Kube-DC

Run virtual machines, Kubernetes, databases, and storage on infrastructure you control.

Kube-DC turns your servers into a self-service private cloud. Departments and teams work in isolated projects with their own networks, virtual machines, Kubernetes clusters, databases, and storage. Your platform team sets capacity, security, and cost controls for the whole environment, while your data stays in your datacenter.

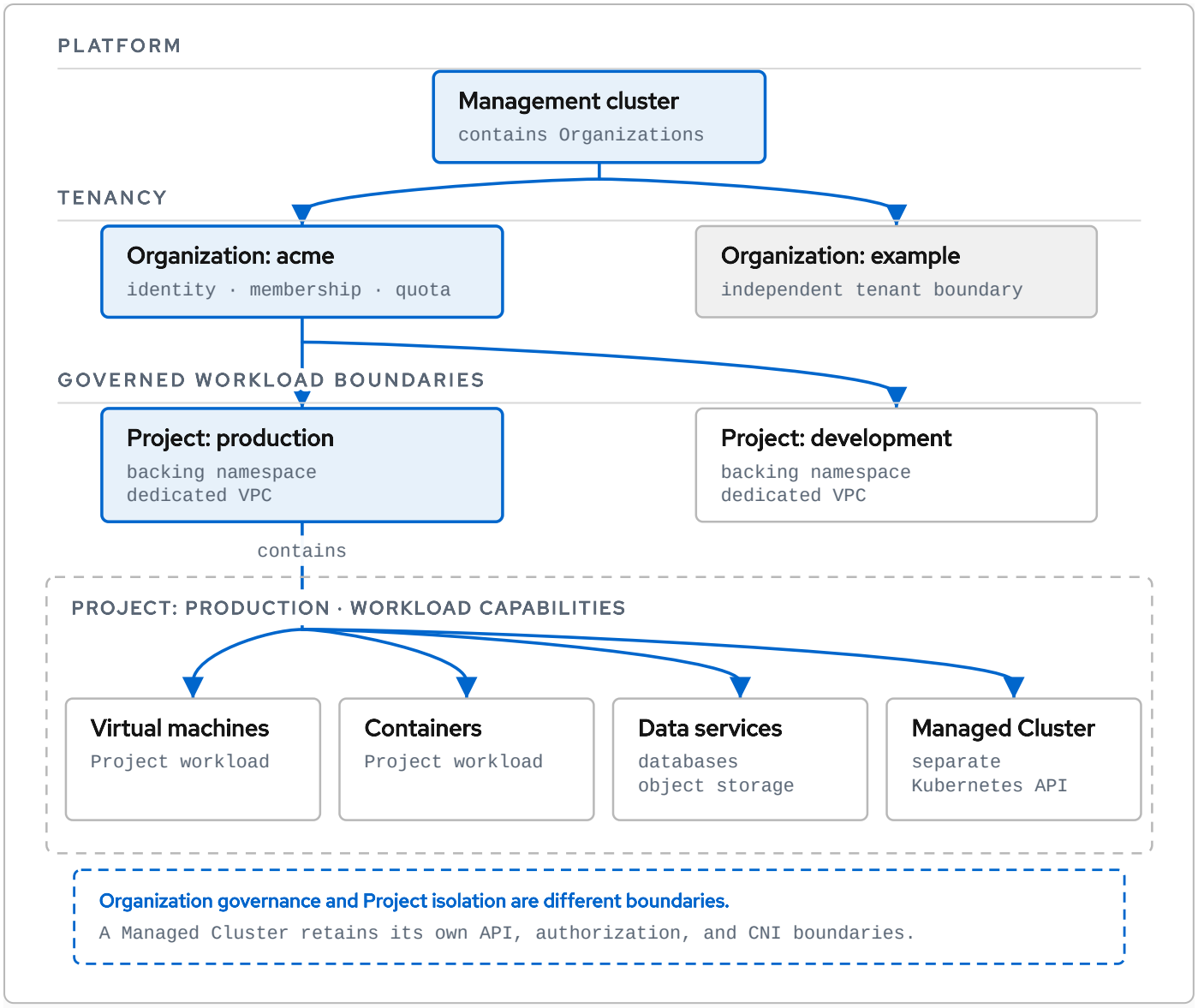
Who it's for

Kube-DC is designed for organizations that operate shared infrastructure for several teams or customers:

- **Public-sector organizations** keeping workloads and data on infrastructure they control, with clear separation between departments.
- **Enterprises and retailers** bringing existing virtual machines and modern container workloads onto one platform.
- **Universities and research institutions** giving faculties, labs, and student groups isolated environments with defined quotas.
- **Datacenter and hosting operators** providing cloud services under their own brand.

At a glance

- **One platform for VMs and containers.** Virtual machines and Kubernetes workloads use the same network fabric, access model, quota, and billing.
- **Tenancy that reaches the network.** Every project gets its own VPC and subnet, not just a namespace label.
- **Self-service with central controls.** Teams provision VMs, clusters, databases, storage, and public endpoints within the quotas you set. They can use the web console, `kubectl`, the CLI, or a supported coding assistant.
- **Runs on standard x86-64 servers.** A platform starts at three nodes; capacity grows by adding nodes. Production sizing is validated against your workload in an architecture review.
- **Built from named open-source components.** Kubernetes, KubeVirt, Kube-OVN, Ceph, Keycloak, CloudNativePG, and other components are integrated and operated as one product.



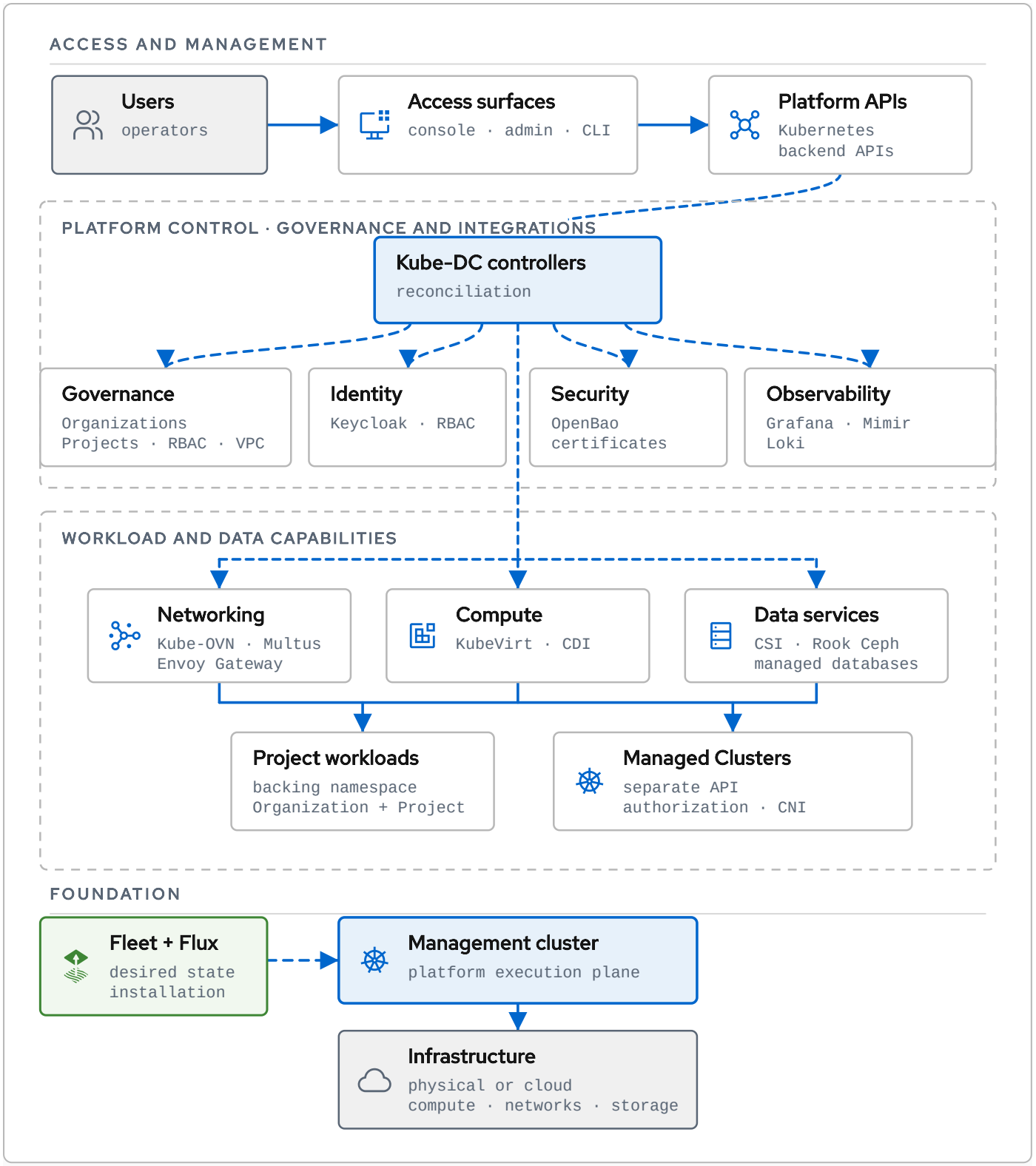
Kube-DC product hierarchy. Organizations are tenant boundaries; Projects are governed workload boundaries inside an Organization.

How it works

Kube-DC runs on a Kubernetes management cluster on your servers. Its controllers turn product resources such as organizations, projects, machines, clusters, databases, and keys into the underlying networking, virtualization, identity, storage, ingress, and observability resources.

The operating model has three levels:

1. **Your platform team** installs Kube-DC with the `kube-dc` CLI. The CLI bootstraps a GitOps repository, and platform configuration remains declarative and versioned from then on.
2. **Organizations** are your tenants: a ministry's directorates, a university's faculties, a retailer's business units, or an operator's customers. Each organization has its own identity realm, members, plan and billing scope.
3. **Projects** are where teams deploy. Each project has a namespace, RBAC, its own VPC, and optional quotas. Teams use the console or a standard kubeconfig to work with project resources.



