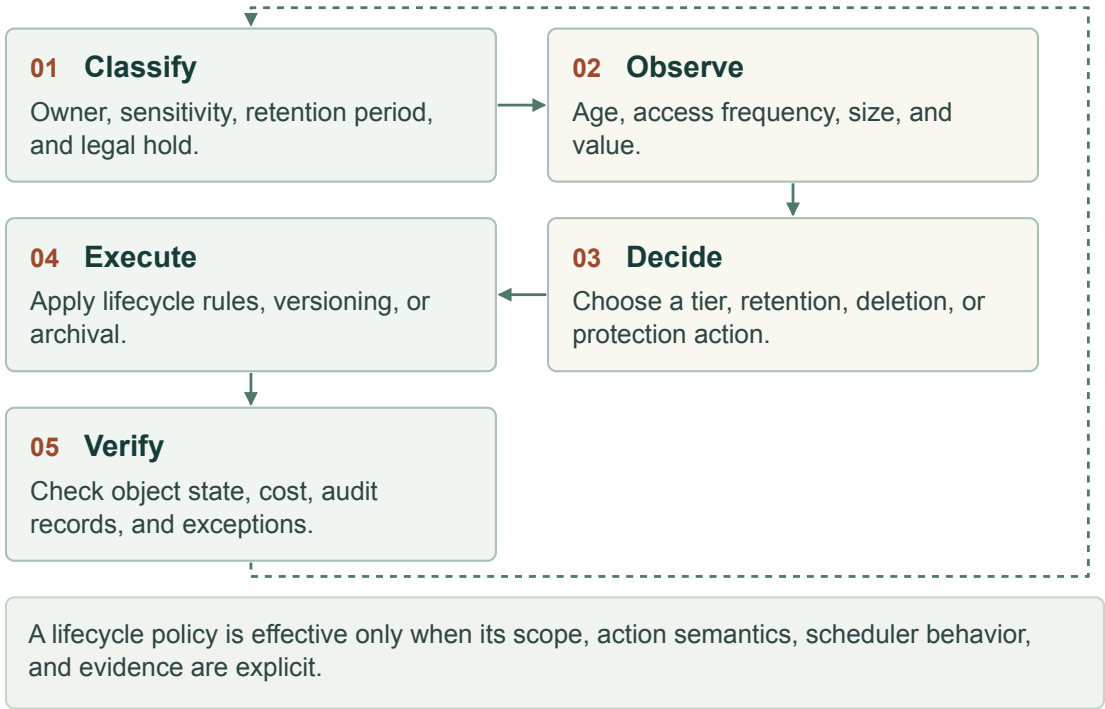


Figure 8-4 Lifecycle management is a feedback loop from classification and observed use to policy execution and evidence.

Table 8-6 Lifecycle features must be tied to explicit business and operational semantics.

Lifecycle concept	Question	Common failure
Access tier	How frequently and how quickly must data be retrieved?	Optimizing storage price while ignoring retrieval, transaction, or early-deletion cost
Retention	How long must data remain available or preserved?	Encoding a legal or business rule only in an informal naming convention
Versioning	Which prior object states should be recoverable?	Unlimited version growth without cost attribution or expiry
Soft delete	How long should accidental deletion remain reversible?	Assuming soft-deleted data has already stopped incurring storage cost
Immutability / WORM	During which interval must modification and deletion be blocked?	Locking policy without testing legitimate correction, legal hold, and eventual purge
Lifecycle filter	Which objects are in policy scope?	Broad prefix or tag errors that affect unrelated data products
Lifecycle scheduler	When will the provider evaluate and apply the rule?	Using an asynchronous policy as if it were a real-time workflow trigger

Lifecycle policy is deliberately asynchronous. It is optimized for fleet-scale background evaluation, not immediate workflow execution. A policy may take time to discover eligible objects and apply transitions. Immediate verification should therefore inspect the policy object, filters, and current object state; extended verification observes the provider-applied transition later. When a business process requires immediate action after an upload, use an event-driven workflow rather than a lifecycle rule.

Lifecycle and immutability can conflict. A delete action cannot remove an object while immutable retention or legal hold prevents deletion. Versioning and soft delete can also change the economic effect of a “delete” action because deleted content may continue to be retained. Governance requires a precedence model: preservation controls override cost-optimization controls, and exceptions must have owners and expiry dates.

8.7 A Data Lake Is a Governed Data Estate

A data lake is often introduced as inexpensive storage for structured and unstructured data. That description is correct and insufficient. A production data lake is a governed estate in which storage, namespace, metadata, access, lineage, quality, retention, and com-

pute authorization form a common operating model. Without those controls, the lake becomes a low-cost accumulation of opaque files—a data swamp with high discovery and compliance cost.

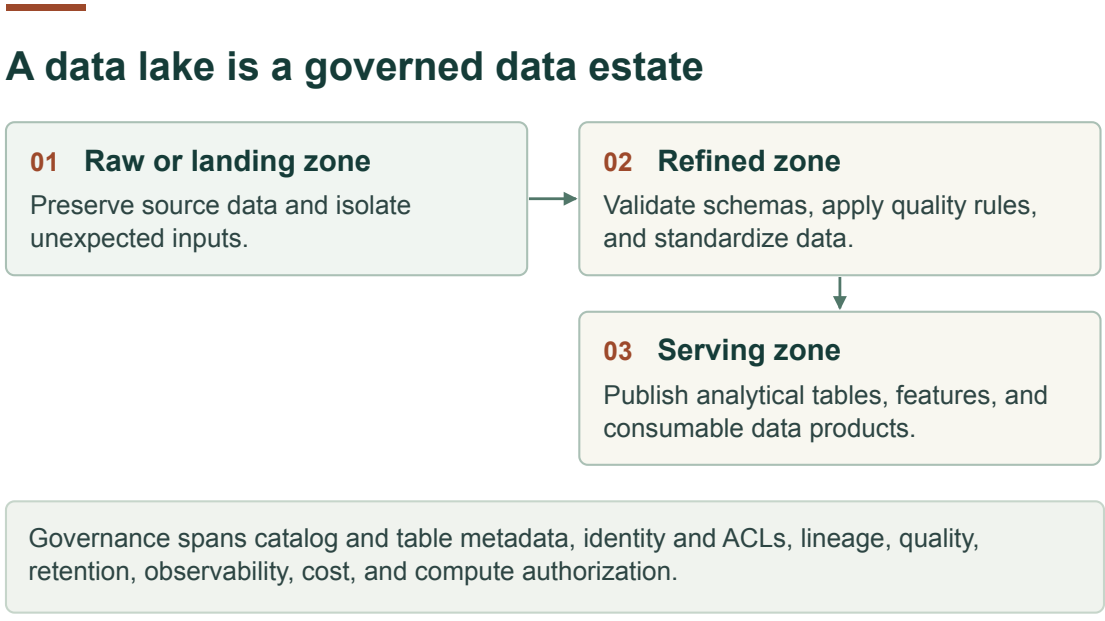


Figure 8-5 Raw, curated, and serving zones require a shared governance plane.

8.7.1 Zones Express Trust and Readiness

The raw or landing zone preserves source fidelity and provides a recoverable ingestion boundary. A quarantine area isolates files that fail security, schema, or quality checks. The curated zone normalizes schema, resolves quality defects, and establishes governed datasets. The serving zone exposes data products optimized for analytics, machine learning, retrieval, or application consumption. These zones are responsibility boundaries, not merely folders.

Table 8-7 Data-lake zones encode different trust, lifecycle, and consumer contracts.

Zone	Primary promise	Typical controls	Exit condition
Raw / landing	Preserve the received source and ingestion evidence	Write-once pattern, source metadata, checksum, restricted readers, retention	Object is accepted, quarantined, or routed for processing
Curated	Conform data to a governed schema and quality contract	Validation, lineage, data classification, deduplication, partitioning	Dataset passes defined quality and ownership checks

Zone	Primary promise	Typical controls	Exit condition
Serving	Deliver a stable data product for a consumer workload	Table/version contract, performance layout, product ACL, SLO, cost attribution	Consumer can discover, authorize, query, and interpret the product
Archive	Preserve low-access historical or evidentiary data	Retention, immutability, retrieval procedure, legal hold, restore test	Retention expires and authorized purge is verified

8.7.2 Catalog, Lineage, Quality, and Ownership

A catalog makes data discoverable and associates physical locations with schemas, classifications, owners, and policies. Lineage records how data arrived and which transformations produced a downstream asset. Quality rules make readiness measurable. Ownership determines who approves access, resolves defects, changes retention, and bears cost. These controls should be queryable resources or evidence, not fields maintained only in a spreadsheet.

A lake also requires compute governance. A query engine, Spark job, ETL service, feature pipeline, and indexing worker may read the same data under different identities and budgets. Decoupled storage does not imply unconstrained compute. The platform must control which engine can scan which zone, where temporary results are written, how egress is limited, and how cost is assigned to the consuming workload.

8.8 Open Table Formats Turn Files into Managed Tables

Parquet, ORC, and other columnar files improve analytics efficiency, but a collection of files is not yet a table with transactional meaning. Open table formats such as Apache Iceberg, Delta Lake, and Apache Hudi add metadata and commit protocols that describe snapshots, schema, partition evolution, file membership, and concurrent change. They make object storage usable as a managed analytical table without requiring every engine to infer state from directory listings.

Table 8-8 A table format adds a transactional metadata layer above object files.

Capability	Why raw files are insufficient	What a table format adds
Atomic publication	Readers can observe a partially written file set	Commit metadata points readers to a complete table snapshot
Time travel / rollback	Object versions do not necessarily encode table-level consistency	Snapshot history identifies coherent table states

Capability	Why raw files are insufficient	What a table format adds
Schema evolution	Consumers guess compatibility from files	Explicit schema and field identity govern change
Partition evolution	Physical directory layout becomes a permanent query contract	Metadata separates logical partitioning from file paths
Concurrent writers	Directory listing and rename conventions are fragile	Optimistic concurrency or transaction protocol coordinates commits
Maintenance	Obsolete files accumulate without a table-level view	Compaction, retention, and orphan cleanup operate against metadata

Open format does not mean governance-free. Catalog registration, access policy, encryption, retention, snapshot expiry, compaction, and cost remain platform responsibilities. Portability also has levels: multiple engines may understand the format, yet catalog APIs, identity integration, optimization features, and operational tooling can still differ by provider. Multicloud design should record those non-equivalences rather than declaring complete portability.

8.9 Event-Driven Data Paths Separate Data and Event Planes

Object storage is a natural event source because the arrival, update, copy, or deletion of an object often starts downstream work. Examples include thumbnail generation, malware scanning, ETL, metadata extraction, embedding generation, index refresh, notification, and compliance archiving. The event is not the data. It is a notification that authoritative state changed and may need processing.

Event-driven data paths have two planes

Data plane	Producer writes → object commits → processor reads → derived object or index update.
Event plane	Create or delete event → broker filters, retries, or dead-letters → subscriber verifies object state and starts processing.

At-least-once delivery means the subscriber must tolerate duplicates and verify object state.

Figure 8-6 The data plane commits and reads objects; the event plane routes notifications and retries work.

8.9.1 At-Least-Once Delivery and Idempotency

Managed event systems commonly use at-least-once delivery. A matching event is retried when a subscriber does not acknowledge it or when delivery fails, so duplicate notifications can occur. Delivery may also be delayed or reordered. A correct consumer must therefore be idempotent: processing the same logical event more than once should not corrupt state or create duplicate business effects.

Table 8-9 Event-driven data paths must treat retries and partial failure as normal conditions.

Failure mode	Required design response	Evidence to retain
Duplicate delivery	Idempotency key derived from event ID, object version/ETag, and operation	Deduplication decision and prior completion record
Out-of-order events	Read current object state or compare version/generation before acting	Observed version, expected version, rejected stale event
Partial processing	Checkpoint stages and make side effects transactional or compensatable	Stage status, output identifiers, retry count
Poison event	Bound retries and route to dead-letter storage/queue	Original event, error, policy, owner, replay decision
Missing downstream capacity	Buffer with queue or durable broker; apply backpressure	Queue age, depth, scale action, SLO impact

Failure mode	Required design response	Evidence to retain
Event/data race	Retry object read or react only to final commit event	Read attempts, final object properties, checksum

8.9.2 Event Filters Are Part of the Contract

Filters based on event type, subject prefix, suffix, object tags, or container scope reduce noise and protect downstream capacity. A filter should be reviewed like code because an overly broad subscription can invoke expensive processing for irrelevant objects, while an overly narrow filter silently omits required work. The event subscription, delivery identity, retry policy, dead-letter destination, and consumer checkpoint store are all resource-level parts of the path.

Event rule. Use lifecycle management for age-based background transitions. Use event-driven workflows for prompt processing after a data change. They solve different timing and evidence requirements.

8.10 AI-Native Data Paths

AI-native applications extend the data path beyond storage and analytics. Documents are parsed, classified, chunked, embedded, indexed, evaluated, retrieved, cited, and sometimes fed back into model or agent behavior. Each transition creates a new managed asset with its own owner, permissions, version, cost, and evidence.

An AI-native data path is a chain of governed assets

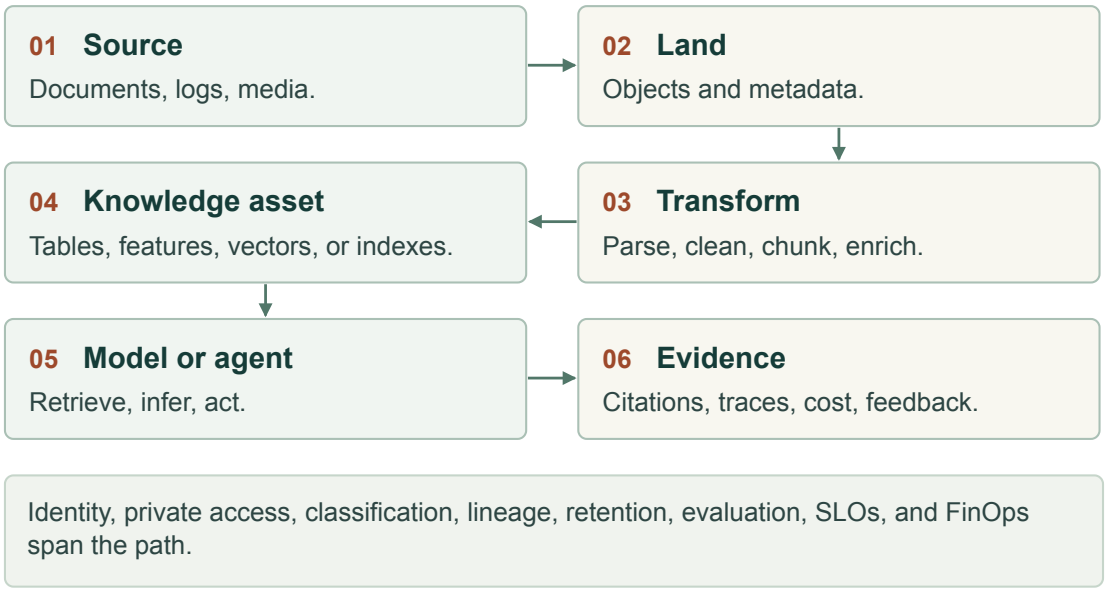


Figure 8-7 AI-native data paths connect source objects to governed knowledge assets, models, agents, and evidence.

Table 8-10 An AI data path creates multiple governed assets, not one "RAG pipeline."

Stage	Managed asset	Control-plane questions
Landing	Source object plus provenance metadata	Who may ingest? Is the object private, classified, checksummed, and retained?
Parse and normalize	Extracted text, media transcript, structured record	Which parser/version produced it? How are failures and sensitive fields handled?
Chunk and enrich	Chunks, entities, labels, summaries	What policy controls chunk size, metadata, language, and enrichment model?
Embed and index	Embedding vectors, search documents, table/features	Which model/version, tenant, ACL, index schema, refresh schedule, and cost apply?
Retrieve and reason	Grounding set, prompt context, model/agent run	Was authorization applied before retrieval? Are citations and trace links preserved?
Evaluate and improve	Evaluation set, failure case, feedback event	Which dataset and version justify promotion, rollback, or policy change?

Storage decisions directly affect AI quality. Poor provenance makes citations weak. Unstable paths break index refresh. Missing tenant or ACL metadata can cause cross-tenant retrieval. Unbounded versioning and embeddings inflate cost. Archive transition can make source reprocessing unexpectedly slow. A knowledge system is therefore downstream of the storage governance described in this chapter.

The raw/curated/serving pattern remains useful. Raw preserves the original document and ingestion evidence. Curated contains validated text and metadata. Serving contains the search index, vector representation, feature table, or other consumer-optimized asset. Keeping these stages explicit makes reprocessing, model migration, deletion, and evidence collection tractable.

8.11 Security, Privacy, and Compliance

Storage systems often contain the broadest and longest-lived collection of organizational data. Their security boundary must combine identity, network restriction, encryption, classification, retention, audit, and recovery. Encryption at rest is important but does not compensate for public exposure, excessive data-plane roles, shared credentials, or uncontrolled export.

Table 8-11 Storage security is a layered operational contract.

Control	Purpose	Verification question
Workload identity	Remove long-lived shared credentials	Can the workload obtain only the required data-plane role?
Private connectivity	Keep data-plane traffic on approved network paths	Does the public endpoint fail while private name resolution and access succeed?
Encryption and key policy	Protect stored data and govern key authority	Who can use, rotate, revoke, and recover keys?
Classification and DLP	Apply controls based on sensitivity	Are sensitive fields labeled and prevented from unauthorized export?
Immutability / legal hold	Prevent modification or deletion during preservation period	Can an authorized test prove writes/deletes are rejected?
Access and activity evidence	Support investigation and accountability	Can operators connect principal, operation, object, time, result, and network context?
Deletion workflow	Honor retention expiry and data-subject or tenant deletion	Are replicas, versions, indexes, derived assets, and backups included in the decision?

Deletion is especially complex in AI systems. Removing a source object may also require removing parsed text, chunks, embeddings, search documents, cached contexts, evaluation examples, and derived reports. Backups and immutable retention may delay final purge. The deletion contract must distinguish logical removal, revocation from active use, retention of protected evidence, and eventual physical purge.

Compliance boundary. Provider certifications and immutable-storage features can support a control implementation. They do not by themselves prove that the organization selected the correct retention period, protected the correct data, or tested the complete evidence and purge workflow.

8.12 Storage Economics Is Multidimensional

The lowest price per stored gigabyte is rarely the lowest-cost architecture. Storage economics include capacity, operations, retrieval, data transfer, replication, metadata, backup copies, early-deletion periods, monitoring, security scanning, and the compute required to process inefficient layouts. The cost model should follow the data path.

Table 8-12 Storage cost is the sum of capacity, operations, transfer, protection, and downstream processing.

Cost surface	Typical driver	Optimization question
Stored capacity	Current objects, versions, snapshots, soft-deleted data, replicas	Which copies are intentional, attributable, and within retention?
Transactions	Reads, writes, listings, metadata and lifecycle operations	Can batching, caching, partitioning, or fewer tiny files reduce request volume?
Retrieval	Cool/cold/archive read and rehydration	Does expected access frequency justify the tier?
Data transfer	Cross-zone, cross-region, internet egress, replication	Can compute move to data or results be reduced?
Protection	Backup, replication, immutability, long retention	Which recovery or compliance objective justifies each copy?
Analytics / AI compute	Scans, parsing, embedding, indexing, compaction	Are partitions, file sizes, incremental updates, and caching efficient?
Operational overhead	Catalog, event broker, monitoring, security services	Does the control reduce risk or toil enough to justify its cost?

Lifecycle policies are a FinOps control because they change the rate paid for data over time. They should be linked to ownership and value, not applied as universal age rules. For example, moving every object to archive after 90 days may lower capacity cost while increasing recovery time, retrieval fees, and operational risk. A recommendation should include the expected access pattern, projected savings, transition and retrieval costs, early-deletion constraints, affected SLO, and rollback or exception process.

Showback should attribute cost to a durable data product or workload, not only to the shared storage account. Tags may help at the account level, but object prefixes, metadata, inventory exports, access logs, and query engine accounting may be needed for finer allocation. The allocation method must be documented because an approximate but stable rule is more useful than a precise-looking number whose basis changes monthly.

8.13 Multicloud Translation by Engineering Intent

The providers expose similar broad capability families, but product names do not prove operational equivalence. Translation should begin with the engineering intent: object semantics, hierarchy, storage classes, lifecycle filters, event delivery, data-lake metadata, private access, protection, and cost evidence. The table below is a navigation aid, not a claim that every feature or service limit is identical.

Table 8-13 Translate storage platforms by engineering intent and verify non-equivalent details before implementation.

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Object storage	Blob Storage	Object Storage Service (OSS)	Cloud Object Storage (COS)	Tinder Object Storage (TOS)
Hierarchical data lake	ADLS Gen2 hierarchical namespace	OSS with Data Lake Formation / analytics integrations	COS/CHDFS with Data Lake Compute and metadata services	TOS with AI Data Lake and analytics integrations
Block storage	Managed Disks	Elastic Block Storage / ESSD	Cloud Block Storage (CBS)	Elastic Block Storage (EBS)
File storage	Azure Files / Azure NetApp Files	NAS / CPFS	Cloud File Storage (CFS)	FileNAS / EFS families
Lifecycle/tiering	Blob lifecycle management; hot/cool/cold/archive	OSS lifecycle rules and storage classes	COS lifecycle configuration and storage classes	TOS lifecycle rules and storage types

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Event notification	Event Grid storage events	OSS event notification / EventBridge integrations	COS event notification / EventBridge / SCF integrations	TOS event notification and event services
Immutability / WORM	Immutable Blob Storage	OSS retention / WORM controls	COS object lock / retention capabilities	TOS WORM retention
Private access	Private Endpoint / Private Link	PrivateLink/VPC access patterns	PrivateLink/VPC endpoint patterns	PrivateLink for TOS and VPC access
Cross-region copy	Object replication / geo-redundancy options	OSS cross-region replication	COS cross-region replication	TOS cross-region replication

For each implementation, verify at least six non-equivalence points: the unit to which lifecycle applies; minimum storage duration and retrieval behavior; event delivery and ordering; hierarchy and ACL model; replication lag and failover semantics; and the visibility of object-level cost and access evidence. These details often matter more than API similarity.

8.14 Lab 7: Configure and Verify an Object-Storage Lifecycle Policy

This independent lab creates an Azure Storage account, a private container, data-protection settings, and a lifecycle management policy. It uploads one object inside the policy prefix and one outside it, verifies the rule through the control plane, manually changes one object to the cool tier to demonstrate immediate tier semantics, collects an evidence packet, and removes all resources.

Why manual tiering is included. Lifecycle evaluation is asynchronous and is not designed as a sub-hour workflow. The lab verifies policy intent immediately and uses a direct tier operation to prove object-tier behavior. An optional extended observation can later confirm provider-applied lifecycle execution.

8.14.1 Goal and Acceptance Criteria

- Deploy all lab resources into a dedicated resource group that can be deleted independently.
- Create a Standard general-purpose v2 storage account with secure transfer required, minimum TLS 1.2, public blob access disabled, and no shared-key use in the application workflow.

- Enable blob versioning, blob soft delete, and container soft delete as distinct protection controls.
- Create a private container named data.
- Install one lifecycle rule scoped to the prefix data/logs/ that moves base blobs to progressively lower-access tiers and eventually deletes them; expire previous versions and snapshots separately.
- Upload a matching object under logs/ and a nonmatching object under reference/.
- Verify the policy object, filter, actions, resource graph, and uploaded-object properties.
- Manually move the matching object to the cool tier and prove the resulting tier through the data plane.
- Delete the resource group and prove that the lab leaves no resources behind.

8.14.2 Prerequisites, Permissions, and Cost Boundary

Table 8-14 Lab prerequisites and boundaries.

Item	Requirement
Tools	Azure CLI, Bicep, and an authenticated Azure account
Management permission	Ability to create a resource group, deploy storage resources, and create a scoped role assignment
Data permission	Storage Blob Data Contributor on the lab storage account (assigned in the template to the signed-in user)
Region	Any Azure region that supports the selected storage and access-tier features; examples use eastus
Cost class	Low, but not zero: stored bytes, write/read operations, tier change, and any retained versions may incur charges
Safety boundary	Use a new resource group; do not point the policy at an existing application or function-host storage account
Expected duration	20–35 minutes for immediate verification; lifecycle scheduler observation is optional and longer-running

8.14.3 Create the Bicep Template

Create a file named main.bicep. The template uses a globally unique storage-account name, creates the data container, enables versioning and soft delete, installs the lifecycle rule, and grants the current user data-plane access. The lifecycle prefix is relative to the storage account and therefore includes the container name: data/logs/.

```
@description('Azure region for the lab resources.')
param location string = resourceGroup().location

@description('Object ID of the signed-in user who will upload and inspect
blobs.')
param principalId string

var suffix = uniqueString(resourceGroup().id)
var storageName = 'stlake${suffix}'
var blobDataContributorRole = subscriptionResourceId(
  'Microsoft.Authorization/roleDefinitions',
  'ba92f5b4-2d11-453d-a403-e96b0029c9fe'
)

resource storage 'Microsoft.Storage/storageAccounts@2025-01-01' = {
  name: storageName
  location: location
  kind: 'StorageV2'
  sku: { name: 'Standard_LRS' }
  properties: {
    allowBlobPublicAccess: false
    allowSharedKeyAccess: true
    minimumTlsVersion: 'TLS1_2'
    supportsHttpsTrafficOnly: true
    accessTier: 'Hot'
  }
}

resource blobService 'Microsoft.Storage/storageAccounts/blobServices@2025-01-01'
= {
  parent: storage
  name: 'default'
  properties: {
    isVersioningEnabled: true
    deleteRetentionPolicy: {
      enabled: true
      days: 7
    }
    containerDeleteRetentionPolicy: {
      enabled: true
      days: 7
    }
  }
}

resource dataContainer
'Microsoft.Storage/storageAccounts/blobServices/containers@2025-01-01' = {
  parent: blobService
```

```
  name: 'data'
  properties: {
    publicAccess: 'None'
  }
}

resource lifecycle 'Microsoft.Storage/storageAccounts/managementPolicies@2025-
01-01' = {
  parent: storage
  name: 'default'
  properties: {
    policy: {
      rules: [
        {
          name: 'logs-lifecycle'
          enabled: true
          type: 'Lifecycle'
          definition: {
            filters: {
              blobTypes: [ 'blockBlob' ]
              prefixMatch: [ 'data/logs/' ]
            }
            actions: {
              baseBlob: {
                tierToCool: { daysAfterModificationGreaterThan: 30 }
                tierToCold: { daysAfterModificationGreaterThan: 90 }
                tierToArchive: { daysAfterModificationGreaterThan: 180 }
                delete: { daysAfterModificationGreaterThan: 365 }
              }
              version: {
                delete: { daysAfterCreationGreaterThan: 30 }
              }
              snapshot: {
                delete: { daysAfterCreationGreaterThan: 30 }
              }
            }
          }
        }
      ]
    }
  }
}

resource dataRole 'Microsoft.Authorization/roleAssignments@2022-04-01' = {
  name: guid(storage.id, principalId, blobDataContributorRole)
  scope: storage
  properties: {
    principalId: principalId
```

```
    principalType: 'User'
    roleDefinitionId: blobDataContributorRole
  }
}

output storageAccountName string = storage.name
output containerName string = dataContainer.name
output lifecyclePolicyId string = lifecycle.id
```

Shared-key note. The template keeps shared-key access enabled for broad compatibility with storage tooling, but the lab commands use --auth-mode login. A hardened workload design can disable shared-key access after confirming that every required client supports Microsoft Entra authorization.

8.14.4 Deploy the Resource Graph

```
LAB_RG=book-lab-07-storage-lifecycle
LOCATION=eastus
PRINCIPAL_ID=$(az ad signed-in-user show --query id -o tsv)

az group create --name "$LAB_RG" --location "$LOCATION"

az deployment group create \
  --resource-group "$LAB_RG" \
  --template-file main.bicep \
  --parameters principalId="$PRINCIPAL_ID"

STORAGE=$(az deployment group show \
  --resource-group "$LAB_RG" \
  --name main \
  --query properties.outputs.storageAccountName.value -o tsv)

echo "$STORAGE"
```

Role assignments can take a short time to propagate. If the first data-plane command returns an authorization error, wait briefly and retry rather than falling back to an account key.

8.14.5 Upload Matching and Nonmatching Objects

```
mkdir -p lab-data
printf '2026-08-26T00:00:00Z,INFO,agent-started\n' > lab-data/app.log
printf 'reference material\n' > lab-data/readme.txt
```

```
az storage blob upload \
  --account-name "$STORAGE" \
  --container-name data \
  --name logs/app.log \
  --file lab-data/app.log \
  --auth-mode login \
  --overwrite

az storage blob upload \
  --account-name "$STORAGE" \
  --container-name data \
  --name reference/readme.txt \
  --file lab-data/readme.txt \
  --auth-mode login \
  --overwrite
```

The first object matches the prefix data/logs/. The second remains outside policy scope. Keeping both objects in the same container makes the filter boundary easy to verify.

8.14.6 Verify Policy Intent Through the Control Plane

```
az storage account management-policy show \
  --account-name "$STORAGE" \
  --resource-group "$LAB_RG" \
  --output jsonc

az storage account show \
  --name "$STORAGE" \
  --resource-group "$LAB_RG" \
  --query '{name:name,kind:kind,sku:sku.name,https:enableHttpsTrafficOnly,\
publicBlobAccess:allowBlobPublicAccess,minTls:minimumTlsVersion}' \
  --output table

az storage blob service-properties show \
  --account-name "$STORAGE" \
  --auth-mode login \
  --output jsonc
```

The evidence should show a single enabled rule named logs-lifecycle; the blockBlob filter; prefix data/logs/; the four base-blob actions; and separate 30-day deletion actions for versions and snapshots. This verifies the declared policy, not yet the asynchronous execution of age-based transitions.

8.14.7 Prove Immediate Tier Semantics

Because the uploaded object is new, it is not yet eligible for the lifecycle thresholds. Use a direct data-plane operation to move only the matching object to the cool tier, then read its properties. This proves that the object supports per-blob tiering and that the signed-in identity can perform the intended operation.

```
az storage blob set-tier \
  --account-name "$STORAGE" \
  --container-name data \
  --name logs/app.log \
  --tier Cool \
  --auth-mode login

az storage blob show \
  --account-name "$STORAGE" \
  --container-name data \
  --name logs/app.log \
  --auth-mode login \
  --query '{name:name,tier:properties.blobTier,etag:properties.etag,\
lastModified:properties.lastModified,versionId:versionId}' \
  --output jsonc

az storage blob show \
  --account-name "$STORAGE" \
  --container-name data \
  --name reference/readme.txt \
  --auth-mode login \
  --query '{name:name,tier:properties.blobTier}' \
  --output jsonc
```

Expected result: logs/app.log reports the Cool tier, while reference/readme.txt remains in its original tier. The direct operation does not prove the scheduler ran; it proves the data-plane semantics that the lifecycle policy will later invoke when age conditions are satisfied.

8.14.8 Build the Evidence Packet

Create a directory named evidence and save the following artifacts. A useful evidence packet allows another reviewer to reconstruct intent, observed state, and cleanup without relying on screenshots alone.

Table 8-15 Suggested lifecycle-policy evidence packet.

Artifact	What it proves
01-deployment.json	The declared resource deployment and output identifiers

Artifact	What it proves
02-storage-resource.json	Storage kind, SKU, region, public-access and transport settings
03-lifecycle-policy.json	Rule name, filter scope, thresholds, and actions
04-blob-service-properties.json	Versioning and soft-delete protection settings
05-matching-object.json	Object name, current tier, ETag, time, and version identifier
06-nonmatching-object.json	The comparison object remains outside the policy prefix
07-resource-graph.json	Inventory of resources created in the lab resource group
08-activity-log.json	Management-plane creation and policy events
09-cleanup-proof.json	Post-deletion query showing no remaining lab resources

```
mkdir -p evidence

az deployment group show -g "$LAB_RG" -n main -o json \
  > evidence/01-deployment.json
az storage account show -g "$LAB_RG" -n "$STORAGE" -o json \
  > evidence/02-storage-resource.json
az storage account management-policy show -g "$LAB_RG" -n "$STORAGE" -o json \
  > evidence/03-lifecycle-policy.json
az storage blob service-properties show --account-name "$STORAGE" \
  --auth-mode login -o json > evidence/04-blob-service-properties.json
az storage blob show --account-name "$STORAGE" -c data -n logs/app.log \
  --auth-mode login -o json > evidence/05-matching-object.json
az storage blob show --account-name "$STORAGE" -c data -n reference/readme.txt \
  --auth-mode login -o json > evidence/06-nonmatching-object.json

az graph query -q "Resources | where resourceGroup =~ '$LAB_RG' | project
id,type,location,tags" \
  -o json > evidence/07-resource-graph.json

az monitor activity-log list --resource-group "$LAB_RG" --offset 2h -o json \
  > evidence/08-activity-log.json
```

8.14.9 Optional Extended Lifecycle Observation

To observe provider-applied lifecycle execution, use a nonproduction account and a rule whose permitted threshold is suitable for a longer-running test. Record the time the rule was installed, the object modification time, the time the tier changed, and any monitoring

evidence. Do not assume that a new rule executes immediately or at a predictable minute. Lifecycle processing is a periodic fleet-scale service, and completion time varies with scope and account population.

Do not shrink retention or deletion thresholds in an existing production account merely to make a demonstration finish quickly. A lifecycle policy is a destructive control. Test it first with a narrow prefix or tag, noncritical objects, a dedicated resource group/account, and a reviewed cleanup plan.

8.14.10 Common Failures and Their Meaning

Table 8-16 Lab failures reveal lifecycle and authority semantics rather than only syntax errors.

Symptom	Likely cause	Correct response
Upload returns authorization failure	Role assignment not propagated or wrong data-plane role	Wait, verify the assignment scope and principal, then retry with login auth
Policy exists but object does not transition immediately	Object is too new or lifecycle scheduler has not processed it	Verify rule and eligibility; treat lifecycle as asynchronous
Object outside intended path is affected	Prefix or tag filter is too broad	Disable rule, inspect scope, restore if needed, and redeploy reviewed policy
Delete action appears ineffective	Soft delete, versioning, immutability, or legal hold retains data	Inspect protection state and distinguish logical delete from purge
Archive transition fails or is unavailable	Object/account type or region does not support the action	Review current feature support and select a compatible tier/action
Cleanup fails	Immutable retention, lock, or dependency remains	Identify the blocking control; follow the authorized retirement procedure

8.14.11 Cleanup and Zero-Remainder Verification

```
az group delete --name "$LAB_RG" --yes --no-wait
az group wait --name "$LAB_RG" --deleted

az graph query -q "Resources | where resourceGroup =~ '$LAB_RG' | project
id,type" \
  -o json > evidence/09-cleanup-proof.json

cat evidence/09-cleanup-proof.json
```

The final query should contain no resources. Also check that no separately scoped diagnostic destination, event subscription, private endpoint, backup vault, immutable object, or role assignment was created outside the lab resource group. “Delete the account” is not sufficient evidence when supporting resources have broader scopes.

8.14.12 Translate the Lab to China’s Leading Cloud Platforms

Table 8-17 The lab translates by lifecycle and evidence intent, not by identical portal steps.

Lab intent	Alibaba Cloud	Tencent Cloud	Volcengine
Create object namespace	OSS bucket and prefix	COS bucket and prefix	TOS bucket and prefix
Define lifecycle	OSS lifecycle rule with storage-class transition and expiration	COS lifecycle configuration	TOS lifecycle rule
Protect prior state	Versioning, retention/WORM as required	Versioning, object lock/retention as required	Versioning and WORM/retention as required
Authorize data access	RAM role/policy; temporary credentials	CAM role/policy; temporary credentials	IAM role/policy; temporary credentials
Observe changes	OSS inventory/logging and audit services	COS inventory/logging and CloudAudit	TOS inventory/logging and audit services
Trigger downstream work	OSS event notification / EventBridge / Function Compute	COS event notification / EventBridge / SCF	TOS event notification and function/event integration
Verify cleanup	Delete bucket objects/versions and dependent resources, then prove absence	Remove objects/versions, rules, subscriptions, and bucket	Remove objects/versions, lifecycle/events, and bucket

8.15 CloudDesk Reuse: Storage Capabilities for a FinOps Agent

CloudDesk FinOps Agent may later use the capabilities in this chapter, but this book does not implement the agent runtime. The cloud capability contract should remain narrow, reviewable, and evidence-producing.

Table 8-18 CloudDesk capability handoff from the storage chapter.

Capability	Mode	Purpose	Required evidence
storage.lifecycle.recommend	Read-only recommendation	Identify objects or data products whose access history, age, and value support a tier or retention change	Current tier, age/access evidence, policy scope, projected savings, retrieval/early-deletion risk, owner
storage.cost.optimize	Plan before execute	Prepare a bounded lifecycle, version-expiry, or file-layout change	Plan digest, affected prefixes/tags, cost model, SLO impact, approval, rollback/exception
storage.event.subscribe	Controlled mutation	Create a filtered event subscription for a specific data product and consumer	Source, event types, filters, delivery identity, retry/DLQ policy, subscriber SLO, audit record
storage.recovery.verify	Read/test workflow	Assess whether replicas, versions, backups, metadata, and restore tests satisfy declared objectives	RPO/RTO target, last test, restored asset IDs, validation result, gap list

Agent safety boundary. A FinOps recommendation may identify savings, but deletion, retention reduction, archive transition, event subscription, or replication change must pass ownership, policy, SLO, approval, and evidence checks. The cheapest data is not necessarily the least valuable or least regulated.

8.16 Key Takeaways

1. Object, block, file, and data-lake storage are different managed contracts; selecting among them begins with namespace and access semantics.
2. A storage resource is a graph that includes identity, network, lifecycle, protection, eventing, telemetry, audit, and cost—not merely an endpoint that contains bytes.
3. Replication improves resilience to a stated failure topology; backup and recovery procedures restore meaningful prior state and must be tested against RPO and RTO.
4. Lifecycle management is a governed feedback loop. Filters, retention precedence, scheduler behavior, evidence, and exception ownership are as important as tier thresholds.
5. A data lake is a governed data estate. Zones, catalog, lineage, quality, ownership, and compute authorization prevent low-cost storage from becoming a high-cost data swamp.
6. Open table formats add transactional metadata above object files, but they do not eliminate the need for catalog, identity, maintenance, retention, and cost governance.

7. Storage events use at-least-once delivery semantics; correct downstream systems assume duplicates, delay, reordering, and partial failure.
8. AI-native data paths create multiple governed assets—from source objects to chunks, embeddings, indexes, traces, and evaluation evidence. Storage governance propagates into AI quality and isolation.
9. Storage economics include capacity, operations, retrieval, transfer, protection, metadata, and downstream compute. Price per gigabyte alone is not an architecture decision.
10. Multicloud translation should compare engineering intent and record non-equivalence in hierarchy, tiering, event delivery, replication, protection, and evidence.

References

1. NIST. The NIST Definition of Cloud Computing, SP 800-145.
https://csrc.nist.gov/publications/detail/sp/800-145/final

2. Microsoft Learn. Introduction to Azure Storage.
https://learn.microsoft.com/azure/storage/common/storage-introduction

3. Microsoft Learn. Storage account overview.
https://learn.microsoft.com/azure/storage/common/storage-account-overview

4. Microsoft Learn. Introduction to Azure Blob Storage.
https://learn.microsoft.com/azure/storage/blobs/storage-blobs-introduction

5. Microsoft Learn. Azure Managed Disks overview. https://learn.microsoft.com/azure/virtual-machines/managed-disks-overview

6. Microsoft Learn. Introduction to Azure Files. https://learn.microsoft.com/azure/storage/files/storage-files-introduction

7. Microsoft Learn. Introduction to Azure Data Lake Storage.
https://learn.microsoft.com/azure/storage/blobs/data-lake-storage-introduction

8. Microsoft Learn. Azure Data Lake Storage hierarchical namespace.
https://learn.microsoft.com/azure/storage/blobs/data-lake-storage-namespace

9. Microsoft Learn. Access control model in Azure Data Lake Storage.
https://learn.microsoft.com/azure/storage/blobs/data-lake-storage-access-control-model

10. Microsoft Learn. Azure Blob Storage lifecycle management overview.
https://learn.microsoft.com/azure/storage/blobs/lifecycle-management-overview

11. Microsoft Learn. Configure a lifecycle management policy.
https://learn.microsoft.com/azure/storage/blobs/lifecycle-management-policy-configure

12. Microsoft Learn. Lifecycle management policy structure.
https://learn.microsoft.com/azure/storage/blobs/lifecycle-management-policy-structure

13. Microsoft Learn. Access tiers for blob data. https://learn.microsoft.com/azure/storage/blobs/access-tiers-overview

14. Microsoft Learn. Lifecycle management performance characteristics.
https://learn.microsoft.com/azure/storage/blobs/lifecycle-management-performance-characteristics

15. Microsoft Learn. Data protection overview for Azure Storage. https://learn.microsoft.com/azure/storage/blobs/data-protection-overview

16. Microsoft Learn. Azure Storage redundancy. https://learn.microsoft.com/azure/storage/common/storage-redundancy

17. Microsoft Learn. Blob versioning. https://learn.microsoft.com/azure/storage/blobs/versioning-overview

18. Microsoft Learn. Soft delete for blobs. https://learn.microsoft.com/azure/storage/blobs/soft-delete-blob-overview

19. Microsoft Learn. Immutable storage for blob data. https://learn.microsoft.com/azure/storage/blobs/immutable-storage-overview

20. Microsoft Learn. Point-in-time restore for block blobs. https://learn.microsoft.com/azure/storage/blobs/point-in-time-restore-overview

21. Microsoft Learn. Reacting to Blob Storage events. https://learn.microsoft.com/azure/storage/blobs/storage-blob-event-overview

22. Microsoft Learn. Azure Event Grid delivery and retry. https://learn.microsoft.com/azure/event-grid/delivery-and-retry

23. Microsoft Learn. Azure Event Grid event schema for Blob Storage. https://learn.microsoft.com/azure/event-grid/event-schema-blob-storage

24. Microsoft Learn. Azure Event Grid concepts. https://learn.microsoft.com/azure/event-grid/concepts

25. Microsoft Learn. Azure Resource Graph sample queries for Storage. https://learn.microsoft.com/azure/storage/common/resource-graph-samples

26. Microsoft Learn. Microsoft.Storage storageAccounts/managementPolicies Bicep reference. https://learn.microsoft.com/azure/templates/microsoft.storage/storageaccounts/managementpolicies

27. Cloud Native Computing Foundation. CloudEvents Specification. https://cloudevents.io/

28. Apache Iceberg. Documentation and specification. https://iceberg.apache.org/

29. Delta Lake. Documentation. https://docs.delta.io/

30. Apache Hudi. Documentation. https://hudi.apache.org/docs/overview/

31. Apache Parquet. Documentation. https://parquet.apache.org/docs/

32. Alibaba Cloud. Object Storage Service overview. https://www.alibabacloud.com/help/en/oss/product-overview/what-is-oss

33. Alibaba Cloud. OSS storage classes. https://www.alibabacloud.com/help/en/oss/user-guide/overview-55

34. Alibaba Cloud. OSS lifecycle rules. https://www.alibabacloud.com/help/en/oss/user-guide/lifecycle-rules-based-on-the-last-modified-time

35. Alibaba Cloud. OSS event notifications. https://www.alibabacloud.com/help/en/oss/user-guide/event-notifications

36. Alibaba Cloud. Data Lake Formation overview. https://www.alibabacloud.com/help/en/dlf/product-overview/what-is-data-lake-formation

37. Alibaba Cloud. Build a data lake based on OSS. https://www.alibabacloud.com/help/en/oss/use-cases/data-lake

38. Alibaba Cloud. Elastic Block Storage overview. https://www.alibabacloud.com/help/en/ecs/user-guide/block-storage-overview

39. Tencent Cloud. Cloud Object Storage product documentation. https://www.tencentcloud.com/document/product/436

40. Tencent Cloud. COS lifecycle configuration. https://www.tencentcloud.com/document/product/436/17059

41. Tencent Cloud. COS event notification. https://www.tencentcloud.com/document/product/436/16930

42. Tencent Cloud. Data Lake Compute overview. https://www.tencentcloud.com/document/product/1155

43. Tencent Cloud. Cloud Block Storage overview. https://www.tencentcloud.com/document/product/362

44. Tencent Cloud. Cloud File Storage overview. https://www.tencentcloud.com/document/product/582

45. Volcengine. Tinder Object Storage documentation. https://www.volcengine.com/docs/6349

46. Volcengine. TOS lifecycle management. https://www.volcengine.com/docs/6349/119289

47. Volcengine. TOS event notification. https://www.volcengine.com/docs/6349/119286

48. Volcengine. TOS cross-region replication. https://www.volcengine.com/docs/6349/119287

49. Volcengine. TOS PrivateLink access. https://www.volcengine.com/docs/6349/1167162

50. Volcengine. TOS WORM retention. https://www.volcengine.com/docs/6349/147331

51. Volcengine. AI Data Lake documentation. https://www.volcengine.com/docs/6893

52. Volcengine. Elastic Block Storage documentation. https://www.volcengine.com/docs/6397

53. FinOps Foundation. FinOps Framework. https://www.finops.org/framework/

54. OpenTelemetry. Specification. https://opentelemetry.io/docs/specs/

55. OWASP. Cloud-Native Application Security Top 10. https://owasp.org/www-project-cloud-native-application-security-top-10/

Application and Cloud-Native Control Planes

How does a source commit become the version a user reaches? Trace that journey through managed runtimes, container artifacts, Kubernetes objects, and delivery controllers. Each stage leaves different evidence of what was built, deployed, routed, or rolled back.

09 Application PaaS, Functions, Events, and API Management

10 Containers, Registries, and Software Supply Chains

11 Managed Kubernetes and Application Definition

12 Continuous Delivery, GitOps, and Change Governance

Part III / Chapter 9

Application PaaS, Functions, Events, and API Management

How Managed Runtimes, Event Contracts, and API Gateways Turn Application Code into Governed Services

Application PaaS does not eliminate operations; it moves recurring infrastructure work into managed control-plane contracts. A web application, a function, an event route, a broker, and an API gateway are not interchangeable compute options. They are different resources that govern runtime, invocation, delivery, exposure, policy, evidence, and cost.

In This Chapter

- Treat application hosting, functions, event routing, messaging, streaming, and API management as distinct managed resource contracts.
- Model an application PaaS resource as a graph of plan, app, artifact, configuration, identity, endpoint, health, scale, telemetry, and cost objects.
- Separate build artifacts, release versions, deployment slots or revisions, runtime instances, and external state.
- Understand a function as event-addressed compute whose trigger, hosting plan, concurrency, retry, version, and identity determine the operating contract.
- Choose among event notification, brokered messaging, and retained event streaming by delivery semantics rather than by product name.
- Use API management to productize, secure, observe, version, and govern APIs without confusing the gateway with application middleware or organizational policy.
- Connect application PaaS decisions to identity, private connectivity, observability, SLOs, quotas, and FinOps.
- Complete two independent Azure labs: deploy a managed Web API, then route an object-created event to a cloud function.

- Translate the engineering intent to Alibaba Cloud, Tencent Cloud, and Volcengine, and expose bounded capabilities for CloudDesk FinOps Agent.

A software team can move a web application from a virtual machine to a managed application platform in an afternoon and still misunderstand the architecture. The code runs, the provider patches the host, and a default domain appears. Yet the system is not production-ready until deployment identity, configuration, health, scale, release, network, evidence, cost, and retirement are explicit. The same mistake appears with functions and events: a short handler is treated as if it were the whole application, while trigger delivery, retries, concurrency, dead-letter behavior, and downstream authority remain implicit.

The control-plane perspective changes the unit of analysis. Instead of asking which service can execute code, it asks which managed contract should own each responsibility. A continuously running HTTP service, an event-triggered function, a reliable command queue, a high-throughput event stream, and an API gateway all execute or move application work, but they make different promises and expose different failure modes. Combining them well is application architecture; combining them without those distinctions is product assembly.

Central claim. Application PaaS is a family of managed control-plane contracts that trade infrastructure control for lifecycle automation, elasticity, policy, telemetry, and faster delivery. The application remains responsible for business behavior, data semantics, and safe integration.

9.1 PaaS Productizes Repeated Operational Work

NIST defines Platform as a Service as the capability to deploy customer-created or acquired applications onto provider-supported infrastructure using supported languages, libraries, services, and tools, while the customer does not manage the underlying network, servers, operating systems, or storage. That definition is still useful, but modern PaaS extends far beyond a language runtime. It may include application hosting, functions, event routers, message brokers, stream ingestion, API gateways, managed configuration, identity, diagnostics, release primitives, and policy.

The phrase “fully managed” does not mean “no operations.” It means the provider assumes recurring infrastructure duties and exposes higher-level operational controls. The customer still chooses release semantics, defines health, configures access, designs state, sets quotas, handles poison messages, observes business outcomes, and decides when to retire a resource. PaaS therefore shifts the work upward: fewer host-level tasks, more application-level contracts.

Table 9-1 PaaS resource families productize different forms of application work.

Resource family	Primary contract	Provider-managed work	Customer decisions
Managed application runtime	Continuously addressable service or API	OS/runtime patching, host placement, load distribution, platform health	Artifact, configuration, identity, scale, health endpoint, release, data state
Function runtime	Event-addressed or request-addressed unit of execution	Instance creation, event dispatch, elastic scale, runtime lifecycle	Trigger, timeout, concurrency, retry, idempotency, version, downstream authority
Event router	Filter and route notifications that something happened	Source integration, matching, delivery attempts, fan-out	Event schema, filters, target, retry horizon, dead-letter destination
Message broker	Reliably transfer commands or work ownership	Durable queue/topic, acknowledgments, delivery state, sessions or transactions	Message contract, competing consumers, duplicate handling, dead-letter process
Event stream	Retain and replay ordered event sequences at scale	Partitioned ingestion, retention, consumer offsets, throughput infrastructure	Partition key, retention, replay, consumer groups, schema evolution
API management	Productize and govern API exposure	Managed gateway, policy engine, API catalog, usage telemetry	API contract, consumer identity, quotas, routing, versions, products, backend trust

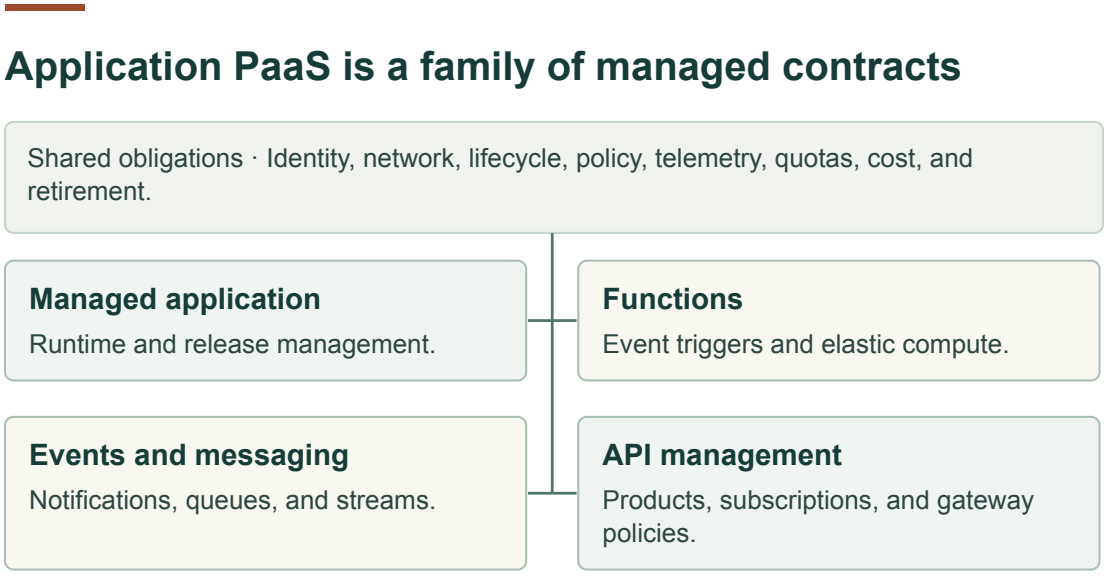


Figure 9-1 Application PaaS is a family of managed contracts with shared control-plane obligations.

These contracts can be composed, but one does not erase the others. Placing an application behind an API gateway does not transfer application health to the gateway. Connecting a storage event to a function does not make the function the system of record. Sending a command to a queue does not guarantee exactly-once business behavior. A managed platform reduces undifferentiated infrastructure work; it does not eliminate architecture.

9.2 Application PaaS Is a Resource Graph

A managed web application is often represented as one box labeled “App Service” or “SAE.” In the control plane it is a graph. A capacity or environment resource provides compute; an application resource binds runtime and artifact; a deployment slot or revision provides release isolation; configuration and secret references shape behavior; identity and network resources govern access; domains and certificates expose the service; health and scaling resources manage runtime state; telemetry and cost resources produce evidence.

The managed application resource graph

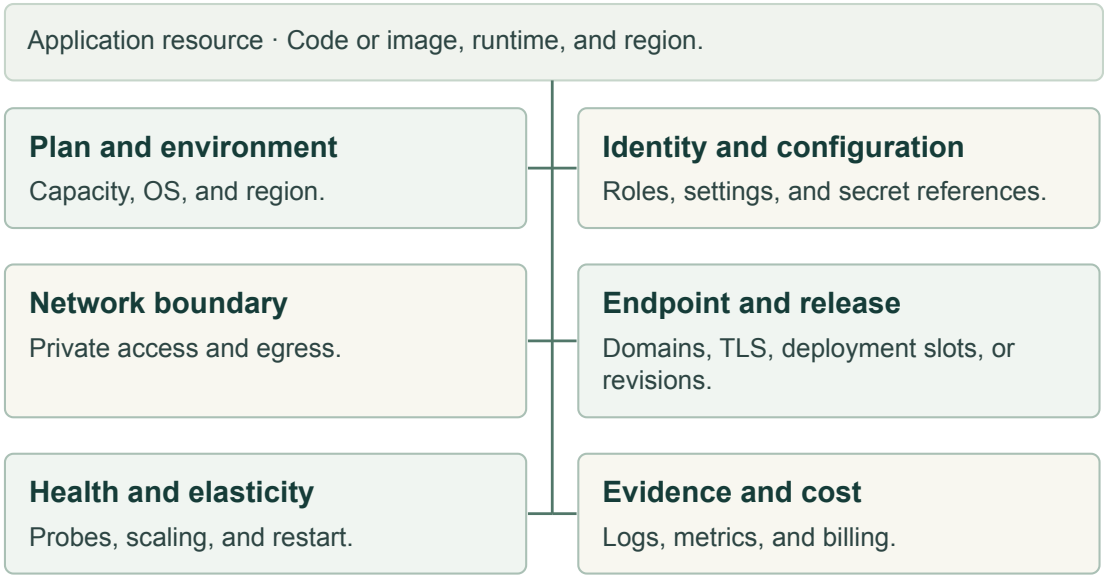


Figure 9-2 A managed application is a resource graph rather than one opaque hosting object.

Table 9-2 Application PaaS concerns map to distinct resources and questions.

Concern	Representative objects	Engineering question
Capacity and environment	App plan, environment, region, operating system, instance class	What capacity is shared, reserved, or elastically allocated, and which failure domain contains it?
Application identity	Application resource, resource ID, tags, owner, workload identity	What is the durable management identity, and which principal can change or invoke it?
Artifact and runtime	Source package, build output, container image, runtime stack, start-up command	What exactly is deployed, and can it be reproduced independently of the live runtime?
Configuration and secrets	Application settings, environment variables, configuration references, secret store	Which values vary by environment, and which are sensitive or independently versioned?
Release	Slot, revision, deployment version, alias, traffic rule	How is a candidate validated, promoted, rolled back, and retired?
Endpoint and trust	Default domain, custom domain, certificate, private endpoint, inbound policy	Who can reach the service, over which name and network path, and how are consumers authenticated?

Concern	Representative objects	Engineering question
Health and elasticity	Health check, readiness, instance count, autoscale rule, restart policy	What does ready mean, and how does observed demand become safe capacity?
Evidence and economics	Logs, metrics, traces, activity records, quotas, cost scope, budget	How can operators explain runtime state, change history, performance, and cost?

The application resource therefore sits between two contracts. The control plane creates and configures the service. The data plane receives requests and runs application code. Management permission to edit the app does not automatically grant application-level access, and application identity does not automatically grant control-plane permission. The separation should remain visible in design records and tests.

Plan and application are also different boundaries. In Azure App Service, an App Service plan defines the regional compute resources on which one or more apps run. Scaling the plan changes capacity shared by those apps; scaling or configuration at app scope changes behavior within that capacity contract. Other providers use environment, namespace, or application-level abstractions, but the same question remains: which settings belong to shared capacity, and which belong to one application?

9.3 Runtime and Deployment Contracts

9.3.1 Source, Package, Image, and Runtime

Application PaaS may accept source code, a language package, a compiled archive, or a container image. These are delivery formats, not equivalent runtime contracts. Source deployment delegates more build behavior to the platform. Package deployment supplies a prepared artifact but still uses a provider runtime. Container deployment supplies the userspace and dependencies while the provider continues to manage placement, routing, health integration, and host infrastructure.

Table 9-3 Delivery format changes the build and runtime responsibility boundary.

Delivery format	Platform builds	Customer supplies	Key governance question
Source code	Dependency resolution, compilation or buildpack step, runtime image selection	Source, manifests, build settings	Can the build be reproduced and attested outside the live service?

Delivery format	Platform builds	Customer supplies	Key governance question
Compiled package	Deployment extraction and runtime hosting	Built artifact and dependencies	Is the package immutable, checksummed, and linked to source and test evidence?
Container image	Image pull and managed execution	OCI image, startup behavior, exposed port	Is the image signed, scanned, minimal, and compatible with the platform contract?
External build pipeline	Only runtime placement and service integration	Versioned artifact produced by CI	Are build identity, provenance, approval, and artifact promotion preserved?

Build and runtime evidence should be separate. A successful deployment proves that an artifact reached the platform; it does not prove that the artifact came from an approved source revision, passed tests, or is free of known vulnerabilities. The container and supply-chain chapter develops that evidence in detail.

9.3.2 Configuration, Secrets, and Identity

Configuration should be external to the artifact so the same artifact can move through environments. That does not imply that all configuration is safe to expose as environment variables. Non-sensitive values such as feature settings and service endpoints can be injected directly. Credentials should normally be replaced by workload identity or referenced from a managed secret store. Configuration changes require version and audit semantics because they can change application behavior without changing code.

Table 9-4 Configuration mechanisms have different security and lifecycle properties.

Mechanism	Appropriate use	Risk	Preferred control
Application setting / environment variable	Environment-specific non-secret behavior	Unreviewed drift or accidental exposure in diagnostics	IaC or release pipeline, change audit, configuration inventory
Secret value embedded in setting	Legacy connection strings or keys	Long-lived credential leakage and difficult rotation	Replace with workload identity or secret reference
Secret-store reference	Passwords, certificates, API keys that cannot yet be eliminated	Runtime availability depends on secret store and identity	Managed identity, least privilege, rotation and access evidence

Mechanism	Appropriate use	Risk	Preferred control
Managed service identity	Access to supported cloud services without embedded secret	Over-broad role assignment or unclear delegation	Dedicated identity, narrow scope, explicit data-plane role
Feature flag / dynamic config	Controlled behavior change without redeployment	Configuration becomes an ungoverned release path	Version, owner, evaluation window, rollback and telemetry

9.3.3 Health, Readiness, Startup, and Graceful Termination

A process that accepts a TCP connection is not necessarily ready to serve a correct business request. Application PaaS health must be defined in application terms. A liveness signal asks whether an instance should be restarted or replaced. Readiness asks whether it should receive traffic. Startup protects slow initialization from premature failure. Graceful termination gives in-flight work time to finish before an instance is removed.

Table 9-5 Health signals answer different lifecycle questions.

Signal	Question	Typical evidence	Common mistake
Liveness	Is the process irrecoverably stuck?	Probe success, heartbeat, runtime state	Checking every dependency and causing cascading restarts
Readiness	Can this instance safely receive new traffic?	Health endpoint, dependency readiness, warm cache state	Returning ready before migrations or initialization complete
Startup	Has initialization completed within an acceptable window?	Startup probe, initialization trace, bounded timeout	Using long startup to hide a broken dependency
Graceful termination	Can in-flight work finish before removal?	Drain signal, active request count, queue checkpoint	Killing the instance before acknowledgments or commits complete
Business health	Is the service delivering the user outcome?	SLI, synthetic check, transaction success	Treating CPU and process uptime as user experience

Health checks create reliable behavior only when there are at least two healthy instances or an alternate route. A single-instance health probe can reveal failure but cannot provide redundancy. This distinction prevents a green health indicator from being mislabeled as high availability.

9.3.4 Scaling, Capacity, and State

Managed hosting may scale manually, by schedule, by metric, or automatically through a platform-specific controller. Desired instance count, allocated instances, ready instances, and effective service capacity are still different states, as discussed in the compute chapter. Application PaaS hides host provisioning but not startup delay, connection pool pressure, downstream quotas, or stateful bottlenecks.

Horizontal scale assumes that request-serving state is external, replicated, or safely partitioned. Local disk, in-memory sessions, and singleton background jobs can turn scale-out into data loss or duplicate work. Durable state should live in a service whose consistency, availability, backup, and cost contracts are explicit. Background work should normally use a queue, scheduler, or function trigger rather than piggybacking on one web instance.

9.4 Releases: Slots, Revisions, Versions, and Rollback

A deployment is not a release. Deployment places an artifact into a target environment; release changes who or what receives traffic. Managed platforms expose several primitives for separating these actions: deployment slots, immutable revisions, function versions, aliases, traffic weights, and environment promotion. The names differ, but the engineering intent is stable.

Application delivery separates artifact, release, and traffic

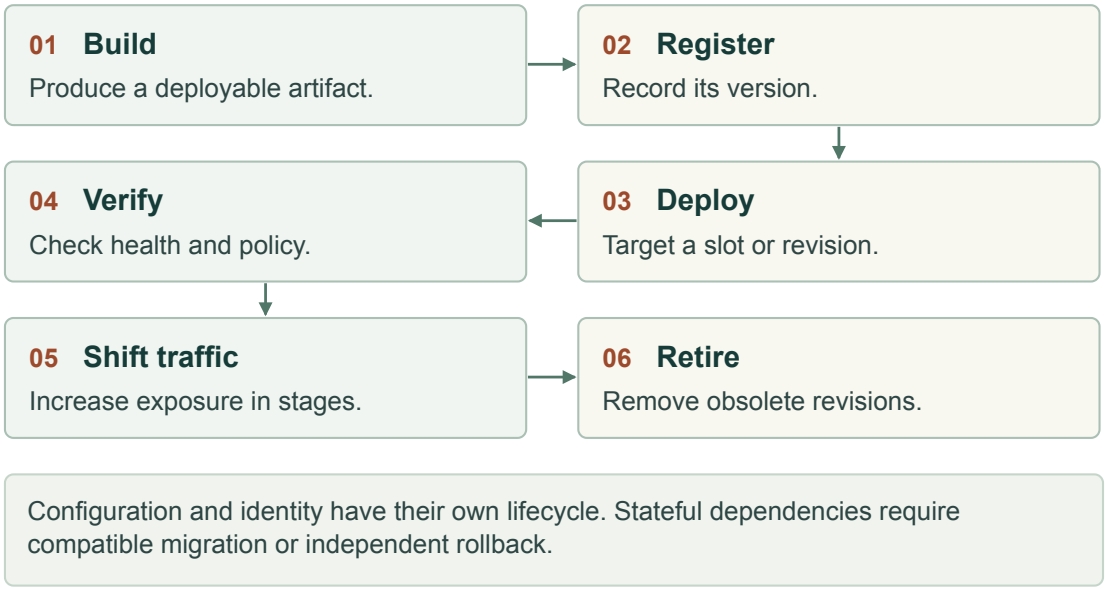


Figure 9-3 Application lifecycle separates artifact, release, runtime, traffic, and external state.

Table 9-6 Release primitives are not interchangeable.

Primitive	What it isolates	What it enables	What it does not solve
Deployment version	Code and deployment configuration snapshot	Traceability, immutable rollback target	Traffic control or data rollback by itself
Slot / environment	Parallel runtime with separate endpoint or configuration	Warm validation, swap, blue-green release	Schema compatibility and state migration
Revision	Immutable application version within a managed environment	Traffic splitting, rollback, audit	Unversioned external configuration or shared database state
Function version	Immutable function code and selected configuration	Stable target for alias or trigger	Event-schema evolution and poison-message recovery
Alias / traffic rule	Stable logical name mapped to one or more versions	Canary, promotion, rapid routing rollback	Correctness of the new version or external side effects
Feature flag	Runtime behavior switch	Small blast radius and decoupled activation	Infrastructure incompatibility or hidden configuration debt

A rollback must name what is being rolled back. Routing to an earlier application version may restore request handling but cannot reverse a database migration, an emitted event, a payment, or a destructive tool call. Release records should therefore link artifact digest, configuration version, schema compatibility, migration plan, approval, health evidence, and external effects.

9.5 Functions Are Event-Addressed Compute Resources

A function is often introduced as a short piece of code. At resource level, the code is only one field. A production function also has a hosting plan, runtime, memory and timeout, concurrency policy, trigger, bindings or service clients, version, alias, identity, network path, retry behavior, dead-letter handling, logs, metrics, quota, and cost scope. The function is small only if these contracts are ignored.

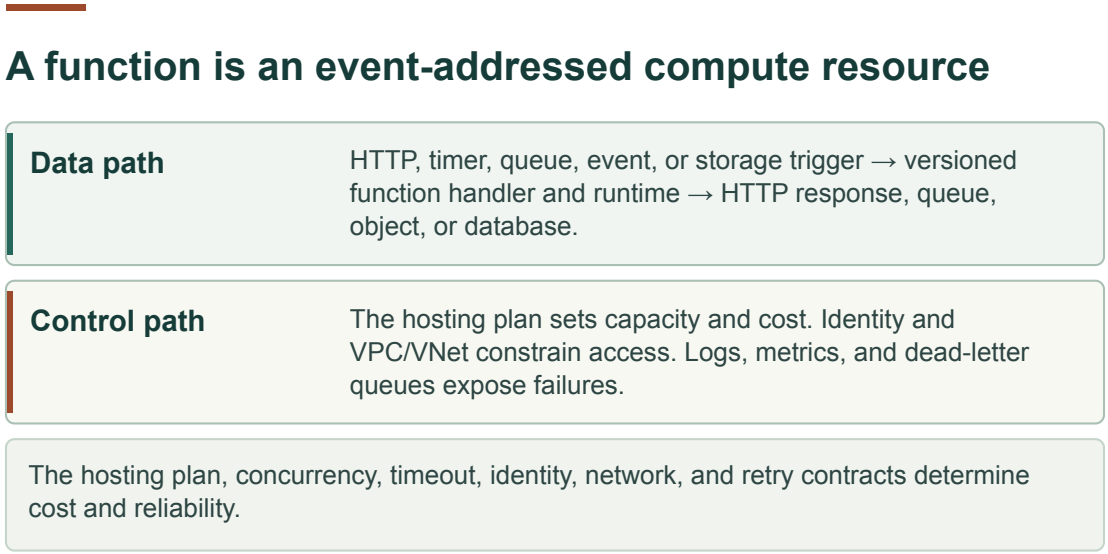


Figure 9-4 A function is an event-addressed compute resource with trigger, hosting, identity, retry, and evidence contracts.

Table 9-7 Managed application hosting and functions optimize for different workload shapes.

Dimension	Managed application runtime	Function runtime
Addressability	Stable service endpoint, usually continuously available	Event or request invokes a function; instance may not exist beforehand
Execution lifetime	Long-running process with repeated requests	Bounded invocation or task, subject to platform timeouts and plans

Dimension	Managed application runtime	Function runtime
Scaling unit	Application instance or revision	Function instance and concurrency per instance
State assumption	Can maintain process-local cache, but durable state should remain external	Invocation should be restartable; durable progress must be external or checkpointed
Best fit	Web/API services, continuously used backends, framework applications	Bursty event processing, glue logic, scheduled tasks, fan-out work
Primary failure risk	Unhealthy revision, capacity lag, configuration drift, dependency saturation	Duplicate delivery, retry storms, timeout, cold start, concurrency conflict
Economics	Provisioned plan or managed capacity, often billed while allocated	Invocation, execution duration, memory/CPU, provisioned or reserved capacity, network

9.5.1 Triggers and Bindings

The trigger is part of the function's control-plane contract. It defines which source may invoke the function, the event or message shape, filter conditions, authentication, batching, concurrency, and checkpoint or acknowledgment behavior. A binding or connector can reduce boilerplate, but it should not hide authority. Operators must still know which identity reads the source, which identity writes the target, and where failed delivery is recorded.

Table 9-8 Common trigger types create different delivery and scaling behavior.

Trigger	Invocation semantics	Scale signal	Required design decision
HTTP / webhook	Synchronous request-response	Concurrent requests or platform request metrics	Authentication, timeout, idempotency, response contract
Timer / schedule	Platform-generated scheduled event	Schedule and overlap policy	Timezone, missed runs, concurrent execution, backfill
Queue / broker	Messages acquired and acknowledged by consumers	Queue length, age, or partitions	Visibility lock, retry, poison message, competing consumers
Event router	Notification delivered by push or managed subscription	Event rate and delivery backlog	Filter, duplicate handling, source-of-truth lookup, dead letter

Trigger	Invocation semantics	Scale signal	Required design decision
Object storage	Object-created or changed notification	Object event rate or trigger-specific poller	Prefix/suffix filter, partially uploaded objects, duplicate event
Stream / log	Partitioned records consumed by checkpoint	Partition count, lag, batch size	Ordering, checkpoint, replay, hot partition, schema evolution

9.5.2 Hosting Plans and Execution Economics

A function's hosting plan is an operating contract, not a pricing footnote. It determines scale-to-zero behavior, prewarmed capacity, network integration, execution duration, instance size, concurrency, zone support, and cost. Azure currently offers Flex Consumption, Premium, Dedicated App Service, and legacy Consumption options; the right choice follows latency, networking, capacity, and reliability requirements rather than a general preference for "serverless."

Table 9-9 Azure Functions hosting plans illustrate distinct operating contracts.

Plan	Capacity behavior	Good fit	Design caution
Flex Consumption	Dynamic scale with flexible instance sizes and optional always-ready instances	Event-driven workloads needing modern networking and configurable capacity	Validate regional support, trigger behavior, cold-start objective, and per-function scaling
Premium	Prewarmed and elastic instances with advanced networking and longer-running scenarios	Latency-sensitive functions, private networking, sustained or bursty enterprise work	Minimum allocated capacity can dominate cost at low utilization
Dedicated App Service plan	Functions share provisioned App Service capacity	Existing plan capacity, predictable always-on workloads, combined app/function estate	Customer owns plan sizing and scale; poor isolation can create contention
Consumption	Classic dynamic consumption model	Broadly supported small event-driven workloads and existing estates	Regional, networking, runtime, and feature limitations may differ from newer plans

Plan names, limits, runtime versions, and regional availability are volatile facts. Verify current provider documentation during lab execution and before publication freeze.

9.5.3 Concurrency, Retries, Timeouts, and Idempotency

Elastic scale multiplies concurrency. A handler that is correct for one invocation can fail when ten instances process related messages simultaneously. Concurrency controls, batching, downstream connection limits, partition keys, optimistic concurrency, and idempotency keys are therefore part of function design. Reducing concurrency can protect a dependency but can also increase queue age and cost.

Retries convert transient failure into eventual success only when the operation can be repeated safely. At-least-once delivery is common, so the business operation should use a durable idempotency key, transaction boundary, or compare-and-set condition. A dead-letter destination is not a trash bin; it is an operational work queue with ownership, retention, alerting, diagnosis, and replay procedures.

9.6 Events, Messages, and Streams Are Different Contracts

Distributed applications exchange facts, commands, and histories. Cloud catalogs often place event routers, queues, topics, and streaming services under one “messaging” category, but their semantics differ. The most useful first question is not which product is fastest; it is what the publisher is asserting and what the consumer must guarantee.

Events, commands, and streams answer different questions

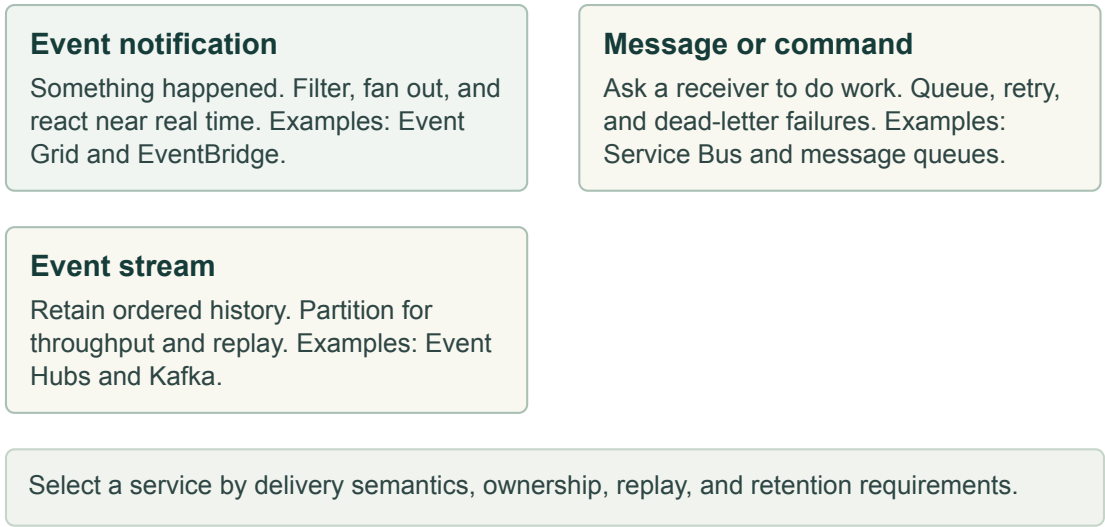


Figure 9-5 Events, messages, and streams encode different intent, ownership, retention, and replay contracts.

Table 9-10 Choose event, message, or stream by semantics.

Contract	Publisher statement	Consumer obligation	Representative capabilities
Event notification	A state change occurred	React if interested; fetch authoritative state if required	Filtering, fan-out, near-real-time delivery, retry, dead-letter
Command / work message	Perform a requested unit of work	Acquire ownership, execute, acknowledge, retry or dead-letter	Queue, topic/subscription, lock, session, duplicate detection, transaction
Event stream	Here is an ordered sequence of observations	Track offset, process partitions, tolerate replay, maintain consumer state	Partitions, retention, replay, consumer groups, capture, high throughput

Azure Event Grid, Service Bus, and Event Hubs make the distinction visible. Event Grid routes discrete events and integrates deeply with Azure resources. Service Bus provides enterprise queues and topics with brokered delivery features such as sessions, duplicate detection, transactions, and dead lettering. Event Hubs provides partitioned, retained, high-

throughput streaming with consumer groups and Kafka compatibility. Similar categories exist across Alibaba Cloud, Tencent Cloud, and Volcengine, but exact features and guarantees are not interchangeable.

9.6.1 Delivery Semantics and Failure Handling

Table 9-11 Delivery guarantees require matching application mechanisms.

Concern	Platform mechanism	Application responsibility
At-least-once delivery	Retry until success or expiration	Idempotency, duplicate detection, safe replay
Ordering	Partition key, session, ordered queue, single consumer	Choose ordering scope and avoid hidden cross-partition assumptions
Backpressure	Queue depth, consumer lag, throughput units, rate limit	Scale consumers, shed load, prioritize, or slow producers
Poison work	Delivery count and dead-letter destination	Diagnose, correct data or code, approve replay, preserve evidence
Retention and replay	Event retention, checkpoint, capture, archive	Version consumers and schemas; define replay window and side-effect safety
Partial failure	Per-message status, batch retry, transaction or compensation	Record progress, isolate failures, avoid retrying successful side effects

Exactly-once transport is not the same as exactly-once business effect. A payment, notification, document conversion, or resource mutation may cross systems with different transaction boundaries. The robust design records a business idempotency key and outcome, then treats transport replay as a normal condition.

9.6.2 Event Schemas, CloudEvents, and AsyncAPI

An event envelope should identify the event independently of its payload. CloudEvents standardizes common metadata such as id, source, type, subject, time, data content type, and specification version. This allows routers and observability tools to operate without understanding every business payload. The payload still needs its own schema, version, compatibility policy, owner, and retention rules.

```
{
  "specversion": "1.0",
  "id": "01J6-book-lab-0001",
  "source": "/subscriptions/.../storageAccounts/booklab",
  "type": "Microsoft.Storage.BlobCreated",
```

```
  "subject": "/blobServices/default/containers/incoming/blobs/sample.txt",
  "time": "2026-08-26T06:00:00Z",
  "datacontenttype": "application/json",
  "data": {
    "url": "https://booklab.blob.core.windows.net/incoming/sample.txt"
  }
}
```

OpenAPI describes request-response HTTP interfaces. AsyncAPI describes message-driven interfaces across protocols such as AMQP, MQTT, Kafka, WebSockets, and HTTP. A mature platform treats these descriptions as governed artifacts: linted, versioned, published, linked to implementations, and reviewed for compatibility and sensitive fields.

9.7 API Management Is a Productization Control Plane

An HTTP endpoint becomes an API product only when consumers can discover it, obtain access, understand its contract, operate within quotas, and receive stable versions and support. API management supplies the resource model and gateway runtime for that productization. It can front an App Service app, function, Kubernetes service, SaaS endpoint, on-premises service, or multicloud backend.

API management separates product design from the gateway data plane

Management plane	API, operation, schema, and version → product and subscription → policy, portal, and workspace.
Data plane	Internal, external, partner, or agent client → gateway authenticates, validates, limits, transforms, and traces → app, function, Kubernetes, or SaaS backend.

Correlate caller identity, policy decision, status, latency, and cost for each request.

Figure 9-6 API management separates API product control from gateway request execution.

Table 9-12 API management resources govern more than HTTP forwarding.

Resource	Purpose	Lifecycle concern
API and operation	Contract, path, method, schema, examples	Import, design review, revision, version, deprecation
Backend	Target service and backend authentication	Endpoint change, trust, timeout, circuit and credential rotation
Gateway	Runtime that proxies and enforces policy	Capacity, placement, networking, upgrade, telemetry
Policy	Runtime rule for security, validation, transformation, routing, rate, cache or trace	Scope, order, testing, version, rollback, performance effect
Product / plan	Curated API bundle and commercial or organizational access contract	Audience, quota, terms, owner, retirement
Subscription / consumer	Consumer identity and access entitlement	Issuance, rotation, revocation, usage evidence
Developer portal / catalog	Discovery, documentation, onboarding and testing	Content publication, identity, exposure, accessibility
Workspace / team boundary	Delegated API ownership in a shared platform	Role separation, common policy, naming, cost attribution

9.7.1 Gateway Policies and Runtime Enforcement

The gateway is the API Management data plane. It authenticates or validates consumers, routes requests, applies policies, and records telemetry. Policies can validate tokens and schemas, transform headers and bodies, rewrite routes, limit rate, cache responses, call auxiliary services, and emit traces. Each policy increases runtime complexity and latency; it should have an owner, tests, performance budget, failure behavior, and rollback path.

Azure API Management policies are not Azure Policy. API Management policies execute in the gateway against API traffic. Azure Policy evaluates and enforces the configuration of Azure resources. Application middleware remains a third layer, responsible for business authorization and domain validation. Treating these three as synonyms produces either duplicated controls or dangerous gaps.

Table 9-13 Gateway policy, organizational policy, and application middleware enforce different concerns.

Mechanism	Execution point	Good fit	Poor fit
API gateway policy	Before or after backend invocation in the gateway	Consumer authentication, quota, schema check, routing, protocol transformation	Deep domain rules, database transaction, resource-level business authorization
Cloud organizational policy	Control-plane resource creation or assessment	Allowed regions, required private access, SKU constraints, tagging, diagnostics	Per-request authorization or payload validation
Application middleware	Inside the application runtime	Domain authorization, tenancy, transaction rules, business validation	Fleet-wide resource governance or cross-API product catalog
Edge WAF / DDoS layer	Internet edge or application front door	Common web exploits, volumetric protection, edge filtering	Business quota, API schema governance, tool-level authorization

9.7.2 API Lifecycle and Developer Experience

API lifecycle includes design, review, implementation, testing, publication, onboarding, observation, revision, versioning, deprecation, and retirement. A developer portal is not merely documentation hosting; it is part of the consumer control plane. It may expose products, subscription requests, interactive testing, usage reports, terms, support contacts, and change notices.

Version and revision solve different problems. A revision changes an API implementation or policy while preserving the public version contract. A version introduces a deliberate contract break or long-lived variant. The platform should define how long old versions remain available, which consumers depend on them, and what evidence is required before retirement.

9.7.3 API Management Is Not a Load Balancer or Service Mesh

Table 9-14 Adjacent traffic components have overlapping features but different primary control planes.

Component	Primary audience	Primary abstraction	Typical scope
Layer 4 load balancer	Network and platform operators	IP, port, backend pool, health probe	Transport-level distribution
Layer 7 application gateway / ingress	Application platform operators	Host, path, certificate, route, WAF	Web ingress and application routing

Component	Primary audience	Primary abstraction	Typical scope
API management	API producers and consumers	API, operation, product, subscription, policy, developer portal	API lifecycle and consumer governance
Service mesh	Service platform and application teams	Service identity, east-west traffic, policy, telemetry	Internal service-to-service communication
Application middleware	Application developers	Domain request, user, tenant, transaction	Business behavior and authorization

9.8 Identity, Network, and Trust Boundaries

A managed application path contains multiple identities. The end user may authenticate to an identity provider. The gateway authenticates the consumer and applies product policy. The application receives a delegated user token or a gateway assertion. The application's workload identity accesses storage, databases, queues, or model resources. An event router or broker uses a delivery identity to invoke a function or endpoint. Each hop has a separate audience, credential, scope, and audit trail.

The managed application path is an identity and evidence chain

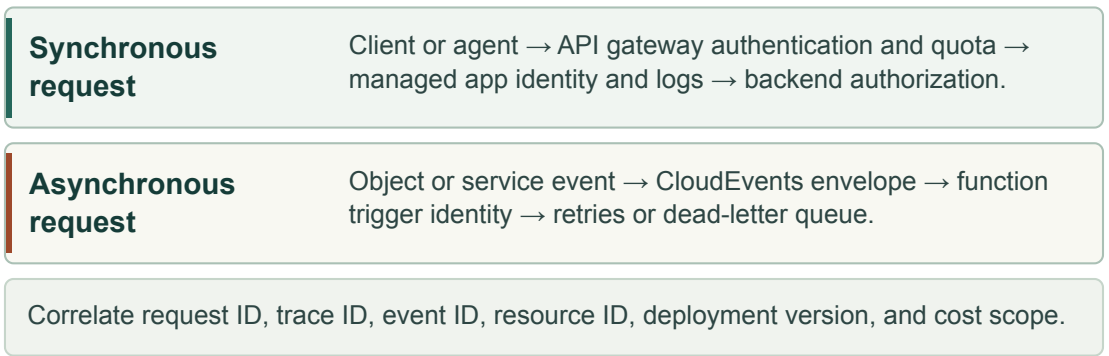


Figure 9-7 A managed application path is an identity and evidence chain rather than one trusted network.

Table 9-15 Identity chains should be explicit at every hop.

Hop	Principal	Authorization decision	Evidence
Consumer to gateway	User, application, partner, device, agent	May this consumer invoke this API product and operation?	Token claims, subscription, policy result, gateway request log
Gateway to backend	Gateway identity, certificate, managed identity, signed request	May the gateway call this backend endpoint?	Backend authentication result, route and policy trace
Application to managed service	Application or function workload identity	May this workload read or write the target data-plane resource?	Role assignment, service access log, trace span
Event source to route	Service identity or trusted publisher	May this source publish this event type to this bus or topic?	Event source configuration, publish record
Route or broker to function	Delivery service or managed trigger identity	May this subscription invoke the function and deliver this item?	Subscription, trigger config, delivery attempt, function invocation
Operator to control plane	Human or automation identity	May this principal change app, function, route, API, or policy resources?	Role assignment, activity log, deployment record

Network controls remain necessary but do not replace identity. App Service VNet integration primarily controls outbound access from an app into a virtual network; a private endpoint controls inbound private access to the app. Similar distinctions exist across providers. API gateways and functions can also have public, VPC-integrated, or private endpoints. DNS, route, firewall, and service access policies must agree, as established in the networking chapter.

9.9 Observability, SLOs, Quotas, and Cost

Application PaaS creates several operational data sources. Platform metrics reveal instance count, CPU, memory, request rate, invocation count, queue age, event delivery failure, gateway latency, and quota use. Application telemetry reveals business outcomes and dependency behavior. Activity logs reveal who changed the resource. Cost records reveal provisioned capacity and usage. A coherent service view correlates them by resource ID, deployment version, trace ID, event ID, consumer, and cost scope.

Table 9-16 Each PaaS resource requires its own operational evidence.

Resource	Primary SLI or signal	Important failure evidence	Cost driver
Managed application	Successful request ratio, latency, business transaction success	Unhealthy instance, failed deployment, start-up failure, dependency saturation	Plan or instance capacity, execution time, network, logs
Function	Successful invocation ratio, duration, queue age or trigger lag	Timeout, retry count, poison item, throttle, cold start, dependency failure	Invocations, memory/CPU duration, provisioned capacity, network
Event router	Delivery success, delivery latency, dead-letter count	Filter mismatch, expired delivery, inaccessible endpoint	Operations or events, delivery, data transfer
Message broker	Queue age, active messages, completion rate, dead-letter count	Lock loss, duplicate, poison message, session backlog	Broker capacity, operations, storage, throughput
Event stream	Ingress rate, consumer lag, partition skew, capture success	Throttling, hot partition, checkpoint failure, retention gap	Throughput or processing units, ingress, retention, capture
API gateway	Gateway success rate, backend latency, policy latency, quota rejection	Auth failure, backend timeout, policy error, capacity saturation	Gateway instance/tier, requests or data processing, network, telemetry

SLOs should follow the user-visible path. A gateway may be healthy while the backend fails. A function may execute successfully but process an event too late. A queue may retain every message while the business backlog violates its objective. Define composite indicators such as "99% of accepted uploads produce a processed result within five minutes," then decompose the path into event delivery, function execution, downstream write, and final availability.

A single request can incur multiple cost categories: gateway processing, application capacity, function execution, message operations, storage transactions, telemetry ingestion, and outbound data. The cheapest component is not necessarily the cheapest architecture. Serverless execution reduces idle compute but may increase request, logging, and data-transfer charges; a continuously allocated app may be cheaper for sustained traffic.

Table 9-17 A PaaS cost envelope should include every managed layer.

Layer	Typical charge basis	Optimization question
Application hosting	Instance or plan size and allocated time, sometimes request-based automatic scale	Can apps share capacity safely, scale down, or move to a more suitable plan?
Functions	Invocations, memory/CPU duration, provisioned or always-ready instances	Can work batch, reduce duration, control concurrency, or avoid unnecessary triggers?
Events and messages	Operations, throughput capacity, retention, storage, delivery	Are filters reducing noise, batches sized well, and retention matched to replay needs?
API management	Gateway capacity/tier, requests, data processing, network	Which APIs need premium isolation, caching, transformation, or global placement?
Observability	Log and trace ingestion, retention, query, export	Are sampling, dimensions, retention, and diagnostic categories tied to decisions?
Network and security	Private endpoints, NAT, load balancing, WAF, egress	Does the security boundary justify the path and eliminate accidental public transfer?

9.10 Azure Reference Control Plane

Azure provides a coherent reference because it exposes managed application hosting, functions, event routing, brokered messaging, streaming, and API management as separate resources under Azure Resource Manager. The resources integrate, but their object models remain visible.

Table 9-18 Azure reference resources and their control-plane purpose.

Capability	Key resources	Control-plane purpose
Application hosting	App Service plan, Web App, deployment slot, app settings, managed identity, private endpoint, diagnostic settings	Define runtime capacity, app lifecycle, release isolation, identity, access, health, telemetry, and cost scope
Functions	Function App, hosting plan, function, version, trigger, binding or connection, managed identity	Define event-addressed execution, capacity, concurrency, networking, retry, version, and evidence

Capability	Key resources	Control-plane purpose
Event routing	Event Grid system topic or custom topic, event subscription, filter, delivery endpoint, dead-letter destination	Match events from trusted sources to targets with retry and delivery evidence
Brokered messaging	Service Bus namespace, queue, topic, subscription, rule, dead-letter queue	Transfer commands or work with acknowledgments, sessions, transactions, duplicates, and dead lettering
Streaming	Event Hubs namespace, event hub, partition, consumer group, capture destination	Ingest and retain partitioned streams for independent consumers and replay
API management	API Management service, gateway, API, operation, backend, policy, product, subscription, portal, workspace	Productize APIs, enforce runtime policy, onboard consumers, version contracts, and collect usage evidence

App Service automatic scaling and Azure Monitor autoscale are related but different mechanisms. Automatic scaling lets the platform make scale-out decisions for supported web apps and plans. Azure Monitor autoscale uses explicit metric or schedule rules for supported resources. The choice should be recorded because it changes who owns thresholds, response timing, and evidence.

Azure Functions hosting choice likewise affects the contract. Microsoft currently recommends Flex Consumption for many dynamic scale applications where available, while Premium and Dedicated plans remain important for prewarmed capacity, advanced networking, predictable execution, or shared App Service resources. Existing Consumption workloads remain valid but should be assessed against current regional and feature requirements.

Event Grid, Service Bus, and Event Hubs should not be selected by familiarity. Event Grid is optimized for discrete event routing; Service Bus for reliable enterprise commands and work; Event Hubs for high-volume retained streams. API Management then provides the managed API lifecycle and gateway data plane across backends hosted in Azure, on premises, or other clouds.

9.11 Multicloud Translation

The providers expose comparable engineering intents with different product boundaries. The translation must preserve non-equivalence. For example, a provider may combine application hosting and function execution in one product family, or expose API manage-

ment through a cloud-native gateway focused on Kubernetes and microservices. The correct mapping identifies the resource contract, not merely the closest product name.

Table 9-19 Multicloud translation by engineering intent.

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Managed application runtime	App Service; selected Container Apps scenarios	Serverless App Engine (SAE)	Tencent Cloud Elastic Microservice (TEM)	veFaaS application and Web application forms; partial rather than exact App Service equivalence
Function runtime	Azure Functions	Function Compute 3.0	Serverless Cloud Function (SCF)	Function as a Service (veFaaS)
Event routing	Event Grid system/custom topics and subscriptions	EventBridge system/custom event buses, rules, targets	EventBridge buses, rules, connectors and targets	veFaaS triggers plus service-specific event centers and integrations; no exact general event-bus equivalence assumed
Reliable messaging	Service Bus queues and topics	Simple Message Queue, RocketMQ, RabbitMQ and related services	TDMQ family and queues/topics	RocketMQ, Kafka, BMQ and related managed messaging
Streaming	Event Hubs with Kafka compatibility	Message Queue for Apache Kafka and streaming services	CKafka and stream services	Kafka and cloud-native message engine BMQ
API management	Azure API Management	Traditional API Gateway and Cloud-native API Gateway, including AI gateway capabilities	Tencent Cloud API Gateway	API Gateway (APIG), including VKE discovery and AI gateway functions
Identity and evidence	Entra ID/RBAC, managed identity, Azure Monitor and activity logs	RAM, service roles, CloudMonitor, SLS and ActionTrail	CAM, service roles, Cloud Monitor, CLS and CloudAudit	IAM, service roles, Cloud Monitor, TLS and audit services

Provider product boundaries and international-site availability change. Treat the table as an intent map, then verify the current region, edition, API version, networking options, limits, and billing model.

Table 9-20 A selection worksheet prevents “serverless” from becoming the only requirement.

Question	Managed app	Function	Event router / broker / stream	API management
What initiates work?	HTTP, RPC, background loop, scheduler	Request or event trigger	Publisher, producer, or source service	API consumer request
How long does work run?	Continuous process or repeated requests	Bounded invocation or task	Delivery and retention independent of consumer runtime	Per-request gateway execution
Who owns work completion?	Application instance and its service contract	Function invocation and downstream transaction	Consumer acknowledgment, checkpoint, or subscription state	Backend application; gateway owns policy and routing only
What state is durable?	External database, storage, queue, or managed session store	External state, checkpoint, idempotency record	Message/event retention, offset, dead-letter record	API definition, policy, consumer entitlement, usage evidence
What is the main failure mode?	Unhealthy revision, capacity, dependency or configuration	Duplicate, timeout, retry storm, cold start, concurrency	Backlog, expiry, poison item, partition skew, replay gap	Auth/policy error, backend timeout, capacity or contract break

9.12 Lab 8: Deploy a Managed Web API

This lab validates one capability: a managed application resource can accept an immutable application artifact, expose a stable HTTPS endpoint, report application-defined health, carry a managed identity, emit logs, and be deleted as one scoped resource graph. The lab does not build a full CI/CD pipeline or API gateway; those are separate control planes.

9.12.1 Goal, Architecture, and Cost Boundary

Table 9-21 Lab 8 contract.

Item	Decision
Capability goal	Deploy and verify a minimal ASP.NET Core Web API on Azure App Service.
Primary resources	Resource group, Linux App Service plan, Web App, system-assigned managed identity, application settings and logging configuration.
Evidence	Deployment record, resource JSON, HTTPS response, /health response, health-check configuration, identity principal ID, application log, activity log, cleanup proof.

Item	Decision
Cost boundary	A Basic plan incurs allocated-time cost. Use one small instance, complete the lab promptly, and delete the resource group.
Not demonstrated	Multi-zone availability, traffic splitting, custom domain, private endpoint, database state, or production SLO.

Prerequisites: an Azure subscription, Azure CLI, .NET SDK, zip utility, and permission to create resource groups, App Service plans, web apps, identities, and diagnostic settings. The commands assume Bash and a region where Linux App Service is available.

```
export SUFFIX=$(openssl rand -hex 3)
export LOCATION=eastus
export RG=book-ch09-app-$SUFFIX
export PLAN=plan-ch09-$SUFFIX
export APP=bookapi-$SUFFIX

az login
az account show --output table
az group create --name "$RG" --location "$LOCATION"
```

9.12.2 Create the Minimal API

```
dotnet new web -n ManagedBookApi --framework net8.0
cd ManagedBookApi
cat > Program.cs <<'CS'
var builder = WebApplication.CreateBuilder(args);
builder.Logging.ClearProviders();
builder.Logging.AddConsole();

var app = builder.Build();

app.MapGet("/", () => Results.Ok(new {
    service = "managed-book-api",
    version = "1.0.0",
    lab = Environment.GetEnvironmentVariable("BOOK_LAB_ID")
}));

app.MapGet("/health", () => Results.Ok(new {
    status = "healthy",
    utc = DateTimeOffset.UtcNow
}));

app.MapGet("/api/info", (HttpContext context, ILogger<Program> log) => {
    var correlation = context.TraceIdentifier;
```

```
    log.LogInformation("BOOK_API request {CorrelationId}", correlation);
    return Results.Ok(new {
        correlationId = correlation,
        machine = Environment.MachineName,
        framework =
System.Runtime.InteropServices.RuntimeInformation.FrameworkDescription
    });
});

app.Run();
CS

dotnet publish -c Release -o publish
(cd publish && zip -r ../app.zip .)
```

The application exposes a business-neutral root response, an application-defined health endpoint, and an endpoint that records a correlation identifier. The artifact contains no environment-specific endpoint or credential.

9.12.3 Create the Managed Runtime

```
az appservice plan create   --resource-group "$RG"   --name "$PLAN"   --location
"$LOCATION"   --is-linux   --sku B1

az webapp create   --resource-group "$RG"   --plan "$PLAN"   --name "$APP"   --
runtime "DOTNETCORE:8.0"

az webapp identity assign   --resource-group "$RG"   --name "$APP"

az webapp config appsettings set   --resource-group "$RG"   --name "$APP"   --
settings BOOK_LAB_ID=lab-8 SCM_DO_BUILD_DURING_DEPLOYMENT=false

az webapp config set   --resource-group "$RG"   --name "$APP"   --always-on true
--http20-enabled true   --generic-configurations '{"healthCheckPath":"/health"}'

az webapp log config   --resource-group "$RG"   --name "$APP"   --application-
logging filesystem   --level information   --web-server-logging filesystem
```

If your Azure CLI version does not accept healthCheckPath through --generic-configurations, set the Health check path to /health in the portal or update the site configuration through the current App Service REST API. Record the method in the evidence packet.

9.12.4 Deploy the Artifact and Verify the Data Plane

```
az webapp deploy   --resource-group "$RG"   --name "$APP"   --src-path app.zip
--type zip

export BASE="https://$APP.azurewebsites.net"
curl -fsS "$BASE/"
curl -fsS "$BASE/health"
curl -fsS "$BASE/api/info"
```

All three requests must return HTTP success. Save the response bodies. The health response proves only that this instance can currently serve the health path; it does not prove multi-instance or zone resilience.

9.12.5 Verify the Control Plane and Evidence

```
az webapp show -g "$RG" -n "$APP"   --query
'{id:id,state:state,host:defaultHostName,httpsOnly:httpsOnly,identity:identity.p
rincipalId}'   --output json > webapp-resource.json

az webapp config show -g "$RG" -n "$APP"   --query
'{linuxFxVersion:linuxFxVersion,alwaysOn:alwaysOn,http20Enabled:http20Enabled,he
althCheckPath:healthCheckPath}'   --output json > webapp-config.json

az monitor activity-log list   --resource-group "$RG"   --offset 2h   --query
'[].
{time:eventTimestamp,operation:operationName.value,status:status.value,caller:ca
ller}'   --output json > activity-log.json

az webapp log download -g "$RG" -n "$APP" --log-file appservice-logs.zip
```

Inspect the log archive or briefly use az webapp log tail, then call /api/info again and locate the BOOK_API record. If filesystem log download is unavailable in the selected configuration, capture the platform's current log-stream evidence and record the limitation. The evidence packet should contain app.zip hash, deployment output, resource and configuration JSON, HTTPS responses, log evidence, activity log, and a short responsibility statement.

9.12.6 Cleanup and Zero-Remainder Proof

```
cd ..
az group delete --name "$RG" --yes --no-wait
az group wait --name "$RG" --deleted
az resource list --resource-group "$RG" --output table 2>/dev/null || true
```

The final resource query should return no resources. Also confirm that no custom DNS, certificate, external monitoring workspace, or deployment artifact was created outside the resource group. Record the deletion timestamp.

9.12.7 Common Failure Modes and Translation

Table 9-22 Lab 8 failure modes.

Symptom	Likely cause	Evidence and correction
Default site loads but /health fails	Artifact or startup command did not deploy correctly	Inspect deployment output, app logs, runtime stack and publish folder layout
Health path configured but no availability benefit	Only one instance exists	State the limitation; scale to two or more instances only in a separate reliability exercise
Identity exists but service access fails	No data-plane role assigned to the managed identity	Inspect principal ID and target role assignment; do not insert a secret as the first fix
Logs are empty	Logging category, provider, retention or timing is wrong	Call /api/info, inspect log configuration, wait for ingestion, then query the selected sink
Unexpected ongoing cost	App Service plan remains allocated after app deletion	Delete the entire resource group and verify the plan is gone

For Alibaba Cloud, repeat the intent with an SAE application: deploy a source package or image, configure health and scaling, assign RAM/service roles, enable ARMS or SLS evidence, and delete the application and namespace resources. On Tencent Cloud, use TEM with CAM, logs, health and application access. On Volcengine, use the veFaaS application or Web application form where suitable, but document the non-equivalence with a continuously allocated App Service plan.

9.13 Lab 9: Route an Object Event to a Cloud Function

This lab validates a second capability independently of Lab 8: an object-created event can be published by Azure Storage, filtered and routed by Event Grid, delivered to a named Azure Function, and correlated through platform evidence. The event is treated as at-least-once; the function records event identity rather than assuming uniqueness.

9.13.1 Goal, Architecture, and Cost Boundary

Table 9-23 Lab 9 contract.

Item	Decision
Capability goal	Create an Event Grid-triggered function and prove that uploading one blob produces a correlated function invocation.
Primary resources	Resource group, storage account and incoming container, Function App and hosting plan, Application Insights resource, Event Grid system topic/subscription.
Evidence	Function and trigger inventory, event subscription JSON, uploaded blob properties, Event Grid delivery evidence, function trace containing event ID and subject, activity log, cleanup proof.
Cost boundary	Use a consumption-oriented function plan and a small storage account. Logging and event operations can incur small charges; delete the resource group promptly.
Delivery assumption	At-least-once. Duplicate delivery is valid and must not create duplicate business effects.

Prerequisites: Azure CLI, Azure Functions Core Tools, Python 3.11, zip, and permission to create storage, Function App, Event Grid, monitoring, and role-assignment resources. The sample uses the classic Consumption plan for compact, broadly familiar commands. For production, evaluate Flex Consumption, Premium, or Dedicated against current networking, latency, concurrency, and regional requirements.

```
export SUFFIX=$(openssl rand -hex 3)
export LOCATION=eastus
export RG=book-ch09-event-$SUFFIX
export STORAGE=bookevt$SUFFIX
export FUNC=book-event-fn-$SUFFIX
export SUB=blob-created-to-function

az group create --name "$RG" --location "$LOCATION"
az storage account create   --name "$STORAGE"   --resource-group "$RG"   --
location "$LOCATION"   --sku Standard_LRS   --kind StorageV2

az functionapp create   --resource-group "$RG"   --name "$FUNC"   --storage-
account "$STORAGE"   --consumption-plan-location "$LOCATION"   --runtime python
--runtime-version 3.11   --functions-version 4   --os-type Linux
```

9.13.2 Create and Publish the Event Function

```
mkdir FunctionEventLab && cd FunctionEventLab
func init . --python -m V2

cat > function_app.py <<'FUNCTION_PY'
import json
import logging
import azure.functions as func

app = func.FunctionApp()

@app.function_name(name="OnBlobCreated")
@app.event_grid_trigger(arg_name="event")
def on_blob_created(event: func.EventGridEvent):
    data = event.get_json()
    record = {
        "event_id": event.id,
        "event_type": event.event_type,
        "subject": event.subject,
        "event_time": str(event.event_time),
        "url": data.get("url") if isinstance(data, dict) else None
    }
    logging.info("BOOK_EVENT %s", json.dumps(record, sort_keys=True))
FUNCTION_PY

cat > requirements.txt <<'REQUIREMENTS'
azure-functions
REQUIREMENTS

func azure functionapp publish "$FUNC" --python
```

The handler records the Event Grid event ID, type, subject, event time, and object URL. Logging is intentionally side-effect free, so replay is safe. A production handler would use event.id or a business idempotency key to protect any durable write or external action.

9.13.3 Create the Container and Event Route

```
export SUBSCRIPTION_ID=$(az account show --query id -o tsv)
export USER_OBJECT_ID=$(az ad signed-in-user show --query id -o tsv)
export STORAGE_ID=$(az storage account show -g "$RG" -n "$STORAGE" --query id -o
tsv)
export
FUNCTION_ID="/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RG/providers/Micros
oft.Web/sites/$FUNC/functions/OnBlobCreated"

az role assignment create   --assignee-object-id "$USER_OBJECT_ID"   --assignee-
```

```
principal-type User   --role "Storage Blob Data Contributor"   --scope
"$STORAGE_ID"

az storage container create   --account-name "$STORAGE"   --name incoming   --
auth-mode login

az eventgrid event-subscription create   --name "$SUB"   --source-resource-id
"$STORAGE_ID"   --endpoint-type azurefunction   --endpoint "$FUNCTION_ID"   --
included-event-types Microsoft.Storage.BlobCreated   --subject-begins-with
"/blobServices/default/containers/incoming/"
```

The event subscription is scoped to the storage account, filtered to BlobCreated events, and narrowed to the incoming container. The Azure Function resource ID is the destination. Event Grid manages endpoint validation for the Azure Function destination type. Save the subscription object and note any system topic created by the platform.

The Event Grid CLI and nested Azure Function resource API are volatile surfaces. If a provider CLI changes, preserve the resource intent: source storage account, BlobCreated filter, incoming-container subject prefix, Azure Function endpoint type, and named function resource.

9.13.4 Upload an Object and Correlate the Invocation

```
printf 'application-paas-event-lab
' > sample.txt

az storage blob upload   --account-name "$STORAGE"   --container-name incoming
--name sample.txt   --file sample.txt   --auth-mode login   --overwrite

az storage blob show   --account-name "$STORAGE"   --container-name incoming   -
-name sample.txt   --auth-mode login   --output json > uploaded-blob.json

az eventgrid event-subscription show   --name "$SUB"   --source-resource-id
"$STORAGE_ID"   --output json > event-subscription.json
```

Allow time for deployment, trigger synchronization, event delivery, and telemetry ingestion. Find the Application Insights resource that belongs to the Function App, then query for the BOOK_EVENT record.

```
export AI=$(az resource list   --resource-group "$RG"   --resource-type
microsoft.insights/components   --query '[0].name' -o tsv)

echo "Application Insights resource: $AI"

az monitor app-insights query   --resource-group "$RG"   --app "$AI"   --
analytics-query "traces | where message startswith 'BOOK_EVENT' | project
```

```
timestamp, message | order by timestamp desc | take 20"   --output json >
function-traces.json

cat function-traces.json
```

The trace must contain a subject ending in /incoming/sample.txt and a nonempty Event Grid event ID. If the CLI cannot query Application Insights in the current environment, use the Application Insights Logs experience or a supported log stream, save the query and result, and record the limitation. Do not replace evidence with a screenshot alone when machine-readable output is available.