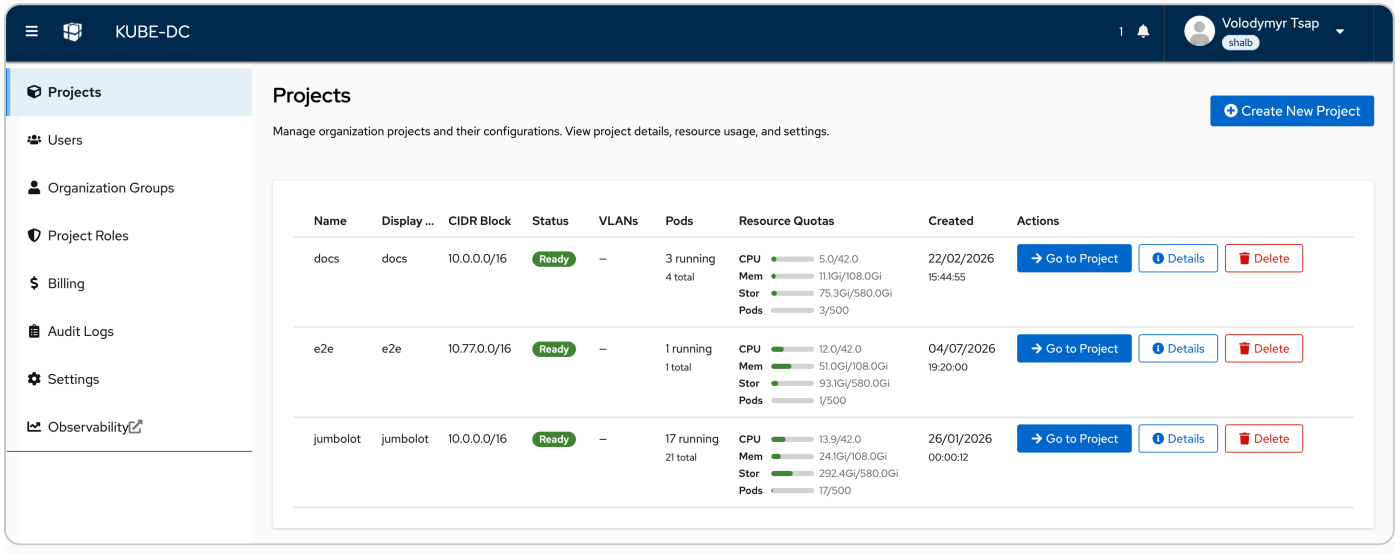
Kube-DC architectural layers. Organizations and Projects govern resources in the management cluster; Managed Clusters retain separate API, authorization, and CNI boundaries.

Multi-tenancy: organizations and projects

Layer	Scope	Mechanism
Identity	Organization	A dedicated identity realm per organization (Keycloak SSO — console, API, CLI)
Namespace & RBAC	Project	Hierarchical namespaces with project-scoped roles
Network	Project	A dedicated VPC and subnet per project
Quota	Project	Per-project quotas within the organization's plan
Billing & chargeback	Organization	Plans, usage and subscription billing per organization

Organization quotas cover configured Kubernetes resources such as CPU, memory, storage, pods, and load balancers. Optional project quotas divide that capacity between teams. Object storage and physical capacity use their own controls and capacity plans.

Project roles cover the work needed to deploy and operate workloads. Cluster-scoped administration stays with the platform team. Supported project roles do not include interactive pod `exec` or `attach`; an admission policy enforces that boundary, and teams run one-off tasks as auditable Jobs instead.



Organization administrators see Project readiness, network allocation, running workloads and quota consumption in one view.

Self-service projects: deploy applications directly

A project comes with a kubeconfig, so teams can deploy applications, Helm charts, Jobs, and services without first requesting a separate cluster. Plain Kubernetes manifests can describe a complete stateful stack: application pods, a managed database, an S3 bucket, shared volumes, an HTTPS endpoint, and autoscaling. The platform supplies the ingress, certificates, database engine, and storage.

When a team needs cluster-scoped control — operators, CRDs, custom controllers — it provisions a managed Kubernetes cluster instead (below).

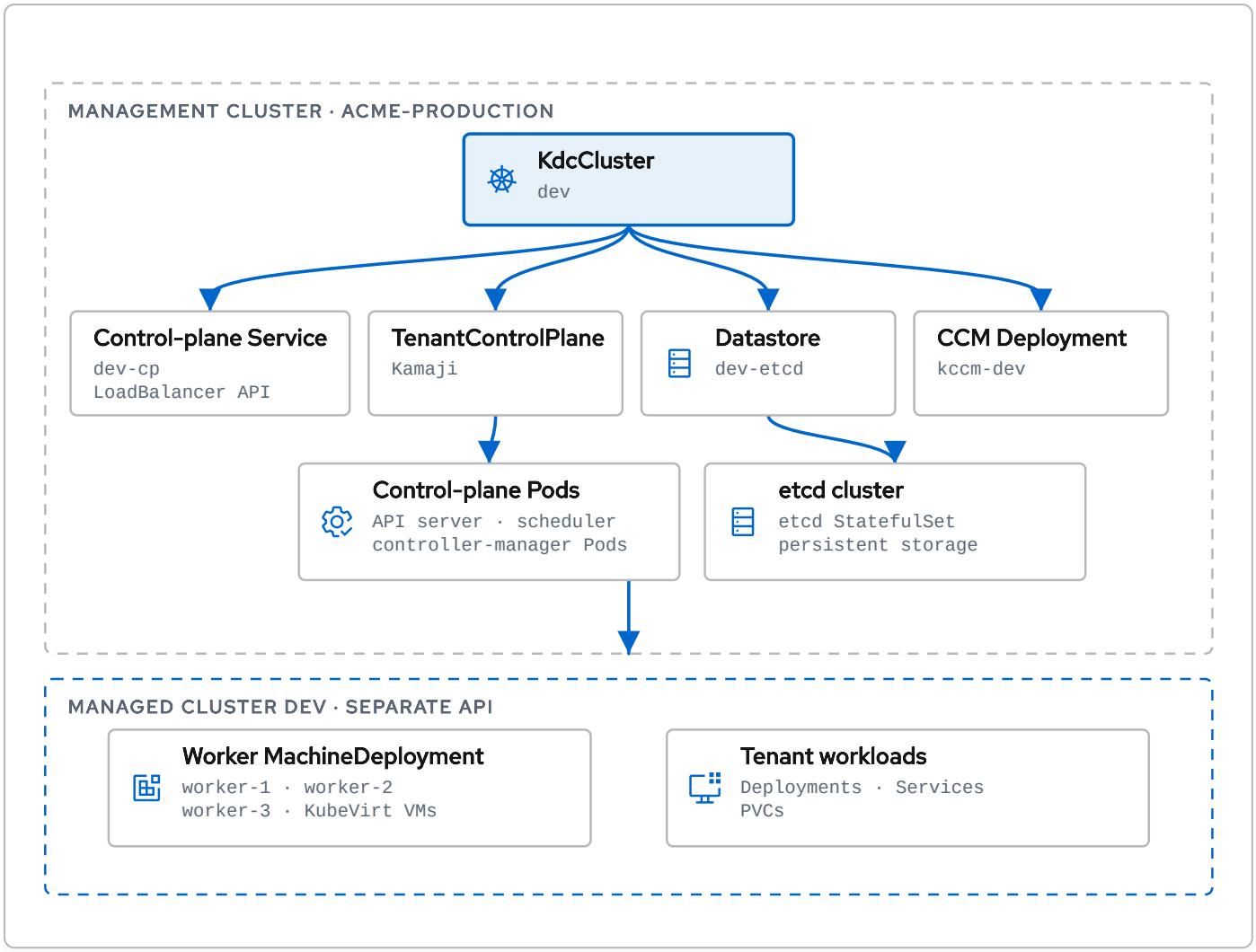
Virtual machines

- Linux and Windows guests from a prepared-image catalog — new VMs clone a prepared image snapshot, with no per-VM image download; bring your own images, with guest compatibility validated on import.
- Cloud-init configuration, SSH-key injection, interactive console in the web UI.
- Snapshots of running VMs; disk, CPU and memory sizing per VM.
- **Live migration**: VMs on the Migratable profile (shared block storage) are eligible to move between CPU-compatible hosts with sufficient capacity — the mechanism behind draining a host for maintenance.
- VMs attach to the project's VPC like any other workload: same networks, same floating IPs, same load balancers as containers.

Managed Kubernetes clusters

Full tenant-administered Kubernetes clusters, provisioned from a project with one manifest:

- **Hosted control planes** run as managed pods on the platform — tenants get a tenant-administered Kubernetes API with per-cluster PKI, without operating masters.
- **Worker pools** sized per pool (CPU, memory, disk, image), with autoscaling bounds per pool.
- **Control-plane and etcd vertical autoscaling** on by default — API server and etcd resources are right-sized within configured bounds as cluster load grows.
- **Staged, tenant-controlled upgrades** — control plane and worker pools move separately, in steps.
- **Scheduled etcd snapshots** (with optional envelope encryption) and restore — self-service for single-replica etcd datastores, assisted for multi-replica; private in-VPC endpoints by default, public HTTPS endpoints via the platform's gateway and certificate issuer on request.
- Admin and break-glass kubeconfigs retrievable by the tenant.



KdcCluster owns a Kamaji control plane, datastore, worker MachineDeployment, and CCM inside the Project infrastructure namespace; users reach a separate tenant Kubernetes API and run workloads on its workers.

Managed databases

- PostgreSQL and MariaDB, provisioned by manifest or console.
- Multi-replica engine replication; after an eligible primary failure an available replica is promoted automatically — behavior depends on your deployment's topology and capacity.
- Scheduled and on-demand backups to the project's S3 bucket, envelope-encrypted when a KMS key is configured; restore into a new database or in place.
- Auto-generated credentials delivered as Kubernetes Secrets; optional credential-rotation policies.

GPU services

- **Shared GPU for containers**: several workloads on one physical card, each holding a catalogued fixed slice of a specific GPU model — inference, notebooks, small training runs. The memory slice is enforced; the compute share is cooperative, so it is a density mechanism rather than a performance or security guarantee.
- **Dedicated GPU VMs**: one whole device attached to one guest, for workloads needing a stronger boundary (not live-migratable).

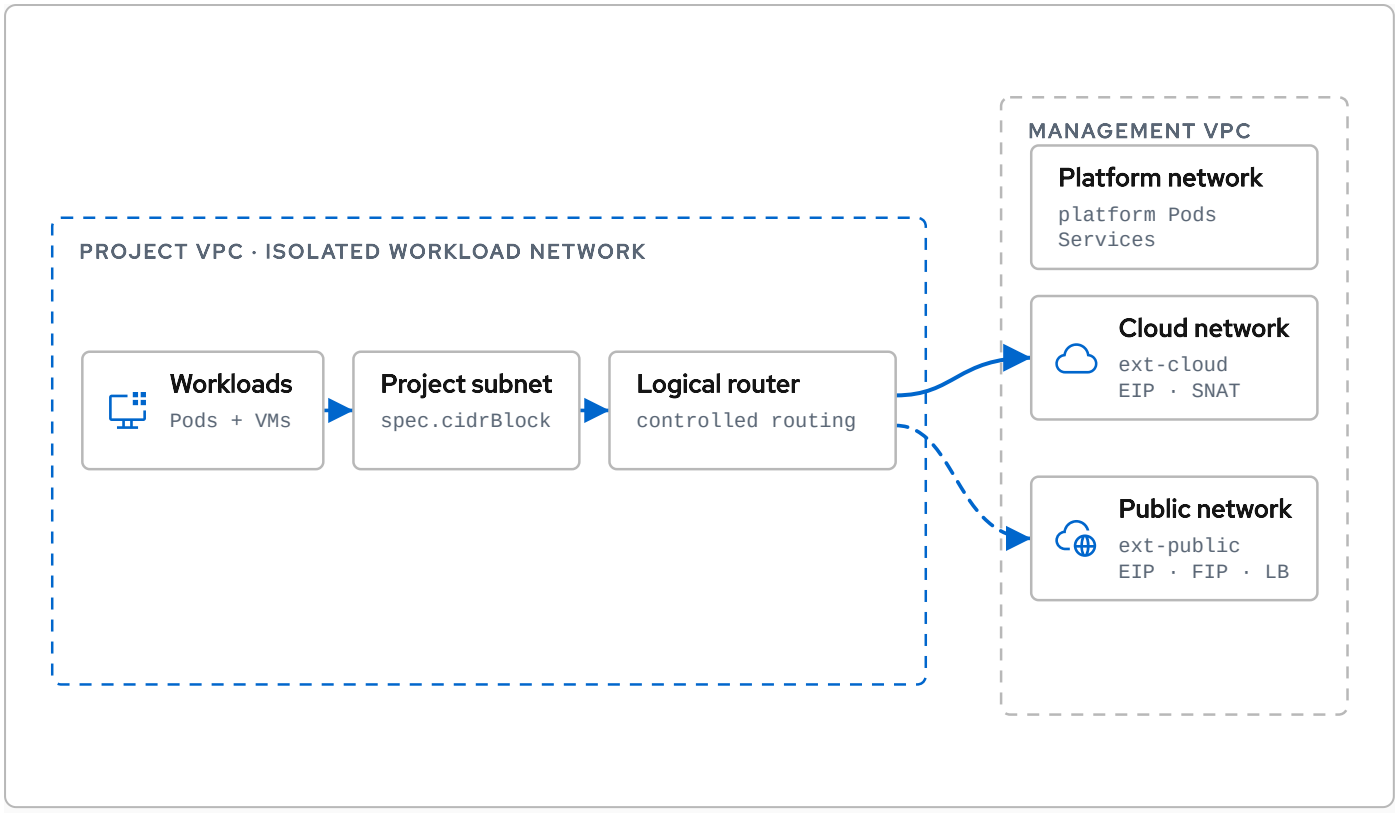
- **GPU as a governed product**: per-model entitlements enforced as quota, operator-held capacity reservations for organizations that can't queue, holder-safe node-mode transitions, an upgrade gate that qualifies device/kernel/driver/operator as one tuple, and a published threat and supply-chain model.
- GPU capabilities are enabled per cluster once your hardware, drivers and component set are qualified — GPU nodes couple those versions together, so enablement follows qualification.

Storage

- **Block storage** for VMs and workloads (Ceph RBD), including a shared read-write-many tier for multi-pod volumes and live-migratable VMs.
- **S3-compatible object storage** with per-project buckets, provisioned through a standard `ObjectBucketClaim`.
- **Volume snapshots** and instant clones from golden images.
- Storage classes, replication factors and capacity are yours to define — the platform runs on the disks and topology you give it.

Networking

- **A VPC per project** with private subnets; no private cross-project route exists by default — reachability exists only where a service is deliberately exposed.
- **External and floating IPs**: shared egress per project, dedicated addresses per load balancer, floating IPs that map to individual VMs — allocated from address pools your platform team defines.
- **LoadBalancer services and HTTPS ingress**: one annotation on a Service publishes it at a hostname with TLS, using the platform's gateway and certificate issuer on your address pools and DNS.
- **Physical VLAN attachment**: where the datacenter fabric trunks delegated VLANs to the platform nodes, a project's network bridges onto an existing VLAN at layer 2 — lab equipment, legacy systems, dedicated links — allocated from per-organization VLAN pools.
- Egress control and allowlists, operated by the platform team.



Each Project has its own VPC and creator-supplied workload subnet. Its default EIP and SNAT use the selected external network; public EIPs, FIPs, and Services are optional explicit paths.

Security and identity

- **Single sign-on** across console, API and CLI, with an identity realm per organization (Keycloak; OIDC).
- **Role-based access** per project from a curated, supported role set; custom roles under platform-team control.
- **Key management service**: per-project, purpose-scoped encryption keys backed by non-exportable keys in the platform's secrets backend (OpenBao).
- **Secrets manager** for application credentials, with sync into Kubernetes Secrets — keep secrets out of Git.
- **Managed certificates**: public trust via ACME or your organization's private CA.
- Admission policies enforce tenant boundaries at the API — including the platform-wide block on interactive pod access.

Observability

- **Configured automatically per organization.** When an organization is created, the platform's controllers provision its observability — a Grafana organization of its own, pre-built dashboards, a metrics tenant and log routing on the bundled multi-tenant Grafana/Mimir/Loki stack. Teams open Grafana and see their workloads; nothing to install.
- Metrics and logs are separated per tenant at the data layer — each organization sees only its own, including the control-plane telemetry of its managed Kubernetes clusters. Coverage and retention are operator-defined.
- Platform-level monitoring for the operator: cluster health, storage, networking and capacity, with alerting.

Billing and chargeback

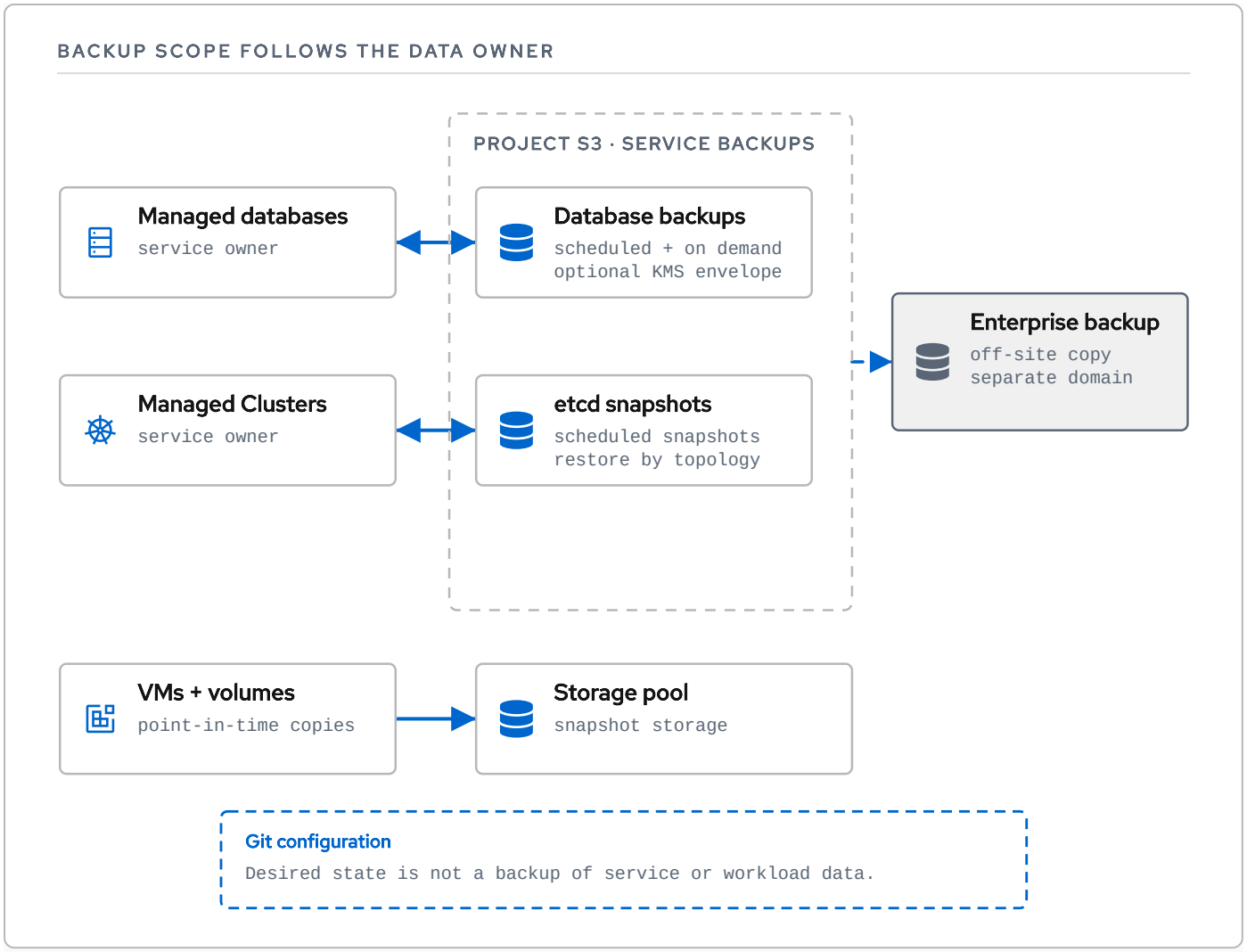
Organizations subscribe to plans that define their capacity; usage is enforced as quota and visible per organization — the basis for internal chargeback between departments or for invoicing external customers. Subscription billing integrates with payment providers for operators selling the platform as a service.