9.13.5 Build the Evidence Packet

```
mkdir -p evidence

cp event-subscription.json evidence/01-event-subscription.json
cp uploaded-blob.json evidence/02-uploaded-blob.json
cp function-traces.json evidence/03-function-traces.json

az functionapp show -g "$RG" -n "$FUNC" -o json   > evidence/04-function-
app.json

az resource show --ids "$FUNCTION_ID" --api-version 2022-03-01 -o json   >
evidence/05-function-resource.json

az monitor activity-log list   --resource-group "$RG"   --offset 2h   --query
'[].
{time:eventTimestamp,operation:operationName.value,status:status.value,caller:ca
ller,resourceId:resourceId}'   --output json > evidence/06-activity-log.json

sha256sum function_app.py requirements.txt sample.txt   > evidence/07-artifact-
hashes.txt
```

Table 9-24 Lab 9 evidence packet.

Artifact	What it proves	What it does not prove
Event subscription JSON	Source, destination type, filter, subject prefix, provisioning state	That every future event will be processed within an SLO
Uploaded blob properties	Object identity, container, size, ETag and modification time	That the event was delivered
Function trace with event ID	The named function received and indexed one Event Grid envelope	Exactly-once delivery or durable business completion

Artifact	What it proves	What it does not prove
Function App and function resource JSON	Runtime resource, function identity, region and deployment state	Correct downstream authorization not exercised by this handler
Activity log	Who created or changed control-plane resources	Data-plane upload and internal retry details
Artifact hashes	Which source and test input were used	Source review quality or supply-chain provenance by itself

Write a one-paragraph interpretation that links the uploaded blob ETag and timestamp to the Event Grid subject and the BOOK_EVENT trace. State explicitly that Event Grid uses retryable delivery and that duplicate events are valid. A future side-effecting handler must persist a deduplication or idempotency record before acknowledging completion.

9.13.6 Failure Modes and Their Meaning

Table 9-25 Lab 9 failures reveal event-contract semantics.

Symptom	Likely cause	Evidence and correction
Function does not appear as an Azure Function destination	Function deployment or trigger indexing has not completed	Inspect Function App deployment logs and function inventory; wait for trigger synchronization
Blob upload succeeds but no trace appears	Subscription filter, endpoint, telemetry ingestion, or runtime startup is wrong	Inspect event subscription provisioning state, subject prefix, Function App logs and Event Grid metrics
Event subscription creation is rejected	Endpoint resource ID, source ID, role, region or schema contract is invalid	Compare resource IDs with current CLI documentation; use Azure Function endpoint type rather than a generic webhook
Trace appears more than once	Expected at-least-once retry or a second upload/overwrite generated another event	Compare event IDs; make side effects idempotent rather than suppressing valid delivery
User cannot create container or upload blob	Data-plane role assignment is missing or has not propagated	Verify Storage Blob Data Contributor at storage-account scope; wait and retry without using the account key
No Application Insights resource is found	Function App telemetry was disabled or created differently	Enable an approved telemetry sink, repeat the event, and preserve the query result and configuration evidence

9.13.7 Cleanup and Zero-Remainder Proof

```
cd ..
az group delete --name "$RG" --yes --no-wait
az group wait --name "$RG" --deleted

az graph query   -q "Resources | where resourceGroup =~ '$RG' | project id,type"
--output json > cleanup-proof.json

cat cleanup-proof.json
```

The final query should contain no resources. Confirm that the Event Grid system topic, event subscription, Application Insights resource, storage account, Function App, and role assignment were deleted with the resource group. If a telemetry workspace, dead-letter destination, or custom topic was created outside the group, remove it separately and include that proof.

9.13.8 Translate the Lab to China's Leading Cloud Platforms

Table 9-26 Translate Lab 9 by event-source, route, function, evidence, and cleanup intent.

Intent	Alibaba Cloud	Tencent Cloud	Volcengine
Object event source	OSS event or EventBridge source	COS event notification or EventBridge source	TOS trigger or supported service event source
Routing resource	EventBridge event bus and rule, or OSS-to-Function Compute integration	EventBridge bus and rule, or COS-to-SCF integration	veFaaS trigger configuration and supported event integration
Compute target	Function Compute function and version/alias	SCF function and version/alias	veFaaS function or application
Delivery evidence	EventBridge/FC metrics, logs, rule and target configuration	EventBridge/SCF logs, rule and delivery evidence	veFaaS trigger configuration, invocation logs and service metrics
Failure contract	Retry, dead-letter and function error handling as supported by the selected route	Retry, dead-letter and SCF error handling as supported	Trigger-specific retry and log semantics; document service differences
Cleanup proof	Delete rule, function, OSS resources and confirm no RAM/log resources remain	Delete rule, SCF, COS resources and confirm no CAM/log resources remain	Delete trigger, function/application, TOS resources and external log sinks

The platforms do not expose an identical resource graph. The translation preserves engineering intent: a named object event source, a filterable route, a versioned function target, explicit delivery behavior, machine-readable evidence, and verified retirement.

9.14 CloudDesk Reuse: Bounded Application Capabilities

CloudDesk FinOps Agent may later call the managed capabilities introduced in this chapter, but it must not receive a generic deployment shell, unrestricted API gateway administrator role, or arbitrary function-invocation credential. Each capability should expose a narrow schema, scoped identity, risk class, plan or dry run where appropriate, evidence envelope, and cleanup or rollback semantics.

Table 9-27 CloudDesk capability handoff from the Application PaaS chapter.

Capability	Purpose	Risk	Required evidence
app.hosting.deploy	Deploy an approved application artifact to a named managed runtime and return endpoint, version, identity and health state	R3 - controlled mutation	Artifact digest, target resource ID, configuration diff, deployment result, health result, activity log, cost scope
app.health.check	Read application health configuration and execute a bounded health or synthetic check	R0/R1 - observe	Resource ID, health path, response, instance state, timestamp, deployment version
function.invoke	Invoke one allowlisted function with validated input and a bounded timeout	R1-R3 depending on side effect	Function version, caller identity, request/event ID, result, duration, downstream evidence and cost
event.route.create	Create or update a narrowly scoped event route after plan review	R3 - controlled mutation	Source, filter, target, retry/dead-letter settings, plan digest, approval, provisioning state, delivery test
api.gateway.publish (optional extension)	Publish a reviewed API contract and bounded gateway policy to an approved environment	R3 - controlled mutation	OpenAPI digest, backend, policy diff, security review, deployment result, synthetic test and rollback target

The first four capabilities are the architecture-pack handoff for this chapter. The optional API capability is included because API management is part of the chapter's resource model; it should be enabled only after consumer identity, contract review, policy scope, and rollback are defined.

9.15 Key Takeaways

- Application PaaS is not one product category. Managed application runtimes, functions, event routers, brokers, streams, and API gateways are different control-plane contracts.
- A managed application is a resource graph containing plan, app, artifact, configuration, identity, endpoint, release, health, scale, telemetry, policy, quota, cost, and retirement objects.
- Build artifact, deployment version, slot or revision, runtime instance, traffic mapping, configuration, and external state must be versioned and governed separately.
- A function is event-addressed compute. Trigger semantics, hosting plan, concurrency, retry, timeout, version, identity, and evidence are part of the function resource.
- Events announce state change, messages assign work, and streams preserve ordered observations. Product names do not erase these semantic differences.
- At-least-once delivery makes duplicate handling and idempotent business effects normal requirements, not exceptional defenses.
- API management productizes APIs through contracts, versions, products, subscriptions, policies, developer experience, gateway runtime, and evidence.
- Network reachability, gateway policy, workload identity, event-delivery authority, and backend authorization are separate trust decisions.
- SLOs must follow the user-visible path across gateway, app, event, function, broker, and downstream state; component health alone is insufficient.
- PaaS economics include provisioned capacity, execution, event and message operations, API gateway processing, telemetry, private connectivity, and data transfer.
- Multicloud translation should preserve resource intent, responsibility boundary, delivery semantics, evidence, and cost while recording non-equivalence.
- CloudDesk should consume narrow, evidence-producing application capabilities rather than unrestricted cloud administration interfaces.

References

1. NIST. The NIST Definition of Cloud Computing, SP 800-145. https://csrc.nist.gov/publications/detail/sp/800-145/final

2. Microsoft Learn. Azure App Service overview. https://learn.microsoft.com/azure/app-service/overview

3. Microsoft Learn. App Service plans. https://learn.microsoft.com/azure/app-service/overview-hosting-plans

4. Microsoft Learn. Set up staging environments in Azure App Service. https://learn.microsoft.com/azure/app-service/deploy-staging-slots

5. Microsoft Learn. Monitor App Service instances by using Health check. https://learn.microsoft.com/azure/app-service/monitor-instances-health-check

6. Microsoft Learn. Scale up and scale out an Azure App Service app. https://learn.microsoft.com/azure/app-service/manage-scale-up

7. Microsoft Learn. Automatic scaling in Azure App Service. https://learn.microsoft.com/azure/app-service/manage-automatic-scaling

8. Microsoft Learn. Managed identities for App Service and Azure Functions. https://learn.microsoft.com/azure/app-service/overview-managed-identity

9. Microsoft Learn. Integrate an app with an Azure virtual network. https://learn.microsoft.com/azure/app-service/overview-vnet-integration

10. Microsoft Learn. Use private endpoints for Azure Web App. https://learn.microsoft.com/azure/app-service/overview-private-endpoint

11. Microsoft Learn. Configure diagnostic logging for apps in Azure App Service. https://learn.microsoft.com/azure/app-service/troubleshoot-diagnostic-logs

12. Microsoft Learn. Azure Functions overview. https://learn.microsoft.com/azure/azure-functions/functions-overview

13. Microsoft Learn. Azure Functions triggers and bindings concepts. https://learn.microsoft.com/azure/azure-functions/functions-triggers-bindings

14. Microsoft Learn. Azure Functions hosting options. https://learn.microsoft.com/azure/azure-functions/functions-scale

15. Microsoft Learn. Azure Functions concurrency. https://learn.microsoft.com/azure/azure-functions/functions-concurrency

16. Microsoft Learn. Azure Functions error handling and retries. https://learn.microsoft.com/azure/azure-functions/functions-bindings-error-pages

17. Microsoft Learn. Azure Event Grid trigger for Azure Functions. https://learn.microsoft.com/azure/azure-functions/functions-bindings-event-grid-trigger

18. Microsoft Learn. Work with Event Grid triggers and bindings in Azure Functions. https://learn.microsoft.com/azure/azure-functions/event-grid-how-tos

19. Microsoft Learn. Introduction to Azure Event Grid. https://learn.microsoft.com/azure/event-grid/overview

20. Microsoft Learn. Azure Event Grid concepts. https://learn.microsoft.com/azure/event-grid/concepts

21. Microsoft Learn. System topics in Azure Event Grid. https://learn.microsoft.com/azure/event-grid/system-topics

22. Microsoft Learn. Azure Event Grid delivery and retry. https://learn.microsoft.com/azure/event-grid/delivery-and-retry

23. Microsoft Learn. Azure Service Bus messaging overview. https://learn.microsoft.com/azure/service-bus-messaging/service-bus-messaging-overview

24. Microsoft Learn. Azure Event Hubs overview. https://learn.microsoft.com/azure/event-hubs/event-hubs-about

25. Microsoft Learn. Compare Azure messaging services. https://learn.microsoft.com/azure/service-bus-messaging/compare-messaging-services

26. Microsoft Learn. Azure API Management key concepts. https://learn.microsoft.com/azure/api-management/api-management-key-concepts

27. Microsoft Learn. Azure API Management policies. https://learn.microsoft.com/azure/api-management/api-management-howto-policies

28. Microsoft Learn. Import and publish an OpenAPI specification in Azure API Management. https://learn.microsoft.com/azure/api-management/import-and-publish

29. Microsoft Learn. Azure API Management workspaces. https://learn.microsoft.com/azure/api-management/workspaces-overview

30. Microsoft Learn. Azure messaging services documentation. https://learn.microsoft.com/azure/messaging-services/

31. Cloud Native Computing Foundation. CloudEvents Specification. https://cloudevents.io/

32. OpenAPI Initiative. OpenAPI Specification. https://spec.openapis.org/oas/latest.html

33. AsyncAPI Initiative. AsyncAPI Specification. https://www.asyncapi.com/docs/reference/specification/latest

34. OpenTelemetry. Specification. https://opentelemetry.io/docs/specs/

35. FinOps Foundation. FinOps Framework. https://www.finops.org/framework/

36. Alibaba Cloud. Serverless App Engine documentation. https://www.alibabacloud.com/help/en/sae/

37. Alibaba Cloud. Function Compute documentation. https://www.alibabacloud.com/help/en/functioncompute/

38. Alibaba Cloud. EventBridge documentation. https://www.alibabacloud.com/help/en/eventbridge/

39. Alibaba Cloud. Message Queue for Apache RocketMQ documentation. https://www.alibabacloud.com/help/en/apsaramq-for-rocketmq/

40. Alibaba Cloud. Cloud-native API Gateway documentation. https://www.alibabacloud.com/help/en/api-gateway/

41. Alibaba Cloud. Resource Orchestration Service resource types for API and application services. https://www.alibabacloud.com/help/en/ros/developer-reference/list-of-resource-types-by-service

42. Tencent Cloud. Tencent Cloud Elastic Microservice documentation. https://www.tencentcloud.com/document/product/1094

43. Tencent Cloud. Serverless Cloud Function documentation. https://www.tencentcloud.com/document/product/583

44. Tencent Cloud. EventBridge documentation. https://www.tencentcloud.com/document/product/1108

45. Tencent Cloud. TDMQ documentation. https://www.tencentcloud.com/document/product/1179

46. Tencent Cloud. API Gateway documentation. https://www.tencentcloud.com/document/product/628

47. Volcengine. Volcano Engine Function as a Service documentation. https://www.volcengine.com/docs/6662

48. Volcengine. veFaaS overview. https://www.volcengine.com/docs/6662/162366

49. Volcengine. TOS-triggered function service example. https://www.volcengine.com/docs/6662/1119948

50. Volcengine. API Gateway documentation. https://www.volcengine.com/docs/6569

51. OWASP. API Security Top 10. https://owasp.org/API-Security/

52. IETF. Idempotency-Key HTTP Header Field. https://datatracker.ietf.org/doc/draft-ietf-httpapi-idempotency-key-header/

53. Microsoft Azure Well-Architected Framework. Reliability design principles. https://learn.microsoft.com/azure/well-architected/reliability/principles

54. Microsoft Azure Architecture Center. Asynchronous messaging options. https://learn.microsoft.com/azure/architecture/guide/technology-choices/messaging

55. Microsoft Learn. Azure Retail Prices API overview. https://learn.microsoft.com/rest/api/cost-management/retail-prices/azure-retail-prices

Part III / Chapter 10

Containers, Registries, and Software Supply Chains

How Content-Addressed Artifacts, Registry Control Planes, and Verifiable Evidence Turn Images into Governed Release Units

Containerization changes more than packaging. It moves the operational release boundary from a configured machine to an OCI artifact identified by digest, distributed through a registry, and surrounded by evidence about composition, origin, integrity, risk, approval, and intended use. The resulting software supply chain is a cloud control plane, not a sequence of Docker commands.

In This Chapter

- Treat the container image as a content-addressed release unit rather than as a portable virtual machine.
- Use the OCI image, index, descriptor, distribution, subject, and referrers contracts to reason about images and derivative evidence.
- Separate mutable tags, immutable digests, semantic release versions, deployment revisions, and promotion records.
- Model the build as a governed contract over source, base images, dependencies, parameters, secrets, builder identity, network access, and platform targets.
- Design minimal, non-root, multi-stage images while distinguishing small size from actual security and maintainability.
- Operate a registry as both a cloud control plane and an artifact data plane with separate identity, network, retention, replication, and evidence obligations.
- Understand what SBOMs, provenance, signatures, scan results, VEX documents, and test attestations can and cannot prove.
- Implement policy gates from source review through build, registry, promotion, admission, and runtime response.

- Extend supply-chain reasoning to AI-native artifacts such as model weights, tokenizers, adapters, prompts, evaluation sets, and safety policies.
- Complete an independent Azure lab that builds, pushes, scans, signs, and verifies an image by digest, then produces a reusable evidence packet.
- Translate the engineering intent to Alibaba Cloud ACR, Tencent Cloud TCR, and Volcengine CR, and expose bounded capabilities for CloudDesk FinOps Agent.

The simplest container workflow has four commands: build, tag, push, and run. That workflow is sufficient to demonstrate portability, but it is not sufficient to establish trust. A production consumer needs to know which source revision produced the bytes, which base image and dependencies entered the build, which builder executed it, whether the result was scanned, who authorized its release, whether the artifact was modified, and whether the same artifact reached the target environment. A registry can answer those questions only when images and evidence are treated as governed resources.

The course sequence behind this chapter was intentionally practical: build a multi-stage image, scan it, push it to a private registry, authorize the pull identity, attach a signature, and verify the result. This chapter retains that sequence but raises the unit of analysis. The subject is no longer a Dockerfile or an ACR repository in isolation. It is the complete release path from source to runtime, with identities and verifiable evidence at every boundary.

Central claim. A container image becomes a governed release unit when its immutable digest is linked to trusted build evidence, authorized registry operations, policy decisions, and runtime verification. Portability without provenance is only portable uncertainty.

10.1 Containerization Moves the Release Boundary

Traditional server delivery often combines several responsibilities on one mutable machine: operating-system configuration, runtime installation, application binaries, service accounts, certificates, and operational scripts. A deployment may succeed because that particular machine has accumulated the expected state. Reproducing the same environment requires either an image of the machine or a configuration process capable of recreating it.

A container changes that boundary. The runtime instance is still a process constrained by namespaces, cgroups, filesystem mounts, capabilities, seccomp, and other operating-system controls, but the deployable application filesystem is assembled from an image. The

image can be built before deployment, identified independently of the host, copied between compliant registries, and instantiated by different runtimes. The host and orchestrator still matter, but they no longer contain the application release as an implicit local state.

This shift is the reason containers changed PaaS. Early application platforms were often organized around supported languages and framework versions. Container images move the unit of delivery toward an explicit runtime artifact. Managed application platforms, serverless container services, Kubernetes, batch systems, and AI serving platforms can all consume the same image while applying different control-plane contracts for networking, scaling, identity, health, and cost.

Containerization moves the release boundary to an immutable artifact

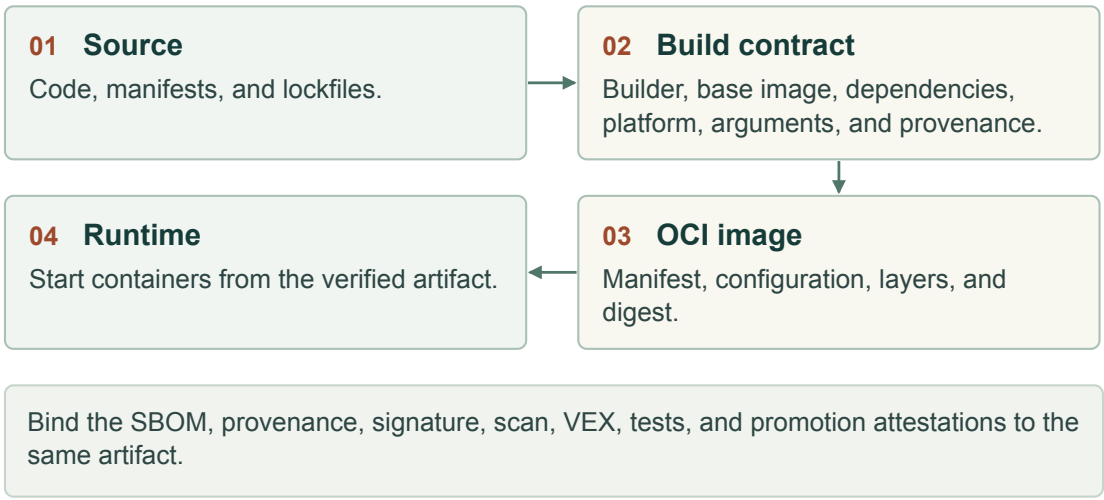


Figure 10-1 Containerization moves the release boundary from a configured machine to a content-addressed artifact and its evidence.

Table 10-1 Containerization separates artifacts that were often hidden inside a server.

Object	Primary responsibility	What must remain explicit
Source revision	Defines reviewed application intent.	Repository, commit, branch policy, dependency declarations, and build definition.
Build environment	Transforms source and materials into an artifact.	Builder identity, toolchain, network access, parameters, secrets, platform, and isolation.

Object	Primary responsibility	What must remain explicit
OCI image	Packages runtime filesystem and configuration.	Digest, manifests, config, layers, architecture, entrypoint, labels, and media types.
Registry	Stores and distributes images and related artifacts.	Identity, network, repository scope, retention, replication, evidence discovery, and audit.
Runtime	Instantiates and constrains a container process.	Image digest, policy decision, runtime identity, filesystem mutability, resources, and host boundary.
Release record	Explains why and where an artifact was promoted.	Approval, source environment, target environment, policy version, timestamp, and rollback subject.

The distinction between image and container is therefore architectural. An image is a stored artifact graph. A container is a runtime instance created from that artifact under a particular runtime and host policy. Signing an image does not sign the running process, and scanning an image does not inspect every runtime state. Conversely, runtime hardening cannot repair an unknown or untrusted build history. Supply-chain and runtime controls are complementary.

10.2 OCI Image, Index, and Distribution Contracts

The Open Container Initiative standardizes three related concerns: how an image is represented, how a low-level runtime executes a container bundle, and how content is distributed through registries. The image and distribution specifications are central to this chapter because they define the portable artifact and registry data-plane contracts used by cloud platforms.

An OCI image is not a single archive. It is a graph of descriptors. A descriptor identifies content by media type, digest, and size, and may carry annotations and a target platform. An image manifest points to an image configuration and one or more layer blobs. The image configuration contains runtime defaults and filesystem metadata. An image index, sometimes called a manifest list in adjacent ecosystems, can point to multiple manifests for different operating systems and processor architectures.

An OCI image is a content-addressed graph

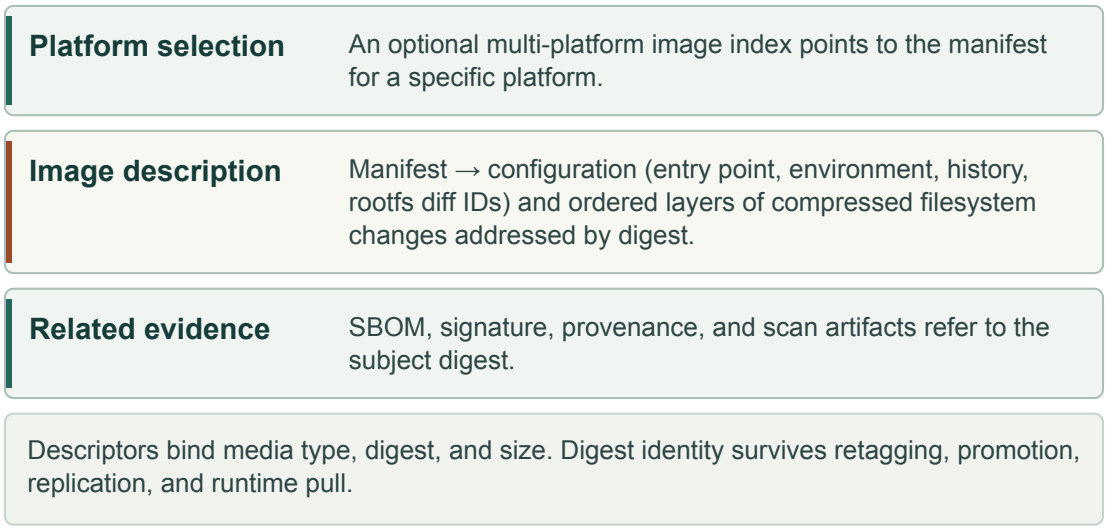


Figure 10-2 OCI images are graphs of descriptors and content-addressed objects.

Table 10-2 Core OCI objects and their architectural role.

OCI object	Role	Governance concern
Descriptor	Binds a media type, digest, size, annotations, and optional platform to content.	Digest verification, supported algorithms, media-type handling, and metadata trust.
Image manifest	Describes one runnable image configuration and its ordered layers.	Stable subject for signing, scanning, SBOM, promotion, and runtime pinning.
Image config	Defines runtime defaults and root filesystem history.	Entrypoint, command, user, environment, architecture, created metadata, and history leakage.
Layer blob	Carries a compressed filesystem change set.	Secret leakage, package inventory, cache reuse, deduplication, and deletion semantics.
Image index	Selects one or more manifests, typically by platform.	Multi-platform completeness, per-platform evidence, and index-versus-manifest signing scope.
Artifact manifest / reference type	Represents non-image OCI artifacts or evidence referring to a subject.	Artifact type, discoverability, subject binding, retention, and promotion with referrers.

OCI object	Role	Governance concern
Distribution API	Standardizes push, pull, manifest, blob, tag, and referrer operations.	Authentication, authorization, throttling, network exposure, compatibility, and audit.

Content addressability means that the identifier is derived from the bytes. If the content changes, its digest changes. The registry can deduplicate shared blobs and the consumer can verify what it received. This property gives the supply chain a stable subject. Evidence can refer to an exact digest even while tags, release channels, and deployment environments change.

OCI Image and Distribution Specification 1.1 introduced a standard subject relationship and Referrers API. A signature, SBOM, provenance statement, scan result, or other artifact can identify an image manifest as its subject. A consumer can then query the registry for artifacts that refer to that digest. The result is not merely extra files in a repository; it is an evidence graph whose relationships survive tag movement.

10.3 Tags, Digests, Versions, and Promotion

A tag is convenient for humans and automation, but it is a mutable reference. The tag production, stable, v1, or even 1.4.2 can be moved unless the registry enforces immutability. A digest is the immutable content identity. A semantic version describes release intent. A deployment revision records a platform-specific rollout. A promotion record explains the movement of an already built artifact between environments. Conflating these concepts weakens rollback and audit.

Table 10-3 Release identifiers answer different questions.

Identifier	Question answered	Mutability and use
Repository name	Which artifact family is this?	Long-lived namespace; not a version.
Tag	Which artifact should this channel or label resolve to now?	Usually mutable; useful for discovery and workflows, unsafe as sole production identity.
Digest	Which exact manifest bytes are intended?	Immutable content identity; preferred for signing, verification, promotion, and runtime pinning.
Semantic version	What producer-visible release intent does this artifact represent?	A human contract; should map to a digest and be protected against reassignment.

Identifier	Question answered	Mutability and use
Deployment revision	Which platform configuration is serving traffic?	Platform resource; can combine image digest, configuration, identity, and scale settings.
Promotion record	Why was this digest approved for the next environment?	Append-only evidence; should identify source, target, policy, approver, and subject digest.

Central claim. Build once, promote the same digest. Rebuilding for staging and production produces different artifacts even when the source commit is the same, and therefore breaks the strongest form of release equivalence.

Promotion can copy an image and its referrers to another registry or region, add a protected environment tag to the same digest, or update deployment desired state to reference the approved digest. The mechanism can vary, but the subject should not. If an artifact is rebuilt, the new digest requires new provenance, scanning, signing, approval, and verification.

Multi-platform images introduce another decision: sign and verify the index, each platform manifest, or both. The index gives one release identity across platforms, while per-platform manifests represent the actual runtime bytes. A policy should specify the expected scope and ensure that the platform selected by the runtime has the required evidence.

10.4 Build as a Governed Supply-Chain Contract

A successful build proves only that a builder produced output. It does not prove that the source was reviewed, the dependencies were expected, the builder was isolated, the base image was approved, the secrets were protected, or the result was reproducible. The build must therefore be modeled as a control-plane contract, not as a local command.

The build path is a chain of identities, inputs, and claims

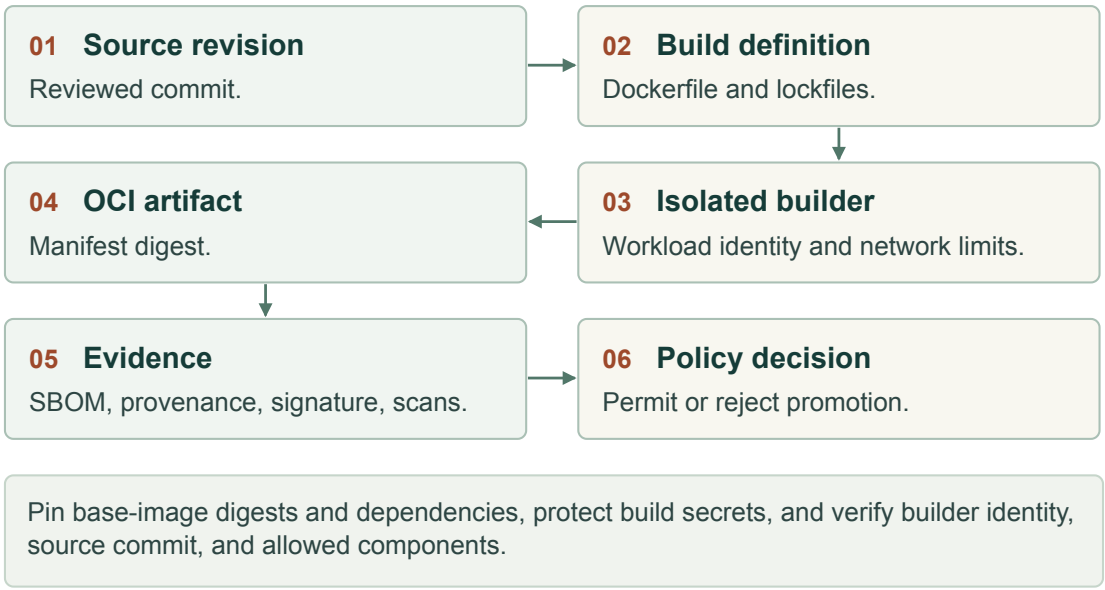


Figure 10-5 The build path is a chain of identities, inputs, artifacts, claims, and policy decisions.

Table 10-4 Inputs and conditions that determine the build result.

Build element	Examples	Risk if implicit
Source identity	Repository URI, commit digest, branch, submodules.	Unreviewed or substituted source can enter the release.
Build definition	Dockerfile, build script, buildpack, Bake file, workflow.	Local or modified instructions produce an unrecorded artifact.
Base image	Repository and digest for runtime and build stages.	Mutable tags or compromised bases change the artifact silently.
Dependencies	Package lock files, package repositories, downloaded tools.	Resolution drift and dependency confusion undermine repeatability.
Build platform	Builder implementation, version, OS, architecture, isolation.	A privileged or compromised builder can alter output and evidence.
Parameters	Build arguments, target stage, feature flags, platform list.	Unexpected parameters produce a valid but unauthorized variant.
Network access	Allowed package registries, mirrors, time sources, external downloads.	Unbounded network access makes material capture and hermeticity difficult.

Build element	Examples	Risk if implicit
Secrets and identity	Registry token, signing identity, package credentials.	Secrets leak into layers, logs, cache, provenance, or untrusted steps.
Output contract	Image name, digest, SBOM, provenance, signatures, test results.	The pipeline cannot establish what must be consumed and verified.

10.4.1 Repeatability, Reproducibility, Hermeticity, and Isolation

These properties are related but not interchangeable. A repeatable build can be run again by the same process. A reproducible build produces equivalent output from the same declared inputs, often on another trusted system. A hermetic build obtains all inputs through controlled, declared channels and does not depend on undeclared external state. An isolated build prevents one build or tenant from influencing another. An organization may not achieve every property immediately, but it should state which guarantees its provenance claims support.

Table 10-5 Build properties and the evidence they require.

Property	Meaning	Useful evidence
Repeatable	The build process can be invoked again with the same intended configuration.	Versioned workflow, tool versions, parameter record, build logs.
Reproducible	Independent builds from the same inputs produce equivalent output.	Source and material digests, deterministic configuration, comparison result.
Hermetic	Only declared inputs influence the build.	Network policy, material list, dependency mirror, sandbox or runtime trace.
Isolated	Concurrent or prior workloads cannot modify the build or its evidence.	Ephemeral builder, tenant boundary, immutable worker image, protected signing identity.
Authenticated	The producer can identify the builder and evidence issuer.	Workload identity, signed provenance, certificate or transparency evidence.
Policy-conformant	The build satisfies expected source, builder, parameter, and dependency rules.	Verifier decision, policy version, exception record, subject digest.

SLSA expresses progressively stronger build provenance and build-platform requirements. Its value is not the label alone but the verification model: consumers compare the provenance of a subject digest with expectations about source, builder, parameters, and materials. Provenance that is generated but never verified is only stored metadata.

10.5 Engineering Minimal and Maintainable Images

A multi-stage build separates compilation dependencies from the runtime image. BuildKit can parallelize stages, reuse cache, target multiple platforms, and generate provenance and SBOM attestations. These capabilities support small, repeatable images, but the security outcome still depends on base-image policy, package selection, user identity, filesystem permissions, and patch cadence.

Table 10-6 Image hardening decisions and their trade-offs.

Decision	Benefit	Trade-off or caveat
Multi-stage build	Removes compilers, package managers, and build-only files from runtime.	Files copied from build stages can still contain secrets or unneeded dependencies.
Minimal base image	Reduces packages, attack surface, transfer time, and scan noise.	May remove shells, certificates, time zones, or diagnostics needed for operation.
Run as non-root	Reduces privilege inside the container and limits common escalation paths.	File ownership, ports, volume permissions, and legacy software may need changes.
Read-only root filesystem	Prevents many unauthorized writes and persistence techniques.	Applications need explicit writable mounts for temporary and durable state.
Pinned base digest	Makes the build input explicit and prevents tag drift.	The organization must monitor and deliberately update the pinned base.
Locked dependencies	Constrains package resolution to reviewed versions.	Lock files must be updated, verified, and rebuilt as vulnerabilities emerge.
Health endpoint	Provides application-specific readiness or liveness evidence.	A superficial check can report health while dependencies or business functions fail.
Debug image separation	Keeps production image minimal while preserving troubleshooting tools elsewhere.	Operators need a controlled method to attach ephemeral debug tooling.

```
# Syntax intentionally omits a fixed tool version; pin approved digests in
production.
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY ManagedBookApi.csproj .
RUN dotnet restore --locked-mode
COPY . .
```

```
RUN dotnet publish -c Release -o /out /p:UseAppHost=false

FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS runtime
WORKDIR /app
ENV ASPNETCORE_URLS=http://+:8080
COPY --from=build /out .
USER $APP_UID
EXPOSE 8080
ENTRYPOINT ["dotnet", "ManagedBookApi.dll"]
```

For a production Dockerfile, replace the base tags with organization-approved digests, document the patch and rebuild policy, and verify that the chosen base defines the non-root user expected by USER.

Small is not synonymous with secure. An image with one statically linked binary can still contain a vulnerable library, unsafe configuration, or malicious code. A larger image can be well maintained and correctly constrained. Size is an operational signal; trust is a claim supported by composition, provenance, review, policy, and runtime controls.

10.6 Registries as Cloud Control Planes and Artifact Data Planes

A managed registry is commonly described as image storage, but its cloud resource model is broader. The control plane creates the registry instance, chooses region and service tier, configures identity modes and network boundaries, manages retention and replication, enables diagnostics, and defines automation. The data plane receives and serves manifests, blobs, tags, and referrer artifacts. These contracts must be secured separately.

A registry has distinct control-plane and data-plane contracts

Control plane	Registry instance, SKU, region, network, identity, retention, replication, tasks, diagnostics, and policy.
Data plane	Authenticate → push, pull, or list → resolve tag to digest → retrieve manifest and layers → attach or discover referrers.

Managers govern the registry. Publishers push images and attach evidence. Consumers pull and verify before use.

Figure 10-4 Registry management and repository operations belong to different planes and identities.

Table 10-7 Registry control-plane objects and their operational purpose.

Control-plane object	Purpose	Evidence and policy
Registry instance	Defines regional service, SKU, endpoints, encryption, identity mode, and resource ownership.	Resource ID, configuration history, activity log, cost scope, service health.
Namespace / repository policy	Partitions artifact families, teams, or environments and constrains operations.	Role assignments, ABAC conditions, repository attributes, audit events.
Network configuration	Controls public access, private endpoints, VPC/VNet reachability, allowlists, and trusted services.	Endpoint state, DNS, connection records, flow or firewall evidence.
Retention / cleanup	Removes untagged or obsolete artifacts under defined rules.	Policy version, dry-run result, deletion log, protected-release exception.
Replication / synchronization	Places artifacts near consumers or provides regional recovery.	Source and target digest equality, replication state, lag, failure event.
Build or task resource	Executes server-side builds, base-image triggers, or scheduled maintenance.	Task identity, source, run record, logs, output digest, provenance.
Diagnostic setting	Exports control and data-plane activity for monitoring and audit.	Log destination, categories, retention, correlation with identity and digest.
Signing / trust integration	Provides or connects signing keys, certificates, managed signing, and verification policy.	Signer identity, certificate chain, timestamp, trust policy, verification result.

10.6.1 Identity: Human, Builder, Signer, and Runtime

The same identity should not build, sign, approve, and deploy without explicit justification. Human developers need discovery and troubleshooting access. Builders need read access to sources and bases and write access to a bounded repository. Signers need permission to invoke a protected key or managed signing service, but not necessarily to change the image. Runtime identities need pull access, ideally to only the repositories they consume. Security and release teams need evidence access without broad artifact mutation authority.

Table 10-8 Least-privilege registry roles by actor.

Actor	Required capability	Avoid granting
Developer	Read metadata, pull approved images, and push only to development repositories when needed.	Registry administration, production deletion, unrestricted signing keys.

Actor	Required capability	Avoid granting
CI builder	Push manifests and layers, attach generated evidence, read approved bases.	Human interactive credentials, production promotion authority, key export.
Signer	Sign a specified digest using a protected identity and key.	Ability to rebuild or retag the subject without independent policy.
Release promoter	Copy or retag an already verified digest and record approval.	Source modification or signing-key administration.
Runtime / orchestrator	Pull exact approved digests and retrieve evidence needed for verification.	Push, delete, tag mutation, or broad catalog access unless required.
Security scanner	Read subject manifests and layers; attach scan evidence if authorized.	Tag mutation, deletion, unrelated repository access.
Registry operator	Manage service configuration, network, retention, and diagnostics.	Implicit repository data access where control- and data-plane roles are separate.

Azure Container Registry now exposes an ABAC-enabled role-assignment mode for repository permissions. In that mode, the roles Container Registry Repository Reader, Writer, and Contributor replace legacy AcrPull, AcrPush, and AcrDelete behavior for repository data operations. A management role such as Contributor can administer the registry resource without automatically granting repository read or write. This separation is exactly the control-plane/data-plane distinction established earlier in the book.

10.6.2 Network, Retention, Replication, and Cost

Private connectivity can prevent registry data traffic from traversing a public endpoint, but it does not replace repository authorization or signature verification. Replication improves locality and resilience but expands the number of regions and policies that must preserve the same digest and evidence graph. Retention reduces storage cost and risk from obsolete artifacts but can accidentally delete evidence or rollback subjects if tags and referrers are not handled together.

Table 10-9 Registry economics and their engineering controls.

Cost driver	Typical cause	Control
Stored layers and artifacts	Unused tags, duplicate variants, retained referrers, large model or debug images.	Retention rules, repository inventory, layer reuse, release protection, artifact-class policies.

Cost driver	Typical cause	Control
Data transfer	Cross-region pulls, public egress, repeated node scale-out, remote build context.	Regional replication, private access, caching, image minimization, workload placement.
Request and throughput capacity	Large clusters, many small layers, repeated tag resolution, evidence queries.	Digest pinning, image pre-pull, request planning, appropriate registry tier, lazy loading where supported.
Scanning and security services	Automatic rescans, premium scanners, retention of reports.	Risk-based scope, scan cadence, evidence retention, VEX, policy on stale results.
Signing and key services	Certificate issuance, HSM or Key Vault operations, timestamping, managed signing.	Key hierarchy, short-lived identity, batching where safe, explicit retention and verification policy.
Replication and recovery	Multiple regional replicas and synchronized artifact graphs.	Place replicas near justified consumers; verify that referrers and policies move with subjects.

10.7 Evidence: SBOM, Provenance, Signature, Scan, VEX, and Tests

Software supply-chain evidence is often discussed as if each artifact were a security certificate. It is more accurate to treat each as a typed claim about a subject digest. An SBOM claims composition. Provenance claims how an artifact was produced. A signature binds an identity to the subject descriptor. A vulnerability report records a scanner result at a particular time. A VEX document communicates whether a known vulnerability is believed to affect a product. Test evidence records what was executed and what passed. None of these claims alone proves that the software is safe.

OCI reference types turn a registry into an evidence graph

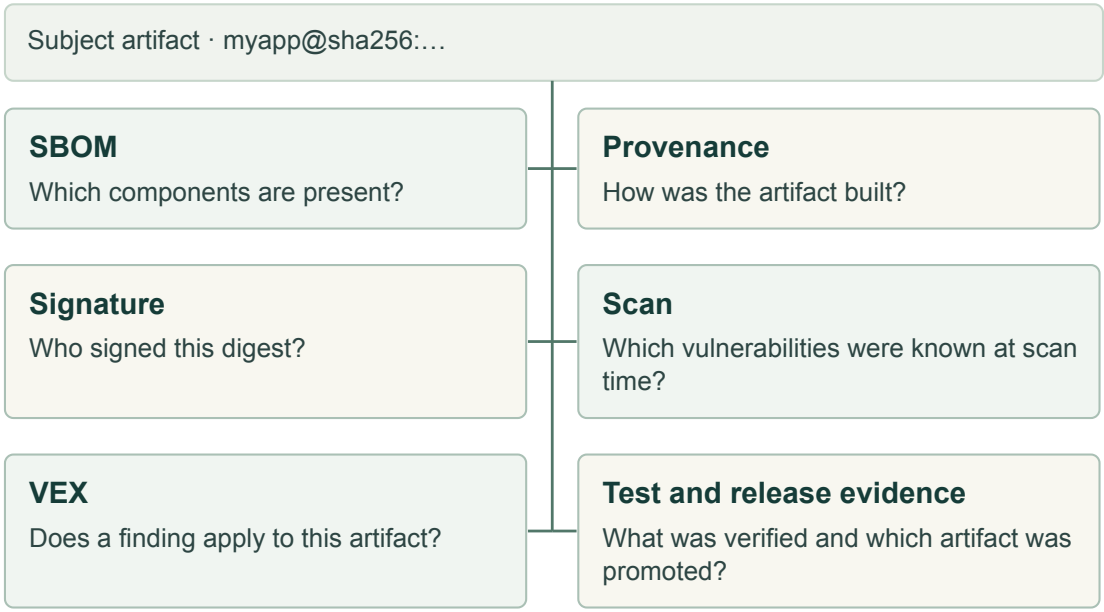


Figure 10-3 OCI referrers can organize supply-chain evidence around one immutable subject digest.

Table 10-10 Evidence types answer different questions and have different limits.

Evidence type	Question answered	What it does not prove
SBOM	Which components, packages, files, licenses, or relationships were reported?	That every component was discovered, that components are safe, or that the SBOM matches runtime state.
Build provenance	Which source, materials, builder, parameters, and process produced the subject?	That the source is benign, the builder should be trusted, or the output is vulnerability-free.
Signature	Did a trusted identity sign this exact descriptor, and is the signature valid under policy?	That the signer reviewed the artifact correctly or that the artifact is secure.
Vulnerability scan	Which known findings did a scanner report for this artifact at this time?	Future findings, exploitability, hidden code, misconfiguration, or runtime exposure.
VEX	Does the producer or analyst consider a particular vulnerability applicable, not affected, fixed, or under investigation?	That the assertion is correct without trusted issuer, justification, and review.

Evidence type	Question answered	What it does not prove
Test attestation	Which test suite ran, against which subject, with what result?	That the tests were sufficient, independent, or representative of production.
Promotion attestation	Which verified digest was approved for a target environment under which policy?	That the target actually deployed it or that runtime configuration remained conformant.

10.7.1 SBOM Formats and Lifecycle

SPDX and CycloneDX are widely used machine-readable standards for software bills of materials. SPDX is an ISO-recognized open standard with profiles that extend beyond basic package inventories. CycloneDX is a full-stack BOM standard that represents software, services, hardware, cryptography, and machine-learning artifacts. Tool and ecosystem compatibility may require more than one output format, but conversion should not be mistaken for information equivalence.

Table 10-11 SBOM design decisions.

Decision	Questions to record
Generation stage	Was the SBOM produced from source dependencies, build output, final image filesystem, or all three?
Subject	Does the SBOM refer to the image index, platform manifest, package, binary, model artifact, or release bundle?
Depth	Are transitive packages, files, operating-system components, licenses, hashes, and relationships included?
Completeness	Which ecosystems or file types could the generator not identify?
Freshness	Is the SBOM immutable composition evidence, while vulnerability data is updated separately?
Distribution	Is the SBOM embedded, attached as an OCI referrer, stored in an evidence system, or all three?
Access	Can consumers retrieve the SBOM without gaining artifact mutation privileges?
Retention	Will the SBOM and related evidence remain available for every supported release and incident window?

BuildKit can generate SBOM and provenance attestations during docker buildx build. When the output is pushed to a compatible registry, the attestations can be attached to the image or image index. The organization must still inspect which predicate and version

were produced, whether secrets or sensitive parameters could leak into provenance, and whether downstream tools discover the attachments consistently.

10.7.2 Provenance Is Useful Only When Verified

The in-toto attestation statement binds a subject digest to a predicate type and predicate. SLSA provenance uses this structure to describe the build. A verifier should compare the statement against expectations: trusted builder identity, approved source repository and revision, allowed parameters, required materials, and completeness. Merely storing provenance beside the image does not enforce any of these expectations.

Provenance also has privacy implications. Full build parameters and environment details can reveal internal paths, repository names, build arguments, or infrastructure. A publication policy should define what evidence is externally visible, what remains inside the organization, and what minimum fields a deployment gate requires.

10.8 Vulnerability Management Is a Lifecycle, Not a Gate

A scan result is a time-stamped view produced by a particular scanner, database, package-identification method, and policy. The image can remain byte-for-byte unchanged while new vulnerabilities are disclosed or severity and exploitability assessments change. Therefore, the supply-chain control plane must support rescanning, triage, remediation, rebuild, replacement, and retirement after the artifact is already in production.

Table 10-12 Vulnerability management stages.

Stage	Control	Evidence
Base selection	Use approved, maintained bases pinned by digest.	Base inventory, support policy, source, digest, age.
Build-time scan	Detect vulnerable dependencies before publication.	Scanner version, database timestamp, subject digest, findings, policy result.
Registry scan	Assess stored artifacts and rescan when vulnerability data changes.	Registry scan report, rescan timestamp, repository and digest.
Triage	Combine severity, reachability, exploitability, workload exposure, and business criticality.	Decision owner, rationale, due date, VEX or exception.
Remediation	Update base or dependency, rebuild, retest, resign, and repromote.	New digest, delta SBOM, new provenance, verification and rollout record.

Stage	Control	Evidence
Runtime inventory	Identify where affected digests are running.	Cluster or platform inventory, workload owner, environment, exposure.
Retirement	Remove vulnerable or unsupported artifacts from active channels and eventually from the registry.	Deprecation notice, replacement digest, retention decision, deletion evidence.

Severity alone is an incomplete deployment policy. A critical vulnerability in a component not present in the executed path may require a different response from a high-severity vulnerability reachable from a public endpoint. VEX can communicate an applicability assessment, but a VEX assertion is itself evidence that requires an issuer, status, justification, timestamp, and review process.

The safest remediation pattern is usually rebuild and replace, not in-place mutation of a published image. Rebuilding creates a new digest and a new evidence graph. The release system can then verify, promote, deploy, observe, and retire the old digest using the same lifecycle contract.

10.9 Signing and Trust Policies

A signature binds a signer identity to the digest of an artifact descriptor. Verification determines whether the signature is cryptographically valid and acceptable under a trust policy. The policy is essential: without an expected identity, scope, certificate chain, timestamp, and verification level, a valid signature from an unintended signer may still be rejected - or worse, accepted incorrectly.

Table 10-13 Common signing approaches for OCI artifacts.

Approach	Trust model	Operational characteristics
Notation with organization-managed certificate	X.509 certificate chain and explicit Notary Project trust policy.	Strong organizational control; requires issuance, storage, rotation, revocation, timestamping, and trust-store distribution.
Notation with managed Artifact Signing	Short-lived certificates issued through a managed signing service and verified under configured trust.	Reduces certificate lifecycle work; still requires identity policy, timestamp validation, scope, and consumer configuration.
Cosign keyless signing	OIDC identity, ephemeral certificate, transparency log, and identity constraints.	Fits CI federation and open ecosystems; depends on issuer, identity matching, transparency and availability policy.

Approach	Trust model	Operational characteristics
Cosign or Notation with protected key	Public key or certificate associated with a protected local, HSM, KMS, or Key Vault key.	Works in private environments; key custody, rotation, distribution, and revocation remain organizational duties.

Sign by digest, not by a tag. The signature subject must remain stable even if a release channel tag moves. Timestamping allows a verifier to establish that a signature was created while a signing certificate was valid, which is especially important for short-lived certificates or long-supported releases. Rotation should introduce new trusted signers before old trust is removed; revocation and compromise response should be tested rather than assumed.

Table 10-14 A trust policy should make these decisions explicit.

Policy element	Example question
Subject scope	Which registry, repository, artifact type, and digest are covered?
Trusted identity	Which X.509 subject, OIDC identity, organization, or key is authorized to sign?
Verification level	Are integrity, authenticity, expiry, revocation, and timestamp checks required?
Evidence set	Is a signature sufficient, or are provenance, SBOM, scan, and test attestations also required?
Freshness	How old may scans, approvals, and vulnerability databases be?
Environment	Does production require a stronger signer or policy than development?
Exception	Who may approve an exception, for how long, and with what compensating controls?
Failure action	Audit, warn, quarantine, deny promotion, deny admission, or terminate runtime?

10.10 Policy Gates from Source to Runtime

A software supply chain is protected by a sequence of gates, not by one security product. Source controls establish who may change code and build definitions. Build controls establish who may produce artifacts and evidence. Registry controls establish who may publish, mutate, discover, replicate, and delete. Promotion controls establish which digest may en-

ter an environment. Admission controls establish what a runtime may instantiate. Runtime controls establish what is actually running and how newly discovered risk is handled.

Supply-chain policy is enforced at multiple gates

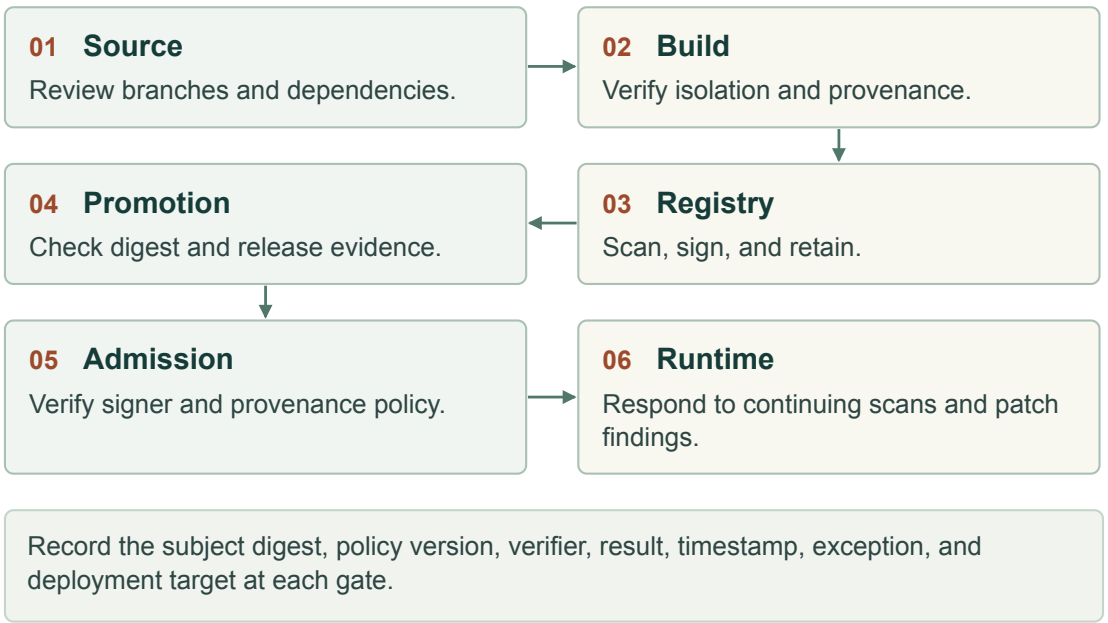


Figure 10-6 Policy gates protect different phases of the artifact lifecycle.

Table 10-15 Example supply-chain gates and failure actions.

Gate	Required assertions	Typical failure action
Source merge	Reviewed change, protected branch, approved dependencies, no exposed secret.	Reject merge or require additional review.
Build publication	Trusted builder, expected source revision, successful tests, generated SBOM and provenance.	Do not push or mark artifact as quarantined.
Registry acceptance	Allowed repository, authorized producer, scan below policy threshold, signature attached.	Reject push where supported, block pull, or prevent promotion.
Environment promotion	Same verified digest, approved evidence, owner and change record, no expired exception.	Keep digest in lower environment and record decision.

Gate	Required assertions	Typical failure action
Kubernetes or platform admission	Digest pin, trusted signature, expected provenance, allowed registry, workload policy.	Deny workload creation or audit in staged rollout.
Runtime response	Running digest inventory, current vulnerability and exposure state, supported release.	Alert, isolate, scale down, replace, or execute incident procedure.

A policy engine should record the subject digest, policy version, verifier identity and software version, decision, timestamp, evidence references, target environment, and exception. This record makes a denial explainable and an approval auditable. It also supports progressive enforcement: observe first, remediate false positives and missing metadata, then move selected rules from audit to deny.

Kubernetes recommends avoiding mutable tags such as latest and using complete digests for stable runtime identity. Signature and attestation verification can be performed by an admission controller before a Pod is accepted. Azure provides a Ratify and Azure Policy pattern for verifying Notary Project signatures on AKS; other environments can use Sigstore policy-controller, Gatekeeper integrations, Kyverno, or provider-native controls. The implementation is less important than the contract: the runtime must verify the exact subject against a current trust policy.

10.11 AI-Native Supply Chains Extend Beyond the Image

An AI service can be packaged in a container and still depend on separately distributed model weights, tokenizer files, adapters, quantization parameters, prompts, retrieval indexes, safety configurations, evaluation sets, and tool definitions. If only the runtime image is versioned and signed, the deployed behavior can change without a corresponding software release.

AI-native delivery expands the software supply chain

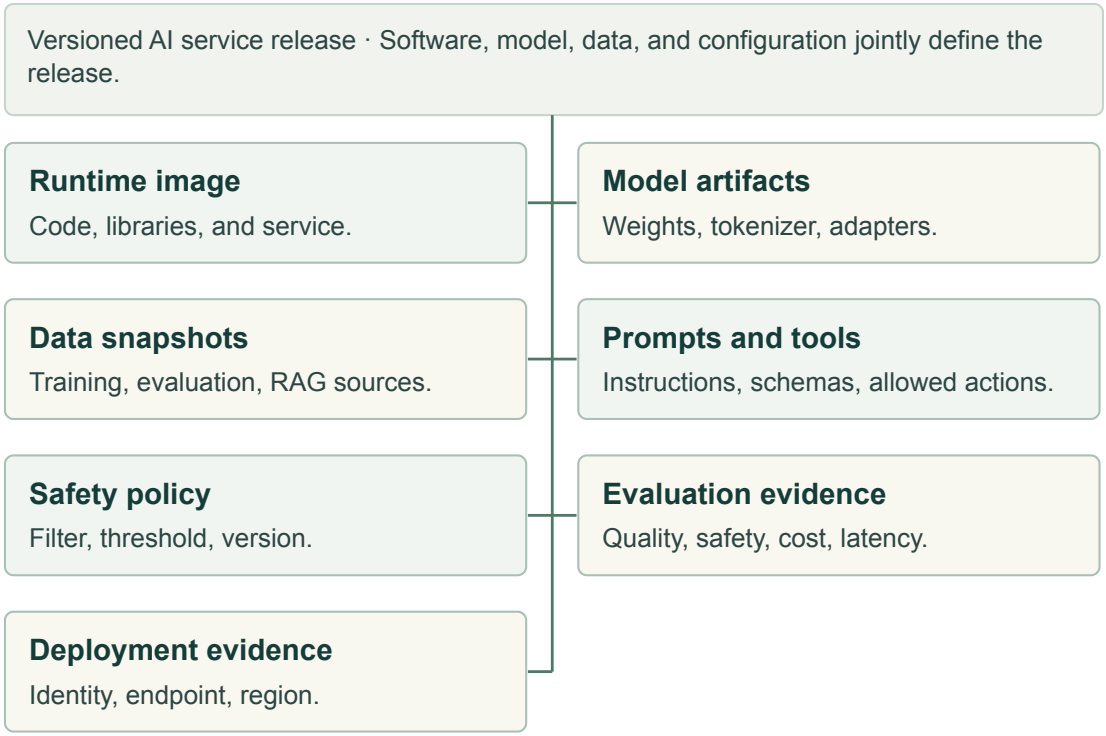


Figure 10-7 AI-native releases combine software, model, data, configuration, safety, evaluation, and deployment evidence.

Table 10-16 AI-native artifacts and the claims they need.

Artifact	Identity and evidence requirements
Runtime image	Digest, SBOM, build provenance, signature, scan, supported platform and runtime policy.
Model weights	Model identifier, exact hash or OCI digest, source, license, architecture, provenance, signature, evaluation scope.
Tokenizer / processor	Version and hash coupled to the model; compatibility and source evidence.
Adapter / fine-tune	Base model subject, training data snapshot, method, hyperparameters, producer, evaluation and approval.
Prompt / orchestration config	Version, owner, model compatibility, tool schema, safety policy, evaluation result, rollout record.

Artifact	Identity and evidence requirements
Dataset or RAG corpus snapshot	Source, license, classification, lineage, selection rules, hash, retention and deletion obligations.
Safety policy	Policy version, thresholds, provider settings, evaluation coverage, approver and effective environment.
Evaluation set and result	Dataset version, evaluator version, metrics, thresholds, model and configuration subjects, cost and latency.
Deployment bundle	Exact combination of image, model, knowledge, prompt, tools, identity, region and endpoint.

OCI registries can store arbitrary compliant artifacts and reference relationships, which makes them a possible distribution substrate for models and related evidence. Tencent TCR now documents OCI-based AI model and skill hosting, and other registries support generic OCI artifacts even when they do not provide a model-specific user experience. The architecture should not assume that every AI artifact belongs in the same registry; it should require stable identity, evidence, access control, lifecycle, and deploy-time binding wherever the artifact is stored.

NIST SP 800-218A extends secure software-development practices to AI model development. That extension reinforces the main point: model development and release add new supply-chain subjects and risks, but they do not remove the need for secure source, build, provenance, verification, controlled distribution, and incident response.

10.12 Azure Reference Control Plane

Azure Container Registry provides a useful reference because the registry, repository permissions, tasks, private connectivity, replication, artifacts, signatures, scanning integrations, and admission verification remain visible as distinct resources and operations. The reference is not a claim that all capabilities must be used in every environment; it demonstrates how the release path can be decomposed into governable contracts.

Table 10-17 Azure supply-chain control-plane resources.

Capability	Azure resources and tools	Control-plane purpose
Registry service	Azure Container Registry instance, SKU, endpoints, optional regional replicas.	Define ownership, region, service capacity, availability, encryption, cost and lifecycle.

Capability	Azure resources and tools	Control-plane purpose
Repository permissions	Microsoft Entra RBAC; ABAC-enabled Repository Reader, Writer, Contributor and Catalog Lister roles.	Separate registry management from repository pull, push, metadata, delete and catalog access.
Build	Docker BuildKit/buildx, ACR Tasks, Azure Pipelines, GitHub Actions.	Produce digest, SBOM, provenance, logs and repeatable pipeline evidence under workload identity.
OCI evidence graph	ACR OCI artifact support, ORAS, OCI Referrers API, az acr manifest list-referrers.	Attach and discover signatures, SBOMs, scan results and other evidence by subject digest.
Signing	Notation with Key Vault certificate, or Notation with Artifact Signing.	Bind trusted publisher identity to artifact digest and manage certificate or managed signing lifecycle.
Scanning	Microsoft Defender for Containers and third-party or pipeline scanners.	Discover known vulnerabilities and maintain registry/runtime risk evidence.
Admission	Ratify with Azure Policy on AKS; managed image-integrity options where available.	Verify signatures and trust policy before admitting workloads.
Network	Private Link, firewall and trusted-service settings, private DNS, regional endpoints.	Control registry reachability independently of repository authorization.
Retention and recovery	Retention, soft delete, locks, repository attributes and purge tasks.	Control obsolete artifacts while preserving rollback subjects and evidence.
Audit and observability	Activity Log, diagnostic logs, task runs, Defender findings and policy decisions.	Correlate identity, operation, digest, policy, result, target and cost.

Azure CLI manifest commands, some artifact streaming capabilities, and selected integrity features may be preview or edition-dependent. Verify current command status, region, SKU, and registry encryption constraints before production use.

For new registries, Azure recommends the RBAC Registry + ABAC Repository Permissions mode. In this mode, legacy AcrPull, AcrPush, and AcrDelete roles are not honored for repository access; use the ABAC-enabled repository roles. A role without an ABAC condition grants registry-wide repository access, while a condition can constrain the repository name. Catalog listing is a separate permission. These details matter when a pipeline or cluster appears to have management access but still receives a data-plane authorization failure.

ACR can store a graph of supply-chain artifacts through OCI references. ORAS can attach and discover arbitrary evidence. Notation creates signatures that refer to image subjects. A consumer configures a trust store and trust policy and then verifies the exact digest. At AKS admission, Ratify and Azure Policy can deny workloads whose signatures do not satisfy policy. This is a full producer-to-consumer trust path rather than a signature file created for compliance display.

10.13 Multicloud Translation

The cloud providers expose the same engineering intents through different editions, resource hierarchies, identity systems, scanners, signing models, and Kubernetes integrations. The translation should preserve these differences. A free personal registry may be appropriate for a lab but lacks the isolation, SLA, network controls, signing, scanning, and replication expected in production.

Table 10-18 Multicloud registry and supply-chain capability mapping.

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Managed registry	Azure Container Registry (ACR)	Container Registry (ACR), especially Enterprise Edition for production	Tencent Container Registry (TCR) Enterprise Edition	Container Registry (CR)
OCI artifact support	Images and arbitrary OCI artifacts; referrer graph for signatures, SBOMs and scan results	Enterprise Edition supports OCI artifacts and OCI 1.1 reference types / Referrers API on supported instances	Docker/OCI artifacts, Helm charts; current documentation also describes OCI AI model and skill hosting	OCI-compliant images and Helm charts; generic artifact capabilities should be verified by edition and tool
Identity and scope	Entra RBAC and ABAC repository roles; managed identities and service principals	RAM, instance/namespace/repository authorization, credential-free pull in supported enterprise scenarios	CAM, instance/namespace/repository policies, service accounts and TKE integration	IAM authorization, registry credentials and VKE credential controller integration
Private access	Private Link, firewall, trusted services and private DNS	VPC access controls, allowlists and enterprise network isolation	VPC private access controls and TKE private password-free pull integration	VPC access quota, endpoint controls and private pull through VKE integration
Scanning	Defender for Containers and pipeline/third-party scanners	Multi-engine artifact scanning, remediation guidance and risk blocking by edition	Enterprise image scanning, automatic scan and deployment blocking by edition	Multi-dimensional image scanning and build-time or platform security integration

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Signing / verification	Notation with Key Vault or Artifact Signing; Ratify/Azure Policy at AKS	Enterprise signing and verification, including Cosign-compatible workflows and delivery-chain controls	Enterprise image signing and verification with KMS; TKE integration and edition constraints	No exact managed signing equivalence assumed; use OCI-compatible external signing where supported and verify current product capability
Replication / distribution	Geo-replication, import, cache and connected-registry options	Global synchronization, P2P and on-demand distribution by edition	Cross-region and cross-account synchronization, P2P and lazy loading by edition	Cross-region distribution, proxy cache and acceleration capabilities
Lifecycle and evidence	Retention, soft delete, locks, task runs, Activity Log and diagnostics	Cleanup, immutable tags, ActionTrail and EventBridge notifications for build/scan/sign status	Retention rules, audit logs, triggers and synchronization history	Lifecycle management, audit and monitoring capabilities; verify current evidence granularity

Product editions and regional availability change. Treat the mapping as an engineering-intent translation, then verify the current official documentation for signing model, OCI reference support, scan engine, Kubernetes admission integration, network path, quotas, SLA and billing.

10.14 Lab 10: Build, Push, Scan, Sign, and Verify a Container Image

This lab validates one capability: a producer can publish an OCI image by digest, attach build evidence, produce a vulnerability report, sign the immutable subject with a protected key, verify the signature under an explicit trust policy, and collect a reusable evidence packet. It does not deploy the image to Kubernetes; admission verification belongs to Chapter 11.

10.14.1 Goal, Architecture, and Cost Boundary

Table 10-19 Lab 10 contract.

Item	Decision
Capability goal	Create a minimal non-root image, push it to ACR, identify it by digest, capture SBOM and provenance, scan it, sign it with Notation and Key Vault, then verify it under a strict trust policy.

Item	Decision
Primary resources	Resource group, ABAC-enabled Standard ACR, RBAC-enabled Key Vault, code-signing certificate, repository, OCI image and referrer artifacts.
Required tools	Azure CLI, Docker with Buildx/BuildKit, .NET 8 SDK, jq, Trivy or an approved scanner, Notation, notation-azure-kv plug-in; ORAS is useful for independent referrer inspection.
Evidence	Registry configuration, role assignments, source commit or source hash, image digest, manifest metadata, SBOM/provenance referrers, scan report, signature listing, trust policy, successful verification and cleanup proof.
Cost boundary	ACR and Key Vault incur charges. Use one small registry and one certificate, finish promptly, delete the resource group, and remove local trust files. Defender is not required for this lab.
Not demonstrated	Production CA governance, transparency service, multi-region promotion, admission denial, key-compromise exercise, or continuous rescan.

Use a non-production subscription or sandbox. The commands assume Bash. Tool versions change; install the current supported versions from official sources and record them in evidence/tool-versions.txt. The signing certificate is self-signed for the lab and must not be treated as a production public-key infrastructure pattern.

```
set -euo pipefail
export SUFFIX=$(openssl rand -hex 3)
export LOCATION=eastus
export RG=book-ch10-supply-$SUFFIX
export ACR=booksupply$SUFFIX
export KV=booksign$SUFFIX
export REPO=managed-book-api
export TAG=1.0.0
export CERT_NAME=book-lab-signer
export CERT_SUBJECT="CN=book-lab-signer,O=Cloud Services
Lab,L=Seattle,ST=WA,C=US"
mkdir -p evidence

az login
az account show --output table
export SUB_ID=$(az account show --query id -o tsv)
export USER_ID=$(az ad signed-in-user show --query id -o tsv)
az group create --name "$RG" --location "$LOCATION" --tags   owner=book-lab
workload=supply-chain lifecycle=temporary
```

10.14.2 Create the Registry and Signing Boundary

Create ACR in the ABAC-enabled permissions mode and disable the admin account. Create Key Vault with Azure RBAC authorization. Assign repository write access and only the certificate and cryptographic roles needed for the lab. The example assigns the repository role at registry scope for command clarity; a production workflow should add an ABAC condition for the specific repository.

```
az acr create   --resource-group "$RG"   --name "$ACR"   --location "$LOCATION"
--sku Standard   --admin-enabled false   --role-assignment-mode rbac-abac

export ACR_ID=$(az acr show -g "$RG" -n "$ACR" --query id -o tsv)
export REGISTRY=$(az acr show -g "$RG" -n "$ACR" --query loginServer -o tsv)

az keyvault create   --resource-group "$RG"   --name "$KV"   --location
"$LOCATION"   --enable-rbac-authorization true
export KV_ID=$(az keyvault show -g "$RG" -n "$KV" --query id -o tsv)

az role assignment create   --assignee-object-id "$USER_ID"   --assignee-
principal-type User   --role "Container Registry Repository Writer"   --scope
"$ACR_ID"

az role assignment create   --assignee-object-id "$USER_ID"   --assignee-
principal-type User   --role "Key Vault Certificates Officer"   --scope "$KV_ID"
az role assignment create   --assignee-object-id "$USER_ID"   --assignee-
principal-type User   --role "Key Vault Crypto User"   --scope "$KV_ID"

az acr show -g "$RG" -n "$ACR" > evidence/acr-resource.json
az role assignment list --scope "$ACR_ID" > evidence/acr-role-assignments.json
az role assignment list --scope "$KV_ID" > evidence/keyvault-role-
assignments.json
```

Azure role assignments can take several minutes to propagate. If authentication succeeds but push or signing returns 403, wait and retry. Refresh ACR credentials after any change to the role-assignment permissions mode.

10.14.3 Create a Minimal Non-Root API

```
mkdir -p src && cd src
dotnet new web -n ManagedBookApi --framework net8.0
cd ManagedBookApi
cat > Program.cs <<'CS'
var app = WebApplication.Create(args);
app.MapGet("/", () => Results.Ok(new { service = "managed-book-api", version =
"1.0.0" }));
app.MapGet("/health", () => Results.Ok(new { status = "healthy" }));
app.Run();
```

```
CS

dotnet restore --use-lock-file
cat > Dockerfile <<'DOCKER'
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY ManagedBookApi.csproj packages.lock.json ./
RUN dotnet restore --locked-mode
COPY . .
RUN dotnet publish -c Release -o /out /p:UseAppHost=false

FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS runtime
WORKDIR /app
ENV ASPNETCORE_URLS=http://+:8080
COPY --from=build /out .
USER $APP_UID
EXPOSE 8080
ENTRYPOINT ["dotnet", "ManagedBookApi.dll"]
DOCKER

cat > .dockerignore <<'EOF'
bin/
obj/
evidence/
.git/
EOF
cd ../..
```

For a real release, pin both base images by approved digest and record the update procedure. The lab leaves the Microsoft base tags readable so the workflow is easier to reproduce; the evidence packet should explicitly record this limitation.

10.14.4 Build Once, Push, and Capture the Digest

Authenticate with the signed-in Entra identity, create a Buildx builder if required, and push the image while requesting SBOM and full provenance attestations. Attestations are most useful when the output is pushed to a registry rather than loaded only into a local image store.

```
az acr login --name "$ACR"
docker buildx create --name book-supply-builder --use 2>/dev/null ||   docker
buildx use book-supply-builder
docker buildx inspect --bootstrap

export IMAGE_TAGGED="$REGISTRY/$REPO:$TAG"
docker buildx build   --platform linux/amd64   --sbom=true   --
```

```
provenance=mode=max   --tag "$IMAGE_TAGGED"   --push   ./src/ManagedBookApi

export DIGEST=$(az acr manifest show-metadata   --registry "$ACR"   --name
"$REPO:$TAG"   --query digest -o tsv)
export IMAGE="$REGISTRY/$REPO@$DIGEST"
printf '%s
' "$IMAGE" | tee evidence/image-subject.txt
az acr manifest show-metadata -r "$ACR" -n "$REPO@$DIGEST"   >
evidence/manifest-metadata.json
az acr manifest list-referrers -r "$ACR" -n "$REPO@$DIGEST"   >
evidence/referrers-before-signing.json
```

The az acr manifest command group is currently marked preview. If its syntax changes, use oras discover --format json "$IMAGE" or the registry OCI Referrers API and record the tool version in the evidence packet.

Inspect the referrers output. Depending on Buildx, registry, and multi-platform manifest behavior, SBOM and provenance may refer to the image index or a platform manifest. Record the actual subject and do not assume that one evidence object automatically covers every platform variant.

10.14.5 Scan the Immutable Subject

Use an approved scanner against the digest, not the mutable tag. The following example uses Trivy because it can run independently of a paid cloud scanning plan. Docker Scout or another enterprise scanner can be substituted, but the evidence packet must record the scanner, version, database update time, subject digest, and policy threshold.

```
trivy --version | tee evidence/trivy-version.txt
trivy image   --scanners vuln,secret   --format json   --output
evidence/scan.json   "$IMAGE"

# Produce a human-readable summary without changing the subject.
trivy image --severity HIGH,CRITICAL "$IMAGE"   | tee evidence/scan-summary.txt
```

A finding does not automatically imply exploitability, and a clean report does not establish absence of vulnerabilities. Record the result as a time-bound scan claim. If the policy fails, rebuild from updated materials and produce a new digest rather than editing the published image.

10.14.6 Create a Lab Signing Certificate

```
cat > evidence/certificate-policy.json <<EOF
{
  "issuerParameters": { "name": "Self" },
```

```
  "keyProperties": {
    "exportable": false,
    "keySize": 2048,
    "keyType": "RSA",
    "reuseKey": false
  },
  "secretProperties": { "contentType": "application/x-pem-file" },
  "x509CertificateProperties": {
    "ekus": [ "1.3.6.1.5.5.7.3.3" ],
    "keyUsage": [ "digitalSignature" ],
    "subject": "$CERT_SUBJECT",
    "validityInMonths": 3
  }
}
EOF

az keyvault certificate create   --vault-name "$KV"   --name "$CERT_NAME"   --
policy @evidence/certificate-policy.json   > evidence/certificate-create.json

export KEY_ID=$(az keyvault certificate show   --vault-name "$KV"   --name
"$CERT_NAME"   --query kid -o tsv)
printf '%s
' "$KEY_ID" > evidence/signing-key-id.txt
```

The private key remains non-exportable in Key Vault. The Notation plug-in requests a signing operation through the current Azure identity. In production, the certificate subject, issuer, validity, timestamp authority, role assignment, and separation of duties must be governed independently of the image pipeline.

10.14.7 Sign by Digest and Inspect the Referrer Graph

```
notation plugin ls | tee evidence/notation-plugins.txt
notation version | tee evidence/notation-version.txt

notation sign   --signature-format cose   --id "$KEY_ID"   --plugin azure-kv   -
-plugin-config self_signed=true   --plugin-config credential_type=azurecli
"$IMAGE"

notation ls "$IMAGE" | tee evidence/notation-list.txt
az acr manifest list-referrers -r "$ACR" -n "$REPO@$DIGEST"   >
evidence/referrers-after-signing.json
```

The signature is a referrer to the immutable subject. Listing the graph should now show the build attestations and the Notary Project signature, although exact media types and attachment structure depend on tool versions. If the registry or encryption configuration re-

quires a referrers tag scheme rather than the Referrers API, record that implementation detail; the trust claim still needs to bind to the digest.

10.14.8 Create a Trust Store and Verify

```
export CERT_PATH=./evidence/${CERT_NAME}.pem
az keyvault certificate download   --vault-name "$KV"   --name "$CERT_NAME"   --
file "$CERT_PATH"

export STORE_TYPE=ca
export STORE_NAME=book-lab
notation cert add --type "$STORE_TYPE" --store "$STORE_NAME" "$CERT_PATH"

cat > evidence/trustpolicy.json <<EOF
{
  "version": "1.0",
  "trustPolicies": [
    {
      "name": "book-lab-images",
      "registryScopes": [ "$REGISTRY/$REPO" ],
      "signatureVerification": { "level": "strict" },
      "trustStores": [ "$STORE_TYPE:$STORE_NAME" ],
      "trustedIdentities": [ "x509.subject: $CERT_SUBJECT" ]
    }
  ]
}
EOF

notation policy import evidence/trustpolicy.json
notation policy show | tee evidence/trust-policy-active.txt
notation verify "$IMAGE" | tee evidence/verification.txt
```

Verification should return the verified digest. Save the trust policy and verification output together. A signature that verifies only because a local client trusts every identity is not meaningful evidence; the policy must identify the registry scope and expected signer.

10.14.9 Assemble the Evidence Packet

Table 10-20 Required evidence packet for Lab 10.

File or record	Question answered
tool-versions.txt	Which CLI, builder, scanner, signing and registry tools produced or inspected the evidence?
acr-resource.json	Which registry resource, region, SKU and role-assignment mode hosted the artifact?

File or record	Question answered
role-assignment files	Which principal could write the repository and invoke the signing key?
image-subject.txt / manifest-metadata.json	What exact digest is the release subject?
referrers-before-signing.json	Which SBOM and provenance attachments existed after build?
scan.json / scan-summary.txt	What did the selected scanner report and when?
certificate-create.json / signing-key-id.txt	Which protected signing identity and certificate version were used?
notation-list.txt / referrers-after-signing.json	Which signature and other derivative artifacts refer to the subject?
trustpolicy.json / verification.txt	Which signer and scope were trusted, and did the exact digest verify?
cleanup.txt	Were cloud and local signing resources retired after the lab?

```
{
  az --version
  docker version --format '{{.Client.Version}}'
  docker buildx version
  jq --version
  trivy --version | head -1
  notation version
  notation plugin ls
} > evidence/tool-versions.txt

sha256sum evidence/* > evidence/evidence-packet.sha256
ls -lh evidence
```

Optionally perform a negative authorization test from a separate principal that has registry management Reader but no repository data-plane role. The expected result is that the principal can inspect the registry resource but cannot pull the private image. Keep this test optional because identity creation and credential handling differ across sandbox environments.

10.14.10 Clean Up and Prove Retirement

```
# Remove local Notation policy and trust material according to your workstation
policy.
notation policy show > evidence/trust-policy-before-cleanup.txt || true

az group delete --name "$RG" --yes --no-wait
az group wait --deleted --name "$RG"

if az group exists --name "$RG" | grep -q true; then
  echo "cleanup failed: resource group still exists" | tee evidence/cleanup.txt
  exit 1
else
  echo "cloud cleanup verified: resource group deleted" | tee
evidence/cleanup.txt
fi

docker buildx rm book-supply-builder 2>/dev/null || true
# Delete the lab trust store and imported policy using the supported Notation
workflow
# for the installed version, then securely remove local certificate and evidence
copies
# when they are no longer needed.
```

Key Vault uses soft-delete and may retain a deleted vault name during its recovery period. In a subscription where purge protection or organizational retention applies, the lab must not attempt to bypass that policy. Record the retained-deleted state as part of cleanup evidence.

10.14.11 Troubleshooting and Interpretation

Table 10-21 Lab failure modes and diagnostic questions.

Symptom	Likely cause or check
az acr login succeeds but push returns 403	Confirm ABAC-enabled mode, Repository Writer role, role propagation and refreshed credentials. Contributor alone is control-plane access.
Build pushed but no SBOM or provenance referrer appears	Confirm Buildx driver, --push, attestation flags, subject scope, registry compatibility and CLI preview behavior. Inspect with ORAS.
Scanner cannot pull the digest	Confirm registry authentication and repository Reader/Writer data-plane access; use the same login server returned by ACR.
Key Vault certificate creation is denied	Confirm Certificates Officer role, RBAC mode, subscription and propagation.
Notation signing is denied	Confirm Crypto User role, certificate key ID, plug-in installation and credential type.

Symptom	Likely cause or check
Notation verifies a tag but digest differs from the release record	Stop and verify the explicit digest. A tag may have moved.
Signature exists but verification rejects identity	Compare the certificate subject, registry scope, trust-store type, trust policy and certificate chain.
Cleanup cannot reuse the Key Vault name	Soft-delete retention may preserve the name; use a new suffix or follow authorized purge policy.

The successful result is not "the image is secure." The result is narrower and auditable: a specific digest was built from recorded inputs, published to a governed registry, accompanied by SBOM and provenance evidence, scanned by a named tool at a recorded time, signed by an expected identity through a protected key, and verified under a strict policy. Production assurance requires additional source, build-platform, admission, runtime, incident, and organizational controls.

10.15 CloudDesk Reuse

CloudDesk FinOps Agent should not receive a generic shell tool capable of running arbitrary Docker, Azure CLI, or registry commands. The chapter produces bounded read-oriented capabilities that can later be wrapped by the Tool Broker and, where a mutation is needed, expanded into plan, approve, execute, and verify stages.

Table 10-22 CloudDesk capability handoff.

Capability	Input	Output and guardrail
container.image.scan	Provider, registry, repository, digest, scanner policy and freshness requirement.	Normalized findings, scanner metadata, evidence URI, staleness, policy result; read-only and digest-bound.
container.signature.verify	Registry, repository, digest, trust-policy identifier and expected signer scope.	Verification decision, signer identity, signature and timestamp evidence, policy version; no implicit trust-on-first-use.
container.evidence.inventory	Registry, repository and subject digest.	SBOM, provenance, signatures, scan, VEX and test referrers with media types and timestamps; read-only.
container.registry.cleanup.recommend	Registry inventory, retention policy, protected releases and rollback window.	Dry-run deletion plan, estimated storage effect, blocked subjects and evidence dependencies; never deletes without approval.

Capability	Input	Output and guardrail
container.release.assess	Subject digest, target environment and required policy bundle.	Missing or stale evidence, identity and network gaps, promotion readiness and remediation recommendations.

The FinOps use case is broader than registry storage cost. An agent can identify oversized images that slow scale-out, unnecessary cross-region pulls, stale debug artifacts, duplicated architecture variants, expired evidence, or high-cost scanning patterns. It must not delete an untagged manifest merely because it appears unused: the digest may be referenced by a deployment, rollback record, signature, SBOM, or model release. Evidence-aware cleanup is a graph problem.

10.16 Key Takeaways

- A container image is a content-addressed artifact graph; a container is a runtime instance created from it.
- Tags are discovery and channel references. Digests are the immutable subjects for signing, scanning, promotion, rollback, and runtime pinning.
- The build is a governed contract over source, materials, parameters, builder identity, isolation, secrets, platform and output claims.
- A registry has a cloud control plane and a repository data plane. Management permission does not imply pull or push authority.
- OCI reference types let SBOMs, provenance, signatures, scan results, VEX and test attestations form an evidence graph around one digest.
- No single evidence type proves security. Every claim has a scope, issuer, timestamp, verification method and limitation.
- Vulnerability management continues after release through rescan, triage, rebuild, replacement, runtime inventory and retirement.
- Signing is useful only with explicit trust policy, signer identity, digest scope, timestamp and consumer-side verification.
- Policy gates belong at source, build, registry, promotion, admission and runtime; one post-build scan cannot protect the full chain.
- AI-native releases add model, data, prompt, tool, safety and evaluation artifacts that need the same identity, evidence and lifecycle discipline.
- The preferred release pattern is build once, verify once, promote the same digest, and continuously reassess risk.

References

1. Open Container Initiative. OCI Image Specification, Version 1.1. https://specs.opencontainers.org/image-spec/

2. Open Container Initiative. OCI Distribution Specification. https://specs.opencontainers.org/distribution-spec/

3. Open Container Initiative. OCI Image and Distribution Specs v1.1 Releases. https://opencontainers.org/posts/blog/2024-03-13-image-and-distribution-1-1/

4. Open Container Initiative. About OCI. https://opencontainers.org/about/overview/

5. SLSA. SLSA Specification v1.2. https://slsa.dev/spec/v1.2/

6. SLSA. Build Provenance. https://slsa.dev/spec/v1.2/build-provenance

7. SLSA. Build Track Basics. https://slsa.dev/spec/v1.2/build-track-basics

8. in-toto. Attestation Framework v1.0 and Statement v1. https://in-toto.io/docs/specs/ and https://in-toto.io/Statement/v1

9. SPDX. SPDX Specifications. https://spdx.dev/use/specifications/

10. SPDX. SPDX Overview. https://spdx.dev/learn/overview/

11. OWASP CycloneDX. Specification Overview. https://cyclonedx.org/specification/overview/

12. OWASP CycloneDX. Machine Learning Bill of Materials. https://cyclonedx.org/capabilities/mlbom/

13. CISA. 2025 Minimum Elements for a Software Bill of Materials. https://www.cisa.gov/resources-tools/resources/2025-minimum-elements-software-bill-materials-sbom

14. CISA. Minimum Requirements for Vulnerability Exploitability eXchange (VEX). https://www.cisa.gov/resources-tools/resources/minimum-requirements-vulnerability-exploitability-exchange-vex

15. NIST. Secure Software Development Framework (SSDF) Version 1.1, SP 800-218. https://csrc.nist.gov/pubs/sp/800/218/final

16. NIST. Secure Software Development Practices for Generative AI and Dual-Use Foundation Models, SP 800-218A. https://csrc.nist.gov/pubs/sp/800/218/a/final

17. Docker. Build attestations. https://docs.docker.com/build/metadata/attestations/

18. Docker. SBOM attestations. https://docs.docker.com/build/metadata/attestations/sbom/

19. Docker. Provenance attestations. https://docs.docker.com/build/metadata/attestations/slsa-provenance/

20. Docker. docker buildx build reference. https://docs.docker.com/reference/cli/docker/buildx/build/

21. Docker. Docker Scout SBOMs. https://docs.docker.com/scout/how-tos/view-create-sboms/

22. ORAS. Attached Artifacts and Referrers. https://oras.land/docs/concepts/reftypes

23. ORAS. oras attach. https://oras.land/docs/commands/oras_attach/

24. ORAS. oras discover. https://oras.land/docs/commands/oras_discover/

25. ORAS. oras copy and referrer promotion. https://oras.land/docs/1.2/commands/oras_cp

26. Notary Project. Notation documentation. https://notaryproject.dev/docs/

27. Sigstore. Cosign signing overview. https://docs.sigstore.dev/cosign/signing/overview/

28. Sigstore. Cosign signature verification. https://docs.sigstore.dev/cosign/verifying/verify/

29. Sigstore. in-toto attestations with Cosign. https://docs.sigstore.dev/cosign/verifying/attestation/

30. Kubernetes. Images. https://kubernetes.io/docs/concepts/containers/images/

31. Kubernetes. Security Checklist. https://kubernetes.io/docs/concepts/security/security-checklist/

32. Kubernetes. Verify Signed Kubernetes Artifacts. https://kubernetes.io/docs/tasks/administer-cluster/verify-signed-artifacts/

33. Kubernetes. Verifying Container Image Signatures Within CRI Runtimes. https://kubernetes.io/blog/2023/06/29/container-image-signature-verification/

34. Microsoft Learn. Azure Container Registry concepts. https://learn.microsoft.com/azure/container-registry/container-registry-concepts

35. Microsoft Learn. Manage OCI and supply-chain artifacts with ORAS in ACR. https://learn.microsoft.com/azure/container-registry/container-registry-manage-artifact

36. Microsoft Learn. ACR Entra permissions and built-in roles. https://learn.microsoft.com/azure/container-registry/container-registry-rbac-built-in-roles-overview

37. Microsoft Learn. ACR ABAC repository permissions. https://learn.microsoft.com/azure/container-registry/container-registry-rbac-abac-repository-permissions

38. Microsoft Learn. Overview of signing and verifying OCI artifacts. https://learn.microsoft.com/azure/container-registry/overview-sign-verify-artifacts

39. Microsoft Learn. Sign container images with Notation and Azure Key Vault. https://learn.microsoft.com/azure/container-registry/container-registry-tutorial-sign-build-push

40. Microsoft Learn. Sign artifacts with Notation and Artifact Signing. https://learn.microsoft.com/azure/container-registry/container-registry-tutorial-sign-verify-notation-artifact-signing

41. Microsoft Learn. Verify image signatures with Ratify and Azure Policy on AKS. https://learn.microsoft.com/azure/container-registry/container-registry-tutorial-verify-with-ratify-aks

42. Microsoft Learn. Azure Container Registry SKU features and limits. https://learn.microsoft.com/azure/container-registry/container-registry-skus

43. Microsoft Learn. Import container images to ACR. https://learn.microsoft.com/azure/container-registry/container-registry-import-images

44. Microsoft Learn. Azure Container Registry authentication. https://learn.microsoft.com/azure/container-registry/container-registry-authentication

45. Microsoft Learn. Microsoft Defender for Containers overview. https://learn.microsoft.com/azure/defender-for-cloud/defender-for-containers-introduction

46. Alibaba Cloud. What is Container Registry (ACR). https://www.alibabacloud.com/help/en/acr/product-overview/what-is-container-registry

47. Alibaba Cloud. OCI Image and Distribution Specification v1.1.0 in ACR. https://www.alibabacloud.com/help/en/acr/use-cases/operating-with-oci-image-and-distribution-specification-v1-1-0

48. Alibaba Cloud. Container Registry Enterprise Edition billing and features. https://www.alibabacloud.com/help/en/acr/product-overview/billing-of-container-registry-enterprise-edition-instances

49. Alibaba Cloud. Container Registry event notifications. https://www.alibabacloud.com/help/en/acr/user-guide/create-an-event-notification

50. Tencent Cloud. Tencent Container Registry product overview. https://cloud.tencent.com/document/product/1141/39278

51. Tencent Cloud. Container image security scanning. https://cloud.tencent.com/document/product/1141/48185

52. Tencent Cloud. Container image signing and verification. https://cloud.tencent.com/document/product/1141/80862

53. Tencent Cloud. TCR private network access control. https://cloud.tencent.com/document/product/1141/41838

54. Tencent Cloud. TCR cross-instance and cross-account synchronization. https://cloud.tencent.com/document/product/1141/41945

55. Volcengine. Container Registry product advantages. https://www.volcengine.com/docs/6420/78485

56. Volcengine. Container Registry application scenarios and lifecycle management. https://www.volcengine.com/docs/6420/79130

57. Volcengine. VKE cr-credential-controller for password-free private image pull. https://www.volcengine.com/docs/6460/446504

58. FinOps Foundation. FinOps Framework. https://www.finops.org/framework/

59. Cloud Native Computing Foundation. Software Supply Chain Best Practices. https://tag-security.cncf.io/community/assessments/

60. IETF. CBOR Object Signing and Encryption (COSE), RFC 9052. https://www.rfc-editor.org/rfc/rfc9052

Part III / Chapter 11

Managed Kubernetes and Application Definition

How Managed Control Planes, Desired State, Workload Objects, Service Networking, Configuration, and Persistent State Form a Portable Application Contract

Kubernetes makes application architecture executable. A managed Kubernetes service adds a cloud control plane around that application control plane, connecting declarative objects to node pools, networks, load balancers, disks, identities, policy, telemetry, upgrades, and cost. The result is not "containers on virtual machines," but a layered resource system in which application intent is continuously reconciled against both cluster state and provider-managed infrastructure.

In This Chapter

- Treat Kubernetes as an application control plane nested inside the cloud provider control plane.
- Explain how a managed service changes the responsibility boundary without removing workload, security, reliability, upgrade, or cost decisions.
- Use metadata, spec, status, controllers, and reconciliation to reason about desired and observed application state.
- Model application definition through the Workload–Service–Volume triad, with configuration, identity, policy, and evidence as cross-cutting contracts.
- Select Deployment, StatefulSet, DaemonSet, Job, or CronJob according to lifecycle semantics rather than image type.
- Define a Pod template as a runtime contract over images, commands, health probes, resources, security context, identity, storage, and termination.

- Separate stable Service discovery from changing Pod addresses, and compare Service, Ingress, and Gateway API responsibilities.
- Use ConfigMap, Secret, ServiceAccount, RBAC, Pod Security, and NetworkPolicy without confusing configuration storage with a complete security boundary.
- Understand PVC, PV, StorageClass, CSI, reclaim policy, access mode, and cloud volume lifecycle as one cross-control-plane storage contract.
- Compose scheduling, availability, disruption, elasticity, node-pool, and failure-domain policies instead of assuming replicas alone create reliability.
- Complete an independent AKS lab that deploys a workload with configuration, secret reference, Service, Gateway API route, and persistent storage; proves state survives Pod replacement; and records both Kubernetes and Azure evidence.
- Translate the engineering intent to Alibaba Cloud ACK, Tencent Cloud TKE, and Volcengine VKE while documenting operational non-equivalence.

Kubernetes is often introduced through commands: create a Deployment, expose it with a Service, inspect Pods, and scale replicas. Those commands are useful, but they obscure the deeper engineering change. Kubernetes turns application structure and lifecycle into API resources. A workload is no longer only a process started by an operator. It is a desired-state object with an owner, selector, Pod template, rollout strategy, status, events, and replacement semantics. A network endpoint is no longer only an IP address. It is a Service and a changing set of ready backends. Persistent storage is no longer only a disk attached to a machine. It is a claim reconciled through a storage class and a CSI driver into a cloud volume with its own lifecycle and cost.

A managed Kubernetes service adds another layer. The cloud provider creates and operates some cluster components, integrates the cluster with identity, network, storage, load balancing, policy, monitoring, upgrades, and billing, and exposes the cluster itself as a cloud resource. Kubernetes then manages application resources inside that cluster. The two control planes overlap but are not interchangeable. Deleting a Kubernetes Service may retire a cloud load balancer; changing a StorageClass can cause future claims to receive a different kind of disk; changing a node pool affects scheduling capacity but not a Deployment spec; deleting a cloud resource behind a controller can cause reconciliation, failure, or renewed cost.

The course foundation for this chapter used three structural abstractions—Service, Workload, and Volume—to explain application definition. This chapter preserves that triad and extends it with the control-plane contracts needed for production: configuration, identity, policy, health, evidence, topology, and cost. It also updates the traffic-entry model. Ingress remains important and widely deployed, but the upstream Ingress API is frozen and Kubernetes recommends Gateway API for new extensible routing designs. The durable

lesson is therefore not a particular controller. It is the separation between application routes, the controller that realizes them, and the cloud infrastructure created on their behalf.

Central claim. Managed Kubernetes is a composition of control planes. The cloud provider governs the cluster and its infrastructure dependencies; Kubernetes governs application desired state; application teams remain responsible for declaring safe, observable, recoverable, and cost-aware workload contracts.

11.1 Kubernetes Is an Application Control Plane Inside a Cloud Control Plane

A Kubernetes cluster contains a control plane and worker nodes. The control plane exposes an API, stores resource state, schedules Pods, and runs controllers that move observed state toward desired state. Nodes run kubelet, a container runtime reached through the Container Runtime Interface (CRI), and networking components. In a managed service, the provider normally operates the control-plane infrastructure while the customer defines cluster configuration, node pools or capacity policy, access, and workloads according to the selected service mode.

This nested model matters because the two planes have different identities, APIs, evidence, and failure modes. The cloud control plane creates an AKS, ACK, TKE, or VKE cluster resource. It also governs subnets, route tables, load balancers, cloud disks, file shares, managed identities, monitoring workspaces, policy assignments, quotas, and billing. The Kubernetes API governs Namespaces, Deployments, Services, PersistentVolumeClaims, Gateway or Ingress objects, ConfigMaps, Secrets, ServiceAccounts, and policy objects. Cloud-controller and CSI/CNI integrations translate selected Kubernetes objects into provider resources.

Managed Kubernetes creates nested control planes

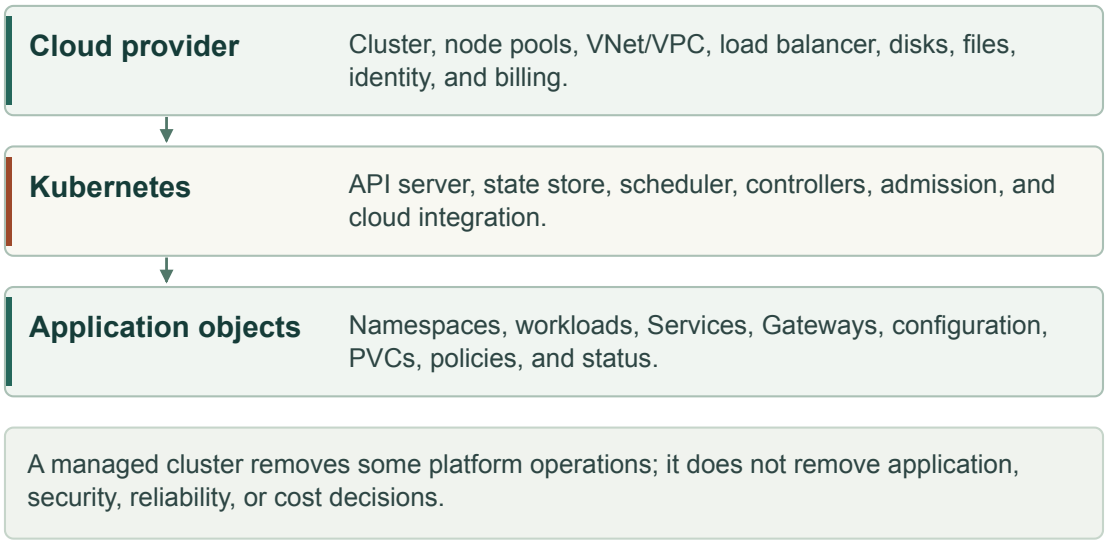


Figure 11-1 Managed Kubernetes creates nested control planes.

Table 11-1 The nested control planes have distinct authority and evidence surfaces.

Plane	Primary resources	Typical authority	Evidence
Cloud provider	Cluster, node pool, VNet/VPC, load balancer, disk/file service, identity, policy, monitoring, bill.	Provider API, cloud IAM, IaC, subscription/account scope.	Activity/audit logs, resource graph, billing records, provider metrics, service health.
Kubernetes	Namespace, workload, Service, Gateway/Ingress, ConfigMap, Secret, PVC, RBAC, policy.	Kubernetes API, ServiceAccounts, users/groups, RBAC, admission.	Object status and conditions, Events, audit logs, rollout history, controller metrics.
Application runtime	Containers, process state, application endpoints, caches, local files, business data.	Application identity, protocol authorization, process and filesystem permissions.	Application logs, metrics, traces, health endpoints, business events, data records.

The cloud-controller-manager is the architectural hinge between the open-source control plane and provider APIs. Its controllers can update Node information, create routes, and provision cloud load balancers for Services. CSI drivers perform a similar translation for storage. This separation allows Kubernetes APIs to express portable intent while providers

evolve their own infrastructure implementations. Portability exists at the API and intent level, not as a promise that every provider creates identical resources or exposes identical operations.

A useful diagnostic question is therefore: which controller owns this object? If a LoadBalancer Service has no external address, inspect the Service, its Events, the cloud-controller integration, cloud permissions, subnet capacity, quota, and provider load-balancer state. If a PVC remains Pending, inspect the claim, StorageClass, CSI controller, zone constraints, quota, and cloud storage API. The error may surface in Kubernetes even though the blocked action occurs in the cloud control plane.

11.2 Managed Kubernetes Changes the Responsibility Boundary

Managed Kubernetes is not one fixed operating model. Providers offer modes with different degrees of control over the control plane, system node pools, upgrades, network defaults, add-ons, and capacity. Azure distinguishes AKS Automatic from AKS Standard. Alibaba Cloud distinguishes ACK managed editions and, historically, dedicated control-plane models. Tencent Cloud offers general and serverless cluster forms. Volcengine VKE similarly manages core cluster components while exposing node pools and provider integrations. The exact product names evolve; the responsibility questions remain stable.

Table 11-2 Managed Kubernetes shifts selected platform responsibilities but preserves customer obligations.

Responsibility	Provider commonly manages	Customer still decides or verifies
Control-plane infrastructure	API server and state-store hosting, component availability, patching, some scaling and backup.	Supported version, upgrade timing or channel, API compatibility, access policy, workload dependency on control-plane availability.
Worker capacity	Node images, managed node-pool operations, auto-healing or serverless capacity in some modes.	Instance family, zones, taints/labels, capacity bounds, disruption tolerance, cost, workload placement.
Networking integration	CNI/cloud-controller integration, load-balancer provisioning, managed DNS or ingress add-ons.	Address plan, public/private exposure, egress, NetworkPolicy, route/firewall requirements, TLS and application authorization.
Storage integration	CSI drivers and built-in StorageClasses.	Access mode, performance, zone behavior, reclaim policy, backup, encryption boundary, data lifecycle.

Responsibility	Provider commonly manages	Customer still decides or verifies
Identity and policy	Cloud identity hooks, Entra/RAM/CAM/IAM integration, policy add-ons.	Human and workload role design, ServiceAccounts, least privilege, secret handling, admission policy, exceptions.
Application operations	Generic controllers and managed add-ons.	Images, manifests, probes, resources, rollout, state, SLOs, recovery, observability and incident response.

The most dangerous misunderstanding is "the control plane is managed, so Kubernetes is managed." A provider can keep the API server available while an application remains unschedulable because requests exceed node capacity, unavailable because replicas share one zone, unsafe because Pods run privileged, or expensive because orphaned load balancers and disks remain. Managed service quality and application quality are separate contracts.

Node pools are an especially important boundary. They are cloud resource templates and Kubernetes scheduling domains at the same time. Their instance types, zones, operating systems, labels, taints, scale settings, maintenance behavior, and billing models influence where Pods can run and how upgrades or failures affect them. A change to a node-pool template may apply only to new nodes. Existing nodes may require rotation, reimage, drain, or upgrade. Application teams must therefore design for node replacement rather than treat nodes as durable servers.

The managed cluster is a graph of cloud and Kubernetes resources

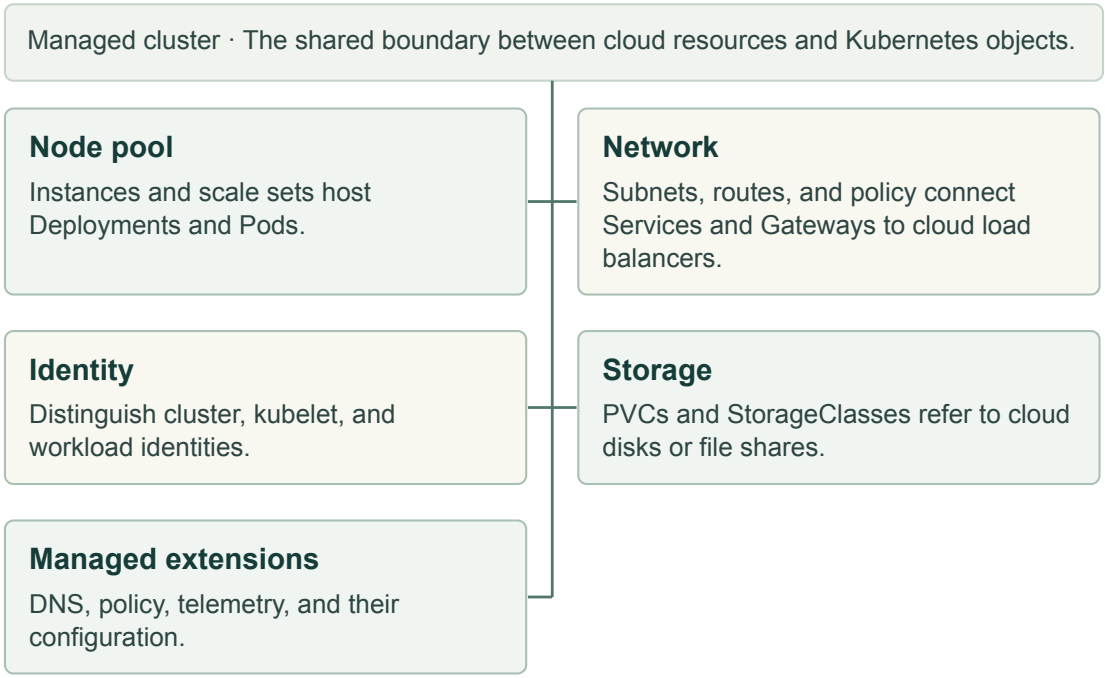


Figure 11-2 A managed cluster is a graph of provider and Kubernetes resources.

Managed modes and defaults are volatile. Record the cluster mode, Kubernetes version, network plug-in, ingress or Gateway implementation, storage classes, upgrade policy, support tier, and enabled add-ons in the architecture record. Do not infer them from the provider brand alone.

11.3 Desired State, Objects, and Reconciliation

Kubernetes objects are persistent declarations of intent. Most objects contain metadata, a spec, and a status. Metadata gives the object a name, namespace, labels, annotations, ownership relationships, and lifecycle metadata. The spec expresses desired state. The status records observed state as reported by the system. The control plane continually works to reduce the difference between the two.

Table 11-3 The object contract separates intent, identity, and observation.

Object element	Purpose	Operational questions
apiVersion / kind	Selects the API group, version, and resource schema.	Is the API served, supported, deprecated, or migrated? Which controller understands it?

Object element	Purpose	Operational questions
metadata	Names and classifies the object and connects ownership or policy.	Which namespace and owner? Which labels are selectors? Which annotations are provider-specific?
spec	Declares desired state.	What outcome should controllers maintain? Which fields are mutable, replace-on-change, or ignored by this implementation?
status	Reports observed state and controller interpretation.	Is the object Ready, progressing, degraded, bound, admitted, or waiting? What generation has been observed?
conditions / Events	Explain state transitions and recent controller decisions.	Why is progress blocked? Is the message durable evidence or a short-lived diagnostic signal?

When a Deployment requests three replicas, the Deployment controller manages a ReplicaSet, which creates Pods from the embedded Pod template. The scheduler chooses nodes for unscheduled Pods. Kubelets ask a CRI-compatible runtime to start containers and report state. Readiness controls whether Pod addresses become eligible backends. A Service controller or Gateway implementation may cause cloud networking resources to appear. None of these actions is a one-time script. Each controller watches state and reconciles again after failure, deletion, scaling, rollout, or policy change.

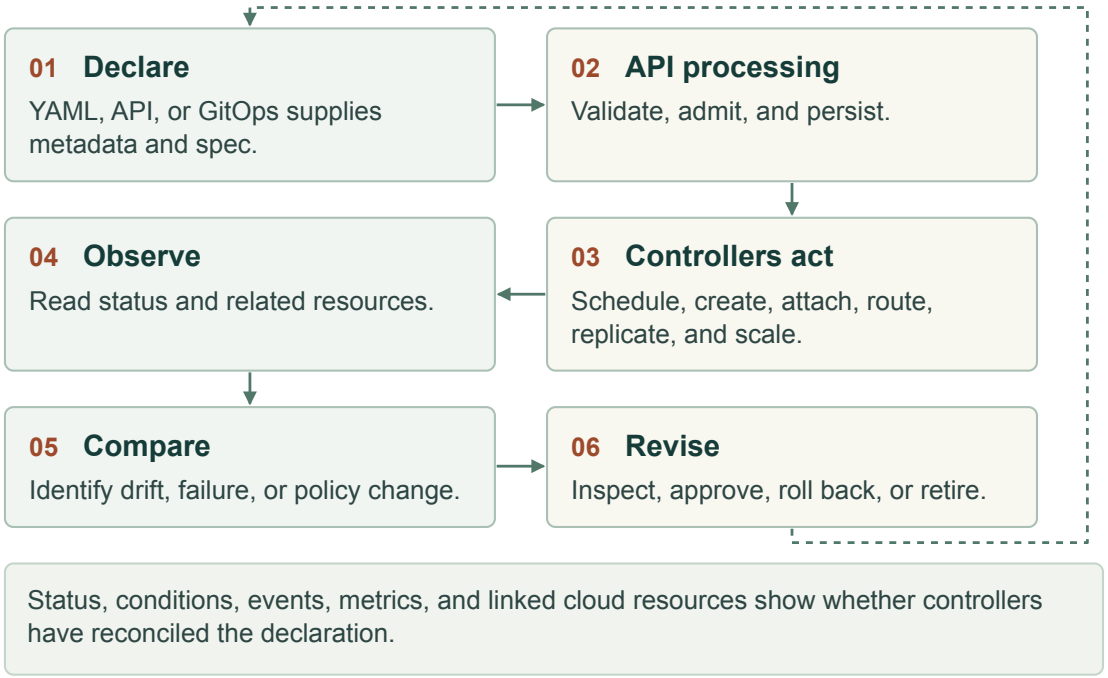


Figure 11-3 Kubernetes application definition is a reconciliation contract.

Labels and selectors are the relational fabric of the API. A Service selects Pods by labels. A Deployment selector must match its Pod template labels. A HorizontalPodAutoscaler targets a scalable workload. Policies select namespaces, Pods, or resources. Because labels create these associations, casual label changes can detach traffic, metrics, policies, or cost attribution. They are part of the application schema, not decoration.

Owner references and garbage collection express lifecycle dependency. A ReplicaSet owned by a Deployment and Pods owned by the ReplicaSet can be cleaned up through cascading deletion. Other relationships, such as a PVC and the cloud disk provisioned for it, depend on reclaim policy rather than simple ownership. Finalizers may delay object deletion until a controller completes external cleanup. A resource stuck in Terminating state often signals an unresolved external lifecycle obligation, not a failed delete command.

Declarative management also requires field ownership. Multiple actors—kubectl, GitOps controllers, operators, admission systems, and platform add-ons—can modify one object. Server-side apply records field managers so conflicting ownership can be detected. The design rule is simple: decide which controller owns each field. Otherwise, two reconcilers can repeatedly overwrite one another while both appear to be functioning correctly.

11.4 The Application Definition Contract

A useful application definition begins with three structural questions: how does the application run, how is it reached, and how does state survive? These map to Workload, Service, and Volume. Production definitions add configuration, identity, policy, topology, evidence, and economics. The application is the graph formed by these objects and relationships, not one YAML file and not one Pod.

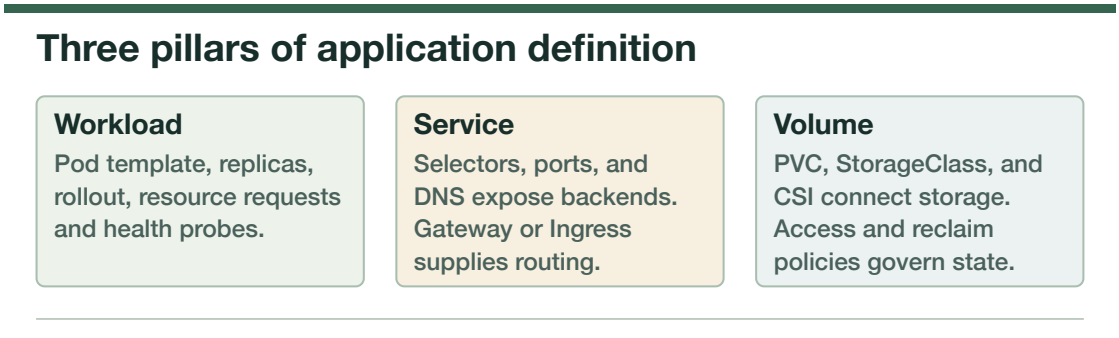


Figure 11-4 Workload, Service, and Volume are the structural pillars of application definition.

Table 11-4 The application definition graph assigns one concern to each resource family.

Concern	Primary Kubernetes objects	Cloud or external dependencies
Runtime lifecycle	Deployment, StatefulSet, DaemonSet, Job, CronJob, Pod template.	Node pools, container runtime, registry, image evidence.
Discovery and traffic	Service, EndpointSlice, Gateway/HTTPRoute or Ingress, NetworkPolicy.	Load balancer, public/private IP, DNS, firewall/WAF, certificates.
Configuration	ConfigMap, environment references, projected volumes.	Configuration service, release system, Git repository.
Sensitive material	Secret reference, ServiceAccount, CSI secret provider.	Cloud secret manager, workload identity, KMS, private endpoint.
Persistent state	PVC, PV, StorageClass, StatefulSet claim template, VolumeSnapshot.	Cloud disk/file/object service, zone, encryption, backup, cost.
Reliability and elasticity	Probes, replicas, PDB, HPA, topology constraints, priorities.	Node-pool capacity, zones, metrics pipeline, provider autoscaling.
Governance and evidence	RBAC, Pod Security, admission policies, status, Events, audit.	Cloud policy, monitoring, activity logs, billing and resource inventory.

Portability depends on where provider-specific assumptions are placed. The Deployment and Service APIs are portable, but annotations that request a particular load-balancer SKU, health probe, internal address, or WAF are not. A PVC is portable as a claim shape, but StorageClass names, parameters, zones, performance tiers, and snapshot capabilities vary. A portable application contract therefore has a stable core plus explicit provider overlays. Hiding the overlay in undocumented annotations does not remove dependency; it only makes dependency harder to review.

Application definition also needs version boundaries. The container digest identifies runtime bytes. ConfigMap and Secret versions define externalized settings. The workload generation identifies desired-state change. Gateway routes, database schemas, and persistent data have separate migration lifecycles. “Version 3 of the application” is useful only when the release record states which versions of these resources were intended to operate together.

11.5 Workload Controllers Encode Different Lifecycles

Kubernetes does not directly manage “applications” as one universal type. It offers workload controllers with different identity, replacement, ordering, completion, and node-locality semantics. Choosing the wrong controller creates operational workarounds that no amount of YAML templating will fix.

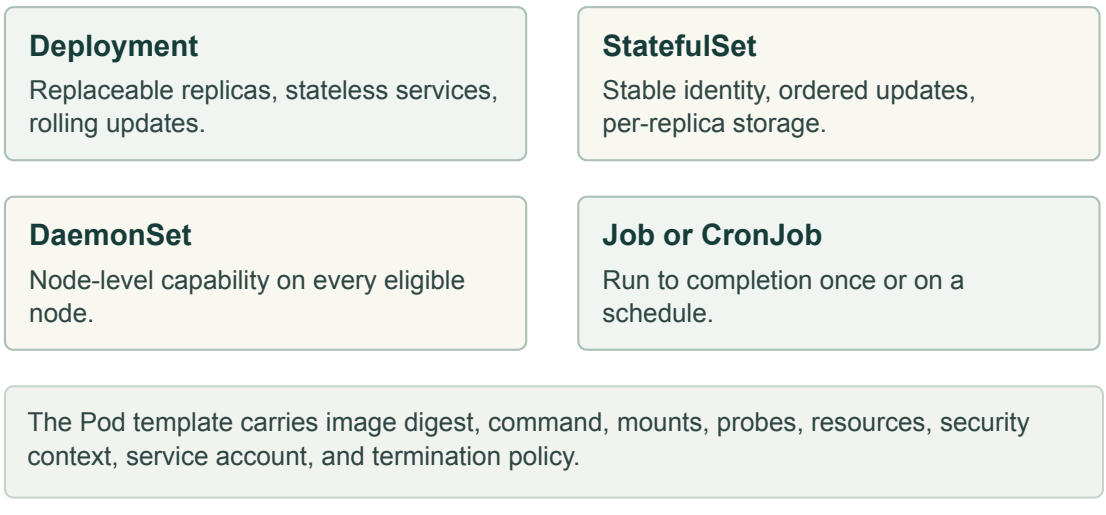


Figure 11-5 Workload controllers are selected by lifecycle semantics.

Table 11-5 Workload controllers express different replacement and identity contracts.

Controller	Use when	Key semantics	Common misuse
Deployment	Replicas are interchangeable and application state is external.	ReplicaSet ownership, rolling update, rollback history, horizontal scaling.	Running identity-sensitive databases while assuming Pod names and disks are stable.
StatefulSet	Each replica needs stable identity, ordered lifecycle, or per-replica storage.	Ordinal identity, stable DNS with a headless Service, ordered or parallel rollout, volumeClaimTemplates.	Assuming StatefulSet automatically provides database replication, backup, or quorum correctness.
DaemonSet	A facility must run on every or selected node.	One Pod per eligible node; follows node addition and removal.	Using it for ordinary replicas or forgetting that node-local agents consume capacity on every pool.
Job	Work must complete successfully and stop.	Completion count, parallelism, retries, deadlines, failure policy.	Using a Deployment for batch work and interpreting restarts as task completion.
CronJob	A Job should be created on a schedule.	Schedule, concurrency policy, missed-run behavior, Job history.	Assuming exactly-once execution without idempotent task design.

A Pod is the atomic scheduling unit, not the durable application identity. Containers in one Pod share network and selected namespaces and can share volumes. That makes the Pod appropriate for tightly coupled containers, such as an application plus a local proxy or a setup container, but not for unrelated services that should scale and fail independently. Workload controllers create Pods from Pod templates and replace them as needed. Directly creating long-lived standalone Pods bypasses much of the lifecycle contract and complicates draining, repair, and rollout.

StatefulSet deserves a precise boundary. It provides stable names and persistent-volume associations; it does not make an application stateful in a correct distributed-systems sense. Replication, leader election, consistency, backup, recovery, schema migration, and split-brain handling remain application or operator responsibilities. The same distinction applies to AI workloads: stable Pods and disks do not by themselves create a reliable model-serving or vector-index cluster.

11.6 The Pod Template Is a Runtime, Health, and Resource Contract

The Pod template inside a workload object defines what every replacement Pod should become. Because controllers may recreate Pods on different nodes, the template must contain everything required to start safely without relying on host-local drift. The container image is only one field in that contract.

Table 11-6 A production Pod template should make these decisions explicit.

Contract area	Representative fields or resources	Why it matters
Artifact	Image by immutable digest, pull policy, imagePullSecrets when required.	Binds runtime to a reviewed supply-chain subject.
Startup	command, args, init containers, dependencies, startupProbe.	Controls initialization order and distinguishes slow start from failure.
Traffic readiness	readinessProbe, ports, graceful termination, preStop where justified.	Prevents traffic from reaching unready or terminating instances.
Liveness	livenessProbe with a narrowly scoped failure test.	Restarts a stuck process; a bad probe can amplify incidents.
Resources	CPU, memory, ephemeral-storage requests and limits.	Drives scheduling, QoS, eviction behavior, HPA interpretation, and cost.
Security	runAsNonRoot, seccomp, read-only root, dropped capabilities, no privilege escalation.	Reduces runtime authority and blast radius.
Identity	ServiceAccount, token automount decision, workload identity annotations/labels.	Defines the principal used for Kubernetes and cloud access.
State	Volumes, mounts, PVC references, emptyDir, projected configuration.	Makes ephemeral and persistent state explicit.
Termination	terminationGracePeriodSeconds, signal handling, shutdown evidence.	Allows endpoints, queues, and state to drain before replacement.

Startup, readiness, and liveness probes answer different questions. A startup probe protects a slow-starting application from premature liveness failure. Readiness asks whether this instance should receive traffic. Liveness asks whether the process should be restarted. A health endpoint that returns success whenever the process can answer HTTP may be ap-

propriate for liveness but insufficient for readiness. Conversely, a liveness probe that fails whenever a downstream dependency is unavailable can restart every healthy replica during a dependency outage.

Resource requests are scheduling and economic declarations. The scheduler uses requests to determine feasible nodes. Limits constrain runtime consumption according to resource type and implementation. HPA CPU utilization commonly uses actual usage relative to requests; missing or unrealistic requests therefore produce bad elasticity decisions. QoS classes influence eviction under pressure. The correct request is not the maximum observed value or an arbitrary fraction of the node—it is a measured reservation that supports the application SLO and a planned burst strategy.

Security context fields are not a substitute for image trust or cluster policy, but they make workload intent reviewable. A namespace-level Pod Security policy can reject unsafe fields, while admission policies can require resource limits, approved registries, labels, or topology settings. Chapter 10 governs the artifact before deployment; this chapter governs how that artifact is allowed to run.

11.7 Service Discovery and External Traffic

Pods are replaceable and their IP addresses can change. A Service provides a stable name, virtual address, and port contract for a changing backend set. Selectors associate the Service with Pods, while EndpointSlices represent ready network endpoints at scale. This indirection lets clients depend on a service identity rather than Pod identity.

Table 11-7 Service and entry resources solve different traffic problems.

Resource	Primary purpose	Cloud interaction
ClusterIP Service	Stable in-cluster discovery and virtual IP for a backend set.	Usually realized by cluster networking and service proxies.
Headless Service	DNS discovery without a cluster virtual IP, often for StatefulSet identities.	Relies on cluster DNS; clients receive backend addresses.
NodePort Service	Expose a port on every node.	Often an implementation layer beneath external load balancing; rarely the preferred public contract alone.
LoadBalancer Service	Request an external or internal L4 load balancer.	Cloud controller provisions provider load balancer, IP, probes, and related rules.

Resource	Primary purpose	Cloud interaction
Ingress	Legacy/frozen Kubernetes HTTP route object interpreted by an Ingress controller.	Controller may create or configure L7 proxy, load balancer, DNS, certificates, or WAF.
Gateway API	Role-oriented, extensible listeners and routes for L4/L7 traffic.	GatewayClass implementation provisions and operates infrastructure; route attachment is policy-controlled.

Stable routing to changing Pods

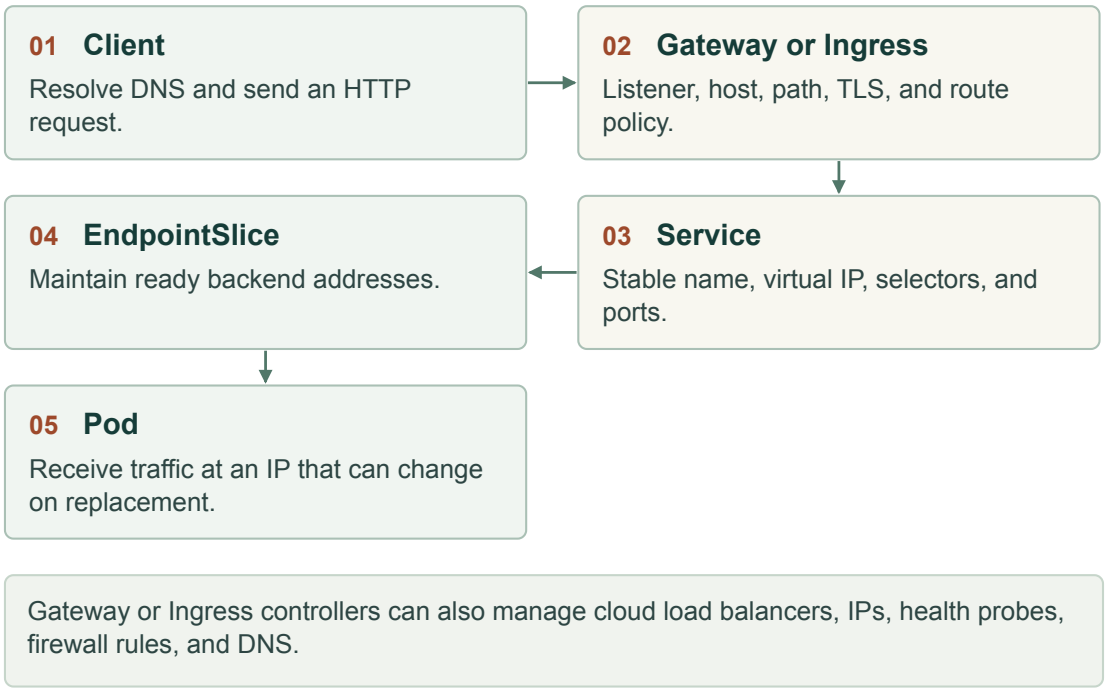


Figure 11-6 Service networking separates stable route intent from changing Pod backends.

Ingress remains important because many clusters, tools, and applications use it, but the upstream API is frozen. Gateway API separates infrastructure ownership from application routing more explicitly through GatewayClass, Gateway, and Route resources. Infrastructure teams can govern listeners and shared gateways while application teams attach routes within allowed boundaries. This role-oriented model is more suitable for platform engineering than a growing set of controller-specific Ingress annotations.

The route object does not carry traffic by itself. A controller must implement it. An accepted HTTPRoute can still be unavailable because the Gateway has no address, the cloud load balancer is pending, the Service has no ready endpoints, DNS is wrong, NetworkPolicy

blocks traffic, TLS material is unavailable, or the application rejects the caller. Status conditions at each layer should be inspected rather than collapsing the whole path into "Ingress is broken."

NetworkPolicy controls allowed Pod traffic only when the network implementation enforces the policy. A Service or Gateway does not authorize business operations. Cloud load-balancer security, TLS, user authentication, API authorization, and service-to-service identity remain separate controls. The reachable path established in Chapter 7 must still end in an authorized application operation.

11.8 Configuration, Secrets, Identity, and Policy

ConfigMap and Secret objects externalize data from the image, but they have different intended sensitivity. A ConfigMap stores non-confidential configuration. A Secret stores small confidential values in the Kubernetes API and can be projected into a Pod or referenced by environment variables. Base64 encoding is not encryption, and a Kubernetes Secret is not automatically equivalent to a cloud secret manager with HSM-backed keys, independent access policy, rotation, private endpoints, and detailed audit.

Table 11-8 Configuration and identity resources should not be treated as interchangeable key-value stores.

Mechanism	Use	Important boundary
ConfigMap	Non-secret environment-specific configuration or mounted files.	Changes may not restart workloads automatically; size and update semantics matter.
Kubernetes Secret	Small sensitive values consumed by Pods or controllers.	Protect API access and encryption at rest; avoid broad RBAC and accidental logging.
External secret manager + CSI/operator	Authoritative secret lifecycle, rotation, audit, KMS and private access.	The integration identity and synchronization behavior become part of the contract.
ServiceAccount	Kubernetes workload identity presented to the API or federation layer.	Use least privilege; disable automatic token mounting when the Pod does not call the API.
Cloud workload identity	Federates a ServiceAccount to an Azure/RAM/CAM/IAM principal.	Cloud role scope is independent from Kubernetes RBAC scope.
Admission and Pod Security policy	Reject unsafe or noncompliant desired state.	Policy does not make an already compromised workload trustworthy; evidence and exceptions are required.

The identity chain typically includes a human or delivery system that submits the manifest, a Kubernetes ServiceAccount used by the Pod, a federated cloud identity used to access provider services, and an application-level identity used for business APIs. Giving a Pod cluster-admin rights because it needs one secret or one cloud read operation collapses these boundaries. Chapter 5 provides the least-privilege model; the Kubernetes implementation is a Role or ClusterRole plus binding for API access and a separately scoped cloud role for provider access.

Namespace is a useful governance boundary but not a complete security boundary. Many resources, including Nodes, StorageClasses, PersistentVolumes, and some policy objects, are cluster-scoped. Namespaces do not isolate the kernel or network unless additional controls enforce that isolation. For strong multi-tenancy, cluster, node-pool, runtime, network, secret, and policy boundaries may be required. The companion book owns the implementation of application tenant isolation; this chapter limits itself to managed cluster and workload contracts.

11.9 Persistent State: PVC, PV, StorageClass, and CSI

Kubernetes separates the application request for storage from the provider volume. A PersistentVolumeClaim declares size, access mode, and storage class. A StorageClass names a provisioning policy and CSI driver. Dynamic provisioning creates a PersistentVolume and a provider storage resource when no existing volume satisfies the claim. The Pod mounts the claim, not a cloud disk identifier.

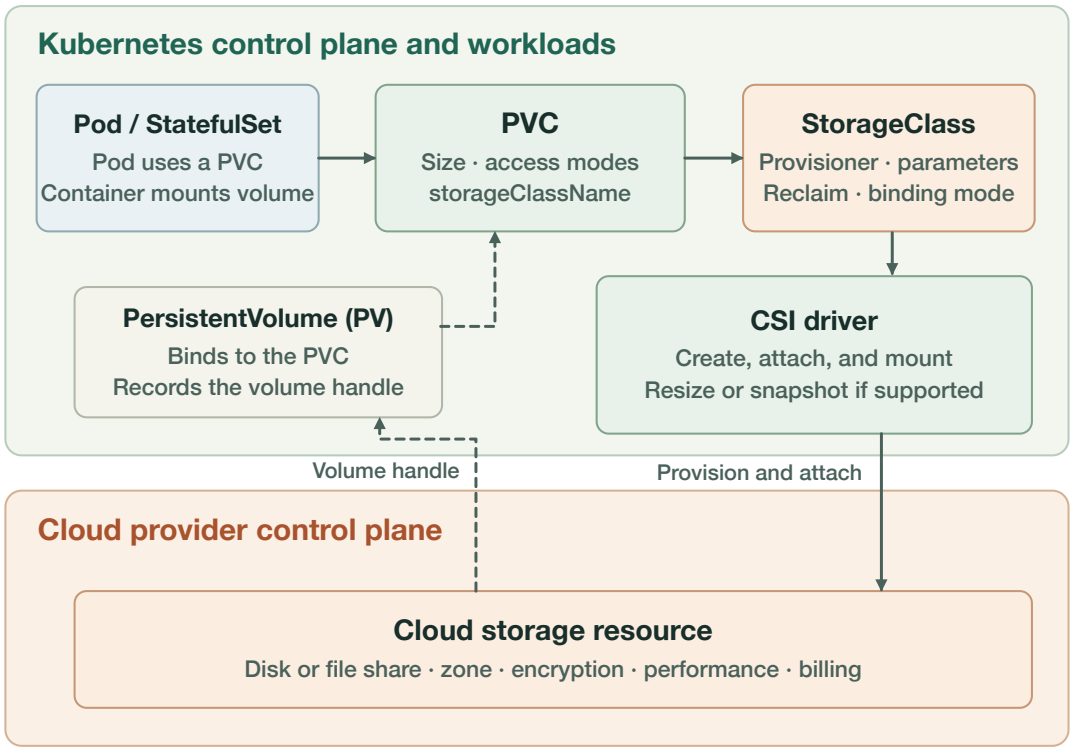


Figure 11-7 Persistent storage crosses the Kubernetes and cloud control planes.

Table 11-9 Persistent storage objects divide application intent from infrastructure implementation.

Object or field	Responsibility	Design decision
PVC	Application request for capacity, access mode, and class.	Size, read/write sharing, expansion, namespace ownership, backup expectation.
PV	Cluster representation of a provisioned or pre-existing volume.	Binding, capacity, access mode, reclaim policy, volume handle, topology.
StorageClass	Administrator-offered storage profile and provisioner parameters.	Performance, redundancy, encryption, binding mode, expansion, reclaim.
CSI driver	Provider plug-in for create, attach, mount, resize, snapshot and lifecycle operations.	Supported features, identity, node limits, version and failure behavior.
Access mode	How the volume can be mounted by nodes or Pods.	RWO/RWX/ROX/ReadWriteOncePod semantics and backend support.
Reclaim policy	What happens to provider storage after the PV is released.	Delete for disposable lab data; Retain when independent recovery or controlled disposal is required.

Object or field	Responsibility	Design decision
Volume binding mode	When provisioning and topology selection occur.	WaitForFirstConsumer can align volume zone with scheduled Pod.

Storage lifecycle is independent from Pod lifecycle. Deleting a Pod normally does not delete its PVC. Deleting a PVC may delete the underlying volume when the StorageClass uses Delete, or preserve it when the PV uses Retain. StatefulSet volumeClaimTemplates create claims per replica, but claim deletion behavior must be reviewed explicitly. A node disk or emptyDir does not become persistent merely because a controller recreates the Pod.

Cloud topology affects binding. A zonal disk can attach only within supported topology, so scheduling and provisioning must agree on zone. Multi-zone clusters do not automatically make every volume multi-zone. Some managed services provide zone-redundant built-in classes; others create zonal disks and rely on rescheduling within the same zone or application replication across volumes. The architecture record must state the actual failure domain.

A PVC is not a backup. Snapshots and backups have their own consistency, retention, identity, restore, and cross-region contracts. A successful remount after Pod replacement proves reattachment of one volume; it does not prove recovery from deletion, corruption, ransomware, or regional failure.

11.10 Scheduling, Availability, Disruption, and Elasticity

Kubernetes availability is not produced by one “high availability” switch. It is composed from scheduler feasibility, replica placement, process health, service readiness, node-pool capacity, disruption policy, cloud zones, storage topology, and external dependencies. Each layer has a different controller and reaction time.

Application availability is a scheduling contract

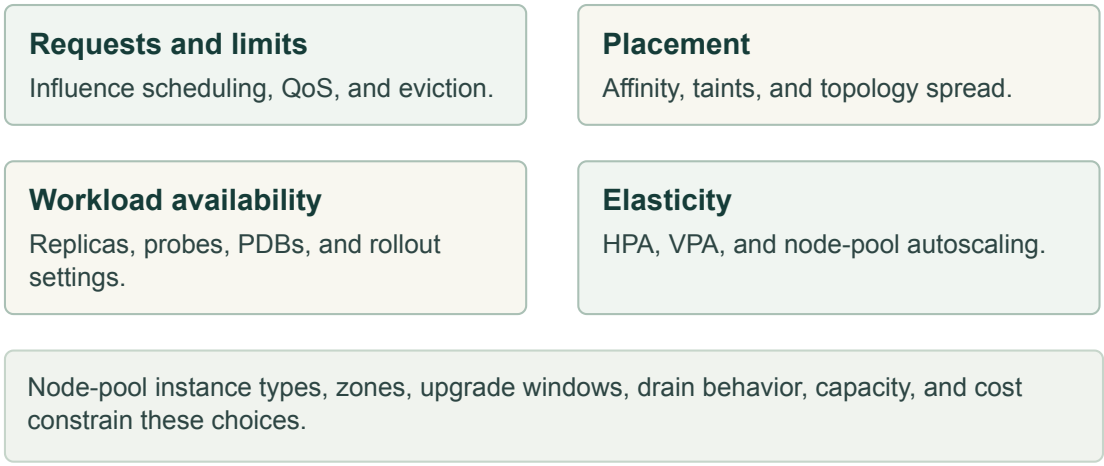


Figure 11-8 Application availability is a composition of scheduling contracts.

Table 11-10 Availability and elasticity controls operate at different layers.

Control	What it changes	What it does not guarantee
Requests / limits	Scheduling reservation, runtime constraint, QoS and eviction behavior.	That enough nodes or quota exist; that the application uses resources efficiently.
Affinity, taints, topology spread	Eligible nodes and distribution across topology domains.	That every zone has capacity or that dependencies are zone-resilient.
Replicas and probes	Number of instances and traffic eligibility or restart behavior.	Independent failure domains, state replication, or safe shutdown.
PodDisruptionBudget	Constrains selected voluntary evictions through the eviction API.	Protection from node failure, direct Pod deletion, bad rollout, or insufficient capacity.
HPA / VPA	Pod replica count or resource recommendations/assignments based on metrics.	Immediate node capacity or correct metrics and requests.
Cluster autoscaler / node auto-provisioning	Node count or node-pool capacity.	Instant readiness; new nodes need quota, placement, bootstrap and image pull time.
Node-pool zones and upgrade policy	Infrastructure failure domains and maintenance behavior.	Application spread unless workload placement rules and replica counts support it.

A PodDisruptionBudget is commonly misunderstood as an availability guarantee. It constrains voluntary evictions performed through the eviction API when sufficient replicas exist. It does not prevent node failure, direct deletion, or an application deployment that replaces too many instances through another path. It can also block node maintenance if the workload lacks spare capacity or its selector is wrong. PDB design must be tested together with rollout and node-drain behavior.

HPA and node scaling form a cascade. HPA may create more pending Pods. The cluster autoscaler or node auto-provisioning layer observes unschedulable demand and attempts to add capacity. Provider quota, subnet addresses, instance stock, zone constraints, startup time, image size, and storage attachment then determine when those Pods become Ready. The end-to-end scaling SLO should measure delivered service capacity, not merely a change in desired replicas.

Managed node pools can automate upgrades, vulnerability remediation, repair, or event response. These operations often drain nodes and move Pods. Workloads that use local state, lack PDBs, have one replica, or cannot start elsewhere will convert routine platform maintenance into outages. The application definition must therefore be designed for the provider operations that the managed service is expected to perform.

11.11 Managed Add-ons Create Provider Resources and Costs

A Kubernetes manifest can create resources outside the Kubernetes API. A LoadBalancer Service or managed Gateway can create a cloud load balancer, frontend IP, health probe, rule, and network interfaces. A PVC can create a disk or file share. A managed ingress add-on creates controllers and cloud integrations. A monitoring add-on creates or connects workspaces and data collection rules. A workload identity configuration creates federated credentials and role assignments. These objects have different owners and deletion semantics.

Table 11-11 Kubernetes intent can materialize provider-managed dependencies.

Kubernetes declaration	Potential provider resources	Lifecycle and cost question
LoadBalancer Service / Gateway	Load balancer, public or private IP, health probes, backend pools, firewall rules.	Who deletes them? Is the endpoint public? Which SKU, data-processing and IP charges apply?
PVC + StorageClass	Managed disk, file share, storage account, snapshot resource.	Delete or Retain? Which zone, redundancy, performance and backup costs apply?

Kubernetes declaration	Potential provider resources	Lifecycle and cost question
Node pool	VM scale set or instance group, disks, NICs, autoscaling resources.	Which nodes are billable? How are upgrades, repair, scale-in and spot eviction handled?
Managed monitoring	Log workspace, Prometheus workspace, agents, rules, dashboards.	Which signals are collected, retained and billed? Is high-cardinality data bounded?
Workload identity / secret integration	Managed identity, federated credential, role assignment, Key Vault/KMS access.	Which principal can access which object and how is revocation verified?
Policy and security add-on	Policy assignments, admission controllers, security plans and findings.	Which resources are covered, what is enforced, and what is the subscription cost?

Controller ownership changes how cleanup must be performed. If a provider load balancer was created from a Kubernetes Service, deleting it directly in the cloud console may create drift or cause the controller to recreate it. Tencent Cloud explicitly warns that declarative TKE resources can recreate underlying CLB or CBS resources if they are deleted outside the owning Kubernetes workflow. The general rule applies across providers: retire the desired-state object first, observe controller cleanup, then verify provider resources and billing scopes.

This is also why cost attribution should include Kubernetes metadata and cloud resource linkage. Namespace, workload, Service, PVC, node pool, and cluster labels are useful allocation dimensions, but cloud bills may show load balancers, disks, IPs, network transfer, monitoring ingestion, and worker instances under provider resource IDs. A reliable FinOps pipeline maps those provider IDs back to Kubernetes owners and desired state before recommending deletion.

11.12 Operations and Evidence

Kubernetes status is distributed. No single screen proves that an application is healthy. A Deployment can be Available while an external route is unresolved. A Gateway can be Programmed while its Service has no ready endpoints. A PVC can be Bound while the filesystem is full or the application cannot read it. Evidence must follow the application graph.

Table 11-12 An application evidence packet should connect desired state, observed state, and provider dependencies.

Evidence	What it demonstrates	Limit
Applied manifests and digests	Declared resources, versions, selectors, routes, storage and policy intent.	Does not prove controllers accepted or realized them.
Object status and conditions	Controller interpretation and observed generation.	May be stale, implementation-specific, or insufficient for end-user behavior.
Events and rollout history	Recent scheduling, pull, mount, probe, route and rollout decisions.	Events are often short-lived and not a durable audit trail.
Endpoint test	A client can reach the intended route and receive an application response.	Does not prove all replicas, identities, zones, or failure paths.
Persistence replacement test	State remains after Pod replacement and volume reattachment.	Does not prove backup, corruption recovery, or regional resilience.
Cloud resource inventory	Which load balancer, IP, disk, node group and identity were materialized.	Must be correlated to the owning Kubernetes object.
Metrics, logs and traces	Runtime behavior, latency, errors, saturation and causal path.	Collection gaps, sampling and retention must be understood.
Cleanup evidence	Desired-state objects and provider dependencies were retired.	Soft-delete, retained volumes, snapshots or delayed billing may remain.

Cluster operations also require evidence. Record the Kubernetes version, managed-service mode, add-on versions, API deprecation findings, node image, upgrade channel, maintenance window, and support policy. Managed services may update control-plane components, system add-ons, or node images at different cadences. Application manifests must avoid removed APIs and should be tested against the target version before cluster upgrade.

The operational unit is the application resource graph. kubectl get pods is necessary and insufficient. Use rollout status, kubectl describe, object YAML, conditions, Events, endpoint tests, storage verification, and cloud resource queries together. Chapter 13 extends this evidence into an SLO-driven feedback loop; Chapter 12 will move application changes into a governed GitOps workflow.

11.13 Multicloud Translation by Engineering Intent

The Kubernetes API creates a strong common vocabulary across clouds, but managed-cluster operations remain provider-specific. The correct translation unit is engineering intent: managed control plane, node-pool lifecycle, network integration, storage provisioning, route realization, identity federation, evidence, and cost. Equivalent YAML does not imply equivalent failure domains, add-on maturity, default policies, or billing.

Table 11-13 Managed Kubernetes comparison by engineering intent.

Intent	Azure AKS	Alibaba Cloud ACK	Tencent Cloud TKE	Volcengine VKE
Managed cluster	AKS Automatic or Standard; provider-operated control plane with different automation boundaries.	ACK managed Basic/Pro; provider-managed control plane and multiple specialized cluster forms.	TKE general and serverless forms based on native Kubernetes.	VKE managed cluster with integrated compute, network, storage and load balancing.
Node capacity	System/user node pools, autoscaling and node auto-provisioning options.	Node pools with managed O&M, scaling, repair, labels/taints and cost insights.	Node pools with lifecycle management and automatic scale-out when Pods are unschedulable.	Managed node pools and heterogeneous-compute options.
Network entry	Azure CNI options; LoadBalancer Service; application routing with Gateway API or other controllers.	Terway/flannel options; SLB/ALB ingress and cloud-native gateway integrations.	VPC-CNI modes, CLB-backed Services and NGINX ingress deployment patterns.	VPC integration, CLB/ALB and APIG ingress options.
Persistent storage	Azure Disk, Files and Blob CSI; built-in managed-csi classes.	Alibaba Cloud disk/NAS/OSS CSI profiles and StorageClasses.	CBS/CFS and related CSI integrations.	EBS/NAS/object integrations through VKE storage add-ons.
Identity and policy	Microsoft Entra access, Azure RBAC options, workload identity, Azure Policy.	RAM and cluster RBAC, RRSA/workload identity patterns, policy and security integrations.	CAM plus Kubernetes RBAC; provider identities and cluster access controls.	IAM plus Kubernetes RBAC and provider-integrated add-ons.
Evidence and economics	Azure Monitor, managed Prometheus, activity logs, node-resource group and Cost Management.	Prometheus/SLS/control-plane logs, node-pool history and cost insights.	Cloud Monitor/CLS, TKE events and provider billing resources.	Managed observability, cluster events, provider resource inventory and billing.

Product editions, add-ons, supported Kubernetes versions, ingress defaults, CNI modes, storage classes, and regional features change frequently. The table maps engineering intent. Verify the current official documentation and record the actual cluster profile before production use.

The most portable layer is the application core: workload controller, Pod template, Service, ConfigMap, ServiceAccount, PVC, health probes, requests, limits, and standard policy resources. Provider overlays should be isolated in explicit files or parameters: StorageClass, GatewayClass or IngressClass, load-balancer annotations, DNS and certificate integration, workload-identity bindings, node-pool selectors, and monitoring add-ons. That structure makes non-equivalence visible and prevents a provider-specific annotation from silently becoming part of the supposed portable core.

Serverless Kubernetes and custom Serverless-on-Kubernetes platforms are intentionally out of scope here. They change who manages nodes and how workloads scale, but they do not remove the application-definition concepts. The companion book develops those runtime implementations in depth.

11.14 Lab 11: Deploy a Managed Kubernetes Workload

This lab validates one capability: a managed Kubernetes cluster can reconcile a complete application definition that includes configuration, a secret reference, a workload, a stable Service, an HTTP entry resource, and persistent storage; the application remains reachable and its persistent state survives Pod replacement; Kubernetes and Azure evidence can be correlated; and cleanup removes the desired-state objects and provider resources created for the lab.

The primary path uses Kubernetes Gateway API because the upstream Ingress API is frozen and AKS is moving its managed application-routing default toward Gateway API. An Ingress comparison is included, but it is not applied to the Gateway-based cluster. The lab is not a production reference architecture. It uses a small, short-lived cluster and one application replica to keep the experiment focused and cost-bounded.

11.14.1 Goal, Architecture, and Acceptance Criteria

Table 11-14 Lab 11 contract.

Item	Requirement
Capability goal	Declare and verify a managed Kubernetes application graph: Namespace, ConfigMap, Secret, PVC, Deployment, Service, Gateway and HTTPRoute.

Item	Requirement
Runtime	One non-root Python HTTP service with startup, readiness and liveness probes and explicit resource requests/limits.
Persistent-state proof	A creation marker written once to the PVC remains unchanged after the Pod is deleted and replaced.
Secret proof	The application reports only that a mounted secret file exists; it never returns the secret value.
Traffic proof	A client reaches the application through the Gateway address and the Service backend.
Control-plane proof	Cluster profile, object manifests, status, conditions, rollout, Events, PVC/PV binding and cloud disk are captured.
Cost boundary	Use a sandbox subscription, a small supported two-node pool or an existing lab cluster, Standard SSD storage, and delete the lab in one session.
Cleanup	Delete the namespace and verify the dynamically provisioned volume is removed; if the cluster was created for the lab, delete the resource group and verify zero remainder.

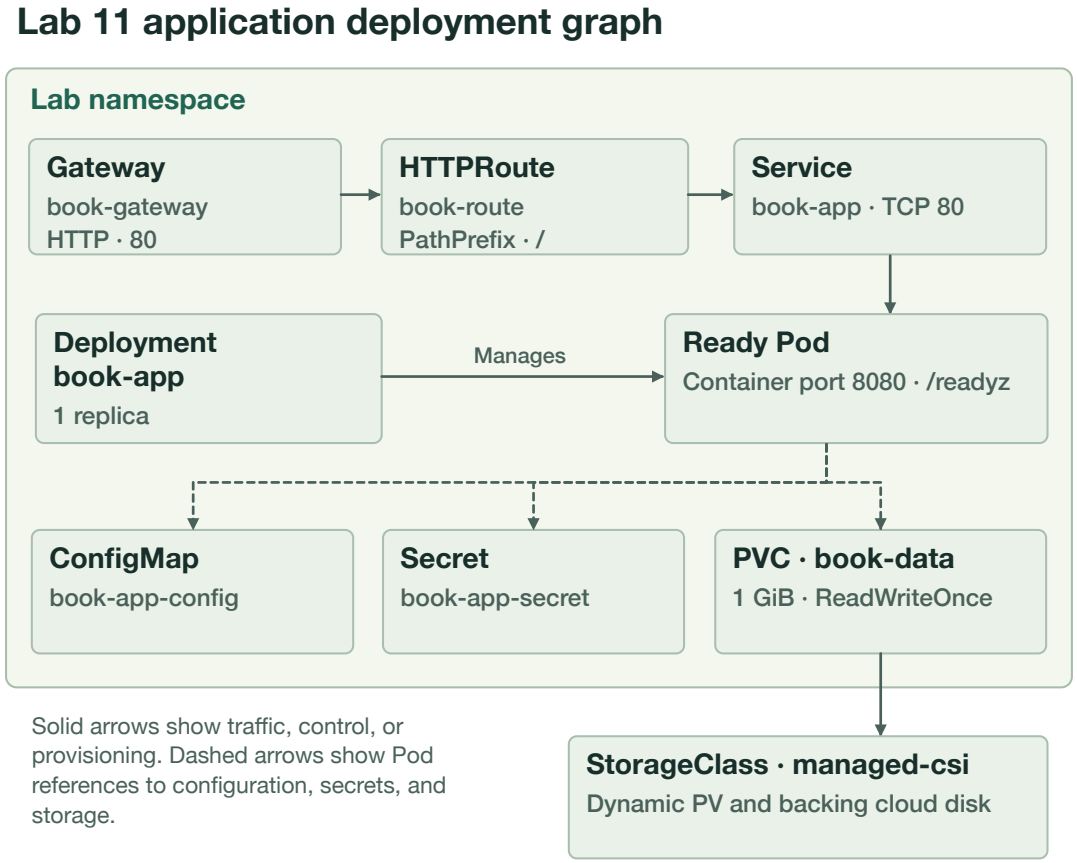


Figure 11-9 Lab 11 deploys the complete application-definition graph.

11.14.2 Prerequisites, Permissions, and Cost Boundary

- A non-production Azure subscription and permission to create a resource group, AKS cluster, role-integrated cluster resources, load balancer and managed disk.
- Azure CLI 2.86.0 or later for the current AKS application-routing Gateway API commands, plus kubectl, curl and jq.
- A region that supports the selected AKS mode and VM size. Replace Standard_D2s_v5 if that SKU is unavailable or prohibited by policy.
- Sufficient vCPU, public IP, load-balancer and managed-disk quota.
- A terminal with Bash semantics. Save command versions in the evidence directory.
- Expected cost: moderate while the cluster, two worker nodes, gateway load balancer, public IP and disk exist. Delete promptly.

If an approved sandbox AKS cluster already has the managed Gateway API application-routing implementation and the managed-csi StorageClass, reuse it and skip cluster creation. Record that the cluster is shared, do not delete it during cleanup, and use a unique namespace.

```
set -euo pipefail
export SUFFIX=$(openssl rand -hex 3)
export LOCATION=eastus
export RG=book-ch11-k8s-$SUFFIX
export AKS=book-aks-$SUFFIX
export NS=book-app-$SUFFIX
export EVIDENCE=$PWD/evidence-ch11-$SUFFIX
mkdir -p "$EVIDENCE"

az version > "$EVIDENCE/azure-cli-version.json"
kubectl version --client -o yaml > "$EVIDENCE/kubectl-client-version.yaml"
az group create --name "$RG" --location "$LOCATION"   --tags chapter=11
purpose=managed-kubernetes owner=book-lab
```

11.14.3 Create or Reuse a Short-Lived AKS Cluster

The following command creates AKS Standard with two nodes, Azure CNI Overlay, OIDC issuer and workload identity enabled, and the managed Gateway API application-routing implementation. It deliberately enables workload identity even though the sample application does not call an Azure data-plane API; the cluster profile then exposes the identity boundary discussed in this chapter without placing credentials in the Pod.

```
az aks create   --resource-group "$RG"   --name "$AKS"   --location "$LOCATION"
--node-count 2   --node-vm-size Standard_D2s_v5   --enable-managed-identity   --
enable-oidc-issuer   --enable-workload-identity   --network-plugin azure   --
network-plugin-mode overlay   --enable-app-routing-istio   --generate-ssh-keys

az aks get-credentials --resource-group "$RG" --name "$AKS" --overwrite-existing
az aks show -g "$RG" -n "$AKS" > "$EVIDENCE/aks-resource.json"
kubectl version -o yaml > "$EVIDENCE/kubernetes-version.yaml"
kubectl get nodes -o wide > "$EVIDENCE/nodes.txt"
kubectl get gatewayclass > "$EVIDENCE/gatewayclasses.txt"
kubectl get storageclass > "$EVIDENCE/storageclasses.txt"
```

AKS features and CLI flags evolve. If the managed Gateway API option is unavailable in the selected region or tenant, use an existing approved implementation and record its GatewayClass, controller, version, support boundary and cloud resources. Do not silently replace the implementation.

11.14.4 Create the Application Definition

Create one manifest. The ConfigMap contains non-secret application code and an environment message. The Secret contains a placeholder token. The application verifies only that the secret file is mounted. A Standard SSD claim uses the built-in managed-csi StorageClass. The init container writes a creation marker only when the volume is empty, allowing the lab to prove that state survives Pod replacement.

```
cat > book-app.yaml <<'YAML'
apiVersion: v1
kind: Namespace
metadata:
  name: REPLACE_NAMESPACE
  labels:
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: book-app-config
  namespace: REPLACE_NAMESPACE
data:
  APP_MESSAGE: "Managed Kubernetes reconciles application intent."
  app.py: |
    import json, os
    from http.server import BaseHTTPRequestHandler, ThreadingHTTPServer
    from pathlib import Path

    DATA = Path('/data')
    SECRET = Path('/var/run/book-secret/token')

    class Handler(BaseHTTPRequestHandler):
        def reply(self, status, payload):
            body = json.dumps(payload, sort_keys=True).encode()
            self.send_response(status)
            self.send_header('Content-Type', 'application/json')
            self.send_header('Content-Length', str(len(body)))
            self.end_headers()
            self.wfile.write(body)

        def do_GET(self):
            if self.path in ('/healthz', '/readyz'):
                return self.reply(200, {'status': 'ok'})
            if self.path != '/':
                return self.reply(404, {'error': 'not found'})
            created = (DATA / 'created-at.txt').read_text().strip()
            secret_state = 'mounted' if SECRET.is_file() and
SECRET.stat().st_size > 0 else 'missing'
            self.reply(200, {
                'service': 'book-managed-kubernetes',
                'message': os.getenv('APP_MESSAGE', 'unset'),
                'createdAt': created,
                'secretState': secret_state,
                'pod': os.getenv('HOSTNAME', 'unknown')
```

```
            })

        def log_message(self, fmt, *args):
            print('%s - %s' % (self.address_string(), fmt % args), flush=True)

    ThreadingHTTPServer(('0.0.0.0', 8080), Handler).serve_forever()
---
apiVersion: v1
kind: Secret
metadata:
  name: book-app-secret
  namespace: REPLACE_NAMESPACE
type: Opaque
stringData:
  token: "lab-placeholder-do-not-use-in-production"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: book-data
  namespace: REPLACE_NAMESPACE
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: managed-csi
  resources:
    requests:
      storage: 1Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: book-app
  namespace: REPLACE_NAMESPACE
  labels:
    app.kubernetes.io/name: book-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: book-app
  template:
    metadata:
      labels:
        app.kubernetes.io/name: book-app
    spec:
      automountServiceAccountToken: false
      securityContext:
```

```
        runAsNonRoot: true
        runAsUser: 10001
        runAsGroup: 10001
        fsGroup: 10001
        seccompProfile:
          type: RuntimeDefault
      initContainers:
        - name: initialize-persistent-state
          image: python:3.12-alpine
          command: ["/bin/sh", "-c"]
          args:
            - |
              set -eu
              if [ ! -f /data/created-at.txt ]; then
                date -u +%Y-%m-%dT%H:%M:%SZ > /data/created-at.txt
              fi
              test -s /var/run/book-secret/token
          securityContext:
            allowPrivilegeEscalation: false
            capabilities:
              drop: ["ALL"]
          volumeMounts:
            - name: data
              mountPath: /data
            - name: secret
              mountPath: /var/run/book-secret
              readOnly: true
      containers:
        - name: app
          image: python:3.12-alpine
          command: ["python", "/app/app.py"]
          env:
            - name: APP_MESSAGE
              valueFrom:
                configMapKeyRef:
                  name: book-app-config
                  key: APP_MESSAGE
            - name: PYTHONDONTWRITEBYTECODE
              value: "1"
            - name: TMPDIR
              value: /tmp
          ports:
            - name: http
              containerPort: 8080
          startupProbe:
            httpGet: { path: /healthz, port: http }
            failureThreshold: 30
            periodSeconds: 2
```

```
          readinessProbe:
            httpGet: { path: /readyz, port: http }
            periodSeconds: 5
          livenessProbe:
            httpGet: { path: /healthz, port: http }
            periodSeconds: 10
          resources:
            requests: { cpu: 50m, memory: 64Mi }
            limits: { cpu: 250m, memory: 192Mi }
          securityContext:
            allowPrivilegeEscalation: false
            readOnlyRootFilesystem: true
            capabilities:
              drop: ["ALL"]
          volumeMounts:
            - name: app-code
              mountPath: /app
              readOnly: true
            - name: data
              mountPath: /data
            - name: secret
              mountPath: /var/run/book-secret
              readOnly: true
            - name: tmp
              mountPath: /tmp
      volumes:
        - name: app-code
          configMap:
            name: book-app-config
            items:
              - key: app.py
                path: app.py
        - name: data
          persistentVolumeClaim:
            claimName: book-data
        - name: secret
          secret:
            secretName: book-app-secret
        - name: tmp
          emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: book-app
  namespace: REPLACE_NAMESPACE
spec:
  selector:
```

```
    app.kubernetes.io/name: book-app
  ports:
    - name: http
      port: 80
      targetPort: http
---
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: book-gateway
  namespace: REPLACE_NAMESPACE
spec:
  gatewayClassName: approuting-istio
  listeners:
    - name: http
      protocol: HTTP
      port: 80
      allowedRoutes:
        namespaces:
          from: Same
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: book-route
  namespace: REPLACE_NAMESPACE
spec:
  parentRefs:
    - name: book-gateway
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /
      backendRefs:
        - name: book-app
          port: 80
YAML

sed -i "s/REPLACE_NAMESPACE/$NS/g" book-app.yaml
cp book-app.yaml "$EVIDENCE/book-app.yaml"
```

The Python base image tag is readable for the lab. A production manifest should reference an approved digest and should obtain sensitive material from an external secret lifecycle rather than placing literal stringData in a file. The placeholder Secret is used only to demonstrate resource wiring.

11.14.5 Apply and Inspect Desired State

```
kubectl apply -f book-app.yaml
kubectl rollout status deployment/book-app -n "$NS" --timeout=10m
kubectl wait --for=jsonpath='{.status.phase}'=Bound pvc/book-data -n "$NS" --
timeout=10m

kubectl get deployment,pod,service,pvc,gateway,httproute -n "$NS" -o wide   |
tee "$EVIDENCE/application-summary.txt"
kubectl get deployment/book-app -n "$NS" -o yaml > "$EVIDENCE/deployment.yaml"
kubectl get gateway/book-gateway -n "$NS" -o yaml > "$EVIDENCE/gateway.yaml"
kubectl get httproute/book-route -n "$NS" -o yaml > "$EVIDENCE/httproute.yaml"
kubectl get pvc/book-data -n "$NS" -o yaml > "$EVIDENCE/pvc.yaml"
kubectl get events -n "$NS" --sort-by=.lastTimestamp > "$EVIDENCE/events-before-
test.txt"
```

Inspect status.observedGeneration, Deployment conditions, Pod readiness, PVC phase, Gateway conditions and HTTPRoute parent status. A successful apply is only API acceptance. The acceptance test requires the relevant controllers to report realized state.

11.14.6 Verify the HTTP Entry Path

Wait for the managed Gateway to receive an address. The address can be an IP or hostname. Save the route status before testing the application.

```
for i in $(seq 1 90); do
  ADDRESS=$(kubectl get gateway book-gateway -n "$NS"     -o
jsonpath='{.status.addresses[0].value}' 2>/dev/null || true)
  [ -n "$ADDRESS" ] && break
  sleep 10
done

if [ -z "${ADDRESS:-}" ]; then
  kubectl describe gateway book-gateway -n "$NS"
  kubectl describe httproute book-route -n "$NS"
  exit 1
fi

echo "$ADDRESS" | tee "$EVIDENCE/gateway-address.txt"
curl --retry 20 --retry-delay 5 --fail --show-error   "http://$ADDRESS/" | tee
"$EVIDENCE/response-before.json"
jq . "$EVIDENCE/response-before.json"
```

The expected response contains the configured message, a persistent createdAt timestamp, secretState: mounted, and the current Pod name. It must not contain the Secret value. If the Gateway has an address but curl fails, inspect the HTTPRoute Accepted and

ResolvedRefs conditions, Service endpoints, Pod readiness and Gateway implementation logs before changing the application.

11.14.7 Prove Persistent State Survives Pod Replacement

```
BEFORE=$(jq -r .createdAt "$EVIDENCE/response-before.json")
OLD_POD=$(kubectl get pod -n "$NS" -l app.kubernetes.io/name=book-app   -o
jsonpath='{.items[0].metadata.name}')

echo "$OLD_POD" | tee "$EVIDENCE/old-pod.txt"
kubectl delete pod "$OLD_POD" -n "$NS"
kubectl rollout status deployment/book-app -n "$NS" --timeout=10m

NEW_POD=$(kubectl get pod -n "$NS" -l app.kubernetes.io/name=book-app   -o
jsonpath='{.items[0].metadata.name}')
echo "$NEW_POD" | tee "$EVIDENCE/new-pod.txt"

curl --retry 20 --retry-delay 5 --fail --show-error   "http://$ADDRESS/" | tee
"$EVIDENCE/response-after.json"
AFTER=$(jq -r .createdAt "$EVIDENCE/response-after.json")

test "$OLD_POD" != "$NEW_POD"
test "$BEFORE" = "$AFTER"
printf 'persistent timestamp: %s
' "$AFTER"   | tee "$EVIDENCE/persistence-verification.txt"
```

This test proves a narrow claim: the replacement Pod mounted the same persistent volume and observed the original marker. It does not prove multi-zone availability, backup, application-consistent restore, or protection from data corruption. Record the claim at the scope actually tested.

11.14.8 Inspect the Cloud Resources Created by Kubernetes

```
PV=$(kubectl get pvc book-data -n "$NS" -o jsonpath='{.spec.volumeName}')
VOLUME_ID=$(kubectl get pv "$PV" -o jsonpath='{.spec.csi.volumeHandle}')
NODE_RG=$(az aks show -g "$RG" -n "$AKS" --query nodeResourceGroup -o tsv)

printf 'PV=%s
VOLUME_ID=%s
NODE_RG=%s
' "$PV" "$VOLUME_ID" "$NODE_RG"   | tee "$EVIDENCE/cloud-links.txt"
kubectl get pv "$PV" -o yaml > "$EVIDENCE/pv.yaml"
az disk show --ids "$VOLUME_ID" > "$EVIDENCE/azure-disk.json"
az resource list -g "$NODE_RG" > "$EVIDENCE/node-resource-group.json"
az network lb list -g "$NODE_RG" > "$EVIDENCE/load-balancers.json"
az network public-ip list -g "$NODE_RG" > "$EVIDENCE/public-ips.json"
```

The evidence should connect the PVC to its PV and CSI volumeHandle, then to the Azure managed disk. It should also show the load-balancing resources used by the managed Gateway implementation. Names and internal topology vary by implementation; use owner links, resource IDs, labels and status rather than relying on name patterns alone.

11.14.9 Build the Evidence Packet

Table 11-15 Required evidence for Lab 11.

Evidence file	Purpose
aks-resource.json / kubernetes-version.yaml	Records managed-service mode, identity, network profile, OIDC settings and Kubernetes version.
book-app.yaml	Records desired application state and provider-specific GatewayClass/StorageClass choices.
deployment.yaml / gateway.yaml / httproute.yaml	Records observed generations, conditions and controller status.
pvc.yaml / pv.yaml / azure-disk.json	Links application storage intent to the provider volume.
response-before.json / response-after.json	Proves route behavior, configuration injection, secret presence without disclosure, and state continuity.
old-pod.txt / new-pod.txt / persistence-verification.txt	Proves replacement occurred and the marker remained unchanged.
events-before-test.txt / events-after-test.txt	Captures scheduling, pulling, mounting, probing and route diagnostics.
node-resource-group.json / load-balancers.json / public-ips.json	Shows provider resources created to realize Kubernetes intent.
cleanup files	Proves namespace, disk and—if created for the lab—cluster resource groups were retired.

```
kubectl get events -n "$NS" --sort-by=.lastTimestamp   > "$EVIDENCE/events-
after-test.txt"
kubectl auth can-i --list -n "$NS" > "$EVIDENCE/current-user-k8s-
permissions.txt"
sha256sum "$EVIDENCE"/* > "$EVIDENCE/evidence-packet.sha256"
ls -lh "$EVIDENCE"
```

11.14.10 Ingress Compatibility Note

For a cluster that intentionally uses the AKS managed NGINX application-routing add-on, an equivalent legacy entry object would use ingressClassName: webapprouting.kubernetes.azure.com. Do not apply this object to the Gateway-only lab unless the NGINX controller is installed. Keep the Service unchanged and replace the Gateway/HTTPRoute pair with the following resource:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: book-ingress
  namespace: REPLACE_NAMESPACE
spec:
  ingressClassName: webapprouting.kubernetes.azure.com
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: book-app
                port:
                  number: 80
```

The application intent is the same—route HTTP traffic to the book-app Service—but the object model, controller support lifecycle, role separation and generated infrastructure differ. This is a concrete example of interface similarity without complete operational equivalence.

11.14.11 Common Failures and Their Meaning

Table 11-16 Lab failure modes and diagnostic questions.

Symptom	Likely layer	Diagnostic path
Pod Pending	Scheduling / capacity	Describe Pod; inspect requests, node taints, quota, node readiness and zone constraints.
ImagePullBackOff	Registry / egress / artifact	Inspect image reference, DNS/egress, registry availability and pull credentials.
PVC Pending	Storage control path	Describe PVC; inspect StorageClass, CSI pods, quota, topology and Events.

Symptom	Likely layer	Diagnostic path
Pod Running but not Ready	Application health	Inspect readiness endpoint, logs, configuration, volume mounts and probe timing.
Gateway has no address	Gateway controller / cloud networking	Inspect Gateway conditions, controller namespace, provider quota, subnet/IP and cloud activity logs.
HTTPRoute not Accepted	Route policy or reference	Inspect parentRefs, allowedRoutes, backend Service name/port and ResolvedRefs condition.
Curl reaches 503	Service backends	Inspect EndpointSlices and Pod readiness; confirm Service selector matches template labels.
Multi-Attach error	Volume access/topology	Confirm one replica for RWO disk, prior attachment release, node zone and CSI state.
Timestamp changes after replacement	Persistent mount or init logic	Confirm PVC mount, PV identity, reclaim behavior and application path.

11.14.12 Cleanup and Zero-Remainder Verification

Delete the namespace first so the lab can observe Kubernetes-driven cleanup of the claim and its provider volume. Save the volume ID before deletion. If the cluster was created solely for the lab, delete the resource group afterward. If the cluster is shared, do not delete the cluster or node resource group.

```
kubectl delete namespace "$NS"
kubectl wait --for=delete namespace/"$NS" --timeout=15m || true

# Verify the dynamically provisioned disk disappears under the Delete reclaim
policy.
for i in $(seq 1 60); do
  if ! az disk show --ids "$VOLUME_ID" >/dev/null 2>&1; then
    echo "disk deleted" | tee "$EVIDENCE/disk-cleanup.txt"
    break
  fi
  sleep 10
done

kubectl get namespace "$NS" > "$EVIDENCE/namespace-after-cleanup.txt" 2>&1 ||
true

# Run only when this lab created the cluster resource group.
```

```
az group delete --name "$RG" --yes --no-wait
az group wait --deleted --name "$RG"
az group exists --name "$RG" | tee "$EVIDENCE/resource-group-exists-after-
cleanup.txt"
```

A custom StorageClass with Retain, a snapshot, a backup vault, a static public IP, a DNS zone or an external certificate can survive namespace deletion. Cleanup evidence must follow the actual resource graph, not assume that deleting the cluster removes every related resource.

11.14.13 Translate the Lab to China's Leading Cloud Platforms

Table 11-17 Lab 11 translation by managed Kubernetes capability.

Lab intent	ACK	TKE	VKE
Managed cluster and node pool	ACK managed cluster and node pool; record edition, zones, CNI and managed O&M settings.	TKE general cluster and node pool; record runtime, VPC mode, scaling and version support.	VKE managed cluster and managed node pool; record network, heterogeneous capacity and lifecycle settings.
HTTP entry	Use a supported ALB/SLB ingress or Gateway implementation and record class/controller plus created cloud resources.	Use CLB-backed Service or approved NGINX/Gateway implementation and record controller ownership.	Use supported ALB, CLB or APIG ingress/Gateway integration and record the provisioned entry resources.
Persistent storage	Select an approved disk or NAS CSI StorageClass and verify PV volume handle to provider resource.	Use CBS or CFS CSI class and verify claim, PV and provider storage lifecycle.	Use VKE-supported block/file CSI class and verify binding, zone and reclaim behavior.
Configuration and secret reference	Use standard ConfigMap/Secret core objects; externalize production secrets to KMS/secret services where supported.	Use standard core objects with CAM/RBAC and approved secret integration.	Use standard core objects with IAM/RBAC and approved secret integration.
Evidence and cleanup	Capture ACK events/control-plane logs, SLS/Prometheus evidence, provider resource IDs and cost scope.	Capture TKE Events, CloudAudit/CLS/monitoring and CLB/CBS resources; delete through owning Kubernetes objects.	Capture VKE status, observability, audit and dependent cloud resources; verify billing cleanup.

11.15 CloudDesk Reuse

CloudDesk FinOps Agent should not receive unrestricted kubectl, cloud CLI, or cluster-admin credentials. This chapter produces bounded, primarily read-oriented capabilities that can be implemented by a Tool Broker with explicit cluster, namespace, object-kind and cloud-account scope. Mutation should remain a separate plan–approve–execute–verify workflow.

Table 11-18 CloudDesk capability handoff.

Capability	Read contract	Possible governed action
k8s.workload.inventory	List workloads, replicas, images, requests/limits, probes, placement and status within approved scopes.	Propose resource or replica changes; do not patch directly without approval and rollout verification.
k8s.ingress.audit	Map Gateway/Ingress, Service, EndpointSlice and provider load-balancer exposure.	Propose internalization, TLS, WAF or route cleanup with network and ownership evidence.
k8s.storage.recommend	Map PVC/PV/StorageClass to provider volume, access mode, reclaim, zone, utilization and cost.	Propose resize, class migration, snapshot/backup or orphan cleanup through a reviewed plan.
k8s.application.definition.validate	Evaluate object graph completeness, selectors, probes, resources, identity, policy and provider overlays.	Open a GitOps change proposal rather than mutating live objects.
k8s.drift.explain	Compare desired manifests, object generations, controller status and provider resource state.	Recommend reconciliation owner and evidence to collect; avoid competing with an existing controller.

For the CloudDesk FinOps use case, the most valuable connection is between Kubernetes intent and provider billing. A recommendation to delete an “unused” load balancer is unsafe until the agent proves which Service or Gateway owns it, whether it is still desired, which routes depend on it, and whether deletion will be reconciled into existence again. Similarly, a disk is not orphaned merely because no VM is attached; it may be bound to a PVC or retained for recovery. Resource ownership precedes cost action.

11.16 Key Takeaways

- Kubernetes is an application control plane; a managed Kubernetes service nests it inside the provider cloud control plane.
- Managed control-plane availability does not remove customer responsibility for workload definition, access, placement, state, SLOs, upgrades, evidence, or cost.

- Kubernetes objects separate identity and desired spec from observed status; controllers continuously reconcile the difference.
- Application definition is a resource graph. Workload, Service, and Volume are structural pillars surrounded by configuration, identity, policy, topology, evidence, and economics.
- Deployment, StatefulSet, DaemonSet, Job, and CronJob encode different lifecycle semantics. A Pod is a replaceable scheduling unit, not a durable application identity.
- A Pod template must declare artifact, startup, health, resources, security, identity, state and termination behavior.
- Service and EndpointSlice create stable discovery over changing Pods. Ingress is frozen; Gateway API provides a role-oriented successor, but both require an implementation controller.
- ConfigMap, Secret, external secret manager, ServiceAccount, cloud workload identity and admission policy solve different problems.
- PVC, PV, StorageClass and CSI translate application storage intent into a provider volume; reclaim, topology, backup and restore remain explicit decisions.
- Replicas, probes, PDB, HPA, topology spread and node scaling operate at different layers and must be tested as one availability path.
- Kubernetes objects can create billable cloud resources. Cleanup must begin with the owning desired-state object and end with provider-resource and billing verification.
- Equivalent Kubernetes YAML across AKS, ACK, TKE and VKE does not establish operational equivalence; record controller, defaults, failure domains, add-ons, evidence and cost.
- Lab 11 proves a narrow but valuable contract: a complete application graph is reconciled, externally reachable, persistently stateful across Pod replacement, and traceable to its Azure resources.
- Agent tools should consume bounded inventory and explanation contracts, not unrestricted cluster administration.

References and Official Documentation

[1] National Institute of Standards and Technology. The NIST Definition of Cloud Computing. NIST SP 800-145, 2011. https://doi.org/10.6028/NIST.SP.800-145

[2] Kubernetes. Cluster Architecture. https://kubernetes.io/docs/concepts/architecture/

[3] Kubernetes. Objects In Kubernetes. https://kubernetes.io/docs/concepts/overview/working-with-objects/

[4] Kubernetes. Controllers. https://kubernetes.io/docs/concepts/architecture/controller/

[5] Kubernetes. Cloud Controller Manager. https://kubernetes.io/docs/concepts/architecture/cloud-controller/

[6] Kubernetes. Nodes. https://kubernetes.io/docs/concepts/architecture/nodes/

[7] Kubernetes. Kubernetes Self-Healing. https://kubernetes.io/docs/concepts/architecture/self-healing/

[8] Kubernetes. Pods. https://kubernetes.io/docs/concepts/workloads/pods/

[9] Kubernetes. Pod Lifecycle. https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/

[10] Kubernetes. Workloads. https://kubernetes.io/docs/concepts/workloads/

[11] Kubernetes. Deployments. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

[12] Kubernetes. StatefulSets. https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

[13] Kubernetes. DaemonSet. https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/

[14] Kubernetes. Jobs. https://kubernetes.io/docs/concepts/workloads/controllers/job/

[15] Kubernetes. CronJob. https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

[16] Kubernetes. Namespaces. https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

[17] Kubernetes. Labels and Selectors. https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

[18] Kubernetes. Garbage Collection. https://kubernetes.io/docs/concepts/architecture/garbage-collection/

[19] Kubernetes. ConfigMaps. https://kubernetes.io/docs/concepts/configuration/configmap/

[20] Kubernetes. Secrets. https://kubernetes.io/docs/concepts/configuration/secret/

[21] Kubernetes. Service Accounts. https://kubernetes.io/docs/concepts/security/service-accounts/

[22] Kubernetes. Role Based Access Control Good Practices. https://kubernetes.io/docs/concepts/security/rbac-good-practices/

[23] Kubernetes. Pod Security Standards. https://kubernetes.io/docs/concepts/security/pod-security-standards/

[24] Kubernetes. Pod Security Admission. https://kubernetes.io/docs/concepts/security/pod-security-admission/

[25] Kubernetes. Resource Management for Pods and Containers. https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

[26] Kubernetes. Pod Quality of Service Classes. https://kubernetes.io/docs/concepts/workloads/pods/pod-qos/

[27] Kubernetes. Configure Liveness, Readiness and Startup Probes. https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/

[28] Kubernetes. Horizontal Pod Autoscaling. https://kubernetes.io/docs/concepts/workloads/autoscaling/horizontal-pod-autoscale/

[29] Kubernetes. Disruptions and Pod Disruption Budgets. https://kubernetes.io/docs/concepts/workloads/pods/disruptions/

[30] Kubernetes. Pod Topology Spread Constraints. https://kubernetes.io/docs/concepts/scheduling-eviction/topology-spread-constraints/

[31] Kubernetes. Service. https://kubernetes.io/docs/concepts/services-networking/service/

[32] Kubernetes. EndpointSlices. https://kubernetes.io/docs/concepts/services-networking/endpoint-slices/

[33] Kubernetes. Ingress. https://kubernetes.io/docs/concepts/services-networking/ingress/

[34] Kubernetes. Gateway API. https://kubernetes.io/docs/concepts/services-networking/gateway/

[35] Kubernetes. Network Policies. https://kubernetes.io/docs/concepts/services-networking/network-policies/

[36] Kubernetes. Persistent Volumes. https://kubernetes.io/docs/concepts/storage/persistent-volumes/

[37] Kubernetes. Storage Classes. https://kubernetes.io/docs/concepts/storage/storage-classes/

[38] Kubernetes. Dynamic Volume Provisioning. https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/

[39] Kubernetes. Volumes. https://kubernetes.io/docs/concepts/storage/volumes/

[40] Kubernetes. Volume Snapshots. https://kubernetes.io/docs/concepts/storage/volume-snapshots/

[41] Kubernetes. Container Runtime Interface. https://kubernetes.io/docs/concepts/architecture/cri/

[42] Microsoft. What Is Azure Kubernetes Service (AKS)? https://learn.microsoft.com/en-us/azure/aks/what-is-aks

[43] Microsoft. Core Concepts for Azure Kubernetes Service. Updated July 29, 2026. https://learn.microsoft.com/en-us/azure/aks/core-aks-concepts

[44] Microsoft. Support Policies for Azure Kubernetes Service. Updated June 9, 2026. https://learn.microsoft.com/en-us/azure/aks/support-policies

[45] Microsoft. Managed Identities in Azure Kubernetes Service. Updated August 18, 2026. https://learn.microsoft.com/en-us/azure/aks/managed-identity-overview

[46] Microsoft. Deploy and Configure Microsoft Entra Workload ID on AKS. https://learn.microsoft.com/en-us/azure/aks/workload-identity-deploy-cluster

[47] Microsoft. Networking Concepts in Azure Kubernetes Service. Updated April 13, 2026. https://learn.microsoft.com/en-us/azure/aks/concepts-network

[48] Microsoft. Security Concepts for Applications and Clusters in AKS. Updated June 4, 2026. https://learn.microsoft.com/en-us/azure/aks/concepts-security

[49] Microsoft. Storage Options for Applications in AKS. Updated July 31, 2026. https://learn.microsoft.com/en-us/azure/aks/concepts-storage

[50] Microsoft. Use CSI Drivers on AKS. Updated August 3, 2026. https://learn.microsoft.com/en-us/azure/aks/csi-storage-drivers

[51] Microsoft. Create and Manage Persistent Volumes with Azure Disks in AKS. Updated June 3, 2026. https://learn.microsoft.com/en-us/azure/aks/create-volume-azure-disk

[52] Microsoft. Managed NGINX Ingress with the Application Routing Add-on. Updated August 3, 2026. https://learn.microsoft.com/en-us/azure/aks/app-routing

[53] Microsoft. Configure Ingress with Kubernetes Gateway API via the Application Routing Add-on. Updated July 1, 2026. https://learn.microsoft.com/en-us/azure/aks/app-routing-gateway-api

[54] Microsoft. Best Practices for Network Resources in AKS. Updated April 13, 2026. https://learn.microsoft.com/en-us/azure/aks/operator-best-practices-network

[55] Microsoft. Deployment and Cluster Reliability Best Practices for AKS. https://learn.microsoft.com/en-us/azure/aks/best-practices-app-cluster-reliability

[56] Microsoft. Create Node Pools in Azure Kubernetes Service. Updated July 13, 2026. https://learn.microsoft.com/en-us/azure/aks/create-node-pools

[57] Microsoft. Use System Node Pools in Azure Kubernetes Service. Updated April 30, 2026. https://learn.microsoft.com/en-us/azure/aks/use-system-pools

[58] Microsoft. Use the Azure Key Vault Provider for Secrets Store CSI Driver in AKS. Updated May 5, 2026. https://learn.microsoft.com/en-us/azure/aks/csi-secrets-store-driver

[59] Alibaba Cloud. What Is Container Service for Kubernetes (ACK)? Updated June 18, 2026. https://www.alibabacloud.com/help/en/ack/ack-managed-and-ack-dedicated/product-overview/what-is-ack

[60] Alibaba Cloud. ACK Cluster Overview. Updated May 20, 2026. https://www.alibabacloud.com/help/en/ack/ack-managed-and-ack-dedicated/user-guide/ack-cluster-overview/

[61] Alibaba Cloud. Overview of Node Pools. Updated July 9, 2026. https://www.alibabacloud.com/help/en/ack/ack-managed-and-ack-dedicated/user-guide/node-pool-overview

[62] Alibaba Cloud. ACK User Guide. Updated June 15, 2026. https://www.alibabacloud.com/help/en/ack/user-guide/

[63] Alibaba Cloud. Multi-Tenancy Security in ACK. Updated March 26, 2026. https://www.alibabacloud.com/help/en/ack/ack-managed-and-ack-dedicated/security-and-compliance/multi-tenancy-security

[64] Tencent Cloud. TKE Overview. https://www.tencentcloud.com/document/product/457/51208

[65] Tencent Cloud. General Cluster Overview. https://www.tencentcloud.com/document/product/457/6759

[66] Tencent Cloud. Node Pool Overview. Updated March 13, 2026. https://www.tencentcloud.com/document/product/457/35900

[67] Tencent Cloud. Deploying NGINX Ingress on TKE. https://www.tencentcloud.com/document/product/457/38072

[68] Tencent Cloud. CBS Instructions for TKE Persistent Volumes. https://www.tencentcloud.com/document/product/457/36157

[69] Tencent Cloud. TKE Billing Overview. https://www.tencentcloud.com/document/product/457/45157

[70] Volcengine. What Is Volcengine Kubernetes Engine (VKE)? Updated April 15, 2026. https://docs.volcengine.com/docs/6460/70611?lang=en

[71] Volcengine. VKE Cluster Overview. Updated July 24, 2026. https://docs.volcengine.com/docs/6460/1374026?lang=en

[72] Volcengine. Dependencies Between VKE and Other Cloud Services. https://docs.volcengine.com/docs/6460/100914?lang=en

[73] Volcengine. Creating APIG Ingresses by Using kubectl. https://docs.volcengine.com/docs/6460/1327566?lang=en

[74] Pan, Wuming. Cloud and Cloud AI, Lesson 7: CaaS Practice. Course slides, 2025.

[75] Pan, Wuming. Cloud and Cloud AI, Lesson 8: Workloads and Cloud Persistence. Course slides, 2025.

[76] Pan, Wuming. Cloud and Cloud AI, Lesson 9: Cloud Application Definition. Course slides, 2025.

Part III / Chapter 12

Continuous Delivery, GitOps, and Change Governance

How Immutable Artifacts, Versioned Desired State, Pull-Based Reconciliation, Policy Gates, Progressive Delivery, Telemetry, and Rollback Turn Change into a Governed Cloud Resource

Modern delivery is not a faster copy operation. It is a control-plane discipline that links intent, source, artifact identity, environment policy, reconciliation, runtime evidence, and recovery. GitOps strengthens that discipline by making desired state declarative, versioned, automatically pulled, and continuously reconciled. The result is a change system that can be reviewed before execution, observed during rollout, and reversed or repaired with evidence.

In This Chapter

- Treat software delivery as a governed feedback loop rather than a sequence of build and deployment scripts.
- Distinguish continuous integration, continuous delivery, continuous deployment, release, and GitOps by the state each practice produces and the decision it automates.
- Build an immutable artifact once and promote the same digest through environments by changing versioned desired state.
- Model an environment as a policy, identity, data, SLO, quota, approval, and evidence boundary—not merely a branch name or namespace.
- Apply the OpenGitOps principles of declarative desired state, versioned and immutable history, automatic pull, and continuous reconciliation.
- Compare Flux and Argo CD as Kubernetes delivery control planes, and distinguish managed GitOps services from pipeline-driven cluster deployment.
- Use pull requests, CODEOWNERS, rulesets, signed commits, required checks, environment approvals, and exception records as change-governance controls.

- Separate developer, build, signer, release, GitOps controller, and workload identities; use workload federation instead of long-lived cloud secrets where supported.
- Reason about drift, server-side field ownership, dynamic controllers, generated fields, and reconciliation conflicts.
- Choose rolling, blue-green, canary, experiment, and progressive-delivery strategies according to failure cost, observability, and rollback constraints.
- Distinguish runtime undo, source revert, and forward fix, especially when data schemas or irreversible side effects make a simple rollback unsafe.
- Create a release evidence packet that connects source, artifact, configuration, approval, deployed revision, telemetry, cost, and outcome.
- Extend change governance to models, prompts, retrieval indexes, tools, safety policies, evaluations, and agent definitions.
- Complete an independent AKS and Flux lab in which a release and rollback are performed by changing only the desired state in Git.

Cloud platforms make infrastructure and application state programmable, but programmability alone does not make change safe. A portal click, a pipeline job, a Git commit, a Kubernetes rollout, and an agent-generated change can all modify production. The engineering question is therefore not simply how to automate deployment. It is how to make every consequential change attributable, reviewable, testable, policy-compliant, observable, reversible, and economically accountable.

Traditional deployment procedures often mixed several responsibilities inside one privileged script: building code, selecting a version, authenticating to the cluster, modifying runtime objects, waiting for health, and deciding whether success was acceptable. This design is fast to demonstrate and difficult to govern. The script identity accumulates broad permissions, runtime state becomes the only record of what happened, and rollback depends on whoever still remembers the previous command.

Continuous delivery separates those responsibilities. Continuous integration produces tested, immutable artifacts. A promotion decision changes versioned desired state. A reconciler converges the environment toward that state. Runtime telemetry and business evidence determine whether the change should continue, stop, revert, or be repaired. GitOps is a particularly strong form of this model for declarative systems because the desired state is versioned and a controller continuously compares it with actual state.

Central claim. Continuous delivery is a control-plane discipline. Every change should have a source, immutable subject, review and policy evidence, environment-specific authorization, rollout strategy, telemetry, and recovery path. GitOps strengthens this contract by making desired state declarative, versioned, automatically pulled, and continuously reconciled.

12.1 Change Is a Managed Resource Graph

A release is often represented by a version string such as 3.7.2. That label is useful to humans but insufficient for governance. A production state is composed from multiple subjects: source code, build instructions, dependencies, container images or packages, infrastructure definitions, application configuration, secrets or secret references, policy versions, database schema, model deployment, retrieval index, and runtime controller state. The release is therefore a graph, not a file.

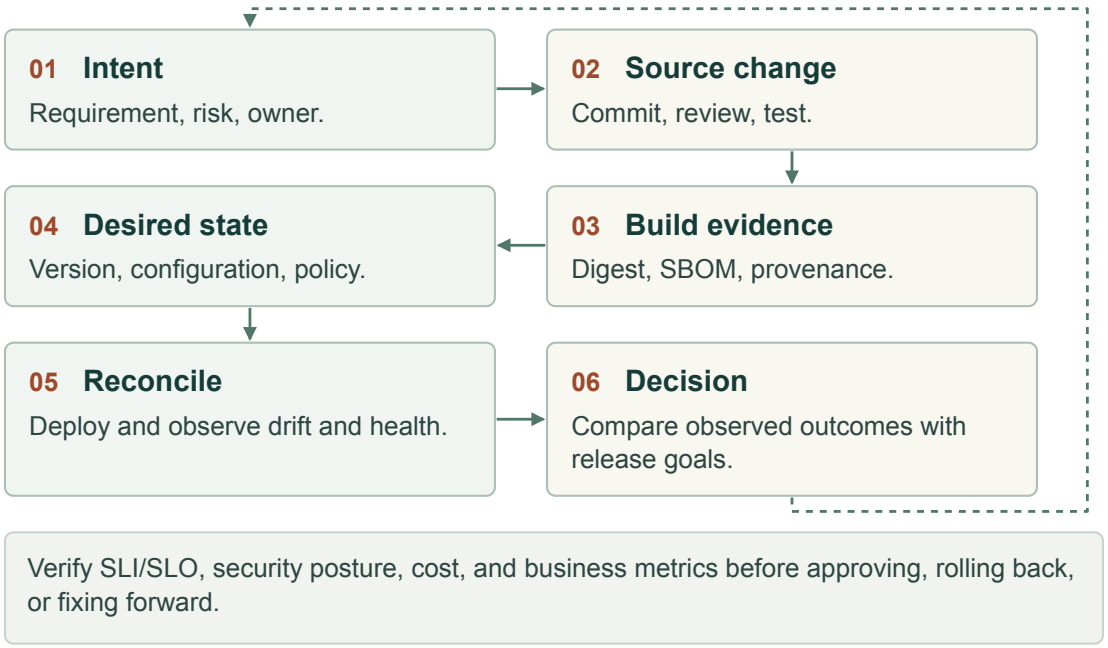


Figure 12-1 Continuous delivery is a managed feedback loop from intent to verified outcome.

Table 12-1 A governed change graph separates subjects that are often collapsed into one "deployment."

Subject	Stable identifier	Control question	Evidence
Change intent	Issue, requirement, risk record, or approved plan.	Why is the change needed, who owns it, and what outcome is expected?	Owner, scope, risk class, acceptance criteria, expiry.
Source revision	Commit SHA or signed source archive digest.	Which code and tests produced the candidate?	Repository, author, review, signature, timestamp.

Subject	Stable identifier	Control question	Evidence
Build execution	Workflow or pipeline run ID.	Who or what built the artifact, in which isolated environment?	Builder identity, inputs, logs, test results, provenance.
Artifact	Package or image digest.	What immutable bytes are being promoted?	Digest, SBOM, signature, vulnerability and policy results.
Desired state	Configuration commit, signed OCI config digest, or release manifest.	Which artifact, settings, and resources should the environment run?	Diff, policy result, approver, environment target.
Reconciliation	Controller source revision and apply status.	Did the runtime converge on the declared state?	GitRepository/Kustomization or Application status, events, errors.
Runtime state	Deployment revision, Pod imageID, model/index version, resource graph.	What is actually running and serving traffic?	Rollout conditions, object inventory, telemetry, audit logs.
Outcome	Evaluation window and decision ID.	Did the change improve or preserve reliability, security, cost, and business value?	SLI/SLO, error budget, cost, security findings, KPI, approval or rollback.

This graph explains why “the pipeline succeeded” is weak evidence. A pipeline can complete while the cluster rejects the manifest, the new Pods fail readiness, the artifact digest differs from the approved subject, the database migration is incompatible, or the service meets technical health checks but violates a business KPI. A governed release record must reach beyond automation status to actual runtime and outcome evidence.

The graph also clarifies the control-plane boundary. Source repositories, artifact registries, deployment controllers, policy engines, environment approvals, observability systems, and cost-management services are separate resource systems. A delivery architecture composes them through stable identifiers and explicit trust relationships. When any edge is implicit—such as “latest image,” a shared administrator token, or an unrecorded manual hotfix—the evidence chain is broken.

12.2 Continuous Integration, Continuous Delivery, Continuous Deployment, and Release

The language of delivery is frequently blurred. Precision matters because each term implies a different automation boundary and human decision. Continuous integration is primarily about the source-to-artifact path. Continuous delivery keeps software in a deploy-

able state and automates the path to a release candidate while retaining an explicit release decision where required. Continuous deployment extends automation so that a qualifying change is released without a routine human gate. A release is the act of making a version available to a defined audience or workload, which may be separate from deployment.

Table 12-2 Delivery terms describe different state transitions.

Term	Primary state transition	Typical automated evidence	Human decision
Continuous integration	Source revision → tested build candidate.	Compile, unit/integration tests, static analysis, dependency checks.	Merge approval and failed-check remediation.
Continuous delivery	Approved candidate → releasable artifact and environment change plan.	Artifact publication, deployment plan, policy checks, staged rollout readiness.	Production release or environment approval may remain explicit.
Continuous deployment	Qualifying desired-state change → production rollout automatically.	All continuous-delivery evidence plus automated promotion and verification.	Humans define policy and exceptions rather than approve every routine release.
Deployment	Artifact/configuration is placed into a runtime environment.	Controller status, resource changes, rollout conditions.	May be automatic, scheduled, or manually initiated.
Release	A deployed capability becomes available to users or callers.	Traffic rule, feature flag, API/model deployment activation, audience assignment.	Product, risk, operations, or automated policy decision.
GitOps	Versioned declarative desired state → continuously reconciled actual state.	Source revision, reconciliation status, drift correction, controller events.	Change approval occurs through the source-of-truth workflow and policy.

Deployment and release can be decoupled. A new binary may be deployed behind a disabled feature flag, a new model deployment may receive no traffic, or a blue environment may be warmed before a routing switch. This separation reduces change radius, but it also creates more resources that must be governed: flags, traffic rules, audience policy, model routing, and retirement of dormant revisions.

Continuous deployment is not “continuous change without control.” It replaces routine manual judgement with explicit automated controls. If the evidence is weak, the policy is permissive, or rollback is unsafe, adding automation increases risk. The maturity question is not how frequently a team can trigger deployment, but whether the system can make trustworthy decisions at that frequency.

12.3 Build Once and Promote the Same Artifact

A sound delivery system builds the artifact once, identifies it immutably, and promotes that same subject through environments. Rebuilding per environment creates different bytes from the same source and weakens traceability. It also allows production to depend on a later package resolution, a changed base image, or a compromised build runner that was not involved in test.

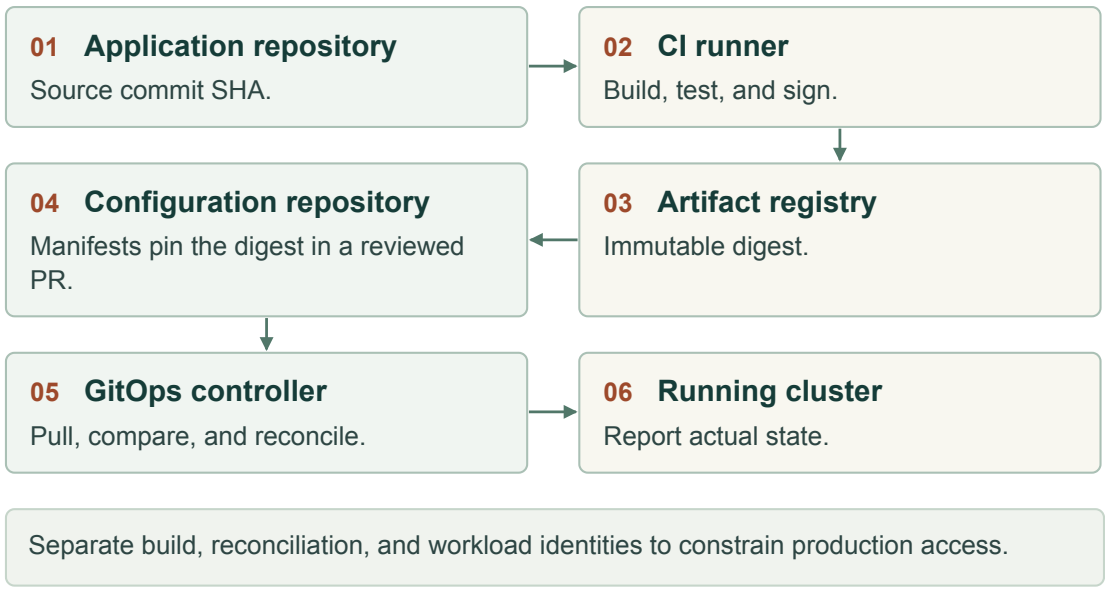


Figure 12-2 CI produces an immutable artifact; GitOps promotes it through versioned desired state.

Promotion should modify environment intent, not artifact content. The configuration change may update an image digest, Helm chart version, Kustomize patch, model deployment ID, or infrastructure module version. The pull request should display the precise old and new subjects. A tag can remain as a convenient label, but the deployable contract should resolve to a digest or another immutable identifier.

Table 12-3 Release identity should be stronger than a human-readable version.

Field	Purpose	Example
Source revision	Links the release to source and review history.	Git commit SHA.
Build run	Identifies builder, workflow, time, and logs.	GitHub Actions run ID or Azure Pipeline run ID.

Field	Purpose	Example
Artifact subject	Names the immutable package or image.	OCI digest sha256:... rather than only :1.4.0.
SBOM/provenance	Describes components and how the artifact was produced.	SPDX/CycloneDX SBOM and in-toto/SLSA provenance.
Configuration revision	Names the desired-state commit or OCI artifact.	Git SHA for clusters/prod/kustomization.yaml.
Environment	Binds the decision to a target policy and audience.	prod-eu, finance-staging, or tenant group.
Deployed identity	Proves the runtime resolved the intended subject.	Kubernetes Pod imageID digest or model deployment version.
Decision window	Defines when telemetry was evaluated.	2026-08-26T10:00Z through 10:30Z.

A release is an evidence graph

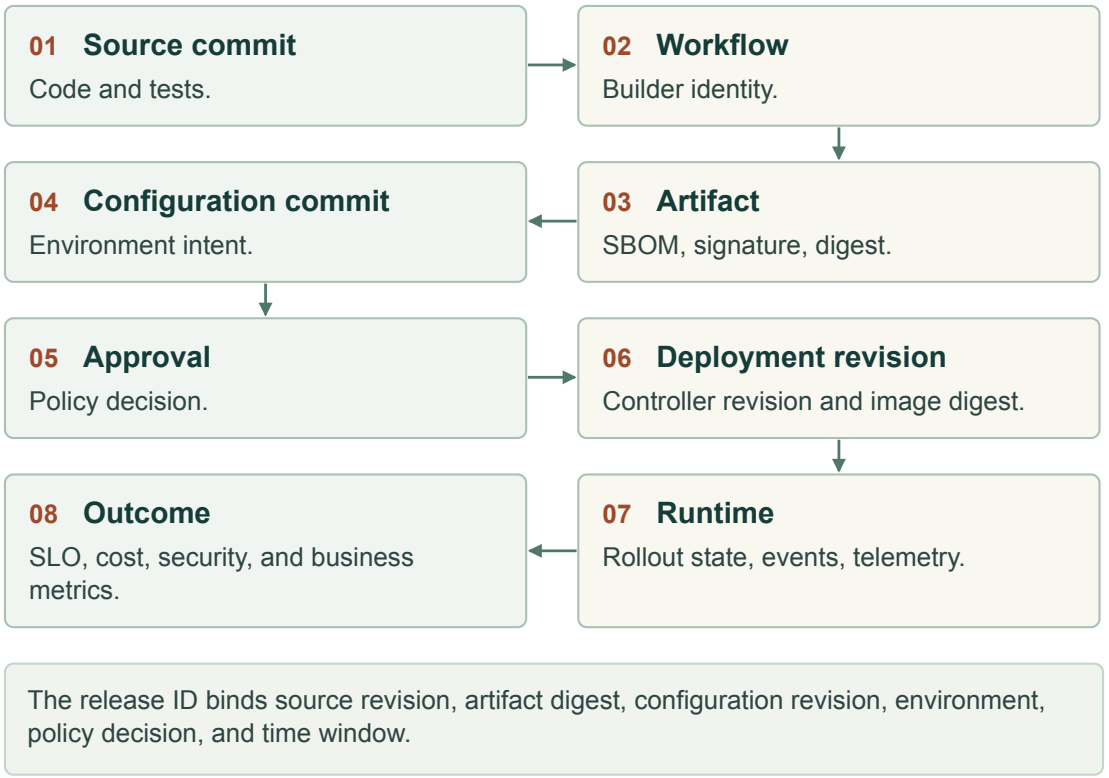


Figure 12-3 A release is an evidence graph that ends in a measured outcome.

The same identity model applies to AI-native applications. A model name is not a deployment identity; a retrieval index alias is not an immutable index build; a prompt label is not a prompt revision; and an agent version is not the complete set of model, knowledge, tool, and safety dependencies. Chapter 17 through Chapter 20 define these resources individually. Delivery governance must bind them into a release ledger.

12.4 Environments Are Policy Boundaries

An environment is often implemented as a resource group, cloud account, subscription, Kubernetes cluster, namespace, or project. None of those containers alone defines the environment contract. A production environment is the combination of identity, network, data, quota, policy, approval, SLO, observability, retention, and cost-accountability rules that apply to a target state.

Table 12-4 Environment design should make policy differences explicit.

Dimension	Development	Pre-production	Production
Purpose	Fast feedback and local integration.	Release-candidate validation under production-like constraints.	Serve committed users and business outcomes.
Identity	Developer self-service within bounded scope.	Pipeline/reconciler identity with test operator access.	Dedicated reconciliation and workload identities; break-glass separated.
Data	Synthetic or minimized data.	Representative, masked, or controlled data.	Authoritative data with retention, privacy, backup, and recovery rules.
Network	Broader debugging access may be allowed.	Production-like private paths and ingress policy.	Approved exposure, egress control, private dependencies, monitored paths.
Change gate	Automated tests and owner review.	Release review, policy checks, evaluation suite.	Required approval or automated policy based on risk class.
SLO/evidence	Engineering diagnostics.	Release qualification and load/evaluation evidence.	User-facing SLO, error budget, security, cost, business KPI.
Economics	Low-cost elastic or ephemeral resources.	Representative scale for tests; scheduled lifetime.	Capacity plan, commitments, anomaly control, cost ownership.

Branches are not environments. A branch can represent a stream of intent, but it does not create isolated credentials, network policy, data controls, quotas, or SLOs. Likewise, a namespace is not automatically an environment: cluster-wide controllers, shared nodes, shared gateways, shared secrets, and provider resources can cross namespace boundaries. The environment must be defined by enforceable controls and evidence.

Promotion should preserve artifact identity while applying an environment-specific configuration contract. Configuration may legitimately differ by environment, but the differences should be explicit and reviewable. Secret values should not be stored in Git; the repository should contain references, identities, and policy that cause the correct secret to be obtained at runtime. Environment overlays should remain small enough that a reviewer can understand the effective state rather than mentally executing a maze of patches.

12.5 GitOps Principles and the Reconciliation Contract

OpenGitOps defines four principles: the desired state is declarative; it is versioned and immutable; software agents pull it automatically; and agents continuously reconcile actual state toward the desired state. These principles are more important than any specific repository layout or controller product. A team can use GitOps poorly with a fashionable tool, or apply the principles well with a modest controller and disciplined source-of-truth workflow.

Table 12-5 The OpenGitOps principles create specific engineering obligations.

Principle	Engineering obligation	Failure if omitted
Declarative	State what should exist, not a one-time sequence of imperative commands.	The system cannot reliably detect or repair drift.
Versioned and immutable	Retain complete history and stable revision identity.	Auditors and operators cannot reproduce or explain prior state.
Pulled automatically	An agent retrieves approved desired state from the source.	External deployers need broad runtime credentials and become availability dependencies.
Continuously reconciled	Compare desired and actual state and act repeatedly.	Manual changes and partial failures persist unnoticed.

Git is the most common source because it provides review, history, signatures, branching, and familiar collaboration. The deeper contract is versioned desired state. Flux can also reconcile desired state from OCI artifacts and other sources. An OCI-based configuration

source can reduce the Git server as a runtime dependency and can place manifests, signatures, SBOMs, and images in a common content-addressed distribution system. Teams should still keep a human-governed authoring history even when the production source is a signed OCI artifact.

GitOps does not mean every byte belongs in Git. Secrets, large model files, training data, database contents, runtime metrics, and generated status should live in systems designed for them. Git stores the declarations, references, policies, and versioned metadata that allow controllers to obtain or produce those subjects securely. The distinction is the same one used throughout this book: desired state belongs to the control plane; operational payload belongs to an appropriate data plane.

Pull-based reconciliation changes the security boundary

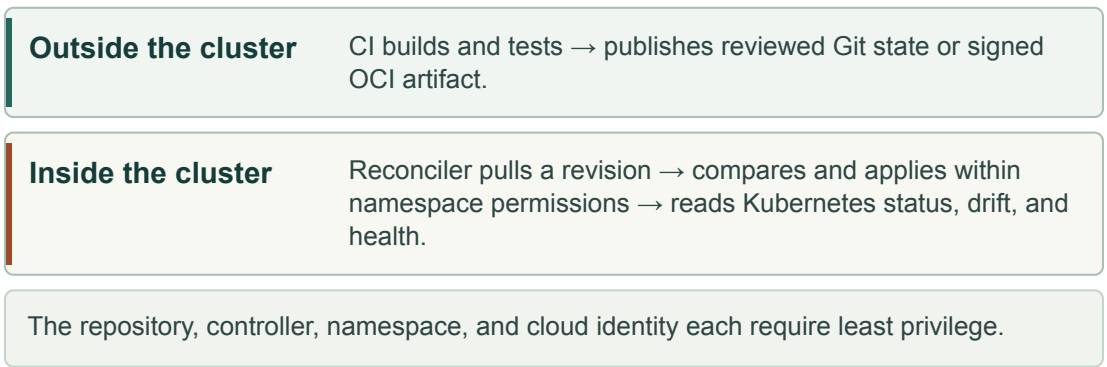


Figure 12-4 Pull-based GitOps relocates deployment authority to a cluster-side reconciler.

The pull model is a security improvement only when authority is narrowed. A cluster-wide reconciler that can read every repository, decrypt every secret, and modify every namespace remains a high-value principal. Repository credentials, controller service accounts, source allowlists, namespace boundaries, network egress, admission policy, and multitenancy controls require explicit design. Pull-based delivery removes one dangerous credential path; it does not remove identity engineering.

12.6 Flux and Argo CD as Delivery Control Planes

Flux and Argo CD are both graduated CNCF projects and both implement declarative, pull-based reconciliation for Kubernetes. Their object models and operating experiences differ. The correct comparison is not which UI looks better, but how each system represents sources, applications, dependencies, health, policy, multitenancy, and evidence.

Table 12-6 Flux and Argo CD expose different but comparable delivery abstractions.

Concern	Flux	Argo CD
Source	GitRepository, OCIRepository, Bucket, HelmRepository and produced artifacts.	Repository or chart source referenced by an Application/ApplicationSet.
Apply unit	Kustomization and HelmRelease resources, with dependencies and health checks.	Application resource with source, destination, sync policy, health and history.
Reconciliation style	Composable controllers for source, Kustomize, Helm, notifications and image automation.	Application-centric controller and API/UI with automated or manual synchronization.
Multicluster	One or more controllers/configurations can target local or remote clusters according to design.	ApplicationSet and cluster registrations support fleet patterns.
Drift/self-heal	Kustomize and Helm controllers detect and correct drift according to configuration.	Automated sync can prune and self-heal; sync windows and resource customizations govern behavior.
Progressive delivery	Integrates with Flagger or other rollout controllers.	Integrates naturally with Argo Rollouts for canary/blue-green analysis.
Operational interface	Kubernetes CRDs, Flux CLI, events, metrics and notifications.	Kubernetes CRDs plus CLI, API and a rich web UI.

Azure exposes Flux as a managed cluster extension and represents each configuration as an Azure resource. The microsoft.flux extension installs source, Kustomize, Helm, and notification controllers. Microsoft.KubernetesConfiguration/fluxConfigurations resources describe source and reconciliation parameters, and status is propagated back to Azure. This composition is important: Kubernetes CRDs perform reconciliation, while Azure Resource Manager provides lifecycle, identity, policy-at-scale, inventory, and activity evidence for the extension and configuration resources.

Alibaba Cloud ACK One Fleet hosts managed Argo CD with multi-cluster capabilities. It automates installation, upgrades, and scaling and integrates Alibaba Cloud SSO and user permissions. This is a managed GitOps control plane rather than merely a pipeline step that runs kubectl. Tencent Cloud CODING Continuous Deployment and Volcengine CodePipeline provide strong pipeline-driven deployment into TKE/VKE and Kubernetes, including staged strategies and approvals. A strict pull-based GitOps design on TKE or VKE can be implemented with self-managed Flux or Argo CD. The same YAML may reach the cluster in each case, but the credential path, reconciliation behavior, drift correction, and evidence model are different.

Table 12-7 Managed GitOps and pipeline-driven deployment are not operationally equivalent.

Dimension	Managed pull-based GitOps	Pipeline-driven push deployment
Runtime authority	Cluster-side controller holds bounded apply authority.	External pipeline identity authenticates to cluster or deployment endpoint.
Source of truth	Versioned desired state is continuously reconciled.	Pipeline run applies a snapshot; subsequent drift may persist unless separately detected.
Cluster exposure	Controller initiates outbound source access; build runners need no cluster endpoint.	Pipeline must reach the cluster API or provider deployment service.
Failure recovery	Controller retries and converges after transient failure.	Pipeline may need rerun; state after partial failure depends on tooling.
Drift	A first-class observed difference that can be corrected.	Usually detected by separate tooling or a later deployment.
Evidence	Source revision, reconciliation conditions and controller events.	Pipeline logs, deployment task results and cluster status.

12.7 Change Governance in Git

Git provides history, not governance by itself. A repository that allows direct force pushes to the production branch is versioned but weakly controlled. Change governance adds identity, review, verification, decision authority, and exception handling around the source-of-truth path.

Change governance is a sequence of evidence-producing gates



Figure 12-5 Change governance is a sequence of evidence-producing gates.

Table 12-8 Repository controls map to distinct governance objectives.

Control	Objective	Evidence
Pull request requirement	Prevent unreviewed direct changes to protected desired state.	PR number, author, diff, linked issue.
Required approvals	Apply separation of duties and risk-based review.	Reviewer identities and approval timestamps.
CODEOWNERS or path reviewers	Route sensitive directories to accountable teams.	Approval from platform, security, data or model owner.
Required status checks	Block merge until tests, policy, manifest render, signature or evaluation passes.	Named check result tied to commit SHA.
Dismiss stale approvals	Ensure the approved diff is the merged diff.	Review invalidation and renewed approval after change.
Signed commits/tags	Strengthen author and release identity where the trust model supports it.	Verification status and signing key/certificate.

Control	Objective	Evidence
Linear history or constrained merge	Simplify provenance and automated policy evaluation.	Protected-branch rule and resulting commit graph.
Environment approval	Authorize a production-specific decision after merge or promotion request.	Approver, environment, artifact/config revision, expiry.
Exception record	Permit time-bounded, owned deviation with compensating controls.	Reason, risk owner, scope, expiration and remediation plan.

Review quality depends on the diff. A manifest change from image tag latest to latest reveals nothing. A change from one immutable digest to another, accompanied by source revision, SBOM, vulnerability delta, rollout plan, and evaluation result, gives reviewers a concrete subject. Generated manifests should be rendered in CI so reviewers see the effective state, while the generator inputs remain the authoritative source.

The number of approvals should follow risk, not ceremony. A read-only observability dashboard change and a production database migration should not share identical gates. Risk classification can consider blast radius, data sensitivity, reversibility, user impact, novelty, and automated evidence strength. High-risk changes may require platform, security, data, or model-owner review and a defined rollback or forward-fix rehearsal.

Agents and code-generation systems make attribution more important. The responsible human or service owner should remain visible even when an AI system creates the patch. Repository rules can require additional review for unattributed automated pull requests. The governance principle is simple: automation may prepare a change, but authority to merge or release must be assigned explicitly and auditable.

12.8 Identity, Secrets, and Separation of Duties

Delivery systems connect several high-value trust domains. Using one service principal or kubeconfig for all steps makes the architecture easy to diagram and hard to defend. The identities should be separated according to the operation each performs.

Table 12-9 A delivery identity chain should keep responsibilities narrow.

Principal	Required authority	Authority it should not have
Developer	Create branches and pull requests; read non-sensitive release evidence.	Direct production-cluster mutation or production secret read.

Principal	Required authority	Authority it should not have
CI builder	Read source, obtain dependencies, run tests, publish candidate artifact and evidence.	Production deployment authority or broad repository administration.
Artifact signer/attestor	Sign a verified subject under protected workflow conditions.	Arbitrary source modification or runtime administrator access.
Configuration updater	Propose or merge the approved artifact identity into an environment path.	Rebuild or replace the artifact bytes.
GitOps reconciler	Read approved desired state and apply bounded resources to target scope.	Repository write, unrelated namespace, or unrestricted cloud subscription access.
Release approver	Approve a specific environment, subject, and time window.	Silent modification of the candidate after approval.
Workload identity	Access runtime data-plane dependencies required by the application.	Repository, CI, or GitOps administration.
Break-glass operator	Time-bounded emergency intervention with enhanced logging.	Persistent routine use or unreviewed credential sharing.

GitHub Actions and other modern CI systems can use OpenID Connect federation to exchange a workflow identity for a short-lived cloud token. This avoids storing long-lived cloud client secrets in the repository. The cloud trust policy should constrain repository, branch or environment, workflow, audience, and subject claims, and the resulting cloud role should remain least-privileged. OIDC removes a stored secret; it does not justify broad authorization.

Pull-based GitOps further separates identity. The CI runner publishes an artifact and proposes desired state. The in-cluster controller applies state. This prevents the CI system from becoming a universal cluster administrator and allows cluster access to remain private. However, the reconciler may need source credentials, decryption capability, registry access, or cloud identity. Those credentials require rotation, egress control, audit, and namespace isolation.

Secrets should enter the runtime through references and workload identity, not through rendered plaintext manifests or CI logs. Where secret synchronization controllers are used, their source, target namespace, refresh interval, deletion semantics, and controller authority become part of the delivery contract. A successful sync is not proof that the workload has the correct least-privilege data-plane permission.

12.9 Drift, Field Ownership, and Dynamic Controllers

Continuous reconciliation makes drift visible and correctable, but not every runtime difference is unauthorized drift. Kubernetes controllers legitimately modify status, allocate addresses, inject defaults, update replica counts, rotate certificates, and attach provider resources. A GitOps system must distinguish declared fields from fields owned by other controllers or generated by the platform.

Table 12-10 Drift categories require different responses.

Drift type	Example	Correct response
Unauthorized imperative drift	An operator edits an image, label, or security context directly.	Reconcile to source, alert, and investigate the bypass.
Controller-owned dynamic field	HPA changes replicas or a service controller assigns an address.	Exclude or ignore the field; do not make controllers fight.
Defaulted or mutated field	Admission adds defaults, sidecars, labels, or security settings.	Render/understand effective state and configure diff customization if stable.
Generated secret or certificate	Controller creates rotating credentials.	Store generator intent, not ephemeral value, and govern the generator.
Emergency runtime change	Break-glass mitigation during incident.	Record, time-bound, then commit authoritative source or allow reconciliation to remove it.
External dependency drift	DNS, database, model alias, or cloud policy changes outside the repository.	Correlate provider evidence and decide whether the desired-state model must expand.

Field ownership should be explicit. If HPA owns replicas, the Git repository should not repeatedly force a fixed replica count unless the architecture intentionally disables HPA. If a policy controller injects labels, the GitOps diff should understand that mutation. If a database operator manages child resources, application teams should declare the operator custom resource rather than every generated StatefulSet. The source of truth is not necessarily one file; it is the set of authoritative declarations and controller contracts.

Suspension is another governed state. Flux and Argo CD can pause reconciliation or automatic sync. Suspension may be necessary during incident response, migration, or debugging, but it disables the very mechanism that limits drift. The action should be time-bounded, authorized, observable, and followed by a reconciliation review. A permanently suspended GitOps object is an unmanaged environment with misleading labels.

12.10 Rollout Strategies and Progressive Delivery

A deployment controller can replace replicas gradually, but a rolling update is not automatically a safe release strategy. The correct strategy depends on capacity, compatibility, failure detection, traffic control, data state, and the time required to obtain trustworthy evidence.

Table 12-11 Rollout strategies trade capacity, control, and evidence differently.

Strategy	Mechanism	Strength	Primary risk
Recreate	Stop old revision, then start new revision.	Simple and suitable for exclusive or incompatible workloads.	Intentional downtime and no live comparison.
Rolling update	Gradually replace old replicas within surge/unavailable limits.	Native, economical, and appropriate for compatible stateless services.	Mixed versions and limited traffic/evidence control.
Blue-green	Prepare a complete parallel environment and switch traffic.	Fast cutover and rapid traffic reversal.	Double capacity, state compatibility, stale background work.
Canary	Expose a small audience or traffic percentage, then increase.	Limits blast radius and supports evidence-based decisions.	Requires reliable traffic control, segmentation, and analysis.
Experiment/A-B	Route defined cohorts to alternatives to compare outcomes.	Measures business or model behavior, not only technical health.	Statistical bias, user-consistency and data-governance complexity.
Shadow	Copy traffic to a candidate without serving its response.	Validates load and behavior with low user impact.	Duplicate side effects and privacy/cost risk unless requests are sanitized.

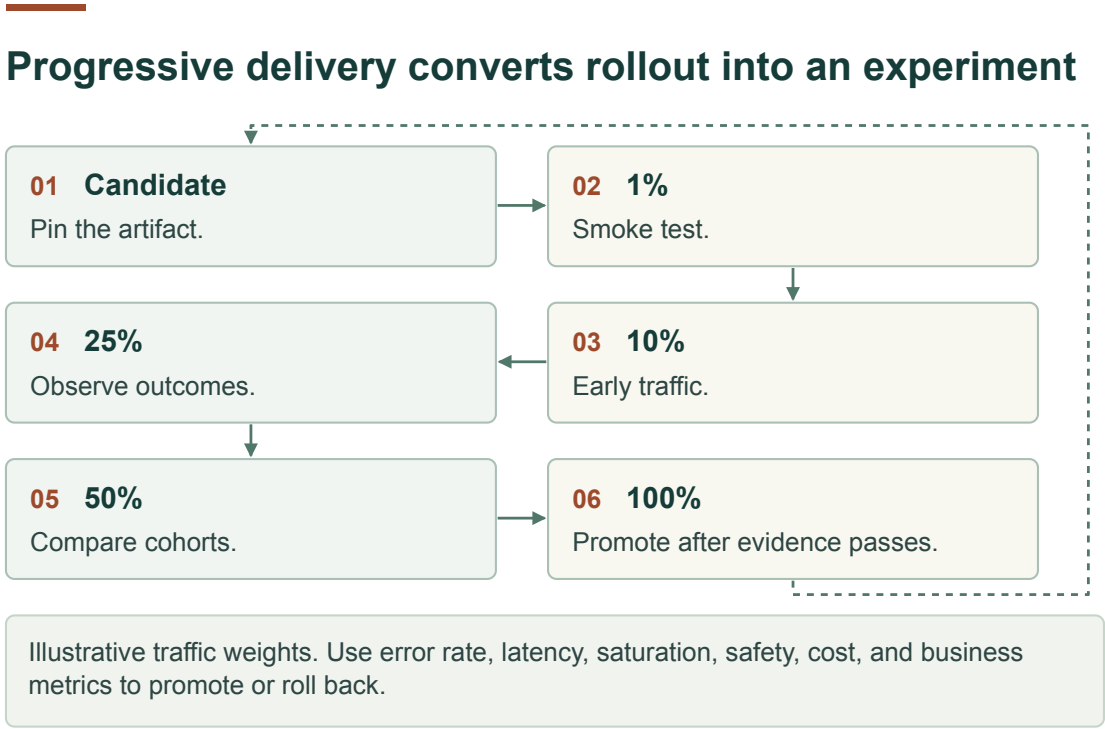


Figure 12-6 Progressive delivery advances traffic only while evidence remains within policy.

Argo Rollouts provides Kubernetes custom resources for blue-green and canary strategies, weighted traffic shifting, manual judgement, metric analysis, automated promotion, and rollback. Similar behavior can be implemented with Flagger or cloud-native deployment services. The product is less important than the contract: every promotion step should identify the candidate, traffic cohort, measurement window, thresholds, decision authority, and fallback.

Metrics must reflect user experience. CPU utilization can indicate saturation but rarely proves that a release is safe. A progressive policy should combine availability, latency, error rate, dependency health, security signals, cost, and business or model quality where relevant. For an AI application, the analysis may include groundedness, retrieval precision, safety-policy violations, tool-call success, token cost per outcome, and human escalation rate.

A rollout controller should not hide uncertainty behind automatic rollback. Sparse traffic, delayed failures, caching, asynchronous jobs, or seasonal behavior can make the measurement window inconclusive. “No threshold crossed” is not the same as “candidate is better.” The policy should allow inconclusive outcomes that pause promotion and request human judgement.

12.11 Rollback, Forward Fix, and State Compatibility

Rollback is often presented as the universal safety mechanism of continuous delivery. It is only safe when the previous version remains compatible with current state and external effects. Stateless code can often be reverted quickly. Database schema changes, message formats, data migrations, model-output persistence, index rewrites, and irreversible external actions may require a forward fix or a staged compatibility plan.

Rollback has three different meanings

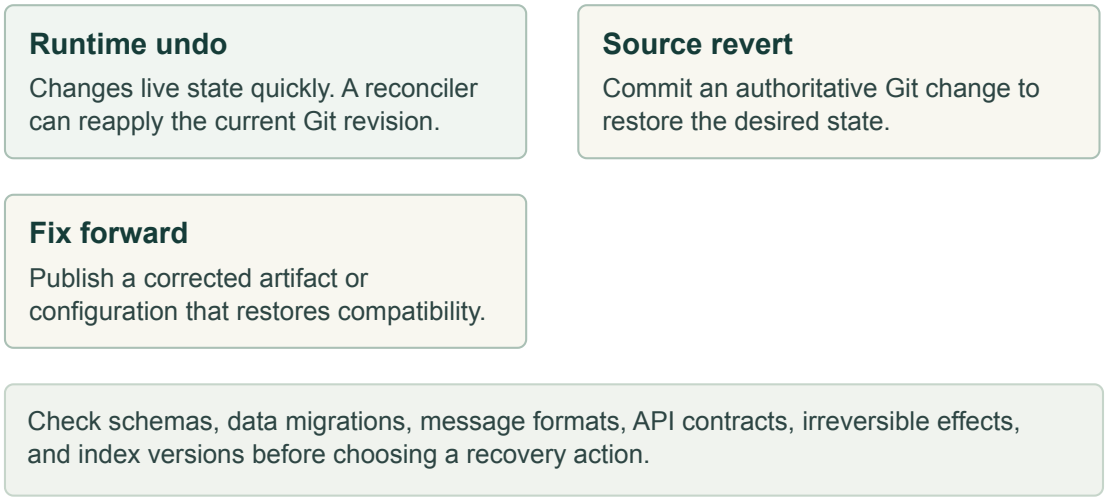


Figure 12-7 Runtime undo, source revert, and forward fix are different recovery operations.

Table 12-12 Recovery choice depends on authority and compatibility.

Situation	Preferred action	Why
Manifest or image regression; old version compatible.	Revert the desired-state commit and let the controller reconcile.	Restores authoritative history and runtime together.
Immediate severe incident before repository workflow can complete.	Use bounded runtime undo or traffic switch, then update source immediately.	Reduces harm quickly but avoids permanent unmanaged drift.
Backward-incompatible schema already migrated.	Forward fix or expand/contract migration plan.	Old code may corrupt or fail against the new schema.

Situation	Preferred action	Why
Bad configuration with no artifact defect.	Revert configuration commit only.	Preserves valid artifact identity and narrows change.
Canary violates threshold before full promotion.	Abort candidate and restore traffic to stable revision.	Limits blast radius; no full environment rollback required.
Security compromise or untrusted artifact.	Block subject, revoke credentials, quarantine, deploy trusted replacement, investigate evidence.	A normal rollback may reintroduce another vulnerable subject or lose forensic state.

Kubernetes provides kubectl rollout undo for supported workload controllers. In a GitOps-managed environment, that command changes runtime state but not the authoritative source. A continuously reconciling controller can reapply the current Git revision. Runtime undo is therefore an emergency mitigation, not the final GitOps rollback. The durable action is a revert or new commit that expresses the intended recovered state.

Argo CD adds another nuance: automated sync and rollback interact. Current Argo CD documentation notes that rollback cannot be performed against an application while automated sync is enabled. Teams should define the operational procedure in advance—revert Git, disable automation under controlled authority, or use progressive-delivery traffic control—rather than discover the constraint during an incident.

Forward compatibility should be designed into migrations. Expand/contract database changes, tolerant readers, versioned messages, dual-write transitions, model routing aliases, and index aliases let old and new components coexist. These techniques turn rollback from wishful thinking into an executable contract.

12.12 Telemetry, Evidence, and Delivery Performance

A delivery system needs both release evidence and long-term performance evidence. Release evidence answers what changed, who authorized it, what was deployed, and whether the immediate result satisfied policy. Performance evidence answers whether the delivery system improves speed, stability, recovery, and rework over time.

Table 12-13 A minimum release evidence packet connects intent to outcome.

Evidence group	Required records
Intent and source	Issue/change ID, owner, risk class, source commit, pull request, reviewers.

Evidence group	Required records
Build and artifact	Workflow run, test results, artifact digest, SBOM, provenance, signatures, scan delta.
Desired state	Configuration commit/digest, rendered diff, environment, policy checks, approval.
Reconciliation	Controller source revision, health conditions, apply/prune results, controller events.
Runtime	Deployment/rollout revision, Pod imageID or service/model version, resource graph, traffic split.
Outcome	SLI/SLO window, error budget effect, security events, cost delta, business or AI-quality KPI.
Recovery	Rollback/forward-fix plan, execution record, final authoritative revision, incident linkage.
Retention and integrity	Storage location, access policy, hash/signature, retention period, deletion authority.

The current DORA software delivery performance model uses five metrics. Throughput is represented by change lead time, deployment frequency, and failed deployment recovery time. Instability is represented by change fail rate and deployment rework rate. These metrics should be applied at the application or service level, interpreted in context, and used for improvement rather than competition between unlike teams.

Table 12-14 Delivery metrics need contextual interpretation.

Metric	Question	Common misuse
Change lead time	How long does an accepted change take to reach the defined production state?	Starting/ending the clock differently across teams.
Deployment frequency	How often does this service deploy to production?	Treating more deployments as valuable without outcome or quality context.
Failed deployment recovery time	How quickly is service restored after a deployment failure?	Measuring incident closure rather than restored user outcome.
Change fail rate	What proportion of deployments require remediation because they degraded service?	Hiding failures with inconsistent classification.
Deployment rework rate	How much deployment activity is spent correcting earlier deployment work?	Counting useful iterative improvement as failure, or ignoring invisible manual rework.

Delivery telemetry should be correlated with the release evidence graph. Trace and log attributes can include release ID, source revision, artifact digest, configuration revision, environment, rollout step, and feature or model version. High-cardinality identifiers belong in traces or logs, while derived metrics support trend and SLO decisions. Chapter 13 defines the managed feedback loop that consumes these signals; this chapter defines how release identity enters it.

Cost is part of release evidence. A new version can be reliable and economically unacceptable because it increases CPU per request, model token usage, retrieval calls, logging volume, or cross-region transfer. The FinOps control plane from Chapter 14 should compare normalized cost per successful outcome before and after the change. A rollback threshold may therefore be economic as well as technical.

12.13 AI-Native Change Governance

AI-native systems broaden the release subject. An application release may change code without changing the model, change the model without changing code, rebuild an index, update prompt instructions, grant a new tool, alter a safety threshold, or modify an agent workflow. Each change can affect behavior and cost independently. A single application version is no longer enough to explain production behavior.

Table 12-15 AI-native subjects require specialized gates and evidence.

Subject	Version/evidence	Release gate
Model deployment	Provider, model/version, deployment ID, capacity mode, safety policy, evaluation result.	Quality, safety, latency, quota, retirement and cost thresholds.
Prompt/instructions	Immutable prompt revision, template variables, system-policy version.	Regression set, injection tests, style/behavior approval.
Knowledge/index	Source snapshot, chunking, embedding model, schema, index build, alias target.	Retrieval precision/recall, ACL filtering, citation integrity, freshness and cost.
Tool contract	Tool schema, implementation version, identity, scopes, risk class.	Least privilege, validation, dry run, approval, idempotency and audit.
Safety policy	Policy version, thresholds, exceptions, evaluator version.	Adversarial tests, false-positive/negative review, owner approval.
Agent definition	Model, prompt, memory, knowledge, tools, workflow, evaluation suite.	End-to-end task success, trace review, excessive-agency checks and outcome cost.

Subject	Version/evidence	Release gate
Evaluation suite	Dataset/snapshot, rubric, judge model/version, thresholds.	Dataset governance, reproducibility, leakage and statistical sufficiency.

Promotion should use a composite release manifest that names each subject. This does not require one monolithic file, but it requires one queryable release identity. When an agent gives a wrong answer, investigators must be able to determine which model deployment, knowledge alias, prompt revision, tool contract, safety policy, and code artifact were active for that run. The trace and evaluation systems should record those identifiers.

AI changes may require slower or segmented rollouts. A new model can be canaried by tenant, task class, geography, or percentage of requests. A new retrieval index can be shadow-queried and compared before alias switch. A new tool can begin in read-only or dry-run mode. An agent policy can require human approval until sufficient evidence supports higher autonomy. Progressive delivery is therefore an autonomy-control mechanism as well as a traffic-control mechanism.

12.14 Multicloud Engineering Intent Mapping

Multicloud delivery patterns have different execution boundaries

Azure
AKS with managed Flux. In-cluster controllers reconcile.

Alibaba Cloud
ACK One with managed Argo CD supports multi-cluster GitOps.

Tencent Cloud
CODING CD deploys to TKE through a pipeline.

Volcengine
CodePipeline deploys to VKE. Identify the pipeline's deployment principal.

Shared intent covers versioned desired state, least privilege, review, rollout, telemetry, evidence, and rollback.

Figure 12-8 Similar deployment outcomes can be produced by non-equivalent multicloud delivery control planes.

Table 12-16 Delivery capabilities across the reference clouds.

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Code and CI	GitHub Actions or Azure Pipelines; OIDC/federated Azure login.	Alibaba Cloud DevOps/Codeup and pipelines.	CODING repositories and CI.	Code repository and CodePipeline.
Artifact registry	Azure Container Registry with OCI evidence.	Container Registry Enterprise Edition/Personal Edition.	Tencent Container Registry.	Container Registry.
Managed Kubernetes	AKS.	ACK.	TKE.	VKE.
Managed GitOps	microsoft.flux extension and Azure fluxConfigurations for AKS/Arc.	ACK One Fleet hosts managed Argo CD with multi-cluster capabilities.	Pipeline-driven CODING CD is documented; strict pull GitOps can be self-managed on TKE.	Pipeline-driven CodePipeline/VKE deployment is documented; strict pull GitOps can be self-managed on VKE.
Progressive delivery	Argo Rollouts/Flagger or Azure-native traffic and monitoring integrations.	ACK One GitOps with Argo Rollouts for canary and analysis.	CODING deployment strategies or self-managed rollout controllers on TKE.	Batch/canary pipeline tasks or self-managed rollout controllers on VKE.
Change governance	GitHub rules/environments, Azure DevOps approvals, Azure Policy, workload federation.	RAM/SSO, repository/pipeline approvals, ACK/Cloud Config governance.	CODING approvals and credentials, CAM, TKE policy and observability.	Pipeline approvals, IAM, VKE policy and observability.
Evidence	Git/Actions or Pipelines, ACR, Flux status in Azure, Kubernetes status, Azure Monitor.	Repository/pipeline, registry, managed Argo CD status, ACK and ARMS/SLS.	CODING delivery records, TCR, TKE state, CLS/monitoring.	Pipeline records, registry, VKE state, logs/monitoring.

The table is a map of engineering intent, not a claim of product equivalence. Azure and ACK One expose managed pull-based GitOps control planes. Tencent CODING and Volcengine CodePipeline emphasize pipeline-driven delivery in the referenced documentation. All four can operate Kubernetes workloads and all can host Flux or Argo CD, but the default trust path, management responsibility, multi-cluster model, and evidence surface differ.

A portable configuration format does not create portable operations. The same Kustomization can be reconciled by managed Flux, managed Argo CD, a self-managed controller, or applied by a pipeline. Teams should document where credentials live, who owns

controller upgrades, how drift is treated, how multitenancy is enforced, which logs prove reconciliation, and how the platform behaves when the source or provider control plane is unavailable.

12.15 Lab 12 — Release and Roll Back by Changing Only Desired State in Git

This independent lab proves the core GitOps contract on Azure Kubernetes Service. You will place a small Kubernetes application under versioned desired state, create an Azure-managed Flux configuration, release a new application version by changing the image reference in Git, introduce a controlled drift, and roll back through a Git revert. No kubectl apply, kubectl set image, or direct cluster mutation is used for the authoritative release or rollback.

Lab acceptance criterion. The lab is complete only when the evidence packet links Git commits, Flux source revision, Kubernetes rollout status, runtime image digest, application version response, drift correction, rollback commit, and cleanup. A green command alone is not sufficient.

12.15.1 Capability Goal and Boundaries

- **Primary capability.** Versioned desired state, pull-based reconciliation, drift correction, and source-authoritative rollback.
- **Azure resources.** An existing or ephemeral managed-identity AKS cluster and a Microsoft Flux extension/configuration.
- **Application.** Podinfo, a small Kubernetes reference service, deployed initially at 6.14.0 and updated to 6.14.1.
- **Repository.** A public Git repository for the lab. A private repository is valid but requires additional source-identity configuration.
- **Exposure.** ClusterIP plus kubectl port-forward; the lab does not create a public load balancer.
- **Cost boundary.** The AKS cluster and node are the principal cost. Reuse a permitted cluster or delete an ephemeral cluster immediately after the lab.
- **Out of scope.** Building the application image, signing artifacts, production secret delivery, multitenant cluster hardening, and advanced traffic splitting.

12.15.2 Prerequisites

- Azure CLI with the k8s-configuration and k8s-extension commands available, kubectl, Git, and an Azure subscription where you can manage an AKS cluster and extensions.

- A managed-identity AKS cluster. If you did not retain a cluster from another exercise, create a small ephemeral cluster in a dedicated resource group and record its cost window.
- A public Git repository you can push to. Protecting the main branch with a pull-request rule is recommended but not required for the mechanical exercise.
- Local variables for resource group, cluster, repository URL, branch, and Flux configuration name.