Automation and integration

- Tenant-facing services are Kubernetes-native or custom resources — the documented console and CLI workflows drive the same APIs, so `kubectl` and GitOps pipelines can too.
- The `kube-dc` CLI handles login, context switching and platform bootstrap.
- **Published agent skills** let AI coding assistants (Claude Code, Cursor and others) create projects, deploy applications, provision VMs and databases through guarded, documented workflows.

Data protection

Each managed service carries its own protection: database backups to project S3 (envelope-encrypted when a KMS key is configured); managed-cluster etcd snapshots and restore; VM and volume snapshots on demand. Platform configuration lives in Git as desired state — it is not a backup of service or workload data. For copies that must survive site or storage loss, integrate the platform's S3 endpoints and snapshot APIs with your enterprise backup system — Kube-DC does not replace it.



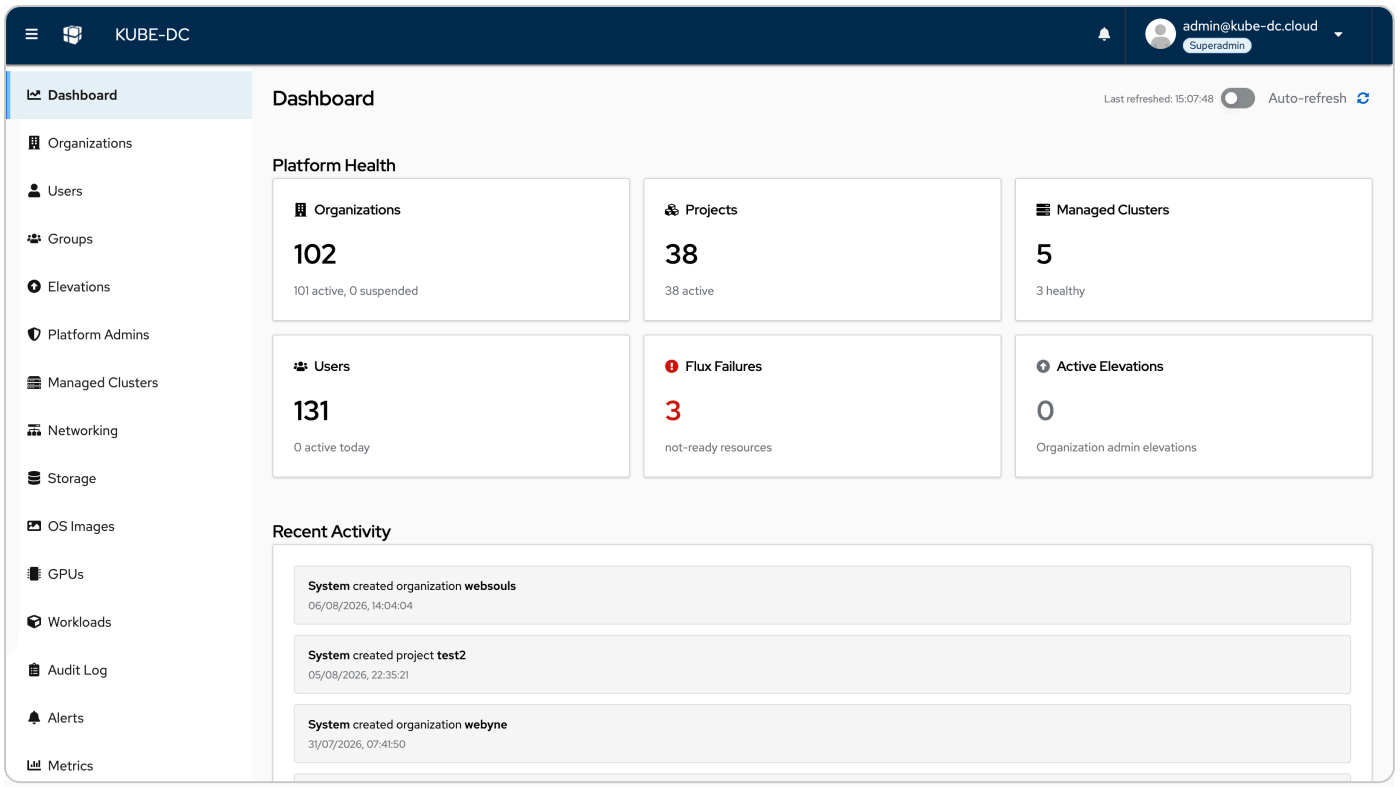
Each service protects its own data class: databases and Managed Cluster etcd use Project S3, VM and volume snapshots stay in the storage pool, and an enterprise backup integration is required for copies outside the platform's storage failure domain.

Operating the platform

Your platform team runs Kube-DC through three surfaces that cover the whole lifecycle:

- **Administration console** – a dedicated web panel for platform administrators: organizations and their plans, billing and subscriptions, capacity and storage health.
- **GitOps** – platform configuration is declaratively reconciled from a Git repository (Flux): component versions, platform settings and day-2 changes land as version-controlled commits.
- **`kube-dc` CLI** – carries the platform from installation through day-2: bootstrap of the GitOps repository, status and configuration commands, adoption of existing clusters, and login/context management for daily `kubectl` work.

Operator-side monitoring and alerting come from the same bundled observability stack the tenants use (see Observability).



The administration dashboard gives platform operators a consolidated health and activity view across the tenant estate.

Deployment and requirements

Reference baseline — subject to architecture validation. Production sizing and supported hardware depend on workload, storage topology, failure domains and validated NIC/firmware compatibility.

	Evaluation	Production
Server nodes	3	3+ (scale by adding nodes)
CPU per node	8 cores	16+ cores
RAM per node	32 GB	64+ GB
Storage per node	500 GB SSD	Per your capacity plan; dedicated disks for Ceph
Operating system	Ubuntu 24.04 LTS	Ubuntu 24.04 LTS
Network	VLAN-capable NIC; management, cloud and provider networks	Redundant NICs

Installation is CLI-driven: `kube-dc bootstrap init` scaffolds the GitOps repository and hands the platform to Flux for reconciliation. Day-2 operations — upgrades, configuration changes, component versions — are Git commits.

Platform components

Kube-DC integrates named upstream open-source components: Kubernetes, KubeVirt (virtualization), Kube-OVN (SDN), Rook-Ceph (storage), Keycloak (identity), Kamaji + Cluster API (hosted control planes), CloudNativePG and the MariaDB operator (databases), OpenBao (keys and secrets), Envoy Gateway and cert-manager (ingress and TLS), Prometheus/Mimir/Loki/Grafana (observability), Flux (GitOps). Component versions are pinned per Kube-DC release. Workloads built on standard Kubernetes objects and standard VM guest formats carry no platform-specific dependencies unless they use the platform's own resources — which are enumerated, not hidden.

Next steps

- **Evaluate:** install on three servers, or start in a hosted evaluation environment.
- **Talk to us:** architecture review against your hardware, network and tenancy requirements.

Function datasheets in this guide

This document is the overview; each major function has its own datasheet with full technical depth:

Function	Datasheet
Managed Kubernetes clusters	function-managed-kubernetes.md
Virtual machines	function-virtual-machines.md
Managed databases	function-managed-databases.md
Networking & VLAN attachment	function-networking.md
Storage & object storage	function-storage.md
Security, identity & keys	function-security.md
Observability	function-observability.md
GPU services	function-gpu.md

Managed Kubernetes clusters

Each team gets its own Kubernetes API. Your platform team operates the control planes.

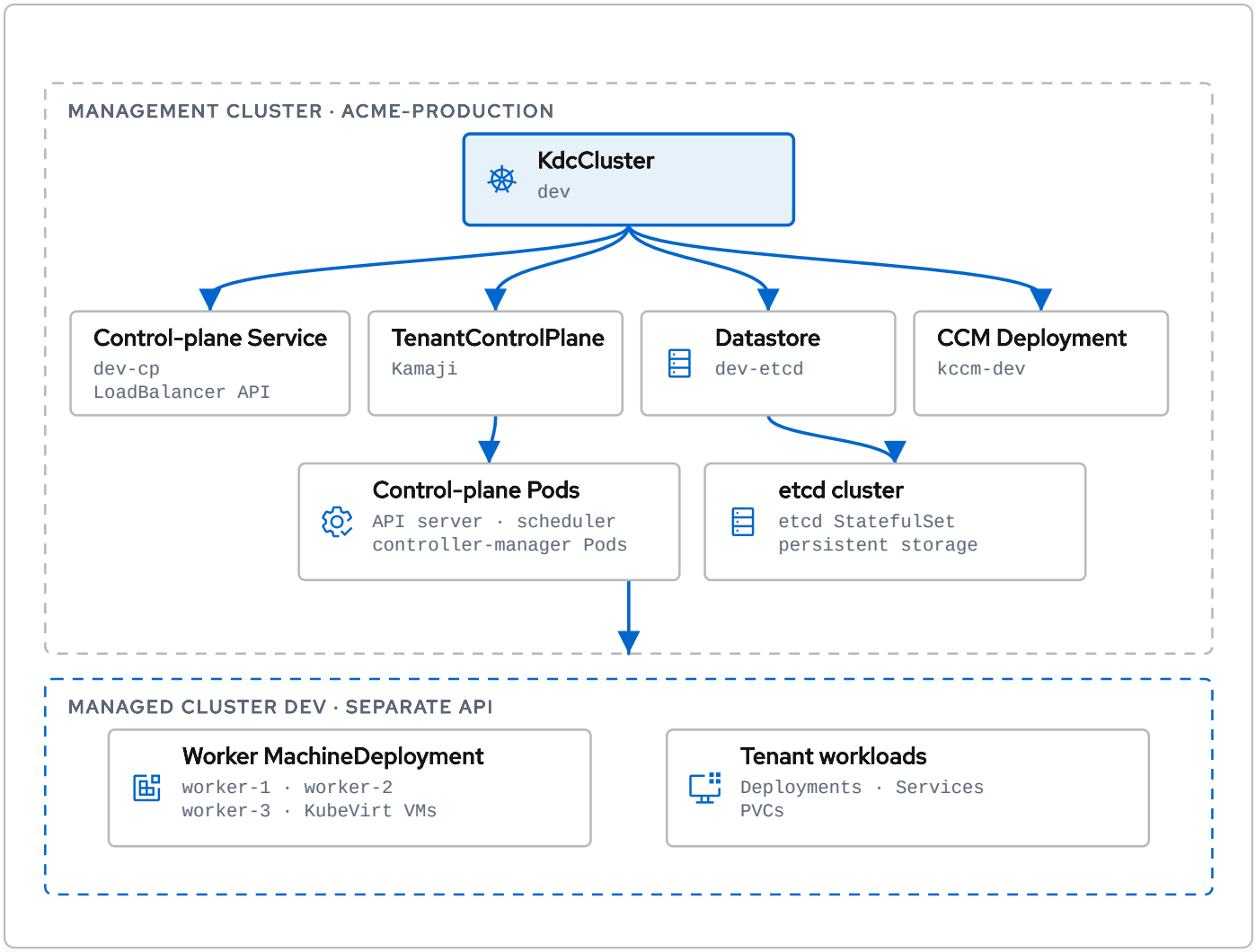
Departments, faculties, or customers can provision tenant-administered Kubernetes clusters from their projects. Kube-DC runs the control planes. Tenant administrators manage their clusters' workloads, upgrades, and access.