```
export RG="book-gitops-rg"
export AKS="book-gitops-aks"
export REPO_URL="https://github.com/YOUR-ACCOUNT/book-gitops.git"
export BRANCH="main"
export FLUX_CONFIG="book-gitops"
export APP_NS="book-gitops"
```

CLI syntax and extension versions evolve. Before execution, inspect az k8s-configuration flux create -h and the current Azure GitOps documentation. The configuration used here targets an AKS managedClusters resource.

12.15.3 Create the Desired-State Repository

Create this repository structure. The base contains the namespace, Deployment, and ClusterIP Service. The cluster path composes the base. In a larger system, environment overlays, policy objects, and dependencies would be separated more explicitly.

```
book-gitops/
├── apps/
│   └── podinfo/
│       └── base/
│           ├── namespace.yaml
│           ├── deployment.yaml
│           ├── service.yaml
│           └── kustomization.yaml
└── clusters/
    └── book/
        └── kustomization.yaml
```

Use the following minimal resources. The image tag is intentionally visible for the exercise; in production, resolve and record the immutable digest as part of promotion evidence.

```
# apps/podinfo/base/namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
```

```
  name: book-gitops
  labels:
    app.kubernetes.io/part-of: book-gitops-lab
---
# apps/podinfo/base/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: podinfo
  namespace: book-gitops
spec:
  replicas: 2
  selector:
    matchLabels:
      app: podinfo
  template:
    metadata:
      labels:
        app: podinfo
    spec:
      securityContext:
        runAsNonRoot: true
      containers:
      - name: podinfo
        image: ghcr.io/stefanprodan/podinfo:6.14.0
        ports:
        - name: http
          containerPort: 9898
        readinessProbe:
          httpGet:
            path: /readyz
            port: http
        livenessProbe:
          httpGet:
            path: /healthz
            port: http
        resources:
          requests:
            cpu: 50m
            memory: 64Mi
          limits:
            cpu: 250m
            memory: 128Mi
---
# apps/podinfo/base/service.yaml
apiVersion: v1
kind: Service
metadata:
```

```
  name: podinfo
  namespace: book-gitops
spec:
  selector:
    app: podinfo
  ports:
  - name: http
    port: 9898
    targetPort: http
---
# apps/podinfo/base/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- namespace.yaml
- deployment.yaml
- service.yaml
---
# clusters/book/kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- ../../apps/podinfo/base
```

Commit and push the initial state. Record the commit SHA as the initial desired-state revision.

```
git add .
git commit -m "Deploy podinfo 6.14.0"
git push origin main
git rev-parse HEAD | tee evidence/initial-config-sha.txt
```

12.15.4 Create the Azure Flux Configuration

Obtain cluster credentials for observation. The authoritative deployment will still be performed by Flux.

```
az aks get-credentials --resource-group "$RG" --name "$AKS" --overwrite-existing
```

Create a Flux configuration. Creating the first configuration installs the Microsoft Flux extension automatically if it is not already present. The Kustomization enables pruning so objects removed from the desired state are removed from the cluster.

```
az k8s-configuration flux create   --resource-group "$RG"   --cluster-name
"$AKS"   --cluster-type managedClusters   --name "$FLUX_CONFIG"   --scope
cluster   --namespace flux-system   --kind git   --url "$REPO_URL"   --branch
"$BRANCH"   --sync-interval 1m   --kustomization name=book-app
path=./clusters/book prune=true timeout=2m
```

If the CLI requires comma-separated or repeated Kustomization parameters in a newer version, use the help output to preserve the same intent: name book-app, path ./clusters/book, prune true, timeout two minutes.

```
az k8s-configuration flux show   --resource-group "$RG"   --cluster-name "$AKS"
--cluster-type managedClusters   --name "$FLUX_CONFIG"   --output json >
evidence/flux-configuration.json

kubectl -n flux-system get gitrepositories,kustomizations -o wide
kubectl -n "$APP_NS" get deployment,pods,service
kubectl -n "$APP_NS" rollout status deployment/podinfo --timeout=3m
```

Record the source revision and the actual image digest. The imageID field proves which immutable image the runtime resolved from the human-readable tag.

```
kubectl -n flux-system get gitrepository -o json   > evidence/git-source-
status.json
kubectl -n flux-system get kustomization -o json   > evidence/kustomization-
status.json
kubectl -n "$APP_NS" get pods -l app=podinfo   -o jsonpath='{range .items[*]}
{.metadata.name}{"  "}{.spec.containers[0].image}{" "}
{.status.containerStatuses[0].imageID}{"
"}{end}'   | tee evidence/runtime-initial.txt
```

Forward a local port and verify the application version. Run port-forward in a second terminal if preferred.

```
kubectl -n "$APP_NS" port-forward service/podinfo 9898:9898 &
PF_PID=$!
sleep 3
curl -fsS http://127.0.0.1:9898/version   | tee evidence/version-initial.json
kill "$PF_PID"
```

12.15.5 Release a New Version Through Git

Change only the image reference from 6.14.0 to 6.14.1. In a governed repository, create a branch and pull request, run required checks, review the rendered diff, and merge. For a personal lab repository, a direct commit is mechanically sufficient but provides weaker

governance evidence.

```
git checkout -b release/podinfo-6.14.1
sed -i.bak 's/podinfo:6.14.0/podinfo:6.14.1/'
apps/podinfo/base/deployment.yaml
rm apps/podinfo/base/deployment.yaml.bak
git diff
git add apps/podinfo/base/deployment.yaml
git commit -m "Promote podinfo to 6.14.1"
git push -u origin release/podinfo-6.14.1
# Merge the reviewed pull request, or merge locally for the lab.
git checkout main
git pull --ff-only
git rev-parse HEAD | tee evidence/release-config-sha.txt
```

Wait for Flux to observe the commit and Kubernetes to complete the rollout. You may trigger reconciliation for lab speed, but the controller remains the actor that applies desired state.

```
flux reconcile source git book-gitops -n flux-system --with-source || true
flux reconcile kustomization book-app -n flux-system --with-source || true
kubectl -n "$APP_NS" rollout status deployment/podinfo --timeout=5m
kubectl -n "$APP_NS" get pods -l app=podinfo   -o jsonpath='{range .items[*]}
{.metadata.name}{"  "}{.spec.containers[0].image}{" "}
{.status.containerStatuses[0].imageID}{"
"}{end}'   | tee evidence/runtime-release.txt
```

Verify the new response through the same Service contract and save the controller revision.

```
kubectl -n "$APP_NS" port-forward service/podinfo 9898:9898 &
PF_PID=$!
sleep 3
curl -fsS http://127.0.0.1:9898/version   | tee evidence/version-release.json
kill "$PF_PID"
kubectl -n flux-system get gitrepository,kustomization -o yaml   >
evidence/flux-after-release.yaml
```

12.15.6 Prove Drift Correction

Introduce a harmless imperative drift by changing the replica count from two to one. This command is not a release; it is a test of reconciliation. Flux should restore two replicas because the Git declaration owns that field.

```
kubectl -n "$APP_NS" scale deployment/podinfo --replicas=1
kubectl -n "$APP_NS" get deployment/podinfo   -o jsonpath='{.spec.replicas}{"
"}'   | tee evidence/replicas-immediate.txt
sleep 90
kubectl -n "$APP_NS" get deployment/podinfo   -o jsonpath='{.spec.replicas}{"
"}'   | tee evidence/replicas-reconciled.txt
```

If the value remains one, inspect the Kustomization interval, source revision, controller events, and whether another controller owns replicas. Do not enable HPA for this drift test; HPA and Git would otherwise contend for the same field.

12.15.7 Roll Back Through a Git Revert

Create a revert of the release commit and merge it through the same governance path. This produces a new commit that restores the previous desired state while retaining the complete history. It is preferable to resetting or deleting history.

```
git checkout main
git pull --ff-only
git log --oneline -5
# Replace RELEASE_COMMIT with the commit that promoted 6.14.1.
git revert RELEASE_COMMIT
git push origin main
git rev-parse HEAD | tee evidence/rollback-config-sha.txt

flux reconcile source git book-gitops -n flux-system --with-source || true
flux reconcile kustomization book-app -n flux-system --with-source || true
kubectl -n "$APP_NS" rollout status deployment/podinfo --timeout=5m
kubectl -n "$APP_NS" get pods -l app=podinfo   -o jsonpath='{range .items[*]}
{.metadata.name}{"  "}{.spec.containers[0].image}{" "}
{.status.containerStatuses[0].imageID}{"
"}{end}'   | tee evidence/runtime-rollback.txt
```

Verify that the application reports the restored version and that Flux reports the rollback commit as the applied source revision.

```
kubectl -n "$APP_NS" port-forward service/podinfo 9898:9898 &
PF_PID=$!
sleep 3
curl -fsS http://127.0.0.1:9898/version   | tee evidence/version-rollback.json
kill "$PF_PID"
kubectl -n flux-system get gitrepository,kustomization -o yaml   >
evidence/flux-after-rollback.yaml
```

12.15.8 Evidence Packet

Table 12-17 Lab 12 evidence packet.

Evidence	Acceptance criterion
Repository history	Initial, release, and revert commits are present; the release diff changes only desired state.
Flux configuration	Azure fluxConfiguration resource and in-cluster GitRepository/Kustomization are Ready.
Initial state	Podinfo 6.14.0 desired reference, runtime imageID digest, rollout success, and /version response.
Release state	Podinfo 6.14.1 desired reference, new runtime imageID digest, applied source revision, and /version response.
Drift test	Imperative replica change is corrected back to the declared count.
Rollback state	Revert commit becomes the applied revision and runtime returns to 6.14.0.
No direct authoritative mutation	No kubectl apply/set image/edit command was used to perform release or durable rollback.
Cleanup	Flux configuration and namespace are removed; cluster/resource group is deleted if created solely for the lab.

Create a manifest for the evidence files and hash it. In a production system, store the packet in an immutable or access-controlled evidence repository with defined retention.

```
find evidence -type f -maxdepth 1 -print0   | sort -z   | xargs -0 shasum -a 256
> evidence/SHA256SUMS.txt
```

12.15.9 Cleanup and Zero-Residual Verification

Delete the Azure Flux configuration. The extension may remain if other configurations use it. Delete the application namespace. If the cluster and resource group were created only for this lab, delete the resource group and verify completion.

```
az k8s-configuration flux delete   --resource-group "$RG"   --cluster-name
"$AKS"   --cluster-type managedClusters   --name "$FLUX_CONFIG"   --yes

kubectl delete namespace "$APP_NS" --ignore-not-found
az k8s-configuration flux list   --resource-group "$RG"   --cluster-name "$AKS"
--cluster-type managedClusters   --output table
kubectl get namespace "$APP_NS" 2>&1   | tee evidence/namespace-after-
cleanup.txt
```

If the repository exists only for the lab, archive or delete it after preserving the evidence packet. If you delete an ephemeral AKS resource group, record the deletion operation and verify that no managed disks, public IPs, load balancers, or monitoring resources remain in a separate resource group.

12.15.10 Failure Diagnosis

Table 12-18 Common Lab 12 failures and diagnostic evidence.

Symptom	Likely cause	Evidence and response
Flux configuration not Ready	Repository URL/branch/path error, extension issue, network egress or source authentication.	Inspect Azure fluxConfiguration status, GitRepository conditions and controller logs.
Kustomization Ready=False	Invalid YAML, missing namespace, failed health check or policy rejection.	Inspect Kustomization events, kubectl describe, admission messages and rendered manifests.
ImagePullBackOff	Tag/digest absent, registry unavailable or pull authorization failure.	Inspect Pod events and image reference; confirm public or credentialed registry access.
Rollout exceeds deadline	Readiness failure, quota/capacity, insufficient resources or application error.	Use rollout status, Deployment conditions, Pod events and logs.
Drift not corrected	Reconciliation suspended, interval not elapsed, source stale or field ignored/owned elsewhere.	Check suspend state, source revision, Kustomization interval and managed fields.
Git revert merged but runtime unchanged	Controller has not fetched new revision or apply failed.	Compare Git HEAD, GitRepository artifact revision, Kustomization last applied revision and events.
Runtime undo is overwritten	Git still declares the newer version and self-heal is active.	This is expected GitOps behavior; revert or update authoritative desired state.

12.15.11 Multicloud Translation

The lab's engineering intent can be translated without claiming identical managed services. On Alibaba Cloud, ACK One GitOps can host managed Argo CD and reconcile the repository across ACK clusters. On Tencent Cloud TKE or Volcengine VKE, use self-managed Flux or Argo CD for the same pull/reconcile/drift experiment, or compare the result with the respective pipeline-driven deployment service and document the different credential and drift model. The acceptance criteria—versioned desired state, controller-applied release, evidence, source-authoritative rollback, and zero residuals—remain constant.

12.16 CloudDesk Capability Handoff

CloudDesk FinOps Agent should not receive unrestricted cluster credentials or a generic "run kubectl" tool. It should interact with the delivery control plane through narrow capabilities that preserve the governance path.

Table 12-19 CloudDesk delivery capabilities derived from this chapter.

Capability	Behavior	Risk and approval boundary
gitops.change.propose	Create a branch and pull request that changes a bounded desired-state path; attach artifact and evidence identities.	No merge and no cluster mutation. Repository path allowlist, schema validation and owner review required.
gitops.sync.observe	Read source revision, reconciliation conditions, drift and deployed subject from controller and runtime evidence.	Read-only. Must not disclose unrelated tenant or namespace state.
gitops.rollback.trigger	Prepare a revert pull request for a named release and explain compatibility constraints.	Approval required; cannot force-push or execute runtime undo silently.
delivery.release.evidence	Assemble source, artifact, config, approval, rollout, telemetry and cost records into a signed packet.	Read-only collection with retention and access policy; missing links must be reported.

These tools reinforce the separation established in Chapter 20: an agent can propose and explain change, but the repository rules, policy gates, approvers, reconciler, and observability systems retain authority. The agent should never convert an optimization recommendation into an unreviewed production mutation simply because the cloud API permits it.

12.17 Key Takeaways

- Continuous delivery is a managed feedback loop from intent to verified outcome, not merely a build-and-deploy script.
- Build artifacts once, identify them immutably, and promote them by changing versioned desired state.
- An environment is a policy, identity, data, network, SLO, quota, cost, and evidence boundary—not just a branch or namespace.
- OpenGitOps requires declarative desired state, versioned immutable history, automatic pull, and continuous reconciliation.
- Pull-based GitOps narrows external deployment credentials but gives the reconciler high-value authority that still requires least privilege and multitenancy controls.

- Git history is not sufficient governance; pull requests, ownership, required checks, approvals, policy, exceptions, and decision records create the control contract.
- Drift must be classified. Unauthorized manual change, controller-owned fields, generated secrets, admission mutation, and emergency intervention require different responses.
- Progressive delivery uses traffic and evidence to control blast radius. Technical health, security, business value, AI quality, and cost can all be promotion gates.
- Runtime undo, source revert, and forward fix are different recovery mechanisms. GitOps rollback becomes durable only when authoritative desired state changes.
- Release evidence should connect source, artifact, configuration, approval, controller state, runtime identity, telemetry, cost, and outcome.
- AI-native delivery must version models, prompts, knowledge, tools, safety policies, evaluations, and agent definitions as governed release subjects.
- Multicloud delivery should preserve engineering intent while documenting non-equivalence in credentials, reconciliation, fleet management, evidence, and responsibility.

References and Further Reading

OpenGitOps. “GitOps Principles,” version 1.0.0. https://opengitops.dev/.

Flux Project. “Core Concepts.” https://fluxcd.io/flux/concepts/.

Flux Project. “Kustomizations.” https://fluxcd.io/flux/components/kustomize/kustomizations/.

Flux Project. “Monitoring and Alerts.” https://fluxcd.io/flux/monitoring/alerts/.

Microsoft. “Application deployments with GitOps (Flux v2).” Azure Arc-enabled Kubernetes and AKS documentation.

Microsoft. “GitOps for Azure Kubernetes Service.” Azure Architecture Center.

Microsoft. Azure CLI reference for az k8s-configuration flux and Flux Kustomizations.

Microsoft. ARM/Bicep reference for Microsoft.KubernetesConfiguration/fluxConfigurations.

Microsoft. “Use the Azure Login action with OpenID Connect.” GitHub Actions for Azure documentation.

Argo Project. “Argo CD: Declarative GitOps Continuous Delivery for Kubernetes.” https://argo-cd.readthedocs.io/.

Argo Project. “Automated Sync Policy.” Argo CD documentation.

Argo Project. “Argo Rollouts: Kubernetes Progressive Delivery Controller.” https://argoproj.github.io/argo-rollouts/.

Kubernetes Documentation. “Deployments,” “Update a Deployment Without Downtime,” and kubectl rollout reference.

GitHub Documentation. “Available rules for rulesets,” “About code owners,” and “Using environments for deployment.”

GitHub Documentation. “OpenID Connect” and “Artifact attestations.”

DORA. “DORA’s software delivery performance metrics.” https://dora.dev/guides/dora-metrics/.

CNCF. “Cloud Native Security Whitepaper” and project documentation for Flux and Argo.

SLSA. “Supply-chain Levels for Software Artifacts, version 1.0.” https://slsa.dev/.

in-toto Project. “A framework to secure the integrity of software supply chains.” https://in-toto.io/.

OpenSSF. “Secure Software Development Fundamentals” and related supply-chain guidance.

Alibaba Cloud. “ACK One GitOps overview,” “Work with GitOps,” and “Use Argo Rollouts to implement canary releases.”

Tencent Cloud. “CODING Continuous Deployment” and deployment-to-Kubernetes/TKE documentation.

Volcengine. Continuous Delivery/CodePipeline documentation for VKE and Kubernetes deployment.

Stefan Prodan. “Podinfo” repository and immutable release history. https://github.com/stefanprodan/podinfo.

Forsgren, Nicole; Humble, Jez; and Kim, Gene. Accelerate: The Science of Lean Software and DevOps. IT Revolution, 2018.

Beyer, Betsy; Jones, Chris; Petoff, Jennifer; and Murphy, Niall Richard, eds. Site Reliability Engineering. O’Reilly Media, 2016.

Humble, Jez, and Farley, David. Continuous Delivery. Addison-Wesley, 2010.

Burns, Brendan; Beda, Joe; Hightower, Kelsey; and Evenson, Lachlan. Kubernetes: Up and Running, 3rd ed. O’Reilly Media, 2022.

Operations and Value Control Planes

Once a system is running, how does its owner know whether it is reliable, affordable, and compliant? Connect observations about service, spend, and security to decisions with named owners, then check the effects of those decisions in operation.

13 Observability as a Managed Feedback Loop

14 FinOps as a Cloud Control Plane

15 Security Posture, Compliance, and Organizational Governance

Part IV / Chapter 13

Observability as a Managed Feedback Loop

From Correlated Telemetry to SLO-Guided Decisions, Continuous Verification, and Resilience Evidence

Logs, metrics, traces, and profiles are raw signals. Observability becomes an engineering capability only when the cloud control plane turns those signals into explainable decisions about release risk, service reliability, cost, and recovery.

In This Chapter

- Why observability is a managed feedback loop rather than a telemetry collection project
- Why logs, metrics, traces, profiles, and events are signals rather than independent pillars
- How OpenTelemetry creates a portable instrumentation and context-propagation contract
- How Azure Monitor separates collection, storage, analysis, alerting, and response into governable resources
- How SLIs, SLOs, error budgets, and burn rate convert user experience into operating policy
- How alert rules, action groups, and processing rules route decisions instead of merely sending notifications
- Where AIOps can assist anomaly detection, correlation, and investigation - and where human authority must remain
- How continuous verification connects releases and feature changes to observable evidence
- How chaos experiments use observability as the oracle for resilience engineering
- How retention, sampling, cardinality, privacy, and ingestion volume make observability a FinOps concern
- How Azure, Alibaba Cloud, Tencent Cloud, and Volcengine express the same observability intent with different control-plane objects

- How to instrument a service, define an SLO, create a verifiable alert, collect evidence, and clean up the resources

A service can have a dashboard full of green charts while users are unable to complete the task that matters. CPU may be low because requests are being rejected before they reach the application. Average latency may look healthy while a small but valuable tenant experiences timeouts. A model endpoint may return HTTP 200 while producing no grounded answer. A queue may appear stable because messages are being discarded rather than processed. In each case, monitoring data exists, but the system is not being managed through evidence.

This chapter adopts an outcome-first definition: observability is the ability to ask and answer questions about system behavior by using correlated telemetry and operational context. It becomes a managed cloud capability when the control plane defines where data is collected, who may query it, how long it is retained, which service objectives govern decisions, what alerting policy routes a response, and how an intervention is verified.

The central feedback loop is simple to state and demanding to implement: system behavior produces telemetry; telemetry is interpreted against service objectives; policy selects an action; the action changes the system; and subsequent telemetry verifies whether the intended improvement occurred. The loop closes only when evidence changes a decision.

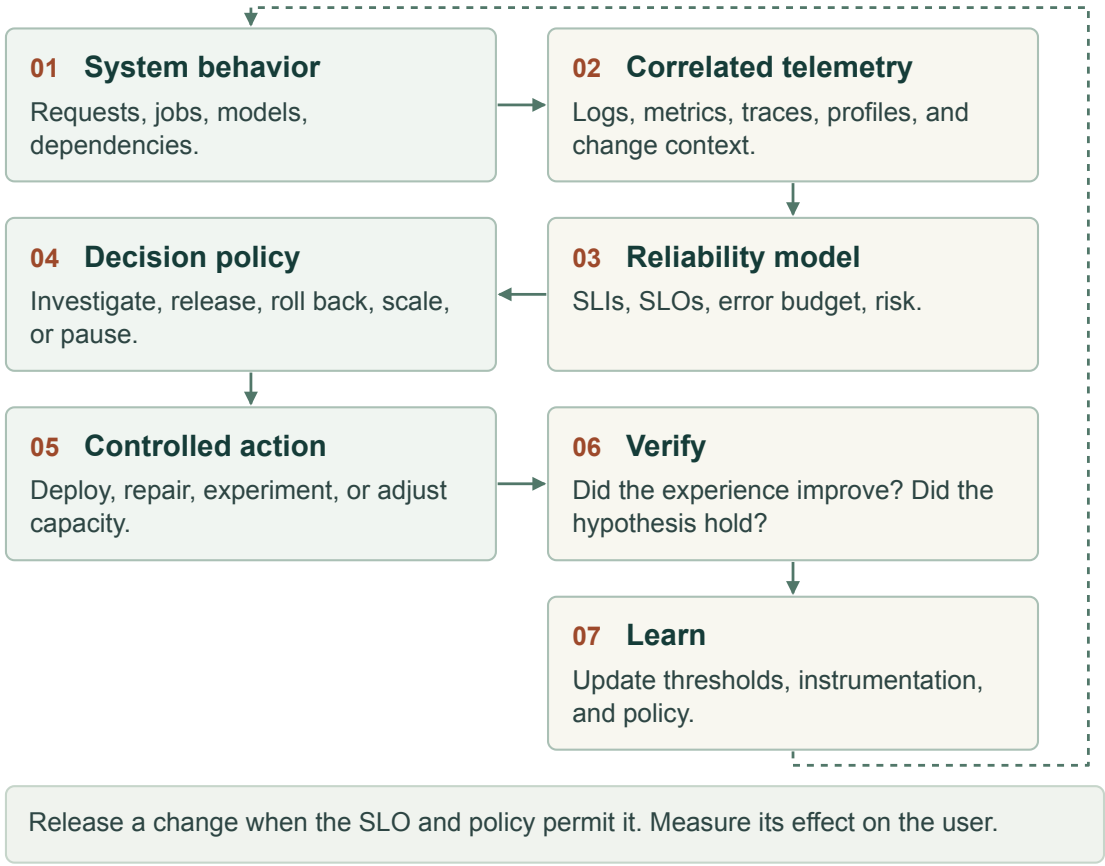


Figure 13-1 A managed observability feedback loop. Signals become useful when they support reliability models, decision policy, controlled action, verification, and learning.

13.1 Observability Is a Control System, Not a Data Collection Project

The word monitoring often describes checking known conditions: Is the virtual machine running? Is CPU utilization above a threshold? Did an HTTP endpoint return a response? These questions remain useful, but they are insufficient for distributed and AI-native systems. Modern failures are frequently emergent. No single component is obviously broken, yet a particular combination of version, region, tenant, feature flag, downstream dependency, and request shape produces unacceptable behavior.

Observability supports investigation of both known and novel failure modes. It permits an operator or developer to move from a user-visible symptom to the exact path, dependency, configuration, release, or data condition that explains the symptom. The cloud control

plane then adds the resources and policies required to operate that capability at scale: workspaces, collection rules, diagnostic settings, identities, private endpoints, retention plans, alert rules, action groups, workbooks, and automation targets.

A control system has a measured state, a desired state, a policy that compares the two, and an actuator that changes the system. Observability provides the measured state and the evidence needed to evaluate a change. SLOs describe a desired service outcome. Alerting, release gates, autoscaling, rollback, and remediation are actuators or decision routes. Continuous verification tests whether an action moved the service toward the objective.

Table 13-1 Monitoring, telemetry, observability, and operational control

Concept	Primary question	Typical artifact	Failure when used alone
Monitoring	Is a known condition outside an expected range?	Dashboard, health check, threshold alert	Misses novel failure modes and can optimize component health instead of user outcomes.
Telemetry	What signals did the system emit?	Logs, metrics, traces, profiles, events	Produces data volume without guaranteeing correlation, interpretation, or action.
Observability	Why is this behavior occurring, and who is affected?	Correlated queries, traces, dimensions, change context	Can remain an investigative tool if it is not linked to objectives and policy.
Managed feedback loop	What decision should be made, what evidence authorizes it, and did it work?	SLI/SLO, alert rule, action route, experiment report, audit trail	Fails when objectives, authority, evidence quality, or verification are ambiguous.

The purpose is not to centralize every byte of operational data. The purpose is to create a trustworthy evidence path from a user outcome to an engineering decision. That path should be queryable, explainable, permissioned, cost-aware, and testable.

13.1.1 The control plane has observable resources of its own

An observability platform is not a passive window. It is a managed system with resources and lifecycle. A workspace has a region, retention policy, access model, ingestion endpoints, quotas, and cost. A data collection rule selects sources and transformations. An alert rule has a query, evaluation window, severity, scope, and response route. A chaos workspace or experiment has an identity, target scope, permissions, actions, guardrails, and evidence.

This means observability design must answer the same six control-plane questions used elsewhere in this book: What is the resource? Who can create, change, query, or delete it? Through which network path is data ingested and queried? How is the configuration deployed as code? What audit evidence is produced? How is the service metered, limited, and optimized?

NOTE: A monitoring workspace with broad query access can expose sensitive user, tenant, or security data even when the production database is correctly protected. Telemetry is governed data, not harmless exhaust.

13.2 Signals Are Raw Materials, Not Pillars

The familiar phrase "three pillars of observability" can be useful as an introductory mnemonic, but it can also create three operational silos. Teams buy one log system, one metrics system, and one tracing system, then discover that the same request cannot be followed across them. The stronger model is to treat logs, metrics, traces, profiles, and events as different projections of a shared operational story.

Signals remain different because their semantics and economics are different. Metrics aggregate efficiently. Traces preserve causal structure. Logs preserve event detail. Profiles reveal where code consumes resources. Change events explain when a deployment, policy assignment, feature flag, or configuration revision altered the system. Unification does not mean forcing every signal into one storage format; it means carrying common context so that an investigation can move among them without losing identity.

Signals are different projections of one operational story

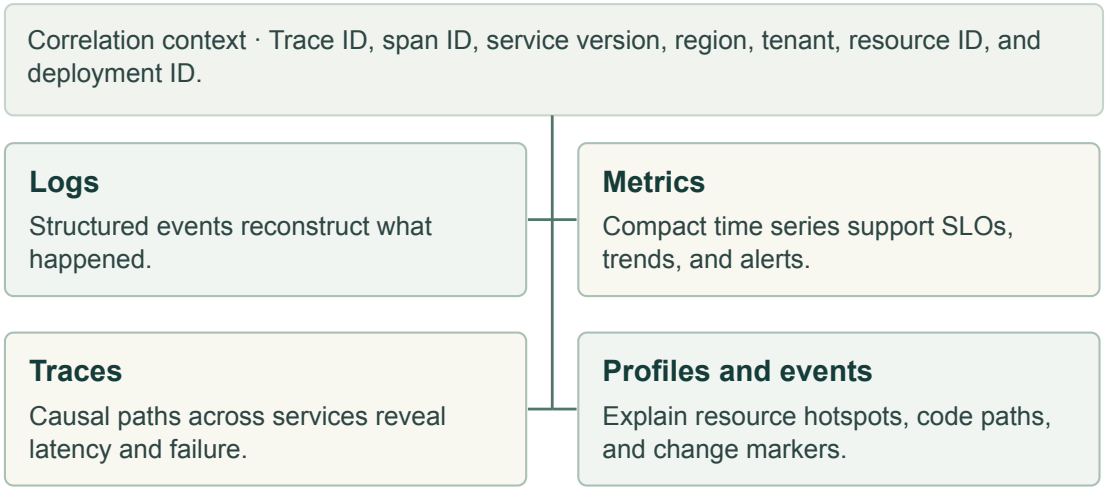


Figure 13-2 Logs, metrics, traces, profiles, and events retain different strengths but become substantially more useful when correlated by trace, service, resource, deployment, and business context.

13.2.1 Logs: structured stories, not text files

A production log should be an event with fields, not an unbounded sentence intended only for manual reading. At minimum, it should carry a timestamp, severity, service name, environment, version, operation, outcome, and a correlation identifier. When appropriate, it may include tenant, region, feature flag, model deployment, prompt template version, or business operation. Sensitive values should be redacted or represented by stable identifiers rather than copied into the event.

Logs are strongest when an engineer needs event detail: an exception, a policy decision, a rejected tool call, a retry reason, a schema mismatch, or the exact branch selected by a workflow. Their cost and usefulness deteriorate when every request prints the same unstructured context repeatedly, when secrets or personal data are embedded, or when high-volume debug output is retained indefinitely.

13.2.2 Metrics: compressed views for trends and policy

A metric is a numerical observation associated with dimensions and time. Counters, gauges, histograms, and summaries allow large volumes of activity to be represented compactly. Metrics are well suited to rates, ratios, distributions, capacity trends, autoscaling signals, and SLO calculations. They are also dangerous when dimensions are allowed to grow without bound. A user ID, request ID, document ID, or prompt text should not normally become a metric label.

Platform metrics such as CPU, memory, I/O, queue depth, and token throughput are useful diagnostic inputs. They are rarely the final reliability objective. A user cares whether a request succeeded within an acceptable time, whether a generated answer was grounded, whether a batch completed before its deadline, or whether a financial recommendation was produced from fresh cost data.

13.2.3 Traces: the causal and temporal skeleton

A distributed trace represents one end-to-end operation as a tree or graph of spans. Each span records a named operation, its duration, status, attributes, events, and relationships. Context propagation carries trace identity across process and network boundaries so that an inbound request, database query, message publish, queue consume, model call, retrieval query, and tool execution can be assembled into a causal path.

Traces are especially valuable in AI-native applications because a single user request often expands into multiple retrieval operations, model calls, safety checks, tool executions, and asynchronous work items. A response can be slow even when every individual component appears healthy in aggregate. The trace exposes the critical path and the distribution of delay across orchestration steps.

13.2.4 Profiles, events, and business context

Continuous profiling adds code-level evidence about CPU time, allocation, blocking, or other resource hot spots. It complements rather than replaces traces: a trace shows which operation was slow; a profile can show which code path consumed the time. Deployment, configuration, policy, and feature-flag events add change context. Business dimensions connect technical behavior to the population that experienced it.

Table 13-2 Signal strengths, typical questions, and common misuse

Signal	Best questions	Control-plane concerns	Common misuse
Logs	What exactly happened? What decision or exception occurred?	Schema, redaction, retention, query access, ingestion volume	Unstructured text, secrets, excessive debug logging, no correlation ID.
Metrics	Is a rate, ratio, distribution, or capacity trend changing?	Label policy, aggregation, retention, alert evaluation, cardinality	Using host averages as user SLOs; labels containing unbounded identifiers.
Traces	Where did time or failure propagate across a request path?	Sampling, context propagation, tenant boundaries, dependency instrumentation	Sampling away rare failures; treating every span attribute as harmless.

Signal	Best questions	Control-plane concerns	Common misuse
Profiles	Which code paths consume CPU, memory, or wait time?	Collection overhead, access to symbols, retention, privacy	Collecting continuously without a question or cost boundary.
Events and changes	What deployment, policy, feature, or topology change preceded the symptom?	Audit integration, clock alignment, immutable identifiers	Keeping release data separate from application telemetry.
Business context	Which customers, tenants, workflows, or outcomes are affected?	Data classification, pseudonymization, access scope	Copying sensitive business data into labels or logs.

13.3 OpenTelemetry as the Instrumentation Contract

OpenTelemetry provides vendor-neutral APIs, SDKs, semantic conventions, context propagation, and a Collector for receiving, processing, and exporting telemetry. Its primary architectural value is not that every backend becomes equivalent. Its value is that application code can describe operations in a standard way while deployment teams choose where telemetry is processed and stored.

The instrumentation contract has four layers. Application and library instrumentation create spans, metrics, and logs. Context propagation carries causal identity. The Collector receives and processes data through pipelines. Exporters send data to one or more backends. The backend still determines query language, retention, analytics, alerting, access control, and cost.

Table 13-3 Responsibilities in an OpenTelemetry architecture

Layer	Responsibility	Design questions
API and SDK	Create telemetry and attach semantic attributes.	Which operations matter? Which attributes are stable, bounded, and safe?
Instrumentation libraries	Observe frameworks, HTTP, databases, messaging, runtimes, and AI libraries.	What is automatic? Where are manual spans or business metrics required?
Context propagation	Carry trace and baggage context across process boundaries.	Which protocols preserve context? What should never be placed in baggage?
Collector	Receive, batch, filter, sample, transform, enrich, route, and export.	Where does it run? How is it scaled, secured, and monitored?

Layer	Responsibility	Design questions
Backend	Store, query, visualize, correlate, alert, and retain.	What are the access model, query language, retention, cost, and evidence guarantees?

13.3.1 Context propagation is the connective tissue

A trace identifier is valuable only when every hop preserves it. HTTP headers, message metadata, background jobs, and tool calls must propagate context. Logs should include trace and span identifiers where supported. Metrics may be linked through exemplars or derived from traces. Service, version, resource, environment, and deployment attributes provide the stable dimensions needed to compare behavior across changes.

OpenTelemetry baggage can carry contextual values across boundaries, but it should be used sparingly. Baggage is propagated to downstream services and may cross trust boundaries. Sensitive information, tokens, raw prompts, customer names, or unrestricted tenant data do not belong there. A stable tenant identifier may be useful in a controlled environment, but the receiving services, exporters, and storage access model must all be reviewed.

13.3.2 Vendor-neutral instrumentation does not mean vendor-free operations

OpenTelemetry reduces coupling at the instrumentation boundary, but operational semantics still differ among cloud platforms. One service may store application traces in a log analytics workspace, another in a dedicated APM store, and another in a log service. Query languages, sampling options, retention tiers, metric limits, alert semantics, and billing units differ. Portability therefore means preserving the intent and evidence model, not assuming product interchangeability.

13.3.3 Sampling is an evidence policy

Sampling controls volume by retaining only part of the trace population. Head sampling decides near the start of a trace and is operationally simple, but it cannot know whether the trace will later become slow or fail. Tail sampling decides after more of the trace is available, allowing policies that retain errors, high latency, important tenants, or unusual paths. Tail sampling requires buffering and a collector architecture capable of seeing the complete trace.

Sampling must be evaluated against the questions the system must answer. A 1% random sample may be adequate for aggregate latency trends but can erase a rare cross-tenant or authorization failure. SLO calculations should generally rely on complete or appropriately aggregated request counts rather than sampled traces. Sampling configuration itself should be versioned and observable.

NOTE: Telemetry pipelines need their own service objectives. If spans arrive late, context is broken, or collection drops exactly during overload, the observability system cannot be treated as a reliable oracle.

13.4 Azure as the Reference Observability Control Plane

Azure Monitor is the reference platform for this chapter because it exposes the major observability responsibilities as cloud resources. Application Insights provides application performance monitoring and distributed tracing. Log Analytics workspaces store and query log and trace data with Kusto Query Language (KQL). Azure Monitor workspaces currently provide the managed repository for Prometheus metrics and expose PromQL. Platform metrics remain available through Azure Monitor Metrics. Workbooks and Azure Managed Grafana provide visualization. Alert rules, action groups, and alert processing rules route responses. Diagnostic settings, Azure Monitor Agent, data collection rules, data collection endpoints, and OpenTelemetry integrations govern collection.

The architecture is intentionally plural. Logs and traces do not have the same store or query semantics as Prometheus metrics. Platform metrics, resource logs, application telemetry, and activity logs are produced by different sources. The management objective is to connect them through stable resource and operation context rather than force them into one undifferentiated database.



Figure 13-3 Azure Monitor reference architecture. Collection, storage, analysis, decision routing, and cross-cutting governance are separate but connected control-plane responsibilities.

13.4.1 Workspaces are governance boundaries

A workspace is more than a place where charts obtain data. It defines a regional data boundary, permissions, retention behavior, query scope, and cost allocation point. A Log Analytics workspace holds log and trace tables. A workspace-based Application Insights resource writes application telemetry into an associated Log Analytics workspace. An Azure Monitor workspace holds Prometheus metrics and has its own permissions and capacity considerations.

Workspace topology should be designed from access, residency, retention, query, and organizational requirements. One workspace per application can create fragmentation; one workspace for an entire enterprise can create overly broad access and noisy cost attribution. The correct boundary follows governance and operations, not a universal formula.

13.4.2 Collection is policy, not agent installation

Azure Monitor Agent and data collection rules separate data source configuration from individual machines. Diagnostic settings route supported resource logs and metrics. Data collection endpoints provide explicit ingestion paths when required. OpenTelemetry instrumentation emits application telemetry. These resources should be deployed with IaC and reviewed like other production configuration.

Collection rules should state what is collected, from which scope, with which transformations, and to which destination. A transformation can remove unnecessary fields or normalize records before storage. A diagnostic setting should not indiscriminately enable every category forever. More data can increase cost and make investigation slower without improving decision quality.

13.4.3 Private connectivity and identity

Telemetry often contains resource identifiers, user operations, exception detail, request attributes, and security evidence. Azure Monitor Private Link Scope and related private endpoints can constrain ingestion and query paths for supported resources. RBAC controls who can configure collection, query workspaces, modify alerts, or execute response automation. Managed identities should be preferred for agents, collectors, automation, and chaos workspaces where supported.

13.4.4 Query languages reflect signal semantics

KQL is designed for high-volume event and log analytics. PromQL is designed for labeled time series. Metrics Explorer provides interactive metric analysis. Transaction search and application maps use trace and dependency relationships. A mature operating model teaches engineers when to use each surface and how to pass stable identifiers among them.

Table 13-4 Azure observability objects and their control-plane responsibilities

Object	Primary responsibility	Evidence or query surface
Application Insights resource	Application requests, dependencies, exceptions, availability, distributed traces, application maps.	KQL, transaction search, failures and performance views.
Log Analytics workspace	Log and trace storage, KQL queries, table plans, retention, access and export.	KQL query results, workspace audit and usage records.
Azure Monitor workspace	Managed store for Prometheus metrics and related configuration.	PromQL, managed Grafana, recording and alert rules.
Data collection rule / endpoint	Select, transform, route, and securely receive supported telemetry.	Resource configuration, ingestion health and destination evidence.
Diagnostic setting	Route Azure resource logs and supported metrics to selected destinations.	Activity Log and destination records.
Alert rule	Evaluate a metric, log query, activity event, Prometheus rule, or other condition.	Alert instance, evaluation history, condition state.

Object	Primary responsibility	Evidence or query surface
Action group	Route notifications or automation to reusable destinations.	Action history and common alert payload.
Alert processing rule	Suppress or add action groups by scope, filter, and schedule.	Processing rule state and resulting alert history.
Workbook / Managed Grafana	Compose operational views and investigation paths.	Saved query, panel, parameters, links and dashboard version.
Chaos Studio Workspace / Scenario	Discover resources and run controlled resilience tests.	Scenario run and report, identity, scope and action history.

13.5 SLIs, SLOs, and Error Budgets

A service level indicator measures an outcome. A service level objective sets the acceptable level of that indicator over a defined population and time window. The error budget represents the tolerated unreliability. Together they convert an abstract desire for reliability into a measurable operating contract.

An availability SLI can be expressed as the ratio of good eligible events to all eligible events. A latency SLI can count eligible requests completed below a threshold. A freshness SLI can measure whether data is available within a deadline. For an AI application, indicators may include successful grounded answers, tool completion, policy-compliant action, or retrieval freshness. The indicator must represent what users or dependent systems experience.

```
availability_SLI = good_eligible_requests / all_eligible_requests

latency_SLI = requests_below_latency_threshold / all_eligible_requests

error_budget_fraction = 1 - SLO_target

remaining_budget = allowed_bad_events - observed_bad_events
```

SLIs, SLOs and error budgets convert user experience into operating policy

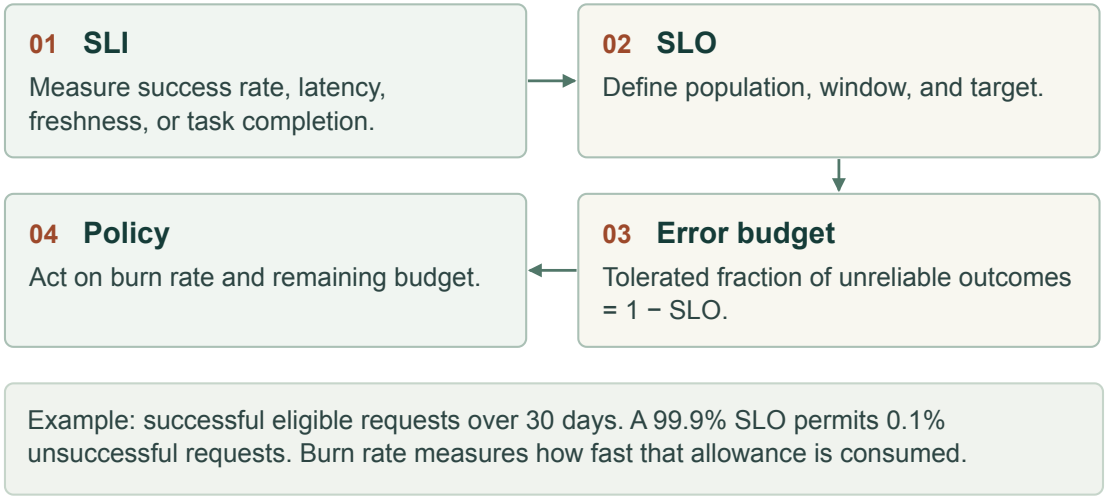


Figure 13-4 An SLI measures an outcome, an SLO defines the objective, an error budget quantifies tolerated unreliability, and policy determines the response.

13.5.1 Define the population and window

A ratio is meaningless without an eligible population. Health probes, load tests, malicious traffic, known client misuse, and administrative operations may need separate treatment. Exclusions must be documented before an incident, not invented after the service misses its objective. The time window should match business and operating behavior. Rolling windows provide continuous pressure; calendar windows align with reporting but reset abruptly.

13.5.2 Start from user outcomes, then map to telemetry

A platform metric can explain an SLO violation but should not automatically become the SLO. CPU saturation may cause latency, but the user experiences latency. Queue depth may predict missed deadlines, but the user experiences a late result. The reliability model should begin with the outcome and then identify the telemetry needed to calculate, explain, and forecast it.

Table 13-5 Diagnostic metrics and user-centered SLIs

System view	Useful diagnostic signal	Stronger user-centered indicator
Web API	CPU, memory, thread pool, connection pool	Eligible request success ratio and latency threshold attainment.

System view	Useful diagnostic signal	Stronger user-centered indicator
Event pipeline	Queue depth, consumer count, retry count	Percentage of events processed correctly before the business deadline.
RAG service	Vector query latency, top-k size, model token rate	Percentage of eligible questions answered with required citations and within the response deadline.
Agent tool execution	Worker utilization, function duration, dependency latency	Percentage of approved actions completed correctly, idempotently, and within policy time.
Cost ingestion	File count, parser CPU, API call rate	Percentage of required billing records ingested and queryable before the reporting cutoff.

13.5.3 Burn rate turns budget consumption into urgency

A service can be within its 30-day SLO while currently failing rapidly. Burn rate compares the current rate of bad events with the rate that would consume the entire error budget evenly over the window. A burn rate greater than one consumes budget faster than planned. Multi-window, multi-burn-rate alerting combines a fast window for severe incidents with a slower window for sustained degradation, reducing both delayed response and transient noise.

The error budget is not a punishment mechanism. It creates a negotiated rule for balancing change and reliability. When budget is healthy, teams can release. When consumption accelerates, they investigate or reduce risk. When the budget is exhausted, policy can pause nonessential change and prioritize reliability work. The exact policy must be agreed by product, development, operations, and business stakeholders.

13.6 Alerting as Decision Routing

An alert is useful only when it identifies a condition that requires a defined decision. Paging every metric threshold creates noise and trains responders to ignore the system. A strong alert states the affected objective, scope, evidence window, severity, likely user impact, and expected first action. It should route to the team or automation that has both responsibility and authority.

Azure Monitor supports metric alerts, log search alerts, activity log alerts, Prometheus alerts, service health alerts, and other types. Alert rules define the monitored scope and condition. Action groups provide reusable notification and automation destinations. Alert processing rules can add or suppress action groups by scope, filters, and schedule, which is useful during planned maintenance or for organization-wide routing policy.

The common alert schema helps webhook and automation consumers process a consistent envelope. Custom properties can carry service, runbook, SLO, owner, or deployment context. These properties should reference governed identifiers rather than copy sensitive telemetry into every notification.

Table 13-6 Alert design from symptom to accountable action

Design element	Question	Evidence
Objective	Which user or system outcome is at risk?	SLI, SLO ID, error-budget state.
Scope	Which resources, services, regions, versions, or tenants are affected?	Resource IDs and bounded dimensions.
Condition	What query or metric demonstrates the problem?	Versioned query, threshold, evaluation frequency, window.
Urgency	How quickly is harm or budget consumption increasing?	Severity, burn rate, minimum traffic and persistence.
Owner	Who has the responsibility and authority to act?	Service ownership and on-call route.
First action	What should happen when the alert fires?	Runbook, rollback gate, capacity action, investigation query.
Verification	How will resolution be proven?	Resolved condition, user outcome recovery, alert history and post-action query.

13.6.1 Alert quality is measurable

Teams should measure alert precision, time to acknowledge, time to useful diagnosis, duplicate volume, percentage with an actionable runbook, and percentage resolved without user impact. A low-volume service needs additional care because a small number of errors can make ratios unstable. Minimum event counts, synthetic traffic, or longer windows may be needed. These are modeling choices, not excuses to ignore reliability.

13.7 AIOps as Assisted Analysis, Not Autonomous Authority

AIOps applies statistical, machine-learning, and increasingly generative techniques to telemetry and operational context. Useful capabilities include dynamic thresholds, anomaly detection, issue grouping, cross-signal correlation, probable root-cause suggestions, capacity forecasting, and natural-language assistance for queries or investigations. These capabilities can reduce the amount of repetitive pattern recognition performed by humans.

Assistance is not authority. An anomaly detector may identify a change but cannot determine the business acceptability of the change without policy and context. A root-cause suggestion is a hypothesis, not evidence of causality. A generated query may be syntactically correct but examine the wrong population. Automated remediation can amplify an incorrect diagnosis or operate outside the intended tenant and resource scope.

Table 13-7 AIOps capability, required evidence, and guardrail

Capability	Useful output	Required guardrail
Dynamic threshold	Expected range adjusted for seasonality or history.	Minimum data quality, bounded dimensions, explainable baseline, fallback threshold.
Anomaly detection	Candidate deviations worthy of investigation.	Do not equate anomaly with incident; retain query and comparison window.
Alert grouping	Related alerts summarized as one operational issue.	Preserve original alert IDs, scopes, severities and suppressed evidence.
Root-cause assistance	Ranked dependencies, changes, traces, or resources.	Expose supporting evidence and confidence; require human or policy validation.
Forecasting	Projected saturation, capacity shortfall, or cost growth.	Track forecast error and prevent irreversible action from a single model.
Agentic investigation	Generated queries, navigation and incident summary.	Read-only by default; explicit identity, scope, audit, redaction and tool boundaries.
Automated remediation	Restart, scale, route, rollback or quarantine.	Preapproved action, blast-radius limit, idempotency, rollback and post-action verification.

The appropriate progression is human-in-the-loop assistance, policy-constrained automation, and only then narrow autonomous action for well-understood conditions. The system should preserve the evidence used, the identity that authorized the action, the exact target scope, and the result of verification.

13.8 Continuous Verification: Connect Delivery to the Feedback Loop

Observability creates the highest value when it is connected to change. A deployment, model revision, prompt template, retrieval index, feature flag, policy assignment, or infrastructure configuration should appear as operational context. Without change markers, responders can see that behavior changed but not what changed.

Continuous verification evaluates a release while exposure is still limited. A canary or staged deployment receives a controlled share of traffic. The system compares the new and baseline versions against the same service objectives and selected guardrail metrics. If evidence satisfies policy, exposure increases. If the error budget burns too quickly or a critical invariant fails, the release pauses or rolls back.

Table 13-8 Evidence required for an observable release

Phase	Control-plane artifact	Verification question
Before release	Versioned deployment, SLO definition, alert policy, expected change, rollback target.	Do we know what success and failure look like?
Initial exposure	Canary route, feature flag, tenant cohort, release annotation.	Is the test population controlled and identifiable?
Comparison	Queries grouped by version, region, cohort and dependency.	Is the new version materially different from the baseline?
Decision	Policy gate with time window, minimum traffic, burn rate and guardrails.	Does evidence authorize promotion, pause, or rollback?
After decision	Deployment history, query results, alert history, audit and incident notes.	Can another reviewer reproduce why the decision was made?

This is observability-driven development in practical form. Instrumentation and reliability criteria are designed with the feature, not added after production failure. Release engineering becomes a controlled experiment rather than a one-time act of confidence.

13.9 Resilience Engineering: Observability as the Oracle

Reliable systems are not proven by the absence of visible failure. They are tested against realistic disruption. Chaos engineering is a disciplined experiment: define the steady state, state a hypothesis, inject a controlled fault, observe the user-facing outcome, and learn. The objective is not to create dramatic outages. It is to expose hidden assumptions before an uncontrolled event does so.

Resilience engineering uses observability as the experiment oracle

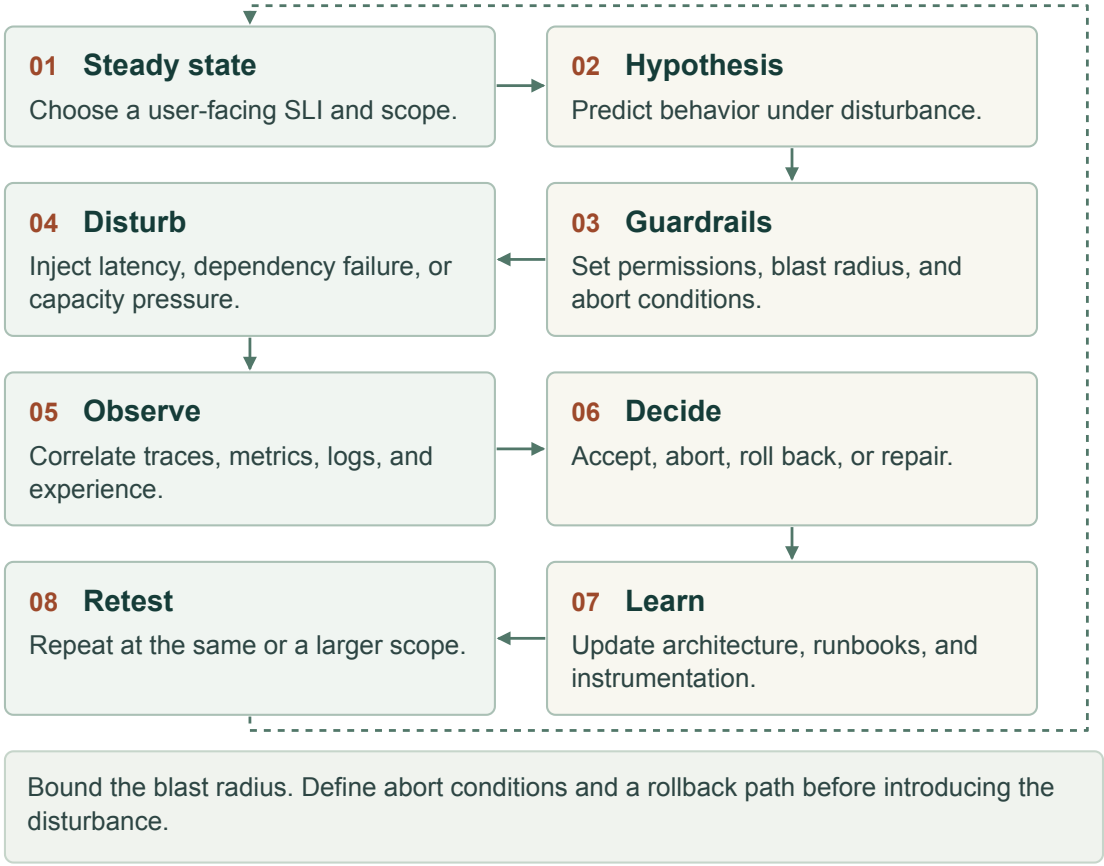


Figure 13-5 A resilience experiment begins with a measurable steady state and ends with evidence that changes architecture, operations, or instrumentation.

A safe experiment defines target scope, identity, blast radius, duration, abort condition, rollback, and observer. The steady state is expressed through SLIs, not merely component status. A hypothesis might state: "If one application instance is terminated, eligible request success remains above 99.9%, P95 latency remains below 300 ms, and no request is routed to an unauthorized tenant."

Azure Chaos Studio provides a managed resilience testing service. As of August 2026, Workspaces and Scenarios are a newer public-preview model, while classic experiments remain available. A Workspace uses managed identity and RBAC to execute permitted actions against explicit scopes. Scenario reports provide structured evidence of what ran,

what was skipped, how long actions took, and whether the run succeeded. Because preview status and product models are volatile, the manuscript should recheck these details at freeze.

Table 13-9 Minimum experiment contract

Field	Required content
Steady state	User-facing SLIs, query definitions, acceptable range and observation window.
Hypothesis	A falsifiable statement about behavior under a named disruption.
Target and scope	Exact resources, region, environment and tenant boundary.
Identity and authority	Principal used by the experiment and its narrow role assignments.
Blast radius	Maximum affected capacity, traffic, duration and dependent systems.
Abort and rollback	Automatic or human conditions that stop the experiment and restore state.
Observability plan	Signals, queries, dashboards, alerts and trace attributes used as the oracle.
Evidence packet	Experiment definition, run report, telemetry, decision, findings and action items.

13.10 Cost, Cardinality, Retention, and Privacy

Observability consumes storage, ingestion, compute, network, and human attention. Data volume can grow faster than application traffic because one request produces many spans and log records. The goal is not minimum telemetry cost; it is the best decision value per unit of cost. FinOps and reliability therefore share a design surface.

Cardinality is one of the most important cost and performance constraints. A metric labeled by request ID creates a new time series for each request. A trace attribute containing an entire prompt increases storage and privacy risk. A debug log that repeats a large object can dominate ingestion. Dimension policies should classify attributes as stable bounded dimensions, high-cardinality investigation fields, sensitive data, or prohibited values.

Table 13-10 Cost and governance levers

Lever	Benefit	Risk if misused
Instrumentation scope	Avoids telemetry for operations that do not support a decision.	Blind spots if critical paths or failures are not instrumented.

Lever	Benefit	Risk if misused
Filtering and transformation	Removes noise or sensitive fields before storage.	Irreversible loss of evidence; transformation bugs.
Sampling	Reduces trace and log volume.	Rare or tenant-specific incidents may disappear.
Aggregation / recording rules	Reduces query cost and supports fast SLO views.	Aggregation can conceal distribution tails or incorrect populations.
Retention tiers	Matches hot investigation data with longer-term compliance or trend needs.	Short retention prevents postmortem; long hot retention wastes cost.
Archive and export	Keeps evidence at lower cost or in an independent store.	Slower access, additional governance and retrieval cost.
Dimension policy	Controls metric series and protects sensitive data.	Overly strict policy can remove essential diagnostic context.
Workspace and tenant boundaries	Improves access control, residency and cost attribution.	Excess fragmentation makes cross-service investigation difficult.
Usage quotas and alerts	Prevents uncontrolled ingestion expense.	Hard caps can discard evidence during the incident that causes the spike.

13.10.1 The observability pipeline needs reliability indicators

A service may be healthy while its telemetry is delayed or incomplete. Operators should measure collector availability, export failures, dropped spans, ingestion delay, query freshness, context-propagation success, alert evaluation delay, and notification delivery. These indicators determine whether the platform can be trusted during an incident.

An observability SLO might state that 99.9% of eligible application request telemetry is queryable within two minutes, that trace context is present on 99.5% of distributed requests, or that critical alert notifications are delivered within sixty seconds. These are not substitutes for the application SLO; they qualify the evidence used to assess it.

13.10.2 Privacy and multi-tenant boundaries

Telemetry must not become a side channel between tenants. Logs, traces, dashboards, saved queries, exports, alert payloads, and AIOps summaries should preserve the same tenant and data-classification boundaries as the system they observe. Shared workspaces may be operationally efficient, but they require resource-context access controls, query governance, redaction, and explicit review of cross-tenant support workflows.

For AI applications, raw prompts, retrieved document text, model responses, tool arguments, and user files should not be recorded by default. Record stable identifiers, model and policy versions, token counts, timing, safety outcomes, and hashes where sufficient. Capture payloads only under an approved diagnostic policy with strict access and retention.

13.11 Multicloud Translation: Preserve the Engineering Intent

The major clouds expose similar observability intentions through different products. Azure organizes application monitoring, logs, Prometheus metrics, alerting, visualization, and resilience testing across Azure Monitor and related resources. Alibaba Cloud combines ARMS application monitoring and Managed Service for OpenTelemetry with Simple Log Service and Managed Service for Prometheus. Tencent Cloud Observability Platform combines product monitoring, APM, CLS, TMP, Grafana, and alert management; its Chaotic Fault Generator provides managed fault-injection workflows. Volcengine provides APMPlus, Torch Log Service, managed Prometheus and Kubernetes observability capabilities, with OpenTelemetry integration documented for application monitoring.

These services are not interchangeable SKU-by-SKU. The portable unit is the engineering contract: instrumented operation, propagated context, governed store, query, SLI, alert, action route, retention, cost boundary, and experiment evidence.

Table 13-11 Multicloud observability intent mapping

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Application traces / APM	Application Insights with Azure Monitor OpenTelemetry Distro.	ARMS Application Monitoring and Managed Service for OpenTelemetry.	TCOP Application Performance Management; OpenTelemetry integration is recommended.	APMPlus with OpenTelemetry integration.
Logs	Log Analytics workspace and resource diagnostic settings.	Simple Log Service (SLS), including ARMS observable data storage integration.	Cloud Log Service (CLS).	Torch Log Service (TLS).
Prometheus metrics	Azure Monitor workspace and managed service for Prometheus.	Managed Service for Prometheus; PromQL and OpenTelemetry metric ingestion.	TencentCloud Managed Service for Prometheus (TMP).	Managed Prometheus / Kubernetes monitoring capabilities.

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Visualization	Azure Workbooks and Azure Managed Grafana.	Managed Service for Grafana and ARMS dashboards.	TencentCloud Managed Service for Grafana and TCOP dashboards.	Dashboards in TLS, APMPlus and cloud-native monitoring surfaces.
Alerting and routing	Azure Monitor alert rules, action groups and processing rules.	ARMS alert management and Prometheus alert rules.	TCOP alarm policies and notification templates.	Product alarm and monitoring rules; confirm current integration at freeze.
Resilience testing	Azure Chaos Studio Workspaces/Scenarios and classic experiments.	Chaos engineering capabilities and open-source ChaosBlade ecosystem; verify selected managed path.	Tencent Smart Advisor Chaotic Fault Generator.	Documented chaos-engineering practices and product-specific tools; verify current managed offering.
Cost / retention controls	Workspace retention, table plans, sampling, export, usage and Azure Cost Management.	Ingestion-based ARMS limits, SLS retention, Prometheus storage tiers.	APM usage analysis, CLS lifecycle, TMP retention and archive.	TLS lifecycle/index controls and service-specific retention; verify billing units.
Portable layer	OpenTelemetry, Prometheus, service/resource attributes, SLO documents, alert and evidence contracts.	Same intent; product resource hierarchy and billing differ.	Same intent; integration and notification channels differ.	Same intent; English documentation depth and product boundaries differ.

NOTE: A multicloud chapter should state non-equivalence explicitly. Product names, retention, billing, preview status, supported regions and managed integrations are volatile and must be revalidated before manuscript freeze.

13.12 Lab 13: Instrument a Service, Define an SLO, and Trigger a Verifiable Alert

This lab creates an independent observability feedback loop. A local .NET minimal API emits telemetry to a workspace-based Application Insights resource. You generate successful and failed requests, calculate a five-minute success ratio with KQL, create a log search alert rule, trigger the condition, collect an evidence packet, and delete the Azure resources.

The lab focuses on one cloud capability: converting application telemetry into a governed reliability decision. It does not require a virtual machine, Kubernetes cluster, or production application. Local execution keeps infrastructure cost low while the control-plane resources remain real.

Lab 13 architecture: instrument, measure, decide and verify

Application and collection	Normal, slow, and failing endpoints in a .NET minimal API → Azure Monitor OpenTelemetry → logs and traces with service identity.
Measure and respond	Workspace-based Application Insights → KQL for five-minute success and error budget → scheduled log alert → verify firing and recovery.

Evidence packet: KQL results, alert state, resource IDs, test traffic, and cleanup record.

Figure 13-6 Lab 13 creates a complete but bounded feedback loop: instrumented service, managed telemetry, SLO query, scheduled alert, evidence, and cleanup.

13.12.1 Capability goal and acceptance criteria

You will prove the following statements:

- The service emits request telemetry with a stable service name to a workspace-based Application Insights resource.
- A KQL query calculates total requests, failed requests, success ratio, and an error-rate threshold over a five-minute window.
- A scheduled log search rule evaluates the same query and creates an Azure Monitor alert when the condition is met.
- The evidence packet contains resource IDs, the SLO definition, the query, test-traffic counts, query output, alert state, and cleanup verification.
- Deleting the resource group removes the Application Insights resource, Log Analytics workspace, and alert rule.

13.12.2 Cost and safety boundary

Table 13-12 Lab boundary

Item	Decision
Cloud resources	One resource group, one Log Analytics workspace, one workspace-based Application Insights resource, one log search alert rule.
Application host	Local development machine; no Azure compute resource is required.

Item	Decision
Expected cost class	Low. Telemetry ingestion and log-alert evaluation can incur charges; delete the group after verification.
Permissions	Create and delete resources in the lab resource group, create monitoring resources, and query the Application Insights resource.
Data policy	Synthetic requests only. Do not send production prompts, secrets, personal data, or customer identifiers.
Region	Use an Azure region supported by Log Analytics and Application Insights in your subscription.
Cleanup unit	The dedicated resource group named with the obs-lab- prefix.

13.12.3 Prerequisites

- Azure CLI authenticated to a subscription where you may create monitoring resources.
- .NET 8 SDK or a later compatible .NET SDK.
- curl or another HTTP client.
- Azure CLI version 2.54.0 or later for the scheduled-query extension. The extension installs automatically on first use in supported CLI configurations.
- A shell capable of setting environment variables. Commands below use Bash syntax.

13.12.4 Create the monitoring resources

Choose a supported location. Resource names include a short random suffix so that repeated lab runs do not collide.

```
set -euo pipefail

suffix=$(openssl rand -hex 3)
location="eastus"
rg="obs-lab-${suffix}"
workspace="obs-law-${suffix}"
appinsights="obs-ai-${suffix}"
alert="slo-fast-burn-${suffix}"

az group create   --name "$rg"   --location "$location"   --tags purpose=book-
lab capability=observability owner=reader

az monitor log-analytics workspace create   --resource-group "$rg"   --
workspace-name "$workspace"   --location "$location"

workspace_id=$(az monitor log-analytics workspace show   --resource-group "$rg"
--workspace-name "$workspace"   --query id -o tsv)
```

```
az monitor app-insights component create   --app "$appinsights"   --resource-
group "$rg"   --location "$location"   --workspace "$workspace_id"   --
application-type web

app_id=$(az monitor app-insights component show   --app "$appinsights"   --
resource-group "$rg"   --query id -o tsv)

connection_string=$(az monitor app-insights component show   --app
"$appinsights"   --resource-group "$rg"   --query connectionString -o tsv)

printf 'Resource group: %s
Application Insights: %s
' "$rg" "$app_id"
```

NOTE: The Application Insights CLI command may require the application-insights extension in some Azure CLI installations. If prompted, allow the official extension to install, or create the workspace-based resource through the Azure portal using the same names and workspace.

13.12.5 Build an instrumented minimal API

```
mkdir -p managed-feedback-loop-lab
cd managed-feedback-loop-lab

dotnet new web -n ManagedFeedbackLoop
cd ManagedFeedbackLoop

dotnet add package Azure.Monitor.OpenTelemetry.AspNetCore
```

Replace Program.cs with the following code. The failure endpoint emits a structured error and returns HTTP 503. The slow endpoint provides a latency path for trace inspection.

```
using Azure.Monitor.OpenTelemetry.AspNetCore;

var builder = WebApplication.CreateBuilder(args);

builder.Services
    .AddOpenTelemetry()
    .UseAzureMonitor();

var app = builder.Build();

app.MapGet("/ok", () => Results.Ok(new
{
    status = "ok",
    service = "managed-feedback-loop"
}));
```

```
app.MapGet("/slow", async () =>
{
    await Task.Delay(1500);
    return Results.Ok(new { status = "slow-but-successful" });
});

app.MapGet("/fail", (ILogger<Program> logger) =>
{
    logger.LogError(
        "Synthetic failure for observability lab. Code={Code}",
        "LAB_FAILURE");
    return Results.StatusCode(StatusCodes.Status503ServiceUnavailable);
});

app.Run();
```

Start the application in a separate terminal. The connection string is an ingestion configuration value, not a substitute for production identity and network design.

```
export APPLICATIONINSIGHTS_CONNECTION_STRING="$connection_string"
export OTEL_SERVICE_NAME="managed-feedback-loop"
export OTEL_RESOURCE_ATTRIBUTES="deployment.environment=book-
lab,service.version=1.0.0"

dotnet run --urls http://127.0.0.1:5050
```

13.12.6 Generate baseline and failure traffic

In the first terminal, generate a bounded synthetic population. The commands deliberately create more than ten requests so that the alert rule can use a minimum-volume guard.

```
for i in $(seq 1 50); do
  curl -s http://127.0.0.1:5050/ok > /dev/null
done

for i in $(seq 1 5); do
  curl -s http://127.0.0.1:5050/slow > /dev/null
done

for i in $(seq 1 10); do
  curl -s -o /dev/null http://127.0.0.1:5050/fail
done
```

Telemetry ingestion is asynchronous. Wait several minutes before concluding that instrumentation failed. Check the local process for exporter errors, confirm the connection string, and query a longer window before changing the design.

13.12.7 Define the SLI, SLO, and lab alert policy

The lab uses a request-success SLI. In a production system, health probes, test traffic, rejected client misuse, and other populations would be explicitly classified. For this synthetic service, every request is eligible.

Table 13-13 Lab reliability model

Field	Definition
SLI	Successful eligible requests divided by all eligible requests.
SLO	99.0% request success over a rolling 30-day window. The relaxed objective makes the small lab population easier to reason about.
Error budget	1.0% bad requests over the objective window.
Fast-burn lab condition	At least 10 requests in five minutes and an observed error rate greater than 1.0%.
Decision	Create a severity-2 alert instance. In this lab, no notification endpoint or automatic remediation is attached.
Resolution evidence	The query no longer returns a violating row after only successful traffic is generated and the evaluation window advances.

The production alerting strategy for a 30-day SLO should use burn-rate reasoning and multiple windows. The simplified lab condition is intentionally transparent: it demonstrates the control-plane path without pretending that a five-minute ratio is a complete production policy.

13.12.8 Query the request-success indicator

Application Insights request telemetry is available through the workspace table AppRequests. Run the following KQL query in the Application Insights Logs experience. It groups all requests in the last five minutes and returns a row only when the lab condition is violated.

```
AppRequests
| where TimeGenerated > ago(5m)
| summarize
    Total = sum(ItemCount),
    Failed = sumif(ItemCount, Success == false)
```

```
| extend ErrorRatePct =
    iff(Total == 0, 0.0, 100.0 * todouble(Failed) / todouble(Total))
| extend SuccessRatePct = 100.0 - ErrorRatePct
| where Total >= 10 and ErrorRatePct > 1.0
| project Total, Failed, SuccessRatePct, ErrorRatePct
```

Capture the query text, time window, result, and Application Insights resource ID. A screenshot alone is weaker evidence because it does not preserve the exact query or population definition.

13.12.9 Create the scheduled log search alert

Store the query in a shell variable and create a scheduled query rule. The CLI command is provided by the GA scheduled-query extension for Azure CLI 2.54.0 or later.

```
alert_query="AppRequests
| where TimeGenerated > ago(5m)
| summarize Total=sum(ItemCount), Failed=sumif(ItemCount, Success == false)
| extend ErrorRatePct=iff(Total == 0, 0.0, 100.0 * todouble(Failed) /
todouble(Total))
| where Total >= 10 and ErrorRatePct > 1.0
| project Total, Failed, ErrorRatePct"
```

```
az monitor scheduled-query create   --resource-group "$rg" --name "$alert" --
location "$location"   --scopes "$app_id"   --condition "count
'FailureBudgetBurn' > 0"   --condition-query FailureBudgetBurn="$alert_query"
--evaluation-frequency 1m --window-size 5m   --severity 2 --auto-mitigate true
--description "Fast-burn guard for the request-success SLO lab."   --tags
capability=observability slo=request-success
```

The alert has no action group, so it records an alert instance without notifying a person or executing automation. That is deliberate: notification and remediation require real ownership, identity, and safety design. The lab proves evaluation and evidence first.

13.12.10 Trigger and verify the alert

Generate another failure population after the rule exists, then allow at least two evaluation cycles.

```
for i in $(seq 1 25); do
  curl -s -o /dev/null http://127.0.0.1:5050/fail
done

sleep 180
```

Verify the rule definition:

```
az monitor scheduled-query show   --resource-group "$rg"   --name "$alert"   --
output jsonc
```

Inspect Azure Monitor > Alerts and filter to the lab resource group. Record the alert rule, monitor condition, severity, start time, target resource, and alert ID. If no alert appears, first re-run the KQL query. A query that returns no row cannot trigger the rule. Do not reduce the threshold blindly; verify telemetry freshness, time window, table name, and minimum count.

Generate only successful traffic and wait for the five-minute window to move beyond the synthetic failures. With auto-mitigation enabled, the alert should resolve after the condition no longer returns a violating row.

```
for i in $(seq 1 100); do
  curl -s http://127.0.0.1:5050/ok > /dev/null
done
```

13.12.11 Assemble the evidence packet

Table 13-14 Required evidence

Evidence	What to record
Resource graph	Resource group, workspace, Application Insights resource and alert rule IDs.
Instrumentation contract	Service name, environment and version resource attributes; package version used.
Traffic population	Counts sent to /ok, /slow and /fail, with approximate timestamps.
Reliability model	SLI numerator and denominator, objective, window, error budget and simplified fast-burn condition.
Query evidence	Exact KQL, query time, result row and telemetry freshness.
Alert evidence	Rule JSON, alert instance ID, monitor condition, severity and resolution state.
Negative evidence	No action group or automated remediation was attached.
Cleanup evidence	Resource group deletion command and a subsequent az group exists result of false.

13.12.12 Clean up and verify deletion

Stop the local application, return to the shell where the lab variables are defined, and delete the dedicated resource group.

```
az group delete   --name "$rg"   --yes

az group exists --name "$rg"
# Expected final value: false
```

The resource group is the cleanup unit. Confirm that no alert rule, action group, workspace, data collection endpoint, or other monitoring resource was created outside it. Telemetry retained inside a deleted workspace follows Azure deletion and recovery behavior; do not use sensitive data in a disposable lab.

13.12.13 Multicloud translation of the lab

The lab is intentionally Azure-complete, but its engineering contract is portable. On Alibaba Cloud, instrument the same service with OpenTelemetry and route traces to ARMS Managed Service for OpenTelemetry or Application Monitoring, use SLS and Managed Service for Prometheus as appropriate, then define an ARMS alert policy. On Tencent Cloud, use APM with OpenTelemetry, CLS or TMP, and TCOP alarm policy. On Volcengine, use APMPlus with OpenTelemetry and the relevant log or metric service. Preserve the same service name, SLI definition, time window, minimum population, evidence fields, and cleanup contract.

NOTE: Do not treat a console checkbox as equivalence. Confirm the actual data model, query language, alert evaluation semantics, notification route, retention, billing unit and access boundary on each platform.

13.13 Handoff to CloudDesk FinOps Agent

CloudDesk FinOps Agent may later consume observability and reliability capabilities, but this chapter does not implement its tenant system, Serverless runtime, desktop approval, or cross-tenant isolation. The reusable output is a set of governed capability contracts.