Architecture

A managed cluster has two parts:

- **The hosted control plane** runs as managed pods on the platform cluster. It includes the API server, controller manager, and scheduler, with a dedicated PKI for each cluster. Control planes use a platform-internal network that has no route from tenant networks. Tenants reach only the API endpoints published for their clusters.
- **Worker pools** run as virtual machines in the tenant project's VPC, beside the project's other workloads. Workers join the control plane when the cluster is created and when a pool scales up.

Clusters can share the platform datastore or use a **dedicated etcd datastore**. A dedicated datastore uses three replicas by default and keeps the cluster state on its own instance. The platform manages certificate rotation.



KdcCluster owns a Kamaji control plane, datastore, worker MachineDeployment, and CCM inside the Project infrastructure namespace; users reach a separate tenant Kubernetes API and run workloads on its workers.

Provisioning

A cluster is one manifest (or a console form):

```yaml
apiVersion: k8s.kube-dc.com/v1alpha1
kind: KdcCluster
metadata:
  name: analytics
  namespace: acme-data-platform
spec:
  version: v1.36.1
  controlPlane:
    replicas: 1            # 2+ = multi-replica topology; availability depends on deployment
  dataStore:
    dedicated: true        # own etcd instance for this cluster
  network:
    serviceCIDR: 10.96.0.0/16
    podCIDR: 10.244.0.0/16
  workers:
    - name: general
      replicas: 3
```

```
      cpuCores: 4
      memory: 8Gi
      diskSize: 30Gi
```

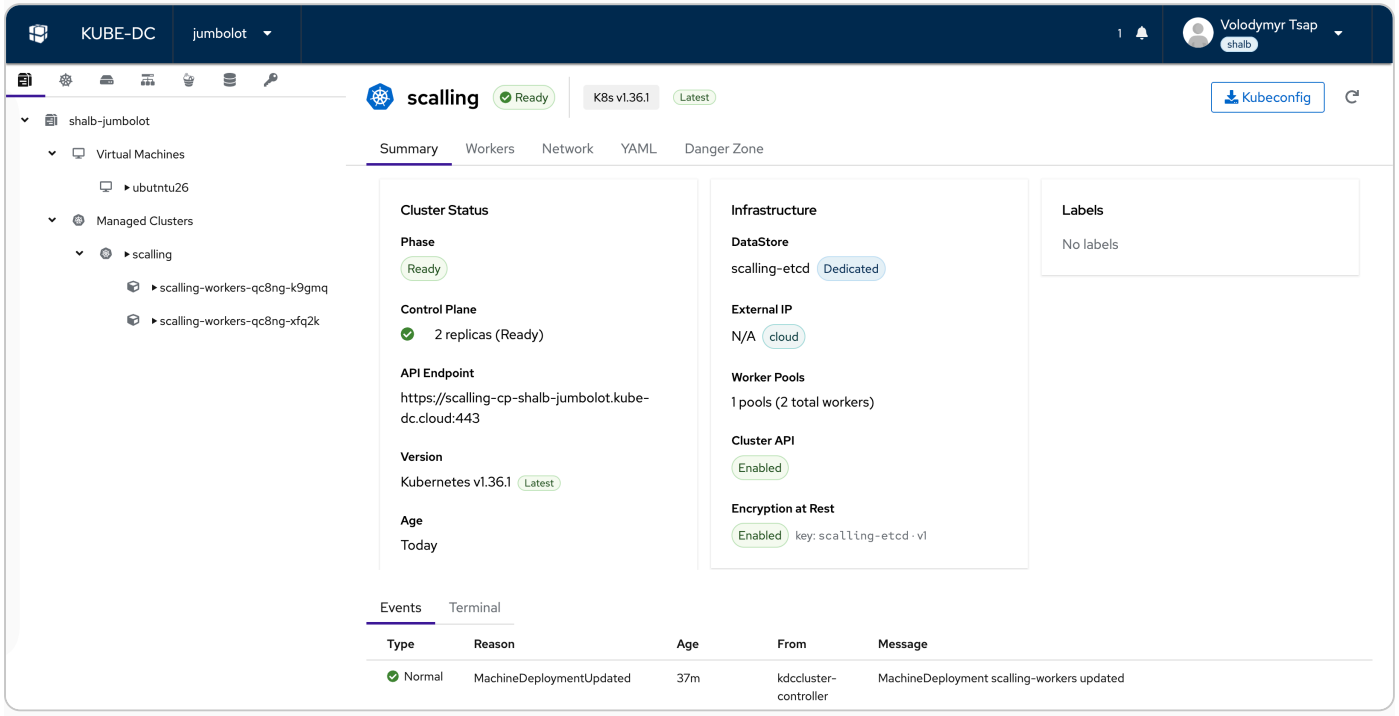
From this resource, the platform provisions the control plane, datastore, worker VMs, cluster PKI, and kubeconfigs, then joins the workers. Deleting the `KdcCluster` removes the cluster. Backup objects follow the configured bucket and retention policy.

Endpoints and access

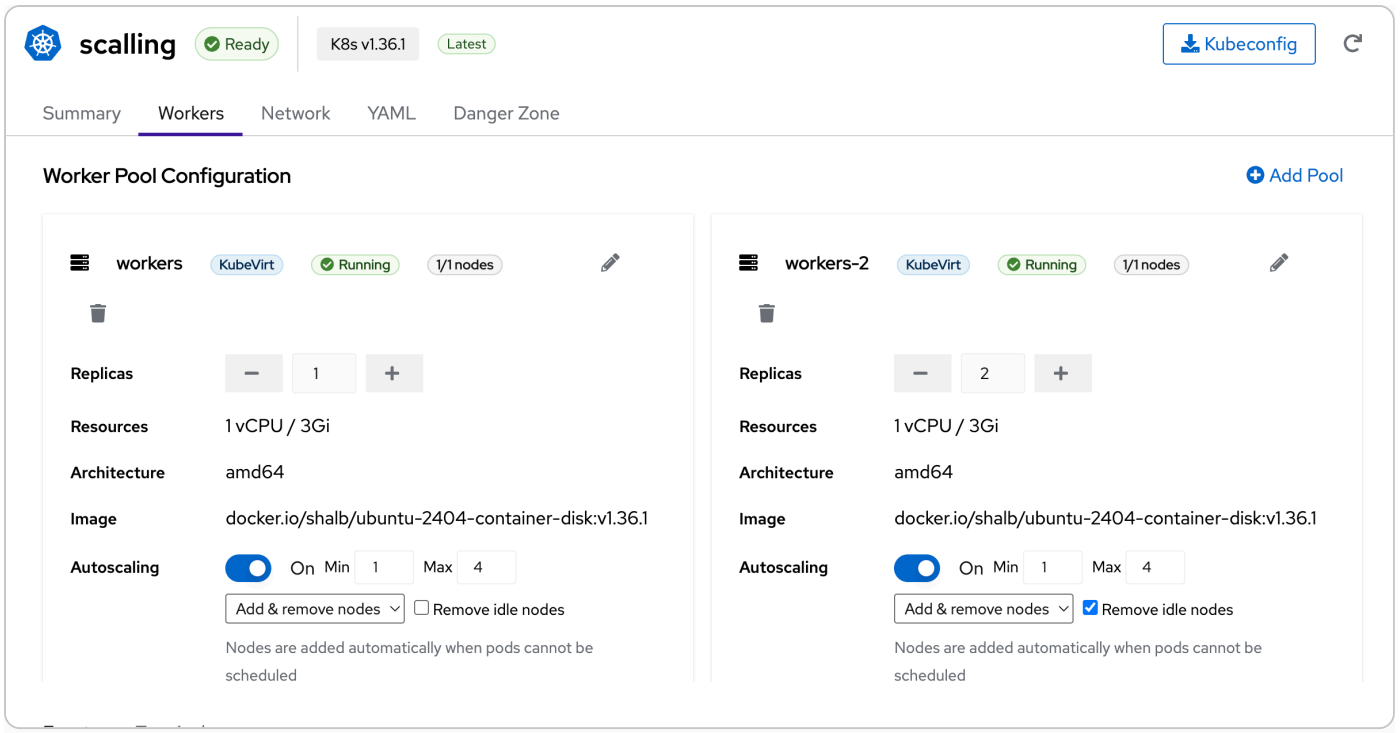
- **Private by default.** The cluster API is reachable from the owning project's VPC.
- **Public HTTPS when requested.** An annotation publishes the API at a stable hostname through the platform gateway, DNS, and certificate issuer. The API server certificate is reissued with the public name.
- **Kubeconfigs stored as Kubernetes Secrets.** The owning project receives an admin kubeconfig for the in-VPC endpoint and, when enabled, an external kubeconfig for the public endpoint. Tenants and CI systems can retrieve them through the console, CLI, or `kubectl`.
- **Independent break-glass credentials.** The admin kubeconfig does not rely on platform SSO. Access still requires an available endpoint, network path, and control plane.

Worker pools

- Configure CPU, memory, disk, OS image, and replica count for each pool. A cluster can have several pools for different workloads.
- Prepared, versioned worker images are matched to the Kubernetes version.
- **Per-pool autoscaling** uses `minReplicas` and `maxReplicas` bounds, plus a maximum change per scaling event.
- Rolling operations honor PodDisruptionBudgets during drains, up to the configured drain timeout.



The cluster summary brings endpoint access, control-plane readiness, worker capacity and security status into one tenant-facing view.

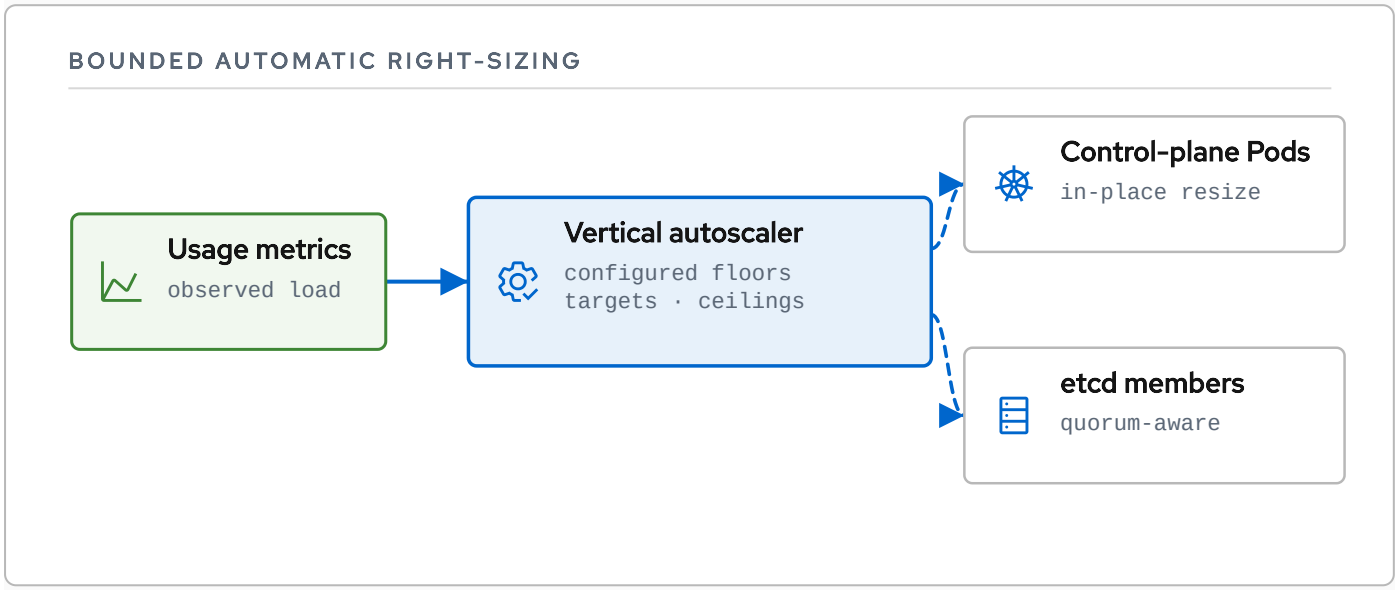


Worker-pool autoscaling is configured independently per pool, including minimum and maximum replicas and whether idle nodes may be removed.

Control-plane and etcd right-sizing

Kube-DC adjusts tenant control-plane resources within configured bounds and available platform capacity. VerticalPodAutoscalers are enabled by default for control-plane pods and etcd members:

- Resource targets follow observed load between configured floors and ceilings. Small clusters stay small, while growing clusters receive more API server and etcd capacity.
- Scaling is quorum-aware by design: recommendations are applied in-place where possible, and etcd resizing respects member quorum. Single-replica datastores use a conservative initial-assignment mode.



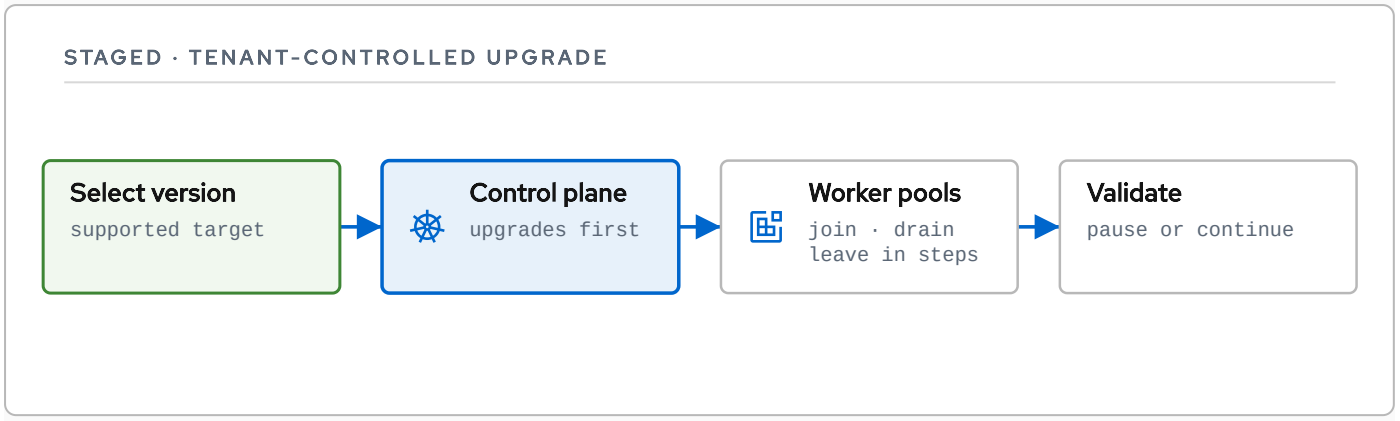
Usage metrics feed bounded recommendations to the control plane and datastore; the platform applies them in place where possible and preserves etcd quorum.

Upgrades

Upgrades are staged and tenant-controlled:

1. The tenant (or platform team, by arrangement) sets the target version.
2. The **control plane upgrades first**, without replacing worker nodes.
3. **Worker pools roll in steps** — new nodes join, old nodes drain and leave, pool by pool, with PodDisruptionBudgets honored during drains.

The control plane and workers move separately. Tenants can pause after a stage, validate the cluster, and continue. Supported version skew follows upstream Kubernetes policy.



The tenant selects a supported target; the hosted control plane moves first, worker pools roll in bounded steps, and the tenant can validate between stages.

Snapshots and restore

- **Scheduled etcd snapshots** are enabled automatically when the project's backup bucket exists (daily by default; schedule and retention configurable per cluster), written to the project's S3-compatible storage.
- **On-demand snapshots** at any time — from the console or by triggering the snapshot job.
- **Optional envelope encryption** of snapshots with a tenant-scoped key reference; without it, snapshots are stored unencrypted.
- **Self-service restore**: pick a snapshot, trigger the restore, and the control plane returns with the cluster's API state rolled back to the snapshot. The tenant API is unavailable during restore.

Scope and current limits, stated plainly:

- Etcd snapshots protect **Kubernetes API state**; they do not protect PersistentVolume contents — protect application data through its owning service (database backups, volume snapshots, object storage).
- Restore currently supports **single-replica etcd datastores**; multi-replica restore is on the roadmap. For clusters using the default three-replica dedicated datastore, restore is an assisted operation until then.

Isolation and security

- Control planes run on a platform-internal network with no route from tenant networks; workers and workloads live in the tenant project's VPC.
- Each cluster has its own PKI — certificates are per-cluster, rotated by the platform.

- Inside their cluster, tenants hold full cluster-admin: their own RBAC, CRDs, operators and admission configuration, independent of the platform's project roles.
- The cluster inherits the project's network boundary: reaching a cluster's workloads from outside happens only through the exposure the tenant configures (LoadBalancer services, HTTPS routes).

Observability

When the observability stack is enabled, managed-cluster control-plane logs and events appear in the owning organization's Grafana. Tenants can inspect their API servers and controllers, while the platform team monitors all control planes centrally. The operator defines coverage and retention.

Responsibilities

Concern	Your platform team (via Kube-DC)	Tenant
Control-plane operation, PKI, datastore		–
Control-plane/etcd sizing	automatic, within configured bounds	–
Worker pool sizing and autoscaling bounds	–	
Kubernetes version and upgrade timing	Controllers execute the staged rollout	selects target and timing from supported versions
Etcd snapshots	Controllers execute scheduled snapshots	policy, on-demand runs, restore initiation
Workloads, in-cluster RBAC, CRDs, operators	–	
Application data protection	–	via managed services
Network exposure of cluster workloads	Platform provides EIPs/routes	configures

Virtual machines

Run existing virtual-machine workloads beside Kubernetes applications.

Virtual machines use the same projects, networks, quotas, and dashboards as containers. Compatible line-of-business systems, virtual appliances, and Windows servers can move onto the platform without a separate virtualization control plane.