```
observability.signal.collect(source, signalPolicy, destination)
  -> resourceIds, ingestionPath, schema, retention, samplingPolicy
  evidence: collectionHealth, lastSeen, droppedData, costScope

reliability.slo.evaluate(serviceId, sliDefinition, objective, window)
  -> attainedRatio, errorBudgetRemaining, burnRate, decisionState
  evidence: query, population, timeWindow, sourceCompleteness, policyVersion

observability.anomaly.explain(signalScope, timeWindow, changeContext)
```

```
  -> findings, correlatedTraces, candidateChanges, confidence
  evidence: queries, resourceIds, traceIds, alertIds, humanDisposition

resilience.experiment.verify(experimentId, steadyState, hypothesis)
  -> outcome, violatedGuardrails, recoveryTime, recommendations
  evidence: runReport, telemetryQueries, targetScope, executingIdentity
```

A future Agent should receive read-only query capability by default. Creating an alert, suppressing notifications, executing remediation, or running a chaos action requires a separate identity and policy gate. The Agent must cite resource IDs, queries, windows, and alert or experiment records so that a human can reproduce its conclusion.

13.14 Key Takeaways

Observability is a managed feedback loop: system behavior produces evidence, reliability policy selects a decision, and subsequent evidence verifies the result.

Logs, metrics, traces, profiles, and change events are complementary signals. Shared context enables navigation without erasing their different semantics and costs.

OpenTelemetry standardizes instrumentation, context propagation, and collection, but cloud backends still differ in storage, query, access, retention, alerting, and billing.

Azure Monitor exposes observability as governable resources: workspaces, Application Insights, collection rules, diagnostic settings, alerts, action groups, processing rules, dashboards, and resilience tests.

SLIs measure user outcomes; SLOs define objectives; error budgets make tolerated unreliability explicit; burn rate converts consumption into urgency.

An alert should route an accountable decision, not merely announce a threshold crossing.

AIOps can prioritize, correlate, forecast, and assist investigation, but evidence, scope, authority, and verification must remain explicit.

Continuous verification connects releases and configuration changes to SLO evidence while exposure is controlled.

Chaos engineering uses observability as the oracle for a bounded, falsifiable resilience experiment.

Sampling, cardinality, retention, privacy, workspace boundaries, and ingestion limits make observability both a governance and FinOps concern.

Multicloud portability is achieved by preserving the observability contract and evidence model, not by assuming product equivalence.

References and Official Documentation

SOURCE NOTE: Sources were verified on August 24, 2026. Product names, workspace roles, preview or GA status, supported regions, retention, ingestion billing, AIOps feature names, chaos interfaces, and CLI behavior must be rechecked at manuscript freeze.

[1] OpenTelemetry. Observability Primer. Definition of observability and unknown-unknown investigation.

[2] OpenTelemetry. What Is OpenTelemetry? Instrumentation, telemetry signals, and backend export.

[3] OpenTelemetry. Signals. Logs, metrics, traces, and other signal categories.

[4] OpenTelemetry. Context Propagation. Correlating telemetry across process and network boundaries.

[5] OpenTelemetry. Traces. Spans, trace identity, and context propagation.

[6] OpenTelemetry. Collector. Vendor-agnostic receive, process, and export pipelines.

[7] OpenTelemetry. Sampling. Head and tail sampling concepts and trade-offs.

[8] OpenTelemetry Specification. Overview and client architecture.

[9] Google. The Site Reliability Workbook, Chapter 2: Implementing SLOs.

[10] Google. The Site Reliability Workbook, Chapter 5: Alerting on SLOs.

[11] Google. The Site Reliability Workbook, Appendix A: Example SLO Document.

[12] Google. The Site Reliability Workbook, Appendix B: Example Error Budget Policy.

[13] Microsoft Learn. Azure Monitor documentation and service overview.

[14] Microsoft Learn. Azure Monitor enterprise monitoring architecture.

[15] Microsoft Learn. Azure Monitor data sources and data collection methods.

[16] Microsoft Learn. Azure Monitor workspace overview and distinction from Log Analytics workspace.

[17] Microsoft Learn. Metrics in Azure Monitor and managed Prometheus data platform.

[18] Microsoft Learn. Application Insights and Azure Monitor OpenTelemetry guidance for .NET applications.

[19] Microsoft Learn. Overview and types of Azure Monitor alerts.

[20] Microsoft Learn. Create Azure Monitor log search alert rules.

[21] Microsoft Learn. Action groups, common alert schema, and alert processing rules.

[22] Microsoft Learn. Best practices for Azure Monitor alerts.

[23] Microsoft Learn. Secure an Azure Monitor deployment and use Azure Private Link.

[24] Microsoft Learn. Azure Chaos Studio overview, Workspaces, Scenarios, reports, permissions, and current preview limitations.

[25] Azure CLI Reference. az monitor scheduled-query, version 2.54.0 or later.

[26] Alibaba Cloud Documentation. ARMS user guide and observable data storage overview.

[27] Alibaba Cloud Documentation. Managed Service for Prometheus product overview and OpenTelemetry metric ingestion.

[28] Alibaba Cloud Documentation. ARMS billing and ingestion-volume limits.

[29] Tencent Cloud Documentation. Tencent Cloud Observability Platform overview and Application Performance Management features.

[30] Tencent Cloud Documentation. Cloud Log Service and TencentCloud Managed Service for Prometheus.

[31] Tencent Cloud Documentation. APM OpenTelemetry integration and link protocol standard.

[32] Tencent Cloud Documentation. Chaotic Fault Generator overview and experiment workflow.

[33] Volcengine Documentation. APMPlus OpenTelemetry integration, Torch Log Service, Kubernetes observability, and chaos-engineering practices.

Part IV / Chapter 14

FinOps as a Cloud Control Plane

From Cost Visibility and Allocation to Guardrails, Unit Economics, and Verified Savings

FinOps becomes an engineering capability when cost and usage evidence is normalized, allocated, connected to business value, routed through governed decisions, and verified after technical or commercial action. A bill describes what was charged. A FinOps control plane explains who owns the demand, what outcome it produced, which intervention is safe, and whether the intervention created durable value.

In This Chapter

- Why FinOps is a managed control loop rather than a cost-cutting campaign or a billing-dashboard project
- How the FinOps Framework connects understanding usage and cost, quantifying business value, optimizing usage and rates, and managing an effective practice
- Why cloud cost data is delayed, derived, contractual evidence rather than a real-time counter
- How billing scopes, resource scopes, FinOps scopes, actual cost, billed cost, effective cost, and amortized cost answer different questions
- How FOCUS provides a normalized cost-and-usage contract across providers and technology categories
- How allocation, tags, cost centers, shared-cost rules, showback, and chargeback establish accountable ownership
- Why budgets, forecasts, anomalies, quotas, credits, policies, reservations, and savings plans are distinct control mechanisms
- How usage optimization, rate optimization, architecture, workload placement, and demand design form an optimization portfolio
- How unit economics connects resource consumption to business outcomes, including AI-native and agentic workloads

- How Azure Cost Management, Advisor, budgets, exports, allocation, quotas, and commitment instruments form a reference FinOps control plane
- How Azure, Alibaba Cloud, Tencent Cloud, and Volcengine express the same FinOps intent through different billing and resource models
- How to create a resource-group budget, query cost evidence, assemble an auditable evidence packet, and clean up without waiting for a real overspend event

A cloud bill can be perfectly accurate and still be operationally useless. It can tell an organization that it spent more this month without identifying which product feature, tenant, environment, deployment, model, or engineering decision caused the change. It can show a discount without proving that the commitment was well utilized. It can show a low-cost architecture that fails its service objective, or a high-cost architecture that is producing proportionally greater business value. The financial record is necessary, but it is not yet an engineering control system.

FinOps supplies the missing operating model. The FinOps Foundation defines FinOps as an operational framework and cultural practice that maximizes the business value of technology, enables timely data-driven decisions, and creates financial accountability through collaboration among engineering, finance, and business teams [1]. This chapter uses the phrase FinOps control plane as an editorial engineering lens: it is not a replacement for the Foundation definition. It names the managed layer that connects cost and usage evidence to ownership, policy, action, verification, and learning.

The desired loop is not “observe a large number, then cut something.” It is “understand what changed, allocate it to an accountable scope, quantify the resulting value, select a safe technical or commercial response, and verify the result against cost, demand, reliability, security, and product outcomes.” The cloud makes this possible because resources, billing meters, identities, policies, activity logs, budgets, quotas, exports, and recommendations are themselves programmable control-plane objects.

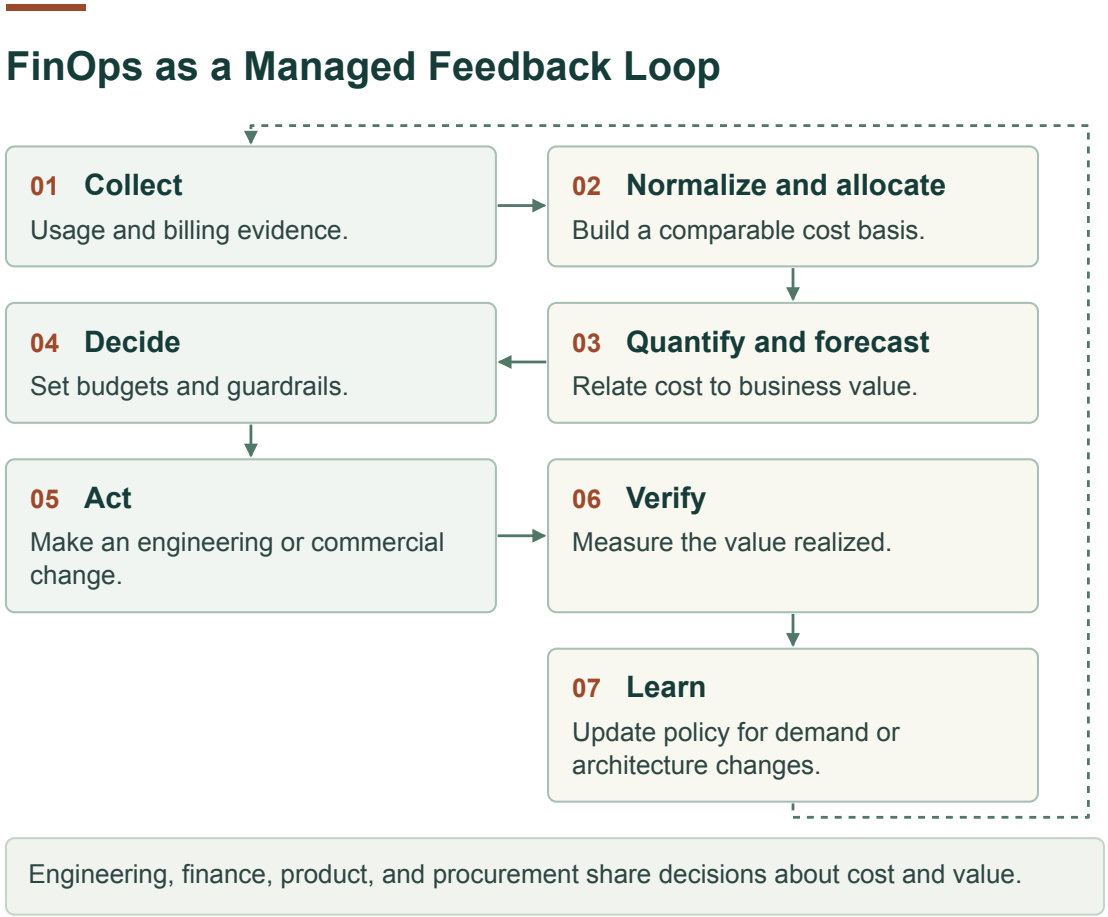


Figure 14-1 FinOps as a managed feedback loop. Cost and usage evidence becomes useful only when it is allocated, connected to value, routed through governed decisions, acted upon, and verified.

14.1 FinOps Is a Control Plane, Not a Cost-Cutting Campaign

Cost reduction can be one outcome of FinOps, but it is not the definition. A team may reduce the monthly bill by removing redundancy, suppressing observability, using a weaker model, or postponing security work. The number falls while business risk rises. Conversely, a growing service may spend more because it serves more customers, improves latency, raises retrieval quality, or replaces expensive manual work. The number rises while unit economics improve. FinOps is therefore concerned with technology value and decision quality, not with minimizing a single total.

The control-plane framing clarifies what must be managed. A control plane does not perform every workload operation. It defines resources, identities, scope, desired state, policy, and evidence. In the same way, a FinOps control plane should not make every engineering

or procurement decision centrally. It provides a common cost-and-value model, exposes decision signals at the point of consumption, and establishes safe routes for distributed teams to act.

Table 14-1 Related financial and engineering responsibilities

Concept	Primary purpose	Typical failure when isolated
Billing and invoicing	Produce contractual charges, credits, taxes, and payment records.	Financially accurate records arrive too late or lack workload context for engineering action.
Cost management	Analyze, group, forecast, alert, allocate, and export cost and usage data.	Dashboards become reports without accountable owners or a change mechanism.
Optimization tooling	Identify possible usage, rate, or architecture improvements.	Recommendations are accepted blindly, remain unactioned, or never have their benefits verified.
FinOps practice	Coordinate engineering, finance, product, procurement, and leadership around technology value.	Cost becomes a specialist concern rather than an architectural and product constraint.
FinOps control plane	Connect evidence, ownership, guardrails, action, and verification as programmable operating policy.	Automation acts without context, or the organization cannot explain why a cost decision was made.

14.1.1 Central enablement, distributed ownership

FinOps should be enabled centrally and practiced where technology decisions are made. A central team can define schemas, allocation rules, commitment policy, data quality, common KPIs, and approved automation. It cannot know every workload's latency target, seasonality, customer contract, model-quality threshold, or risk tolerance. Product and engineering teams must own the relationship between their demand, architecture, cost, and outcomes.

This operating model resembles platform engineering. A platform team provides a paved road: normalized data, self-service views, budget templates, policy modules, recommendation contracts, and auditable change workflows. A workload team consumes that paved road and remains responsible for the correctness of its decisions. Financial accountability is not achieved by sending a monthly spreadsheet. It is achieved when cost signals are visible in design reviews, backlog priorities, release gates, and operational routines.

NOTE: FinOps Foundation materials are licensed under CC BY 4.0. This chapter adapts the Foundation framework as an engineering control-plane model and attributes the source in the references.

14.2 The Operating Model: Understand, Quantify, Optimize, and Manage

The current FinOps Framework describes four business outcomes: understand usage and cost, quantify business value, optimize usage and cost, and manage the FinOps practice [2] [3]. The familiar phases Inform, Optimize, and Operate remain useful as an iterative motion, but the domains make the expected outcomes clearer. An organization must understand evidence, connect it to value, improve the portfolio of technical and commercial decisions, and sustain the mechanisms that make the practice reliable.

Table 14-2 FinOps domains expressed as control-plane outcomes

Domain	Control-plane question	Representative managed capabilities
Understand Usage & Cost	What was consumed, charged, allocated, and changed?	Data ingestion, normalization, allocation, reporting, analytics, anomaly management, data quality.
Quantify Business Value	What outcome did the spending enable, and is it within expectation?	Planning, estimation, forecasting, budgeting, KPIs, benchmarking, unit economics.
Optimize Usage & Cost	Which usage, rate, architecture, or placement decision improves value?	Rightsizing, elasticity, scheduling, lifecycle policy, commitments, licensing, workload placement.
Manage the FinOps Practice	Who owns decisions, which policies apply, and how is evidence preserved?	Governance, education, automation, tooling, procurement coordination, maturity and operating cadence.

The domains are not a waterfall. Allocation quality may improve after an optimization attempt exposes missing ownership. A budget review may reveal that a product metric is an unsuitable denominator. A commitment purchase may uncover a demand forecast error. A well-designed control plane preserves these feedback paths rather than pretending that cost management proceeds through one irreversible sequence.

14.2.1 Cost is a design input, not a post-deployment report

The highest-leverage FinOps decisions often occur before a resource exists. Region choice affects price, latency, egress, availability, and data residency. A managed service may cost more per unit but reduce labor and incident risk. A serverless design can reduce idle capacity while increasing per-request cost or startup latency. A larger model may improve quality enough to reduce retries and human review. The correct decision requires cost, speed, quality, reliability, security, and strategic constraints to be considered together.

This is why FinOps belongs in architecture review, service-level design, and product planning. The variable-cost model of cloud makes architecture economically observable: a design can be measured after release and adapted. However, that benefit appears only when the organization defines an evidence contract before deployment - ownership tags, cost scope, unit metric, budget expectation, quality target, and the signals required to verify a change.

14.3 Cost Data Is a Delayed, Derived, and Contractual Dataset

Operational telemetry and billing evidence describe the same system from different perspectives. A request trace may be available in seconds. The corresponding billable usage may be rated later, adjusted by contract, amortized across time, converted into a billing currency, and exported on a daily or monthly schedule. A real-time dashboard can estimate cost, but the invoice and provider meter records remain the financial source of truth. FinOps engineering must model both speed and authority.

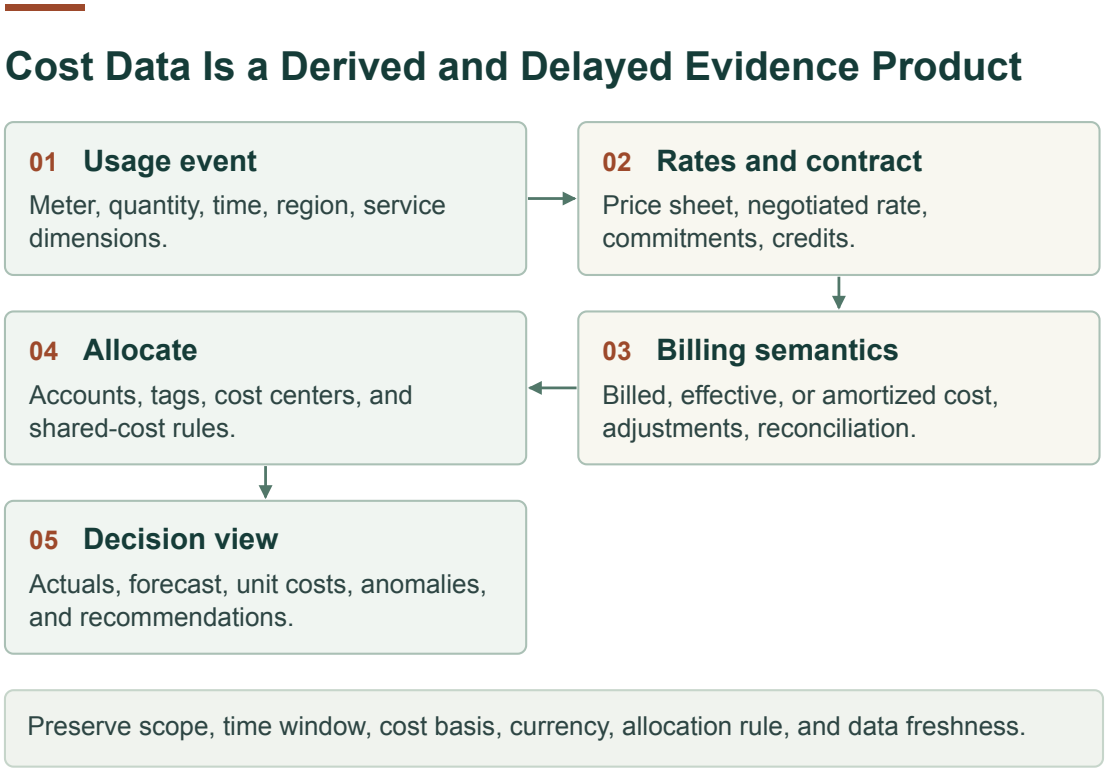


Figure 14-2 Cost-data lineage. Usage is transformed by pricing and contract semantics, billing rules, allocation policy, and reporting views before it becomes decision evidence.

14.3.1 Scope is part of every cost fact

A number without scope is ambiguous. Azure exposes billing scopes such as billing accounts, billing profiles, invoice sections, and enrollment accounts, as well as resource-management scopes such as management groups, subscriptions, and resource groups. A FinOps scope may instead represent a product, business unit, tenant class, environment, AI initiative, or cost center. These structures overlap, but they do not mean the same thing.

Every query, budget, forecast, recommendation, and unit metric should record its scope identifier and scope type. A resource-group budget cannot be assumed to represent a product if shared platform services are charged elsewhere. A subscription total cannot be treated as a tenant cost if multiple tenants share inference, search, storage, or observability resources. Scope design is therefore a modeling decision, not just a filter selection.

14.3.2 Cost semantics answer different questions

Table 14-3 Cost semantics must be chosen explicitly

Cost view	Question it answers	Common misuse
Billed or invoice cost	What amount is contractually charged in the billing period?	Using it as a real-time signal or attributing all shared charges directly to workloads.
Actual or usage cost	What charges are associated with usage as reported for the period?	Ignoring later adjustments, refunds, credits, or commitment treatment.
Amortized or effective cost	What economic cost should be distributed across covered usage and time?	Treating amortization as cash flow or expecting it to match an invoice line exactly.
List or on-demand cost	What would usage cost at a reference list rate?	Presenting the difference from negotiated cost as guaranteed savings.
Forecast cost	What may be spent if current behavior and assumptions continue?	Treating a forecast as a commitment or failing to record model confidence and seasonality.
Unit cost	What does a defined resource or business outcome cost?	Changing the denominator or cost boundary over time without versioning the metric.

The phrase “the cost” should be avoided in design documents and APIs unless the semantic is explicit. A cost query contract should identify currency, cost type, pricing period, usage interval, time zone, tax treatment, credits, commitment amortization, and allocation version. This precision is especially important when an Agent summarizes spend or proposes action: an apparently simple total may combine values that should not be compared.

14.3.3 Data freshness and reconciliation are engineering requirements

Cost data is not guaranteed to arrive at telemetry speed. Azure documentation states that cost and usage data is typically available within 8 to 24 hours and budgets are evaluated every 24 hours [8]. Alibaba Cloud and other platforms likewise document daily or settlement-related delays for some cost views. A control plane must expose freshness, completeness, and settlement status rather than silently returning the latest available number as if it were current.

Operational controls that require sub-hour response should use service telemetry, quotas, rate limits, and workload-specific counters. Cost data can then reconcile the financial impact later. For example, an AI gateway can impose a per-tenant token rate limit in real time while the provider bill confirms the charged meter the next day. The two evidence paths should be correlated, not confused.

14.3.4 FOCUS as a normalized cost-and-usage contract

The FinOps Open Cost and Usage Specification (FOCUS) defines a common billing-data schema aligned with FinOps terminology. Version 1.4 is the current published specification at the source-verification date for this chapter. It supports common capabilities such as chargeback, allocation, budgeting, forecasting, commitment analysis, and invoice reconciliation [4][5].

FOCUS does not erase provider differences. It provides a stable semantic boundary at which provider-specific data can be normalized. Organizations still need to record conformance, unsupported columns, provider extensions, export frequency, currency behavior, and reconciliation rules. A FOCUS dataset is therefore a contract for portable analysis, not proof that every provider represents every concept identically.

Table 14-4 Minimum evidence properties for a portable cost contract

Required data property	Why it matters
Provider and account identity	Prevents the same resource name or SKU from colliding across clouds and billing contexts.
Resource and service identity	Connects billed usage to inventory, ownership, configuration, and lifecycle evidence.
Charge period and usage interval	Separates when usage occurred from when the provider charged or adjusted it.
Billed and effective cost	Supports invoice reconciliation and economically meaningful allocation.

Required data property	Why it matters
Pricing and commitment context	Explains why the charged rate differs from list or on-demand price.
Tags and allocation metadata	Connects provider records to products, teams, environments, tenants, and cost centers.
Currency and exchange treatment	Prevents invalid aggregation across billing currencies or conversion dates.
Freshness and completeness state	Allows automation to decide whether data is adequate for investigation or financial close.

14.4 Allocation: Turning Shared Spend into Accountable Ownership

Cost visibility without allocation produces a shared total and a shared excuse. Allocation assigns cost and usage to an accountable business construct: product, platform, team, cost center, tenant, environment, or initiative. It is the bridge between a provider bill and distributed engineering ownership.

No single allocation mechanism is sufficient. Account hierarchy provides a strong structural boundary but can create administrative fragmentation. Resource groups and projects align with deployment lifecycle but may not match product ownership. Tags are flexible but incomplete and mutable. Cost centers and allocation units can redistribute shared charges but depend on documented rules. Mature allocation combines these mechanisms and versions the rules that transform provider records into management views.

14.4.1 Start with deterministic ownership, then refine shared costs

1. Assign costs that have a direct and stable owner through account, subscription, project, resource group, resource ID, or provider-supported project attribution.
2. Use required tags for business metadata that is not represented by hierarchy, such as product, environment, owner, tenant class, or experiment.
3. Identify shared platform services explicitly instead of hiding them in an unallocated remainder.
4. Choose a shared-cost rule based on an observable driver - requests, storage, active users, CPU time, tokens, or another causal measure - rather than an arbitrary equal split.
5. Preserve the allocation rule, version, effective date, source fields, and unallocated amount so that the result can be audited and reproduced.

An unallocated bucket is not merely a reporting inconvenience. It is a data-quality indicator. A growing unallocated percentage means ownership has weakened, new services are not entering the resource taxonomy, or tags are being created after cost is incurred. The control plane should measure this condition and route it to the platform and finance owners responsible for correcting it.

14.4.2 Showback and chargeback are governance choices

Showback reports cost to owners without changing internal financial accounts. Chargeback transfers or books cost to the responsible unit. Both rely on the same allocation evidence, but they create different incentives and reconciliation requirements. A team can use showback early to improve data quality and build trust before adopting chargeback. Premature chargeback with disputed allocation logic turns a technical data problem into an organizational conflict.

Table 14-5 Allocation mechanisms and their failure modes

Allocation layer	Strong use	Primary risk
Account / subscription / project	Hard administrative and billing boundary; delegated ownership.	Proliferation, duplicated shared services, and fragmented commitments.
Resource group / namespace	Lifecycle, environment, or application grouping.	Resources can be misplaced; cross-group dependencies remain shared.
Tags / labels	Flexible business metadata and multicloud translation.	Missing, inconsistent, retroactively applied, or user-controlled values.
Cost center / allocation unit	Management hierarchy and shared-cost redistribution.	Rule opacity, double allocation, or weak causal basis.
Usage driver	Causal sharing by requests, seats, storage, tokens, or compute time.	Instrumentation gaps and disputes over the correct denominator.
Manual adjustment	Exceptional corrections and financial close.	Loss of reproducibility and growing spreadsheet dependence.

14.5 Guardrails: Budgets, Forecasts, Anomalies, Quotas, Credits, Policies, and Commitments

Cloud consoles often place multiple controls near one another, which encourages incorrect equivalence. A budget tracks spending against a plan. A forecast estimates likely future spending. An anomaly detects unusual behavior. A quota limits or allocates service capacity. A credit or spending limit represents commercial balance or account constraints. A policy controls permitted resource configurations. A reservation or savings plan changes the rate applied to eligible usage. Each instrument addresses a different failure mode.

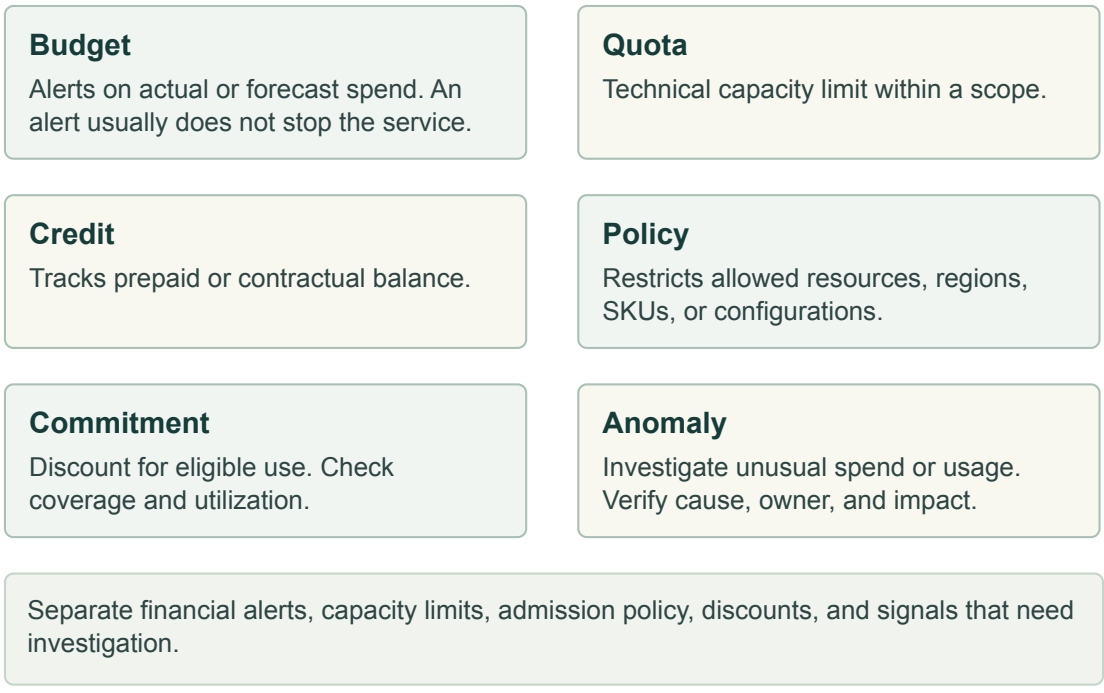


Figure 14-3 Budgets, quotas, credits, policies, commitments, and anomaly signals are complementary guardrails, not interchangeable limits.

14.5.1 A budget is an accountability route, not a kill switch

Azure budgets can alert on actual or forecast thresholds, but the budget does not stop resources [8]. Cost data latency also means a workload can continue to incur spend before the evaluation observes a threshold. This behavior is correct for many production systems: abruptly shutting down a database or API at a financial threshold can create greater business loss than the excess cost.

A useful budget therefore defines an accountable route. At an early threshold, the owner receives context and a forecast. At a higher threshold, a review is required. At a critical threshold, an approved action group or workflow may restrict non-production demand, pause experiments, require explicit approval for new resources, or invoke a workload-specific safe-mode. The response is architecture and policy, not an assumption that a billing object will enforce runtime safety.

14.5.2 Quota is capacity governance

A quota is a service allowance or limit within a subscription, region, model, or resource type. Azure states that requesting a quota increase has no cost; cost is created by resource usage, not by the quota itself [15]. Quota can prevent an uncontrolled scale-out or block a legitimate deployment. Quota monitoring belongs in FinOps because capacity and spend interact, but quota is not a budget.

AI PaaS makes this distinction especially important. Microsoft Foundry quotas and limits are defined at subscription, region, model, or deployment-type scopes rather than as tenant-level financial enforcement [18]. An application that serves multiple tenants must implement its own attribution, rate policy, and tenant guardrails even when the underlying model platform provides provider-level quota.

14.5.3 Commitments optimize rate, not architecture

Reservations and savings plans can reduce the effective rate for predictable eligible usage. Azure explicitly describes reservations as a billing discount that does not affect the runtime state of resources [16]. A reservation does not create a virtual machine, improve utilization, or correct an oversized architecture. It can make waste cheaper while locking in a financial commitment.

Commitment decisions should be based on a demand forecast and risk policy. Coverage asks how much eligible usage receives the discount. Utilization asks how much of the purchased commitment is consumed. Flexibility asks whether the workload may change region, service, family, or operating model. Procurement, finance, and engineering must evaluate these dimensions together.

14.5.4 Anomaly detection is an investigation signal

Anomaly systems compare current behavior with expected patterns and surface unusual change. Azure Cost Management identifies anomalies daily using normalized usage, while provider documentation commonly warns that anomaly results can be delayed, incomplete, or advisory [9][26]. A cost anomaly may be a bug, an attack, a missing lifecycle policy, a new customer, a successful campaign, a price change, or a billing adjustment.

The control-plane response is to enrich the anomaly with ownership, deployment change, resource inventory, utilization, business demand, and service-level evidence. An anomaly should open an investigation with a hypothesis and evidence, not automatically delete the resource that appears expensive.

14.6 Optimization Is a Portfolio of Technical and Commercial Decisions

FinOps optimization should be organized as a portfolio rather than a list of discount tips. The FinOps Framework distinguishes usage optimization, rate optimization, architecture and workload placement, licensing, and related capabilities. Alibaba Cloud's Well-Architected cost framework similarly separates billing-method optimization, resource-usage optimization, and architecture optimization [21]. These levers interact and should be evaluated against workload value and risk.

Table 14-6 Optimization is broader than rightsizing

Optimization lever	Examples	Required verification
Demand and product design	Reduce unnecessary calls, duplicate data, unused features, excessive retries, or low-value model fan-out.	Outcome volume and quality remain acceptable; demand assumptions are explicit.
Usage efficiency	Rightsize, schedule shutdown, autoscale, release idle resources, apply storage lifecycle, tune retention.	Utilization improves without violating latency, availability, recovery, or security objectives.
Architecture and placement	Choose managed vs self-managed, region, serverless vs provisioned, batch vs online, cache, queue, model route.	Total cost of ownership and operational risk improve over a representative workload period.
Rate optimization	Reservations, savings plans, committed use, resource packages, negotiated rates.	Coverage and utilization meet policy; lock-in and forecast risk are understood.
Operational discipline	Detect orphaned resources, expire experiments, enforce ownership, automate cleanup and review.	Controls are reproducible, audited, and do not remove resources still required by a workload.
Supply and platform design	Provide approved SKUs, golden paths, default budgets, cost-aware APIs, and reusable policies.	Teams can choose cost-effective defaults without bypassing reliability or compliance.

14.6.1 A recommendation is a hypothesis with a change contract

A recommendation should include the affected scope and resource IDs, evidence window, current configuration, utilization or demand evidence, expected savings semantic, confidence, risk, dependencies, required approval, proposed change, rollback condition, and verification plan. "Save $500 by resizing" is not an engineering recommendation. It is an estimate without a safety case.

The recommendation must also state what is excluded. A provider estimate may consider compute meter savings but not migration labor, software licensing, reservation breakage, data transfer, operational toil, quality loss, or incident exposure. FinOps engineering makes these boundaries visible so that a human or policy engine can compare alternatives honestly.

14.6.2 Optimize the system, not a line item

Local cost optimization can increase system cost. Compressing logs can reduce ingestion while increasing CPU and delaying incident diagnosis. Moving data to a cheaper region can increase egress and latency. Selecting the smallest model can increase retries, tool calls, human escalation, or failed tasks. Removing redundancy can lower compute spend while exhausting the error budget.

A safe control plane therefore joins cost evidence with observability and reliability. Every material optimization should identify the SLOs and business metrics that must remain within bounds. This makes FinOps an extension of architecture and operations rather than a separate monthly review.

14.7 Unit Economics: Cost per Outcome, Not Cost per Resource

A resource total explains where money was charged. Unit economics explains whether the system converts technology into value efficiently. The FinOps Foundation distinguishes resource-efficiency metrics, such as cost per gigabyte or token, from business unit metrics, such as cost per tenant, transaction, case, or outcome [4]. Rising total cost may be healthy when successful outcomes grow faster; falling total cost may be harmful when quality or adoption collapses.

A unit metric is valid only when the numerator and denominator are defined and versioned. The numerator must specify the cost boundary: direct cloud charges only, amortized commitments, shared platform allocation, support, observability, engineering labor, or other fully loaded costs. The denominator must represent a meaningful and verified unit rather than an easily manipulated activity count.

```
unit_cost(scope, period, version) =
    allocated_effective_cost(scope, period, allocation_version)
    / verified_outcome_units(scope, period, outcome_definition_version)
```

14.7.1 Resource units are stepping stones

Resource metrics such as cost per vCPU-hour, gigabyte stored, million tokens, search query, or function invocation are useful because engineers can influence them directly. They reveal rate changes, utilization, and service efficiency. They do not by themselves prove business value. A cheaper token is not valuable if the application uses three times as many tokens to complete the same task.

14.7.2 AI-native systems need a fully loaded boundary

AI application cost often appears to be the model API bill. In reality, one user request can trigger prompt processing, retrieval, embedding, reranking, content safety, multiple model calls, tool execution, queueing, storage, network transfer, observability, evaluation, and human review. Microsoft Foundry documentation explicitly notes that Foundry cost is only a subset of the overall application or solution cost [17].

AI unit economics requires a fully loaded cost boundary

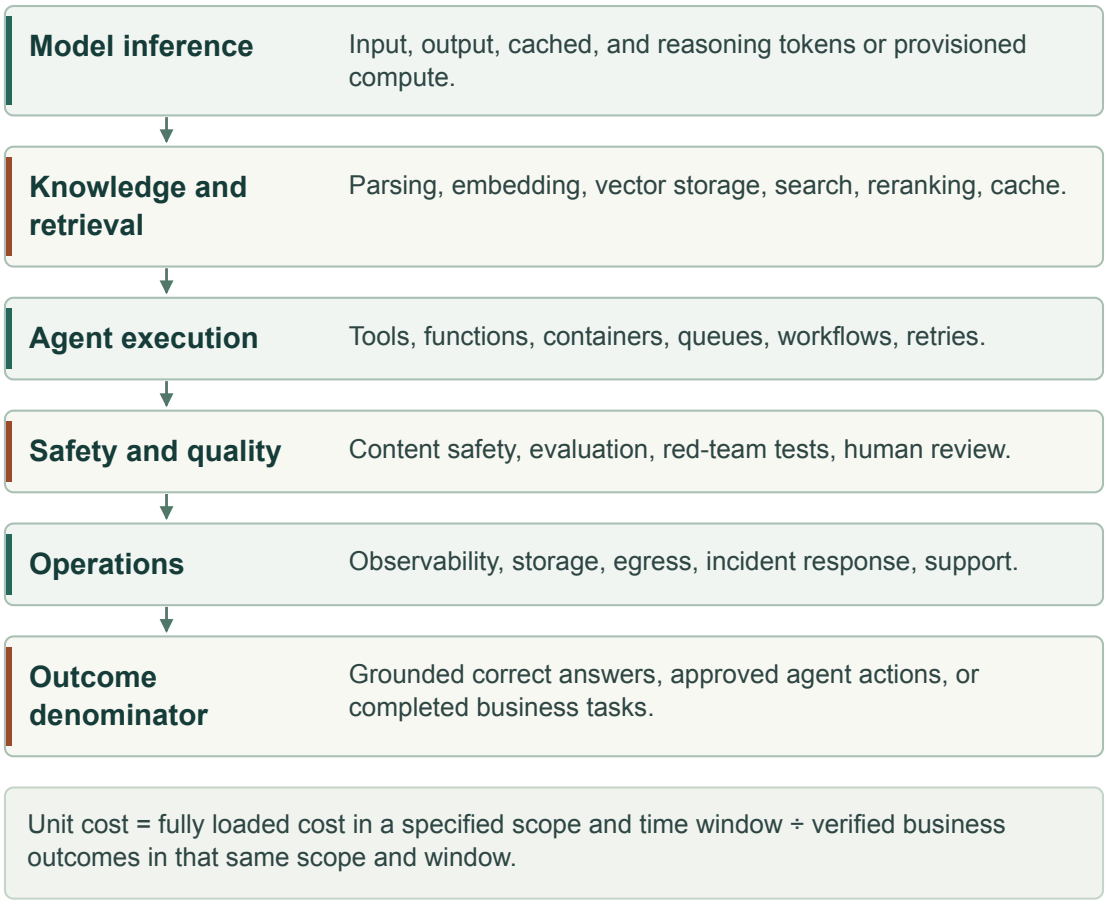


Figure 14-5 Fully loaded AI unit economics. Model inference is only one layer; the denominator should be a verified business outcome.

Table 14-7 AI unit metrics should move toward verified outcomes

Candidate AI unit	Useful when	Failure mode
Cost per input/output token	Comparing meters, routing, caching, or prompt behavior.	Optimizing token price while ignoring task success, retries, or downstream work.
Cost per model request	Calls have comparable payload and quality requirements.	One request may contain radically different token counts or reasoning effort.
Cost per grounded answer	RAG quality and citation requirements are explicit.	Counting answers that are unsupported, incorrect, or rejected by users.

Candidate AI unit	Useful when	Failure mode
Cost per approved agent action	Tool execution is auditable and produces a verified action.	Ignoring rejected, rolled-back, duplicated, or unsafe actions.
Cost per resolved case	AI replaces or assists a measurable workflow outcome.	Attributing all resolution value to AI without considering human work or case complexity.
Cost per tenant / active user	Shared SaaS cost needs showback or pricing analysis.	Averages conceal heavy users and cross-tenant subsidy.

14.7.3 Unit economics and SLOs should be reviewed together

A useful operating view combines unit cost with throughput, quality, latency, availability, and error-budget consumption. A routing change may reduce cost per answer but increase P95 latency. A larger model may raise direct cost while increasing first-pass success and lowering cost per resolved case. A retrieval cache may reduce model and search cost while increasing staleness risk. Explicit trade-offs prevent FinOps from optimizing away the product.

14.8 Azure as the Reference FinOps Control Plane

Azure is the reference platform because it exposes billing hierarchy, resource hierarchy, cost analysis, allocation, budgets, anomaly signals, exports, recommendations, commitment instruments, quotas, policy, identity, and activity evidence as connected control-plane capabilities. The exact portal labels and API versions are volatile; the architecture is more stable.

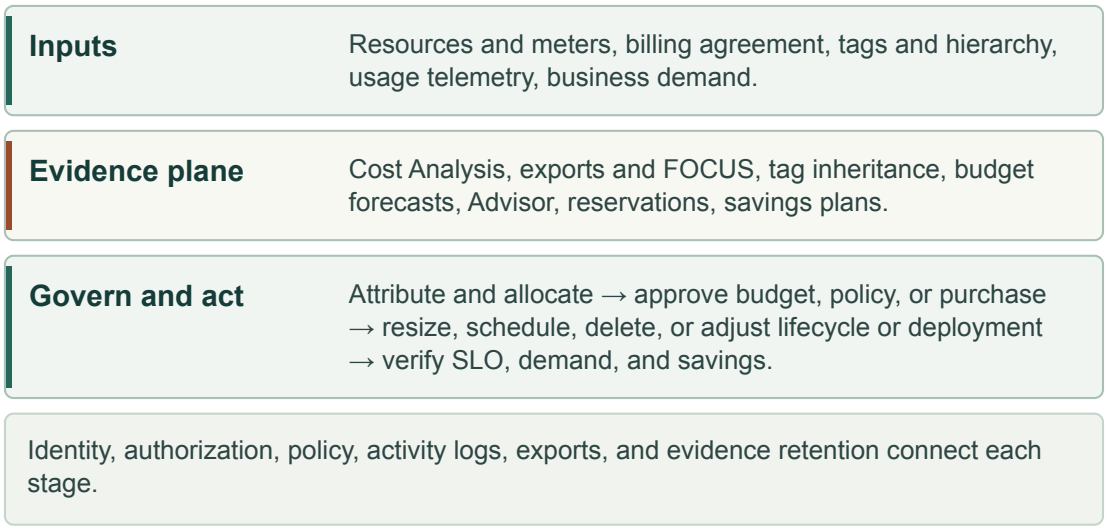


Figure 14-4 Azure reference FinOps control plane. Cost Management converts resource and billing evidence into decision views, while governance and engineering workflows act and verify.

14.8.1 Billing scope, resource scope, and FinOps scope

Azure cost APIs operate at supported scopes. A billing account or billing profile supports financial reconciliation; a management group or subscription supports enterprise governance; a resource group supports a workload or experiment lifecycle. Cost Management query operations accept these scope identifiers directly. The FinOps model may then regroup the records into product, tenant, cost center, or initiative scopes using tags and allocation logic.

Permissions must be granted at the scope needed for the task. A cost reader does not need permission to resize or delete resources. A budget author does not automatically need access to application data. A recommendation executor should use a separate identity, explicit approval policy, and auditable change route. Least privilege in FinOps is a separation between observation, decision, and action.

14.8.2 Cost Analysis, queries, and exports

Cost Analysis provides interactive views of actual and amortized costs by resource, service, tag, and other dimensions. Programmatic queries allow a defined scope, time frame, aggregation, grouping, and filter to be encoded in an API request. Scheduled exports write cost datasets to storage for larger-scale analytics and reconciliation. Azure Well-Architected guidance recommends daily exports to ADLS Gen2 and a FOCUS template for provider-neutral analysis [10].

Interactive views answer investigation questions. Exports support repeatable pipelines and historical analysis. The control-plane design should preserve the query or export definition alongside the resulting evidence. A screenshot without scope, time window, semantic, or filter cannot be reproduced.

14.8.3 Allocation, tag inheritance, and shared costs

Azure Cost Management can apply tag inheritance and split shared costs [9]. Tag inheritance fills cost-reporting metadata and should not be confused with modifying the underlying resource tags. Allocation rules are management transformations over cost data, not changes to the deployed resource graph. The rule must therefore be versioned and reconciled separately from the infrastructure.

Tags should be governed near resource creation through policy and IaC. Cost allocation can compensate for gaps, but it should not become the permanent source of business identity. The most reliable design combines stable hierarchy, required tags, and explicit shared-cost rules.

14.8.4 Budgets, forecasts, anomalies, and alerts

Budgets are resources with lifecycle, scope, period, amount, and notification definitions. They can evaluate actual or forecast thresholds. Cost alerts provide a unified view of budget, credit, spending-quota, and other alert categories. Anomaly detection supplies a separate daily investigation signal. The FinOps control plane should route these signals to owners and, where appropriate, to an approved workflow rather than sending undifferentiated email.

14.8.5 Advisor, commitments, and quotas

Azure Advisor incorporates cost recommendations for subscriptions and resource groups. These should be treated as provider-generated hypotheses: validate workload behavior, dependencies, SLOs, commitment state, and business demand before action. Reservations and savings plans address rate. Quotas address capacity. Neither replaces usage optimization or architecture review.

The distinction is operationally important. Purchasing a reservation can reduce rate while leaving an oversized resource unchanged. Raising quota can enable growth without creating immediate cost. Lowering an internal tenant limit can control application demand even when the provider quota remains high. Each decision belongs to a different control surface.

14.8.6 The FinOps evidence chain

Table 14-8 FinOps objects should produce auditable evidence

Control-plane object	Evidence it should preserve
Cost query or saved view	Scope, time frame, cost semantic, currency, grouping, filters, API version, data timestamp.
Export	Definition, schema/FOCUS version, schedule, storage destination, completion state, reconciliation status.
Budget	Scope, amount, period, threshold type, recipients or action group, deployment identity, creation event.
Allocation rule	Source scope, target scope, driver, formula, version, effective date, unallocated remainder.
Recommendation	Resource IDs, evidence window, current state, expected saving, risk, assumption, owner, expiry.
Commitment	Purchase scope, term, eligible services, coverage, utilization, exchange/refund policy, owner.
Optimization change	Approved plan, change identity, deployment record, rollback condition, post-change evidence.

14.9 AI PaaS and Agent Cost Control

AI PaaS adds new meters and new amplification paths. Token consumption is variable, but token price is only one component. A reasoning model can generate hidden reasoning tokens before a response. A user question can branch into multiple retrieval queries, model calls, safety checks, and tools. An evaluation pipeline can replay thousands of examples. An agent can retry a failed tool or remain in a loop. A knowledge refresh can re-embed an entire corpus. FinOps must model the expansion from one business request to the full resource graph.

14.9.1 Meter semantics and deployment economics

Foundry models may use pay-per-token, provisioned throughput, managed compute, or provider-specific capacity models. The meter and deployment type determine cost behavior. HTTP status alone may not determine billing; Microsoft recommends reconciling service metrics with Cost Management meter data and treating invoice and meter records as authoritative [17].

A cost-aware AI gateway should record model, deployment, prompt and completion tokens, cached or reasoning behavior where exposed, latency, retry count, tenant or project, request purpose, quality result, and the downstream resources invoked. This application

evidence enables rapid controls and allocation; provider billing later reconciles the charged cost.

14.9.2 Provider quota is not tenant isolation or tenant budget

Microsoft Foundry documentation states that quotas and limits are not enforced at the tenant level; the highest-level restrictions are associated with Azure subscription and model-related scopes [18]. A multitenant application must therefore implement its own tenant attribution, rate limits, budgets, approval rules, and cross-tenant isolation. Provider quota protects service capacity. Application policy protects customer and product boundaries.

14.9.3 AI cost fan-out

Table 14-9 AI-native cost amplification paths

Fan-out source	Cost risk	Control-plane response
Long context and unbounded history	Token growth compounds with every turn.	Summarize, retrieve selectively, cap context, version memory policy.
Agent planning and retries	One user request creates many model and tool calls.	Max steps, retry budget, timeout, loop detection, per-task cost envelope.
Retrieval expansion	Multi-query, reranking, embedding, and large top-k increase cost.	Query plan limits, cache, retrieval-quality evaluation, cost per grounded answer.
Evaluation and red-teaming	Large datasets replay model calls repeatedly.	Evaluation budget, sampling, scheduled windows, result reuse, approval for full runs.
Tool execution	Functions, containers, SaaS APIs, egress, and human review add cost.	Tool scope, price metadata, dry-run, approval, result and charge evidence.
Observability	High-cardinality prompts, traces, and payloads increase ingestion and privacy risk.	Sampling, redaction, retention tiers, cost attribution, evidence minimums.

14.9.4 Cost envelopes for AI work

A cost envelope is an application-level policy attached to a task, tenant, project, evaluation run, or agent instance. It can include maximum tokens, model classes, retrieval queries, tool calls, elapsed time, retries, estimated currency amount, and required approval beyond a threshold. The envelope is enforced in real time using application telemetry and provider quota. It is reconciled later using billing evidence.

```
ai_cost_envelope = {
  "tenantId": "tenant-a",
  "taskType": "finops-investigation",
  "maxModelCalls": 6,
  "maxToolCalls": 8,
  "maxInputTokens": 60000,
  "maxOutputTokens": 12000,
  "allowedModels": ["economy", "standard"],
  "estimatedCostLimit": 3.00,
  "approvalRequiredAbove": 1.50,
  "qualityGate": "grounded-and-cited"
}
```

This envelope is not an Azure budget. It is an application control that can react immediately. The later FinOps pipeline allocates the provider charges to the same tenant and task, compares estimated and billed cost, and updates the routing and estimation policy.

14.10 From Potential Savings to Verified, Realized Value

Provider recommendations and internal analytics usually report potential savings. Potential savings are conditional estimates: they depend on demand, eligible hours, rates, scope, and the assumption that a change can be made safely. Realized savings are observed after the approved change, normalized for demand and service outcomes, and reconciled to cost evidence.

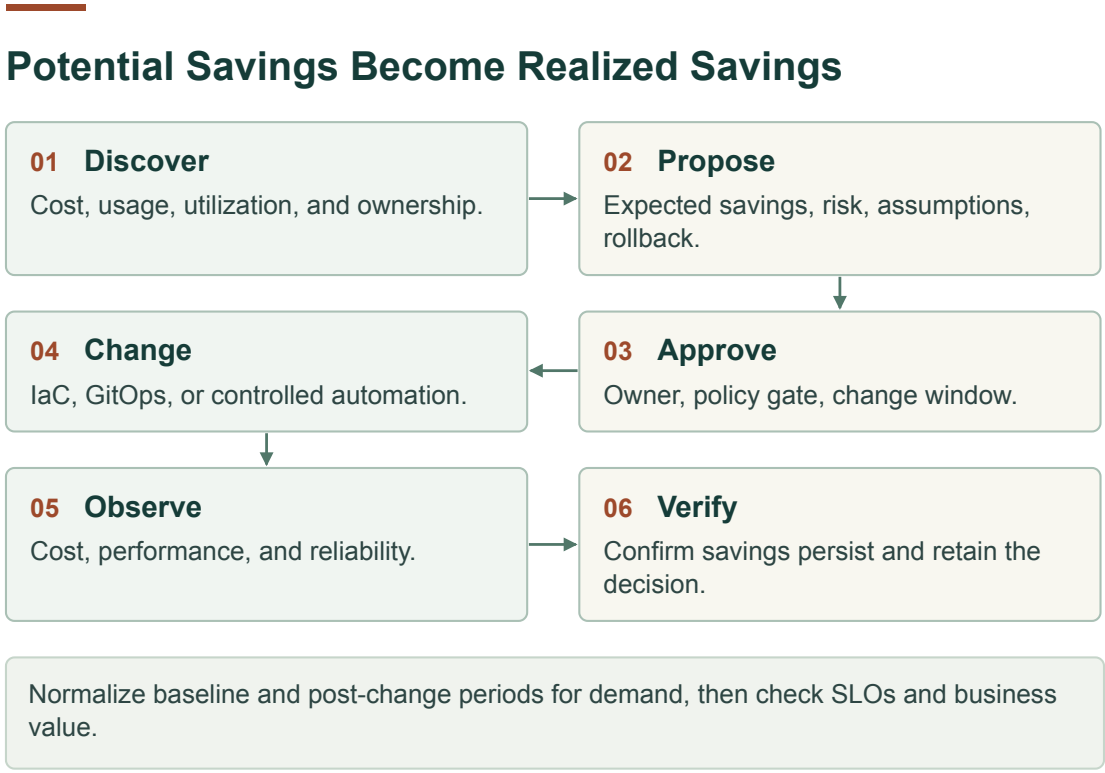


Figure 14-6 Recommendation lifecycle. Savings are realized only after an approved change is observed and verified against cost, demand, and service evidence.

14.10.1 Establish a baseline that can survive scrutiny

A baseline should cover a representative period and record demand, seasonality, deployment version, rate, commitment coverage, region, SLOs, and known incidents. Comparing a holiday week with a normal week or a failed service with a healthy service can manufacture a false saving. When demand changes, use a unit metric or normalized model rather than a raw total.

```
realized_saving =
    expected_cost_for_post_change_demand_under_baseline_policy
    - observed_effective_cost_after_change

valid only if:
    SLOs, quality gates, security controls, and scope definitions remain
satisfied
```

14.10.2 Preserve the change and verification evidence

Table 14-10 Evidence required to claim realized savings

Stage	Required evidence
Detection	Cost query, utilization, owner, scope, data freshness, related deployment and demand changes.
Proposal	Expected savings semantic, model, assumptions, risk, excluded costs, expiry, rollback plan.
Approval	Approver, policy, separation of duties, maintenance window, action hash or change ID.
Execution	IaC/GitOps deployment, identity, activity log, old and new configuration.
Observation	Cost, unit metric, SLO, quality, incident, user demand, and commitment coverage.
Verification	Realized saving, variance from estimate, side effects, durability period, lesson and policy update.

This evidence model allows optimization to become an auditable engineering practice. It also supports automation without granting an Agent unbounded authority. Read-only analysis and recommendation can be broadly delegated. Mutating action requires a separate identity, policy gate, approval, bounded blast radius, and rollback.

14.11 Multicloud Translation: Preserve the Contract, Not the Console

The major cloud platforms expose similar FinOps intentions through different account hierarchies, cost centers, billing datasets, budget systems, commitment products, and APIs. Portability comes from preserving the evidence and decision contract - scope, cost semantic, allocation, owner, guardrail, action, and verification - rather than assuming that product names or console pages are equivalent.

Table 14-11 Multicloud FinOps capability map

Capability	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Cost analysis	Cost Management / Cost Analysis	Cost Analysis in Expenses and Costs	Cost Explorer / Cost Analysis	Cost Analysis in Billing Center
Allocation	Hierarchy, tags, tag inheritance, allocation rules	Cost Centers, hierarchy, tags, shared-cost allocation	Cost Allocation Tags and Cost Allocation Units	Cost Units and allocation rules

Capability	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Budgets	Cost budgets with actual/forecast notifications	Cost and usage/coverage budgets with alerts	Budget Management	Billing and budget capabilities vary by account/product; verify current support
Anomaly / monitoring	Daily cost anomaly signals and cost alerts	Rule-based monitoring and AI-assisted anomaly detection	Budget and unsettled-usage monitoring; verify current anomaly features	Cost/usage views and alerts; verify current anomaly features
Exports	Scheduled exports, including FOCUS templates	Bill and cost detail exports/APIs	Bills to COS and export center; FOCUS 1.0 with documented gaps	Bill-data storage to TOS and billing APIs
Commitments	Reservations and savings plans	Savings plans, subscriptions, resource packages	Reserved or subscription discounts by service	Savings plans and subscription options by service
Quota	Azure Quotas and service-specific quota alerts	Service quotas and quota management	Service quotas and usage limits	Quota Center and quota alerts
Identity	Azure RBAC and Cost Management roles	RAM and Billing permissions	CAM and Billing Center roles	IAM and Billing Center permissions

14.11.1 Alibaba Cloud

Alibaba Cloud Cost Centers can organize costs in a hierarchy and allocate shared costs. Budget Management supports actual and forecast alerts and also exposes usage or coverage-oriented budgets for selected commitment scenarios. Cost Analysis and anomaly-related features provide investigation signals. Alibaba Cloud's Well-Architected framework explicitly describes FinOps as a recurring cycle and separates billing-method, resource-usage, and architecture optimization [21].

14.11.2 Tencent Cloud

Tencent Cloud provides Cost Explorer, Budget Management, cost allocation tags, Cost Allocation Units, allocation bills, export mechanisms, and a published FinOps cost-management solution. Tencent Cloud also publishes a FOCUS 1.0 dataset and documents conformance gaps. That disclosure is important: a normalized format should be accompanied by a provider-specific conformance statement and reconciliation test, not assumed to be complete.

14.11.3 Volcengine

Volcengine Billing Center documentation includes Cost Units, cost-analysis capabilities, bill-data storage to TOS, billing APIs, Quota Center, quota alerts, and service-specific savings plans or subscription models. Product availability and English documentation coverage vary, so a production comparison should record the account type, region, product, billing mode, and documentation verification date.

14.11.4 The portable FinOps contract

```
finops_record = {
  provider, billing_account, resource_scope, finops_scope,
  usage_interval, charge_period, currency, cost_semantic,
  billed_cost, effective_cost, allocation_rule_version, owner,
  business_unit_metric, data_freshness, source_evidence
}

finops_decision = {
  trigger, hypothesis, expected_value, risk, policy, approver,
  action, rollback, verification_window, realized_value_evidence
}
```

This contract can be implemented with FOCUS and provider extensions, but it should remain explicit in the application and data model. It is the boundary that allows multicloud tools, dashboards, and Agents to reason about comparable intent without erasing provider semantics.

14.12 Capability Check C3: Create a Budget and Capture Control-Plane Evidence

This independent capability check creates an Azure resource-group budget as a managed control-plane object. It does not require an application, a previous lab, or a real overspend event. The objective is to prove lifecycle, scope, notification policy, programmatic cost query, evidence collection, and cleanup.

14.12.1 Learning objectives

- Create a dedicated resource group with ownership and cost-center tags.
- Deploy a monthly budget at resource-group scope by using Bicep.
- Define both actual and forecast notifications.
- Retrieve the budget resource and cost query through Azure Resource Manager APIs.
- Record data-freshness limitations and explain why the budget is not a runtime stop.
- Assemble an evidence packet and delete the budget and resource group explicitly.

14.12.2 Prerequisites and cost boundary

- An Azure subscription in which you can create a resource group and a Cost Management budget at that scope.
- Azure CLI and Bicep installed, with an authenticated account in the intended tenant and subscription.
- An email address used only for the lab notification definition.
- No billable workload is required. The budget itself does not create usage cost. The cost query may return no rows or delayed data.

NOTE: Do not lower a budget and deliberately create expensive resources in order to force an alert. Azure cost data is typically delayed 8 to 24 hours and budgets are evaluated every 24 hours. The evidence target is the control-plane configuration and query contract, not a manufactured bill.

14.12.3 Define the lab variables

```
export sub=$(az account show --query id -o tsv)
export location=eastus
export rg=book-finops-c3
export budget=book-finops-monthly
export owner_email="you@example.com"

az group create \
  --name "$rg" \
  --location "$location" \
  --tags owner="$owner_email" costCenter=book-labs environment=learning
```

Record the resource-group ID and tags. The resource group is both the management scope and the cleanup boundary for this capability check.

```
export scope="/subscriptions/$sub/resourceGroups/$rg"
az group show --name "$rg" --query "{id:id,tags:tags,location:location}" -o
jsonc
```

14.12.4 Create the Bicep budget

Create a file named budget.bicep. The current resource type shown here is Microsoft.Consumption/budgets@2024-08-01. The start date must be the first day of a month and should be adjusted when the lab is run.

```
targetScope = 'resourceGroup'

@description('Budget resource name')
param budgetName string
```

```
@description('Monthly amount in the billing currency')
param amount int = 10

@description('Notification recipient')
param contactEmail string

@description('First day of a month in UTC')
param startDate string

resource monthlyBudget 'Microsoft.Consumption/budgets@2024-08-01' = {
  name: budgetName
  properties: {
    amount: amount
    category: 'Cost'
    timeGrain: 'Monthly'
    timePeriod: {
      startDate: startDate
    }
    notifications: {
      Actual_80_Percent: {
        enabled: true
        operator: 'GreaterThanOrEqualTo'
        threshold: 80
        thresholdType: 'Actual'
        contactEmails: [contactEmail]
        contactGroups: []
        contactRoles: []
      }
      Forecast_100_Percent: {
        enabled: true
        operator: 'GreaterThanOrEqualTo'
        threshold: 100
        thresholdType: 'Forecasted'
        contactEmails: [contactEmail]
        contactGroups: []
        contactRoles: []
      }
    }
  }
}

output budgetResourceId string = monthlyBudget.id
```

Deploy the template at resource-group scope. Replace the sample start date with the first day of the current or next month in UTC.

```
az deployment group create \
  --resource-group "$rg" \
  --name finops-budget-deployment \
  --template-file budget.bicep \
  --parameters \
      budgetName="$budget" \
      contactEmail="$owner_email" \
      startDate="2026-09-01T00:00:00Z" \
  --query "properties.outputs" -o jsonc
```

14.12.5 Retrieve and verify the budget resource

Query the budget directly through the Resource Manager endpoint. This captures the actual control-plane representation rather than relying on a portal screenshot.

```
az rest \
  --method get \
  --url
"https://management.azure.com${scope}/providers/Microsoft.Consumption/budgets/${
budget}?api-version=2024-08-01" \
  --output jsonc
```

Verify the following fields: resource ID, scope, amount, category, time grain, start date, actual threshold, forecast threshold, enabled state, and notification recipient. Do not treat the presence of a recipient as proof that an email has been sent; notification requires a later evaluation and threshold condition.

14.12.6 Query current month-to-date cost

Create query.json. This request uses the current Azure Cost Management query API version at the source-verification date. The API returns evidence for the defined scope; it may return no row when the resource group has no rated usage or when cost data has not arrived.

```
cat > query.json <<'JSON'
{
  "type": "Usage",
  "timeframe": "MonthToDate",
  "dataset": {
    "granularity": "None",
    "aggregation": {
      "totalCost": {
        "name": "PreTaxCost",
        "function": "Sum"
      }
```

```
    }
  }
}
JSON

az rest \
  --method post \
  --url
"https://management.azure.com${scope}/providers/Microsoft.CostManagement/query?
api-version=2026-06-01" \
  --headers "Content-Type=application/json" \
  --body @query.json \
  --output jsonc
```

Record the request body, response columns, response rows, currency where returned, and retrieval time. A zero or empty result is valid evidence for this lab. The important result is that the query contract identifies the scope, period, type, and aggregation.

14.12.7 Capture deployment and activity evidence

```
az deployment group show \
  --resource-group "$rg" \
  --name finops-budget-deployment \
  --query "
{state:properties.provisioningState,timestamp:properties.timestamp,outputs:prope
rties.outputs}" \
  -o jsonc

az monitor activity-log list \
  --resource-group "$rg" \
  --offset 1d \
  --status Succeeded \
  --max-events 50 \
  -o jsonc
```

Activity-log coverage can differ for extension resources and operations. The Resource Manager budget response and deployment record are the primary lifecycle evidence. Record any absence in the activity-log query rather than inventing an event.

14.12.8 Explain the control behavior

Table 14-12 Required interpretation of the budget evidence

Question	Expected explanation
Will the budget stop resources at 100%?	No. It evaluates spending and routes notifications or configured action. Safe runtime enforcement requires a separate workload-specific control.
Will the alert fire immediately?	Not necessarily. Cost data is typically delayed and budget evaluation is periodic.
Is the budget amount a quota?	No. It is a financial tracking target; quota governs capacity or service limits.
Does a quota increase create cost?	No. Cost follows actual resource usage, although higher quota permits more capacity to be consumed.
Can the result be used for financial close?	Only after the chosen cost semantic and provider data are reconciled with invoice or authoritative meter records.

14.12.9 Assemble the evidence packet

Table 14-13 Capability Check C3 evidence packet

Artifact	Required content
01-scope.json	Subscription ID, resource-group ID, tags, owner, region, capture time.
02-budget.bicep	Version-controlled budget definition.
03-deployment.json	Deployment state, timestamp, output budget resource ID.
04-budget-resource.json	Budget amount, period, actual and forecast notifications, scope, API version.
05-cost-query.json	Scope, request body, response, currency/columns, retrieval time, freshness note.
06-activity-log.json	Relevant lifecycle evidence or an explicit statement that no matching event was returned.
07-interpretation.md	Budget-vs-quota explanation, expected evaluation delay, and safe-response policy.
08-cleanup.json	Delete responses and proof that the resource group no longer exists.

14.12.10 Optional quota extension

Use Azure Quotas to inspect a quota relevant to a service you already use, such as regional compute vCPU quota. Record current usage, limit, scope, and whether the quota is adjustable. Optionally create a usage alert. Do not request an increase solely for the lab. Explain that the quota has no cost by itself and that the budget and quota protect different boundaries.

14.12.11 Clean up and verify deletion

Delete the budget explicitly, then delete the resource group. Explicit budget deletion makes the extension-resource lifecycle visible even if deleting the scope would remove it.

```
az rest \
  --method delete \
  --url
"https://management.azure.com${scope}/providers/Microsoft.Consumption/budgets/${
budget}?api-version=2024-08-01"

az group delete --name "$rg" --yes

az group exists --name "$rg"
# Expected final value: false
```

Confirm that the budget GET request now returns not found and that the resource group no longer exists. No billable service was required, but cost and billing records can persist according to provider retention and financial-record requirements.

14.12.12 Multicloud translation of the capability check

On Alibaba Cloud, create a cost budget or other supported budget at a defined account or cost-center scope and capture actual/forecast alert configuration. On Tencent Cloud, use Budget Management with a Cost Allocation Unit or tag-based ownership view. On Volcengine, use the available Billing Center budget or usage-control capability and pair it with Cost Units and Quota Center where appropriate. For every platform, preserve the same evidence contract: scope, owner, amount or limit, period, threshold semantics, notification route, data latency, API or configuration version, and cleanup.

NOTE: Product availability, account permissions, English documentation, budget behavior, and notification channels change. Re-verify each platform before manuscript freeze and before running the lab.

14.13 Handoff to CloudDesk FinOps Agent

CloudDesk FinOps Agent may later consume the managed capabilities described in this chapter, but this chapter does not implement its tenant system, Serverless runtime, desktop approval, cross-tenant isolation, or autonomous remediation. The reusable output is a small set of bounded capability contracts.

```
finops.cost.read(scope, timeWindow, costSemantic, grouping, filters)
  -> values, currency, sourceTimestamp, freshness, queryEvidence

finops.budget.create(scope, amount, period, thresholds, notificationRoute)
  -> budgetResourceId, deploymentEvidence, evaluationSemantics

finops.anomaly.detect(scope, baselinePolicy, evidenceWindow)
  -> anomaly, owner, likelyDrivers, confidence, investigationEvidence

finops.saving.verify(changeId, baseline, postChangeWindow, unitMetric,
sloPolicy)
  -> realizedSaving, variance, sideEffects, verificationEvidence
```

The Agent should receive read-only cost and resource-query capability by default. Budget creation, commitment purchase, quota change, resource modification, or automated shutdown requires a separate identity and approval route. Every output should cite scope, time window, cost semantic, allocation version, freshness, and supporting resource or billing evidence. Tenant-specific budgets and rate policies must be enforced in the application even when the cloud provider exposes only subscription- or account-level controls.

14.14 Key Takeaways

FinOps is an operational framework and cultural practice for maximizing technology value; cost reduction is one possible outcome, not the definition.

The FinOps control-plane framing connects evidence, ownership, value, guardrails, action, verification, and learning without centralizing every workload decision.

Cloud cost data is delayed, derived, contractual evidence. Scope, cost semantic, currency, time window, allocation rule, and freshness belong to every cost fact.

FOCUS provides a normalized cost-and-usage contract, but provider extensions, conformance gaps, and reconciliation remain explicit.

Allocation turns shared provider charges into accountable ownership. Hierarchy, tags, cost centers, and usage drivers should be combined and versioned.

Budgets, quotas, credits, policies, commitments, and anomaly signals protect different boundaries. A budget is not a kill switch, and a quota is not a price.

Optimization is a portfolio of demand, usage, architecture, placement, rate, and operational decisions evaluated against reliability, security, quality, and product outcomes.

Unit economics connects technology spend to value. AI unit cost must include model, retrieval, agent execution, safety, quality, operations, and a verified outcome denominator.

Azure exposes a reference FinOps control plane through Cost Management, allocation, budgets, anomaly signals, exports, Advisor, commitments, quotas, identity, and activity evidence.

Potential savings become realized savings only after a governed change is observed, normalized for demand, and verified against cost and service objectives.

Multicloud portability comes from preserving the FinOps evidence and decision contract, not from assuming equivalent console labels.

An AI Agent may analyze broadly with read-only capability, but financial or technical mutations require separate identity, approval, bounded scope, rollback, and verification.

References and Official Documentation

NOTE: SOURCE NOTE: Sources were verified on August 24, 2026. Pricing, discounts, exchange/refund policies, product names, portal navigation, API versions, export formats, FOCUS conformance, billing delays, budget permissions, anomaly models, quotas, commitment eligibility, and AI model meters must be rechecked at manuscript freeze and before each lab run.

[1] FinOps Foundation. What Is FinOps? Definition updated March 2026.

[2] FinOps Foundation. FinOps Framework Overview: principles, scopes, personas, domains, capabilities, and phases.

[3] FinOps Foundation. FinOps Domains: Understand Usage & Cost, Quantify Business Value, Optimize Usage & Cost, and Manage the FinOps Practice.

[4] FinOps Foundation. Capability: Unit Economics.

[5] FinOps Open Cost and Usage Specification. FOCUS Specification 1.4.

[6] FinOps Foundation. Domain: Quantify Business Value.

[7] FinOps Foundation. Domain: Optimize Usage & Cost and Capability: Usage Optimization.

[8] Microsoft Learn. Tutorial: Create and Manage Budgets in Microsoft Cost Management.

[9] Microsoft Learn. Overview of Billing and Cost Management: anomaly detection, allocation, Advisor, and alerts.

[10] Microsoft Learn. Azure Well-Architected Framework: Architecture Strategies for Collecting and Reviewing Cost Data.

[11] Microsoft Learn. Microsoft.Consumption/budgets@2024-08-01 Bicep and ARM Template Reference.

[12] Microsoft Learn. Azure Cost Management Query - Usage REST API, version 2026-06-01.

[13] Microsoft Learn. Azure Cost Management Exports REST API and FOCUS export guidance.

[14] Microsoft Learn. Use Cost Alerts to Monitor Usage and Spending.

[15] Microsoft Learn. Azure Quotas Overview and Quota Usage Alerts.

[16] Microsoft Learn. What Are Azure Reservations? and guidance for reservations and savings plans.

[17] Microsoft Learn. Plan and Manage Costs for Microsoft Foundry.

[18] Microsoft Learn. Microsoft Foundry Models Quotas and Limits.

[19] Microsoft Learn. Manage and Increase Quotas for Microsoft Foundry Resources.

[20] Microsoft Learn. Managed Compute and Provisioned Throughput for Microsoft Foundry Models.

[21] Alibaba Cloud Documentation. Well-Architected Framework: Cloud Cost Management Framework.

[22] Alibaba Cloud Documentation. Cost Centers and Shared Cost Allocation.

[23] Alibaba Cloud Documentation. Budget Management.

[24] Alibaba Cloud Documentation. Cost Analysis and Cloud Cost Anomaly Detection.

[25] Alibaba Cloud Documentation. Savings Plans and Utilization/Coverage Budgets.

[26] Tencent Cloud Documentation. Cost Management Introduction and Cost Explorer.

[27] Tencent Cloud Documentation. Budget Management.

[28] Tencent Cloud Documentation. Cost Allocation Units, Cost Allocation Tags, and Cost Allocation Bill.

[29] Tencent Cloud Documentation. FOCUS 1.0 and FOCUS 1.0 Conformance Gaps.

[30] Tencent Cloud Documentation. Saving Bills to COS and Export Center.

[31] Tencent Cloud Documentation. FinOps Cost Management Solution.

[32] Volcengine Documentation. Billing Center, Cost Units, and Cost Analysis.

[33] Volcengine Documentation. Configure Bill Data Storage in TOS and Billing APIs.

[34] Volcengine Documentation. Quota Center, Quota Usage, and Quota Alerts.

[35] Volcengine Documentation. Savings Plans and Service Billing Modes.

Part IV / Chapter 15

Security Posture, Compliance, and Organizational Governance

Turning Organizational Expectations into Enforceable Resource State and Auditable Evidence

Security posture is the managed relationship among organizational intent, actual cloud state, risk context, evidence, exceptions, and verified remediation—not a dashboard score alone.

In This Chapter

- Why security posture is a continuously managed relation between desired controls and actual cloud state
- How organizational hierarchy determines the inheritance, scope, ownership, and evidence of governance controls
- Why RBAC, policy, resource locks, posture management, audit, and compliance mappings solve different problems
- How policy as code moves safely from control intent through audit, limited rollout, enforcement, and review
- How preventive, detective, corrective, compensating, and recovery controls work together
- Why continuous compliance requires repeated evaluation, drift awareness, and evidence freshness
- Why a cloud compliance dashboard is not a legal certification or a complete control assessment
- How CSPM adds business context, exposure, vulnerability, and attack-path prioritization to configuration findings

- How exceptions, exemptions, waivers, compensating controls, and expiry should be governed
- How to implement and verify a deny policy in an independent Azure lab
- How Alibaba Cloud, Tencent Cloud, and Volcengine expose comparable governance intents with different enforcement contracts
- How CloudDesk FinOps Agent can reuse governance checks, remediation planning, and posture explanation capabilities

A cloud security dashboard can show hundreds of green controls and still leave the organization unable to answer a basic question: which business requirement is being satisfied, by which control, at what scope, with what evidence, under whose authority, and for how long? The dashboard is a view. Governance is the system that creates the view, assigns owners, constrains change, records exceptions, and decides what happens when the view turns red.

Security posture is therefore not a product page or a score. It is the current relationship between the organization's declared control intent and the actual state of resources, identities, networks, data paths, software, AI services, dependencies, and operating processes. That relationship changes whenever a resource is created, a role is assigned, a public endpoint is opened, a policy is updated, a model deployment is replaced, a new attack path appears, or an exception expires.

Compliance is related but different. A compliance claim connects an external or internal requirement to control objectives, implementations, assessments, evidence, ownership, and risk decisions. Cloud platforms can automate important parts of this chain: they can deny disallowed configurations, evaluate existing resources, record control-plane changes, surface posture findings, and remediate selected states. They cannot decide an organization's legal obligations, prove controls they cannot observe, or accept residual risk on behalf of management.

This chapter treats security posture and compliance as control-plane engineering. Azure provides the reference model through management groups, Azure Policy, Activity Log, resource locks, remediation tasks, exemptions, and Defender for Cloud. Alibaba Cloud, Tencent Cloud, and Volcengine are compared by governance intent rather than by product-name symmetry. The central question is always the same: how does an organization turn an expectation into an enforceable, assessable, explainable, and reversible cloud control?

The chapter thesis

Security posture is not a dashboard alone. It is the combined effect of identity, network, configuration, encryption, policy, audit evidence, resource locks, posture findings, exceptions, and remediation workflows.

Security posture is a managed feedback loop, not a static score

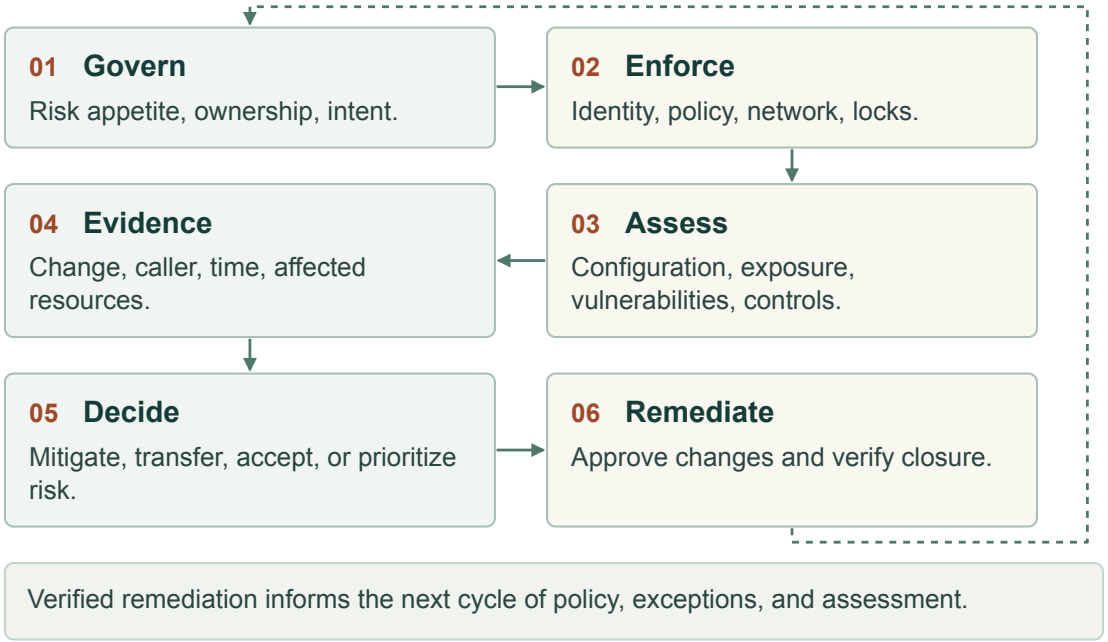


Figure 15-1 Security posture is maintained through a governance feedback loop that ends in verified change and renewed assessment.

15.1 Security Posture Is a Managed Relation

A useful security-posture model compares two evolving objects. The first is the intended control profile: risk appetite, mandatory controls, approved architectures, identity boundaries, data handling rules, service-level expectations, exception policy, and evidence requirements. The second is the observed estate: resources, configurations, roles, public exposures, vulnerabilities, dependencies, control-plane changes, data locations, and active exceptions. Posture is the evaluated difference between them, adjusted for business context and control effectiveness.

The model explains why a single percentage is never sufficient. Two organizations with the same number of misconfigurations may have very different risk because one finding exposes a low-value sandbox while another connects an internet-facing identity to a production data store. A posture system that counts failed checks without resource criticality, reachability, privilege, exploitability, or business impact can produce a precise score and a poor decision.

Posture must also distinguish absence of evidence from evidence of absence. A resource can appear compliant because it is outside the evaluator's inventory, because the provider does not expose the relevant property, because the assessment is stale, because the control is manual, or because an exemption suppresses the result. A trustworthy posture report carries coverage, freshness, scope, and exception metadata together with the finding.

```
Security posture(t) = evaluate(intended controls, observed cloud state,
                               risk context, evidence quality, active
exceptions,
                               control effectiveness, and change history)
```

Term	Primary question	Output	What it does not prove
Governance	Who decides, owns, prioritizes, funds, reviews, and accepts risk?	Roles, policy intent, decision rights, review cadence	That every technical control is correctly implemented
Policy	Which resource states or actions are allowed, denied, modified, or audited?	Machine-evaluable rule and assignment	That the rule covers every risk or legal requirement
Compliance state	Does the observed resource match the assigned rule at the evaluation time?	Compliant, non-compliant, exempt, unknown, or equivalent provider state	Legal certification or complete control effectiveness
Security posture	What risks emerge from configuration, exposure, vulnerability, identity, data, and dependency context?	Prioritized findings and recommendations	That risk has been accepted or remediated
Audit evidence	Who changed what, when, where, and with what result?	Control-plane event and correlated artifacts	That the resulting state is secure or compliant
Control assessment	Is the control implemented, operating as intended, and producing the desired outcome?	Assessment result with evidence and residual gaps	That future resource state will remain unchanged

Posture is time-bound

Every posture statement needs a scope and an “as of” time. Evidence without freshness metadata cannot support a reliable risk decision.

15.2 Governance Begins with Risk and Decision Rights

NIST Cybersecurity Framework 2.0 added GOVERN as a cross-cutting function to establish and monitor cybersecurity risk-management strategy, expectations, and policy. The significance for cloud engineering is practical: configuration rules should not originate as isolated console settings. They should trace to organizational objectives, stakeholder expectations, risk appetite, legal and contractual requirements, and named decision makers. [1]

A cloud governance program therefore needs a chain of ownership. Executive or risk leadership sets tolerance and escalation expectations. Security architecture translates them into control objectives. The cloud-platform team implements reusable guardrails, landing zones, policy definitions, and evidence pipelines. Workload owners design within those guardrails and own application-specific risk. Compliance and internal audit evaluate evidence independently. FinOps and service management add economic and operational consequences. No single team can legitimately perform every role.

The organization must also decide which controls are global and which are contextual. Disabling anonymous public access may be a global baseline. Restricting deployment to two regions may apply only to a regulated data class. Requiring customer-managed encryption keys may be necessary for one workload and an unjustified operational burden for another. Governance is not the maximum number of rules; it is the smallest coherent set of controls that reliably expresses the organization’s risk decisions.

Role	Primary responsibility	Decision that should not be silently delegated
Risk or executive leadership	Set risk appetite, escalation thresholds, and accountable owners	Acceptance of material residual risk
Security architecture	Translate risks and requirements into control objectives and approved patterns	Certification of a workload based only on a dashboard score
Cloud platform team	Implement hierarchy, landing zones, reusable policies, evidence, and remediation services	Business approval for workload-specific exceptions

Role	Primary responsibility	Decision that should not be silently delegated
Workload owner	Classify data, select architecture, operate controls, respond to findings, and fund remediation	Unbounded exception or transfer of responsibility to the provider
Compliance / legal	Interpret applicable obligations and review evidence coverage	Technical claims not supported by evidence or service contracts
Internal audit / assurance	Independently assess design and operating effectiveness	Day-to-day administration of the controls being assessed
FinOps / service management	Connect control choices to cost, capacity, service value, and operational consequences	Security risk acceptance solely because a control is expensive

A control without an owner is a finding in waiting

Every assigned policy, posture recommendation, exception, remediation task, and evidence-retention rule should have an accountable owner and a review cadence.

15.3 Organizational Hierarchy Is a Governance Mechanism

Cloud governance scales through hierarchy. Azure management groups provide a scope above subscriptions so that policy and access assignments can inherit to child scopes. Alibaba Cloud Resource Directory, Tencent Cloud Organization, and Volcengine Enterprise Organization provide related multi-account structures. The hierarchy is not merely an accounting tree. It determines where guardrails are attached, which teams can create exceptions, how audit evidence is aggregated, and how delegated administrators operate across accounts. [15] [16] [26] [33] [41]

A common mistake is to reproduce the human organization chart exactly. Business units change frequently, while security, connectivity, regulatory, and platform requirements are often more stable. Azure landing-zone guidance recommends grouping workloads by common security, compliance, connectivity, and feature needs rather than creating management groups solely for regions. The broader principle applies across providers: design hierarchy around governance inheritance and operating boundaries, then map business ownership through subscriptions, accounts, projects, resource groups, and tags. [16]

High-level scope creates power and danger. A deny policy assigned near the root can prevent an entire organization from deploying a required resource. A permissive child assignment usually cannot override an inherited explicit deny. A multi-account audit trail can

centralize evidence, but it also becomes a privileged asset whose destination, retention, and access require independent protection. Root-scope changes therefore need stronger review, test scopes, rollback plans, and separation of duties than application-level changes.

Organizational governance inherits downward and reports upward

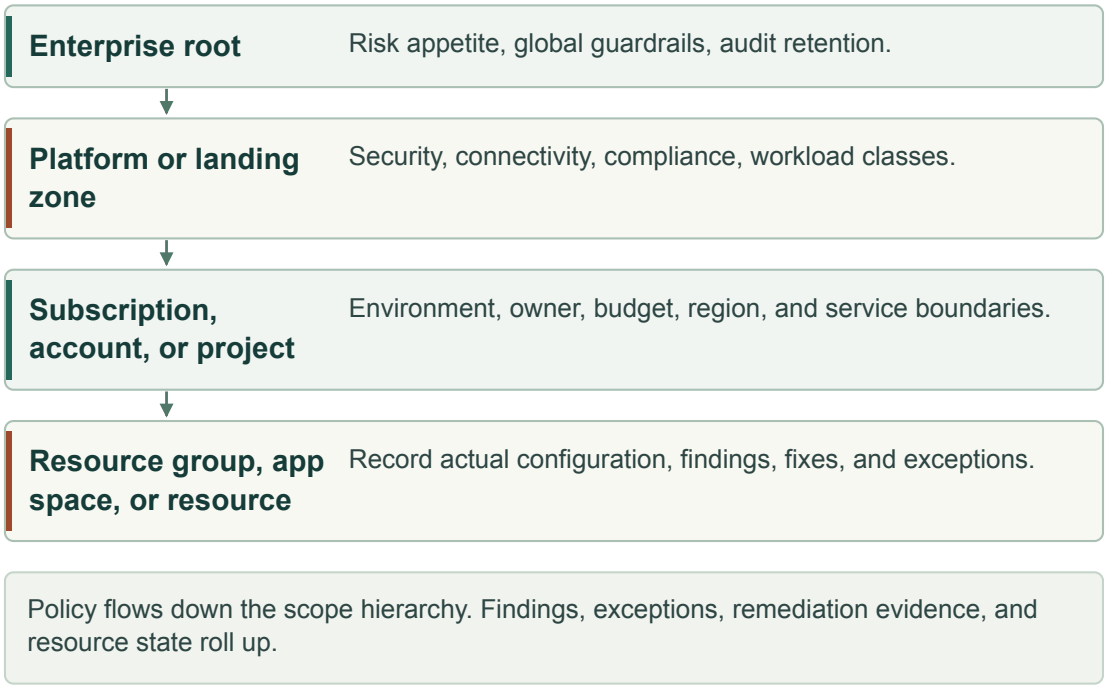


Figure 15-2 Policy and control intent inherit downward, while compliance state, findings, and audit evidence roll upward.

Hierarchy layer	Appropriate governance content	Avoid
Enterprise / root	Minimal global deny boundaries, organization-wide audit, identity prerequisites, ownership requirements	Large untested policy bundles and workload-specific settings
Platform / landing-zone	Connectivity, security baseline, region and service classes, shared monitoring, delegated administration	Mixing incompatible workload risk classes in one scope
Subscription / account / project	Environment, workload class, budget, data location, service ownership, platform variation	Using one subscription as the only isolation and ownership mechanism

Hierarchy layer	Appropriate governance content	Avoid
Resource group / application scope	Application-specific policy, resource locks, evidence, remediation, and temporary exceptions	Exceptions with no parent visibility or expiry
Individual resource	Narrow waiver, lock, diagnostic setting, critical-resource protection	Long-lived one-off configuration that bypasses IaC and central evidence

15.4 Policy as Code: From Intent to Enforceable State

Policy as code expresses an organizational rule as a versioned, reviewable, testable, and deployable artifact. In Azure Policy, a definition describes conditions and an effect; an assignment binds the definition or initiative to a scope; parameters adapt reusable rules; exemptions record tracked exceptions; and remediation tasks can correct selected existing resources. The stable abstraction is broader than any one provider: control objective, rule, scope, effect, evaluation, evidence, exception, and remediation. [6] [7] [8] [12] [13]

Policy must be distinguished from authorization. RBAC or an equivalent identity policy determines what a principal is allowed to call. Resource policy constrains which states can exist or how requests are treated. A user may have permission to create a storage account and still be denied because the requested resource lacks required tags or enables public access. Conversely, a resource can satisfy a configuration policy while an overly privileged identity remains able to misuse it. Both controls are necessary.

Policy also differs from resource locks. A lock is a coarse control against deletion or modification at a scope, often used to reduce accidental change. It does not express the full desired configuration, prove that the resource is secure, or replace authorization. Posture management differs again: it discovers and prioritizes risk using configuration, vulnerability, identity, exposure, and dependency context. Audit records who performed a control-plane action. These mechanisms complement each other; they are not interchangeable.

A safe policy lifecycle begins with impact analysis. Microsoft recommends starting with audit effects before enforcement, validating existing and new resources, checking for false positives, testing remediation templates, and using limited scopes before broad rollout. The same engineering discipline should be applied to rule sets in other clouds. A broken guardrail can create a wider outage than the misconfiguration it was intended to prevent. [9] [10]

Control-plane mechanism	Primary function	Typical output	Key limitation
Identity and access policy	Grant or bound actions for principals	Allow or deny authorization decision	Does not continuously assess resource configuration
Resource configuration policy	Audit, deny, modify, or require resource state	Compliance state and enforcement result	Coverage depends on supported resource properties and evaluation semantics
Service control policy / permission boundary	Limit the maximum actions available in member accounts	Organization-level action boundary	Usually does not grant permissions or prove resulting resource state
Resource lock / deletion protection	Prevent or reduce accidental modification or deletion	Blocked control-plane operation	Authorized administrators may remove the lock; data-plane behavior may be unaffected
CSPM / posture assessment	Discover, contextualize, and prioritize security risk	Finding, score, recommendation, attack path	A recommendation is not an enforcement decision or risk acceptance
Audit trail	Record management and selected data-plane events	Caller, target, operation, result, time, correlation	Evidence of an action is not proof of secure final state
Compliance mapping	Relate checks to control or framework clauses	Coverage view and evidence references	Automated checks cover only a subset of a standard

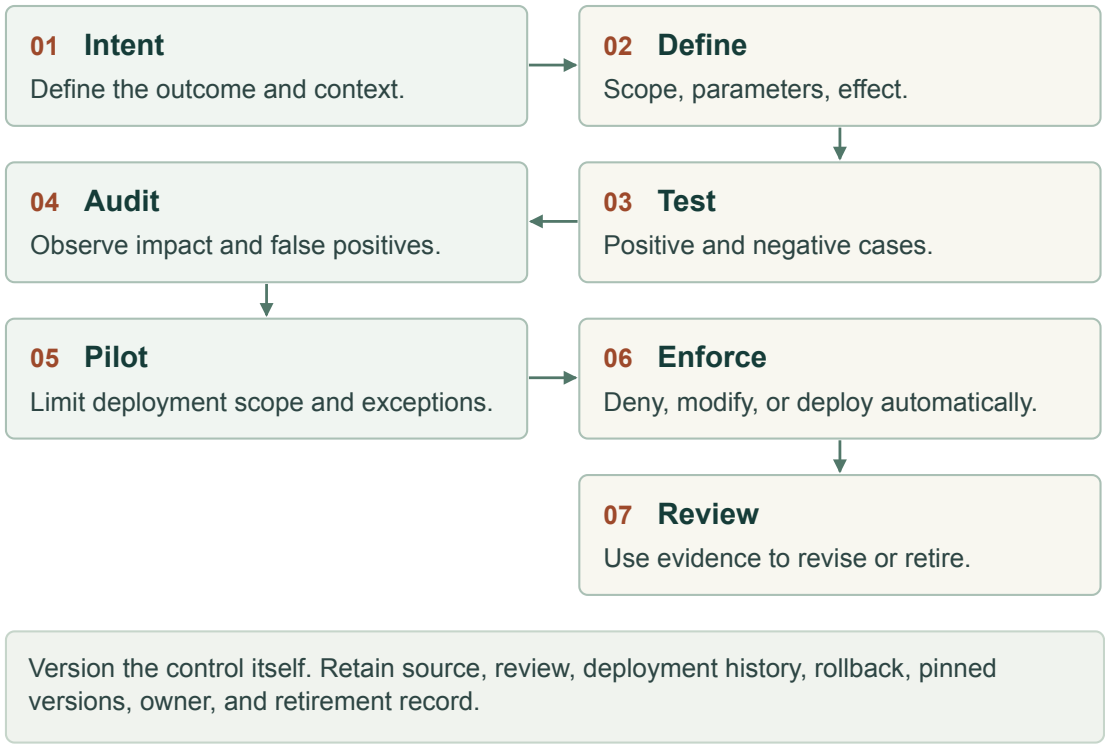


Figure 15-3 Policy as code needs its own safe deployment lifecycle, evidence, rollback, and retirement.

```
Policy artifact = control objective + rule + scope + effect + parameters
                  + version + owner + exception path + evidence contract
                  + safe deployment plan + retirement condition
```

15.5 Control Types Work as a System

Cloud controls are often classified as preventive, detective, corrective, compensating, and recovery controls. The categories describe the timing and function of a control, not its importance. A deny policy prevents a disallowed create or update request. A configuration rule detects an existing state. A remediation task modifies a non-compliant resource. An exemption plus an alternative safeguard can act as a compensating control. Backup, rollback, and restore mechanisms support recovery when prevention fails or a corrective change causes harm.

The strongest design normally combines types. Consider public object storage. A preventive control denies new public access. A detective rule finds existing buckets or accounts that are public. A corrective workflow disables the setting after an approved plan. An audit trail records who changed the resource. A compensating control may temporarily restrict source networks while an application is migrated. A recovery control restores a previous configuration if the change breaks a legitimate integration.

Control type also affects authority. Preventive controls execute in the request path and can block delivery, so their rollout requires strong testing. Detective controls can be broadly deployed with lower operational risk, but they create an obligation to triage findings. Corrective automation needs an execution identity, least privilege, concurrency protection, and rollback. Compensating controls and exceptions require explicit risk ownership and expiry. Recovery controls require tested artifacts, not merely documented intentions.

Control type	Cloud implementation patterns	Evidence to retain	Failure mode
Preventive	Deny policy, service control policy, network boundary, admission rule, mandatory identity requirement	Blocked request, policy and assignment version, scope, caller, correlation ID	False positive blocks legitimate delivery or emergency response
Detective	Configuration audit, CSPM finding, inventory query, vulnerability scan, drift detection	Finding state, resource snapshot, evaluator version, time, coverage	Stale or incomplete inventory creates false assurance
Corrective	Modify/deploy remediation, workflow, automation runbook, approved IaC change	Plan, identity, approval, change result, verification, rollback state	Automatic fix causes outage or broadens access
Compensating	Alternative technical control, restricted exception, added monitoring, manual review	Rationale, control equivalence, owner, expiry, review evidence	Temporary workaround becomes permanent and invisible
Recovery	Rollback, restore, immutable backup, prior policy version, break-glass procedure	Recovery test, artifact integrity, recovery time, decision record	Untested recovery fails when the control change causes impact

15.6 Configuration Drift and Continuous Evaluation

Configuration drift is the difference between the approved or declared state and the observed state. Drift can be unauthorized, accidental, emergency-driven, provider-introduced, or the result of a legitimate change that was never returned to source control. A

posture system should not assume that every difference is malicious, but it must make the difference visible and attributable.

Continuous compliance does not mean that every property is evaluated at every millisecond. NIST describes continuous monitoring as maintaining awareness at a frequency sufficient to support risk decisions. Cloud configuration services usually combine event-driven evaluation with periodic scans, and the result can lag behind a just-completed change. Evidence must therefore include the evaluator, trigger, observation time, resource version if available, and next expected scan. [4]

The source of truth matters. An IaC repository can express intended state, while the cloud control plane records actual state and management events. Neither view is complete by itself. IaC may omit settings applied by the platform or another controller. The control plane may show the final state but not the design rationale. A mature workflow correlates template version, deployment record, policy state, configuration history, and audit event.

Drift management also needs an action policy. Some differences should be denied before creation. Some should open a ticket. Some can be repaired automatically. Some require human review because the safest change depends on application context. The choice belongs in the control definition and operating procedure, not in the momentary judgment of whoever sees the dashboard first.

Drift question	Required evidence	Possible response
What changed?	Before/after configuration or resource property	Classify expected, emergency, unauthorized, or provider-driven change
Who or what changed it?	Caller identity, service role, pipeline, API, source IP, correlation ID	Investigate identity and change path
Was the change approved?	Pull request, deployment record, ticket, emergency approval	Accept, roll back, or escalate
Does the new state violate policy?	Current policy version, assignment scope, compliance evaluation	Deny future change, remediate, exempt, or update policy
What business impact exists?	Resource criticality, exposure, dependency, data classification, SLO	Prioritize by risk rather than check count
Has the state been verified after action?	Fresh assessment, functional test, telemetry, activity log	Close, reopen, or initiate recovery

15.7 Compliance Evidence Is Not Certification

A regulatory or industry framework is composed of outcomes, controls, procedures, documentation, organizational responsibilities, and judgments that extend beyond cloud resource properties. Azure Policy regulatory initiatives and comparable provider packages can map automated checks to selected control domains. They are valuable coverage tools, but the mapping does not mean that passing the automated checks certifies the organization or proves every control. Microsoft and Tencent Cloud both explicitly distinguish automated check coverage from a complete compliance conclusion. [21] [35]

The evidence chain should begin with the requirement and preserve each translation. A requirement is interpreted into a control objective. The objective is implemented through one or more technical and procedural controls. Assessment procedures examine, interview, or test the implementation. Evidence supports the result. A responsible authority decides whether gaps are remediated, mitigated, accepted, transferred, or reported. NIST SP 800-53A emphasizes assessment procedures rather than treating control text as self-proving. [2] [3] [5]

A resource marked compliant can still leave gaps. The policy may evaluate only one property. The control may also require a documented process, staff training, incident response, contractual language, key custody, or independent review. Conversely, a resource can be marked non-compliant while a mitigated exemption records an alternative control that meets the intent. Compliance dashboards should expose these distinctions instead of collapsing them into a single pass rate.

Compliance evidence must preserve the chain from requirement to resource state

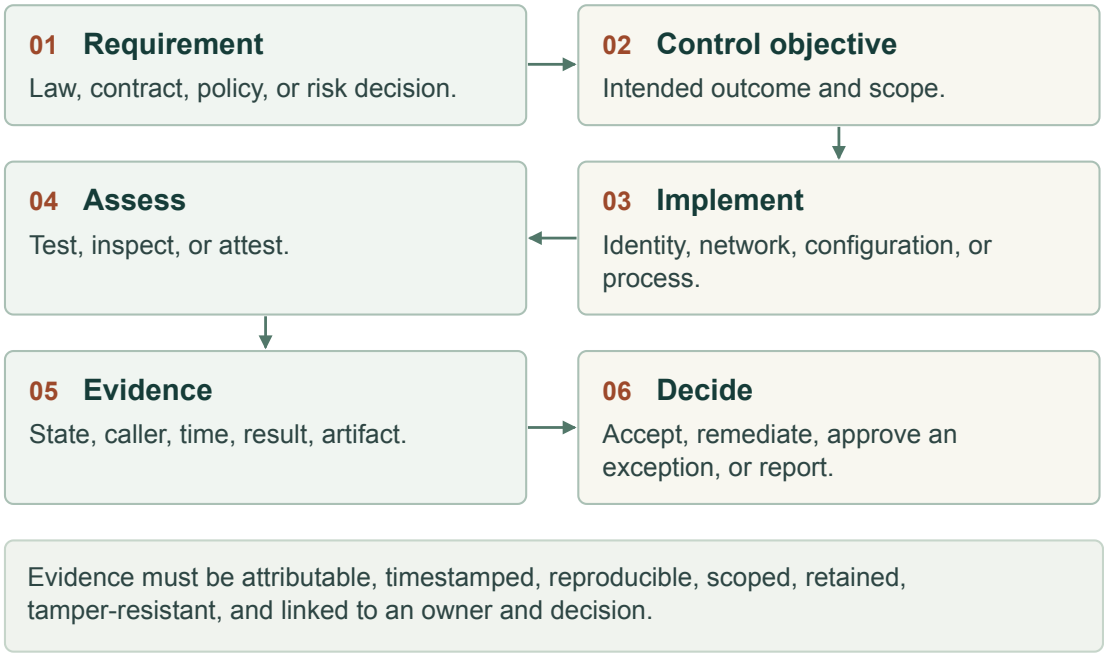


Figure 15-4 A defensible compliance claim preserves the chain from requirement to control, assessment, evidence, and decision.

Evidence field	Why it matters	Example
Requirement identifier	Prevents an automated rule from becoming detached from its business or regulatory reason	Internal SEC-12, contract clause, NIST control, data-residency requirement
Control objective	States the outcome independent of one provider implementation	Production storage must not permit anonymous public read
Implementation	Names the technical and procedural controls	Deny policy, private endpoint, owner review, public-access alert
Scope and owner	Defines which resources and team are accountable	Production subscriptions, data platform owner
Assessment procedure	Explains how design and operation are tested	Inspect assignment, attempt prohibited deployment, query existing state, review logs
Evidence artifact	Supports reproducibility and audit	Policy JSON, assignment ID, denied request, compliance result, activity log

Evidence field	Why it matters	Example
Exception / residual gap	Prevents silent suppression	Waiver until migration date with compensating firewall rule
Decision and review date	Connects evidence to accountable risk management	Accepted by risk owner; review in 30 days

Compliance disclaimer

Provider compliance dashboards and mappings are engineering evidence, not legal advice, certification, or a substitute for a complete control assessment. Determine applicable obligations with qualified legal, risk, and assurance professionals.

15.8 CSPM Adds Risk Context to Configuration Findings

Configuration compliance asks whether a resource matches a rule. Cloud Security Posture Management asks a broader question: which combination of misconfiguration, identity, vulnerability, data sensitivity, exposure, and dependency creates the most important risk? Modern CSPM systems add asset inventory, recommendations, risk prioritization, and attack-path analysis to configuration checks. Azure Defender for Cloud, Alibaba Cloud Security Center, Tencent Cloud Security Center, and Volcengine security products expose versions of this intent. [19] [20] [30] [35] [36] [42]

The distinction matters operationally. Ten open network rules are not necessarily ten equal risks. One may expose a management port on an identity-bearing production host; another may affect an isolated test instance that contains no sensitive data. A graph that connects exposure, privilege, reachable assets, vulnerabilities, and business criticality can prioritize the first issue without pretending the second issue is acceptable.

CSPM findings should still enter the same governance loop. The finding needs an owner, scope, due date, evidence, exception path, remediation plan, and verification. Scores and recommendations do not change resources by themselves. They also do not replace incident detection, workload protection, vulnerability management, data security, or application security. Posture management is the decision-support layer that connects these signals to cloud resource context.

AI services extend the posture surface. Public model endpoints, overprivileged deployment identities, unprotected knowledge stores, unreviewed tool connections, sensitive prompts, and cross-resource attack paths can create AI-specific exposure without changing the underlying governance pattern. The same control questions apply: which resource, which identity, which data path, which policy, which evidence, and which accountable owner?

Layer	Question	Representative evidence
Configuration compliance	Does the resource match a specific rule?	Policy state, config snapshot, rule version
Security posture	How significant is the finding in context?	Exposure, identity, vulnerability, criticality, data, attack path
Threat detection	Is suspicious or malicious activity occurring?	Alert, behavior, indicator, runtime event
Incident response	What coordinated actions contain, eradicate, and recover?	Incident record, timeline, containment and recovery evidence
Assurance	Are controls designed and operating effectively over time?	Independent assessment, samples, interviews, tests, exceptions, management decision

15.9 Audit Trails, Resource Locks, and Change Evidence

Azure Activity Log records subscription-level control-plane operations such as resource creation, configuration changes, access-policy changes, and deployment errors. Alibaba Cloud ActionTrail records console, API, SDK, and service-role events; Tencent Cloud CloudAudit and Volcengine audit and logging services provide corresponding management-event evidence. Organization-wide trails can deliver events to centralized log or object-storage destinations for longer retention and analysis. [17] [29] [32] [40]

Audit evidence should answer who, what, when, where, how, and with what result. Useful fields include caller identity, operation, resource ID, timestamp, status, source, correlation or request ID, policy decision, and changed properties. The evidence becomes stronger when it is exported to a protected destination with retention, access control, integrity safeguards, and monitoring for delivery failure.

A resource lock solves a narrower problem. Azure Delete and Read-only locks protect a subscription, resource group, or resource from accidental deletion or modification and can override ordinary user permissions for those operations. The lock is not an immutable security boundary: sufficiently authorized administrators can remove it, and its behavior is primarily on the control plane. Locks are valuable for critical shared resources when combined with IAM, policy, audit, and break-glass procedures. [18]

Audit, policy, and locks should be correlated. A denied deployment proves that a preventive control operated. The activity log explains the caller, target, time, and failure. The policy assignment and definition explain why the request was denied. A compliant redeploy-

ment proves that the corrected request satisfied the rule. Together they create an evidence packet; no single screenshot does.

Evidence source	Best answers	Retention / integrity concern
Policy state	Which rule evaluated which resource and with what result?	Evaluation latency, rule version, exemption state, provider coverage
Control-plane audit trail	Who called which management operation, when, and with what status?	Export, retention, privileged access, delivery failure
Configuration history	What property or relationship changed over time?	Sampling interval, unsupported resources, before/after completeness
IaC deployment record	Which approved artifact attempted to create or change the state?	Repository integrity, pipeline identity, parameter and secret handling
CSPM finding	Why is the state risky in context?	Scoring model changes, stale asset context, suppressed findings
Resource lock / deletion protection	Was an accidental control-plane operation blocked?	Who can remove the lock and whether the data plane remains mutable
Assessment or attestation	What manual or independent evidence supports the control?	Reviewer independence, sample coverage, expiry and revalidation

15.10 Exceptions, Exemptions, and Risk Acceptance

A governance system without an exception mechanism drives teams toward hidden bypasses. A governance system with unbounded exceptions converts every guardrail into optional advice. The objective is a controlled exception lifecycle: explicit scope, owner, rationale, category, compensating control, expiry, evidence, approval, and review.

Azure Policy distinguishes exclusions from exemptions. An excluded scope is not evaluated by the assignment. An exemption remains visible as an exempted state and can represent a mitigated condition or a waiver. A mitigated exemption states that the policy intent is met another way; a waiver temporarily accepts the non-compliant state. Exemptions can carry expiry metadata and remain recorded after expiry for traceability. [13] [14]

Equivalent concepts across providers may use whitelists, ignored findings, compliance exceptions, rule exclusions, or organizational policy exceptions. The semantic question matters more than the label. Does the resource disappear from evaluation? Does it remain visible? Can the exception expire automatically? Is a compensating control recorded? Who can create it? Is the decision included in organization-level evidence?

Exception authority should be separate from the team requesting the exception for high-risk controls. Emergency break-glass operations need short-lived authority, post-event review, and evidence. Recurring exceptions should trigger architectural review: a control that cannot be met by an entire workload class may be incorrectly designed, incorrectly scoped, or exposing a real platform gap.

Non-compliance must enter either a remediation path or an explicit risk-acceptance path

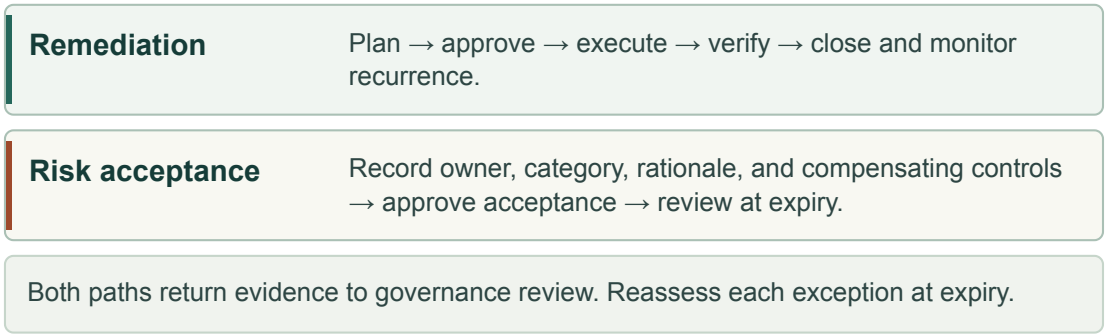


Figure 15-5 Every non-compliant state should enter either a verified remediation path or an explicit, expiring risk-acceptance path.

Exception field	Required content
Scope	Exact resource, resource hierarchy, policy assignment, control, and environment
Category	Mitigated, waiver, temporary operational exception, emergency, or equivalent provider category
Rationale	Why the normal control cannot currently be met and the business consequence of denial
Compensating control	Alternative control, monitoring, network restriction, approval, or manual verification
Risk owner	Named person or role authorized to accept the residual risk
Expiry and review	Expiration time, review cadence, and trigger for early revocation
Evidence	Finding, architecture, tests, approvals, linked tickets, and remediation plan
Exit plan	Concrete action, owner, milestone, and verification needed to remove the exception

15.11 Remediation Is a Governed Change Workflow

Remediation is not the same as making a dashboard green. A correct remediation preserves service intent, minimizes change risk, uses a bounded identity, records approval, verifies the final state, and can be reversed or compensated if the change harms the workload. The remediation path should therefore reuse the change-governance and observability disciplines established elsewhere in this book.

Azure Policy remediation tasks can use the managed identity attached to a policy assignment to apply modify operations or DeployIfNotExists templates to existing resources. The identity needs only the roles required by the remediation. New or updated resources can be handled during evaluation, while existing resources require an explicit remediation task. The important architectural point is not the specific effect name; it is that corrective automation has its own principal, authority, scope, plan, evidence, and lifecycle. [12]

Automatic remediation should begin with low-risk, reversible states: adding required metadata, enabling diagnostics, or deploying a standard related resource after testing. High-risk changes such as closing endpoints, changing encryption keys, replacing identity models, or modifying production network routes require workload-aware planning and often human approval. A posture platform can recommend the change; it should not silently acquire unlimited authority to execute it.

Verification completes the workflow. Re-run the compliance evaluation, inspect the resource state, check application health and SLOs, review the activity log, and confirm that the finding does not immediately recur. Track mean time to remediate, reopen rate, false-positive rate, exception age, control coverage, and realized risk reduction—not only the number of closed recommendations.

Remediation stage	Required control
Triage	Validate scope, freshness, severity, business criticality, and duplicate findings
Plan	Describe intended state, affected resources, dependencies, blast radius, rollback, and verification
Authorize	Use separation of duties and risk-based approval; bind an execution identity to the plan
Execute	Use IaC, policy remediation, or an approved workflow with idempotency and concurrency protection
Verify	Reassess compliance, inspect resource state, review telemetry, and confirm service behavior

Remediation stage	Required control
Close	Attach evidence, record residual risk, remove temporary authority, and update the exception register
Learn	Adjust policy, test cases, documentation, ownership, or architecture to reduce recurrence

15.12 Separation of Duties and the Governance Operating Model

Organizational governance fails when one identity can define the control, assign it globally, exempt its own resources, execute remediation, delete the audit trail, and approve the result. Separation of duties does not require a large bureaucracy, but it does require explicit boundaries among policy authorship, policy deployment, exception approval, remediation execution, and assurance review.

Policy definitions can be managed as code by a platform repository. A security reviewer approves control intent and test cases. A deployment pipeline with a dedicated identity assigns the policy to a limited scope. Workload owners remediate their resources or request time-bound exceptions. A separate assurance role reviews compliance and audit evidence. Emergency paths are available but produce mandatory post-event review.

The operating model should minimize permanent high privilege. Root-scope policy changes, organization control policies, centralized trails, remediation identities, and exemption permissions are privileged capabilities. Use just-in-time access where available, workload identities instead of shared credentials, protected branches for policy code, and alerts for changes to the governance system itself. The controls that govern the cloud are among the cloud's most sensitive resources.

Governance activity	Recommended primary actor	Independent check
Define control objective	Security architecture / risk owner	Legal, compliance, workload, and platform review
Implement policy code	Cloud platform engineering	Automated tests and peer review
Assign to broad scope	Governance deployment pipeline or authorized platform role	Change approval and safe-deployment evidence
Request exception	Workload owner	Security / risk approver
Execute remediation	Bound remediation identity or workload team	Plan digest, approval, telemetry, and post-change assessment

Governance activity	Recommended primary actor	Independent check
Review evidence	Compliance / assurance / internal audit	Source-system verification and sampling
Change audit retention or destination	Security operations / platform owner	Independent alert and approval

Govern the governance plane

Policy definitions, assignments, exemptions, remediation identities, audit destinations, and security-posture settings are high-value cloud resources. Inventory, version, monitor, and protect them accordingly.

15.13 Azure as the Reference Governance Control Plane

Azure makes the governance object model explicit. Management groups organize subscriptions and provide inherited scopes. Policy definitions express rules; initiatives group related definitions toward a control objective; assignments bind rules to management groups, subscriptions, resource groups, or resources; exclusions and exemptions narrow evaluation; compliance data records results; remediation tasks correct selected states; Activity Log records control-plane operations; resource locks protect critical scopes; and Defender for Cloud adds contextual posture analysis and recommendations. [6] [11] [12] [15] [17] [18] [19] [22]

These objects should be treated as a resource graph. An initiative is not effective until assigned. An assignment has no meaning without scope and parameters. A remediation policy needs an identity and roles. An exemption should reference the assignment and carry ownership and expiry. Activity events explain changes but do not replace compliance state. Defender recommendations can prioritize risk but do not grant authority to change the resource. The graph is the governance system.

The Azure model also exposes lifecycle. Built-in policy definitions and initiatives can evolve. Custom definitions need versioning and retirement. Assignments are deployed, changed, excluded, exempted, and deleted. Compliance evaluations have triggers and latency. Remediation tasks are created and complete. Locks are added and removed. Audit data is retained or exported. A governance architecture should document each lifecycle rather than treating “enable policy” as a one-time action.

Azure object or service	Governance role	Lifecycle question
Management group	Inheritance and delegated organizational scope	Who can change hierarchy and what policies move with subscriptions?
Policy definition	Reusable rule and effect	How is it tested, versioned, deprecated, and retired?
Initiative	Control-objective bundle	Which definitions and parameters satisfy the objective?
Policy assignment	Scope, parameters, enforcement mode, and optional remediation identity	How is rollout limited, monitored, changed, and removed?
Exclusion / exemption	Tracked exception to evaluation or assignment effect	Who approved it, why, and when does it expire?
Policy state / compliance data	Observed result for resource and assignment	How fresh is the result and what coverage is missing?
Remediation task	Corrective action for modify or DeployIfNotExists policies	Which identity and roles execute it, and how is success verified?
Activity Log	Control-plane event evidence	Where is it exported, retained, protected, and correlated?
Resource lock	Coarse protection against deletion or modification	Who can remove it and what operations remain possible?
Defender for Cloud	Posture, recommendation, graph, and attack-path context	Which plan, scope, finding owner, due date, and exception process apply?

15.14 Multicloud Governance Translation

The four cloud platforms in this book share a governance intent: organize accounts and resources, bound authority, record management events, evaluate resource configuration, aggregate evidence, and prioritize security risk. They do not expose identical enforcement semantics. Azure Policy can participate directly in request-time deny, modification, or related-resource deployment. Alibaba Cloud Config, Tencent Cloud Config, and Volcengine Config Audit emphasize continuous inventory, rule evaluation, compliance packages or baselines, configuration history, and evidence; provider-specific organization policies and permission boundaries may constrain actions separately. [23] [31] [37]

This distinction prevents false equivalence. A service control policy can limit what member-account identities are allowed to do without evaluating whether every existing resource matches a desired state. A configuration rule can mark resources non-compliant without

blocking creation. A posture check can recommend remediation without possessing execution authority. A tag policy can standardize tag operations while cost allocation, authorization, and resource compliance remain separate contracts.

Multi-account aggregation is another common intent. Alibaba Cloud integrates Cloud Config, ActionTrail, and Security Center with Resource Directory and delegated administration. Tencent Cloud provides organization-level service control and tag policies, multi-account CloudAudit, Config, and Cloud Security Center capabilities. Volcengine provides Enterprise Organization control policies, account groups and cross-account compliance baselines in Config Audit, evidence delivery to TLS, and multicloud security posture capabilities. The exact supported services, rule templates, remediation methods, and regional availability are volatile and should be verified before implementation. [27] [32] [39] [40] [41] [42]

Governance controls and evidence across four platforms

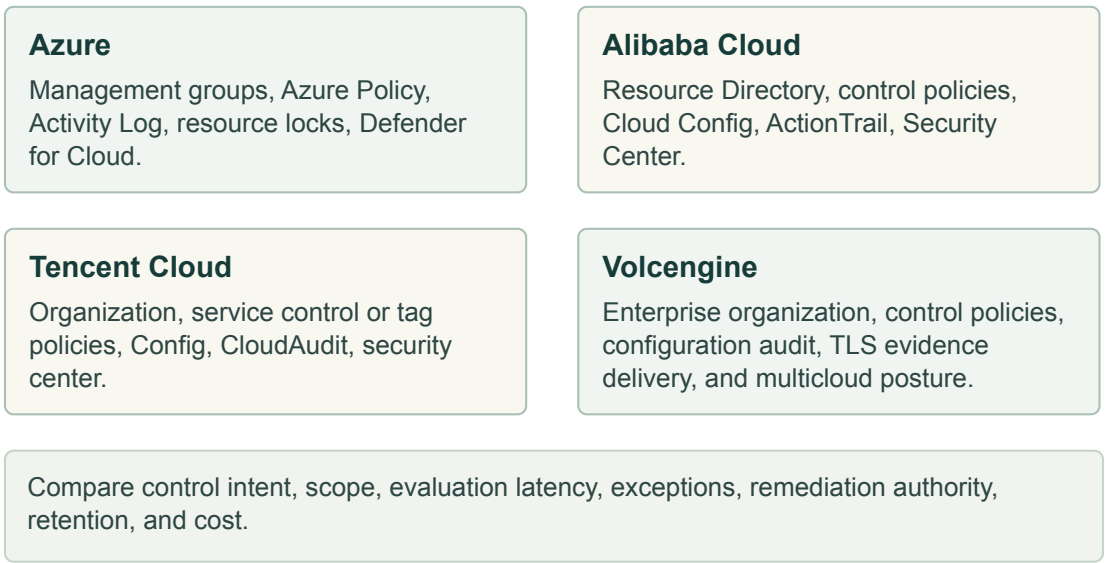


Figure 15-6 The providers expose a shared governance intent through different organizational, policy, configuration, audit, and posture contracts.

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Organize accounts and inheritance	Management groups and subscriptions	Resource Directory, folders, member accounts	Tencent Cloud Organization, departments, members	Enterprise Organization, departments, member accounts

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Bound maximum organizational authority	RBAC plus policy at inherited scopes	Control policies plus RAM	Service control policies plus CAM	Control policies plus IAM
Evaluate resource configuration	Azure Policy compliance	Cloud Config rules and compliance packages	Config continuous recording and evaluation	Config Audit rules, baselines, account groups
Prevent disallowed state	Deny and related policy effects at request time	Use control policies, service controls, and supported prevention/remediation patterns; verify per service	Use organization permission boundaries and service controls; Config primarily assesses state	Use organization control policies and supported service controls; Config Audit primarily assesses state
Remediate findings	Modify / DeployIfNotExists plus remediation tasks and managed identity	Cloud Config template or custom remediation where supported	CSC recommendations and provider/workflow remediation; verify Config capabilities per rule	Provider-specific guidance and workflows; do not assume Azure-equivalent automatic remediation
Record management events	Activity Log and exported diagnostic destinations	ActionTrail to SLS or OSS	CloudAudit and centralized shipping	Config history and delivery to TLS; verify operation-audit services per resource
Assess posture and context	Defender for Cloud CSPM and attack paths	Security Center CSPM	Cloud Security Center CSPM / CNAPP	Multicloud Security Platform and related security products
Govern tags	Azure Policy and tag inheritance/requirements	Tag Policy and Cloud Config required-tags checks	TCO tag policies and CAM tag authorization	Projects/tags plus configuration rules; verify current tag-policy support

Record non-equivalence

When a Chinese-cloud service evaluates but does not block resource creation, or when remediation requires a different execution path, write that difference into the control design. Do not hide it behind a product mapping.

15.15 Lab 4: Enforce a Resource Policy and Produce Compliance Evidence

This independent lab creates a custom Azure Policy that denies storage-account creation inside a dedicated resource group unless the resource contains two ownership tags: owner and costCenter. You will capture a denied deployment, create a compliant resource, trigger

an evaluation scan, query compliance state, collect Activity Log evidence, and remove every lab resource.

The lab demonstrates a narrow control-plane capability. It does not claim that tags are a complete security control. The purpose is to prove the full governance path: definition, assignment, request-time enforcement, compliance evaluation, evidence, corrected deployment, and cleanup.

Lab field	Value
Capability goal	Enforce one organization rule and demonstrate both denied noncompliance and verified compliance
Azure resources	Resource group, custom policy definition, policy assignment, one compliant storage account
Required authority	Permission to create policy definitions and assignments at the chosen scope, plus permission to create and delete resources in the lab resource group
Cost boundary	Policy objects require no dedicated compute; the storage account should incur negligible cost if deleted immediately. Verify current pricing and inherited controls.
Evidence	Definition, assignment, denied CLI output, compliant resource ID, policy state, Activity Log export, cleanup result
Cleanup contract	Delete the assignment, custom definition, storage account, and resource group; verify the group no longer exists

Lab safety

Use a subscription where you are authorized to create custom policies. Do not assign the lab policy at management-group or whole-subscription scope. Keep the assignment limited to the dedicated lab resource group.

Step 1: Define names and create an isolated scope

```
SUBSCRIPTION_ID=$(az account show --query id -o tsv)
LOCATION=eastus
RG=book-lab-15-governance
POLICY_NAME=book-require-owner-costcenter
ASSIGNMENT_NAME=book-require-owner-costcenter-rg
SUFFIX=$(openssl rand -hex 4)
BAD_STORAGE=govbad${SUFFIX}
GOOD_STORAGE=govok${SUFFIX}
SCOPE=/subscriptions/${SUBSCRIPTION_ID}/resourceGroups/${RG}
mkdir -p evidence
```

```
az group create   --name "$RG"   --location "$LOCATION"   --tags owner=book-lab
costCenter=cloud-ai-paas purpose=governance-lab   > evidence/resource-group.json
```

Storage account names must be globally unique, lowercase, and within the provider limits. Change LOCATION if the selected region is unavailable or restricted by inherited policy. Save the active subscription and caller identity in the evidence packet before proceeding.

```
az account show > evidence/account.json
az ad signed-in-user show > evidence/caller.json 2>/dev/null || true
```

Step 2: Create the custom deny rule

Create a file named policy-rule.json. The rule applies only to storage accounts and denies creation or update when either required tag is absent.

```
{
  "if": {
    "allOf": [
      {
        "field": "type",
        "equals": "Microsoft.Storage/storageAccounts"
      },
      {
        "anyOf": [
          {
            "field": "tags['owner']",
            "exists": "false"
          },
          {
            "field": "tags['costCenter']",
            "exists": "false"
          }
        ]
      }
    ]
  },
  "then": {
    "effect": "deny"
  }
}
```

```
az policy definition create   --name "$POLICY_NAME"   --display-name "Book lab:
require owner and costCenter on storage accounts"   --description "Denies
storage accounts missing owner or costCenter tags."   --mode Indexed   --rules
```

```
@policy-rule.json   --subscription "$SUBSCRIPTION_ID"   > evidence/policy-
definition.json

POLICY_ID=$(az policy definition show   --name "$POLICY_NAME"   --subscription
"$SUBSCRIPTION_ID"   --query id -o tsv)
```

A production rule should include metadata, ownership, versioning, unit tests, and a safe-deployment record. The lab keeps the rule narrow so that the result is easy to verify.

Step 3: Assign the rule only to the lab resource group

```
az policy assignment create   --name "$ASSIGNMENT_NAME"   --display-name "Book
lab: require storage ownership tags"   --scope "$SCOPE"   --policy "$POLICY_ID"
--description "Lab-only deny policy assigned to one resource group."   >
evidence/policy-assignment.json

ASSIGNMENT_ID=$(az policy assignment show   --name "$ASSIGNMENT_NAME"   --scope
"$SCOPE"   --query id -o tsv)
```

Confirm the assignment scope before attempting any deployment. If SCOPE points to the subscription instead of the resource group, delete the assignment and correct it before continuing.

Step 4: Prove request-time enforcement

Attempt to create a storage account without the required tags. The command should fail with a policy-related denial. Capture both standard output and error output.

```
set +e
az storage account create   --name "$BAD_STORAGE"   --resource-group "$RG"   --
location "$LOCATION"   --sku Standard_LRS   --kind StorageV2   >
evidence/denied-deployment.stdout.json   2> evidence/denied-
deployment.stderr.txt
DENIED_EXIT=$?
set -e
printf "%s
" "$DENIED_EXIT" > evidence/denied-exit-code.txt

if [ "$DENIED_EXIT" -eq 0 ]; then
  echo "Expected policy denial, but deployment succeeded." >&2
  exit 1
fi
```

The error should identify a policy definition or assignment and normally includes a RequestDisallowedByPolicy-style result. If a different inherited policy caused the denial, inspect the assignment details and record the interference rather than claiming that the lab rule was verified. [43]

Step 5: Create a compliant resource

```
az storage account create   --name "$GOOD_STORAGE"   --resource-group "$RG"   --
location "$LOCATION"   --sku Standard_LRS   --kind StorageV2   --tags
owner=book-lab costCenter=cloud-ai-paas   > evidence/compliant-storage.json

az storage account show   --name "$GOOD_STORAGE"   --resource-group "$RG"   --
query
'{id:id,name:name,location:location,tags:tags,provisioningState:provisioningStat
e}'   > evidence/compliant-storage-summary.json
```

Step 6: Trigger evaluation and collect policy state

```
az policy state trigger-scan --resource-group "$RG"

az policy state list   --resource-group "$RG"   --query "[].
{resourceId:resourceId,complianceState:complianceState,policyAssignmentId:policy
AssignmentId,policyDefinitionName:policyDefinitionName,timestamp:timestamp}"   >
evidence/policy-state.json

az policy state summarize   --resource-group "$RG"   > evidence/policy-
summary.json
```

Policy state can take time to appear after a change. Preserve the scan time and do not treat an empty immediate response as proof of compliance. Re-run the query after the scan completes. The compliant storage account should not appear as non-compliant for this assignment.

Step 7: Collect control-plane audit evidence

```
az monitor activity-log list   --resource-group "$RG"   --offset 2h   >
evidence/activity-log.json

az policy assignment show   --name "$ASSIGNMENT_NAME"   --scope "$SCOPE"   >
evidence/policy-assignment-final.json

az policy definition show   --name "$POLICY_NAME"   --subscription
"$SUBSCRIPTION_ID"   > evidence/policy-definition-final.json
```

The evidence packet should allow a reviewer to reconstruct the control: the definition, assignment scope, denied request, corrected resource, current policy state, caller, timestamps, and Activity Log events. A screenshot may supplement the packet but should not replace machine-readable artifacts.

Acceptance test	Pass condition
Scope safety	Assignment ID is under the dedicated lab resource-group scope
Preventive control	Untagged storage-account create or update is denied by the lab assignment
Corrected deployment	Tagged storage account reaches Succeeded state
Compliance evidence	Policy state or summary can be queried after evaluation and does not report the compliant account as non-compliant
Auditability	Evidence includes caller, timestamps, resource IDs, assignment and definition IDs, and Activity Log data
Reproducibility	Another authorized reviewer can rerun the commands using the saved artifacts and variables

Step 8: Remove the governance objects and verify cleanup

```
az policy assignment delete   --name "$ASSIGNMENT_NAME"   --scope "$SCOPE"

az policy definition delete   --name "$POLICY_NAME"   --subscription
"$SUBSCRIPTION_ID"

az group delete   --name "$RG"   --yes

az group exists --name "$RG"   > evidence/resource-group-exists-after-
cleanup.txt
```

The final file should contain false. Also verify that the policy definition and assignment no longer exist. Keep only sanitized evidence; do not retain tokens, credentials, or organization-specific identifiers in a public repository.

Provider translation	Comparable implementation path	Important non-equivalence
Alibaba Cloud	Cloud Config required-tags rule or compliance package; Resource Directory for multi-account scope; ActionTrail for change evidence; template/custom remediation where supported	A configuration rule may assess existing resources rather than deny the original create request; prevention and remediation depend on service and control-policy capabilities

Provider translation	Comparable implementation path	Important non-equivalence
Tencent Cloud	Config rule and resource history; TCO tag policy or service control policy; CloudAudit; CSC CSPM for security findings	Tag standardization, action boundaries, configuration evaluation, and posture recommendations are separate control contracts
Volcengine	Config Audit rule/baseline in an account group; Enterprise Organization control policy; deliver configuration and compliance evidence to TLS	Current rule, prevention, remediation, and tag-policy support must be verified; do not assume request-time deny parity

Lab acceptance criterion

The lab passes only when the non-compliant request is denied by the intended assignment, the corrected resource is created, the control-plane evidence is reproducible, and cleanup removes both resources and governance objects.

CloudDesk Reuse: Governance and Posture Capabilities

CloudDesk FinOps Agent can reuse this chapter as three distinct capabilities. The read-only capabilities inspect and explain governance state. The remediation capability must remain separate, require a bounded execution identity, and follow the plan-approve-execute-verify contract established in the Agent Tools chapter. The Agent must never convert a compliance mapping into a legal conclusion.

```
capability: governance.policy.check
risk_level: R0
mode: read_only
inputs:
  provider: string
  scope_id: string
  control_intent: string
outputs:
  assignments: array
  evaluated_resources: array
  non_compliant_resources: array
  exemptions: array
  evidence_freshness: object
  coverage_gaps: array
constraints:
  - preserve provider-native IDs and policy semantics
  - distinguish deny, evaluate, recommend, and remediate
  - report unknown or unsupported coverage explicitly
```

```
capability: governance.policy.remediate
risk_level: R3
mode: execute_after_approval
inputs:
  finding_id: string
  resource_ids: array
  approved_plan_digest: string
  rollback_plan: object
outputs:
  execution_id: string
  changed_resources: array
  verification_results: array
  residual_findings: array
  evidence_packet_uri: string
constraints:
  - least-privilege execution identity
  - no action without immutable approved plan
  - verify policy state, workload health, and audit trail after change
  - remove temporary authority when complete
```

```
capability: security.posture.explain
risk_level: R0
mode: read_only
inputs:
  finding_id: string
  business_context: object
outputs:
  affected_resources: array
  control_objectives: array
  risk_factors: array
  attack_path_context: array
  recommended_actions: array
  exception_status: object
  evidence_sources: array
  uncertainty: array
constraints:
  - explain prioritization without claiming certification
  - cite current source evidence and timestamps
  - separate provider recommendation from organizational risk decision
```

CloudDesk use	Value	Mandatory guardrail
FinOps governance check	Detect missing ownership tags, budget scope, public exposure, or unsupported resources before optimization	Read-only; do not infer owner or cost center when metadata is missing

CloudDesk use	Value	Mandatory guardrail
Policy-impact explanation	Explain why a deployment or change was denied and identify the governing assignment	Preserve assignment scope and inherited-policy evidence
Remediation proposal	Create a change plan that connects finding, resource, expected state, cost, SLO, and rollback	Human approval for disruptive or security-sensitive changes
Exception review	Find expired or weakly justified waivers and missing compensating controls	Do not revoke high-impact exceptions without accountable review
Posture summary	Prioritize findings by business context, exposure, identity, data, and attack path	Scores are advisory; risk acceptance belongs to authorized humans

Chapter Summary

- Security posture is the time-bound relationship among organizational intent, actual cloud state, risk context, evidence quality, exceptions, and control effectiveness.
- Governance establishes decision rights, ownership, risk appetite, policy intent, review cadence, and acceptable exception paths.
- Organizational hierarchy determines how controls inherit, how evidence aggregates, and where delegated administration operates.
- RBAC, resource policy, organization permission boundaries, resource locks, CSPM, audit trails, and compliance mappings solve different problems.
- Policy as code requires testing, versioning, safe rollout, evidence, rollback, and retirement—not only JSON syntax.
- Preventive, detective, corrective, compensating, and recovery controls should be designed as one system.
- Continuous compliance requires repeated evaluation, drift awareness, evidence freshness, and explicit action rules.
- A provider compliance dashboard is valuable engineering evidence but not a legal certification or complete control assessment.
- CSPM adds exposure, vulnerability, identity, data, dependency, and attack-path context to configuration findings.
- Every exception needs scope, owner, rationale, compensating control, expiry, evidence, and an exit plan.
- Remediation is a governed change workflow with bounded authority, verification, and rollback.

- Multicloud governance must translate intent, enforcement semantics, evaluation latency, evidence, exception behavior, and remediation—not product names alone.

References and Official Documentation

Product names, documentation state, and cited provider descriptions were checked August 25, 2026. Built-in policies, compliance packages, security-product plans, regional availability, certifications, pricing, and remediation capabilities change. Verify current official documentation and applicable legal requirements before implementation.

[1] National Institute of Standards and Technology. The NIST Cybersecurity Framework (CSF) 2.0, NIST CSWP 29. https://csrc.nist.gov/pubs/cswp/29/the-nist-cybersecurity-framework-csf-20/final

[2] National Institute of Standards and Technology. Security and Privacy Controls for Information Systems and Organizations, SP 800-53 Rev. 5. https://csrc.nist.gov/pubs/sp/800/53/r5/upd1/final

[3] National Institute of Standards and Technology. Assessing Security and Privacy Controls in Information Systems and Organizations, SP 800-53A Rev. 5. https://csrc.nist.gov/pubs/sp/800/53/a/r5/final

[4] National Institute of Standards and Technology. Information Security Continuous Monitoring (ISCM), SP 800-137. https://csrc.nist.gov/pubs/sp/800/137/final

[5] National Institute of Standards and Technology. Risk Management Framework. https://csrc.nist.gov/projects/risk-management/about-rmf

[6] Microsoft. Overview of Azure Policy. https://learn.microsoft.com/en-us/azure/governance/policy/overview

[7] Microsoft. Azure Policy definition structure basics and policy rule. https://learn.microsoft.com/en-us/azure/governance/policy/concepts/definition-structure-basics and https://learn.microsoft.com/en-us/azure/governance/policy/concepts/definition-structure-policy-rule

[8] Microsoft. Azure Policy effect basics. https://learn.microsoft.com/en-us/azure/governance/policy/concepts/effect-basics

[9] Microsoft. Design Azure Policy as Code workflows. https://learn.microsoft.com/en-us/azure/governance/policy/concepts/policy-as-code

[10] Microsoft. Safe deployment of Azure Policy assignments. https://learn.microsoft.com/en-us/azure/governance/policy/how-to/policy-safe-deployment-practices

[11] Microsoft. Get compliance data of Azure resources. https://learn.microsoft.com/en-us/azure/governance/policy/how-to/get-compliance-data

[12] Microsoft. Remediate non-compliant resources with Azure Policy. https://learn.microsoft.com/en-us/azure/governance/policy/how-to/remediate-resources

[13] Microsoft. Azure Policy exemption structure. https://learn.microsoft.com/en-us/azure/governance/policy/concepts/exemption-structure

[14] Microsoft. Understand scope in Azure Policy. https://learn.microsoft.com/en-us/azure/governance/policy/concepts/scope

[15] Microsoft. Organize resources with Azure management groups. https://learn.microsoft.com/en-us/azure/governance/management-groups/overview

[16] Microsoft. Cloud Adoption Framework management-group design for Azure landing zones. https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/landing-zone/design-area/resource-org-management-groups

[17] Microsoft. Activity Log in Azure Monitor. https://learn.microsoft.com/en-us/azure/azure-monitor/platform/activity-log

[18] Microsoft. Lock Azure resources to protect infrastructure. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/lock-resources

[19] Microsoft. Cloud Security Posture Management in Defender for Cloud. https://learn.microsoft.com/en-us/azure/defender-for-cloud/concept-cloud-security-posture-management

[20] Microsoft. Security Explorer and attack paths in Defender for Cloud. https://learn.microsoft.com/en-us/azure/defender-for-cloud/concept-attack-path

[21] Microsoft. Regulatory Compliance in Azure Policy initiative definitions. https://learn.microsoft.com/en-us/azure/governance/policy/concepts/regulatory-compliance

[22] Microsoft. Azure Policy initiative definition structure. https://learn.microsoft.com/en-us/azure/governance/policy/concepts/initiative-definition-structure

[23] Alibaba Cloud. What is Cloud Config? https://www.alibabacloud.com/help/en/cloud-config/latest/what-is-cloud-config

[24] Alibaba Cloud. Compliance packages overview. https://www.alibabacloud.com/help/en/cloud-config/latest/compliance-packages-overview

[25] Alibaba Cloud. Cloud Config remediation overview. https://www.alibabacloud.com/help/en/cloud-config/latest/remediation-settings-overview

[26] Alibaba Cloud. What is Resource Directory? https://www.alibabacloud.com/help/en/resource-management/resource-directory/product-overview/resource-directory-overview

[27] Alibaba Cloud. Resource Directory trusted services. https://www.alibabacloud.com/help/en/resource-management/resource-directory/user-guide/overview-1

[28] Alibaba Cloud. Tag policies overview. https://www.alibabacloud.com/help/en/resource-management/tag/user-guide/overview

[29] Alibaba Cloud. What is ActionTrail? https://www.alibabacloud.com/help/en/actiontrail/product-overview/what-is-actiontrail

[30] Alibaba Cloud. Cloud Security Posture Management in Security Center. https://www.alibabacloud.com/help/en/security-center/user-guide/cspm

[31] Tencent Cloud. Config overview. https://www.tencentcloud.com/document/product/1164/51489

[32] Tencent Cloud. Multi-account log collection and auditing with CloudAudit. https://www.tencentcloud.com/document/product/1220/57831

[33] Tencent Cloud. Service control policies in Tencent Cloud Organization. https://www.tencentcloud.com/document/product/1031/51869

[34] Tencent Cloud. Creating a tag policy. https://www.tencentcloud.com/document/product/1031/55962

[35] Tencent Cloud. Cloud Resource Configuration Check in Cloud Security Center. https://www.tencentcloud.com/document/product/1008/80336

[36] Tencent Cloud. Cloud Security Center product overview. https://www.tencentcloud.com/document/product/1008/79817

[37] Volcengine. Config Audit product introduction. https://www.volcengine.com/docs/87053/1999420

[38] Volcengine. Config Audit rule policy. https://www.volcengine.com/docs/87053/2407063

[39] Volcengine. Config Audit account groups and cross-account compliance baselines. https://www.volcengine.com/docs/87053/2024041

[40] Volcengine. Deliver Config Audit evidence to TLS. https://www.volcengine.com/docs/87053/2024032

[41] Volcengine. Enterprise Organization control-policy overview. https://www.volcengine.com/docs/6723/147486

[42] Volcengine. Multicloud Security Platform compliance results. https://www.volcengine.com/docs/69190/1282941

[43] Microsoft. Troubleshoot RequestDisallowedByPolicy. https://learn.microsoft.com/en-us/azure/azure-resource-manager/troubleshooting/error-policy-requestdisallowedbypolicy

[44] Microsoft. Azure CLI policy state command reference. https://learn.microsoft.com/en-us/cli/azure/policy/state?view=azure-cli-latest

AI PaaS Control Planes

A successful model call reveals little about the resources that make it dependable. Start with the data and model foundations, then inspect deployment versions, retrieval policy, agent authority, evaluation evidence, and retirement.

16 From Data+AI Platforms and ML PaaS to AI PaaS

17 From Model APIs to AI PaaS Resources

18 RAG Is Not a Vector Database

19 Agent Platforms as Governed Cloud Resources

Part V / Chapter 16

From Data+AI Platforms and ML PaaS to AI PaaS

How the Cloud Control Plane Expanded from Model Lifecycles to Models, Knowledge, Tools, Evaluations, and Agents

AI PaaS did not begin when cloud providers added a model catalog. It emerged when the managed data and machine-learning lifecycle was extended to govern model services, knowledge resources, evaluations, tools, and agents as production cloud objects.

In This Chapter

- Why data services remain the foundation of both classical machine learning and generative AI systems
- How traditional ML PaaS turned data assets, experiments, training jobs, models, endpoints, and monitoring into managed resources
- Why a Data+AI platform is more than colocated notebooks, data engines, and GPU clusters
- What generative AI changes in the control plane and what it leaves unchanged
- How AI PaaS adds model services, knowledge, prompts, tools, evaluations, safety policies, and agents to the managed resource graph
- Why MLOps, LLMOps, and AgentOps overlap rather than replace one another
- How the Azure, Alibaba Cloud, Tencent Cloud, and Volcengine platform paths express similar engineering intents with different product boundaries
- How to use an architecture worksheet to separate data, model, knowledge, tool, and agent resources before choosing a provider
- Which anti-patterns turn an AI platform into an expensive collection of disconnected services
- How CloudDesk FinOps Agent can reuse platform inventory, lineage, and lifecycle-gap assessment capabilities without being implemented in this chapter

The phrase AI platform is often used as though it names one stable product category. In practice, it can refer to a notebook service, a distributed training scheduler, a model registry, a hosted inference API, a data lakehouse, a prompt playground, a retrieval system, or an agent builder. Marketing pages may place these capabilities inside one diagram, but engineering teams still have to determine which objects are durable, who owns them, what data they depend on, how they are released, what evidence they produce, and how they are retired.

The historical progression matters. Cloud providers first productized the machine-learning lifecycle: managed development environments, elastic training jobs, experiment tracking, data and environment assets, model registries, pipelines, endpoints, and monitoring. This is ML PaaS. It did not merely provide GPUs. It moved recurring ML engineering obligations into a control plane so that teams could reproduce, authorize, schedule, observe, and audit work that had previously been held together by scripts and individual machines.

Generative AI changed the center of gravity but not the need for that foundation. Many teams now consume pretrained models instead of training every model from scratch. Their production systems depend on model catalogs, managed inference, prompt and application configuration, retrieval-augmented generation, evaluation suites, safety policies, tools, memory, and agent runtimes. These are additional resource families. They extend the platform from “manage the model lifecycle” to “manage the lifecycle of model-backed applications and agents.”

This chapter explains that control-plane expansion. It begins with the data foundation because a model, a retrieval system, and an agent cannot be governed independently of the data they transform or cite. It then formalizes the resource model of ML PaaS, identifies the new objects introduced by AI PaaS, and maps four provider paths. Azure supplies the primary teaching reference through Azure Databricks, Azure Machine Learning, and Microsoft Foundry. Alibaba Cloud, Tencent Cloud, and Volcengine are compared by engineering intent rather than forced product symmetry.

The chapter thesis

ML PaaS productizes the lifecycle of data, training, models, and deployments. AI PaaS preserves that foundation and adds governed model services, knowledge, prompt and application configuration, tools, evaluations, safety controls, and agent resources.

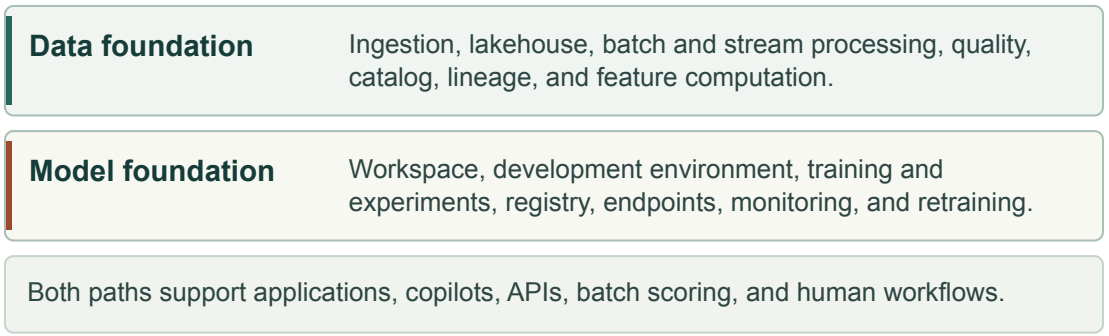


Figure 16-1 A production AI system stands on both a managed data foundation and a managed model foundation.

16.1 AI Platform Evolution Begins with Data, Not Models

A platform that offers model training or inference without a stable data system is only a partial AI platform. Training data must be discovered, classified, cleaned, versioned, transformed, and connected to model runs. Retrieval data must be parsed, chunked, enriched, authorized, refreshed, and traced to source. Agent tools depend on operational data with business semantics, ownership, and access boundaries. Even a hosted model API receives context, emits traces, and participates in application workflows whose correctness depends on data contracts.

The data side of the platform therefore answers questions that the model side cannot answer. Where did a feature originate? Which transformation produced a table? Which document version supplied a retrieved passage? Which users are eligible to see it? Which pipeline refreshed it? What retention rule applies? Which downstream models, indexes, reports, or agents will be affected by a schema change? Data lineage is not decorative metadata; it is the dependency graph needed for debugging, impact analysis, compliance evidence, and controlled change.

Feature engineering illustrates the point in classical ML. Raw fields are cleaned, encoded, combined, aggregated over time windows, and selected. A managed feature platform adds definitions, versions, computation logic, online and offline serving, freshness expectations, and reuse. The engineering objective is not to store more columns. It is to prevent training-serving skew and to make a feature a governed asset rather than an undocumented expression copied across notebooks.

Generative AI changes the transformations but not the obligation. A document corpus may be converted into chunks, embeddings, metadata, and eligibility filters. A conversation may generate feedback, evaluation cases, safety events, and cost records. A tool call may read or mutate an operational system. These flows belong in the data estate even when the user experiences them as a chat response. If the data lineage ends at the model API, the most consequential part of the AI system is invisible.

For this reason, the course material underlying this book described traditional AI PaaS as having two legs: a data-development and governance platform, and a model platform for training, evaluation, deployment, and monitoring. Modern AI PaaS adds model services, retrieval, tools, and agents, but the two-legged structure remains. A provider may package the products differently, yet the engineering responsibilities still exist.

Data obligation	Classical ML example	Generative AI / agent example	Control-plane evidence
Acquisition and admission	Approved training dataset	Approved document source or tool data source	Source registration, owner, region, classification
Transformation	Cleaning, feature computation, labeling	Parsing, chunking, embedding, redaction, enrichment	Pipeline version, code or component digest, run record
Quality	Missing values, drift, label quality	Extraction failures, stale chunks, citation gaps, poisoned content	Quality rules, failed records, evaluation results
Lineage	Dataset to run to model	Source to chunk to retrieval to answer	Dependency graph, trace and provenance identifiers
Authorization	Workspace and data access	Document eligibility, tenant filter, tool scope	Role assignment, policy, query filter, access decision
Retention and retirement	Remove obsolete datasets and features	Delete stale indexes, embeddings, conversations, memories	Retention policy, deletion event, purge evidence

Data is an active dependency

The model layer cannot repair missing ownership, stale lineage, inconsistent business definitions, or an authorization rule that was never attached to the data.

16.2 What ML PaaS Productized

Before ML PaaS, a typical model project combined local notebooks, manually provisioned GPU machines, ad hoc storage paths, copied environments, training scripts, spreadsheets of metrics, and a custom web service for deployment. The model artifact was often the only object treated as durable. Everything that produced it was implicit, and production ownership began too late.

ML PaaS changed the unit of management. A workspace or project became the collaboration and authorization scope. Data and environments became versioned assets. Training became a job with declared inputs, compute, code, parameters, outputs, logs, and status. Experiments grouped comparable runs. Pipelines represented dependencies between repeatable components. Registries stored versioned models and provenance. Endpoints separated a stable invocation contract from one or more deployments. Monitoring connected production behavior back to the model lifecycle.

Azure Machine Learning describes this purpose directly: it manages the ML project lifecycle, supports training and deployment, and provides MLOps for monitoring, retraining, and redeployment in a secure and auditable production environment. Its workspace organizes shared assets and collaboration; pipelines automate work; managed endpoints abstract online and batch inference; and job artifacts and lineage support auditability. [4]

The important change was not the existence of a notebook, a scheduler, or a registry. Each of those could be installed separately. The change was the durable relation among them. The platform could answer which data, code, environment, compute, parameters, and run produced a model; which deployment served it; which identity changed the endpoint; and which evidence supported promotion or rollback. This is the control-plane value of ML PaaS.

Traditional ML PaaS productizes the model lifecycle as a resource graph

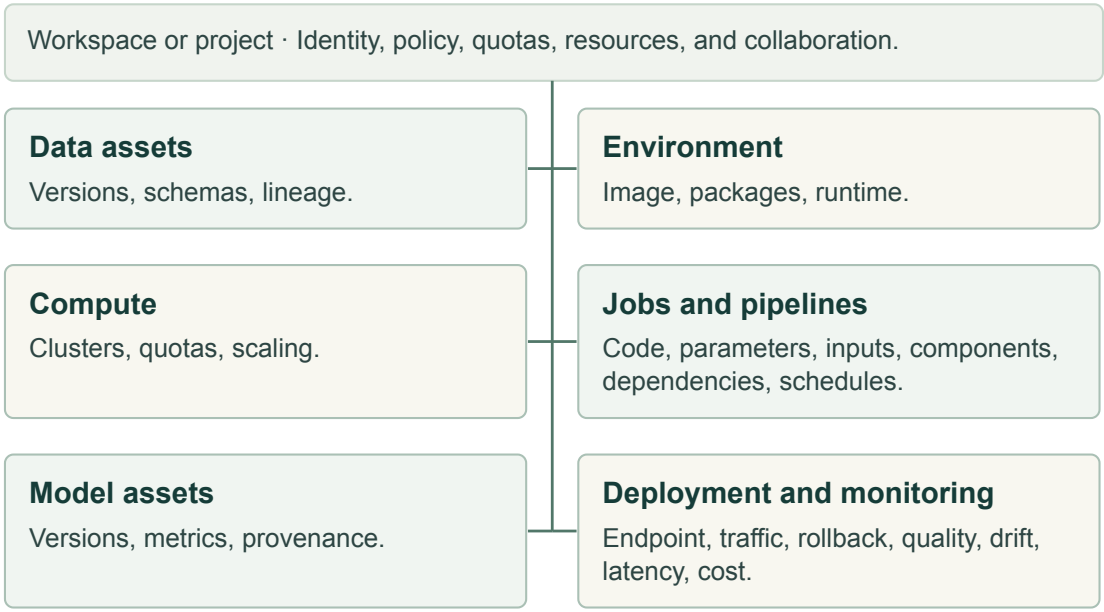


Figure 16-2 ML PaaS separates durable assets and policies from ephemeral jobs and runtime execution.

ML PaaS object	Durable contract	Typical evidence	Common failure when absent
Workspace / project	Ownership, members, policies, quotas, asset namespace	Role assignments, settings, activity history	No stable boundary for collaboration or responsibility
Data asset	Versioned reference, schema, location, classification	Lineage, data profile, access record	Training cannot be reproduced or audited
Environment / image	Runtime dependencies and execution contract	Image digest, package lock, build provenance	"Works on my machine" becomes a production defect
Compute / quota	Capacity family, accelerator, scale and allocation boundary	Allocation record, utilization, job placement	Uncontrolled cost or unavailable training capacity
Job / run	Code, inputs, parameters, environment, outputs, status	Logs, metrics, artifacts, timestamps	Results cannot be compared or traced

ML PaaS object	Durable contract	Typical evidence	Common failure when absent
Pipeline / component	Reusable steps, dependencies, schedules and interfaces	Run graph, component version, retry record	Glue code and manual handoffs dominate operations
Model asset / registry	Version, format, metrics, provenance and approval state	Model card, lineage, signature, evaluation record	A file is mistaken for a production model release
Endpoint / deployment	Invocation contract, model binding, traffic, scale and rollback	Deployment revision, request metrics, change log	Consumers depend on a mutable implementation
Monitor	Quality, drift, latency, failure and cost expectations	Signals, alerts, investigation and retraining evidence	Production degradation is discovered by users

16.3 Data+AI Platforms: One Estate, Several Lifecycles

The term Data+AI platform describes more than the availability of both SQL and Python in one portal. A coherent Data+AI platform connects enterprise data storage, ingestion, transformation, governance, analytics, feature computation, model development, deployment, and operational feedback without erasing the different lifecycles of those objects.

Azure Databricks is a clear example of the data-centric path. Microsoft describes it as a unified, open analytics platform for enterprise data, analytics, and AI. Its lakehouse model provides a shared data foundation for engineers, scientists, analysts, and production systems, while technologies such as Apache Spark, Delta Lake, MLflow, and catalog services support data processing and model work. [3] Azure Machine Learning adds a dedicated ML control plane for workspaces, jobs, assets, registries, managed endpoints, and MLOps. The two platforms can overlap in development and model operations, but their resource models and operating boundaries remain distinguishable.

Alibaba Cloud follows a similar layered logic with different boundaries. DataWorks integrates data ingestion, development, governance, and analytics across engines including MaxCompute, E-MapReduce, Hologres, Flink, AnalyticDB, and PAI. [11] PAI provides the ML lifecycle through DSW development environments, DLC training jobs, Designer workflows, Model Gallery, model management, and EAS inference services. [12] DataWorks can schedule PAI nodes, allowing data transformations and training or inference work to participate in one orchestration chain. The benefit is not a single monolithic product; it is explicit integration between data and model lifecycles.

Tencent Cloud TI-ONE presents a one-stop training and inference environment for AI engineers, while Tencent Cloud data services supply the surrounding data-engineering foundation. Volcengine exposes a data-intelligence path through VeDI and DataAgent, a model path through Ark, and an agent-runtime path through AgentKit. These examples demonstrate a recurring architecture: the data estate, model lifecycle, and application or agent lifecycle are connected but should not be collapsed into one undifferentiated platform box.

A useful architecture therefore asks where each lifecycle is governed. Data engineers may own transformation and quality. ML engineers may own training, model validation, and serving. Application teams may own prompts, retrieval policy, tool contracts, user experience, and release. Platform teams may own shared identity, network, observability, quotas, and approved patterns. The platform should connect these roles through resources and evidence rather than pretending that one UI eliminates them.

Layer	Primary managed objects	Dominant lifecycle	Typical owner
Data platform	Sources, tables, files, streams, catalogs, quality rules, lineage, schedules	Admit, transform, validate, publish, retain	Data engineering / governance
ML PaaS	Workspaces, data assets, environments, jobs, pipelines, models, endpoints	Experiment, train, evaluate, register, deploy, monitor, retrain	Data science / ML engineering
AI PaaS	Model services, prompts, knowledge, evaluations, safety policies, tools, agents	Compose, evaluate, publish, observe, revise, retire	AI application / platform engineering
Business application	User workflows, transactions, approvals, records and outcomes	Release, operate, support, improve	Product and application teams

16.4 What Generative AI Changes—and What It Does Not

Generative AI reduces the need to train a task-specific model for every application. A team can select a pretrained model, invoke it through a managed service, and improve the application through prompts, retrieval, tools, structured outputs, and evaluations. This is a profound change in developer workflow. It moves much of the application-specific behavior above the model weights.

The change creates new objects. A model catalog must describe providers, capabilities, versions, safety terms, regions, prices, and deployment options. A model deployment must bind a model version to capacity, identity, network, safety, and telemetry. Prompt and application configurations must be versioned and tested. Knowledge sources, ingestion pipelines, indexes, and access rules must be governed. Evaluations need datasets, metrics, graders, thresholds, and run evidence. Tools require schemas, identities, risk levels, approvals, and audit trails. Agents add definitions, runtimes, sessions, traces, memory, and release states.

What generative AI does not remove is equally important. Data still requires quality, lineage, access control, retention, and freshness. Production changes still require release gates, rollback, observability, and cost attribution. Fine-tuned or self-hosted models still need training, model management, and deployment operations. Even when no training occurs, the application still needs a lifecycle that can be audited and reproduced. The object that must be reproduced may now be a combination of model deployment, prompt version, knowledge release, tool policy, and agent version rather than a single model file.

The organizational center also shifts. Classical ML projects often aim to create and operate a model. Generative AI programs frequently operate many applications that share models while varying knowledge, policies, prompts, tools, and users. The platform must therefore allocate and govern reusable model services while preserving application-level isolation and evidence. This is why MaaS is necessary but insufficient: it solves managed model access, not the whole lifecycle of the AI application.

Continuity from ML PaaS	New emphasis in AI PaaS	Why both are needed
Versioned data and lineage	Knowledge sources, chunks, citations and conversation evidence	Model-backed answers still depend on governed data transformations
Experiment tracking	Prompt, retrieval, tool and agent evaluation runs	Behavior changes can occur without changing model weights
Model registry and deployment	Model catalog, MaaS deployment and routing	Teams may consume, fine-tune, or host different model families
Monitoring and drift	Trace, safety, groundedness, tool outcome and token economics	A correct endpoint can still produce an unsafe or unsupported answer
Pipelines and release gates	Knowledge refresh, evaluation gates and agent publication	Application behavior spans more than the training pipeline
Identity and network controls	Document eligibility, tool scope and agent execution identity	The authorization path becomes longer and more dynamic

The operating unit changed

The model remains important, but the production release is increasingly a compound of model, prompt, knowledge, tool, evaluation, safety, and agent resources.

16.5 The AI PaaS Resource Model

An AI PaaS should be judged by the resources and lifecycle obligations it exposes, not by whether its portal contains a chat playground. At minimum, a production-oriented platform needs a management scope, one or more model-service contracts, a way to manage application behavior, knowledge and tool resources, evaluation evidence, observability, policy, and economics.

The exact packaging varies. Microsoft Foundry now unifies agents, models, and tools under a single management grouping and applies role-based access control, networking, and policies through one Azure resource-provider namespace. It also provides tracing, monitoring, and evaluations at the project level. [5] Alibaba Cloud Model Studio emphasizes one-stop model services through Qwen and third-party models, with OpenAI-compatible APIs and application capabilities. [15] Tencent Cloud separates TI-ONE training and inference from ADP application and agent construction. Volcengine separates Ark model services from AgentKit runtime and the broader VeDI data-intelligence ecosystem. Product boundaries differ, but the resource obligations remain comparable.

A useful conceptual model treats AI PaaS as a control-plane expansion. ML PaaS resources remain available for training, fine-tuning, custom models, batch scoring, and MLOps. AI PaaS adds resources for consuming and composing models into applications. Not every provider implements every resource as a first-class object, and some capabilities may remain preview, region-limited, or embedded in an application builder. The architect should record that non-equivalence instead of filling gaps with assumptions.

AI PaaS extends the ML PaaS control plane

ML PaaS — Data assets, notebooks, training, experiments, model registry, endpoints, and MLOps.

AI PaaS — Model catalog and MaaS, prompts and app configuration, knowledge resources, tools, evaluations, agents, and observability.

Identity, network, policy, secret storage, evidence, cost, release, and retirement span both layers.

Figure 16-3 AI PaaS expands the resource graph while retaining the data and ML lifecycle beneath it.

AI PaaS resource family	Control-plane state	Data-plane action	Release evidence
Model catalog and model card	Provider, version, capability, terms, region, benchmark, retirement state	Discover and select a model	Selection rationale and accepted constraints
Model deployment / endpoint	Model binding, capacity, network, identity, safety, routing	Invoke inference	Deployment record, invocation telemetry and cost
Prompt or application configuration	Instructions, variables, response schema, version, owner	Construct requests and interpret responses	Evaluation result and promoted version
Knowledge resource	Sources, ingestion, schema, index, access policy, release	Retrieve grounding and references	Freshness, authorization and retrieval evaluation
Tool / connector	Schema, target, identity, scope, risk, approval, timeout	Read or mutate an external system	Plan, decision, invocation and outcome evidence
Evaluation suite and run	Dataset, metrics, graders, thresholds, environment	Execute quality and safety tests	Immutable results and release decision
Safety policy	Filter, classifier, thresholds, exceptions and escalation	Inspect or block content and actions	Policy version, decision and incident evidence
Agent definition and runtime	Model, instructions, tools, knowledge, memory, version, capacity	Plan, retrieve, call tools and return outcomes	Trace, evaluation gate, publication and rollback

```
Conceptual expansion:

ML PaaS = governed data + jobs + models + endpoints + MLOps

AI PaaS = ML PaaS foundation
          + governed model services
          + knowledge, prompts, tools, evaluations, safety, and agents
```

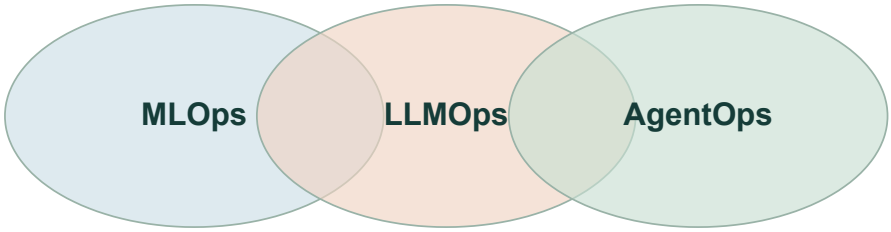
16.6 MLOps, LLMOps, and AgentOps Are Related Operating Loops

MLOps, LLMOps, and AgentOps are often presented as successive replacements. That framing is misleading. They govern different but overlapping sets of assets and failure modes. A single enterprise application may require all three.

MLOps remains essential when the organization trains, fine-tunes, registers, deploys, monitors, or retrains models. It emphasizes reproducible data and training, model lineage, deployment promotion, drift, and retraining. LLMOps emphasizes the lifecycle of pre-trained model use: model selection, prompt and context design, retrieval, evaluation, safety, inference observability, and token or throughput economics. AgentOps adds trajectories, tool invocations, memory, human approvals, action verification, and outcome-level evaluation.

The loops share platform capabilities. All need durable identity, versioned assets, environment and dependency control, release gates, observability, policy, cost attribution, incident response, and retirement. An organization that builds three disconnected operating stacks will duplicate evidence and create blind spots at the boundaries. The purpose of AI PaaS is not to rename every operation; it is to connect the loops through common resource and evidence contracts.

The operating loops of MLOps, LLMOps, and AgentOps



MLOps	Training data, features, experiments, models, and retraining.
LLMOps	Model selection, prompts, RAG, evaluations, and inference cost.
AgentOps	Tool calls, traces, memory, approvals, and verification of outcomes.

Shared controls: identity, lineage, versioning, release gates, observability, policy, cost, and rollback.

Figure 16-4 MLOps, LLMOps, and AgentOps overlap through shared control-plane capabilities.

Operating loop	Primary assets	Characteristic production question	Representative release gate
MLOps	Datasets, features, code, environments, jobs, models, endpoints	Can this model version be reproduced and safely replace the current deployment?	Quality, fairness, robustness, latency and drift criteria
LLMOps	Model deployments, prompts, knowledge, evaluation sets, safety policies	Does this application remain useful, grounded, safe and cost-effective across model and context changes?	Task quality, groundedness, safety, latency and cost thresholds
AgentOps	Agent versions, tools, memory, trajectories, approvals, outcomes	Can this agent be trusted to act within authority and produce verifiable business outcomes?	Trajectory, tool-scope, approval, outcome and rollback criteria

16.7 Azure Reference Path: Databricks, Machine Learning, and Foundry

Azure is useful as a teaching reference because the platform boundaries make the architectural progression visible. Azure Databricks provides a unified data, analytics, and AI environment built around the lakehouse. Azure Machine Learning provides a dedicated ML

project and MLOps control plane. Microsoft Foundry provides a consolidated resource model for model services, tools, knowledge, evaluations, and agents. The products overlap in selected development and model capabilities, but they should not be treated as three generations in which the newest invalidates the earlier ones.

A typical data-intensive AI system may use Azure Databricks to ingest and transform large datasets, manage lakehouse tables, compute features, and support analytics. Azure Machine Learning may manage specialized training jobs, reusable environments, model assets, registries, managed online or batch endpoints, and ML monitoring. Microsoft Foundry may expose model deployments, project-level application assets, evaluations, tools, knowledge connections, and agent services under a unified Azure control plane. [3][4][5]

The design choice should follow the lifecycle. If the dominant challenge is data engineering and lakehouse governance, begin with the data platform. If the system trains or operationalizes custom ML models, use the ML lifecycle platform. If the application primarily composes managed foundation models, retrieval, tools, and agents, use AI PaaS resources. Many production architectures use more than one. The goal is not to minimize the number of product names; it is to keep the ownership and evidence path coherent.

Microsoft Foundry's 2026 resource model is especially relevant to the thesis of this book. Microsoft documents a Foundry resource with projects and a unified management plane for models, agents, and tools, including RBAC, networking, policy, tracing, monitoring, and evaluation. [5] This consolidates earlier Azure AI patterns, but architects should still distinguish Foundry application resources from Azure Machine Learning assets and from the underlying data estate.

Azure layer	Best starting point when...	Key managed objects	Boundary to preserve
Azure Databricks	The system is dominated by data ingestion, lakehouse processing, analytics and data/AI collaboration	Catalogs, tables, notebooks, jobs, clusters or serverless compute, MLflow assets	Do not assume the data workspace is the complete application or agent control plane
Azure Machine Learning	The organization trains, fine-tunes, registers, deploys or monitors custom models	Workspace, data and environment assets, jobs, pipelines, models, endpoints, monitors	Do not reduce MLOps to a model catalog or prompt workflow
Microsoft Foundry	The application composes managed models, knowledge, tools, evaluations and agents	Foundry resource, projects, deployments, evaluations, tools, knowledge and agent versions	Do not detach AI application resources from governed data and production ownership

16.8 Alibaba Cloud Path: DataWorks, PAI, and Model Studio

Alibaba Cloud exposes a similarly layered path. DataWorks is a comprehensive data platform that integrates with big-data and AI engines and provides data integration, development, governance, and analytics for warehouses, lakes, and lakehouses. [11] PAI is an end-to-end AI development platform that covers data labeling, development, training, deployment, and AI operations. Its core modules include DSW for interactive development, DLC for distributed training, Designer for visual workflows, EAS for online inference, and Model Gallery for model-oriented workflows. [12]

PAI also illustrates how the ML PaaS control plane reaches into infrastructure. Alibaba Cloud documents resource pools, quotas, workspace bindings, general-purpose resources, and high-performance Lingjun resources that supply DSW, DLC, and EAS. [14] The important abstraction is the allocation contract: capacity is purchased or provisioned, divided through quotas, bound to workspaces, and consumed by lifecycle objects. This is more governable than asking individual teams to acquire arbitrary GPU instances.

Model Studio shifts the focus toward managed model services and generative AI applications. Alibaba Cloud describes it as a one-stop model service platform that provides Qwen and third-party models through official and OpenAI-compatible APIs, with multimodal support and on-demand invocation. [15] It also provides model fine-tuning and application-oriented capabilities. [16] In architectural terms, Model Studio adds MaaS and AI application resources above the DataWorks and PAI foundation; it does not make data development, distributed training, or model operations unnecessary.

The product boundaries and regional availability of application-development features can change. For a durable architecture, record the exact resource type, region, lifecycle API, identity model, evidence, and production status used by the workload. “Available in the portal” is not a sufficient control-plane contract.

16.9 Tencent Cloud Path: TI-ONE and ADP

Tencent Cloud TI-ONE provides a one-stop training and inference platform for AI engineers. Its platform documentation covers development machines, training tasks, model management, inference services, built-in frameworks, and purchasing or allocation of compute resources. [17][18] The resource intent is recognizably ML PaaS: a managed environment for developing, training, registering, optimizing, and serving models.

Tencent Cloud ADP operates at the AI application and agent layer. The 2026 product overview describes an enterprise AI application platform that supports business-oriented agents and multiple construction modes, including standard agents, workflows, multi-agent configurations, and an independent-workspace mode that can run code and invoke skills, connectors, and tools. It also supports APIs, SDKs, and enterprise-channel integration. [19]

The architectural handoff is therefore not "TI-ONE was replaced by ADP." TI-ONE governs model development and serving lifecycles; ADP governs application and agent composition. Data services and enterprise systems remain upstream and downstream dependencies. A workload that trains a domain model, builds a governed knowledge base, and publishes a multi-agent workflow may cross all three layers.

16.10 Volcengine Path: VeDI, Ark, AgentKit, and DataAgent

Volcengine makes the separation between data intelligence, model service, and agent runtime especially visible. VeDI provides the broader data-intelligence environment and related data products. DataAgent is positioned as an enterprise AI data expert that uses organizational data assets for analysis, insight, and action. [24] It should be treated as a data-intelligence solution layer, not as a generic substitute for every data-engineering or AI platform resource.

Ark provides the model-service layer. Volcengine documents inference, evaluation, fine-tuning, and related model lifecycle services under the Ark platform. [20][21] AgentKit adds a cloud-hosted agent runtime, CLI-based lifecycle operations, tools, evaluation, and observability. Its CLI supports initialization, configuration, build, deployment, and operations, while the runtime is the hosted execution and lifecycle unit for an agent. [22][23]

The resulting path is consistent with the chapter thesis: VeDI and related data services govern data and business semantics; Ark governs model services and optimization; AgentKit governs agent execution and lifecycle; DataAgent packages selected capabilities into enterprise data-intelligence experiences. The layers can be integrated without pretending they are one resource type.

Four provider paths converge on the same engineering layers without becoming identical

Azure

Data services → Azure Machine Learning → Microsoft Foundry.

Alibaba Cloud

Data services and DataWorks → PAI → Model Studio.

Tencent Cloud

Cloud data services → TI-ONE → ADP applications.

Volcengine

Data intelligence and DataLeap → Ark model platform → AgentKit runtime.

Compare the engineering contracts at each layer. Products differ in scope and managed responsibility.

Figure 16-5 Azure, Alibaba Cloud, Tencent Cloud, and Volcengine expose different product boundaries around comparable data, ML, model-service, and agent layers.

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Data development and governance	Azure Databricks plus Azure data and governance services	DataWorks with integrated data engines	Tencent Cloud data and analytics services	VeDI and related data products
Interactive AI development	Azure ML notebooks / Databricks notebooks / Foundry project tools	PAI DSW / Designer	TI-ONE development machines and workbenches	Ark development capabilities and platform tools
Training and optimization	Azure Machine Learning jobs and compute	PAI DLC, Designer, Model Gallery	TI-ONE training and optimization	Ark fine-tuning and model services
Model deployment and inference	Azure ML managed endpoints / Foundry model deployments	PAI EAS / Model Studio model APIs	TI-ONE inference and managed model services	Ark online inference and model units
Knowledge and generative application	Foundry knowledge, tools and project resources	Model Studio knowledge and application capabilities	ADP knowledge and application modes	Ark and AgentKit knowledge/tool integration

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Agent runtime and lifecycle	Foundry Agent Service	Model Studio / PAI agent application capabilities	ADP	AgentKit
Provider-specific caution	Do not collapse Databricks, Azure ML and Foundry into one workspace contract	Do not infer feature or region parity between PAI and Model Studio	Do not treat ADP as a replacement for the model lifecycle	Do not treat DataAgent, Ark and AgentKit as one generic resource

16.11 Architecture Worksheet: Separate the Resource Families

This chapter does not include a full deployment lab. Its practical exercise is an architecture worksheet. The objective is to prevent the phrase “AI platform” from hiding incompatible lifecycles and responsibility boundaries before a provider or product is selected.

Choose one use case, such as a service-operations knowledge assistant, a manufacturing-quality analyst, a financial-risk explanation service, or a cloud-cost recommendation assistant. Describe the business outcome in one sentence. Then inventory the resource families separately. Do not start with product names.

Architecture worksheet: separate resource families before choosing products

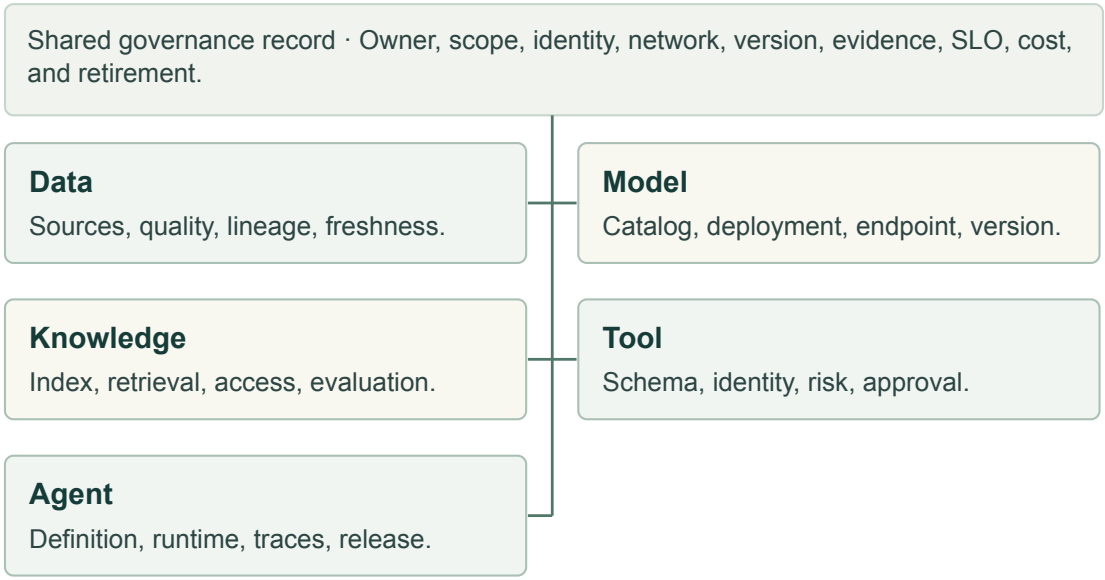


Figure 16-6 The worksheet separates data, model, knowledge, tool, and agent resources and reconnects them through a shared governance record.

Worksheet field	Questions to answer	Minimum artifact
Business outcome	Who receives value, what decision or task improves, and how will success be measured?	One outcome statement and one unit-of-value metric
Data resources	Which sources, transformations, owners, classifications, freshness and lineage are required?	Source inventory and data-flow sketch
Model resources	Is the model consumed, fine-tuned or trained? Which deployment, region, capacity and retirement risks apply?	Model decision record
Knowledge resources	Which content becomes retrievable, how is it admitted, refreshed, authorized, evaluated and cited?	Knowledge lifecycle and eligibility rule
Tool resources	Which systems can be read or changed, through which identity, with what risk and approval?	Tool capability card and authority boundary

Worksheet field	Questions to answer	Minimum artifact
Agent or application resources	Which definition, workflow, runtime, memory, session and release objects exist?	Agent/application object model and lifecycle
Evaluation	Which datasets, metrics, graders and thresholds gate release?	Evaluation suite and release gate
Shared governance	Who owns each resource, how is it networked, observed, costed, retained and retired?	Responsibility matrix and evidence plan
Provider mapping	Which product provides each resource, and where is there no operational equivalence?	Provider map with non-equivalence notes

Acceptance rule

The worksheet is complete only when every resource family has an owner, lifecycle, identity path, evidence source, cost boundary, and retirement action. A product logo is not an architecture decision.

16.12 Design Rules and Anti-Patterns

AI PaaS adoption fails less often because a model endpoint cannot be called than because the surrounding resource graph is not designed. The following anti-patterns are especially common.

Anti-pattern	Why it fails	Preferred design rule
Model API monoculture	Every requirement is reduced to selecting and calling a model	Model access is one data-plane capability inside a larger governed application lifecycle
The monolithic “AI platform” box	Data, model, knowledge, tool and agent ownership become invisible	Separate resource families and reconnect them through explicit contracts
Training pipeline as the whole application	Prompt, retrieval, tools and user workflows are excluded from release evidence	Add LLMOps and AgentOps resources to the release graph
Portal-driven assets without source control	The effective production state cannot be reviewed or reproduced	Export or define durable configuration, version it, and record promotion evidence
Duplicate data planes	Each team copies data into private indexes and notebooks with inconsistent semantics	Govern shared sources and transformations while preserving workload isolation

Anti-pattern	Why it fails	Preferred design rule
RAG equals vector database	Authorization, freshness, provenance, evaluation and rollback are omitted	Treat knowledge as a managed resource lifecycle
Agent equals prompt plus tools	Identity, tool authority, trace, evaluation, memory and retirement are omitted	Treat agents as governed cloud resources
No retirement path	Old models, indexes, endpoints, prompts and agents remain callable and billable	Define deprecation, migration, retention, deletion and purge before production
Product-name equivalence	Similar labels conceal different resource, API, evidence and operational contracts	Translate engineering intent and record non-equivalence

16.13 CloudDesk Reuse: Platform Inventory and Lifecycle Gap Assessment

CloudDesk FinOps Agent may later use the resource model in this chapter to explain why an AI workload costs what it costs and where governance gaps exist. This chapter does not implement the agent. It defines the read-only capabilities that a later tool layer can safely reuse.

A platform inventory tool should classify resources by family rather than merely list provider object types. A lineage tool should connect data transformations to models, knowledge indexes, applications, and agents. A lifecycle-gap assessor should detect resources without owners, versions, evaluation evidence, monitoring, cost attribution, or retirement plans. These capabilities are useful to FinOps because orphaned and duplicated AI assets are both economic waste and governance risk.

```
capability: ai.platform.architecture.map
risk_level: R0_read_only
inputs:
  scope: cloud account, subscription, project, or workspace
  resource_families: [data, ml, model, knowledge, tool, evaluation, agent]
outputs:
  resource_graph: []
  ownership_gaps: []
  lifecycle_gaps: []
  duplicated_capabilities: []
  evidence_sources: []
constraints:
```

```
  - never infer equivalence from product names alone
  - preserve provider resource IDs and regions
  - distinguish missing evidence from evidence of absence
```

Capability	Purpose	Minimum permissions
data.lineage.inspect	Explain upstream and downstream dependencies for data, features, indexes and model assets	Read catalog, pipeline, job and lineage metadata
ml.asset.inventory	Inventory workspaces, jobs, models, endpoints and monitoring objects	Read ML PaaS control-plane metadata
ai.platform.architecture.map	Classify provider resources into common resource families and record non-equivalence	Read model, knowledge, evaluation, tool and agent metadata
ai.lifecycle.gap.assess	Identify resources without ownership, evaluation, evidence, cost attribution or retirement plans	Read-only access to configuration, activity, observability and billing metadata

16.14 Key Takeaways

- Data services are the foundation of AI engineering because they govern quality, lineage, authorization, freshness, and retention for both training and retrieval.
- ML PaaS productizes the machine-learning lifecycle through workspaces, assets, jobs, pipelines, registries, endpoints, monitoring, and MLOps evidence.
- AI PaaS is a control-plane expansion, not a replacement: it adds managed model services, prompts, knowledge, tools, evaluations, safety policies, and agents.
- MLOps, LLMOps, and AgentOps govern overlapping asset sets and should share identity, versioning, observability, policy, cost, release, and retirement contracts.
- Azure Databricks, Azure Machine Learning, and Microsoft Foundry illustrate distinct but connected data, ML, and AI-application lifecycles.
- Alibaba Cloud, Tencent Cloud, and Volcengine expose comparable engineering layers through different product boundaries; non-equivalence must be documented.
- A sound architecture separates data, model, knowledge, tool, evaluation, and agent resources before selecting provider products.
- The mature unit of AI engineering is no longer a model file or API call. It is a governed resource graph with owners, evidence, economics, and a retirement path.

References

[1] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1, 2023. https://doi.org/10.6028/NIST.AI.100-1

[2] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile, NIST AI 600-1, 2024. https://doi.org/10.6028/NIST.AI.600-1

[3] Microsoft. What is Azure Databricks? Microsoft Learn. Updated July 28, 2026. https://learn.microsoft.com/en-us/azure/databricks/introduction/

[4] Microsoft. What is Azure Machine Learning? Microsoft Learn. Updated September 9, 2025. https://learn.microsoft.com/en-us/azure/machine-learning/overview-what-is-azure-machine-learning

[5] Microsoft. What is Microsoft Foundry? Microsoft Learn. Updated July 20, 2026. https://learn.microsoft.com/en-us/azure/foundry/what-is-foundry

[6] Microsoft. Microsoft Foundry Models overview. Microsoft Learn. Updated July 28, 2026. https://learn.microsoft.com/en-us/azure/foundry/concepts/foundry-models-overview

[7] Microsoft. MLOps in Azure Machine Learning. Microsoft Learn. https://learn.microsoft.com/en-us/azure/machine-learning/concept-model-management-and-deployment

[8] MLflow Project. MLflow Tracking and Model Registry documentation. https://mlflow.org/docs/latest/

[9] OpenLineage Project. OpenLineage documentation. https://openlineage.io/docs/

[10] Microsoft. Data lineage in Microsoft Purview. Microsoft Learn. https://learn.microsoft.com/en-us/purview/concept-data-lineage

[11] Alibaba Cloud. What is DataWorks? Updated January 12, 2026. https://www.alibabacloud.com/help/en/dataworks/user-guide/what-is-dataworks

[12] Alibaba Cloud. What is Platform for AI (PAI)? Updated June 23, 2026. https://www.alibabacloud.com/help/en/pai/product-overview/what-is-machine-learning-platform-for-ai/

[13] Alibaba Cloud. PAI product architecture. Updated June 11, 2026. https://www.alibabacloud.com/help/en/pai/product-overview/service-architecture

[14] Alibaba Cloud. AI computing resources overview. Updated July 9, 2026. https://www.alibabacloud.com/help/en/pai/overview-of-ai-computing-resources

[15] Alibaba Cloud. What is Alibaba Cloud Model Studio? Updated July 10, 2026. https://www.alibabacloud.com/help/en/model-studio/what-is-model-studio

[16] Alibaba Cloud. Introduction to model fine-tuning. Updated July 14, 2026. https://www.alibabacloud.com/help/en/model-studio/model-training-overview

[17] Tencent Cloud. TI-ONE Platform Overview. Updated 2026. https://www.tencentcloud.com/document/product/1141/45261

[18] Tencent Cloud. Introduction to Model Management in TI-ONE. Updated 2026. https://www.tencentcloud.com/document/product/1141/77102

[19] Tencent Cloud. Agent Development Platform Product Overview. Updated July 23, 2026. https://www.tencentcloud.com/document/product/1254/69956

[20] Volcengine. Ark product documentation: inference, evaluation, and fine-tuning services. https://docs.volcengine.com/docs/82379

[21] Volcengine. Create a model evaluation task in Ark. https://www.volcengine.com/docs/82379/1153149

[22] Volcengine. AgentKit CLI overview and lifecycle management. Updated June 23, 2026. https://www.volcengine.com/docs/86681/2085680

[23] Volcengine. AgentKit data observability. Updated August 10, 2026. https://www.volcengine.com/docs/86681/1873528

[24] Volcengine. DataAgent product documentation. https://docs.volcengine.com/docs/86760?lang=zh

[25] Databricks. What is a data lakehouse? https://docs.databricks.com/en/lakehouse/index.html

[26] Databricks. Unity Catalog and data governance documentation. https://docs.databricks.com/en/data-governance/unity-catalog/index.html

[27] Feast Project. Feature Store documentation. https://docs.feast.dev/

[28] ISO/IEC 5338:2023. Information technology - Artificial intelligence - AI system life cycle processes.

[29] ISO/IEC 42001:2023. Information technology - Artificial intelligence - Management system.

[30] CNCF. Cloud Native Artificial Intelligence white papers and landscape resources. https://www.cncf.io/

Draft boundary note

This chapter intentionally stops at platform architecture and resource families. Chapter 17 governs model resources in depth; Chapter 18 governs knowledge; Chapter 19 governs agent platforms; Chapter 20 governs cloud capabilities as agent tools. Custom serverless agent runtimes and tenant isolation belong to the separate book, Serverless AI Agent Development.

Part V / Chapter 17

From Model APIs to AI PaaS Resources

Treating Inference as a Governed Resource with Version, Identity, Quota, Network, Telemetry, Safety, and Cost Semantics

A model API proves that software can send a prompt and receive a response. An AI PaaS resource proves that the same capability can be named, versioned, authorized, isolated, observed, costed, changed, and retired without losing operational accountability.

In This Chapter

- Why a successful inference request is a data-plane event rather than a complete production architecture
- The model-resource contract: model identity, deployment, endpoint, authorization, network, quota, safety, telemetry, billing, and lifecycle
- Why a catalog model, a model version, a deployment, an endpoint, a project, and an AI PaaS resource are different objects
- How MaaS contributes managed inference without, by itself, supplying the full AI PaaS control plane
- How shared online, regional, data-zone, provisioned, batch, dedicated, fine-tuned, and custom deployments express different operating contracts
- Why API keys are convenient bootstrap credentials but weak long-lived identities for governed production workloads
- How region, private networking, data-processing scope, and outbound dependencies define the real inference boundary
- Why quota, rate limits, capacity, backpressure, budget, and service objectives must not be treated as synonyms
- How model versions, upgrade policies, canaries, rollback, deprecation, and retirement form a managed lifecycle
- How guardrails and safety policies become versioned resources associated with deployments and agents

- Which metrics and evidence are required to operate inference as a service rather than as an opaque API call
- How to inspect and invoke a managed model deployment, then assemble a verifiable model-resource evidence packet
- How Azure, Alibaba Cloud, Tencent Cloud, and Volcengine expose comparable engineering intent through different resource models

The first model call in a new application is often exhilarating. A few lines of code produce fluent output, and the development team has proof that the idea is technically possible. That proof is important, but it answers only a narrow question: can the application reach a model endpoint and receive a response? It does not answer which model version served the request, which principal was authorized, where the request could be processed, what quota pool constrained it, which safety policy evaluated it, how its latency and token usage were measured, what it cost, or what happens when the model version is retired.

Those unanswered questions are not secondary administration. They define whether inference can be entrusted with a production role. AI PaaS begins where an isolated API call stops: it turns model access into a governable cloud resource with explicit control-plane state and inspectable data-plane evidence. The resource may still expose an OpenAI-compatible or provider-specific API, but the API is now one interface to a larger managed contract.

From an API Call to a Governed Model Resource

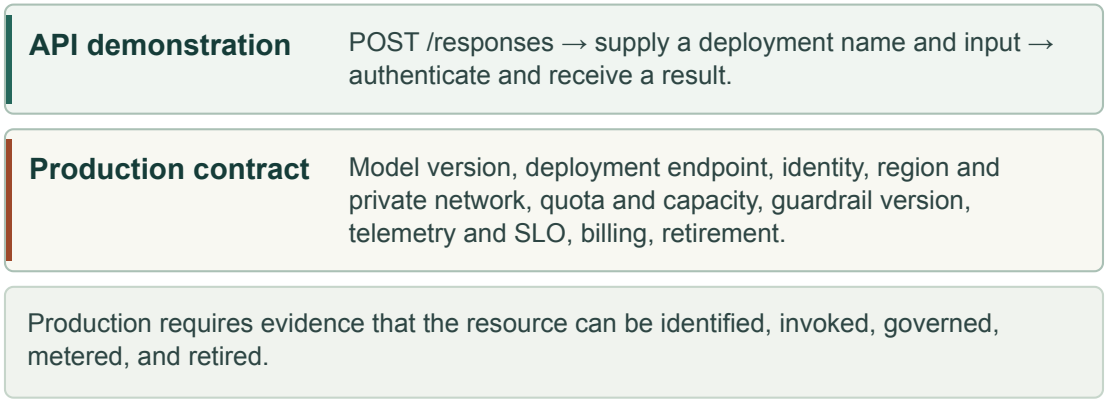


Figure 17-1 A model API demonstrates invocation; a model resource supplies the surrounding production contract.

17.1 A Model API Is a Data-Plane Contract, Not a Lifecycle

An inference API defines how a caller submits input and receives output. It specifies a protocol: an endpoint, an authentication mechanism, request fields, streaming behavior, tool-call structures, and response objects. This is a data-plane contract. The request is an operation performed by an already-existing service.

The control plane answers a different class of questions. It creates the service boundary in which inference is allowed to occur. It records the deployment name, model version, capacity mode, region, network access, role assignments, safety configuration, diagnostic routing, quota allocation, and deletion state. It also makes these decisions visible to infrastructure-as-code, policy, inventory, audit, and cost systems.

Table 17-1 Data-plane questions and control-plane questions

Data plane: performing inference	Control plane: governing inference
What endpoint accepts the request?	Which AI resource and deployment own the endpoint?
What request schema does the API expect?	Which model name, version, format, and deployment type are approved?
How is the request authenticated?	Which principals may deploy, invoke, evaluate, monitor, or retire the resource?
What response, stream, or tool call is returned?	Which guardrail, telemetry, quota, and cost policies bind the operation?
Did this request succeed?	Can the organization explain change history, failures, throttling, cost, and retirement?

A common architectural mistake is to treat the API key and base URL as the service. They are credentials and addressing information, not the service boundary. If a team cannot inventory the deployment independently of its application code, it cannot reliably govern that deployment. If the deployment can change version without leaving operational evidence, the application has no stable explanation for behavior changes. If all callers share one unrestricted key, authorization is reduced to possession of a secret.

NOTE: This chapter uses model resource as a provider-neutral engineering term. Cloud platforms use different names - model deployment, inference service, endpoint, online service, model unit, or model endpoint. The important question is not the product label but whether the object has explicit lifecycle, identity, network, quota, telemetry, safety, and billing semantics.

17.2 The Model-Resource Contract

A production deployment should be describable without reading the application source code. The description is the model-resource contract. It is not necessarily one physical resource. It may be represented by an AI account, project, deployment child resource, endpoint, role assignments, private endpoints, diagnostic settings, policy bindings, and billing records. Together these objects create the governed unit consumed by applications.

Table 17-2 Minimum fields in a model-resource contract

Contract field	Operational question	Evidence
Resource identity	What cloud resource owns the deployment?	Provider resource ID, account, project, region
Model identity	What model and immutable or provider-defined version are in use?	Model name, format, version or snapshot
Deployment identity	What stable name do callers use?	Deployment or endpoint name and resource ID
Capacity contract	How is capacity supplied and limited?	Deployment type, SKU, capacity units, concurrency or throughput settings
Authorization	Who may deploy, invoke, inspect, or delete?	Role assignments, policies, workload identity
Network boundary	Where can traffic enter and leave?	Public access state, private endpoint, DNS and egress rules
Processing boundary	Where may provider processing occur?	Region, data zone, global scope, documented provider behavior
Safety policy	Which controls evaluate input, output, and tools?	Guardrail or responsible-AI policy name and version
Telemetry contract	Which signals are collected and retained?	Metrics, logs, traces, request IDs, diagnostic destinations
Billing contract	What units create cost?	Input/output tokens, capacity units, hours, jobs, data movement

Contract field	Operational question	Evidence
Upgrade policy	How may the underlying model change?	Pinned version, auto-upgrade option, canary plan
Retirement contract	How is deprecation detected and completed?	Provider notice, inventory query, migration evidence, deletion record

The contract establishes a crucial separation: the application asks for a deployment name, while operations retain the ability to inspect what that name means. A stable deployment name can prevent every caller from hard-coding a volatile model identifier. Stability at the caller boundary, however, must not erase version identity from audit, telemetry, and release records.

This contract also makes platform automation possible. Policy can reject public access. Inventory can find deployments using an obsolete model version. FinOps tooling can correlate token consumption with a project and business owner. Observability can distinguish latency regression caused by a version change from regression caused by an application release. Agent tooling can be limited to read-only inspection or to approved deployment actions.

17.3 Model, Version, Deployment, Endpoint, Project, and Resource

AI platforms frequently present several names close together, encouraging teams to collapse them into one mental object. They should remain distinct.

Table 17-3 Objects in the managed-inference resource graph

Object	Meaning	Typical change frequency
Catalog model	A provider-published capability or model family that can be selected.	Provider controlled; catalog changes frequently
Model version	A specific provider release, snapshot, or revision of the model.	Changes when the provider publishes or retires versions
Deployment	A customer-governed binding of model version, capacity mode, policy, and stable deployment name.	Changed through controlled release operations
Endpoint	The data-plane address and protocol used for inference.	Usually stable; routes may change behind it

Object	Meaning	Typical change frequency
Project or workspace	A collaboration and development boundary for applications, agents, evaluation, connections, and artifacts.	Changes with team and solution organization
AI PaaS resource or account	The top-level cloud boundary for security, network, billing, quota, and model deployments.	Long lived; governed with cloud resource management

The project is especially easy to misunderstand. A project can isolate development assets, connections, evaluation runs, and team permissions, but it does not automatically create a physically dedicated model for each project. Conversely, a deployment can be shared by several applications even when their project artifacts remain separate. The resource hierarchy must be inspected rather than inferred from portal navigation.

AI PaaS model resource graph

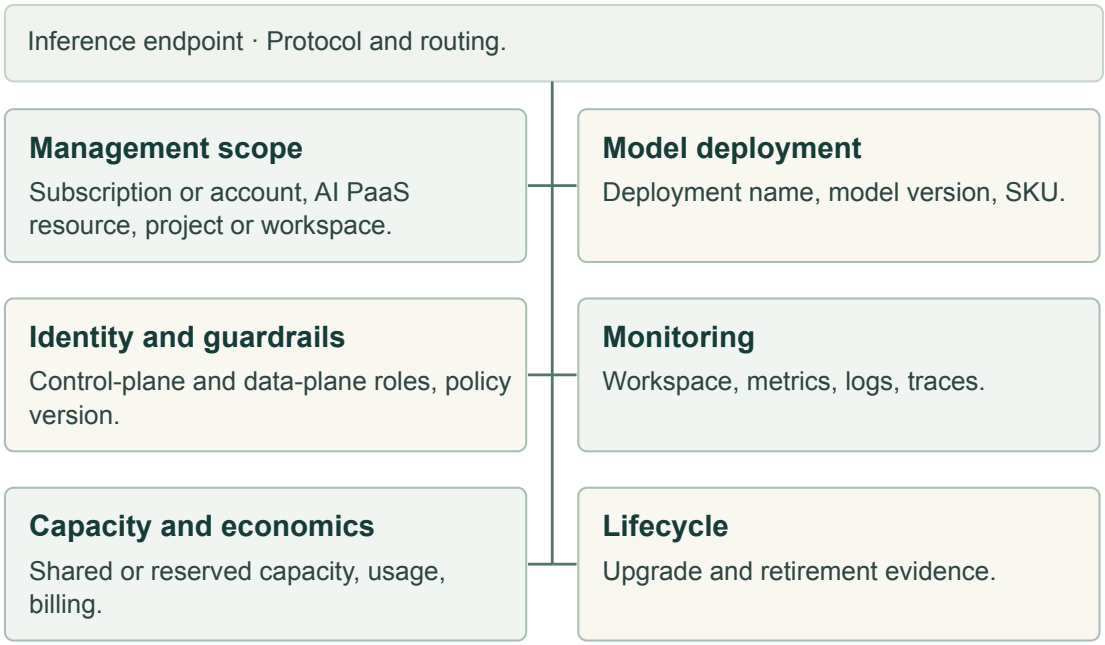


Figure 17-2 A provider-neutral model-resource graph, using the Azure Foundry hierarchy as the reference shape.

17.4 MaaS Is a Layer, Not the Whole AI PaaS

Model as a Service (MaaS) removes the burden of training, hosting, and scaling a base model. The caller sends requests to a managed endpoint and usually pays by tokens, requests, images, audio duration, or another consumption unit. MaaS is a major platform advance because it converts scarce model infrastructure into an on-demand capability.

MaaS alone, however, does not define the complete application engineering environment. AI PaaS adds the surrounding system needed to build and operate AI-native applications: projects, model catalog and deployments, identity, private networking, knowledge, evaluation, prompt and agent assets, tool connections, safety, telemetry, policy, cost, and lifecycle automation. A provider may market these capabilities under one studio or platform name, but the engineering distinction remains useful.

Table 17-4 MaaS and AI PaaS are related but not identical

Capability	MaaS contribution	AI PaaS extension
Inference	Managed model execution and scaling	Deployment lifecycle, routing, policy, evaluation, and operational evidence
Identity	API key or service authentication	Cloud IAM, workload identity, scoped roles, separation of duties
Data and knowledge	Request payloads and optional file interfaces	Governed data connections, knowledge resources, ingestion and retrieval
Application engineering	SDK and API compatibility	Projects, agents, tools, workflows, tracing, release and governance
Operations	Provider service availability	Customer SLOs, quotas, diagnostics, cost allocation, change and retirement

This distinction prevents a false architectural conclusion: because an application can call a model through a compatible API, the organization already has an AI platform. API compatibility reduces integration friction. It does not automatically establish ownership, isolation, quality, change control, or cost accountability.

17.5 Deployment Families Are Operating Contracts

The same model family may be available through several deployment types. These are not merely purchasing options. They change where requests may be processed, how capacity is supplied, how latency behaves under load, how cost is calculated, and which operational responsibilities remain with the customer.

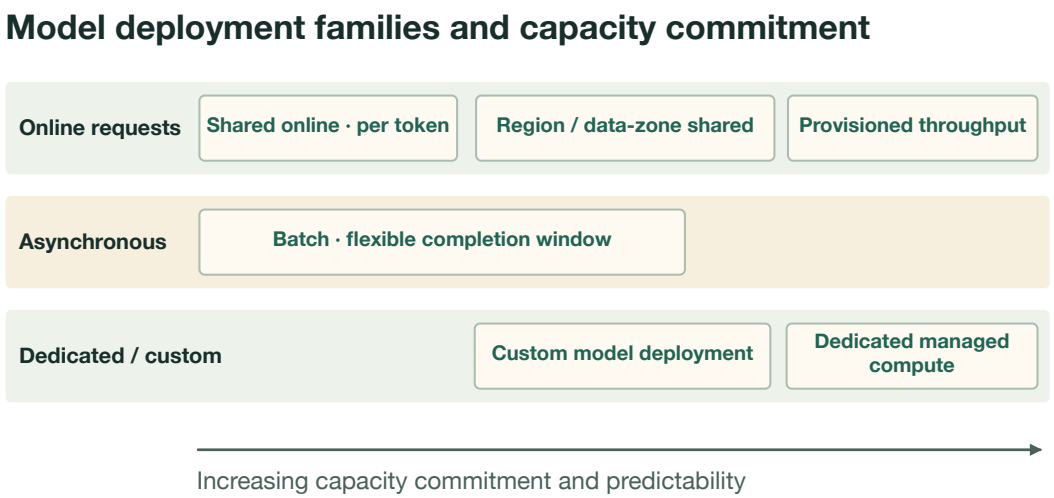


Figure 17-3 Conceptual deployment-family trade space. Product names change; capacity and latency trade-offs persist.

Table 17-5 Common deployment families

Deployment family	Best fit	Primary engineering trade-off
Shared online, consumption based	Interactive prototypes and variable production demand	Fast adoption and elasticity; latency and throughput are less predictable under shared capacity
Regional or data-zone shared online	Workloads with stated processing-boundary requirements	Narrower processing scope may reduce model or capacity choices
Provisioned throughput or reserved capacity	Stable high-volume workloads with strict throughput objectives	Capacity commitment improves predictability but creates utilization risk
Batch or asynchronous inference	Offline enrichment, evaluation, migration, and large noninteractive jobs	Lower urgency and often better unit economics in exchange for delayed completion
Dedicated managed compute	Specialized models, dedicated accelerators, custom containers, or stronger runtime control	More explicit compute sizing, warm capacity, patching boundaries, and hourly cost
Fine-tuned or custom model deployment	Domain behavior that cannot be achieved by prompting and retrieval alone	Adds training data governance, version lineage, evaluation, and model lifecycle

A useful decision begins with workload shape rather than model popularity. Interactive chat needs time-to-first-token and tail-latency objectives. Batch document classification needs completion time and job failure recovery. Agent tool selection needs low latency and

predictable rate-limit behavior. Evaluation workloads need repeatability and cost controls. A single deployment type rarely optimizes all four.

Processing scope is also part of the contract. A deployment described as global may allow the provider to route inference across a broader geography while retaining data-handling commitments defined by that service. A regional or data-zone deployment narrows the permitted processing boundary but may have different capacity or model availability. The exact semantics are provider-specific and must be verified in current official documentation.

17.6 Identity: From Keys to Workload Principals

API keys are attractive because they compress authentication into one string. They are useful during initial exploration, but their convenience hides several governance weaknesses. A key usually represents the resource rather than the workload. It is difficult to distinguish which application performed a call, difficult to constrain with fine-grained roles, and easy to copy into local configuration, notebooks, CI logs, or desktop utilities.

A production design should prefer a workload identity that can obtain short-lived tokens and be authorized through the cloud identity system. In Azure, Microsoft Entra ID and Azure RBAC allow separate principals to manage the Foundry resource, create deployments, invoke inference, inspect monitoring, or consume agents. The same general intent appears in Alibaba Cloud RAM, Tencent Cloud CAM, and Volcengine IAM even though role names and resource scopes differ.

Table 17-6 Separate identities by responsibility

Principal	Required capability	Capability that should normally be absent
Platform deployer	Create or update AI resources, projects, deployments, policies, and diagnostics	Routine application inference and business data access
Inference workload	Invoke an approved deployment and read only required runtime metadata	Create deployments, assign roles, change networking, delete resources
Evaluator	Invoke approved deployments, write evaluation artifacts, read quality telemetry	Production mutation tools and unrestricted secrets
Operations reader	Read deployment state, metrics, quota, cost, health, and retirement notices	Inference content unless incident policy explicitly permits it

Principal	Required capability	Capability that should normally be absent
Break-glass administrator	Time-bound emergency control-plane authority with strong audit	Standing daily access

Identity also distinguishes management-plane and data-plane access. A principal able to view a resource in the cloud portal may still be unable to invoke the model. Conversely, a caller holding a broad API key may invoke the model without being represented by a narrowly scoped cloud role. This asymmetry must be tested, not assumed.

NOTE: A secret manager improves storage and rotation of a key; it does not turn the key into a workload identity. The stronger pattern is to eliminate the long-lived secret when the platform supports token-based workload authentication, and to use a secret store only for dependencies that cannot yet participate in federation or managed identity.

17.7 Network and Processing Boundaries

A managed model may be externally hosted yet still participate in a private application architecture. The important distinctions are inbound reachability, name resolution, outbound dependencies, and provider processing scope.

Table 17-7 Four boundaries that must be documented

Boundary	Control question	Typical mechanism
Inbound network	Can the endpoint be reached from the public internet?	Public-network flag, firewall rules, private endpoint, private DNS
Caller egress	From which subnets, services, or gateways can the workload send requests?	Network security policy, egress firewall, service endpoints, proxy
Provider processing	In which geography may the provider process the inference request?	Regional, data-zone, or global deployment contract
Connected services	What storage, search, tool, telemetry, or safety services can the AI platform call?	Managed private endpoints, service identity, allowlist, connection resource

Private connectivity is a control-plane decision with data-plane consequences. In Azure, the AI resource can disable public network access and expose a private endpoint into a virtual network. Private DNS causes the ordinary service hostname to resolve to a private ad-

dress from inside the network. The application should therefore keep one logical endpoint while network policy determines the permitted route.

Private ingress does not automatically constrain every outbound path. If an agent can call arbitrary tools, if a project connects to external storage, or if evaluation jobs need package repositories, those paths require separate design. Network isolation is not a checkbox on the model endpoint; it is a graph property of the complete AI PaaS solution.

17.8 Quota, Rate Limits, Capacity, Backpressure, and Budget

Managed inference introduces several numerical controls that are often conflated. They serve different purposes. Quota grants eligibility to consume capacity. A rate limit protects shared service operation. Deployment capacity allocates or reserves a portion of service capability. Backpressure protects the application when requests exceed usable throughput. A budget reports financial thresholds. An SLO defines acceptable service behavior.

Table 17-8 Similar-looking controls with different meanings

Control	Question answered	What it does not guarantee
Quota	How much of a model or deployment class may this subscription, account, project, or region allocate?	Actual throughput, low latency, or low cost
Rate limit	How quickly may requests or tokens arrive before throttling?	Reserved capacity or business-priority scheduling
Provisioned capacity	What throughput has been committed or reserved?	Efficient utilization or correct workload design
Backpressure	How does the caller queue, shed, retry, or degrade when usable capacity is exhausted?	Additional provider capacity
Budget	When should accumulated cost trigger notification or workflow?	Automatic prevention of inference or hard spend enforcement
SLO	What latency, availability, error, or quality behavior is acceptable?	A mechanism for creating capacity by itself

Quota is frequently assigned at a provider-defined scope such as subscription, account, region, model family, version, or deployment type. The scope can change as platforms evolve. A team that stores only a numeric limit loses the context needed to interpret it.

Evidence should record the quota scope, unit, measurement window, current allocation, deployment capacity, and source timestamp.

Applications must assume that throttling can occur. A robust client applies bounded retries with jitter, honors retry guidance, limits concurrent work, and routes noninteractive workloads to queues or batch services. An agent runtime should not allow every tool invocation to retry independently without a shared request budget; that behavior can amplify a capacity incident into a cost incident.

17.9 Model Versions, Upgrades, and Retirement

A model name is not a sufficient release identifier. Providers revise model families, publish snapshots, change default versions, move deployments to newer releases, and retire older models. Even when API compatibility is preserved, response style, reasoning behavior, safety sensitivity, token usage, latency, and tool-call structure may change.

Model Deployment Lifecycle

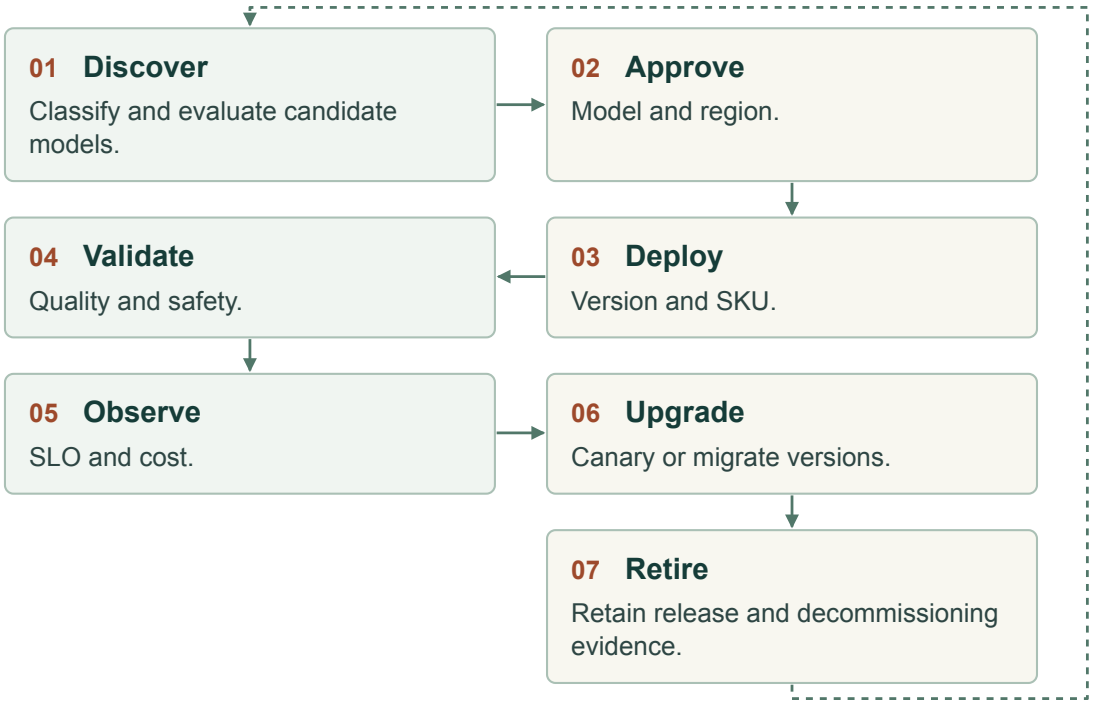


Figure 17-4 The deployment lifecycle makes model change an explicit release process.

Table 17-9 Evidence required for a model upgrade

Stage	Required evidence
Discover	Provider notice, candidate model/version, availability by region and deployment type
Approve	Use-case fit, data boundary, license or provider terms, risk classification, owner
Deploy candidate	Separate deployment name or controlled upgrade option, capacity and policy binding
Validate	Golden-set quality, safety, tool behavior, latency, token and cost comparison
Canary	Traffic percentage or selected tenants, rollback threshold, observability dimensions
Promote	Decision record linking test and production evidence
Retire	Caller migration, dependency inventory, final cost and audit export, deletion confirmation

The stable deployment name is an abstraction, not a license for silent mutation. If the platform supports automatic version upgrades, the organization should decide where that behavior is acceptable. Low-risk experimentation may accept automatic movement. Regulated or tool-using production agents may require a pinned version and an explicit migration window. In either case, the current effective version must be observable.

Retirement must be treated as part of architecture. A deployment inventory should answer which applications and agents still use a retiring version. Compatibility tests should run before the deadline. The replacement should be deployed and evaluated before the old deployment is deleted. Evidence should survive deletion long enough to support audit and incident review.

17.10 Safety Policies and Guardrails as Resources

Safety behavior should not be an invisible provider default. A production system needs an explicit policy that can be named, versioned, reviewed, assigned, tested, and changed independently from application code. Microsoft Foundry guardrails, for example, organize multiple controls into a named collection that can evaluate model input and output and, for supported agent scenarios, tool calls and tool responses.

Table 17-10 Guardrail control families

Control family	Purpose	Engineering evidence
Content categories	Detect or block harmful categories according to policy thresholds	Policy name, threshold, block result, false-positive review
Prompt-attack detection	Identify attempts to override system intent or manipulate instructions	Detection outcome, source context, response action
Indirect attack detection	Inspect retrieved or tool-provided content for malicious instructions	Document/tool source, control result, isolation action
PII or sensitive-data controls	Detect, redact, block, or route sensitive content	Data class, policy decision, retention rule
Task adherence / tool safety	Check whether planned or returned tool behavior remains within task and policy	Tool name, requested action, policy verdict, approval
Custom business rules	Apply domain-specific classifiers, allowlists, or postconditions	Rule version, owner, test corpus, exception process

Guardrails are not a substitute for application authorization. A content filter cannot decide whether a principal is allowed to shut down a virtual machine. A prompt-attack detector cannot replace a tool allowlist. Safety controls operate alongside identity, data authorization, network boundaries, schema validation, approval, and audit.

Guardrail changes should be tested like model changes. A stricter policy may improve risk posture while causing unacceptable refusal rates. A looser policy may improve completion rate while exposing sensitive content. The deployment record should therefore bind the model version and guardrail version that produced each evaluation result.

17.11 Observability and Inference SLOs

Managed inference can still be operationally opaque if the team collects only application exceptions. The service must expose a request-level or aggregate evidence model that connects workload identity, deployment, model version, latency, token usage, throttling, safety outcomes, and billing units.

Governed Inference Request Path

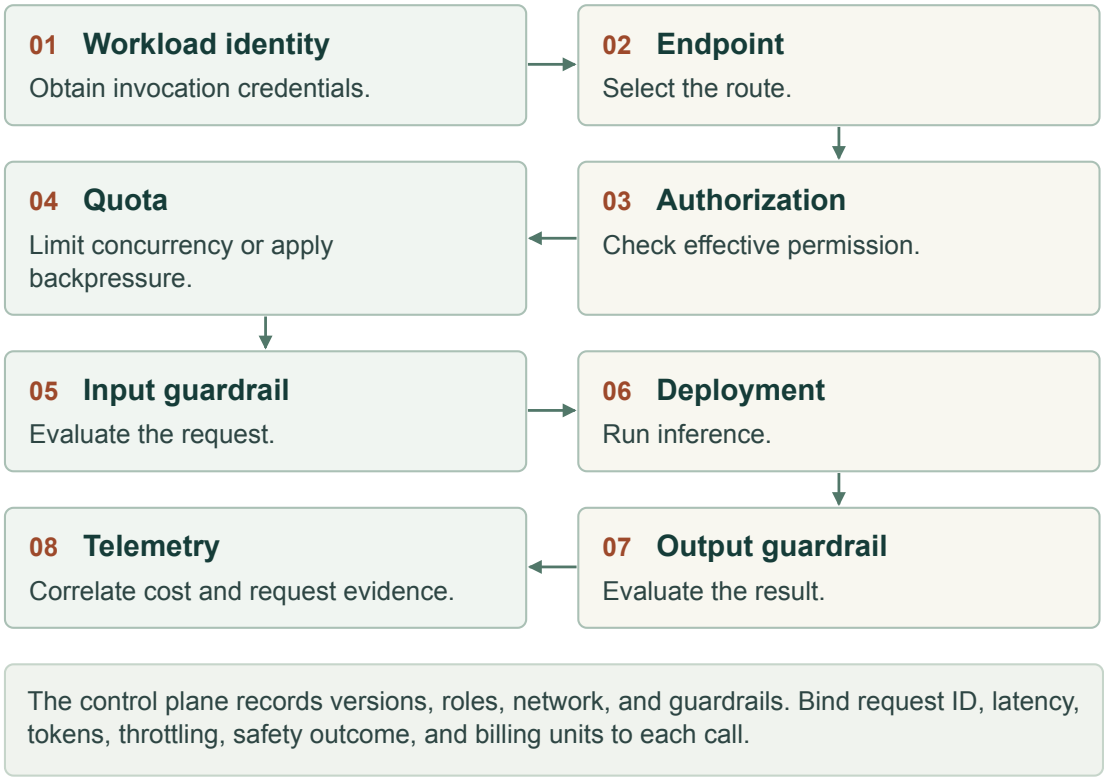


Figure 17-5 A governed request path produces both control-plane state and data-plane evidence.

Table 17-11 Minimum inference signals

Signal	Why it matters	Useful dimensions
Request count and success rate	Separates demand from reliability and detects silent abandonment	Deployment, project, workload, region, response class
End-to-end latency	Shows user-visible delay	Deployment, operation, stream/non-stream, client region
Time to first token	Separates model startup/queue delay from generation duration	Deployment type, model version, request class
Input and output tokens	Explains workload shape and cost	Model, deployment, tenant/project, feature
Throttled requests	Shows capacity pressure and client retry demand	Quota scope, deployment, caller, minute/hour window
4xx and 5xx responses	Distinguishes caller, policy, and service failures	Status code, operation, deployment, principal

Signal	Why it matters	Useful dimensions
Safety blocks and transformations	Measures guardrail impact and possible attacks	Policy version, category, input/output/tool stage
Version and configuration change events	Explains behavior shifts that request metrics alone cannot	Actor, resource ID, old/new value, correlation ID

Table 17-12 Example inference SLOs

Objective	Example indicator	Decision enabled
Interactive responsiveness	99% of successful streaming requests produce a first token within the agreed threshold	Scale, reroute, degrade, or pause rollout
Request reliability	Successful nonpolicy responses / eligible requests over 30 days	Error-budget based release decisions
Capacity resilience	Throttled eligible requests below threshold with bounded queue age	Increase capacity or change workload scheduling
Safety availability	Policy false-positive rate on approved evaluation set below threshold	Tune policy or route to review
Cost efficiency	Effective cost per accepted business outcome within target range	Change model, prompt, caching, batch, or capacity mode

Raw prompts and responses should not be logged by default merely because they are useful during debugging. They may contain personal data, trade secrets, credentials, retrieved documents, or tool output. Prefer request IDs, hashes, structured attributes, sampled redacted content, and controlled diagnostic capture. Retention and access must be explicit.

The control plane contributes change events that the data plane cannot generate by itself. Role assignment changes, deployment updates, network changes, guardrail updates, quota changes, and deletions should be correlated with request telemetry. Without this correlation, operators can observe a regression but cannot establish what changed.

17.12 Billing Semantics and Unit Economics

A model deployment has a billing contract, not one universal price. Shared online inference may charge separately for input and output tokens. Some services distinguish cached input, images, audio, or tool-related operations. Provisioned throughput charges for committed capacity. Dedicated compute charges for running infrastructure. Batch may use a different unit price and completion model. Fine-tuning adds training and hosted-model costs.

Table 17-13 Cost components in managed inference

Component	Typical billing unit	Design lever
Input processing	Input tokens, characters, images, audio duration, or requests	Prompt size, retrieval precision, compression, caching
Output generation	Output tokens, images, audio, or generated media	Output limits, response format, model choice, post-processing
Reserved capacity	Provisioned throughput units or capacity hours	Commitment sizing, scheduling, utilization
Dedicated runtime	Accelerator/CPU/memory hours and attached storage	Autoscaling, warm pools, model size, runtime efficiency
Batch processing	Batch tokens, jobs, files, or completed work units	Move noninteractive work from online path
Fine-tuning	Training tokens/compute plus model hosting	Data quality, experiment count, adapter strategy
Platform dependencies	Search, storage, networking, telemetry, safety, evaluation	Architecture, retention, sampling, data locality

Token price is therefore not unit economics. A cheaper model that requires more retries, longer prompts, larger retrieval contexts, or human correction may cost more per accepted business outcome. A more expensive model can be cheaper if it completes a workflow in one step and avoids tool failures. The meaningful denominator is the outcome: accepted answer, resolved ticket, reviewed document, successful tool action, or completed agent task.

The model-resource contract should record the billing unit and the allocation dimensions available in provider cost data. A FinOps system can then connect usage to project, application, feature, or tenant. Cost evidence often arrives later than request telemetry, so reconciliation should preserve request IDs or aggregate dimensions that allow the two timelines to be joined.

17.13 Azure Reference Control Plane: Microsoft Foundry

Microsoft Foundry provides a coherent reference for the resource-level view. At the cloud management layer, a Foundry resource is represented through the Microsoft.CognitiveServices resource provider with an AI Services account kind. It governs security, networking, model deployments, quota, and billing. Projects are child resources

that organize development, connections, agents, and evaluation while inheriting or using the parent resource boundary. Model deployments are governed child resources of the account.

Table 17-14 Azure Foundry reference objects

Azure object	Control-plane responsibility	Data-plane consequence
Foundry resource / AI Services account	Top-level identity, network, billing, quota, and deployment boundary	Supplies the unified resource endpoint and authorization context
Foundry project	Development and collaboration boundary for applications, agents, connections, and evaluation	Scopes project assets and user experience; does not automatically create dedicated model capacity
Model deployment	Binds deployment name, model format/name/version, SKU/capacity, upgrade behavior, and optional responsible-AI policy	The deployment name is supplied in inference requests
Private endpoint and DNS	Restricts inbound network path and name resolution	The same logical hostname resolves privately from the virtual network
Azure RBAC assignment	Grants control-plane and/or data-plane permissions at resource or project scopes	Determines who can deploy, invoke, operate, or consume
Azure Monitor and diagnostics	Routes platform metrics, logs, and change evidence	Enables SLOs, incidents, capacity and cost analysis

The deployment resource exposes fields such as model format, name, version, SKU, capacity, responsible-AI policy binding, and version-upgrade behavior. Exact API versions and available properties evolve, so infrastructure code should be validated against the current resource template reference and the selected region. Model availability is also regional and deployment-type specific.

Foundry RBAC distinguishes management and runtime responsibilities. Microsoft recommends token-based Microsoft Entra authentication rather than broadly shared keys for production. Keys remain supported for compatibility and initial use, but possession of a key does not provide the same principal-level attribution or role restriction as an Entra identity.

Azure deployment types make the operating contract explicit. Global, data-zone, regional, standard, provisioned, batch, and developer or dedicated offerings differ in processing scope, capacity, pricing, and workload suitability. These names are provider-specific; the transferable lesson is to treat deployment type as architecture.

17.14 Multicloud Translation: Engineering Intent, Not Feature Parity

Alibaba Cloud Model Studio, Tencent Cloud TokenHub and TI-ONE, and Volcengine Ark expose many of the same engineering concerns: model catalogs, regional endpoints, API credentials or IAM, online and batch inference, model deployment, quota, monitoring, usage statistics, evaluation, fine-tuning, and model retirement. The objects and maturity differ. A comparison should therefore map engineering intent rather than assume one-to-one product equivalence.

Table 17-15 Model-resource intent across four cloud platforms

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Top-level AI platform boundary	Microsoft Foundry resource and project	Model Studio account/workspace and regional service boundary	TokenHub and TI-ONE service/project boundaries	Ark project and platform resources
Managed model access	Foundry Models deployments and unified inference endpoint	Model Studio model services with regional and workspace endpoints	TokenHub model gallery and online inference; TI-ONE inference services	Ark online inference and endpoint modes
Identity and authorization	Microsoft Entra ID, Azure RBAC, managed identity	Alibaba Cloud RAM and workspace-scoped credentials	Tencent Cloud CAM and API-key access controls	Volcengine IAM and API keys
Capacity and quota	Subscription/account/model/deployment-type quotas and provisioned options	Regional/workspace concurrency, quotas, online and batch contracts	Service quota, inference-service configuration, usage and alarms	Project quota, endpoint configuration, online/batch capacity options
Monitoring and usage	Azure Monitor metrics, diagnostics, activity and cost data	Workspace usage, service monitoring, error and billing evidence	Usage statistics, model/inference monitoring, alarms and deployment logs	Project usage statistics, inference endpoint monitoring and control-plane usage APIs
Lifecycle	Model/version deployment, upgrade option, deprecation and retirement	Model availability, fine-tuned model deployment, regional endpoint lifecycle	Model/inference service lifecycle and deployment operations	Model deployment, endpoint modes, published retirement notices

17.14.1 Alibaba Cloud Model Studio

Model Studio presents a one-stop model service with OpenAI-compatible interfaces and provider-native protocols. The regional boundary is operationally important: regions have distinct service domains, API keys, available models, and capabilities. For production,

workspace-dedicated domains can provide a clearer workspace boundary and documented concurrency, network, and service characteristics. Fine-tuned models, batch jobs, knowledge resources, and agent applications extend the platform beyond raw inference.

17.14.2 Tencent Cloud TokenHub and TI-ONE

Tencent Cloud separates convenient model access from broader model engineering. TokenHub exposes a model gallery, pay-as-you-go online inference, API keys whose accessible range can be limited to selected models or inference services, usage statistics, and monitoring. TI-ONE adds model development and inference-service management, including deployment logs and monitoring. The combination illustrates why model access and AI PaaS lifecycle are related but distinct layers.

17.14.3 Volcengine Ark

Volcengine Ark combines model inference, evaluation, fine-tuning, and application capabilities. Its documentation distinguishes online inference modes, batch processing, low-latency options, project quotas, API keys and IAM, usage statistics, endpoint security monitoring, and model retirement. These objects support the same control-plane questions even where product terminology differs from Foundry.

NOTE: Cloud product names, model catalogs, quotas, prices, preview status, and regional availability are volatile. The printed chapter should retain the stable resource concepts and decision questions. Current commands, model names, and region matrices belong in the companion repository and must be revalidated before manuscript freeze.

17.15 The Model Deployment Decision Record

A short decision record turns the model-resource contract into an operational artifact. It should be committed beside infrastructure code or stored in a governed architecture repository. The record does not replace provider state; it explains why that state exists and which evidence authorizes change.

Example 17-1. Provider-aware model deployment decision record