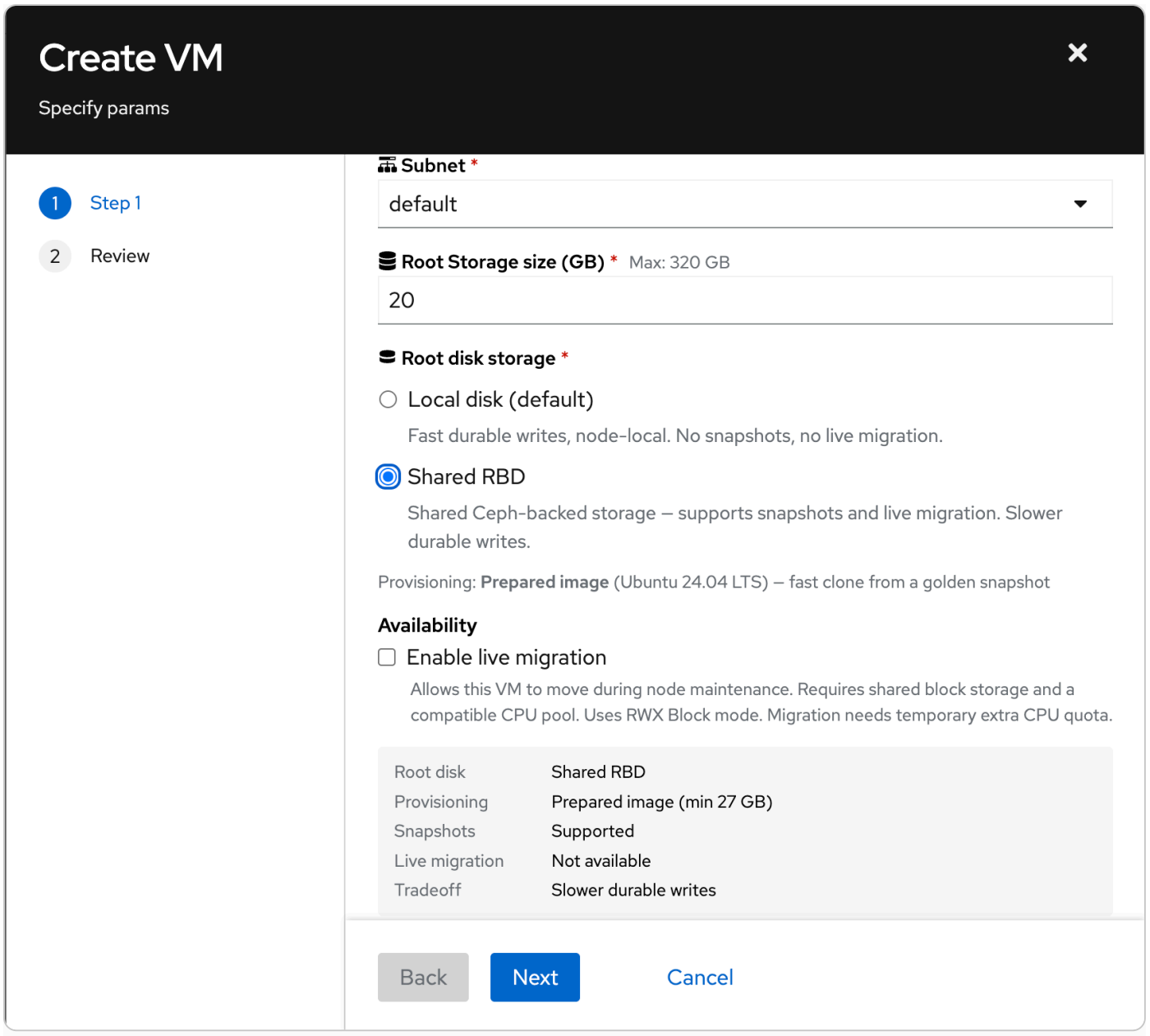
Guest support

- **Linux**: the prepared-image catalog identifies the validated distributions and versions (Ubuntu, Debian and other cloud-image distributions).
- **Windows**: provisioned from prepared images containing the required VirtIO drivers and QEMU guest agent, via the platform's Windows image pipeline — see the published Windows documentation.
- **Bring your own image**: import qcow2/raw disk images from HTTP or S3 sources into a project; guest and driver compatibility validation is yours.

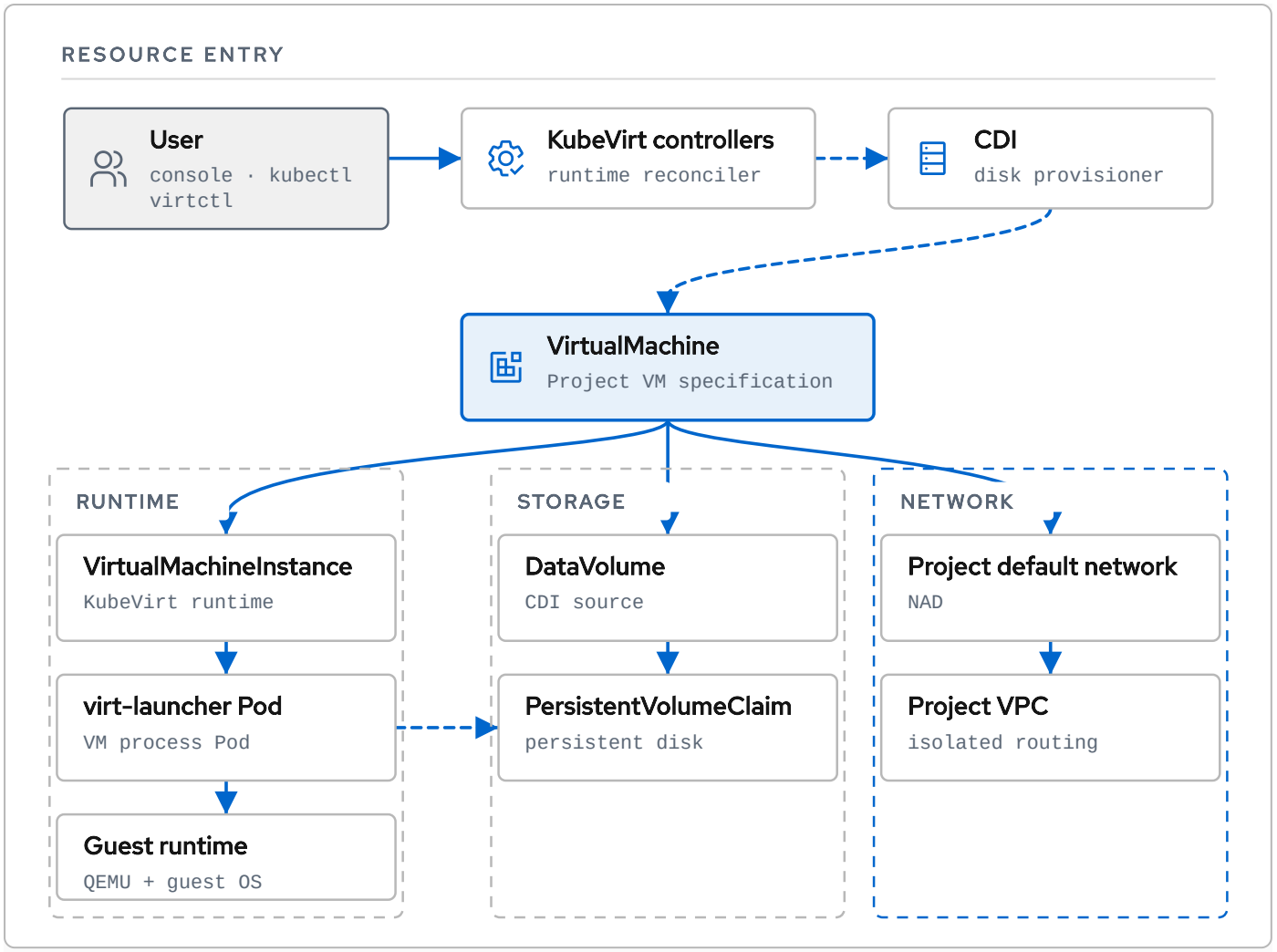
Provisioning: prepared images and instant clones

Each project carries a catalog of prepared OS images published as storage snapshots. Creating a VM clones the snapshot — the root disk is created without a per-VM image download or data copy; completion depends on your hardware, storage and current load. VMs are declared as manifests (or created in the console):

- CPU cores, memory and disk size per VM; resizable as the workload grows (disk grows only).
- **Cloud-init** for first-boot configuration: packages, users, scripts.
- **SSH-key injection** from the project's managed keypair — no password distribution.
- **Interactive console** in the web UI for installation, rescue and break-glass access.



VM creation makes storage durability and mobility trade-offs explicit before provisioning, including prepared-image cloning and the shared-storage requirement for live migration.



KubeVirt reconciles the VM runtime, CDI provisions the persistent disk, and Kube-OVN attaches the VM to its Project VPC; all branches originate from the Project VirtualMachine resource.