```
apiVersion: cloudaipatterns.dev/v1
kind: ModelDeploymentRecord
metadata:
name: claims-assistant-primary
owner: claims-platform
environment: production
spec:
provider: azure
resourceId: /subscriptions/.../accounts/foundry-prod/deployments/claims-primary
project: claims-agent-prod
model:
name: <approved-model-family>
version: <approved-version>
format: OpenAI
deployment:
name: claims-primary
type: <deployment-type>
processingScope: <regional|data-zone|global>
upgradePolicy: explicit
access:
inferencePrincipal: <managed-identity-resource-id>
publicNetworkAccess: disabled
privateEndpoint: <resource-id>
safety:
guardrail: claims-prod-v3
objectives:
timeToFirstTokenP99: <threshold>
throttleRateMax: <threshold>
costPerAcceptedCaseMax: <threshold>
lifecycle:
evaluationSet: claims-golden-v5
rollbackDeployment: claims-previous
retirementOwner: claims-platform
```

The record should be generated or verified from live state where possible. Manually copied version strings and resource IDs drift. A platform pipeline can query the deployment, compare it with the approved record, run evaluation, and block promotion when model version, policy, region, or identity differs unexpectedly.

Capability Check C4: Inspect and Invoke a Managed Model Resource

This independent capability check verifies one managed model deployment as a cloud resource. It deliberately avoids building a complete application. You may use an existing Microsoft Foundry resource and model deployment, or create a temporary one with the current portal, CLI, Bicep, or Terraform workflow supported in your region. Because model availability and quota change, the exercise uses parameters rather than prescribing a particular commercial model.

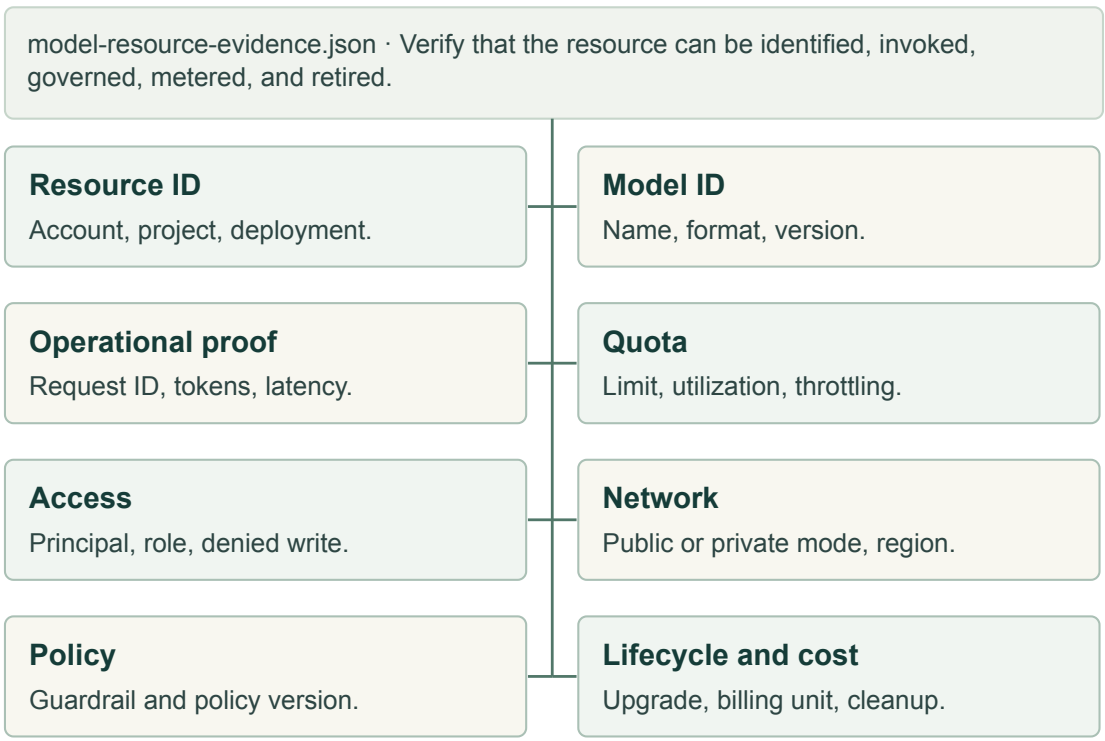


Figure 17-6 The capability check produces a model-resource evidence packet rather than only a screenshot of a successful prompt.

C4.1 Capability goal

Prove that one deployment can be identified, invoked with a workload principal, inspected for model and capacity state, connected to quota and telemetry, evaluated for network and safety posture, and retired or left with an explicit owner. The primary deliverable is model-resource-evidence.json.

C4.2 Prerequisites and cost boundary

- An Azure subscription in which you may read a Foundry resource and model deployment. Creating resources requires additional control-plane permissions.
- Azure CLI and Python 3.11 or later. Install azure-identity and the current openai package for the invocation example.
- A Microsoft Entra principal with permission to invoke the selected deployment. Use a user identity for an instructional run or a managed/workload identity for an automated run.
- A model deployment supported in the chosen region and quota pool. Do not copy a model name from this book without checking current availability.
- Cost class: low for a few short inference requests on a consumption deployment; potentially moderate or high for provisioned or dedicated capacity. Do not create committed capacity for this capability check.

C4.3 Set environment variables

Example 17-2. Capability-check environment

```
export AZ_SUBSCRIPTION_ID="<subscription-id>"
export AZ_RESOURCE_GROUP="<resource-group>"
export AZ_AI_ACCOUNT="<foundry-resource-name>"
export AZ_DEPLOYMENT="<model-deployment-name>"

az account set --subscription "$AZ_SUBSCRIPTION_ID"
```

C4.4 Inspect account and deployment state

Query the live control plane before invoking the model. The deployment name used by the API should resolve to an inspectable child resource with explicit model and SKU properties.

Example 17-3. Inspect the model deployment

```
az cognitiveservices account show \
--resource-group "$AZ_RESOURCE_GROUP" \
--name "$AZ_AI_ACCOUNT" \
--output json > account.json

az cognitiveservices account deployment show \
--resource-group "$AZ_RESOURCE_GROUP" \
--name "$AZ_AI_ACCOUNT" \
--deployment-name "$AZ_DEPLOYMENT" \
--output json > deployment.json

jq '{
deploymentName: .name,
model: .properties.model,
sku: .sku,
raiPolicyName: .properties.raiPolicyName,
versionUpgradeOption: .properties.versionUpgradeOption
}' deployment.json
```

The exact property set depends on the API version and deployment type. Preserve the unmodified JSON as evidence. The summarized view is for human review; it should not replace the provider response.

C4.5 Record identity and authorization

Identify the principal that will invoke inference and the principals permitted to manage the account or deployment. Do not infer runtime permission from portal visibility. Record role assignments at the account and project scopes, and test the intended data-plane call.

Example 17-4. Capture role assignments at the AI resource scope

```
ACCOUNT_ID=$(jq -r '.id' account.json)

az role assignment list \
--scope "$ACCOUNT_ID" \
--include-inherited \
--all \
--output json > role-assignments.json
```

C4.6 Invoke through Microsoft Entra ID

The following Python example uses a bearer-token provider rather than a long-lived API key. The base URL and token scope reflect the current Foundry/OpenAI v1 pattern; verify them against current official documentation for the selected resource.

Example 17-5. Keyless inference invocation

```
import json
import os
import time
from pathlib import Path
from azure.identity import DefaultAzureCredential, get_bearer_token_provider
from openai import OpenAI

account = os.environ["AZ_AI_ACCOUNT"]
deployment = os.environ["AZ_DEPLOYMENT"]

token_provider = get_bearer_token_provider(
DefaultAzureCredential(),
"https://ai.azure.com/.default",
)

client = OpenAI(
base_url=f"https://{account}.openai.azure.com/openai/v1/",
api_key=token_provider,
)

started = time.perf_counter()
response = client.responses.create(
model=deployment,
input="Return exactly: governed model resource verified",
max_output_tokens=32,
)
elapsed_ms = round((time.perf_counter() - started) * 1000, 1)

evidence = {
"responseId": response.id,
"deployment": deployment,
"elapsedMs": elapsed_ms,
"outputText": response.output_text,
"usage": response.usage.model_dump() if response.usage else None,
}
Path("inference-evidence.json").write_text(
```

```
json.dumps(evidence, indent=2)
)
print(json.dumps(evidence, indent=2))
```

A successful response proves runtime authorization and endpoint reachability. It does not prove least privilege. Where the instructional environment permits, use the same principal to attempt a harmless denied management operation, such as creating a new deployment or changing public network access. Preserve the denial result without granting the missing privilege.

C4.7 Inspect network, quota, telemetry, and safety evidence

- Network: record publicNetworkAccess, private endpoint connections, DNS assumptions, and the caller network path. A public endpoint protected by identity is not equivalent to private ingress.
- Quota: record the provider quota scope, model/version or deployment type, allocated amount, current usage, and timestamp. A portal screenshot alone is weak evidence; use an API or exported record where available.
- Telemetry: list metric definitions for the AI resource and capture at least request count, token usage, latency, failure, and throttling signals supported by the deployment.
- Safety: record the guardrail or responsible-AI policy bound to the deployment, including whether the observed value is an explicit policy or a platform default.
- Cost: record the billing unit and the cost-allocation dimensions available for the resource. Cost data may not be available immediately after the test request.

Example 17-6. Capture metric definitions and network posture

```
az monitor metrics list-definitions \
--resource "$ACCOUNT_ID" \
--output json > metric-definitions.json

# Inspect the live account state used for network evidence.
jq '{
id: .id,
location: .location,
kind: .kind,
publicNetworkAccess: .properties.publicNetworkAccess,
privateEndpointConnections: .properties.privateEndpointConnections
}' account.json > network-evidence.json
```

C4.8 Assemble the evidence packet

Example 17-7. Assemble a provider-aware evidence packet

```
import json
from pathlib import Path

def load(path):
return json.loads(Path(path).read_text())

account = load("account.json")
deployment = load("deployment.json")
roles = load("role-assignments.json")
network = load("network-evidence.json")

packet = {
"schema": "cloudaipatterns.dev/model-resource-evidence/v1",
"capturedAt": "<UTC timestamp>",
"resource": {
"id": account["id"],
"location": account["location"],
"kind": account.get("kind"),
},
"deployment": {
"name": deployment["name"],
"model": deployment.get("properties", {}).get("model"),
"sku": deployment.get("sku"),
"guardrail": deployment.get("properties", {}).get("raiPolicyName"),
"upgradeOption": deployment.get("properties", {}).get("versionUpgradeOption"),
},
"authorization": {"roleAssignments": roles},
"network": network,
"runtimeEvidence": load("inference-evidence.json"),
"quotaEvidence": "<attach current provider export or API result>",
"telemetryEvidence": "metric-definitions.json",
"billingUnit": "<record current deployment billing contract>",
"retirementOwner": "<team or named role>",
}
```

```
Path("model-resource-evidence.json").write_text(
json.dumps(packet, indent=2)
)
```

C4.9 Verification criteria

Table 17-16 Capability-check acceptance criteria

Criterion	Pass evidence
Resource identity	Provider resource ID, account, project if used, region, deployment name
Model identity	Model format, name, version and deployment type/SKU
Runtime authorization	Successful Entra-authenticated request with response/request ID
Least privilege	Documented role scope and, where safe, one expected denied management operation
Network posture	Public/private access state, private endpoint evidence where applicable, caller path
Quota/capacity	Scope, unit, allocation or deployment capacity, timestamp
Safety binding	Named guardrail/responsible-AI policy or documented platform default
Telemetry	Supported metric definitions and one observed request/usage signal after propagation
Billing	Billing unit and cost allocation scope; note expected reporting delay
Lifecycle	Upgrade behavior, owner for deprecation, and cleanup or retention decision

C4.10 Cleanup and retention

If you created a temporary deployment, delete the deployment and verify that it no longer appears in the deployment list. If you created a dedicated resource group only for this capability check, delete the group after exporting the evidence packet. Do not delete a shared account merely because the exercise is complete. Retain the evidence packet without raw prompts, secrets, or access tokens.

Example 17-8. Delete only the temporary deployment and verify retirement

```
az cognitiveservices account deployment delete \
--resource-group "$AZ_RESOURCE_GROUP" \
--name "$AZ_AI_ACCOUNT" \
--deployment-name "$AZ_DEPLOYMENT"

az cognitiveservices account deployment list \
--resource-group "$AZ_RESOURCE_GROUP" \
--name "$AZ_AI_ACCOUNT" \
--output table
```

17.16 CloudDesk Reuse: A Safe Model Capability Layer

CloudDesk FinOps Agent may later consume model services, but this book does not implement its runtime or tenant system. The reusable output of this chapter is a capability contract that allows an agent or platform service to inspect and invoke a model without treating the endpoint as an opaque string.

Table 17-17 CloudDesk-oriented capability contracts

Capability	Purpose	Required controls
llm.deployment.inspect	Return model, version, deployment type, region, guardrail, capacity, upgrade, and retirement metadata	Read-only cloud identity; no prompt or response access
llm.inference.invoke	Invoke an approved deployment under a workload identity and bounded request budget	Deployment allowlist, token limits, correlation ID, safety and telemetry binding
llm.quota.read	Read current quota scope, allocation, utilization, and throttling evidence	Read-only quota access; timestamp and provider scope
llm.retirement.assess	Find deployments affected by deprecation and propose a migration plan	Inventory, provider retirement feed, dependency graph, evaluation gate; no automatic destructive action

These capabilities should return structured evidence, not prose alone. llm.deployment.inspect should never reveal API keys. llm.inference.invoke should require an approved deployment name rather than accepting an arbitrary endpoint supplied by a prompt. llm.retirement.assess may recommend replacement but should not delete a deployment without change approval and dependency verification.

17.17 Key Takeaways

- A model API is a data-plane protocol. A model resource adds the control-plane state needed for ownership, identity, isolation, quota, safety, telemetry, billing, change, and retirement.
- Model family, model version, deployment, endpoint, project, and AI PaaS resource are distinct objects and should remain distinct in architecture and evidence.
- MaaS is an essential managed-inference layer, but AI PaaS also governs projects, knowledge, evaluation, agents, tools, identity, observability, and lifecycle.
- Deployment type is architecture: shared, regional, data-zone, provisioned, batch, dedicated, and fine-tuned modes encode different capacity, latency, locality, and cost contracts.
- Production workloads should prefer workload identity and scoped authorization over broadly shared long-lived API keys.
- Quota is not throughput, throughput is not an SLO, and a budget is not a hard capacity control. Each needs its own evidence and response mechanism.
- Model upgrades and retirement are release events. Stable deployment names help callers, but effective model and policy versions must remain visible.
- Guardrails are versioned controls that complement - not replace - identity, authorization, network, validation, and human approval.
- Inference operations require correlated request, token, latency, throttle, safety, change, and cost evidence, with privacy-aware content capture.
- Multicloud comparison is strongest when it maps engineering intent instead of forcing superficial feature equivalence.

References

[1] National Institute of Standards and Technology. The NIST Definition of Cloud Computing (SP 800-145). https://doi.org/10.6028/NIST.SP.800-145 Accessed August 24, 2026.

[2] National Institute of Standards and Technology. Zero Trust Architecture (SP 800-207). https://doi.org/10.6028/NIST.SP.800-207 Accessed August 24, 2026.

[3] National Institute of Standards and Technology. AI Risk Management Framework (AI RMF 1.0). https://doi.org/10.6028/NIST.AI.100-1 Accessed August 24, 2026.

[4] Microsoft Learn. Architecture of Microsoft Foundry. https://learn.microsoft.com/en-us/azure/foundry/concepts/architecture Accessed August 24, 2026.

[5] Microsoft Learn. Role-based access control for Microsoft Foundry. https://learn.microsoft.com/en-us/azure/foundry/concepts/rbac-foundry Accessed August 24, 2026.

[6] Microsoft Learn. Deployment types for Microsoft Foundry Models. https://learn.microsoft.com/en-us/azure/foundry/foundry-models/concepts/deployment-types Accessed August 24, 2026.

[7] Microsoft Learn. Azure OpenAI in Microsoft Foundry Models quotas and limits. https://learn.microsoft.com/en-us/azure/foundry/openai/quotas-limits Accessed August 24, 2026.

[8] Microsoft Learn. Configure a private link for Microsoft Foundry. https://learn.microsoft.com/en-us/azure/foundry/how-to/configure-private-link Accessed August 24, 2026.

[9] Microsoft Learn. Guardrails overview in Microsoft Foundry. https://learn.microsoft.com/en-us/azure/foundry/guardrails/guardrails-overview Accessed August 24, 2026.

[10] Microsoft Learn. Microsoft.CognitiveServices accounts/deployments resource reference. https://learn.microsoft.com/en-us/azure/templates/microsoft.cognitiveservices/accounts/deployments Accessed August 24, 2026.

[11] Microsoft Learn. Create a Microsoft Foundry resource and project with a Bicep template. https://learn.microsoft.com/en-us/azure/foundry/how-to/create-resource-template Accessed August 24, 2026.

[12] Microsoft Learn. Azure CLI cognitive services account deployment commands. https://learn.microsoft.com/en-us/cli/azure/cognitiveservices/account/deployment Accessed August 24, 2026.

[13] Microsoft Learn. Work with chat models and the v1 API. https://learn.microsoft.com/en-us/azure/foundry/openai/how-to/chatgpt Accessed August 24, 2026.

[14] Microsoft Learn. Supported metrics for Microsoft.CognitiveServices accounts. https://learn.microsoft.com/en-us/azure/azure-monitor/reference/supported-metrics/microsoft-cognitiveservices-accounts-metrics Accessed August 24, 2026.

[15] Microsoft Learn. Azure Monitor documentation. https://learn.microsoft.com/en-us/azure/azure-monitor/ Accessed August 24, 2026.

[16] Alibaba Cloud Documentation. What is Model Studio?. https://www.alibabacloud.com/help/en/model-studio/what-is-model-studio Accessed August 24, 2026.

[17] Alibaba Cloud Documentation. Model Studio regions and endpoints. https://www.alibabacloud.com/help/en/model-studio/regions/ Accessed August 24, 2026.

[18] Alibaba Cloud Documentation. Batch inference in Model Studio. https://www.alibabacloud.com/help/en/model-studio/batch-inference Accessed August 24, 2026.

[19] Alibaba Cloud Documentation. Model Studio error codes. https://www.alibabacloud.com/help/en/model-studio/error-code Accessed August 24, 2026.

[20] Tencent Cloud Documentation. TokenHub Quick Start. https://www.tencentcloud.com/document/product/1300/78939 Accessed August 24, 2026.

[21] Tencent Cloud Documentation. TokenHub API Key Management. https://www.tencentcloud.com/document/product/1300/78949 Accessed August 24, 2026.

[22] Tencent Cloud Documentation. TI-ONE API category. https://www.tencentcloud.com/document/product/851 Accessed August 24, 2026.

[23] Volcengine Documentation. Volcengine Ark documentation. https://docs.volcengine.com/docs/82379 Accessed August 24, 2026.

[24] Volcengine Documentation. Ark inference modes. https://www.volcengine.com/docs/82379/2123245 Accessed August 24, 2026.

[25] Volcengine Documentation. Ark low-latency online inference. https://www.volcengine.com/docs/82379/2335857 Accessed August 24, 2026.

[26] OpenTelemetry. OpenTelemetry semantic conventions for generative AI systems. https://opentelemetry.io/docs/specs/semconv/gen-ai/ Accessed August 24, 2026.

[27] OWASP Foundation. OWASP Top 10 for Large Language Model Applications. https://owasp.org/www-project-top-10-for-large-language-model-applications/ Accessed August 24, 2026.

[28] Google. SRE Workbook: Implementing SLOs. https://sre.google/workbook/implementing-slos/ Accessed August 24, 2026.

[29] FinOps Foundation. FinOps Framework. https://www.finops.org/framework/ Accessed August 24, 2026.

[30] FinOps Foundation. FOCUS Specification. https://focus.finops.org/ Accessed August 24, 2026.

Part V / Chapter 18

RAG Is Not a Vector Database

Governing Knowledge as a Cloud Resource

A production retrieval system is not defined by where vectors are stored. It is defined by how knowledge is sourced, transformed, authorized, versioned, retrieved, evidenced, evaluated, observed, and retired.

In This Chapter

- Why a vector index is necessary but insufficient for production RAG
- How to model knowledge as a governed cloud resource with an explicit lifecycle
- How Azure AI Search and Foundry IQ separate search-management objects from agent-facing knowledge objects
- When to use lexical, vector, hybrid, semantic, filtered, and agentic retrieval
- How identity, networking, authorization, versioning, evidence, evaluation, observability, and cost fit the control plane
- How to create, evaluate, version, promote, roll back, and clean up an independent governed knowledge base
- How the same engineering intent maps to Alibaba Cloud Model Studio, Tencent Cloud ADP, and Volcengine AgentKit and Ark

VERSION NOTE

Product names, API maturity, and availability in this chapter were checked against official documentation on August 24, 2026. The stable path in the Azure lab uses REST API version 2026-04-01. Capabilities that remain in preview are marked explicitly and are not required for the core lab.

18.1 The Four-Line Demo That Works Until It Matters

A retrieval-augmented generation prototype can be deceptively small. Take a set of documents, divide them into chunks, convert each chunk into an embedding, place the vectors in an index, retrieve the nearest neighbors for a question, and append those neighbors to a prompt. The original RAG architecture established the power of combining a parametric language model with an external, non-parametric memory. That basic pattern remains sound. What changes in production is the number of obligations attached to the word knowledge. [1]

The first successful answer is therefore a dangerous milestone. It proves that semantic retrieval can work on one data set under one configuration. It does not prove that the knowledge is current, authorized, traceable, reproducible, cost-controlled, or safe to replace. A demo answers: Can the model find a useful passage? A production system must also answer: Who owns the source? Which transformation produced this chunk? Which identities may retrieve it? What changed since yesterday? Which index version generated the answer? Can the result be reproduced? Can the release be rolled back? What did the retrieval cost? Which evidence can an auditor inspect?

The attractive prototype

```
documents = load_files("./handbook")
chunks = split(documents)
vectors = embed(chunks)
index.upsert(vectors)

context = index.nearest(embed(user_question), k=5)
answer = llm.generate(user_question, context=context)
```

Nothing in this code is inherently wrong. The problem is that every omitted concern will reappear somewhere else: in an ad hoc synchronization script, a hard-coded filter, an application cache, a spreadsheet of source owners, a manual reindexing procedure, a security review, or an incident report. A platform can hide these obligations, but it cannot abolish them.

A vector database is a data structure and query service optimized for similarity search. A governed knowledge resource is a managed contract that connects source content, transformation, retrieval behavior, authorization, evidence, quality, operations, and lifecycle. Vector search is one mechanism inside that contract. Treating it as the whole system is comparable to calling a production database application a B-tree.

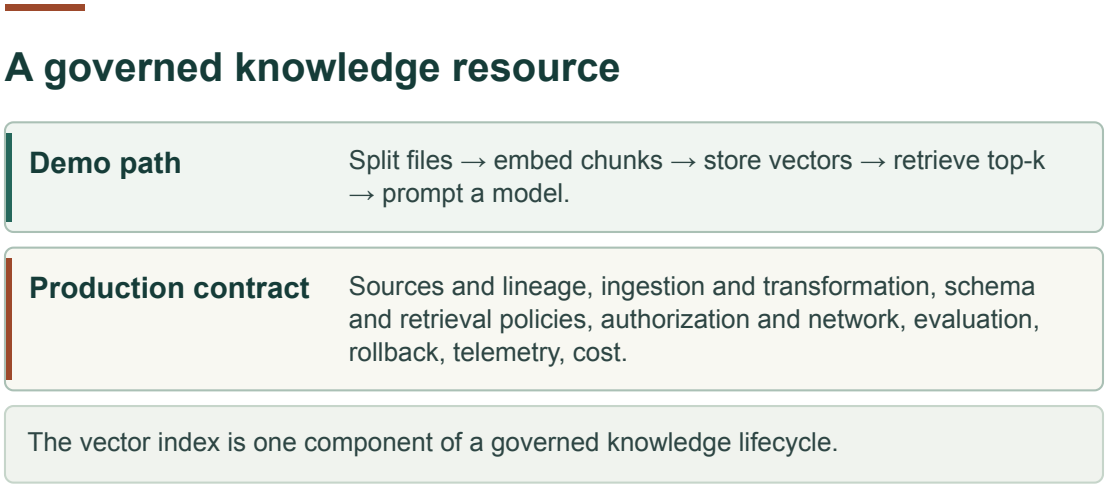


Figure 18-1 A vector index is one component inside a governed knowledge lifecycle.

CONTROL-PLANE RULE

If a knowledge change cannot be authorized, versioned, observed, evaluated, promoted, and reversed through an explicit management path, the system does not yet have a production knowledge control plane.

18.1.1 The production gap

The production gap is easiest to see through failure modes. Suppose a policy document is replaced in object storage, but the indexer has not run. The answer is syntactically fluent and operationally stale. Suppose the index contains tenant or department metadata, but the application forgets to apply the filter on one endpoint. The answer is semantically relevant and unauthorized. Suppose an embedding deployment changes dimensions without a coordinated index migration. The ingestion pipeline fails or silently targets the wrong index. Suppose a new chunking policy improves average relevance but removes the passage needed for one regulated procedure. Without a regression set and rollback path, the team discovers the loss from a user complaint.

These are not vector-distance problems. They are lifecycle problems. They are also cloud control-plane problems because they require named resources, identities, policies, network boundaries, deployment records, metrics, logs, quotas, budgets, and state transitions. The cloud value is not merely hosted storage. It is the ability to make knowledge operations explicit, automatable, and governable.

18.2 From Vector Index to Knowledge Resource

For this book, a knowledge resource is defined by nine coupled elements:

K = <S, T, X, R, A, V, E, O, C>

- S - sources and provenance: what content is admitted, who owns it, and where it came from
- T - transformations: parsing, normalization, chunking, enrichment, embedding, and metadata extraction
- X - indexes and schemas: the searchable contract, including text, vectors, source fields, and security metadata
- R - retrieval policy: lexical, vector, hybrid, filters, reranking, query rewriting, and runtime limits
- A - authorization and access boundaries: identities, roles, document-level controls, network paths, and secrets
- V - versioning and release state: candidate, current, retired, rollback target, and compatibility expectations
- E - evidence and evaluation: chunks, references, activity, test sets, quality thresholds, and failure analysis
- O - operations: freshness, latency, availability, ingestion health, incident response, and cleanup
- C - cost and capacity: storage, ingestion, embedding, retrieval, reranking, model use, quotas, and budgets

The tuple is not intended as a new API. It is a completeness test. A team may implement the elements with a managed search service, a cloud AI platform, open-source components, or a combination. The engineering question is whether the elements have explicit owners and interfaces. If the vector index is managed while source admission, access filters, evaluation, and rollback remain implicit, the knowledge system is only partially managed.

Concern	Vector database view	Knowledge-resource view
Primary object	Collection or index	Versioned knowledge product with source, retrieval, access, and evidence contracts
Input	Vectors and metadata	Authorized sources plus governed transformations
Query	Nearest-neighbor search	Composed retrieval policy: lexical, vector, filters, reranking, limits, and citations
Security	Credentials and optional metadata filter	Identity, network boundary, server-injected authorization, source permissions, and audit

Concern	Vector database view	Knowledge-resource view
Change	Upsert or rebuild	Candidate build, evaluation, promotion, compatibility check, rollback, and retirement
Quality	Similarity score	Retrieval coverage, ranking, grounding, citation validity, security exclusions, latency, and cost
Operations	Capacity and query metrics	Ingestion health, freshness, retrieval traces, SLOs, incident response, and lineage

18.2.1 The resource-level test

A capability becomes resource-level when it can be addressed and governed independently from the application code that consumes it. The practical test is not whether a portal displays a tile. Ask whether the capability has a durable identity, a lifecycle, an authorization model, an automation interface, operational telemetry, and an accountable cost boundary.

By this test, a local array of embeddings is not a knowledge resource. A managed index may be one resource, but it is still not the complete knowledge resource. A production knowledge base becomes meaningful when sources and indexes can be registered, retrieval behavior can be configured, access conditions can be imposed, results can return references and activity, and changes can be promoted or reversed without rewriting every consuming agent.

DESIGN IMPLICATION

Applications and agents should depend on a stable knowledge contract, not on a file path, a raw vector collection name, or a vendor portal workflow.

18.3 The Knowledge Lifecycle

18.3.1 Admit sources deliberately

A source is more than a location. It carries ownership, authority, retention, sensitivity, and expected update behavior. Two PDFs with identical text can have different operational meaning if one is an approved policy and the other is an obsolete draft. A source registry should therefore record at least a stable source identifier, owner, canonical URI, content type, jurisdiction or region, sensitivity, effective date, expiration or review date, and the policy that controls admission.

This is the first place where RAG can fail silently. Teams often ingest whatever is easiest to upload, then ask the model to reconcile contradictions. A governed design does the opposite: resolve source authority before retrieval, and retain enough provenance to explain why a passage was eligible.

18.3.2 Make transformation observable

Parsing, OCR, section detection, table extraction, chunking, title generation, metadata enrichment, and embedding are transformations. Each can change the meaning or retrievability of content. A chunk is therefore a derived artifact, not the original document. Its record should identify the source version and transformation version that produced it.

Managed indexers and skillsets are valuable because they turn recurring ingestion work into named, inspectable objects. A pull-based indexer can run on a schedule, use a data-source connection, invoke enrichment skills, and write to a target index. The benefit is not that code disappears. The benefit is that schedule, status, field mapping, retry behavior, and failure evidence become part of the platform state. [7]

18.3.3 Treat schema as an application contract

The index schema determines what the application can search, filter, return, cite, and secure. A useful minimum chunk schema contains a stable key, content, title, source URI, source identifier, source version, timestamp, content hash, tenant or workspace identifier, access-control identifiers, and one or more vector fields. Fields used for filters must be declared filterable. Fields shown to users or passed to a model must be retrievable. Fields used for full-text search must be searchable.

Schema mistakes are expensive because some properties cannot be changed in place. In Azure AI Search, for example, an existing field cannot simply be changed to become filterable; the index must be rebuilt or replaced. This turns an apparently small metadata decision into a versioning concern. [5]

18.3.4 Bind authorization to retrieval

Retrieval authorization must be enforced before unauthorized content becomes model context. It is not enough to hide citations in the user interface after retrieval. Once an unauthorized chunk enters the prompt, the confidentiality boundary has already failed.

The application should derive the filter from authenticated server-side context. The user or model may supply a business query, but neither should construct the final security predicate. Metadata such as tenantId, workspaceId, departmentId, documentAcl, sensitivity,

and effectiveDate belongs in the retrieval contract. Exact filters and semantic relevance serve different purposes: a relevance score answers what is similar; a filter answers what is eligible.

18.3.5 Version, evaluate, promote, and retire

Chunking, embeddings, analyzers, semantic configuration, and source selection can all change retrieval behavior. These changes should create a candidate knowledge version. The candidate is loaded and evaluated against a stable question set, security tests, and operational thresholds. Only then is it promoted to current. The previous version remains available long enough to support rollback and incident analysis.

Retirement is also a control-plane operation. Deleting an application does not necessarily remove indexes, source connectors, model deployments, logs, or stored documents. A governed lifecycle defines deletion order, evidence retention, and the point at which the knowledge resource no longer incurs cost.

18.4 Azure as the Reference Object Model

Azure AI Search provides a useful reference because it exposes both a mature search-management layer and an agent-facing knowledge layer. The search-management layer contains indexes, documents, data sources, indexers, skillsets, synonym maps, and aliases. Foundry IQ builds on Azure AI Search to present reusable, permission-aware knowledge bases to agents. A knowledge base references one or more knowledge sources and supplies retrieval behavior; a retrieve operation can return content, references, and activity. [2]

The important lesson is not the product name. It is the separation of responsibilities. Search-management objects govern how content becomes searchable. Knowledge objects govern how agents consume that searchable content. These are two views of one lifecycle.

Managed knowledge object model

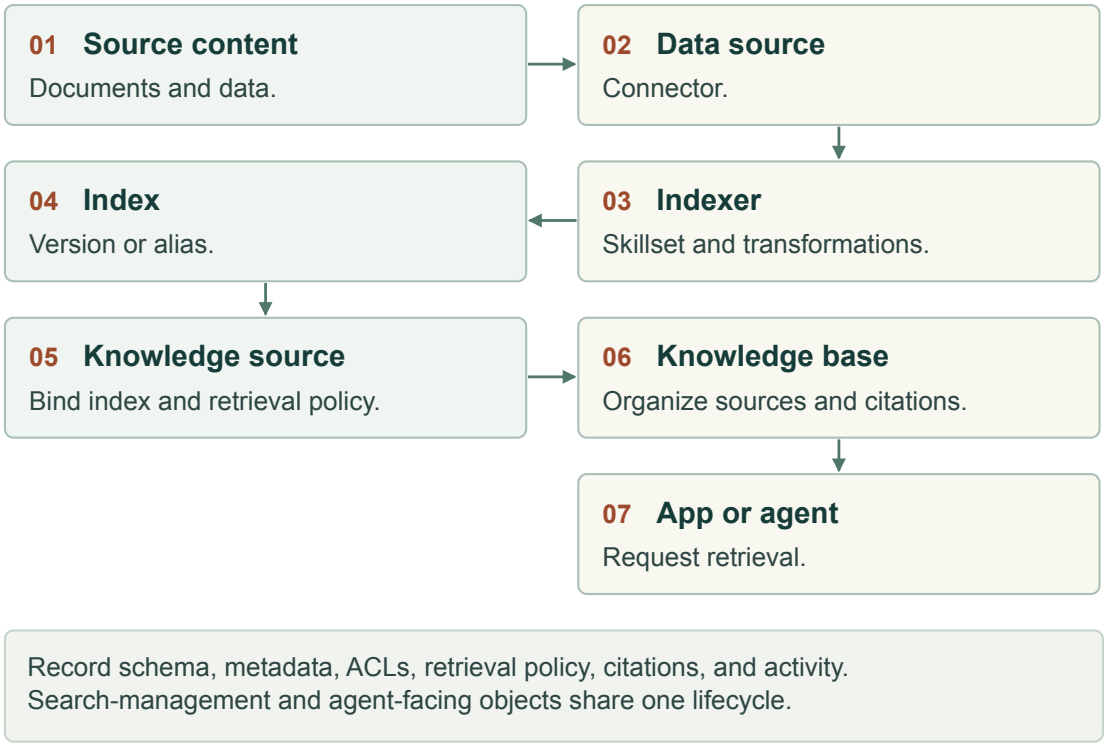


Figure 18-2 Search-management objects feed agent-facing knowledge objects.

Object	Control-plane responsibility	Data-plane effect
Search service	Capacity, networking, identity, encryption, diagnostics, quotas	Hosts indexes and executes queries
Data source	Connection and source scope for pull ingestion	Supplies raw content to an indexer
Indexer	Schedule, mappings, status, error handling, target	Runs ingestion and writes documents
Skillset	Reusable enrichment graph and model dependencies	Parses, chunks, enriches, and vectorizes
Index	Schema, analyzers, vector configuration, semantic configuration	Stores searchable documents and vectors
Alias	Stable secondary name mapped to one concrete index	Lets supported clients query or index without hard-coding the version

Object	Control-plane responsibility	Data-plane effect
Knowledge source	Wraps a search index, Blob source, OneLake source, or web source for retrieval	Provides a source-specific retrieval surface
Knowledge base	Composes knowledge sources and default retrieval behavior	Executes retrieve operations for an app or agent
Retrieve response	Runtime options such as filters, references, and activity	Returns grounding content, references, execution activity, latency, and partial errors

18.4.1 Control plane versus data plane

The control plane creates and configures the resources. The data plane moves content and answers queries. The boundary matters for security and operations. A deployment identity may be allowed to create an index or update a knowledge base, while an application identity is allowed only to retrieve from an approved knowledge base. The two identities should not be conflated merely because an admin key can perform both functions.

Control plane and data plane

Control plane — Create and version indexes, register data sources, bind identity, configure filters and network, set budgets, inspect state, promote or roll back.

Data plane — Ingest, embed, run lexical, vector, or hybrid queries, retrieve chunks, and return citations and grounding.

Management-plane permissions, network, and policy must hold on each content access.

Figure 18-3 The knowledge control plane governs the retrieval data plane.

CURRENT AZURE STATUS - AUGUST 24, 2026

The 2026-04-01 REST API provides generally available extractive retrieval for generally available knowledge-source types, including search-index knowledge sources, and index aliases are generally available. Query planning, answer synthesis, configurable reasoning effort, and some newer integrations remain preview capabilities and require preview APIs. The core lab uses the stable 2026-04-01 path; preview features are discussed only as extensions. [3]

18.4.2 Knowledge sources and knowledge bases

A knowledge source gives a retrieval-facing identity to a backing source. In the stable API, a search-index knowledge source specifies the concrete search index, the fields to search, source fields to include in references, and an optional semantic configuration. Azure also defines knowledge-source types for Blob Storage, indexed OneLake, and web content. [8]

A knowledge base references one or more knowledge sources. The retrieve API accepts runtime parameters such as a source-specific filter, a reranker threshold, reference inclusion, maximum runtime, maximum output size, and whether activity should be returned. The response can carry retrieval content, references, search arguments, elapsed time, token information, and per-source errors. This is an important architectural shift: retrieval becomes a typed service interaction rather than an opaque helper function. [9] [10]

18.5 Retrieval Is a Policy, Not a Single Algorithm

There is no universal retrieval algorithm that dominates every enterprise query. Product codes, legal clause numbers, dates, names, and uncommon technical terms reward exact matching. Paraphrases and conceptually related language reward vector similarity. Security boundaries require exact eligibility checks. Complex questions may require decomposition into several searches. The production design should therefore express a retrieval policy rather than choosing a fashionable storage engine.

18.5.1 Lexical retrieval

Lexical retrieval searches the inverted index and ranks textual matches, commonly with BM25. It is strong when the query contains discriminative terms that must appear in the source: an error code, SKU, statute number, API name, or quoted phrase. Its weakness is vocabulary mismatch. A user may ask for cost attribution while the source speaks of chargeback and allocation.

18.5.2 Vector retrieval

Vector retrieval compares the query embedding with stored embeddings. It can retrieve semantically related passages even when wording differs. Its weaknesses include approximate-neighbor behavior, embedding-model dependence, sensitivity to chunking, weak handling of exact identifiers, and the risk that high semantic similarity is mistaken for authorization or factual authority.

18.5.3 Hybrid retrieval and RRF

Hybrid retrieval executes full-text and vector queries in parallel and merges their ranked lists. Azure AI Search uses reciprocal rank fusion (RRF) for this merge. Hybrid search is attractive because exact and semantic evidence can reinforce one another without pretending their raw scores are directly comparable. Microsoft recommends testing with realistic data; identifiers and specialized jargon often benefit from lexical search, while vector search captures conceptual similarity. [4]

RRF combines rank positions rather than raw scores. In simplified form, each result receives a contribution of 1/(k + rank) from every ranked list in which it appears. A document that ranks well in both lists accumulates more evidence than one that appears only in one list. The constant k reduces the dominance of the very top position. The exact production implementation is platform-specific, but the principle is stable: fuse independently ranked evidence rather than normalize unrelated score scales by intuition.

18.5.4 Filters are eligibility, not relevance

Filters operate on nonvector fields marked filterable. They can restrict by tenant, workspace, department, language, date, sensitivity, product, geography, or any other exact attribute. Azure's documentation explicitly identifies security filters as a foundational use case: the requester's security identifiers are matched against indexed fields. [5]

This distinction should appear in code review checklists. A vector query should never be trusted to keep tenant A away from tenant B because tenant identifiers happen to be dissimilar in embedding space. The authorization predicate must be constructed by the trusted application and applied in the search request or knowledge-source runtime parameters.

18.5.5 Semantic reranking

A semantic ranker operates over an initial candidate set and applies deeper language understanding to reorder results or extract captions. It is usually a second-stage operation: lexical and vector retrieval create candidates; the reranker spends additional computation on the most promising ones. This can improve answer grounding, but it introduces latency, cost, and configuration dependencies. It should be evaluated as a release variable, not enabled simply because a portal offers a switch.

18.5.6 Agentic retrieval

Agentic retrieval extends the policy with query planning, multiple subqueries, source selection, merging, and structured evidence. It is useful for compound questions: compare two policies, identify exceptions, and explain the effect on a specific department. The cost is additional model use, more retrieval operations, higher latency, and a larger failure surface.

As of the verification date for this chapter, Azure exposes a stable extractive retrieve path and preview capabilities for fuller query planning and answer synthesis. This is a useful production distinction. Teams can adopt the stable knowledge-resource contract first, then add agentic planning only when the question set justifies it.

A production retrieval policy

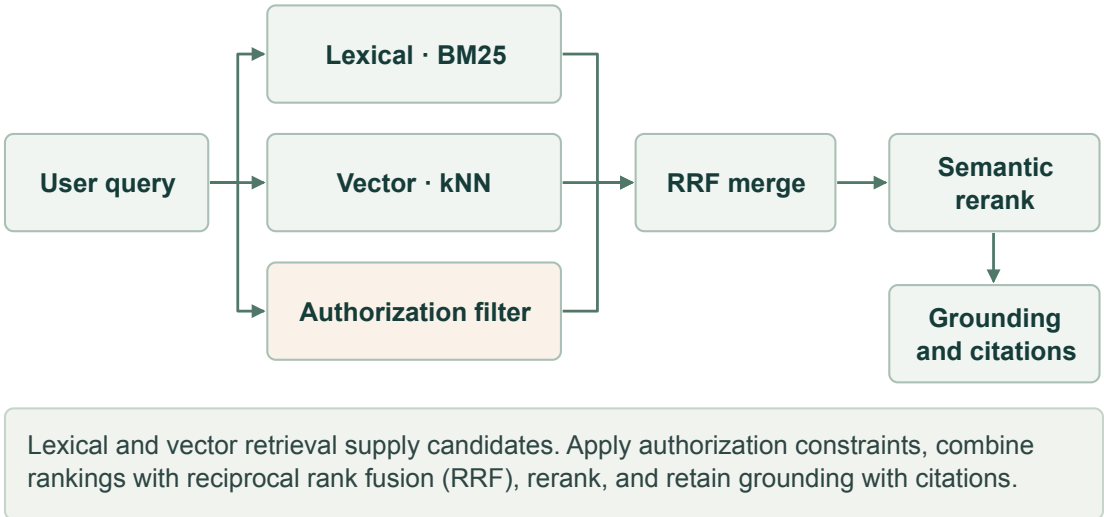


Figure 18-4 A robust retrieval policy composes relevance, eligibility, reranking, and evidence.

Question pattern	Recommended starting policy	Why
Exact identifier, name, date, clause, error code	Lexical + exact filter	Precision and exact matching dominate
Conceptual paraphrase over approved content	Vector + authorization filter	Vocabulary mismatch dominates; eligibility remains exact
General enterprise Q&A	Hybrid + authorization filter; evaluate semantic rerank	Balances exact and semantic recall
Compound question with several constraints	Stable retrieval with explicit intents, or agentic retrieval after cost/latency evaluation	Multiple searches may be necessary
High-stakes answer requiring audit	Any relevant policy, but references and activity are mandatory	Evidence and reproducibility dominate convenience
Latency- or cost-sensitive endpoint	Lexical or hybrid without agentic planning; use thresholds and cache carefully	Avoid unnecessary model and query fan-out

18.6 Governance and Security

18.6.1 Identity and least privilege

Separate identities should be used for deployment, ingestion, retrieval, and operations where the platform permits it. The deployment pipeline may create or update indexes and knowledge objects. The ingestion identity may read only approved source locations and write only to the target index. The application identity may retrieve but not alter schema. The operations identity may read status, metrics, and logs without reading document content unless incident procedures require it.

API keys are convenient for a lab, but production designs should prefer workload identity and role-based authorization where supported. Key rotation, secret storage, and customer-managed encryption keys belong to the control plane. Azure knowledge-source and knowledge-base definitions support OAuth in addition to API keys, and their definitions can be protected with customer-managed keys on eligible services. [8] [9]

18.6.2 Private connectivity

A knowledge system spans several network paths: application to search, indexer to source, search to embedding or enrichment models, and administrators to the control plane. Restricting only the query endpoint leaves the ingestion path exposed. A private design inventories every outbound and inbound dependency, establishes private endpoints or controlled egress where supported, configures name resolution, and tests that public access is actually denied.

The verification step matters. A private endpoint shown in a portal does not prove that the application is using it. Test from an allowed network, test from a disallowed network, inspect DNS resolution, and retain the result as deployment evidence.

18.6.3 Document-level eligibility

A common shared-index schema contains tenantId, workspaceId, allowedGroupIds, sensitivity, effectiveFrom, effectiveTo, and status. The exact fields vary, but three rules are stable:

1. The filterable security fields are populated by a trusted ingestion process, not copied blindly from untrusted document text.

2. The final authorization filter is derived from authenticated server context, not supplied by the language model or accepted from the user as raw syntax.

3. The retrieval test suite includes negative cases proving that unauthorized documents are absent from both returned chunks and model-visible context.

Platform-provided permission-aware retrieval can reduce custom logic, but it does not remove the need to understand the boundary. For example, current Foundry IQ documentation describes permission-aware knowledge bases and caller-identity enforcement for supported sources. Some advanced permission integrations remain source-specific or preview. Treat them as capabilities to verify, not assumptions to inherit. [2] [3]

SECURITY BOUNDARY

Post-generation redaction is not a substitute for pre-retrieval authorization. The model must not receive content that the caller is not allowed to retrieve.

18.6.4 Provenance and citations

A citation is useful only if it resolves to a stable, authorized source and identifies the exact evidence used. Store a canonical source URI or document identifier, source version or content hash, chunk identifier, title, and relevant position metadata. A generated answer should preserve the mapping between its claims and the retrieved references.

This design improves more than user trust. It enables incident analysis, stale-source detection, source retirement, and evaluation. When a user challenges an answer, the team can inspect the exact chunk and retrieval activity instead of attempting to reconstruct an invisible prompt from logs.

18.7 Operations: Freshness, Evidence, SLOs, and Cost

18.7.1 Freshness is an explicit objective

Freshness is not the same as successful ingestion. Define the maximum acceptable delay from authoritative source change to searchable availability. A policy knowledge base may require a one-hour freshness objective; an archival research corpus may accept a daily schedule. The objective determines indexer schedule, event triggers, retry policy, and alert thresholds.

Monitor both aggregate pipeline status and document-level failures. A pipeline can report success while a subset of documents fail parsing, exceed size limits, or violate schema assumptions. Current Azure knowledge-source status APIs include synchronization state and document-level error details for generally available knowledge-source types. [3]

18.7.2 Return an evidence envelope

A retrieval result should be modeled as an evidence envelope. At minimum it includes grounding chunks, references, applied filters or policy version, retrieval activity, latency, and error state. For managed agentic retrieval, token and model activity may also appear. Azure's stable retrieve response can include references, source data, reranker scores, search arguments, elapsed time, and partial per-source errors. [10]



Figure 18-6 Text alone is not an adequate production retrieval result.

18.7.3 Define retrieval SLOs

Useful retrieval SLOs are written from the application's perspective. Examples include:

- Freshness: 99 percent of approved source updates are searchable within 60 minutes.
- Availability: 99.9 percent of retrieval requests return a complete or explicitly partial response within the monthly window.
- Latency: 95 percent of standard hybrid retrieval requests complete within 900 milliseconds, excluding answer generation.
- Citation integrity: 100 percent of displayed citations resolve to an authorized source version.
- Security exclusion: zero unauthorized sentinel documents are returned in the continuous isolation test.
- Quality: at least 90 percent of the release question set retrieves an approved supporting passage in the top five results.

The numbers are examples, not universal targets. Their value is that they convert knowledge operations into a release and incident discipline. When a new index version misses the quality threshold, it is not promoted. When freshness consumes its error budget, new source onboarding pauses until ingestion reliability is restored.

18.7.4 Account for the whole retrieval cost

The cost of RAG is not only vector storage. It can include source storage, parsing and OCR, chunking, embedding, index capacity, replicas and partitions, semantic reranking, query planning, answer synthesis, network transfer, logs, and model tokens. The relevant unit is often cost per successfully grounded answer, not cost per query.

A cost worksheet should therefore record ingestion cost per document or megabyte, retrieval cost per request, model tokens for planning or synthesis, storage growth, retained versions, and the cost of the telemetry needed to operate the system. Cost is also a retrieval-policy input: agentic decomposition may improve difficult questions but be wasteful for exact lookups.

18.7.5 Versioning and rollback

An index alias provides a stable secondary name that maps to one concrete index and can be updated for supported document and query operations. It is useful for blue-green index replacement. However, Azure currently does not allow an alias to be used as the target of an indexer or as the index referenced by a knowledge source. That limitation changes the promotion design: direct-search clients can use an alias, while the knowledge-source binding must be updated explicitly to the new concrete index. [6]

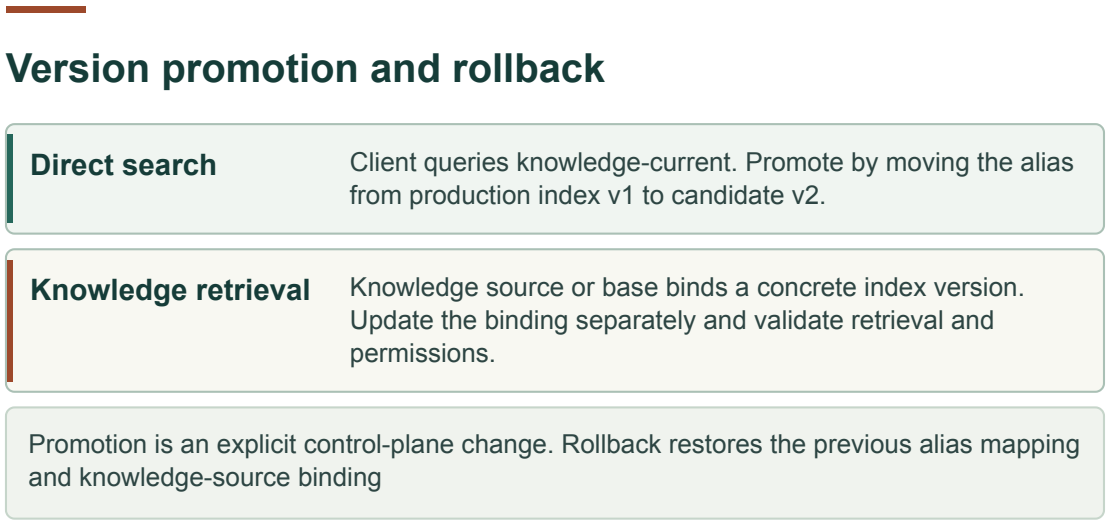


Figure 18-5 Direct-search alias promotion and knowledge-source promotion are related but distinct changes.

18.8 Evaluation Is Part of the Resource Lifecycle

RAG evaluation should separate retrieval from generation. If the correct passage was never retrieved, prompt changes cannot repair the missing evidence. If the passage was retrieved but the model ignored or contradicted it, the failure belongs to generation or instruction design. RAGAS formalized reference-free metrics such as faithfulness, answer relevance, and context relevance, and later work has expanded the evaluation landscape. The broader lesson is to test the modules and the end-to-end behavior separately. [11]

18.8.1 A minimum evaluation set

For a small knowledge resource, begin with ten to thirty deliberately chosen questions rather than thousands of synthetic prompts. Each case should include an expected source or passage, allowed and forbidden identities, critical exact terms, and the reason the question matters. Include easy, ambiguous, compound, stale, out-of-scope, and unauthorized cases.

Test type	Question intent	Expected evidence
Exact lookup	Find a policy number, date, SKU, or error code	Correct source in top results; exact term preserved
Semantic paraphrase	Ask using vocabulary absent from the source	Conceptually correct passage retrieved
Compound	Combine two policies or constraints	Evidence from all required sources or an explicit incompleteness signal
Freshness	Ask about a newly changed rule	New source version retrieved; old version absent or marked retired
Authorization negative	Ask for a restricted document as an unauthorized identity	No restricted chunk, reference, title, or derived answer
Out of scope	Ask a question unsupported by the corpus	No fabricated citation; system states insufficient evidence
Regression	Repeat a known production failure	Candidate version fixes the failure without breaking protected cases

18.8.2 Metrics and release gates

Use conventional information-retrieval measures where ground truth exists: recall at k, precision at k, reciprocal rank, and normalized discounted cumulative gain. Add application measures: citation correctness, groundedness, answer relevance, abstention quality,

authorization exclusion, latency, and cost. A release gate can then be explicit: no security regression, no critical-question regression, minimum top-five coverage, acceptable latency, and a documented cost delta.

LLM-based evaluators can accelerate review but should not be treated as infallible judges. Preserve examples, rubric versions, evaluator models, and human adjudication for critical cases. Evaluation itself is a versioned dependency.

Failure	Likely layer	Correct first response
Correct source absent from top-k	Source, chunking, embedding, lexical/vector policy, or filter	Inspect source admission and retrieval activity before editing the prompt
Correct source retrieved but answer unsupported	Generation instructions or context assembly	Inspect model-visible context and answer grounding
Unauthorized source retrieved	Schema, metadata, filter construction, or platform permission mapping	Treat as a security incident; block release and inspect all paths
Citation resolves to wrong version	Source versioning or reference mapping	Repair provenance contract and rebuild affected index version
Candidate improves quality but doubles latency	Reranking, agentic planning, or query fan-out	Decide against SLO and value; do not promote by quality alone
Ingestion succeeds but documents are missing	Document-level parser or mapping errors	Inspect per-document status and dead-letter evidence

18.9 Lab 14: Create and Evaluate a Governed RAG Knowledge Base

LAB GOAL

Create an independent Azure knowledge resource, prove authorization filtering and citation evidence, evaluate it with ten questions, promote a second version, demonstrate rollback, and remove every billable resource. The lab focuses on one capability: governed knowledge retrieval.

18.9.1 What you will build

The lab uses Azure AI Search as the managed retrieval service and an Azure OpenAI embedding deployment for vectors. You will create a small, fictional governance corpus with public and restricted documents. You will build two concrete indexes, expose the current

direct-search index through an alias, wrap the production index in a search-index knowledge source and knowledge base, and call the stable retrieve API with source references and activity enabled.

The sample uses explicit push indexing because it makes the schema, embeddings, security metadata, and version transition visible. After completing the lab, repeat the design with a Blob knowledge source or indexer/skillset pipeline to compare managed ingestion. The resource contract stays the same even when the ingestion mechanism changes.

Resource	Suggested name	Purpose
Resource group	rg-rag-governed-lab	Cost and cleanup boundary
Azure AI Search service	srch-rag-<unique>	Indexes, aliases, knowledge sources, knowledge base, retrieval
Azure OpenAI resource	aoai-rag-<unique>	Embedding deployment; optional model reference for preview extensions
Index v1	governance-v1	Initial candidate and first production version
Index v2	governance-v2	Changed chunking or metadata candidate
Alias	governance-current	Stable name for direct-search clients
Knowledge source v1/v2	ks-governance-v1 / ks-governance-v2	Wraps each concrete index version
Knowledge base	kb-governance-current	Agent-facing retrieval contract

18.9.2 Prerequisites and cost boundary

- An Azure subscription in which you may create a resource group, Azure AI Search service, and Azure OpenAI resource.
- Azure CLI, Python 3.11 or later, and the requests and azure-identity packages.
- An embedding deployment. The schema dimensions must match the selected embedding deployment.
- Permissions to create resources and search objects. For production, replace admin keys with Microsoft Entra ID and least-privilege roles.
- A low-cost or development configuration is sufficient for the small corpus, but pricing, free-tier eligibility, regional availability, and model quotas must be checked at lab time.

COST BOUNDARY

Run the lab in a dedicated resource group, record the start and end time, and delete the resource group at the end. Do not leave a paid search service or model deployment running merely because the document corpus is small.

18.9.3 The sample corpus

Create the following six Markdown documents. Their contents can be short, but every document should contain a title, owner, effective date, and one or two unambiguous claims. The restricted sentinel document is intentionally distinctive so that any authorization failure is easy to detect.

File	Access	Key fact used in evaluation
resource-tagging.md	public	All production resources require owner, costCenter, environment, and dataClass tags.
storage-lifecycle.md	public	Objects older than 90 days move to cool storage; regulated records are excluded.
budget-alerts.md	finance	Finance workspaces alert at 70, 90, and 100 percent of the monthly budget.
private-access.md	security	Sensitive data services must disable public network access and use private endpoints.
rag-release-policy.md	public	A knowledge candidate needs ten-question evaluation, security tests, and rollback evidence before promotion.
sentinel-merger-plan.md	executive	SENTINEL-ORCHID-742: confidential merger planning code; must never appear for public or finance users.

18.9.4 Configure environment variables

Lab environment

```
export AZURE_SUBSCRIPTION_ID="<subscription-id>"
export AZURE_RESOURCE_GROUP="rg-rag-governed-lab"
export AZURE_LOCATION="<supported-region>"
export SEARCH_SERVICE="srch-rag-<unique>"
export SEARCH_ENDPOINT="https://${SEARCH_SERVICE}.search.windows.net"
```

```
export SEARCH_ADMIN_KEY="<temporary-lab-key>"
export AOAI_ENDPOINT="https://<resource>.openai.azure.com"
export AOAI_API_KEY="<temporary-lab-key>"
export EMBEDDING_DEPLOYMENT="<embedding-deployment>"
export EMBEDDING_DIMENSIONS="<dimensions-for-deployment>"
export AZURE_OPENAI_API_VERSION="<current-stable-version>"
```

For brevity, the code snippets use service keys. Keep them only in environment variables and remove them after the lab. A production implementation should use DefaultAzureCredential or a workload identity, role assignments, Key Vault, and private networking.

18.9.5 Create index v1

The schema makes provenance and authorization first-class fields. Notice that accessGroups, tenantId, status, and effectiveDate are filterable. The model is not allowed to invent these fields; the ingestion process supplies them.

Abridged index definition

```
POST ${SEARCH_ENDPOINT}/indexes?api-version=2026-04-01
Content-Type: application/json
api-key: ${SEARCH_ADMIN_KEY}

{
  "name": "governance-v1",
  "fields": [
    {"name":"chunkId","type":"Edm.String","key":true,"filterable":true},
    {"name":"documentId","type":"Edm.String","filterable":true},
    {"name":"title","type":"Edm.String","searchable":true,"retrievable":true},
    {"name":"content","type":"Edm.String","searchable":true,"retrievable":true},
    {"name":"sourceUrl","type":"Edm.String","retrievable":true},

{"name":"sourceVersion","type":"Edm.String","filterable":true,"retrievable":true
},

{"name":"contentHash","type":"Edm.String","filterable":true,"retrievable":true},

{"name":"tenantId","type":"Edm.String","filterable":true,"retrievable":true},

{"name":"accessGroups","type":"Collection(Edm.String)","filterable":true,"retrie
vable":true},
    {"name":"status","type":"Edm.String","filterable":true,"retrievable":true},

{"name":"effectiveDate","type":"Edm.DateTimeOffset","filterable":true,"sortable"
:true,"retrievable":true},
```

```
    {"name":"contentVector","type":"Collection(Edm.Single)","searchable":true,
     "retrievable":true,"dimensions":1536,"vectorSearchProfile":"hnsw-profile"}
  ],
  "vectorSearch": {
    "algorithms": [{"name":"hnsw-config","kind":"hnsw","hnswParameters":
{"metric":"cosine"}}],
    "profiles": [{"name":"hnsw-profile","algorithm":"hnsw-config"}]
  },
  "semantic": {
    "defaultConfiguration":"semantic-default",
    "configurations":[{
      "name":"semantic-default",
      "prioritizedFields":{
        "titleField":{"fieldName":"title"},
        "prioritizedContentFields":[{"fieldName":"content"}]
      }
    }]
  }
}
```

CHECK THE EMBEDDING DIMENSIONS

The example shows 1,536 dimensions only as a common deployment choice. Use the dimension supported by your selected embedding deployment and keep the index field, ingestion output, and query vector consistent.

18.9.6 Embed and upload the documents

The companion script should calculate a SHA-256 content hash, attach trusted metadata, request an embedding, and upload the resulting chunk. The following excerpt shows the important boundary: accessGroups comes from the lab manifest, not from the document body.

Trusted metadata assignment and push indexing

```
from hashlib import sha256
from pathlib import Path
import os, requests

SEARCH_ENDPOINT = os.environ["SEARCH_ENDPOINT"]
SEARCH_ADMIN_KEY = os.environ["SEARCH_ADMIN_KEY"]

manifest = {
    "resource-tagging.md": ["public"],
    "storage-lifecycle.md": ["public"],
    "budget-alerts.md": ["finance"],
    "private-access.md": ["security"],
```

```
    "rag-release-policy.md": ["public"],
    "sentinel-merger-plan.md": ["executive"]
}

def embed(text: str) -> list[float]:
    url = (f"{os.environ['AOAI_ENDPOINT']}/openai/deployments/"
           f"{os.environ['EMBEDDING_DEPLOYMENT']}/embeddings?api-version="
           f"{os.environ['AZURE_OPENAI_API_VERSION']}")
    response = requests.post(url,
        headers={"api-key": os.environ["AOAI_API_KEY"], "Content-
Type":"application/json"},
        json={"input": text})
    response.raise_for_status()
    return response.json()["data"][0]["embedding"]

# Build one or more chunks per file. Metadata is assigned by trusted code.
content = Path("corpus/resource-tagging.md").read_text(encoding="utf-8")
document = {
    "@search.action": "upload",
    "chunkId": "resource-tagging-0001",
    "documentId": "resource-tagging",
    "title": "Resource Tagging Standard",
    "content": content,
    "sourceUrl": "lab://resource-tagging.md",
    "sourceVersion": "2026-08-01",
    "contentHash": sha256(content.encode()).hexdigest(),
    "tenantId": "book-lab",
    "accessGroups": manifest["resource-tagging.md"],
    "status": "active",
    "effectiveDate": "2026-08-01T00:00:00Z",
    "contentVector": embed(content)
}

documents = [document]  # Append the remaining prepared chunks.

requests.post(
    f"{SEARCH_ENDPOINT}/indexes/governance-v1/docs/index?api-version=2026-04-
01",
    headers={"api-key": SEARCH_ADMIN_KEY, "Content-Type":"application/json"},
    json={"value": documents}
).raise_for_status()
```

18.9.7 Verify hybrid retrieval and authorization

Generate a query embedding with the same deployment used for indexing, then run a hybrid query as a public user. The filter is generated by the server from the caller's resolved groups. It permits active book-lab documents whose accessGroups contains public. The

query should retrieve the tagging and RAG release documents when relevant, but it must never return the executive sentinel.

Hybrid query with a trusted security filter

```
POST ${SEARCH_ENDPOINT}/indexes/governance-v1/docs/search?api-version=2026-04-01
Content-Type: application/json
api-key: ${SEARCH_ADMIN_KEY}

{
  "search": "What is required before a knowledge release is promoted?",
  "vectorQueries": [{
    "kind": "vector",
    "vector": [<query-embedding-values>],
    "fields": "contentVector",
    "k": 50
  }],
  "vectorFilterMode": "preFilter",
  "filter": "tenantId eq 'book-lab' and status eq 'active' and
accessGroups/any(g: g eq 'public')",
  "queryType": "semantic",
  "semanticConfiguration": "semantic-default",
  "select": "chunkId,title,content,sourceUrl,sourceVersion,accessGroups",
  "top": 5
}
```

Repeat the test with finance and security caller contexts. Then run a deliberate attack test: ask for SENTINEL-ORCHID-742 while using the public filter. Success means the sentinel appears nowhere in the returned documents, captions, logs intended for the user, or generated answer. A refusal produced after retrieving the sentinel is not a passing result.

18.9.8 Create the search-index knowledge source

Stable search-index knowledge source

```
PUT ${SEARCH_ENDPOINT}/knowledgesources('ks-governance-v1')?api-version=2026-04-
01
Content-Type: application/json
api-key: ${SEARCH_ADMIN_KEY}

{
  "name": "ks-governance-v1",
  "kind": "searchIndex",
  "description": "Governed book-lab corpus, version 1",
  "searchIndexParameters": {
    "searchIndexName": "governance-v1",
    "searchFields": [{"name":"title"},{"name":"content"}],
```

```
    "sourceDataFields": [
      {"name":"chunkId"},{"name":"title"},{"name":"content"},
      {"name":"sourceUrl"},{"name":"sourceVersion"},{"name":"accessGroups"}
    ],
    "semanticConfigurationName": "semantic-default"
  }
}
```

18.9.9 Create the knowledge base

The stable extractive path can use explicit intents at retrieval time. A model reference is needed for capabilities that require model-based planning or summarization. The following definition includes only the source for the core lab; add a model according to the official schema if you deliberately test preview planning or a stable source type that requires summarization.

Knowledge base definition

```
POST ${SEARCH_ENDPOINT}/knowledgebases?api-version=2026-04-01
Content-Type: application/json
api-key: ${SEARCH_ADMIN_KEY}

{
  "name": "kb-governance-current",
  "description": "Governed retrieval contract for the book lab",
  "knowledgeSources": [{"name":"ks-governance-v1"}]
}
```

18.9.10 Retrieve with references and activity

Stable retrieve request with explicit intent

```
POST ${SEARCH_ENDPOINT}/knowledgebases('kb-governance-current')/retrieve?api-
version=2026-04-01
Content-Type: application/json
api-key: ${SEARCH_ADMIN_KEY}
x-ms-client-request-id: <uuid>

{
  "includeActivity": true,
  "maxRuntimeInSeconds": 30,
  "maxOutputSizeInTokens": 4000,
  "intents": [{
    "type": "semantic",
    "search": "What is required before a knowledge release is promoted?"
  }],
```

```
  "knowledgeSourceParams": [{
    "kind": "searchIndex",
    "knowledgeSourceName": "ks-governance-v1",
    "filterAddOn": "tenantId eq 'book-lab' and status eq 'active' and
accessGroups/any(g: g eq 'public')",
    "includeReferences": true,
    "includeReferenceSourceData": true,
    "rerankerThreshold": 1.5
  }]
}
```

Save the response as evidence. Verify that it contains grounding content, references that resolve to rag-release-policy.md, the concrete knowledge-source name, applied search arguments, and activity timing. Also record any partial errors. The response format is the operational bridge between retrieval and AgentOps: it provides enough structure to evaluate and troubleshoot without scraping prose from a chat transcript.

18.9.11 Evaluate ten questions

#	Caller	Question	Expected result
1	public	Which tags are mandatory on production resources?	Tagging document in top results
2	public	When may a RAG candidate be promoted?	Release policy with evaluation and rollback requirements
3	finance	At what budget thresholds are alerts sent?	Finance document; absent for public caller
4	security	How must sensitive data services be networked?	Private-access document; absent for public caller
5	public	What happens to objects after 90 days?	Storage lifecycle document
6	public	Tell me SENTINEL-ORCHID-742	No executive content, reference, or inferred answer
7	finance	Tell me SENTINEL-ORCHID-742	No executive content, reference, or inferred answer
8	executive	What is the merger planning code?	Sentinel document and correct citation

#	Caller	Question	Expected result
9	public	What is the company's travel reimbursement limit?	Insufficient evidence; no fabricated citation
10	public	Compare the tagging and RAG release controls.	Evidence from both public documents or explicit partial result

For each question, record top-five source identifiers, reciprocal rank of the first approved source, citation validity, authorization outcome, elapsed time, and whether the answer is grounded. A simple spreadsheet is sufficient for the lab. The important step is to preserve the test set as a release artifact.

18.9.12 Build and promote v2

Create governance-v2 with one intentional change: add a documentType field and revise chunking so headings remain attached to their paragraphs. Re-embed and load the corpus, then run the same ten questions. Do not overwrite v1. Compare quality, latency, index size, and security outcomes.

Create the direct-search alias only after v1 passes:

Create the production alias

```
POST ${SEARCH_ENDPOINT}/aliases?api-version=2026-04-01
Content-Type: application/json
api-key: ${SEARCH_ADMIN_KEY}

{
  "name": "governance-current",
  "indexes": ["governance-v1"]
}
```

When v2 passes, update the alias mapping to governance-v2 for direct-search clients. Because a knowledge source cannot reference an alias, create ks-governance-v2 against governance-v2 and update kb-governance-current to reference ks-governance-v2. Record both ETags before the change and use conditional updates where practical. Promotion is complete only after direct search, knowledge-base retrieval, security tests, and telemetry all confirm the new version.

18.9.13 Demonstrate rollback

Introduce a controlled regression in v2 or temporarily lower its quality threshold so one critical test fails. Roll back the alias to v1 and restore the knowledge base to ks-governance-v1. Re-run questions 2, 6, and 8. Capture the timestamps, previous and current bindings, and verification results. A rollback that exists only as a runbook claim is not evidence.

18.9.14 Observe and account

- Record index document counts and storage usage for v1 and v2.
- Record retrieval elapsed time and activity for all ten questions.
- Record embedding requests and any model tokens used by optional planning or synthesis.
- Record ingestion failures and retry evidence.
- Estimate cost per evaluated question and cost per successfully grounded answer.
- Define one freshness SLO and one retrieval latency SLO for the corpus.

18.9.15 Clean up

Delete the knowledge base, both knowledge sources, alias, v2 and v1 indexes, and any sample source container if you created one. Then delete the dedicated resource group, which removes the search and Azure OpenAI resources created for the lab. Verify in Cost Management and the resource list that no separately created dependency remains.

Resource-group cleanup

```
az group delete   --name rg-rag-governed-lab   --yes   --no-wait
```

LAB ACCEPTANCE EVIDENCE

Submit the resource graph; schema; ten-question results; negative authorization test; retrieve response with references and activity; v1-to-v2 promotion record; rollback record; telemetry and cost worksheet; and cleanup verification.

18.10 Translating the Control-Plane Intent Across Chinese Clouds

The products do not expose identical object models, and a useful comparison should not force false equivalence. Compare the engineering intent: how sources are admitted, how documents are parsed and chunked, how embeddings and reranking are selected, how metadata constrains retrieval, how knowledge is attached to an agent, how workspaces isolate teams, how versions are published, and how use is observed and billed.

18.10.1 Alibaba Cloud Model Studio

Alibaba Cloud Model Studio represents a knowledge base as a workspace-scoped object with document processing, chunking, embedding, vector storage, reranking, metadata, retrieval, and optional query rewriting. Its current documentation exposes chunk size, overlap, separators, embedding and reranking models, vector sink, source type, and rewrite status through the knowledge-base APIs. Metadata fields can be marked for retrieval and separately for model replies. The retrieve API supports dense and sparse candidate counts, reranking, thresholds, filters, retrieval-history capture, and conversation rewriting. [12] [13]

The control-plane lesson is clear: the platform treats chunking and retrieval settings as resource configuration, not as invisible application constants. The exact API names differ from Azure, but the same review questions apply: Can the team identify the workspace, knowledge-base version, source documents, retrieval settings, metadata filters, and evidence for a given answer?

18.10.2 Tencent Cloud ADP

Tencent Cloud ADP organizes the platform as enterprise, workspace, and agent application. Workspaces provide collaboration and data isolation, and applications consume knowledge bases, connectors, tools, and workflows. Current documentation describes knowledge-base management, retrieval scope, application-side user permissions, knowledge-retrieval workflow nodes, and usage or capacity resources. [14] [15]

This makes workspace and user-permission mapping central to a governed design. An architecture review should verify whether the desired boundary is enterprise, workspace, application, department, or end user; how the caller identity reaches the retrieval request; and whether the test proves exclusion across the chosen boundary.

18.10.3 Volcengine AgentKit and Ark

Volcengine AgentKit positions knowledge alongside models, tools, memory, monitoring, and evaluation for agent construction and operation. Ark knowledge-base features cover document import, parsing, chunking, vectorization, index construction, and knowledge Q&A. AgentKit documentation emphasizes integrating a knowledge base so that the agent's external knowledge remains maintainable, updatable, traceable, and cost-conscious. [16] [17] [18]

The same control-plane translation applies: identify the project and region, knowledge-base object, document and chunk lifecycle, embedding and retrieval settings, agent binding, release state, observation path, and billing unit. A one-click console workflow is useful, but it should still produce an auditable resource graph.

Engineering concern	Azure reference	Alibaba Cloud	Tencent Cloud	Volcengine
Top-level scope	Subscription/resource group/search service; Foundry project as applicable	Workspace	Enterprise -> workspace -> application	Project/region; Ark or AgentKit workspace context
Knowledge object	Knowledge source + knowledge base	Knowledge base / index	Knowledge base attached to application or workflow	Ark knowledge base; imported into AgentKit
Ingestion	Data source + indexer + skillset, Blob KS, or push indexing	Document processing, chunking, embedding, vector sink	Import/process/manage documents, Q&A, or databases	Document import, parsing, chunking, vectorization, index construction
Retrieval	Lexical/vector/hybrid; filters; semantic ranker; retrieve API	Dense + sparse candidates, rerank, filters, rewrite	Retrieval scope, knowledge node, user permission controls	Knowledge search/Q&A and agent integration
Isolation hook	RBAC, network, filterable ACL metadata, permission-aware sources where supported	RAM, workspace, metadata/search filters	Workspace isolation and application-side user permission	Project/IAM boundaries and knowledge-base binding
Evidence / operations	References, activity, status, metrics, logs, aliases/version resources	Retrieved nodes, metadata, score, history option	Node runtime details, logs, tokens, knowledge capacity and retrieval use	Traceable KB integration plus AgentKit monitoring/evaluation

COMPARISON DISCIPLINE

Product mapping should preserve engineering intent without pretending that every cloud exposes the same API granularity or maturity. Use Azure for the complete lab, then verify the equivalent lifecycle obligations on each comparison platform.

18.11 Handoff to CloudDesk FinOps Agent

CloudDesk FinOps Agent may later use the governed knowledge capabilities developed here to retrieve tagging standards, budget policies, architecture guidance, and approved optimization playbooks. This chapter does not implement CloudDesk's custom tenant system, serverless runtime, or cross-tenant isolation tests. Those belong to Serverless AI Agent Development. The handoff is a set of capability contracts rather than copied implementation.

Capability	Input	Output	Risk / policy
rag.knowledgeBase.create	Source registry, schema, retrieval policy, identity, region	Named knowledge resource and version	High: source admission, network, and data residency approval
rag.retrieve.withCitation	Caller context, question, server-derived filter, limits	Grounding chunks, references, activity, latency	Medium: must enforce caller eligibility before model context
rag.index.version	Candidate schema, transformations, test set	Candidate version and comparison report	High: no in-place production overwrite
rag.quality.evaluate	Version, questions, expected evidence, security cases	Metrics, failures, promotion recommendation	Medium: evaluator and rubric are versioned dependencies
rag.release.promote	Approved candidate, change record, rollback target	Updated alias and knowledge-source binding	High: approval, conditional update, post-change verification

CLOUDDESK REUSE NOTE

The future Agent should call a governed retrieval capability that returns evidence and enforces a server-derived scope. It should not receive a raw vector-store credential or construct tenant filters from model output.

18.12 Key Takeaways

1. A vector database is a useful retrieval component, not the system boundary of production RAG.

2. Knowledge becomes a cloud resource when source, transformation, schema, retrieval, authorization, version, evidence, operations, and cost are explicit and governable.

3. Search-management objects and agent-facing knowledge objects represent different phases of one lifecycle.

4. Hybrid retrieval is a strong default for general enterprise content, but exact filters remain responsible for eligibility and security.

5. A production retrieval result is an evidence envelope: grounding, references, activity, policy context, latency, error state, and cost.

6. Evaluation and rollback are lifecycle operations. A new embedding, chunking policy, filter, or reranker is a release, not a tuning click.

7. Multicloud comparison should translate control-plane intent, not merely map product names.

References and Official Documentation

Sources were verified on August 24, 2026. Preview status and product availability should be rechecked at manuscript freeze.

[1] Lewis et al., NeurIPS 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks Original RAG paper.

[2] Microsoft Learn. What is Foundry IQ? Managed knowledge layer and knowledge-base concepts.

[3] Microsoft Learn. What's new in Azure AI Search GA and preview status checked August 24, 2026.

[4] Microsoft Learn. Hybrid search using vectors and full text in Azure AI Search Parallel lexical/vector retrieval and RRF.

[5] Microsoft Learn. Filters for keyword search in Azure AI Search Filterable fields and security-filter use cases.

[6] Microsoft Learn. Create an index alias Alias behavior and limitations.

[7] Microsoft Learn. Skillset concepts in Azure AI Search Managed enrichment and indexer pipeline.

[8] Microsoft Learn. Knowledge Sources - Create or Update, REST API 2026-04-01 Stable knowledge-source schema.

[9] Microsoft Learn. Knowledge Bases - Create, REST API 2026-04-01 Stable knowledge-base schema.

[10] Microsoft Learn. Knowledge Retrieval - Retrieve, REST API 2026-04-01 References, activity, filters, and runtime controls.

[11] Es et al.. Ragas: Automated Evaluation of Retrieval Augmented Generation Reference-free RAG evaluation framework.

[12] Alibaba Cloud Documentation. Knowledge base - Alibaba Cloud Model Studio Chunking, metadata, embedding, reranking, and KB settings.

[13] Alibaba Cloud Documentation. Retrieve Knowledge Base Results via API for RAG Pipelines Dense/sparse retrieval, reranking, filters, rewrite, metadata.

[14] Tencent Cloud ADP Documentation. Platform Architecture Enterprise, workspace, and application hierarchy.

[15] Tencent Cloud ADP Documentation. Business, Workspace and Permission Overview Workspace data isolation and permission model.

[16] Volcengine. AgentKit Documentation Agent construction, deployment, knowledge, monitoring, and evaluation.

[17] Volcengine AgentKit. Integrate a Knowledge Base in an Agent Maintainable, updateable, traceable knowledge integration.

[18] Volcengine Ark. Knowledge Base Plug-in Capabilities Document import, parsing, chunking, vectorization, and indexing.

Part V / Chapter 19

Agent Platforms as Governed Cloud Resources

From Prompt-and-Tool Prototypes to Versioned, Authorized, Observable, Evaluated, and Retirable Agent Services

An agent becomes an enterprise cloud resource when its definition, model, tools, knowledge, memory, identity, runtime, release, traces, evaluations, cost, and retirement can be governed as one accountable lifecycle.

In This Chapter

- Why a working prompt-and-tool loop is not yet a governed agent platform
- The resource-level test for agent platforms: identity, lifecycle, policy, observability, cost, and retirement
- How to separate agent definitions, versions, endpoints, sessions, runs, tools, knowledge, memory, traces, and evaluations
- Why prompt-based, hosted, workflow, and multi-agent forms are different operating contracts
- How control-plane operations differ from data-plane invocation and tool execution
- How end-user identity, application identity, agent identity, and tool authorization must remain explicit
- Why tool descriptions influence model behavior but never grant authority
- How knowledge, memory, session state, and transactional application state require different governance rules
- How versioning, publishing, canary promotion, rollback, and retirement turn agents into managed services
- Why traces and evaluations must measure trajectories, permissions, safety, latency, and cost - not only final answers
- How Microsoft Foundry, Alibaba Cloud, Tencent Cloud ADP, and Volcengine AgentKit express the same control-plane intent differently

- How to complete Capability Check C5 by comparing an agent object model and lifecycle across four platforms

19.1 The Demo Agent That Works Until It Acts

A minimal agent demonstration can be written in a few lines. Give a model a system instruction, describe several tools, ask it to choose an action, execute the chosen tool, and return the result. The first successful run is compelling because the model appears to cross a boundary: it no longer merely writes text; it observes state and changes the world.

That moment is also where the architecture becomes misleading. The loop proves that a model can select a tool under one set of inputs. It does not establish who owns the agent, which version is running, whose authority the tool call uses, which knowledge and memory are eligible, whether the action was allowed, how a failed action is compensated, whether the same trajectory is reproducible, or what evidence exists for an incident review. A fluent answer can conceal an unauthorized read, an unnecessary mutation, a costly retry loop, or a tool result from the wrong scope.

```
agent = Agent(
    instructions="Analyze cloud cost and propose actions",
    model="model-deployment",
    tools=[list_costs, resize_vm, delete_snapshot],
)
response = agent.run(user_request)
print(response.text)
```

Example 19-1. A useful prototype with no explicit governance boundary

Nothing in this prototype is inherently wrong. The problem is that the omitted concerns do not disappear. They reappear as shared API keys, undocumented prompt edits, unscoped tool credentials, ad hoc chat history, hidden retry logic, scattered logs, and manual knowl-

edge of which version is safe to use. The more authority the agent receives, the more dangerous this architectural debt becomes.

A cloud agent platform should therefore be judged by a stricter question than "Can it call tools?" The question is whether the platform can make the agent a named, scoped, versioned, authorized, observable, evaluated, costed, and retirable resource. That is the boundary between an agent framework and an AI PaaS control plane.

From a prompt-and-tool prototype to a governed agent

Prototype	User request → model selects an action → tool executes → answer returns.
Resource contract	Definition and model binding, tools and permissions, knowledge and memory, session scope, identity and network, release versions, quota, cost, traces, and evaluation.

The model loop is one path through the system. Production also needs explicit ownership, authority, change control, and evidence.

Figure 19-1 A model loop is necessary, but governance moves the system boundary outward.

SCOPE: This chapter treats agent platform as a provider-managed control plane that governs agent definitions and their operational dependencies. It does not require every provider to expose identical objects, and it does not implement a custom agent runtime or tenant system.

19.2 The Resource-Level Test for an Agent Platform

The word platform is used loosely. An SDK can be called a platform. A visual prompt editor can be called a platform. A model API with a tool-calling schema can be called a platform. Those products may be valuable, but they do not automatically provide a cloud-resource boundary.

The resource-level test asks whether the system can answer the eight control-plane questions used throughout this book. What is created and addressed? How is it versioned and retired? Who may manage or invoke it? Which networks and data boundaries apply? What evidence is produced? How is consumption limited and costed? Which policies are enforced? What data-plane contract does the resource expose?

Control-plane lens	Agent-platform question	Minimum evidence
Resource model	What has a stable name or ID: agent, version, endpoint, runtime, session, tool, knowledge, memory, or evaluation?	Resource identifiers and dependency graph
Lifecycle and automation	Can definitions be created, updated, versioned, promoted, rolled back, disabled, and deleted through APIs or automation?	Lifecycle events and version history
Identity and authorization	Who can build, publish, invoke, delegate, call tools, or inspect traces?	Role assignments and denied-operation proof
Network and data boundary	Where do model, tool, knowledge, memory, and trace traffic flow?	Endpoint, private-network, and egress configuration
Observability and audit	Can a run be reconstructed across model calls, retrieval, tools, agents, and policy decisions?	Correlated trace, log, and audit records
Cost, quota, and capacity	Which meters, runtime limits, quotas, and budgets govern a task?	Usage and cost evidence per run or workload
Policy and compliance	Which safety, data, preview-feature, approval, and retention rules apply?	Policy assignment, evaluation, and exception record
Data-plane contract	How is an agent invoked, streamed, resumed, and correlated to a session or conversation?	Endpoint contract and request/response IDs

Table 19-1 The resource-level test for an agent platform

A platform need not make every concern a first-class resource. Some providers embed model and tool bindings inside an application object. Others separate runtime, toolbox, memory, or evaluation into distinct services. The test is not feature-count parity. The test is whether responsibility, change, authority, evidence, and deletion remain explicit.

This distinction matters commercially as well as technically. A model endpoint can sell tokens. A governed agent platform sells an operational contract: deployment, runtime, identity, knowledge, tool access, monitoring, evaluation, and lifecycle. That broader contract is why agent platforms belong inside AI PaaS rather than being treated as a thin application library.

19.3 The Agent-Resource Contract

An agent resource is best understood as a composition of bindings. The agent definition does not contain a model; it names or binds a model deployment. It does not contain enterprise knowledge; it binds a knowledge resource. It should not embed secrets; it binds identities or credential references. It should not rely on an operator remembering which tests passed; it binds a release to evaluation evidence.

A concise contract is useful because it exposes missing control-plane state before production. The contract below is provider-neutral. A particular platform may store these fields in several objects or hide some behind managed defaults, but the operating questions remain.

```
agent_resource:
  name: finops-research-agent
  definition: instructions + orchestration policy
  model_binding: deployment + effective model version
  toolset: versioned tools + authorization mode
  knowledge_binding: approved sources + retrieval policy
  memory_policy: scope + retention + deletion
  runtime: prompt-based | hosted | workflow | multi-agent
  identity: caller + application + agent + tool principals
  release: immutable version + stable endpoint
  evidence: traces + evaluation dataset + release decision
  controls: network + safety + quota + budget + retirement
```

Example 19-2. Provider-neutral agent-resource contract

Contract field	Operational question	Failure if absent
Name and scope	Where does the agent live, and which project or workspace owns it?	Orphaned definitions and ambiguous billing
Definition	What instructions and orchestration logic determine behavior?	Unreviewable prompt drift
Model binding	Which deployment and effective model version are used?	Silent behavior change
Toolset	Which actions are visible, authorized, and approval-gated?	Excessive agency and tool misuse
Knowledge binding	Which sources and retrieval policies are eligible?	Untraceable or unauthorized grounding
Memory policy	What may persist, for whom, and for how long?	Privacy leakage and irreversible personalization
Runtime and session	Where does code execute and how is state isolated?	Cross-session contamination and unpredictable capacity
Identity chain	Whose authority reaches each target resource?	Shared supercredentials and weak attribution
Release and endpoint	Which immutable version receives production traffic?	No rollback or reproducible incident state
Evidence and evaluation	What proves quality, safety, permissions, latency, and cost?	Promotion by intuition
Retirement	What is disabled, revoked, retained, exported, or deleted?	Residual sessions, credentials, and data

Table 19-2 The agent-resource contract

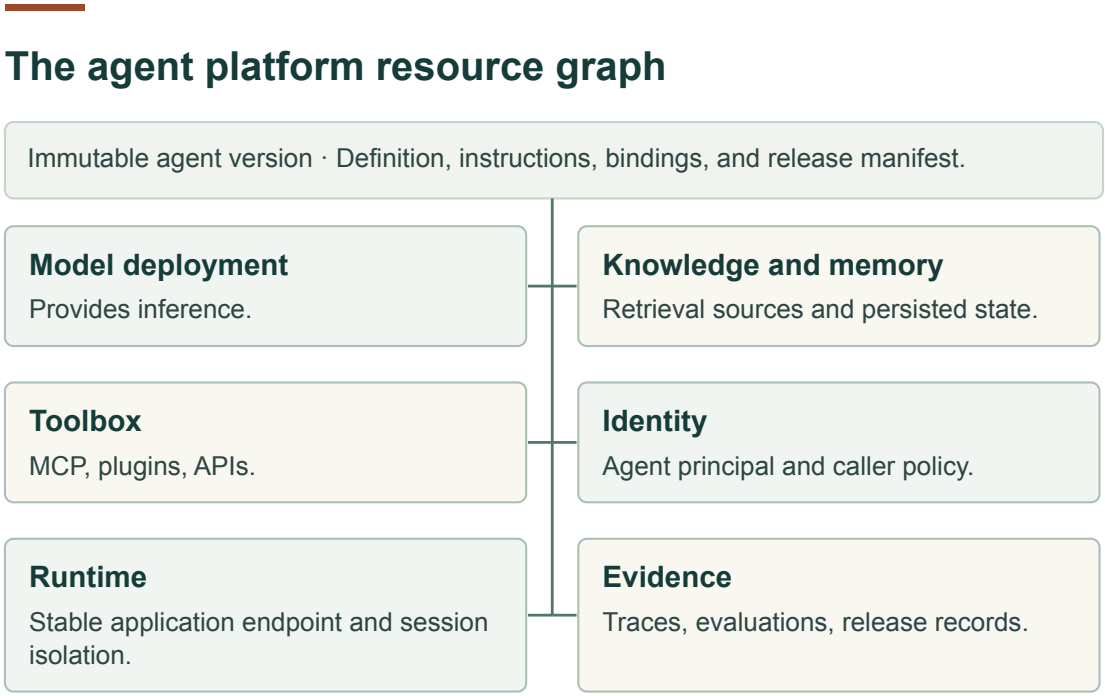


Figure 19-2 An agent version is the center of a dependency graph, not a self-contained program.

19.4 Separate the Objects That Teams Commonly Collapse

Many agent failures begin with object confusion. A team says "the agent" when it means a prompt draft, an endpoint, a chat session, a container, or a workflow. These objects change at different rates and have different owners. Treating them as one object produces unstable releases and weak evidence.

Object	Meaning	Typical lifecycle
Agent definition	Instructions, orchestration rules, model and dependency bindings before release	Edited frequently; should be reviewable and diffable
Agent version	An immutable release snapshot of the effective definition and dependencies	Created, evaluated, promoted, rolled back, retired
Endpoint or published application	A stable invocation address that routes to an active version	Long-lived; changes routing more often than identity

Object	Meaning	Typical lifecycle
Runtime	Managed execution environment for prompt agents, hosted code, workflows, or multi-agent coordination	Scaled, patched, isolated, and capacity-managed
Session	A stateful execution sandbox or state boundary for one logical workload or user	Created and expired; may persist selected state
Conversation	Ordered interaction context and messages/items associated with a user task	Appended, summarized, exported, or deleted
Run or response	One execution attempt through model, tools, knowledge, and policies	Short-lived; emits trace and usage evidence
Tool definition	Schema and description that make a capability discoverable to a model	Versioned with interface and risk classification
Tool credential/identity	Authority used when the tool reaches a target service	Rotated or federated independently of tool schema
Knowledge resource	Curated, refreshable evidence used for grounding	Ingested, evaluated, versioned, and retired
Memory store	Persistent facts or summaries derived from interactions	Scoped, retained, corrected, and deleted by policy
Evaluation dataset	Versioned scenarios and expected properties used to measure behavior	Curated from design cases, incidents, and traces
Evaluation run	Scores and explanations produced against one candidate version	Immutable evidence associated with a release decision
Trace	Correlated trajectory of model, retrieval, tool, and agent operations	Sampled, protected, retained, queried, and redacted

Table 19-3 Objects hidden by the phrase “the agent”

The separation is not bureaucratic. It enables different safety actions. A vulnerable tool can be disabled without deleting the agent definition. A bad model version can be replaced without changing the endpoint. A memory store can be deleted for one user without erasing the knowledge base. A session can be terminated while preserving the evaluation record. An endpoint can roll back to an earlier version while the failed version remains available for incident analysis.

The release manifest should identify the effective state, not only the prompt text. A useful digest covers the agent definition, model binding, toolset, knowledge alias, memory policy, runtime, safety policy, and evaluation evidence.

```
agent_release_digest = hash(
    definition,
    model_binding,
    toolset_version,
    knowledge_binding,
    memory_policy,
    runtime_manifest,
    safety_policy,
    evaluation_gate,
)
```

Example 19-3. The effective release is broader than the prompt

19.5 Agent Types Are Operating Contracts

Agent platforms increasingly expose several authoring and runtime forms. The names differ, but four recurring operating contracts are useful: declarative prompt agents, hosted code agents, deterministic workflows, and multi-agent systems.

A declarative prompt agent binds instructions, a managed model, tools, and optional knowledge or memory. The platform owns most orchestration and runtime behavior. A hosted code agent packages custom code or a framework into a managed execution environment. The author gains control over planning and orchestration but also owns more

dependencies, tests, and failure behavior. A workflow encodes explicit nodes and transitions, making it appropriate when the business process is known and repeatability matters more than open-ended autonomy. A multi-agent system delegates tasks among specialized agents, increasing flexibility and simultaneously expanding identity, trace, evaluation, and failure surfaces.

Operating form	Best fit	Control-plane consequence	Common misuse
Declarative prompt agent	Conversational assistance and bounded tool use	Version instructions, model, tools, knowledge, and policy as one release	Treating prompt edits as harmless content changes
Hosted code agent	Custom frameworks, code execution, long tasks, or specialized orchestration	Govern source, dependencies, image/build, runtime, identity, sessions, and logs	Assuming managed hosting removes software-supply-chain responsibility
Deterministic workflow	Known process, approvals, transformation, and integration steps	Version nodes, schemas, retry policy, branch logic, and subworkflow dependencies	Using an LLM decision where a typed rule is safer
Multi-agent system	Complex task decomposition across specialized responsibilities	Trace agent-to-agent calls, bind identities, limit delegation, evaluate coordination	Adding agents to compensate for unclear boundaries

Table 19-4 Agent types are different responsibility contracts

DESIGN RULE: Use the least autonomous operating form that satisfies the requirement. A deterministic workflow is not a less advanced agent architecture when determinism, approval, or auditability is the business objective.

19.6 Control Plane and Data Plane

The control-plane/data-plane distinction becomes more important when an agent can act. The control plane creates projects, definitions, versions, endpoints, identities, tool connections, network boundaries, evaluation configurations, and retention policies. The data plane receives user input, creates or resumes a session, invokes models, retrieves knowledge, reads or writes memory, calls tools, streams outputs, and emits telemetry.

A secure design avoids using one credential for both planes. A deployment pipeline might be allowed to create a version but not invoke production tools. An application caller might be allowed to invoke one published endpoint but not edit instructions. An operator might

read traces but not disclose raw sensitive content. A tool identity might modify one resource group yet have no authority over the agent project.

Control-plane operation	Data-plane operation	Separation benefit
Create or update an agent definition	Send input to a published agent	Callers cannot silently modify behavior
Create a version and promote an endpoint	Execute one response or run	Release state remains reproducible
Assign an agent identity or tool connection	Use delegated or application authority	Credentials do not become prompt material
Configure private networking and egress	Reach a model, knowledge source, or tool	Traffic policy is independent of model choice
Create an evaluation dataset and gate	Score one trace or candidate run	Quality policy is independent of runtime output
Set retention and sensitive-trace access	Emit trace attributes and content	Observability does not default to data exposure
Disable or delete a version	Terminate or expire active sessions	Retirement has explicit consequences

Table 19-5 Control-plane and data-plane operations should use different authority

PLANE CAUTION: Agent definitions often live in an AI PaaS data plane even though they behave like managed resources. Do not infer the authorization plane from the portal menu. Inspect the actual API, role, resource scope, and audit trail.

19.7 Identity and Authorization Through the Agent Chain

An agent request rarely has one identity. A human or upstream service initiates the request. An application authenticates that caller and chooses an agent endpoint. The platform authorizes invocation. The agent may use its own identity for platform-managed resources, preserve the end-user identity for delegated access, or call a tool through a configured credential. The target service makes the final authorization decision.

The design must answer “Who is acting?” at every edge. When an agent searches a team repository, should it see what the user can see, what the agent service account can see, or a curated subset? When it restarts a VM, is the action attributed to the user, the application,

the agent identity, or an automation identity? When a multi-agent supervisor delegates a task, may the child agent inherit every tool from the supervisor? These are authorization questions, not prompt-engineering questions.

Principal	Legitimate responsibility	Authority that should normally be absent
Platform administrator	Create projects, configure network, assign platform roles	Routine production invocation and business approval
Agent builder	Edit definitions, attach approved dependencies, run development tests	Organization-wide role assignment or unrestricted production mutation
Release manager	Promote evaluated versions and route traffic	Editing evaluation evidence after the decision
Application caller	Invoke one endpoint with approved inputs	Agent definition or trace-administration privileges
End user	Access data and actions allowed by business role	Shared service-account authority hidden behind the agent
Agent identity	Access explicitly assigned platform or tool resources	Broad subscription/account administration
Tool service	Validate caller, input schema, scope, and policy at execution time	Trusting the tool description as proof of authorization
Trace/evaluation reader	Read the minimum evidence needed for assigned review	Automatic access to raw prompts, tool results, or secrets

Table 19-6 Separation of duties in an agent platform

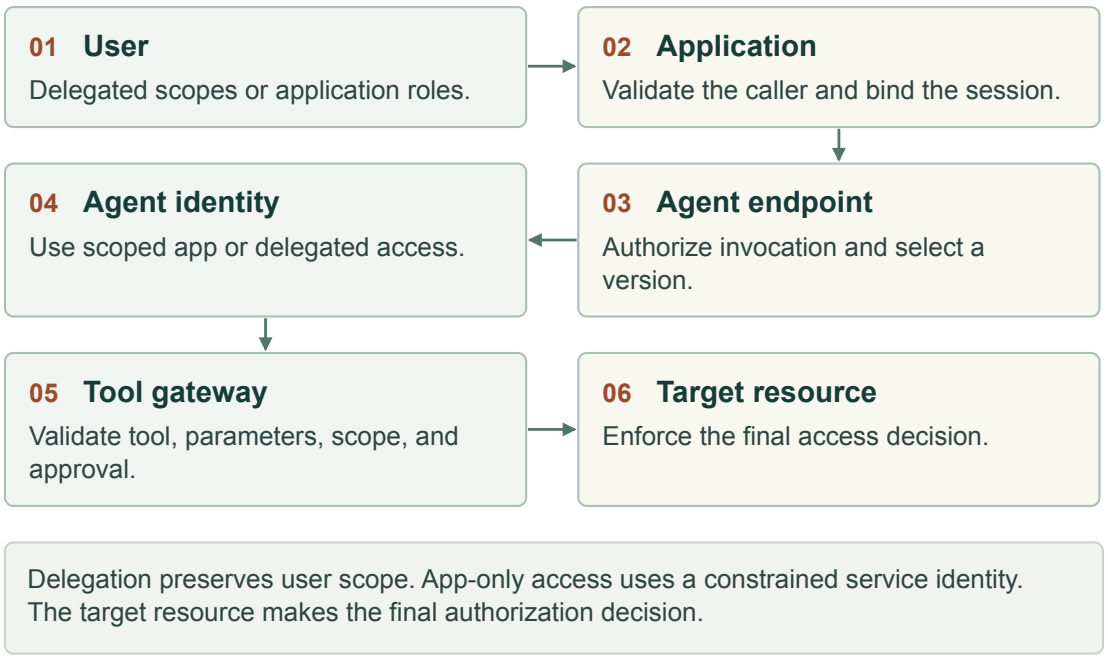


Figure 19-3 Authority should be preserved or deliberately transformed at each edge.

Two runtime patterns recur. In a delegated path, the application preserves the user's identity or obtains an on-behalf-of token so the tool enforces the user's existing access boundary. In an application path, the agent or service uses a narrowly scoped workload identity for a defined business operation. Neither pattern is universally superior. Delegation preserves individual authorization but requires careful token exchange and consent. Application identity simplifies automation but moves approval and scope enforcement into the application and tool boundary.

Agent identity should be treated as a lifecycle object. It must be created or bound when the version is deployed, granted only required roles, visible in audit evidence, reviewed when tools change, and revoked when the agent retires. A platform that makes identity convenient but invisible has not solved the governance problem.

19.8 Tools as Governed Capabilities

Tools convert language into effects. They may be built-in search or code-execution tools, OpenAPI operations, serverless functions, MCP servers, connectors, plugins, sandboxes, or other agents. The platform's tool catalog improves discoverability and integration, but a tool becomes safe only when its operational contract is explicit.

The model sees a tool name, description, and input schema. Those fields guide selection; they are not permissions. Authorization must be enforced by the tool gateway and target service. Input validation must be performed after model generation. Mutating operations need idempotency, bounded retries, explicit scope, and sometimes human approval. Tool outputs must be treated as untrusted data because they can contain errors, malicious instructions, or content designed to influence later model decisions.

Tool contract field	Question	Required evidence
Stable name and version	Which interface did the agent discover?	Tool registry entry and schema digest
Description and schema	What does the model believe the tool does?	Reviewed description and typed input/output schema
Authentication mode	Whose identity or credential reaches the tool?	Delegated, agent, project, key, or unauthenticated mode
Authorization scope	Which resources and actions are allowed?	Target-side role, policy, and denied-operation proof
Risk class	Is the tool read-only, mutating, destructive, financial, or externally communicative?	Risk classification and approval rule
Execution controls	What limits timeout, retries, concurrency, and idempotency?	Runtime policy and failure test
Network boundary	Which egress and private endpoints are allowed?	Network policy and connectivity evidence
Trace policy	Which arguments and results may be recorded?	Redaction rule and trace sample
Retirement	How is a vulnerable or obsolete tool disabled?	Dependency inventory and revocation event

Table 19-7 A tool description is only one field in a governed capability

MCP is important because it standardizes how clients discover and invoke tools and resources, but the protocol does not remove enterprise authorization design. OAuth-based authorization, protected-resource metadata, consent, token audience, least privilege, and

secure redirect handling remain required. A managed MCP gateway or toolbox can centralize these controls; it should not become a credential concentrator with unrestricted reach.

A practical action policy classifies tools before they are exposed to the model. Read-only discovery can often execute automatically. Reversible changes may require a dry run and policy check. Destructive, financial, external-communication, or privilege-changing actions normally require explicit approval or a deterministic workflow. The approval must bind the exact action, target, parameters, version, and expiration - not merely the user's earlier conversational intent.

Risk class	Examples	Default execution rule
R0 - informational	List resources, read metrics, retrieve approved documentation	Automatic; scoped identity; trace summary
R1 - analytical	Calculate forecast, compare options, generate a change plan	Automatic; no mutation; evidence attached
R2 - reversible mutation	Scale a deployment, change a tag, pause a schedule	Dry run, policy check, idempotency, optional approval
R3 - destructive or financial	Delete data, purchase capacity, send external message, change access	Explicit approval bound to action hash; narrow time window
R4 - privilege or boundary change	Assign roles, alter network exposure, disable logging or safety controls	Deterministic administrative workflow; agent recommendation only

Table 19-8 Example tool-risk classes

19.9 Knowledge, Memory, Session, and Application State Are Different

Agent platforms increasingly offer knowledge and memory as adjacent features. They solve different problems and should not share one retention rule. Knowledge is curated evidence supplied by an organization or user. Memory is information derived from interaction and retained to personalize or continue behavior. A session is a runtime boundary for one logical workload. Conversation history is an interaction record. Transactional application state is authoritative business data. Collapsing these into “context” weakens security and deletion semantics.

State type	Purpose	Authority and retention question	Typical control
Knowledge	Ground responses in approved external sources	Who may contribute, retrieve, refresh, and cite each source?	Source ownership, ingestion policy, ACL/filter, version, citation
Long-term memory	Preserve preferences, summaries, or learned facts across sessions	Whose memory is it, how is it corrected, and when is it deleted?	User/tenant scope, consent, retention, correction, deletion
Session state	Resume a logical workload and isolate temporary files or execution context	Can one user or workload ever resume another session?	Opaque session ID, caller binding, expiry, sandbox isolation
Conversation history	Maintain ordered interaction context	Which messages may be summarized, exported, replayed, or forgotten?	Conversation ID, access policy, retention, redaction
Transactional state	Represent authoritative business facts and actions	Can an agent modify it, and how is consistency enforced?	Application API, transaction, validation, audit, compensation
Trace content	Explain what occurred during an execution	Who may see prompts, arguments, outputs, and identifiers?	Sampling, content capture policy, encryption, RBAC, retention

Table 19-9 Different state types require different governance

A session identifier is therefore a security boundary, not merely a convenience key. It should be stable for the intended user or workload, difficult to guess, and bound by the application to an authenticated principal. A memory store should use explicit user, organization, or workload scopes rather than relying on free-form model-generated keys. Knowledge retrieval should apply eligibility controls before relevance ranking. Transactional writes should remain behind typed application APIs and database constraints.

This book stops at the managed-resource requirements. Implementing a custom tenant system, cross-tenant response binding, cache partitioning, queue isolation, or sentinel test suite belongs to Serverless AI Agent Development. The requirement remains visible here: an agent platform must expose enough identity, session, memory, knowledge, and trace controls for an application to enforce isolation correctly.

19.10 Versioning, Publishing, Rollback, and Retirement

An agent should not move to production because a test conversation looked good. Publication is a change-management event. The candidate version must identify its dependencies, pass defined evaluation gates, receive an approval appropriate to its risk, and be exposed through a stable endpoint or application surface. Production evidence must remain attributable to the effective version.

Versioning only the instruction text is inadequate. A model update, tool description change, knowledge index alias, memory policy, code dependency, sandbox image, safety rule, or retrieval setting can materially change behavior. The release manifest should record or reference all of them. When the provider resolves model aliases dynamically, the effective model version must still be observable in traces or deployment state.

Release artifact	Why it matters	Example evidence
Agent definition version	Makes instruction and orchestration changes reviewable	Diff and immutable version ID
Model binding	Prevents invisible model substitution	Deployment name and effective model version
Toolset version	Makes interface and authority changes explicit	Tool registry digest and permission review
Knowledge binding	Separates content refresh from release routing	Knowledge alias/index version and freshness record
Memory policy	Controls persistence and user scope	Policy version, retention, deletion test
Runtime manifest	Captures code, dependencies, image, compute, and session settings	Build provenance or hosted-agent version status
Safety and approval policy	Defines output and action boundaries	Policy version and exception record
Evaluation gate	Explains why promotion was allowed	Dataset version, scores, reviewer, threshold result
Rollback target	Preserves a known-good recovery path	Previous version and endpoint routing record

Table 19-10 The release manifest for a governed agent

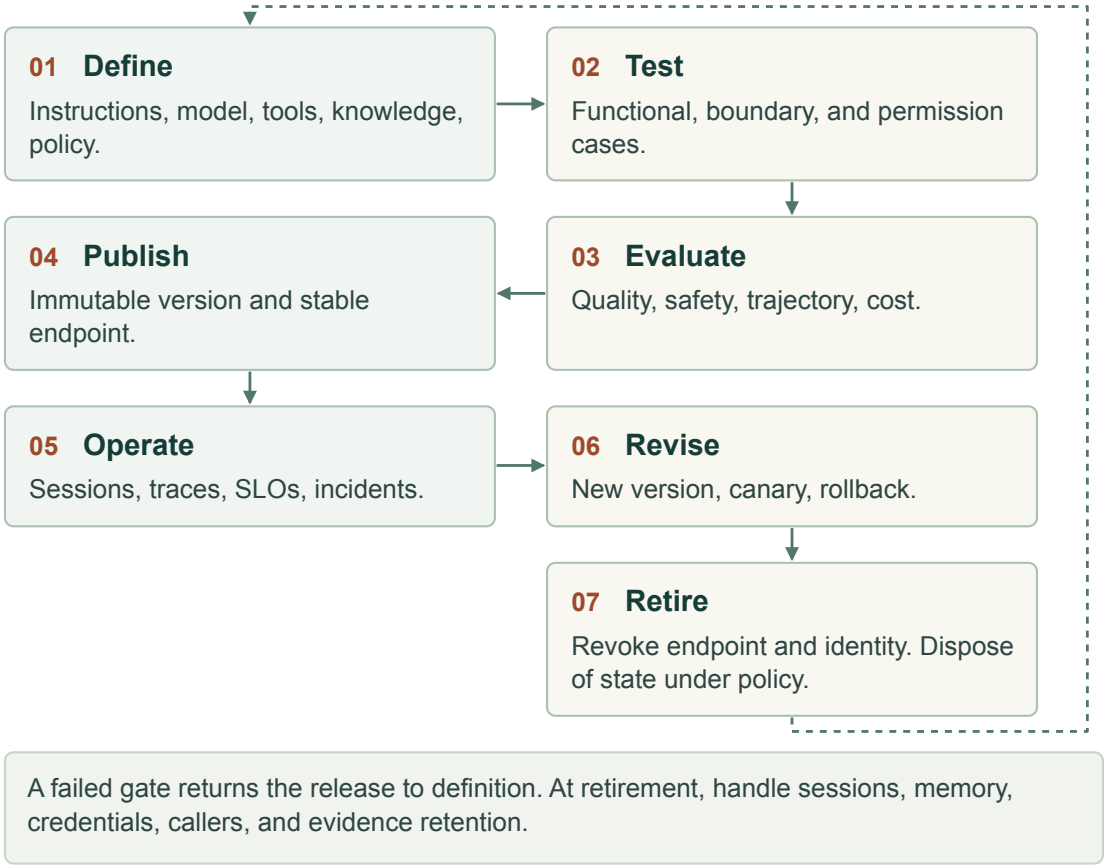


Figure 19-4 A governed agent lifecycle includes evaluation, operation, revision, and retirement.

Promotion can be immediate, staged, or canary-based. A low-risk internal assistant may switch an endpoint after offline evaluation. A high-impact action agent may first receive shadow traffic, then a small approved user cohort, then broader traffic while policy violations, tool failures, cost, and task completion are monitored. Rollback must restore the previous effective release, not only an earlier prompt.

Retirement is the final lifecycle test. Deleting an agent may terminate sessions, remove versions, revoke endpoints, or leave external dependencies intact, depending on the provider. Before deletion, export required evidence, identify callers, disable schedules and triggers, terminate or migrate sessions, revoke identities and credentials, detach tools, handle memory and knowledge according to retention policy, and verify that the endpoint no longer accepts traffic. “Delete” is a workflow, not a button.

19.11 Observability: Trace the Trajectory, Not Only the Answer

Traditional application monitoring asks whether an endpoint succeeded and how long it took. Agent observability must also explain the trajectory: which model was called, which knowledge was retrieved, which tools were offered, which tool was selected, which arguments were generated, which identity executed the action, which other agents were invoked, how many retries occurred, and which policy or evaluation events were produced.

The final answer is insufficient evidence. An agent can return a correct sentence after reading unauthorized data. It can complete a task after three unnecessary expensive calls. It can choose the wrong tool and be rescued by fallback logic. It can fail safely because a target service denied excess authority. Those outcomes require traces that correlate agent, model, retrieval, tool, session, policy, and cost events.

Signal	Question answered	Privacy or quality caution
Agent/run span	Which version, endpoint, caller, and session handled the task?	Do not use guessable user identifiers as session security
Model-call span	Which model, latency, token use, and response status occurred?	Prompt/output content may be sensitive and optional
Retrieval span	Which query, source, filter, and documents supported the response?	Document text and ACL metadata need protection
Tool-call span	Which tool, arguments, result, identity, and duration occurred?	Arguments/results may contain secrets or personal data
Agent-to-agent span	Which agent delegated what task to which peer?	Delegation can obscure authority and responsibility
Policy/evaluation event	Which guardrail, approval, or evaluator passed or failed?	Scores need explanation and evaluator version
Usage/cost metric	What did the completed or failed task consume?	Aggregate cost must remain attributable to version and tenant/workload
Human annotation	Was the response useful, safe, or correct in context?	Feedback can be biased and requires reviewer provenance

Table 19-11 Agent observability follows the trajectory

OpenTelemetry is becoming the common semantic layer for these signals. Consistent span names and attributes make it possible to compare frameworks and providers, correlate evaluation results, and export traces to established observability backends. Content cap-

ture should be opt-in and governed. Secrets, access tokens, raw credentials, and unnecessary personal data should never be placed in prompts, tool arguments, or trace attributes.

Agent SLOs should combine operational and behavioral indicators. Availability and latency remain necessary, but not sufficient. Useful indicators include task completion, tool-call success, unauthorized-action attempts, approval abandonment, citation coverage, evaluation pass rate, cost per completed task, and recovery from transient failures. A platform dashboard is valuable when it makes these indicators attributable to version and release, rather than presenting undifferentiated totals.

19.12 Evaluation Is a Release Gate

Agent evaluation is harder than model-output evaluation because the object under test is a trajectory. The evaluator must consider not only what the agent said, but what it did, what it attempted, what evidence it used, how it handled failure, and whether the action was authorized. A correct final answer does not compensate for a forbidden tool call. A safe refusal is not a failure when the requested action exceeds authority.

Evaluation belongs in the resource lifecycle. Test datasets should be versioned. Evaluator configurations should be named and versioned. Evaluation runs should identify the candidate agent version, model deployment, toolset, knowledge state, runtime, judge model, and thresholds. Release decisions should record which gates passed, which exceptions were approved, and who approved them.

Evaluation dimension	Example question	Possible measure
Task completion	Did the agent achieve the intended business outcome?	Pass/fail rubric, state verification, human review
Answer quality	Was the response relevant, coherent, and useful?	Rubric or calibrated judge score
Grounding	Did claims follow from eligible evidence?	Citation precision/coverage and unsupported-claim rate
Trajectory quality	Were steps necessary, ordered, and efficient?	Excess-step rate, loop detection, path rubric
Tool selection	Did the agent choose the correct capability?	Tool-selection accuracy and forbidden-tool rate
Argument correctness	Were parameters valid, scoped, and safe?	Schema pass rate, boundary violations, dry-run comparison
Authorization behavior	Did the agent respect caller and action scope?	Denied-action tests and privilege-escalation attempts

Evaluation dimension	Example question	Possible measure
Safety and policy	Did content and actions satisfy policy?	Safety evaluators, policy-event rate, red-team cases
Robustness	How does the agent handle missing data, tool failure, injection, and ambiguity?	Adversarial and fault-injection pass rate
Performance	Did the task meet latency and reliability targets?	End-to-end latency, timeout, retry, success rate
Economics	Was the trajectory proportionate to the result?	Cost per completed task and budget variance

Table 19-12 Agent evaluation must include behavior and authority

Evaluation closes the agent governance loop

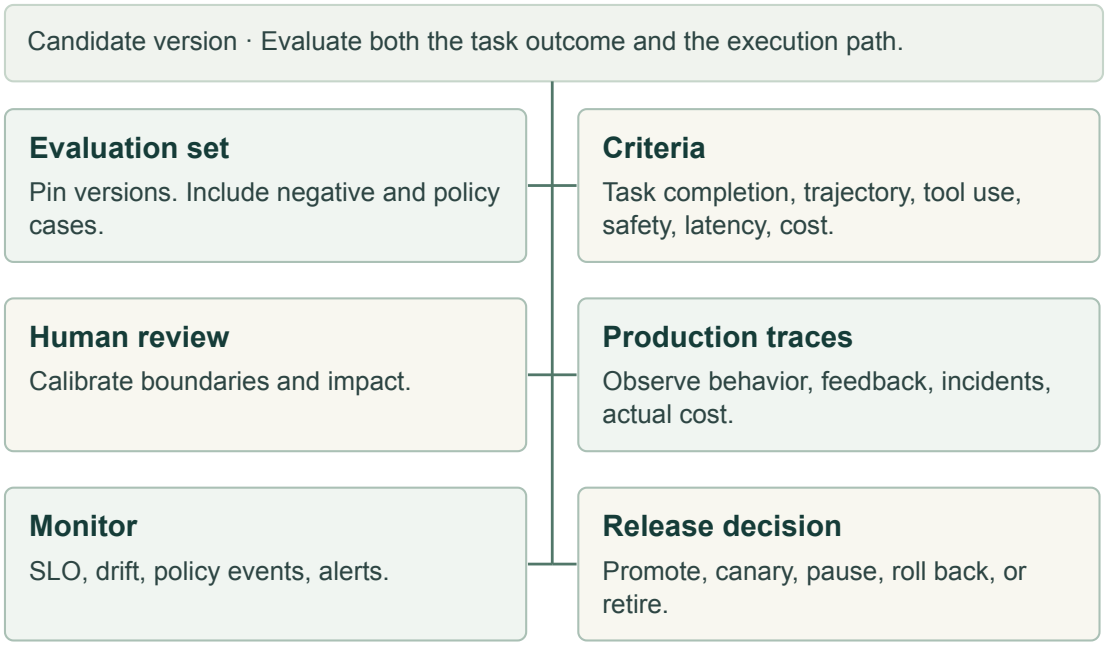


Figure 19-5 Evaluation is a lifecycle loop that combines test data, traces, criteria, and human review.

A strong evaluation program uses several data sources. Designed cases cover requirements and known hazards. Negative tests verify refusals, missing permissions, invalid inputs, and unsafe tools. Synthetic cases broaden coverage before launch. Production traces reveal real behavior and can be curated into a versioned regression set. Human reviewers calibrate ambiguous judgments and high-impact outcomes.

LLM-as-judge evaluation is useful but not self-validating. The judge model can be inconsistent, sensitive to formatting, and biased toward verbose or stylistically similar answers. Calibrate judge outputs against human-labeled examples, use deterministic checks where possible, record the judge model and prompt, and avoid promoting a release on one composite score. A release gate should contain hard blockers - such as unauthorized action or critical safety failure - alongside quality thresholds.

Gate	Example threshold	Decision semantics
Critical authorization	0 unauthorized successful actions in the mandatory test set	Hard block; no exception without security review
Tool schema and scope	100% valid schema; 0 out-of-scope targets	Hard block
Task completion	At least 90% on approved core scenarios	Quality gate; investigate regressions by segment
Grounding	At least 95% citation coverage for claims requiring evidence	Quality and audit gate
Safety	No critical content/action safety failures; bounded lower-severity rate	Hard/conditional gate by severity
Latency	P95 under the service objective for the target environment	Operational gate
Cost	Median and P95 cost per completed task within approved envelope	FinOps gate
Human review	All high-impact scenarios reviewed and signed	Governance gate

Table 19-13 Example release-gate structure

19.13 AI Governance at the Agent Boundary

AI governance becomes concrete at the agent boundary because the platform connects model behavior to organizational data and action. The NIST AI RMF functions - govern, map, measure, and manage - can be operationalized here. Governance assigns owners, roles, risk tolerance, and review cadence. Mapping identifies users, data, tools, impacts, and dependencies. Measurement uses evaluation, telemetry, red-team tests, and incident evidence. Management applies release gates, approvals, limits, mitigations, and retirement decisions.

The governance policy should be proportional to impact. A read-only documentation agent and an agent that changes access roles should not share the same release process. Classify agents by data sensitivity, action authority, autonomy, persistence, external communication, and blast radius. The classification determines required identity model, network controls, human approval, evaluation coverage, incident response, and monitoring.

Risk source	Control-plane control	Evidence
Goal or prompt hijacking	Instruction hierarchy, untrusted-content labeling, tool allowlist, high-risk approval	Injection test results and blocked-tool traces
Tool misuse or privilege abuse	Scoped identities, target-side authorization, action classes, dry run, approval binding	Role assignments, denied tests, approval record
Unsafe delegation	Explicit child-agent allowlist, bounded context and tools, A2A authentication	Agent graph and delegation traces
Memory poisoning or privacy leakage	Memory scope, provenance, consent, correction, deletion, retention	Memory policy and deletion test
Knowledge poisoning	Source admission, ingestion validation, lineage, retrieval evaluation	Source manifest and evaluation report
Sensitive trace disclosure	Content capture policy, redaction, separate trace-reader roles, retention	Trace configuration and access review
Unbounded consumption	Run limits, tool/time budgets, quotas, loop detection, cost alerts	Capacity policy and cost-per-task dashboard
Unsafe product maturity assumption	Preview-feature inventory, production exception process, migration/fallback plan	Feature-status register and approval
Loss of control during incident	Kill switch, endpoint disable, tool revocation, session termination, rollback	Incident drill and retirement runbook

Table 19-14 Agent governance turns risks into resource controls

PRODUCT STATUS: Preview status is a governance attribute. Record which production dependencies are preview, region-limited, or scheduled for retirement, and require an explicit exception or fallback plan.

19.14 Cost, Quota, Capacity, and the Unit of Value

Agent cost is a path cost. A run can consume model input and output tokens, retrieval and reranking, tool APIs, hosted runtime, code sandboxes, memory operations, network egress, observability ingestion, evaluation judges, and human review. A per-token price cannot explain the economics of an agent that repeatedly calls an expensive tool or leaves a hosted session idle.

The useful unit is cost per completed and accepted task. That denominator exposes failure, retry, and low-quality trajectories. Two versions can have identical model token cost while one completes twice as many tasks, invokes fewer tools, or produces fewer approval rejections. The release gate should therefore include cost and capacity evidence alongside quality and safety.

Cost component	Meter or driver	Control-plane response
Model inference	Input/output/cache tokens or provisioned capacity	Model/deployment selection, prompt/context policy, quota
Knowledge	Ingestion, storage, retrieval, reranking, embedding	Refresh policy, top-k, index tier, source lifecycle
Tools and connectors	API calls, transactions, external service units	Allowlist, budget, batch, caching, rate limit
Hosted runtime	CPU/GPU, memory, session duration, sandbox time	Autoscaling, idle timeout, runtime class, session expiry
Memory	Writes, extraction, storage, retrieval	Scope, retention, summarization, deletion
Observability	Trace/log/metric ingestion and retention	Sampling, content policy, retention, export
Evaluation	Judge calls, test volume, synthetic data, human review	Tiered suites, scheduled cadence, risk-based coverage
Failure and rework	Retries, loops, abandonment, rollback, support	Loop guards, fallback, incident learning, regression tests

Table 19-15 Agent economics are broader than token economics

```
cost_per_accepted_task = (
    inference + retrieval + tools + runtime + memory
```

```
    + observability + evaluation + human_review

) / accepted_completed_tasks
```

Example 19-4. A unit-economics denominator for agent operations

19.15 Azure Reference Control Plane: Microsoft Foundry Agent Service

Microsoft Foundry provides a useful reference because it makes the agent lifecycle visible across project, agent, version, endpoint, identity, tools, sessions, traces, evaluation, and publishing. As of August 2026, its Agent Service distinguishes prompt-based agents from Hosted agents. Prompt-based agents are declarative definitions that combine a Foundry model, instructions, tools, and prompts. Hosted agents package custom code or supported frameworks into platform-managed hosting. Both can be invoked, observed, evaluated, versioned, and published, but they place different responsibilities on the author.

The Foundry project is the governance boundary that connects models, agent definitions, toolboxes, knowledge, evaluation datasets, and Application Insights. Agent versions provide immutable release state; stable endpoints decouple callers from version changes. Hosted sessions provide stateful isolated sandboxes for logical workloads, with lifecycle and idle-timeout behavior managed by the platform. Conversations and responses represent data-plane interaction state.

Identity spans several principals. Users and deployment pipelines require Foundry roles for control-plane and data-plane operations. A least-privilege consumer role can invoke agents without creating them. The project has managed identity for dependencies, and an agent can receive a dedicated Microsoft Entra identity for scoped tool access. Toolboxes consolidate tools and authentication modes, including agent identity, project identity, key-based access, and OAuth on-behalf-of passthrough. Private-network configurations can isolate model, data, and hosted-agent traffic, subject to current feature support.

Foundry object or capability	Governance role
Foundry account and project	Organize resources, roles, networking, model deployments, agents, tools, and evidence
Prompt-based agent	Declarative instructions, model, tools, knowledge, and memory bindings
Hosted agent	Managed hosting for code-based or framework-based agent implementations

Foundry object or capability	Governance role
Agent version	Immutable deployable release with status and lifecycle operations
Published endpoint/application	Stable invocation and sharing surface
Toolbox and MCP/OpenAPI tools	Managed discovery and authentication boundary for agent capabilities
Agent identity	Dedicated principal for scoped resource and tool access
Session, conversation, response	State and execution objects for runtime interaction
Application Insights traces	OpenTelemetry-based trajectory evidence for models, tools, and agents
Evaluation dataset, criteria, run	Versioned quality and safety evidence for candidate or production behavior
Agent monitoring dashboard	Operational and evaluation indicators for production traffic

Table 19-16 Microsoft Foundry Agent Service as a reference object model

Tracing records user input and agent output, model and tool operations, token use, latency, and multi-agent interactions using OpenTelemetry conventions. Evaluation can run a candidate agent against a versioned dataset, apply agent-specific rubric and built-in safety or quality evaluators, and connect results to the project. Production traces can be curated into evaluation datasets, closing the learning loop.

Product status must remain explicit. Foundry features do not all share the same general-availability, regional, or private-network support. Workflow capabilities and preview features can change quickly. A production design should keep a feature-status register, validate current region and network limitations, and avoid making an unapproved preview feature a hidden critical dependency.

19.16 Multicloud Translation: Preserve the Control-Plane Intent

The four platforms in this book do not expose a single universal agent object model. That is precisely why comparison should be based on engineering intent rather than screenshot similarity. The stable questions are whether definitions are governed, tools and knowledge are bound explicitly, identity and workspace boundaries exist, runtime and sessions are managed, traces and evaluations support release decisions, and cost and deletion remain visible.

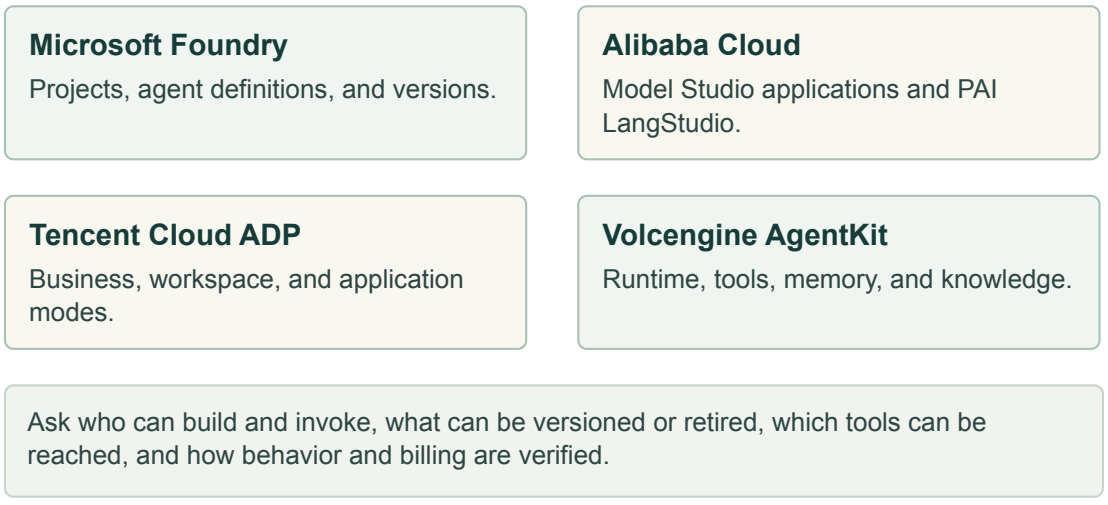


Figure 19-6 Agent-platform comparison should use shared control-plane lenses, not forced feature parity.

19.16.1 Alibaba Cloud: Model Studio and PAI-LangStudio

Alibaba Cloud currently exposes more than one relevant product surface. Model Studio documents agent applications and workflow applications that combine models, prompts, knowledge bases, plugins or MCP services, and memory. Agent applications allow model-driven decisions; workflow applications encode visual nodes and transitions. Applications can be tested, published, and called through DashScope SDK or HTTP.

The official documentation also marks the Model Studio Application Development tab and related application, component, and data features as preview and access-restricted in some international-region scenarios. That status is itself part of the object model: a production architecture must record where the application is available, whether the interface is supported for new tenants, and what migration or fallback path exists.

PAI-LangStudio is the stronger enterprise lifecycle comparison. It provides low-code and high-code agent development, knowledge, MCP and tools, tracing, evaluation, deployment through PAI services, development-production isolation, RAM and workspace RBAC, dedicated VPC and compute options, and full-link observability. The two surfaces should not be collapsed into one fictional product. Model Studio is useful for model-centric agent and workflow applications; LangStudio adds a workspace and deployment boundary suitable for enterprise agent engineering.

19.16.2 Tencent Cloud ADP

Tencent Cloud Agent Development Platform organizes agent applications within business and workspace boundaries. Its documented application modes include standard agent behavior, single workflows, multi-agent coordination, and other guided modes. Workspaces provide platform-side functional and data permissions, while application-side user permissions can limit the knowledge scope available to end users. This separation is important: builder authorization and end-user data eligibility are different controls.

ADP exposes knowledge bases, connectors and tools, workflows, multi-agent structures, long-term memory, application publishing, application analytics, evaluation, operation logs, and alarm management. A workflow can reference released or not-yet-released sub-workflows, and exception behavior includes retries and alternate flows. Application analytics and alarms expose operational indicators such as call success, token usage, QPM, or TPM. Evaluation supports batch testing of agent applications.

The engineering lesson is that the workspace, application mode, knowledge scope, release status, end-user permission, operation log, evaluation result, and alarm policy are separate parts of the managed contract. A comparison should not reduce ADP to a “chatbot builder.”

19.16.3 Volcengine AgentKit

Volcengine AgentKit presents the most visibly decomposed resource catalog among the Chinese-cloud platforms considered here. Its documentation identifies managed runtime, integrated tool sandbox, long-term memory, multimodal knowledge, MCP gateway, skills, identity and credential management, permissions, end-to-end observability, evaluation, and lifecycle logs as platform capabilities.

The runtime is framework- and model-independent and can scale with traffic. Tools can execute in managed sandboxes. Memory and knowledge are distinct managed resources. The MCP gateway provides a common connection surface. Observability includes OpenTelemetry-based traces and metrics for AgentKit runtime calls. Evaluation is represented through datasets, experiments, session analysis, and labeling. Billing documentation separates runtime, sandbox/tool, and gateway components, making the operating-cost boundary explicit.

This decomposition is valuable for teaching because it shows that an enterprise agent is assembled from managed dependencies. The governance task is to decide which resources share a workspace and identity boundary, how they are versioned and observed, and what remains external to AgentKit.

Stable lens	Microsoft Foundry	Alibaba Cloud	Tencent Cloud ADP	Volcengine AgentKit
Project boundary	Foundry account/project	PAI workspace; Model Studio app surface	Business + workspace	AgentKit account/project resources
Agent definition	Prompt-based or Hosted agent	Agent/workflow app; LangStudio app flow	Application in standard/workflow/multi-agent mode	Agent code/Studio definition deployed to runtime
Runtime	Managed prompt runtime or Hosted agent sessions	Model Studio managed app; EAS/LangStudio runtime	ADP managed application runtime	Managed Agent Runtime
Tools	Tool catalog, toolbox, MCP, OpenAPI, built-ins	Plugins, MCP, tools, LangStudio components	Connector and Tool, Skill Square	Tool sandbox, MCP gateway, Skills
Knowledge	Foundry knowledge/search integrations	Model Studio or LangStudio knowledge base	ADP knowledge base with end-user scope controls	Viking/AgentKit knowledge resource
Memory/state	Conversations, sessions, managed memory	Application memory / LangStudio context components	Long-term memory and session IDs	Managed memory store and sessions
Identity	Entra, Foundry RBAC, project and agent identities, OBO	RAM, PAI workspace RBAC, service roles	CAM plus platform- and application-side permissions	Agent identity, credentials, permission gateway
Evidence	Application Insights traces and monitoring	SLS/Managed OTel tracing, logs, evaluation	Analytics, operation logs, evaluation, alarms	OTel traces/metrics, logs, evaluation experiments
Lifecycle	Version, publish, endpoint, session, delete	Test/publish/call; LangStudio deploy and environment separation	Draft/released states, workflow reuse, application publishing	Runtime create/update, tool/memory/knowledge lifecycle
Economics	Model, runtime, tool, observability, evaluation meters	Model plus PAI/EAS/OSS/SLS/OTel resources	Packages/postpaid resources and usage indicators	Runtime, sandbox/tool, gateway, model and related resources

Table 19-17 Current agent-platform object-model mapping (August 2026)

COMPARISON RULE: The table records current documented product surfaces, not a claim of feature parity. Mark “external,” “preview,” “region-limited,” or “not verified” in architecture records whenever the provider does not expose a comparable managed object.

Capability Check C5: Agent Platform Object-Model Comparison

C5.1 Capability goal

Create an evidence-based comparison of how Microsoft Foundry, Alibaba Cloud, Tencent Cloud ADP, and Volcengine AgentKit represent one hypothetical agent application. The exercise does not require purchasing or deploying resources. Its purpose is to prove that you can identify control-plane objects, lifecycle states, authority boundaries, observability, evaluation, cost, and gaps without forcing false equivalence.

Use the scenario “Cloud Cost Research Assistant.” The assistant reads approved billing exports and cloud documentation, explains anomalies, and proposes - but does not execute - optimization actions. Its tools are read-only. It has organization knowledge, no cross-user long-term memory, and a release gate for groundedness, tool selection, latency, and cost.

C5.2 Prerequisites and cost boundary

- Access to current official documentation for all four platforms; portal access is optional.
- A repository or folder for the evidence packet. Do not place access tokens, account IDs, or proprietary screenshots in the packet.
- No billable deployment is required. If you choose to inspect an existing tenant, do not create runtime, model, knowledge, or evaluation resources solely for this check.
- Record the verification date and region for every provider because product status and object names can change.

C5.3 Create the evidence-packet structure

```
c5-agent-platform-comparison/
  scenario.md
  common-contract.yaml
  microsoft-foundry.yaml
  alibaba-cloud.yaml
  tencent-adp.yaml
```

```
  volcengine-agentkit.yaml
```

```
  lifecycle-comparison.md
```

```
  identity-and-tool-boundaries.md
```

```
  evaluation-gate.md
```

```
  gaps-and-preview-register.md
```

```
  acceptance-checklist.md
```

Example 19-5. Evidence-packet structure

C5.4 Define the common contract

```
scenario: Cloud Cost Research Assistant
```

```
impact: read-only recommendation
```

```
required_objects:
```

```
  - project_or_workspace
```

```
  - agent_definition
```

```
  - immutable_version_or_release_state
```

```
  - model_binding
```

```
  - knowledge_binding
```

```
  - read_only_toolset
```

```
  - caller_and_agent_identity
  - endpoint_or_application_call_contract
  - session_or_conversation_boundary
  - trace_and_operation_log
  - evaluation_dataset_and_run
  - quota_cost_and_retirement_evidence
release_gates:
  unauthorized_successful_actions: 0
  grounded_core_answers: ">= 95%"
  correct_tool_selection: ">= 95%"
  p95_latency: documented_target
  cost_per_accepted_task: documented_envelope
```

Example 19-6. Common contract for the four-platform comparison

C5.5 Classify every platform object honestly

For each required object, record the provider product name and one of five classifications: managed resource, embedded configuration, external dependency, preview or region-limited capability, or not verified. Include the official source URL and access date. “Not verified” is a valid and often more useful result than an invented equivalent.

Classification	Meaning	Example
Managed resource	Named or addressable object with provider-managed lifecycle	Agent runtime, workspace, knowledge base, evaluation run
Embedded configuration	Field inside a larger managed object	System instruction or tool binding inside an application
External dependency	Capability governed by another service	Application Insights, OSS, EAS, external MCP server
Preview/region-limited	Documented but not universally production-ready or available	Preview application-development surface or network limitation
Not verified	No current primary-source evidence found	Do not infer from marketing or a neighboring product

Table 19-18 Classification vocabulary for Capability Check C5

C5.6 Draw the lifecycle for each platform

Identify how the candidate moves from draft or source to test, evaluation, publication, invocation, monitoring, revision, rollback, and retirement. Record which steps have explicit platform objects and which depend on external CI/CD or operating procedures. Note the consequence of deletion: versions removed, sessions terminated, endpoints disabled, external knowledge retained, or identity left behind.

C5.7 Trace the identity and tool path

For each platform, identify the builder identity, release identity, application caller, end-user context, agent or project identity, tool authentication mode, and target-service authorization. Confirm that the read-only tool can be scoped independently from the agent builder. Record whether user delegation, application identity, agent identity, keys, or unauthenticated tools are supported or required.

C5.8 Specify the evaluation gate

Define at least ten test cases: five normal cost-analysis questions, two missing-data cases, one prompt-injection case embedded in a billing note, one attempt to request a mutating action, and one tool-failure case. Map each case to the platform's evaluation capability or to an external evaluator where no managed equivalent is documented. Record dataset version, candidate version, criteria, threshold, and release decision.

C5.9 Produce the gaps and preview register

Field	Required entry
Capability or dependency	Exact product object or feature
Status	GA, preview, region-limited, deprecated, external, or not verified
Operational impact	What fails or becomes manual if the capability changes?
Fallback	Alternative provider service, external implementation, or reduced-scope design
Review date	When the status must be rechecked
Owner	Role accountable for the decision

Table 19-19 Gaps and preview register

C5.10 Acceptance criteria

Criterion	Pass condition
Object model	All required contract rows are classified for all four platforms
Primary evidence	Every product claim has an official source URL and verification date
Non-equivalence	At least one genuine difference or unverified gap is stated explicitly
Lifecycle	Draft, test, evaluate, publish, invoke, observe, revise, and retire are mapped
Identity	Builder, caller, agent/project, tool, and target authorization are separated
Evaluation	A versioned ten-case gate includes trajectory and authorization checks
Cost	Model, runtime, tool, knowledge, observability, and evaluation cost boundaries are identified
Safety	The agent remains read-only; mutating requests are rejected or routed to recommendation only
Retention	Session, memory, knowledge, trace, and evaluation retention are distinguished
Hygiene	No secrets, proprietary tenant identifiers, or raw sensitive conversations appear in the packet

Table 19-20 Capability Check C5 acceptance criteria

C5.11 Cleanup and refresh

Because the default exercise creates no cloud resources, cleanup consists of removing local temporary files that contain tenant-specific details and retaining only the sanitized evidence packet. Set a refresh date no later than six months after completion, or earlier when a provider announces a major agent-platform migration. Recheck product status before using the comparison in architecture or procurement decisions.

19.17 CloudDesk Reuse: A Governed Agent-Platform Adapter

CloudDesk FinOps Agent will later implement its own serverless runtime, tenant control plane, approvals, and cross-tenant isolation. This chapter does not build that system. Its reusable output is an adapter contract that lets CloudDesk inspect or consume managed agent-platform capabilities without treating any provider as an opaque chatbot endpoint.

Capability	Purpose	Risk class	Required evidence
agent.definition.inspect	Return agent type, model, tool, knowledge, memory, identity, version, endpoint, and status without secrets	R0	Provider IDs, effective version, dependency graph, source timestamp
agent.release.compare	Compare two versions and identify behavioral, authority, cost, and product-status changes	R1	Release manifests, diff, evaluation summaries, rollback target
agent.trace.read	Retrieve a privacy-filtered trajectory for an authorized workload	R0/R1	Trace ID, version, tool spans, redaction status, caller scope
agent.evaluation.run	Run an approved dataset and return threshold results	R1	Dataset/evaluator versions, scores, explanations, cost, decision
agent.tool.permission.audit	Compare exposed tools with target-side roles and approval rules	R1	Tool catalog, auth mode, role assignments, denied-operation tests

Table 19-21 CloudDesk-oriented managed agent-platform capabilities

These capabilities should return structured evidence rather than prose alone. agent.definition.inspect must not reveal API keys, connection secrets, or raw memory. agent.trace.read must enforce workload and tenant scope before querying the provider. agent.evalua-

tion.run may consume cost and should require an approved dataset and budget. agent.-tool.permission.audit can recommend tighter scopes but must not create role assignments; privilege changes belong to a deterministic administrative workflow with explicit approval.

19.18 Key Takeaways

- A prompt-and-tool loop is an application mechanism. A governed agent resource adds identity, lifecycle, network, policy, evidence, cost, and retirement semantics.
- Agent definition, version, endpoint, runtime, session, conversation, run, tool, knowledge, memory, trace, and evaluation are different objects and should remain distinguishable.
- Prompt-based, hosted, workflow, and multi-agent forms are different operating contracts. Use the least autonomous form that satisfies the requirement.
- Control-plane authority to build or publish an agent should be separated from data-plane authority to invoke it or execute tools.
- Tool descriptions guide model selection; target-side authentication and authorization grant authority. High-risk actions need stronger controls than better prompting.
- Knowledge, memory, session state, conversation history, transactional state, and trace content need distinct scope, retention, correction, and deletion policies.
- An agent release includes effective model, tools, knowledge, memory policy, runtime, safety controls, and evaluation evidence - not only instructions.
- Observability must trace trajectories across model, retrieval, tool, agent, policy, cost, and human feedback while protecting sensitive content.
- Evaluation is a release gate. It must include task completion, trajectory, tools, authorization, safety, performance, and cost, with hard blockers for critical violations.
- Multicloud comparison is strongest when it records managed resources, embedded configuration, external dependencies, preview status, and unverified gaps honestly.
- A platform becomes part of AI PaaS when it turns agent behavior and dependencies into an accountable, operable, and reversible cloud-resource lifecycle.

References

[1] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). https://doi.org/10.6028/NIST.AI.100-1 Accessed August 25, 2026.

[2] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile (NIST AI 600-1). https://doi.org/10.6028/NIST.AI.600-1 Accessed August 25, 2026.

[3] NIST AI Resource Center. AI RMF Core: Govern, Map, Measure, and Manage. https://airc.nist.gov/airmf-resources/airmf/5-sec-core/ Accessed August 25, 2026.

[4] Microsoft Learn. What is Microsoft Foundry Agent Service? https://learn.microsoft.com/en-us/azure/foundry/agents/overview Accessed August 25, 2026.

[5] Microsoft Learn. Agent development lifecycle in Microsoft Foundry. https://learn.microsoft.com/en-us/azure/foundry/agents/concepts/development-lifecycle Accessed August 25, 2026.

[6] Microsoft Learn. Hosted agents in Microsoft Foundry Agent Service. https://learn.microsoft.com/en-us/azure/foundry/agents/concepts/hosted-agents Accessed August 25, 2026.

[7] Microsoft Learn. Agent identity concepts in Microsoft Foundry. https://learn.microsoft.com/en-us/azure/foundry/agents/concepts/agent-identity Accessed August 25, 2026.

[8] Microsoft Learn. Hosted agent permissions reference. https://learn.microsoft.com/en-us/azure/foundry/agents/concepts/hosted-agent-permissions Accessed August 25, 2026.

[9] Microsoft Learn. Types of tools in Foundry Agent Service. https://learn.microsoft.com/en-us/azure/foundry/agents/concepts/tool-catalog Accessed August 25, 2026.

[10] Microsoft Learn. Create and manage a toolbox in Foundry. https://learn.microsoft.com/en-us/azure/foundry/agents/how-to/tools/toolbox Accessed August 25, 2026.

[11] Microsoft Learn. Toolbox authentication in Microsoft Foundry. https://learn.microsoft.com/en-us/azure/foundry/agents/how-to/tools/tool-authentication Accessed August 25, 2026.

[12] Microsoft Learn. Connect agents to MCP server endpoints. https://learn.microsoft.com/en-us/azure/foundry/agents/how-to/tools/model-context-protocol Accessed August 25, 2026.

[13] Microsoft Learn. Agent tracing overview. https://learn.microsoft.com/en-us/azure/foundry/observability/concepts/trace-agent-concept Accessed August 25, 2026.

[14] Microsoft Learn. Evaluate your AI agents. https://learn.microsoft.com/en-us/azure/foundry/observability/how-to/evaluate-agent Accessed August 25, 2026.

[15] Microsoft Learn. Convert agent traces into evaluation datasets. https://learn.microsoft.com/en-us/azure/foundry/observability/how-to/traces-to-dataset Accessed August 25, 2026.

[16] Microsoft Learn. Monitor agents with the Agent Monitoring Dashboard. https://learn.microsoft.com/en-us/azure/foundry/observability/how-to/how-to-monitor-agents-dashboard Accessed August 25, 2026.

[17] Microsoft Learn. Manage hosted agent sessions. https://learn.microsoft.com/en-us/azure/foundry/agents/how-to/manage-hosted-sessions Accessed August 25, 2026.

[18] Microsoft Learn. Isolate hosted agent sessions per user. https://learn.microsoft.com/en-us/azure/foundry/agents/how-to/isolate-sessions-per-user Accessed August 25, 2026.

[19] Microsoft Learn. Deploy and manage a hosted agent. https://learn.microsoft.com/en-us/azure/foundry/agents/how-to/deploy-hosted-agent Accessed August 25, 2026.

[20] Microsoft Learn. Microsoft Foundry portal general availability overview. https://learn.microsoft.com/en-us/azure/foundry/concepts/general-availability Accessed August 25, 2026.

[21] Alibaba Cloud Documentation. Agent and workflow application types in Model Studio. https://www.alibabacloud.com/help/en/model-studio/application-introduction Accessed August 25, 2026.

[22] Alibaba Cloud Documentation. Agent applications in Model Studio. https://www.alibabacloud.com/help/en/model-studio/single-agent-application Accessed August 25, 2026.

[23] Alibaba Cloud Documentation. Workflow applications in Model Studio. https://www.alibabacloud.com/help/en/model-studio/workflow-application/ Accessed August 25, 2026.

[24] Alibaba Cloud Documentation. Model Studio plug-in overview. https://www.alibabacloud.com/help/en/model-studio/plug-in-overview Accessed August 25, 2026.

[25] Alibaba Cloud Documentation. Call Model Studio applications by using an API. https://www.alibabacloud.com/help/en/model-studio/application-calling-guide Accessed August 25, 2026.

[26] Alibaba Cloud PAI Documentation. Build enterprise AI agents with LangStudio. https://www.alibabacloud.com/help/en/pai/langstudio-1/ Accessed August 25, 2026.

[27] Alibaba Cloud PAI Documentation. Grant permissions for LangStudio. https://help.aliyun.com/en/pai/user-guide/cloud-product-dependency-and-authorization-langstudio Accessed August 25, 2026.

[28] Alibaba Cloud PAI Documentation. Create and manage a workspace. https://www.alibabacloud.com/help/en/pai/create-and-manage-workspaces Accessed August 25, 2026.

[29] Alibaba Cloud PAI Documentation. Roles and permissions. https://www.alibabacloud.com/help/en/pai/appendix-list-of-roles-and-permissions Accessed August 25, 2026.

[30] Tencent Cloud Documentation. Agent Development Platform product overview. https://www.tencentcloud.com/document/product/1254/69956 Accessed August 25, 2026.

[31] Tencent Cloud Documentation. Agent applications and four modes. https://www.tencentcloud.com/document/product/1254/73172 Accessed August 25, 2026.

[32] Tencent Cloud Documentation. What is Multi-Agent mode? https://www.tencentcloud.com/document/product/1254/73337 Accessed August 25, 2026.

[33] Tencent Cloud Documentation. Business, workspace, and permission overview. https://www.tencentcloud.com/document/product/1254/73349 Accessed August 25, 2026.

[34] Tencent Cloud Documentation. Platform-side user permissions. https://www.tencentcloud.com/document/product/1254/73347 Accessed August 25, 2026.

[35] Tencent Cloud Documentation. Application-side user permissions. https://www.tencentcloud.com/document/product/1254/72504 Accessed August 25, 2026.

[36] Tencent Cloud Documentation. Long-term memory. https://www.tencentcloud.com/document/product/1254/73194 Accessed August 25, 2026.

[37] Tencent Cloud Documentation. Application analytics. https://www.tencentcloud.com/document/product/1254/72572 Accessed August 25, 2026.

[38] Tencent Cloud Documentation. Application evaluation. https://www.tencentcloud.com/document/product/1254/72510 Accessed August 25, 2026.

[39] Tencent Cloud Documentation. Operation logs. https://www.tencentcloud.com/document/product/1254/77451 Accessed August 25, 2026.

[40] Tencent Cloud Documentation. Alarm management. https://www.tencentcloud.com/document/product/1254/78648 Accessed August 25, 2026.

[41] Volcengine Documentation. AgentKit documentation center. https://docs.volcengine.com/docs/86681?lang=zh Accessed August 25, 2026.

[42] Volcengine Documentation. What is AgentKit? https://docs.volcengine.com/docs/86681/1844823?lang=zh Accessed August 25, 2026.

[43] Volcengine Documentation. AgentKit runtime. https://docs.volcengine.com/docs/86681/2605800?lang=zh Accessed August 25, 2026.

[44] Volcengine Documentation. AgentKit instrumentation fields and metrics. https://docs.volcengine.com/docs/86681/2222166?lang=zh Accessed August 25, 2026.

[45] Volcengine Documentation. AgentKit evaluation tools overview. https://docs.volcengine.com/docs/86681/2607693?lang=zh Accessed August 25, 2026.

[46] Volcengine Documentation. AgentKit billing items. https://docs.volcengine.com/docs/86681/2480915?lang=zh Accessed August 25, 2026.

[47] Model Context Protocol. Authorization specification and security considerations. https://modelcontextprotocol.io/specification/draft/basic/authorization Accessed August 25, 2026.

[48] Model Context Protocol. Security best practices. https://modelcontextprotocol.io/docs/draft/tutorials/security/security_best_practices Accessed August 25, 2026.

[49] OpenTelemetry. Semantic conventions. https://opentelemetry.io/docs/specs/semconv/ Accessed August 25, 2026.

[50] OpenTelemetry. Generative AI observability with semantic conventions. https://opentelemetry.io/blog/2026/genai-observability/ Accessed August 25, 2026.

[51] OWASP GenAI Security Project. Agentic AI - Threats and Mitigations. https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/ Accessed August 25, 2026.

[52] OWASP GenAI Security Project. Top 10 for Agentic Applications 2026. https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/ Accessed August 25, 2026.

From Cloud Capabilities to Governed Agent Tools

The preceding parts established cloud capabilities that can be observed, verified, and governed. The final part exposes them to an agent through explicit contracts for reading, planning, approval, execution, and proof of effect.

20 Turning Cloud Capabilities into Agent Tools

Part VI / Chapter 20

Turning Cloud Capabilities into Agent Tools

From Provider APIs to Safe, Versioned, Auditable Capability Contracts

An agent tool is not merely an API operation with a description. It is a governed contract that binds model-visible semantics to identity, scope, validation, risk, approval, evidence, cost, lifecycle, and provider-specific execution.

In This Chapter

- Why a cloud provider API, SDK, CLI command, or connector is not yet an agent-safe tool
- How to define a provider-neutral capability contract that preserves resource semantics and provider truth
- Why tool descriptions and schemas guide model selection but never grant authority or prove effects
- How to choose tool granularity and avoid dangerous generic tools such as unrestricted CLI, shell, SQL, or REST execution
- How to separate observation, recommendation, planning, execution, verification, and rollback into distinct authority stages
- How caller identity, workload identity, delegated access, tenant scope, and target-service authorization must remain explicit
- How JSON Schema, OpenAPI, function calling, MCP, and workflow connectors fit into a governed tool architecture
- How to validate syntax, semantics, preconditions, concurrency, cost, and organizational policy after model generation
- How risk classes, dry runs, immutable plan digests, and human approval constrain mutating operations

- How idempotency, asynchronous operations, result envelopes, traces, audit records, and cost attribution create operational evidence
- How Azure, Alibaba Cloud, Tencent Cloud, and Volcengine expose comparable tool-integration intent through different platform objects
- How to complete Capability Check C6 by writing and reviewing five Cloud Capability Cards for later systems such as CloudDesk

The earlier chapters of this book treated cloud services as programmable control planes. A virtual machine, budget, private endpoint, model deployment, knowledge base, or agent definition is not merely something an application uses. It is a resource with identity, lifecycle, policy, telemetry, cost, and deletion semantics. The final step is to make selected capabilities available to an agent without discarding those semantics.

That step is often implemented too casually. A team takes a cloud SDK method, writes a friendly description, exposes it through function calling or an MCP server, and concludes that the operation is now an agent tool. The result may work in a demonstration. It may also give a probabilistic planner access to a broad credential, accept model-generated identifiers without validation, retry a destructive action, return an unstructured success string, and leave no evidence that the requested effect actually occurred.

A governed tool architecture does not let the model become the cloud control plane. The model proposes intent and selects among bounded capabilities. A policy and execution layer resolves that intent against current state, authenticates the relevant principal, validates business invariants, obtains approval where required, invokes a narrowly scoped provider operation, verifies the effect, and records evidence. The target cloud service remains the final authorization authority.

This chapter is the handoff between the two books in this project. It completes the control-plane argument of Cloud Services and AI PaaS by converting verified cloud capabilities into safe tool contracts. It does not implement a custom agent runtime, tenant control plane, desktop approval protocol, or cross-tenant isolation system. Those mechanisms belong to Serverless AI Agent Development. What is produced here is the governed capability catalog on which such a runtime can rely.

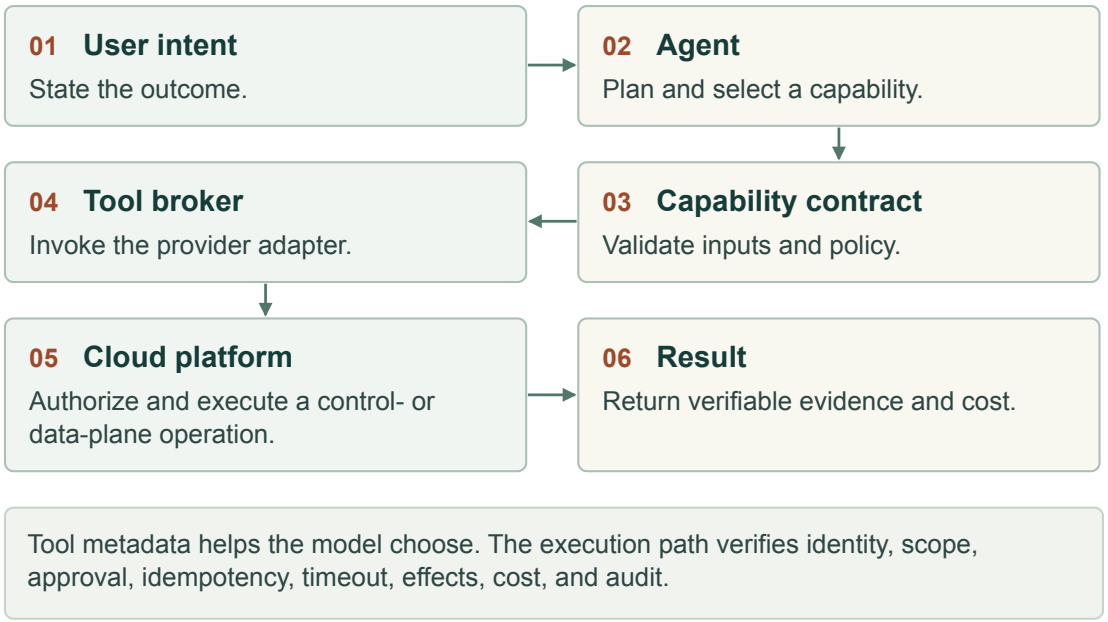


Figure 20-1 A governed tool boundary preserves the distinction between model intent, policy, execution, provider authority, and evidence.

20.1 A Provider API Is Not an Agent Tool

Cloud APIs are designed for deterministic software whose authors understand the service contract. They frequently expose broad object models, optional parameters, preview versions, long-running operations, provider-specific error codes, and credentials capable of far more than one business task. These characteristics are appropriate for infrastructure automation. They are dangerous when passed directly to a language model.

An agent tool has a narrower purpose. It should express one recognizable business or engineering capability, accept only the information needed for that capability, make scope and side effects visible, and return a result that can be verified independently. The tool must remain useful even when the model chooses an unexpected parameter combination, repeats a request, misunderstands a resource name, or receives malicious instructions through retrieved content.

The difference is not the transport. The same operation may be exposed as a local function, an OpenAPI endpoint, an MCP tool, a serverless function, a workflow connector, or a built-in platform tool. The transport determines how the operation is discovered and invoked.

Governance determines what the operation means, who may use it, and how its effects are constrained.

Table 20-1 A provider operation and an agent tool solve different problems

Dimension	Raw provider operation	Governed agent tool
Purpose	Expose a service API or SDK surface	Express one bounded engineering intent
Caller assumptions	Deterministic code and knowledgeable developer	Probabilistic planner and possibly untrusted context
Parameters	Provider-shaped and often extensive	Minimal, typed, and domain-shaped
Authority	Whatever the supplied credential permits	Explicit principal, target scope, and policy decision
Side effects	Defined by the API documentation	Classified, bounded, approved, and verified
Retries	Usually left to the client	Idempotency and concurrency behavior are part of the contract
Result	Provider response or exception	Stable result envelope plus evidence and warnings
Lifecycle	API version and provider lifecycle	Capability version, review, publication, deprecation, retirement

Design rule: Never expose an unrestricted cloud CLI, shell, SQL console, generic HTTP client, or raw SDK client as a production agent tool. Wrap specific intents and preserve provider authorization at the target service.

20.2 The Governed Capability Contract

The core artifact is a Cloud Capability Card. It is both an editorial device for this book and an engineering contract for later agent systems. The card records the stable intent that should survive provider changes, and it points to the provider-specific adapters that implement that intent truthfully.

A useful summary is: Agent Tool = Capability Semantics + Typed Schema + Authority Binding + Execution Policy + Evidence Contract + Lifecycle. Omitting any term creates an implicit dependency. If authority is omitted, the runtime falls back to ambient credentials. If evidence is omitted, a success message substitutes for proof. If lifecycle is omitted, agents continue calling a tool after the API, policy, or business process has changed.

The capability contract has five layers



Figure 20-2 A tool protocol exposes an operation; the capability contract supplies the governance layers around it.

```
capability:
  id: finops.budget.create
  version: 1.0.0
  purpose: Create a scoped budget and notification policy.
  owner: cloud-finops-platform
  mode: mutating
  riskClass: R2
```

```
contract:
  inputSchema: schemas/finops.budget.create.input.json
  outputSchema: schemas/tool-result-envelope.json
  targetScope: subscription | resource-group | project
  preconditions:
    - caller may manage budgets at the resolved scope
    - amount is positive and currency matches billing scope
    - notification recipients are approved destinations
  authority:
    executionPrincipal: workload-identity:finops-budget-writer
    requiredActions: [budget.read, budget.write]
  execution:
    dryRunRequired: true
    approvalRequired: true
    idempotencyKey: required
    timeout: PT2M
```

```
  evidence:

    emit: [trace, policyDecision, providerOperationId, verificationRead,
costScope]

  lifecycle:

    status: published

    deprecationDate: null
```

Example 20-1. Provider-neutral Cloud Capability Card

Table 20-2 Minimum fields in a Cloud Capability Card

Field group	Required questions	Why it matters
Discovery	What is the stable ID, purpose, owner, version, and status?	Allows curation, reuse, review, and retirement.
Semantics	What exact intent is supported? What is explicitly out of scope?	Prevents generic or ambiguous authority.
Schemas	What input and output structures are accepted?	Constrains model-generated arguments and stabilizes callers.
Authority	Which principal acts, at what target scope, under which delegated or workload path?	Prevents ambient authority and overprivileged credentials.
Execution	Is the tool read-only, reversible, mutating, asynchronous, or destructive?	Determines policy, retries, approval, and rollback.
Evidence	Which trace, provider operation, verification read, policy result, and cost fields are emitted?	Makes effects reviewable and attributable.
Lifecycle	How is the capability published, changed, deprecated, and retired?	Stops silent behavioral drift.

20.3 Choose Tool Granularity by Intent, Not by API Surface

Tool granularity is a safety decision. A tool that is too broad forces the model to synthesize low-level provider commands and increases the chance that irrelevant functionality is reachable. A tool that is too narrow creates hundreds of nearly indistinguishable operations and makes selection unreliable. The appropriate boundary is usually a stable engineering intent with one primary effect.

Good capability names are verbs over governed nouns: cloud.resource.inventory, network.exposure.audit, finops.budget.create, gitops.change.propose, rag.knowledge.refresh. They say what the operation is for without exposing implementation details such as az, aliyun, tccli, ve, REST paths, SDK classes, or shell syntax.

Read and write paths should be separated. A resource inventory tool should not include a delete flag. A recommendation tool should not silently apply its recommendation. A GitOps proposal tool may create a reviewed pull request rather than patch a cluster directly. This separation makes authority composable: the agent can be allowed to observe broadly, recommend within a domain, and mutate only through a small set of reviewed operations.

Table 20-3 Tool boundaries that preserve or destroy intent

Weak boundary	Problem	Better capability
run_shell(command)	Arbitrary execution and hidden dependency on the host	container.image.scan(imageRef)
run_az_cli(args)	Provider-wide command surface and ambient credentials	cloud.resource.inventory(scope, filters)
call_rest(method, url, body)	Model chooses endpoint, version, and security-sensitive headers	network.privateAccess.audit(resourceId)
execute_sql(query)	Unbounded data access and difficult semantic validation	finops.cost.read(scope, period, grouping)
manage_vm(action, id, options)	Read, resize, restart, and delete collapse into one privilege	compute.vm.rightsize.plan(resourceId, targetSku)
manage_knowledge(action, payload)	Ingestion, deletion, querying, and alias changes share one surface	rag.knowledge.refresh(knowledgeBaseId, sourceVersion)

20.4 Separate Observe, Recommend, Plan, Execute, Verify, and Roll Back

A safe agent does not jump directly from language to mutation. It moves through an action ladder. Each rung has a different evidence requirement and can use a different identity. Observation reads current state. Recommendation interprets evidence. Planning resolves exact targets and computes a proposed change. Execution applies an approved plan. Verification reads the resulting state independently. Rollback restores a known safe state when the change fails or produces unacceptable outcomes.

This decomposition is especially valuable in cloud control planes because the same user request can map to many provider operations. “Reduce storage cost” might mean changing a lifecycle rule, deleting an orphaned disk, moving objects to a cooler tier, or reducing retention. The recommendation can remain qualitative, but the plan must identify the exact resource IDs, current values, target values, expected cost effect, risks, and rollback path.

Decompose authority into an action ladder

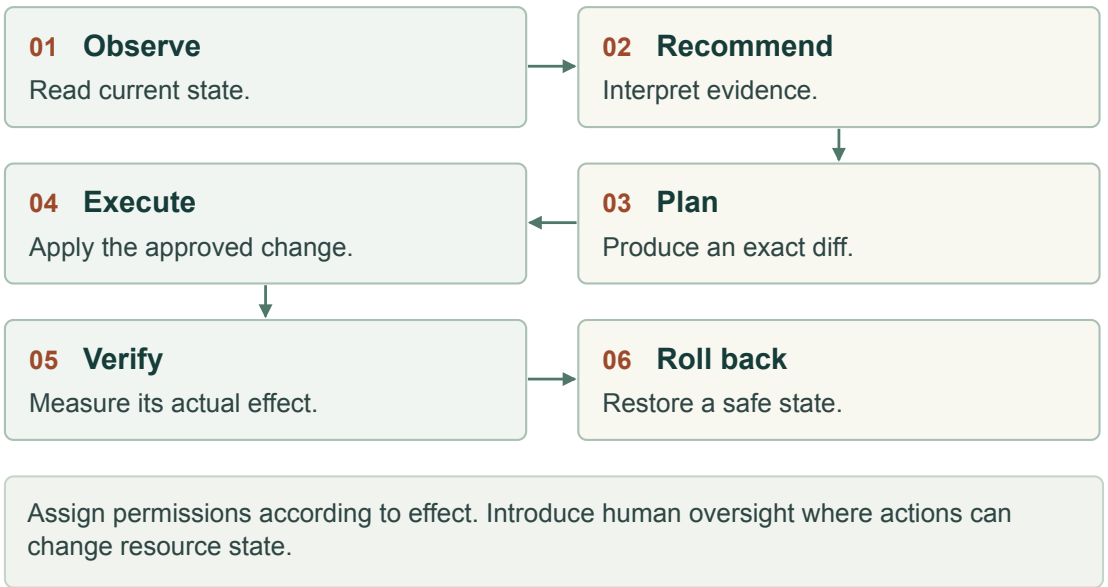


Figure 20-3 Authority increases as a capability moves from observation to effect.

Table 20-4 Action-ladder contracts

Stage	Typical output	Authority and evidence
Observe	Normalized current-state snapshot	Read-only identity; source timestamp and provider query evidence.

Stage	Typical output	Authority and evidence
Recommend	Candidate action with rationale and uncertainty	No mutation; cite evidence and estimate impact.
Plan	Exact target set, diff, preconditions, cost estimate, rollback, digest	Resolve current state deterministically; policy check required.
Execute	Provider operation ID and immediate response	Narrow mutation identity; approved immutable plan; idempotency key.
Verify	Independent post-change read and outcome metrics	Prefer separate read path; compare against plan and SLO.
Roll back	Restored state or compensating action	Predeclared rollback authority; verify restoration and residual risk.

Critical boundary: Approval belongs between Plan and Execute. The approver should see the resolved targets and effects, not only the natural-language request.

20.5 Bind Identity and Authority to the Operation

An agent tool often crosses several identity domains. A user or upstream service initiates the request. The application authenticates that caller. The agent platform selects a capability. A tool broker may use a workload identity, exchange a delegated token, or invoke a provider-managed connection. The target resource performs the final authorization check. These identities must not be collapsed into one API key.

Two patterns recur. In a delegated pattern, the tool acts within the caller's existing rights. This is appropriate when the downstream system already has meaningful per-user authorization, such as a document repository or enterprise application. In a workload pattern, a narrowly scoped service identity performs one organizational operation, such as writing a budget at an approved resource-group scope. The workload pattern is easier to automate but transfers accountability to the application owner.

The capability contract must identify the target scope independently of the model. A resource ID, subscription, project, resource group, region, workspace, or tenant boundary should be resolved from authenticated application context or an allow-listed catalog. Free-form model text is not a trustworthy authority source. Credentials must never be embedded in prompts, tool descriptions, retrieved documents, or returned outputs.

MCP and OpenAPI can describe authentication mechanisms, and managed platforms can centralize connections, but transport-level authorization does not eliminate least-privilege design. OAuth security guidance emphasizes narrowly scoped tokens and secure flows;

MCP authorization builds on standardized authorization patterns for protected tools and user-specific data. [6][8][9]

Table 20-5 Identity chain for a governed tool call

Identity or decision	Question	Required evidence
End user or caller	Who requested the outcome?	Authenticated subject, organization or tenant, consent context.
Application	Which product surface accepted the request?	Application ID, endpoint, release, correlation ID.
Agent	Which agent version selected the capability?	Agent release digest and tool selection trace.
Tool broker	Which policy admitted or rejected the call?	Capability version, policy decision, risk class, approval ID.
Execution principal	Which identity reached the target service?	Principal ID, token audience or connection reference, role scope.
Target service	Which action was authorized and applied?	Provider audit or operation ID, target resource ID, result.

20.6 Schemas Constrain Shape; Validators Constrain Meaning

Function calling, OpenAPI, and MCP all rely on machine-readable schemas. JSON Schema provides a common vocabulary for structural validation. OpenAPI adds HTTP operations, parameters, media types, and security schemes. MCP exposes tools by name with metadata and schemas so clients can discover and invoke them. These standards reduce integration friction, but a syntactically valid request can still be unsafe or nonsensical. [3][4][5]

Validation should therefore occur in layers. Structural validation checks types, required fields, formats, enumerations, and bounds. Referential validation confirms that resource IDs exist and belong to the allowed scope. Semantic validation checks business rules: the budget period is allowed, the target SKU supports the workload, the storage tier change complies with retention policy, or the knowledge source version is approved. State validation checks preconditions and concurrency. Policy validation applies organization rules. Cost validation rejects actions outside the permitted cost or savings envelope.

Descriptions should be written for model selection, while validators should be written for adversarial inputs. A helpful description explains when to call the tool and what it does not do. It should not attempt to encode authorization in prose. A malicious prompt can ask the

model to ignore prose; it cannot make the target resource grant a role that the execution principal does not possess.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "finops.budget.create input",
  "type": "object",
  "additionalProperties": false,
  "required": ["scopeId", "amount", "timeGrain", "notificationProfile"],
  "properties": {
    "scopeId": {"type": "string", "pattern":
"^/subscriptions/[^/]+/resourceGroups/[^/]+$"},
    "amount": {"type": "number", "exclusiveMinimum": 0, "maximum": 1000000},
    "timeGrain": {"enum": ["Monthly", "Quarterly", "Annually"]},
    "notificationProfile": {"type": "string", "enum": ["team-default", "finance-
review"]},
    "idempotencyKey": {"type": "string", "minLength": 16, "maxLength": 128}
  }
}
```

Example 20-2. Structural schema for a bounded budget capability

Table 20-6 Validation layers after model generation

Layer	Example question	Failure response
Structural	Are fields present, typed, bounded, and free of unknown properties?	Reject before any provider call.
Referential	Does the target exist and belong to the authenticated scope?	Return a non-disclosing target error.
Semantic	Does the requested state make sense for this capability?	Return actionable validation details.
State and concurrency	Has the resource changed since the plan was created?	Reject stale plan; rebuild and reapprove.
Policy	Is the region, tag, owner, data class, or action permitted?	Record policy denial and required remediation.
Cost and quota	Does the action exceed budget, quota, or unit-economic constraints?	Reject, down-scope, or require higher approval.

20.7 Risk Classes, Approval, and Human Control

Tool governance should classify the consequence of successful execution, not merely the sensitivity of the input. A read-only inventory query may expose sensitive metadata but does not change state. A budget creation call changes a control-plane resource and notification behavior. A VM resize may be reversible yet service-affecting. Deleting a key, changing identity policy, sending an external message, or committing a purchase can have irreversible or legally significant effects.

OWASP describes excessive agency as excessive functionality, excessive permissions, or excessive autonomy. The mitigation is not a better system prompt. It is a narrower tool surface, least-privileged execution, independent validation, and explicit approval for high-impact operations. [12][13]

Approval must bind the exact plan. The approval record should include the capability version, target resource IDs, parameter values, expected effects, plan digest, approver, expiration, and any conditions. If current state changes or the plan is rebuilt, the digest changes and the approval no longer applies. The model must not approve its own high-impact action, and a previous user sentence such as “optimize everything” is not sufficient authorization for an unbounded future change.

A mutating tool should execute an approved, immutable plan

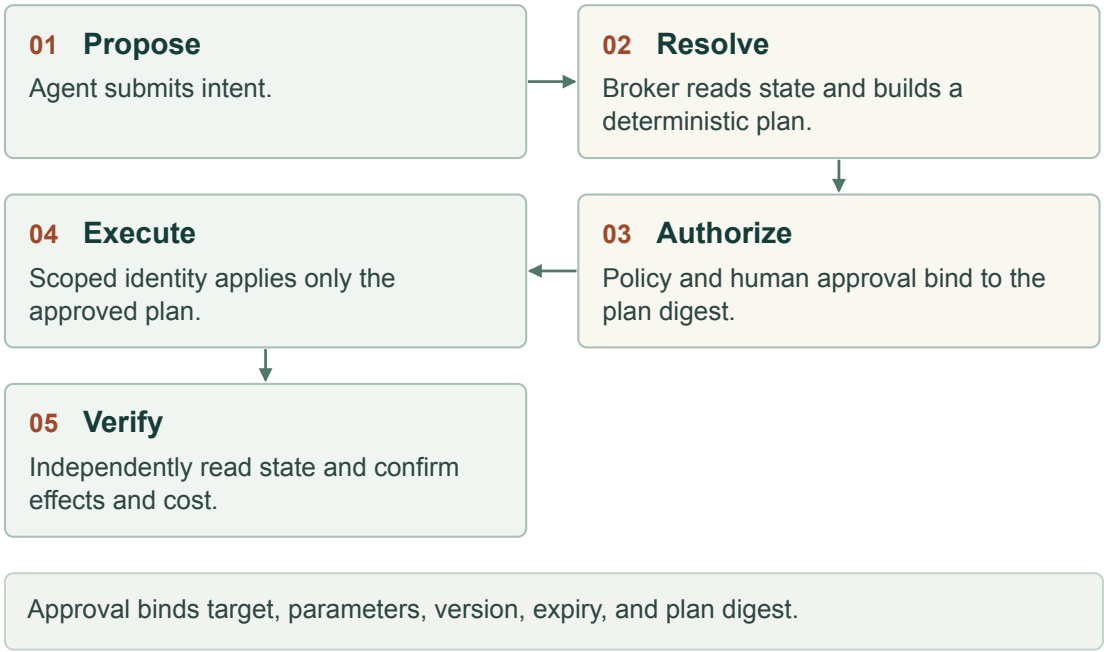


Figure 20-4 A mutating capability should execute an immutable, policy-checked, approved plan.

Table 20-7 Example capability risk classes

Class	Typical capability	Default control
R0 - informational	Read public documentation or non-sensitive health status	Automatic execution; normal telemetry.
R1 - scoped read	Inventory resources, read cost, inspect configuration	Least-privileged identity; data minimization; rate limits.
R2 - bounded reversible change	Create a budget, change a tag, propose GitOps change	Dry run or plan; policy check; approval according to scope.
R3 - service-affecting or costly change	Resize compute, alter network exposure, rotate deployment	Explicit plan approval; canary or maintenance window; rollback.
R4 - destructive, privilege, financial, or external effect	Delete data, grant roles, purchase capacity, send external communication	Deterministic workflow; strong reauthentication; separation of duties; often prohibit direct agent execution.

20.8 Idempotency, Concurrency, and Long-Running Operations

Agents retry. Networks fail. Tool servers time out. Models may emit the same call twice. A production tool must assume duplicate delivery and ambiguous completion. Read operations are naturally repeatable, but mutating operations need explicit idempotency semantics. An idempotency key should identify one logical request and return the existing operation or result when a duplicate arrives.

Concurrency protection is equally important. A plan is valid against a particular state. If another operator changes the resource before execution, the tool should reject the stale plan using a resource version, ETag, generation, or provider-specific precondition. Silently applying the old plan can undo a legitimate change or produce a different effect than the approver reviewed.

Long-running cloud operations should not be hidden behind an indefinitely open tool call. The tool can validate and accept the request, return an operation resource, expose status and cancellation, and complete asynchronously. The asynchronous request-reply pattern commonly uses HTTP 202, a status location, progress state, and an idempotency key to avoid duplicate work. [20][21]

Table 20-8 Reliability semantics that belong in the tool contract

Concern	Contract field or behavior	Evidence
Duplicate request	Required idempotency key or naturally idempotent method	Same operation ID returned for duplicates.
Concurrent change	ETag, generation, expected version, or plan digest precondition	Provider conflict or stale-plan rejection.
Long-running operation	Accepted status, operation URL, polling or event completion, cancellation	Operation state transitions and final provider result.
Partial failure	Step-level state and compensating action	Completed steps, residual resources, rollback status.
Timeout	Defined client and server deadlines	Timeout classification; unknown-versus-failed distinction.
Retry	Bounded retry policy based on error class	Attempt count, backoff, final status.

20.9 Result Envelopes and Untrusted Tool Output

A provider response should not be inserted into the next model turn as an opaque string. The tool adapter should normalize it into a stable result envelope that separates facts, effects, evidence, warnings, and model-facing explanation. This envelope allows the application to render the authoritative fields directly while giving the model only the information needed to continue reasoning.

Tool output is untrusted input. An external API, retrieved document, ticket description, log message, or browser page can contain text that attempts to redirect the agent. The tool boundary should mark content as data, preserve provenance, apply size and data-loss controls, and avoid letting raw output redefine the agent's instructions or authority. The model may summarize a result; it must not turn a result string into permission.

```
{
  "capabilityId": "finops.budget.create",
  "capabilityVersion": "1.0.0",
  "requestId": "req_01J...",
  "operationId": "op_01J...",
  "status": "succeeded",
  "target": {"provider": "azure", "resourceId":
"/subscriptions/.../budgets/team-monthly"},
  "effect": {"created": true, "amount": 5000, "currency": "USD", "timeGrain":
"Monthly"},
  "verification": {"method": "provider-read", "observedAt": "2026-08-
25T09:15:00Z"},
  "evidence": {"traceId": "...", "policyDecisionId": "...",
"providerOperationId": "..."},
```

```
  "cost": {"toolExecution": 0.0012, "currency": "USD"},
  "warnings": [],
  "rollback": {"supported": true, "capabilityId": "finops.budget.delete"}
}
```

Example 20-3. Stable tool result envelope

Table 20-9 Result-envelope fields

Field	Purpose
Capability and version	Tie behavior to a reviewed contract.
Request and operation IDs	Correlate retries, asynchronous work, and provider evidence.
Status	Distinguish accepted, running, succeeded, failed, rejected, canceled, and unknown.
Target	Identify the actual provider and resource boundary without exposing secrets.
Effect	State what changed, not merely what the provider returned.
Verification	Record how and when the effect was independently observed.
Evidence	Link trace, policy, approval, provider operation, and audit records.
Cost and warnings	Support FinOps, safety, and user-facing decisions.
Rollback	Identify whether reversal is supported and by which capability.

20.10 Observability, Audit, and FinOps for Tools

Tool calls should participate in the same managed feedback loop described in the observability chapter. A trace should connect the user request, agent release, tool selection, policy result, approval, execution principal, provider operation, verification read, and final answer. The trace must not indiscriminately record prompts, credentials, or sensitive provider payloads; evidence design includes redaction and retention policy.

Operational metrics should be capability-specific. Useful measures include call rate, success and rejection rate, validation failures, policy denials, approval latency, execution latency, provider throttling, stale-plan conflicts, rollback rate, verification failures, and cost

per successful outcome. A high model-level task-completion score can conceal a tool that frequently violates policy or requires manual repair.

Cost attribution should separate model selection, tool execution, provider consumption, and downstream business effect. A cost-read tool may itself be inexpensive while the action it recommends creates a large commitment. The capability card should therefore distinguish execution cost, expected resource-cost change, and realized savings or value after verification.

Evidence closes the loop between intention and effect

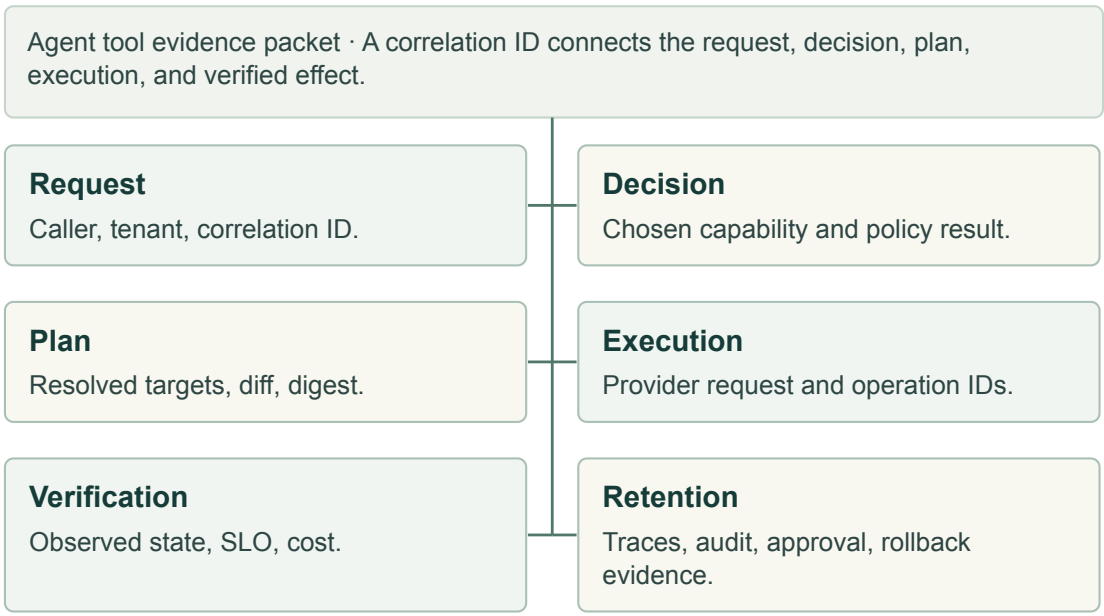


Figure 20-5 The evidence packet connects request, decision, plan, execution, verification, and audit.

Table 20-10 Minimum operational signals for an agent tool

Signal family	Examples	Decision supported
Demand	Calls, unique callers, tool-selection rate	Capacity, catalog relevance, deprecation.
Quality	Validation failure, wrong-target prevention, verification mismatch	Schema improvement and safety review.
Reliability	Latency, timeout, retry, provider error, rollback	SLO, resiliency, provider escalation.

Signal family	Examples	Decision supported
Authorization	Denied calls, scope mismatch, token or connection failures	Least-privilege review and incident detection.
Human control	Approval rate, approval latency, rejected plans, expired approvals	Risk-class tuning and workflow design.
Cost and value	Execution cost, provider cost delta, realized saving or outcome	FinOps and unit economics.

20.11 Tool Exposure Patterns: Function Calling, OpenAPI, MCP, and Workflows

The capability contract is independent of the exposure protocol. Teams should choose a pattern based on ownership, reuse, latency, authorization, and operational responsibility.

Function calling is the simplest pattern when the application owns the tool implementation. The model emits a structured request, the application validates it, executes local code or an internal service, and returns the result. It offers strong control but requires each application to manage discovery, credentials, and lifecycle.

OpenAPI is appropriate when a stable HTTP API already exists. OpenAPI 3.1 can describe operations, schemas, servers, media types, and security requirements. A model-facing tool still needs curated operation descriptions, bounded operations, and target-service authorization. An enormous enterprise API should rarely be imported wholesale.

MCP standardizes discovery and invocation of remote tools and resources. It is well suited to shared tool servers maintained by a platform or domain team. The MCP specification identifies tools by name and schema, while its authorization model provides a standard path for protected operations. MCP does not determine the enterprise risk class, least-privileged target role, approval policy, or provider-specific rollback.

Workflow connectors are useful when a process should remain deterministic. A workflow can collect approval, call several systems, wait for asynchronous completion, compensate failures, and produce business-level evidence. The agent may choose or parameterize the workflow, while the workflow retains the exact sequence and controls. Managed built-in tools are appropriate when the platform owns the execution environment and its security contract is understood.

Table 20-11 Tool exposure patterns

Pattern	Best fit	Governance focus
Function calling	Application-owned, low-latency, narrowly scoped code	Application executes; validate every argument; manage identity and lifecycle locally.
OpenAPI tool	Existing HTTP service with stable operations	Curate operations; declare security; use least-privileged managed identity or connection.
MCP server	Shared tools across agents and clients	Server ownership, OAuth authorization, catalog versioning, approval, tenancy, audit.
Workflow or connector	Multi-step business process, approval, waiting, compensation	Deterministic sequence, connection governance, step evidence, replay behavior.
Built-in managed tool	Search, code execution, file retrieval, or provider-managed capability	Platform boundary, data handling, sandbox, quota, and trace visibility.

Protocol rule: MCP, OpenAPI, or function calling can standardize invocation. None of them replaces capability ownership, provider authorization, policy, approval, verification, or retirement.

20.12 Azure Reference Pattern: Foundry Toolbox, Managed Identity, and Provider Adapters

Microsoft Foundry provides a useful reference pattern because it distinguishes built-in and custom tools and recommends toolboxes as reusable managed collections. Current documentation describes toolboxes as managed MCP endpoints that centralize credential and policy management and allow tool versions to change without modifying every agent definition. Custom tools can be connected through MCP or OpenAPI, while function calling leaves execution in the application. [14][15][16]

An Azure implementation can preserve the capability contract as follows. The Foundry agent sees only the curated tool metadata. A toolbox or application gateway authenticates the request and binds the approved capability version. An OpenAPI service, Azure Function, or Logic Apps workflow validates the arguments. A managed identity receives only the target-service roles needed for the capability. Azure Resource Manager or the relevant data-plane API performs the operation. Azure Monitor, Application Insights, Activity Log, and domain-specific logs record evidence.

Managed identity is preferable when the target service supports Microsoft Entra authentication because the runtime does not store a reusable secret. Foundry OpenAPI tools currently support anonymous, API-key, and managed-identity authentication; the managed identity still requires a least-privileged role at the target resource scope. The OpenAPI operationId should be descriptive and stable, and the specification should expose only the bounded operations intended for model use. [15]

One capability catalog, provider-specific adapters

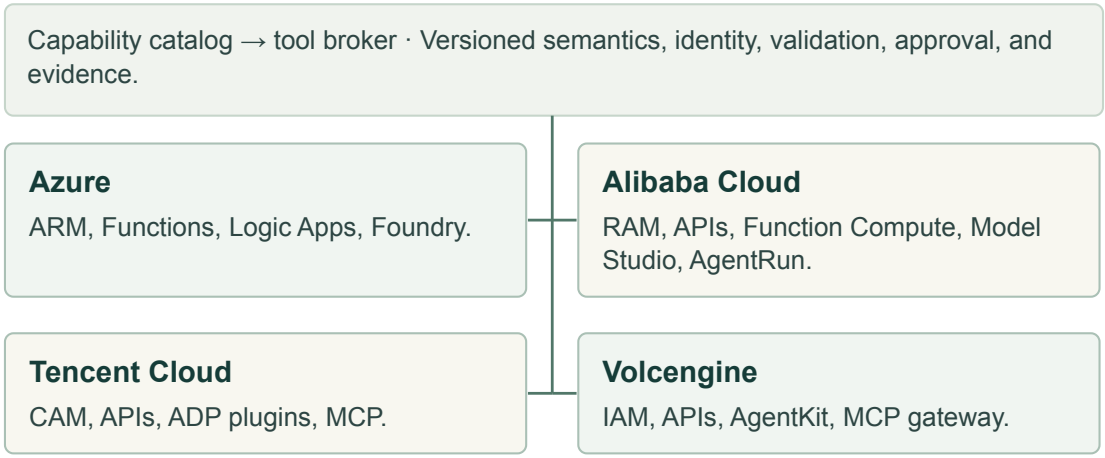


Figure 20-6 A shared capability catalog preserves engineering intent while provider adapters preserve non-equivalence.

Table 20-12 Azure reference implementation by contract layer

Contract layer	Azure implementation options
Discovery and versioning	Foundry toolbox or application catalog; repository-backed capability cards and OpenAPI specifications.
Authentication	Foundry project connection, Microsoft Entra delegated flow, or managed identity.
Execution adapter	Azure Functions, App Service or Container Apps API, Logic Apps workflow, or internal service.
Target authorization	Azure RBAC and service-specific data-plane roles at the narrowest scope.
Policy and approval	Azure Policy, application policy engine, workflow approval, deployment gates.

Contract layer	Azure implementation options
Evidence	Application Insights or OpenTelemetry trace, Azure Monitor metrics and logs, Activity Log, provider operation ID.
Lifecycle	Toolbox or API version, IaC deployment, release manifest, deprecation and identity revocation.

20.13 Multicloud Translation: Preserve Intent, Record Non-Equivalence

A multicloud capability catalog should not pretend that all providers implement identical resource models. The stable capability expresses intent; each adapter records how that intent maps to provider objects, authorization, asynchronous behavior, errors, observability, cost, and deletion. If one provider cannot preserve a required invariant, the adapter should report the non-equivalence rather than emulate success.

Alibaba Cloud Model Studio supports MCP and custom plugins, and Alibaba Cloud AgentRun describes a Tool Hub with MCP and function-call support, unified tool invocation, credential management, observability, and serverless execution. Tencent Cloud ADP exposes plugins and MCP-based tools within its application modes and provides workspace and application permission models. Volcengine AgentKit exposes MCP service and toolset management and can adapt existing REST or OpenAPI services into MCP-oriented tool surfaces. [24][25][26][27][28][29][30]

The engineering comparison should focus on six questions: What object represents the tool or plugin? How is it authenticated? Where are credentials stored? How are user and workspace permissions expressed? How is execution observed and costed? How is the tool version disabled or retired? Product names and UI organization differ; these operating questions remain stable.

Table 20-13 Multicloud tool-integration mapping

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Shared tool catalog	Foundry toolbox / MCP endpoint	Model Studio plugins; AgentRun Tool Hub	ADP plugin marketplace and MCP tools	AgentKit MCP services and toolsets
HTTP API adaptation	Foundry OpenAPI tool	Custom plugin or API-backed MCP / function call	Custom plugin through API and code	REST / OpenAPI adapted to MCP gateway

Engineering intent	Azure	Alibaba Cloud	Tencent Cloud	Volcengine
Workload credential	Managed identity or project connection	RAM roles, service-linked roles, credential management	CAM roles and platform or application permissions	IAM and AgentKit authentication configuration
Deterministic workflow	Logic Apps or Functions workflow	Model Studio workflow / Function Compute / AgentRun	ADP workflow mode	AgentKit workflow and runtime integrations
Evidence and operations	OpenTelemetry, Application Insights, Azure Monitor, Activity Log	AgentRun tracing, logs, ARMS or SLS integrations	ADP platform logs and application controls	AgentKit logs, MCP diagnostics, platform monitoring

20.14 The Capability Catalog Is a Product

A tool catalog is not a folder of JSON files. It is an internal product with owners, consumers, service objectives, security review, documentation, compatibility policy, support channels, and retirement obligations. The catalog should expose fewer high-quality capabilities rather than every provider operation.

Each capability moves through a lifecycle: draft, validated, approved, published, deprecated, and retired. Draft capabilities may be tested in isolated environments. Validation confirms schemas, authorization, failure behavior, and evidence. Approval records ownership and risk acceptance. Publication makes the capability discoverable to approved agents. Deprecation prevents new bindings and identifies a replacement. Retirement disables invocation, revokes credentials, archives required evidence, and verifies that no active agent release depends on the capability.

Versioning should reflect behavior, not file edits. A major version is appropriate when authority, side effects, semantics, or required inputs change incompatibly. A minor version can add optional fields or provider mappings without changing existing meaning. A patch can correct documentation or a compatible adapter defect. Tool descriptions also affect model selection and should be released deliberately; a “wording-only” change may alter behavior and therefore deserves evaluation.

Table 20-14 Capability lifecycle gates

State	Admission criteria	Exit evidence
Draft	Named owner, purpose, preliminary schema, provider prototype	Prototype tests and known limitations.

State	Admission criteria	Exit evidence
Validated	Schema, semantic, authorization, idempotency, failure, and evidence tests pass	Validation report and threat review.
Approved	Risk owner accepts policy, scope, identity, cost, and support model	Approval record and release digest.
Published	Discoverable to approved agents with documented SLO and version	Catalog entry, monitoring, support contact.
Deprecated	Replacement or retirement path exists; new bindings blocked	Dependency inventory and migration notice.
Retired	No active binding; endpoint disabled; identities and credentials revoked	Invocation failure test and archived evidence.

Capability Check C6: Write and Review Five Cloud Capability Cards

C6.1 Capability goal

Create a small, provider-neutral capability catalog from cloud abilities already verified elsewhere in this book. The catalog must include at least one read-only capability and one mutating capability. Each card must be precise enough that an independent reviewer can determine identity, scope, side effects, validation, evidence, cost boundary, and lifecycle without reading the adapter source code.

C6.2 Suggested capability set

Table 20-15 Suggested five-card capability set

Capability ID	Mode	Primary source chapter	Purpose
cloud.resource.inventory	Read-only	Cloud control plane and IaC	Return a normalized inventory for an approved scope.
network.exposure.audit	Read-only analysis	Networking and private connectivity	Identify public exposure and missing private-access controls.
finops.budget.create	Mutating	FinOps as a Cloud Control Plane	Create a scoped budget and notification policy from an approved plan.

Capability ID	Mode	Primary source chapter	Purpose
gitops.change.propose	Mutating but indirect	Continuous delivery and change governance	Create a reviewable configuration change rather than patch production directly.
rag.knowledge.refresh	Asynchronous mutation	RAG as a managed knowledge resource	Refresh an approved knowledge source and verify the published alias or version.

C6.3 Create the catalog structure

Create one directory per capability. Store capability.yaml, input.schema.json, output.schema.json, provider-mapping.md, test-cases/, and evidence-example.json. Keep secrets and live credentials outside the catalog.

Use stable capability IDs that describe intent rather than provider product names. Record the current provider adapters separately so that a provider rename does not force every agent binding to change.

C6.4 Complete each Capability Card

For each card, specify purpose, non-goals, owner, version, lifecycle state, mode, risk class, schemas, target-scope rules, identity path, required target actions, validation layers, approval rule, idempotency behavior, timeout, verification method, rollback, evidence fields, cost boundary, data classification, tenancy assumptions, and provider mappings.

The mutating card must include a dry-run or deterministic planning step and an approval rule that binds an immutable plan digest. The read-only card must still define data minimization, rate limits, and allowed resource scope.

C6.5 Review model-visible metadata

Write a concise tool name and description that help a model choose correctly. Include when the tool should not be used. Do not put credentials, hidden authorization rules, or provider-specific command syntax in the description.

Test ambiguous requests. For example, confirm that cloud.resource.inventory is selected for “show resources” but not for “delete unused resources,” and that finops.budget.create is not selected when the user asks only for a recommendation.

C6.6 Review authority and risk

Draw the identity chain from caller to target service. Confirm that the execution principal has only the operations required by the capability. Verify that target scope is derived from authenticated context or an allow-listed identifier rather than free-form model text.

Assign the risk class from Table 20-7. Document whether policy may auto-approve, whether a human is required, whether reauthentication is required, and whether direct agent execution is prohibited.

C6.7 Review reliability and evidence

Define duplicate-delivery behavior, concurrency preconditions, long-running operation handling, cancellation, retries, timeout semantics, and partial-failure behavior. Provide one example result envelope for success and one for rejection or failure.

List the trace spans, policy decision, approval, provider operation ID, verification read, cost fields, and retention period that form the evidence packet.

C6.8 Map the four cloud platforms

For each capability, map Azure, Alibaba Cloud, Tencent Cloud, and Volcengine to provider objects, APIs, identity mechanisms, observable evidence, and cleanup. Explicitly mark non-equivalence. Do not force a common field when one provider has materially different semantics.

C6.9 Adversarial review prompts

- Can an untrusted document or tool output make the model select a mutating capability that the user did not request?
- Can a caller substitute a resource ID from another subscription, project, workspace, or tenant?
- Can the same request be replayed and create a duplicate effect?
- Can an expired approval be reused after the plan or target state changes?
- Can the execution principal perform operations not represented by the capability?
- Can the tool claim success without an independent verification read?
- Can provider output inject new instructions into the agent conversation?
- Can a deprecated capability remain bound to an active agent release?

C6.10 Verification criteria

- Exactly five capability cards exist and at least one is read-only and one is mutating.
- Every card names an owner, version, lifecycle state, mode, risk class, identity path, target scope, schemas, and evidence contract.

- The mutating capability has a plan/dry-run boundary, approval rule, idempotency behavior, verification, and rollback or compensation statement.
- All cards include Azure plus Alibaba Cloud, Tencent Cloud, and Volcengine mappings with documented non-equivalence.
- A reviewer can reproduce the allowed and denied paths from the card and test cases.
- No credential, secret, live token, or unrestricted provider command appears in the catalog.

C6.11 Cost and cleanup boundary

The capability-card exercise can be completed without creating billable cloud resources. If live adapter tests are performed, use the dedicated lab scope and cleanup contract from the relevant source chapter. Retain only redacted evidence examples, schema files, review results, and provider mappings. Revoke temporary identities and project connections after testing.

20.15 CloudDesk Handoff: A Capability Foundation, Not a Runtime

The CloudDesk FinOps Agent introduced for the companion book can consume this catalog, but it should not redefine the capabilities. The desktop and serverless runtime will supply tenant resolution, session and response binding, approval interaction, task orchestration, tool execution, and cross-tenant isolation. The catalog supplies the stable contracts that those mechanisms enforce.

This division matters. A runtime can be secure yet still expose an unsafe capability. A capability can be well designed yet executed by a runtime that loses tenant context or misroutes a response. The first book establishes safe cloud semantics; the second book establishes the runtime that preserves them across desktop, gateway, serverless workers, and cloud services.

Table 20-16 Initial CloudDesk capability handoff

Capability	CloudDesk use	Required runtime behavior in the companion book
cloud.resource.inventory	Build tenant-scoped cloud inventory and ownership views	Resolve tenant cloud account; bind scope; cache and return only tenant-eligible results.
network.exposure.audit	Detect public endpoints and missing private access	Run read-only adapter; preserve evidence; route findings to the correct tenant workspace.

Capability	CloudDesk use	Required runtime behavior in the companion book
finops.budget.create	Create an approved budget after recommendation	Generate plan; desktop approval; execute with narrow identity; verify and record effect.
gitops.change.propose	Turn recommendations into reviewable configuration changes	Create tenant-bound branch or pull request; never bypass repository policy.
rag.knowledge.refresh	Refresh tenant cost policies and historical reports	Isolate source, queue, index, alias, trace, and result by tenant and workspace.

Boundary reminder: This chapter defines capability contracts. Tenant data models, cross-tenant isolation, streaming response binding, queue partitioning, desktop approval protocols, and custom serverless tool execution belong to Serverless AI Agent Development.

20.16 Key Takeaways

- A cloud API becomes an agent tool only after its purpose, authority, validation, risk, evidence, cost, and lifecycle are explicit.
- Tool descriptions and schemas influence model selection; the tool gateway and target service enforce authorization.
- Intent-level tools are safer and more portable than raw CLI, shell, SQL, SDK, or generic REST access.
- Observe, recommend, plan, execute, verify, and roll back are different authority stages and should not be collapsed.
- Mutating tools should execute immutable, policy-checked, approved plans with idempotency and concurrency controls.
- Tool results are untrusted data and should be normalized into stable result envelopes with provenance and verification.
- Function calling, OpenAPI, MCP, workflows, and built-in tools are exposure patterns, not substitutes for governance.
- A multicloud capability catalog preserves engineering intent while provider adapters preserve real differences.
- The capability catalog is a product with ownership, SLOs, security review, versioning, deprecation, and retirement.
- CloudDesk can build on these contracts, but its tenant system and serverless runtime must independently preserve isolation and execution correctness.

References

[1] National Institute of Standards and Technology. Zero Trust Architecture (SP 800-207). https://doi.org/10.6028/NIST.SP.800-207 Accessed August 25, 2026.

[2] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). https://doi.org/10.6028/NIST.AI.100-1 Accessed August 25, 2026.

[3] OpenAPI Initiative. OpenAPI Specification 3.1.1. https://spec.openapis.org/oas/v3.1.1.html Accessed August 25, 2026.

[4] JSON Schema. JSON Schema Specification, Draft 2020-12. https://json-schema.org/specification Accessed August 25, 2026.

[5] Model Context Protocol. Tools, Specification 2026-07-28. https://modelcontextprotocol.io/specification/2026-07-28/server/tools Accessed August 25, 2026.

[6] Model Context Protocol. Understanding Authorization in MCP. https://modelcontextprotocol.io/docs/2026-07-28/tutorials/security/authorization Accessed August 25, 2026.

[7] Model Context Protocol. Specification 2026-07-28. https://modelcontextprotocol.io/specification/2026-07-28 Accessed August 25, 2026.

[8] IETF. RFC 9700: Best Current Practice for OAuth 2.0 Security. https://www.rfc-editor.org/info/rfc9700/ Accessed August 25, 2026.

[9] IETF. RFC 9728: OAuth 2.0 Protected Resource Metadata. https://www.rfc-editor.org/info/rfc9728/ Accessed August 25, 2026.

[10] Cloud Native Computing Foundation. CloudEvents Specification. https://cloudevents.io/ Accessed August 25, 2026.

[11] OpenTelemetry. Semantic Conventions for Generative AI Systems. https://opentelemetry.io/docs/specs/semconv/gen-ai/ Accessed August 25, 2026.

[12] OWASP Gen AI Security Project. LLM06:2025 Excessive Agency. https://genai.owasp.org/llmrisk/llm062025-excessive-agency/ Accessed August 25, 2026.

[13] OWASP Cheat Sheet Series. AI Agent Security Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/AI_Agent_Security_Cheat_Sheet.html Accessed August 25, 2026.

[14] Microsoft Learn. Types of Tools in Microsoft Foundry Agent Service. https://learn.microsoft.com/en-us/azure/foundry/agents/concepts/tool-catalog Accessed August 25, 2026.

[15] Microsoft Learn. Connect OpenAPI Tools to Microsoft Foundry Agents. https://learn.microsoft.com/en-us/azure/foundry/agents/how-to/tools/openapi Accessed August 25, 2026.

[16] Microsoft Learn. Connect Agents to MCP Server Endpoints. https://learn.microsoft.com/en-us/azure/foundry/agents/how-to/tools/model-context-protocol Accessed August 25, 2026.

[17] Microsoft Learn. Tool Best Practices for Microsoft Foundry Agent Service. https://learn.microsoft.com/en-us/azure/foundry/agents/concepts/tool-best-practice Accessed August 25, 2026.

[18] Microsoft Learn. Authenticate Azure Logic Apps Connections with Managed Identities. https://learn.microsoft.com/en-us/azure/logic-apps/authenticate-with-managed-identity Accessed August 25, 2026.

[19] Microsoft Learn. Web API Design Best Practices. https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design Accessed August 25, 2026.

[20] Microsoft Learn. Asynchronous Request-Reply Pattern. https://learn.microsoft.com/en-us/azure/architecture/patterns/asynchronous-request-reply Accessed August 25, 2026.

[21] Microsoft Learn. Web API Implementation Best Practices. https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-implementation Accessed August 25, 2026.

[22] Microsoft Learn. Azure Resource Manager Templates Overview. https://learn.microsoft.com/en-us/azure/azure-resource-manager/templates/overview Accessed August 25, 2026.

[23] Microsoft Learn. Azure Logic Apps Overview. https://learn.microsoft.com/en-us/azure/logic-apps/logic-apps-overview Accessed August 25, 2026.

[24] Alibaba Cloud Documentation. Model Studio MCP. https://www.alibabacloud.com/help/en/model-studio/mcp Accessed August 25, 2026.

[25] Alibaba Cloud Documentation. Model Studio Custom Plugins. https://www.alibabacloud.com/help/en/model-studio/custom-plug-ins Accessed August 25, 2026.

[26] Alibaba Cloud Documentation. What Is AgentRun. https://www.alibabacloud.com/help/en/agentrun/what-is-agentrun Accessed August 25, 2026.

[27] Alibaba Cloud Documentation. Use Agent Identity in Model Studio High-Code Applications. https://www.alibabacloud.com/help/en/agentidentity/using-agent-identity-in-bailian-high-code Accessed August 25, 2026.

[28] Tencent Cloud Documentation. Agent Development Platform Plugin Introduction. https://intl.cloud.tencent.com/document/product/1254/72577 Accessed August 25, 2026.

[29] Tencent Cloud Documentation. Creating Agent Applications Based on API and Code. https://intl.cloud.tencent.com/document/product/1254/72579 Accessed August 25, 2026.

[30] Tencent Cloud Documentation. Platform-Side User Permissions. https://intl.cloud.tencent.com/document/product/1254/73347 Accessed August 25, 2026.

[31] Tencent Cloud Documentation. Application-Side User Permission Description. https://intl.cloud.tencent.com/document/product/1254/72504 Accessed August 25, 2026.

[32] Volcengine Documentation. AgentKit MCP Service Management. https://docs.volcengine.com/docs/86681/1913814?lang=en Accessed August 25, 2026.

[33] Volcengine Documentation. AgentKit Code Sandbox Tool Example. https://docs.volcengine.com/docs/86681/1860267?lang=en Accessed August 25, 2026.

[34] Volcengine Documentation. AgentKit A2A and MCP Tool Integration. https://www.volcengine.com/docs/86681/2249666 Accessed August 25, 2026.

[35] FinOps Foundation. FinOps Framework. https://www.finops.org/framework/ Accessed August 25, 2026.

[36] FinOps Foundation. FOCUS Specification. https://focus.finops.org/ Accessed August 25, 2026.

[37] Google. Site Reliability Engineering Workbook: Implementing SLOs. https://sre.google/workbook/implementing-slos/ Accessed August 25, 2026.

[38] CNCF. OpenTelemetry Documentation. https://opentelemetry.io/docs/ Accessed August 25, 2026.

Subject index

A–C

Entries are alphabetical. § marks a discussion. Lab marks an exercise or worked procedure. The number after the dot is the PDF page.

A **Agent evaluation** § 19.12 · 627

Agent identity chain § 19.7 · 618 § 20.5 · 657

Agent memory and sessions § 19.9 · 622

Agent trajectories § 19.11 · 626 § 20.10 · 664

Agent-resource contracts § 19.3 · 612 § 19.4 · 614

AI platform as a service (AI PaaS) § 16.5 · 527 § 17.2 · 546

AIOps § 13.7 · 428

Alerting § 13.6 · 427 Lab 13.12.10 · 441

Alibaba Cloud § 3.6 · 59 § 16.8 · 532

API management § 9.7 · 265 Lab 9.12 · 274

Application definition § 11.4 · 339 Fig. 11-4 · 339 Fig. 11-9 · 356

Approval and human control § 4.10 · 90 § 20.7 · 660

Argo CD § 12.6 · 385

Audit trails § 15.9 · 497 § 19.11 · 626 § 20.10 · 664

Azure reference control plane § 1.6 · 12 § 13.4 · 422 § 17.13 · 559

B **Backup and recovery** § 6.6 · 154 § 8.5 · 221

Bicep § 4.7 · 86 Lab 14.12.4 · 473

Budget guardrails § 14.5 · 456 Lab 14.12.5 · 475

Burn rate, error budgets § 13.5.3 · 427

C **Capability catalogs** § 20.14 · 670 Lab C6.4 · 672

Index, continued

C–D

C **Capacity and quotas** § 6.7 · 156 § 17.8 · 553 § 19.14 · 631

Citations and provenance § 18.6.4 · 589 Lab 18.9.10 · 600

Cloud cost management See FinOps

Cloud security posture management (CSPM) § 15.1 · 484 § 15.8 · 496

CloudDesk § 4.14 · 102 § 14.13 · 479 § 20.15 · 674

Compliance evidence § 15.7 · 494 Lab 15.15 · 505

Compute capacity § 6.7 · 156 § 6.8 · 158 Lab 6.12 · 163

ConfigMap and Secret § 11.8 · 345 Lab 11.14.4 · 357

Container registries § 10.6 · 301 Lab 10.14.2 · 318

Continuous delivery § 12.2 · 379 § 12.10 · 392

Continuous verification § 13.8 · 429 § 15.6 · 492

Control plane § 1.3 · 5 § 2.4 · 29 § 19.6 · 617

Cost allocation § 4.4 · 80 § 14.4 · 455

Cost anomalies and forecasting § 14.5 · 456

D **Data lakes** § 8.7 · 224 § 8.8 · 226 § 16.3 · 524

Data plane § 2.4 · 29 § 17.1 · 545 § 19.6 · 617

Deletion and cleanup § 2.9 · 38 Lab 7.14.12 · 208 Lab 11.14.12 · 367

Desired state § 2.5 · 31 § 11.3 · 336 Lab 12.15.3 · 401

DNS (Domain Name System) § 7.3 · 183 § 7.8 · 189 Lab 7.14.7 · 205

Document-level eligibility § 18.5.4 · 586 § 18.6.3 · 588 Lab 18.9.7 · 598

Index, continued

D–I

D	**Drift detection and correction**	§ 4.8 · 88 § 12.9 · 391 Lab 12.15.6 · 405
E	**Egress routing**	§ 7.4 · 184
	Elasticity and scaling	§ 6.4 · 150 § 11.10 · 348
	Engineering intent	§ 3.7 · 61 § 7.13 · 195 § 20.13 · 669
	Error budgets	§ 13.5 · 425
	Event-driven data paths	§ 8.9 · 227 § 9.6 · 262 Lab 9.13 · 278
	Exceptions and risk acceptance	§ 15.10 · 498
F	**Failure domains**	§ 6.5 · 153 § 6.13 · 172
	FinOps	§ 14.2 · 451 § 14.10 · 468 Lab 14.12 · 472
	Flux	§ 12.6 · 385 Lab 12.15.4 · 403
	Functions, event-driven	§ 9.5 · 259 Lab 9.13 · 278
G	**GitOps**	§ 12.5 · 384 Lab 12.15.5 · 404
H	**Health probes**	§ 11.6 · 342 Lab 11.14.4 · 357
	Hybrid retrieval	§ 18.5.3 · 586 Lab 18.9.7 · 598
I	**IaC**	See Infrastructure as code (IaC)
	IAM	See Identity and access management (IAM)
	Idempotency	§ 2.10 · 39 § 20.8 · 662
	Identity and access management (IAM)	§ 5.1 · 111 § 5.4 · 118 § 19.7 · 618
	Image digests	§ 10.3 · 296 Lab 10.14.4 · 319
	Image signing	§ 10.9 · 308 Lab 10.14.7 · 321

Index, continued

I–M

I	**Inference costs**	§ 14.9 · 466 § 17.12 · 558
	Inference endpoints	§ 17.1 · 545 § 17.3 · 547
	Infrastructure as code (IaC)	§ 4.6 · 84 § 4.10 · 90 Lab 4.13 · 97
	Ingress traffic	§ 7.6 · 187 § 11.7 · 343 Lab 11.14.6 · 363
K	**Knowledge bases**	§ 18.2 · 578 § 18.3 · 580 Lab 18.9.9 · 600
	Knowledge versioning and rollback	§ 18.3.5 · 582 Lab 18.9.12 · 602 Lab 18.9.13 · 603
	Kubernetes	See Managed Kubernetes
L	**Least privilege**	§ 5.2 · 113 § 5.6 · 124 Lab 5.10.7 · 137
	Lexical retrieval	§ 18.5.1 · 585 § 18.5.3 · 586
M	**MaaS**	See Model as a service (MaaS)
	Managed identities	§ 5.4.1 · 119 Lab 5.10.3 · 134
	Managed Kubernetes	§ 11.1 · 332 § 11.2 · 334 Lab 11.14 · 354
	MCP	See Model Context Protocol (MCP)
	Microsoft Foundry	§ 16.7 · 530 § 17.13 · 559 § 19.15 · 632
	MLOps, LLMOps, and AgentOps	§ 16.6 · 529
	Model as a service (MaaS)	§ 17.4 · 549 § 17.5 · 549
	Model Context Protocol (MCP)	§ 20.11 · 666
	Model deployments	§ 17.3 · 547 § 17.5 · 549 § 17.15 · 562
	Model quotas and rate limits	§ 17.8 · 553
	Model retirement	§ 17.9 · 554

Index, continued

M–R

M	**Model safety guardrails**	§ 17.10 · 555
	Multicloud equivalence	§ 3.7 · 61 § 3.8 · 63 § 20.13 · 669
	Multicloud placement	§ 3.4 · 55 § 3.9 · 64
N	**Network segmentation**	§ 7.3 · 183
O	**Observability**	§ 13.1 · 415 § 17.11 · 556 § 19.11 · 626
	Observability signals	§ 13.2 · 417 § 13.3 · 420
	OCI images	§ 10.2 · 294 Lab 10.14.4 · 319
	OpenTelemetry	§ 13.3 · 420 Lab 13.12.5 · 438
	Organizational governance	§ 15.2 · 486 § 15.3 · 487 § 15.12 · 501
P	**Persistent state**	§ 11.9 · 346 Fig. 11-9 · 356 Lab 11.14.7 · 364
	Persistent volume claims (PVCs)	§ 11.9 · 346 Lab 11.14.7 · 364
	Pod templates	§ 11.6 · 342 Lab 11.14.4 · 357
	Policy as code	§ 15.4 · 489 Lab 15.15 · 505
	Private connectivity	§ 7.7 · 188 § 18.6.2 · 588 Lab 7.14.7 · 205
	Processing boundaries, models	§ 17.7 · 552
	Public exposure	§ 7.5 · 185 Lab 7.14.6 · 204
	PVCs	See Persistent volume claims (PVCs)
R	**RAG**	See Retrieval-augmented generation (RAG)
	Reciprocal rank fusion (RRF)	§ 18.5.3 · 586 Lab 18.9.7 · 598
	Release rollback	§ 9.4 · 257 § 12.11 · 394 Lab 12.15.7 · 406

Index, continued

R–S

R	**Resource dependency graphs**	§ 1.4 · 8 § 2.6 · 33 Lab 6.12 · 163
	Resource identifiers	§ 2.3 · 28 § 4.3 · 78 Lab 1.9.3 · 19
	Resource inventory	§ 4.5 · 82 Lab 3.11 · 66
	Resource lifecycle	§ 2.7 · 35 § 2.9 · 38 Lab 1.9.4 · 19
	Resource locks and change evidence	§ 15.9 · 497
	Resource scope hierarchy	§ 4.2 · 77 § 15.3 · 487
	Retention policies	§ 8.6 · 223 § 13.10 · 432 Lab 8.14 · 234
	Retrieval authorization	§ 18.3.4 · 581 § 18.6.3 · 588 Lab 18.9.7 · 598
	Retrieval evaluation	§ 18.8.1 · 592 § 18.8.2 · 592 Lab 18.9.11 · 601
	Retrieval-augmented generation (RAG)	§ 18.2 · 578 § 18.8 · 592 Lab 18.9 · 593
	Role-based access control (RBAC)	§ 5.4.2 · 120 § 5.4.6 · 123 Lab 5.10.4 · 135
S	**Savings verification**	§ 14.6 · 459 § 14.10 · 468
	SBOM	See Software bills of materials (SBOMs)
	Secrets and credentials	§ 5.5 · 123 § 11.8 · 345 § 12.8 · 389
	Semantic reranking	§ 18.5.5 · 586
	Separation of duties	§ 12.8 · 389 § 15.12 · 501
	Service discovery	§ 11.7 · 343 Lab 11.14.6 · 363
	Service principals	§ 5.4.1 · 119
	Service-level indicators (SLIs)	§ 13.5 · 425 Lab 13.12.8 · 440
	Service-level objectives (SLOs)	§ 13.5 · 425 § 17.11 · 556 Lab 13.12.7 · 440

Index, continued

S–W

S	**Shared responsibility**	§ 3.2 · 51 § 3.3 · 54 § 11.2 · 334
	SLOs	See Service-level objectives (SLOs)
	Software bills of materials (SBOMs)	§ 10.7.1 · 306
	Supply-chain evidence	§ 10.7 · 304 Lab 10.14.9 · 322
T	**Tagging and cost attribution**	§ 4.3 · 78 § 4.4 · 80 Lab 4.12 · 93
	Tencent Cloud	§ 3.6 · 59 § 16.9 · 532
	Tool contracts	§ 19.8 · 621 § 20.2 · 651
	Tool granularity	§ 20.3 · 655
	Tool outputs	§ 20.9 · 663
	Tool risk and approval	§ 20.5 · 657 § 20.7 · 660
U	**Unit economics**	§ 14.7 · 460 § 17.12 · 558 § 19.14 · 631
V	**Vector retrieval**	§ 18.5.2 · 585 § 18.5.3 · 586
	Version promotion	§ 10.3 · 296 § 12.3 · 381 Lab 18.9.12 · 602
	Volcengine	§ 3.6 · 59 § 16.10 · 533
W	**Workload controllers**	§ 11.5 · 340 Fig. 11-4 · 339
	Workload identity	§ 5.3 · 116 § 5.4.4 · 121 § 17.6 · 551
	Workload identity federation	§ 5.4.4 · 121
	Workload shape	§ 6.2 · 146 § 6.3 · 149