VM profiles: standard and migratable

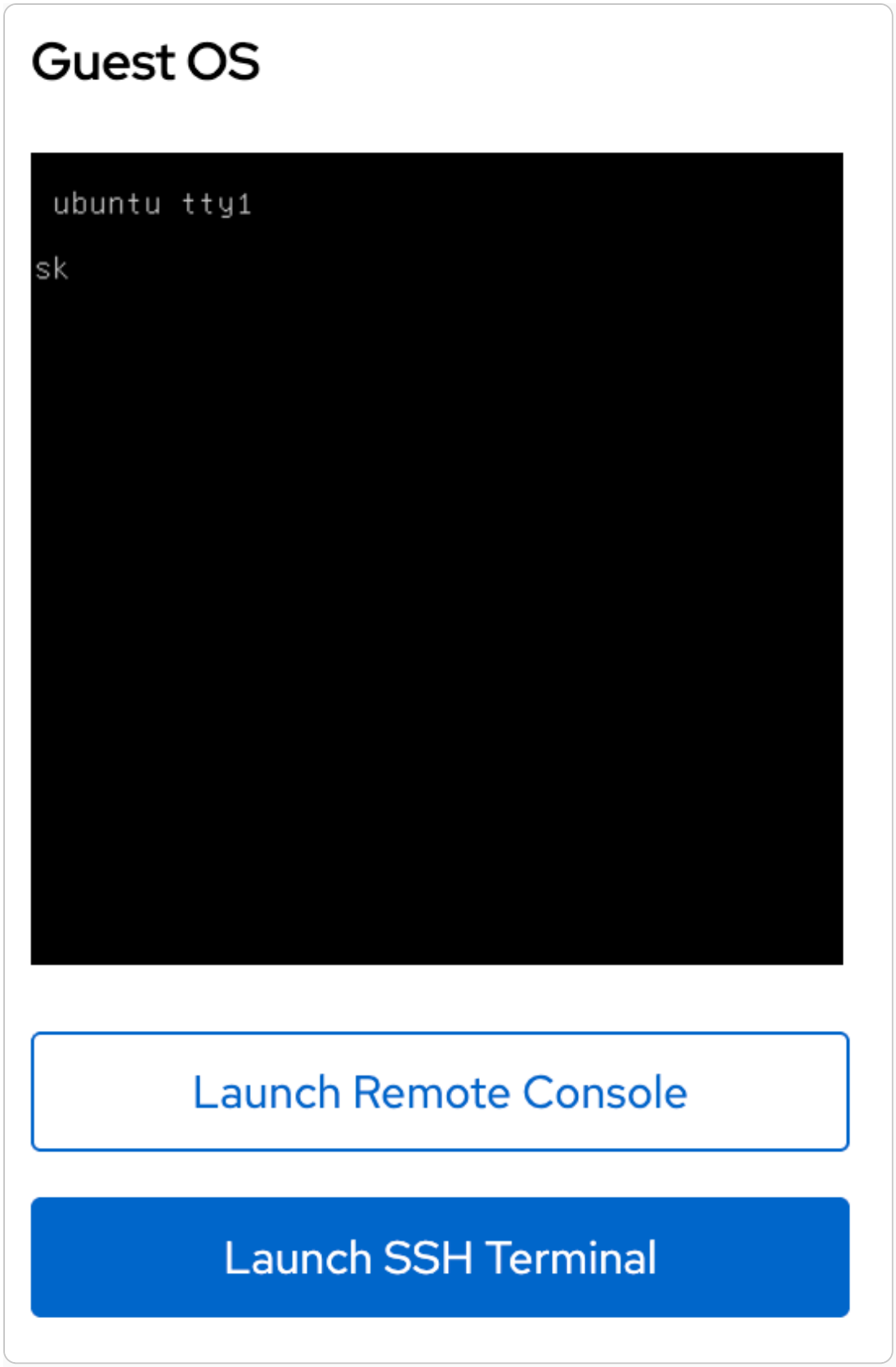
Two documented profiles, selected at creation:

	Standard profile	Migratable profile
Root disk	Node-local or replicated block	Shared block storage (read-write-many)
Host maintenance	VM restarts on another host	**Eligible for live migration** during host drains
Typical use	Stateless or restart-tolerant workloads	Long-lived servers, stateful systems in VMs

The Migratable profile pins a common CPU model across the migration pool. A VM is eligible for live migration when its disks use the configured shared block tier and a CPU-compatible destination host has sufficient capacity; failed or ineligible migrations may require a restart.

Lifecycle and operations

- Start, stop, restart and delete from the console, `kubectl` or the CLI.
- **Snapshots of running VMs** – online snapshots via the standard Kubernetes snapshot API; install the QEMU guest agent for filesystem-consistent captures.
- Eviction behavior per VM: live-migrate or restart on drain.
- VM manifests are declarative Kubernetes resources – keep them in Git like the rest of your infrastructure.



The VM detail view provides browser-based console and SSH entry points for installation, rescue and break-glass access.

Networking

VMs attach to the project's VPC like any pod:

- Private addresses in the project subnet; reachable from the project's other workloads.
- **Floating IPs** give a VM its own external address — inbound and outbound — without a load balancer.
- LoadBalancer services and HTTPS routes can front VM-hosted services exactly as they front pods.
- With **VLAN attachment**, VMs talk layer-2 to existing datacenter equipment — the consolidation path for systems that expect to sit on a specific network segment.

Storage

- Root and data disks as block volumes; disk expansion online.
- Additional volumes attach and detach as Kubernetes resources.
- Volume snapshots and clones for copies, templates and test environments.

GPU guests

A VM can be given a whole physical GPU (dedicated passthrough) where the platform team has enabled and qualified GPU support — a stronger boundary than software GPU sharing, at the cost of live migration. See the GPU services datasheet for products, entitlements and the qualification model.

Responsibilities

Concern	Your platform team	Tenant
Virtualization layer, hosts, live-migration machinery	□	–
Prepared image catalog	□ publishes	□ may bring own images
VM sizing, lifecycle, in-guest configuration	–	□
Guest OS patching and hardening	–	□
VM data protection	Snapshot APIs provided	□ snapshots + in-guest/enterprise backup
Network exposure	IPs and routes provided	□ configures

What to know before migrating

Disk import from other virtualization platforms is supported (qcow2/raw); migration itself is a planned operation — inventory, guest preparation (VirtIO drivers for Windows), disk conversion, cutover — not an automatic "lift". VMs do **not** need to be refactored into containers: the point of this function is that they run as VMs, on the same platform your containerized services use.

Managed databases

Teams provision PostgreSQL and MariaDB without operating the database engine themselves.

A tenant creates a database from a short manifest or a console form. Platform controllers manage the engine, replication, and backups. The tenant receives a stable endpoint and a Kubernetes Secret containing the credentials.

Engines and provisioning

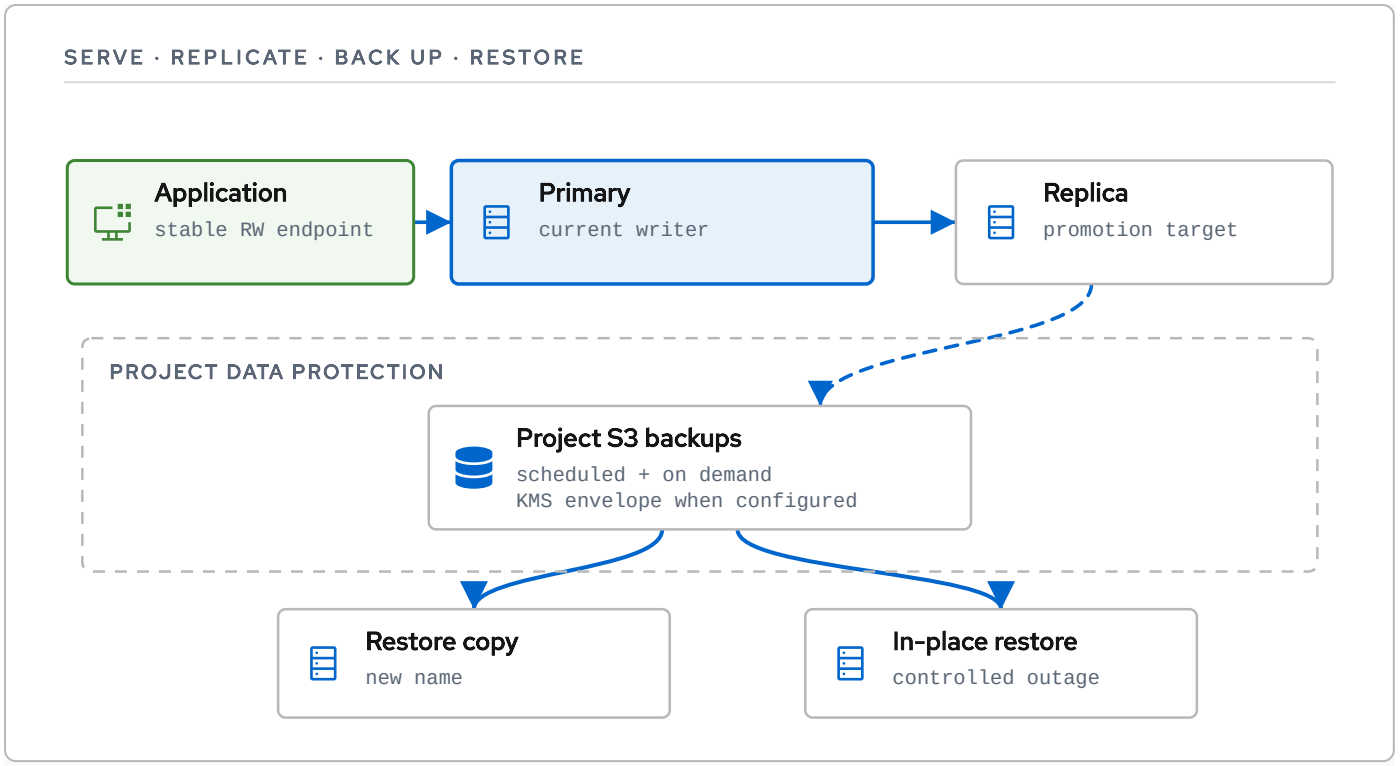
PostgreSQL and MariaDB, in the versions published by the platform's catalog. A database is one resource:

```yaml
apiVersion: db.kube-dc.com/v1alpha1
kind: KdcDatabase
metadata:
  name: orders
  namespace: acme-shop
spec:
  engine: postgresql
  version: "16"
  databaseName: orders
  username: app
  cpu: "1"
  memory: 2Gi
  storage: 20Gi
  replicas: 2
  backup:
    enabled: true
    schedule: "0 2 * * *"
    retentionDays: 7
```

The platform provisions the engine, storage and credentials from the manifest, exposing a stable in-project endpoint (`orders-rw.<project>.svc` for PostgreSQL; a primary-routing endpoint for MariaDB) and an auto-generated credentials Secret.

Topology and failover

- **1 replica** — standalone instance.
- **2+ replicas** — engine-level replication: PostgreSQL streaming replication; MariaDB primary–replica replication. With sufficient independent failure domains and capacity, instances schedule across separate hosts, and after an eligible primary failure the operator promotes an available replica automatically — recovery behavior depends on engine state and deployment topology, and the read-write endpoint follows the current primary.



Applications keep one read-write endpoint while engine replication and operator promotion protect service continuity; scheduled and on-demand backups land in Project S3 for new-name or in-place restore.

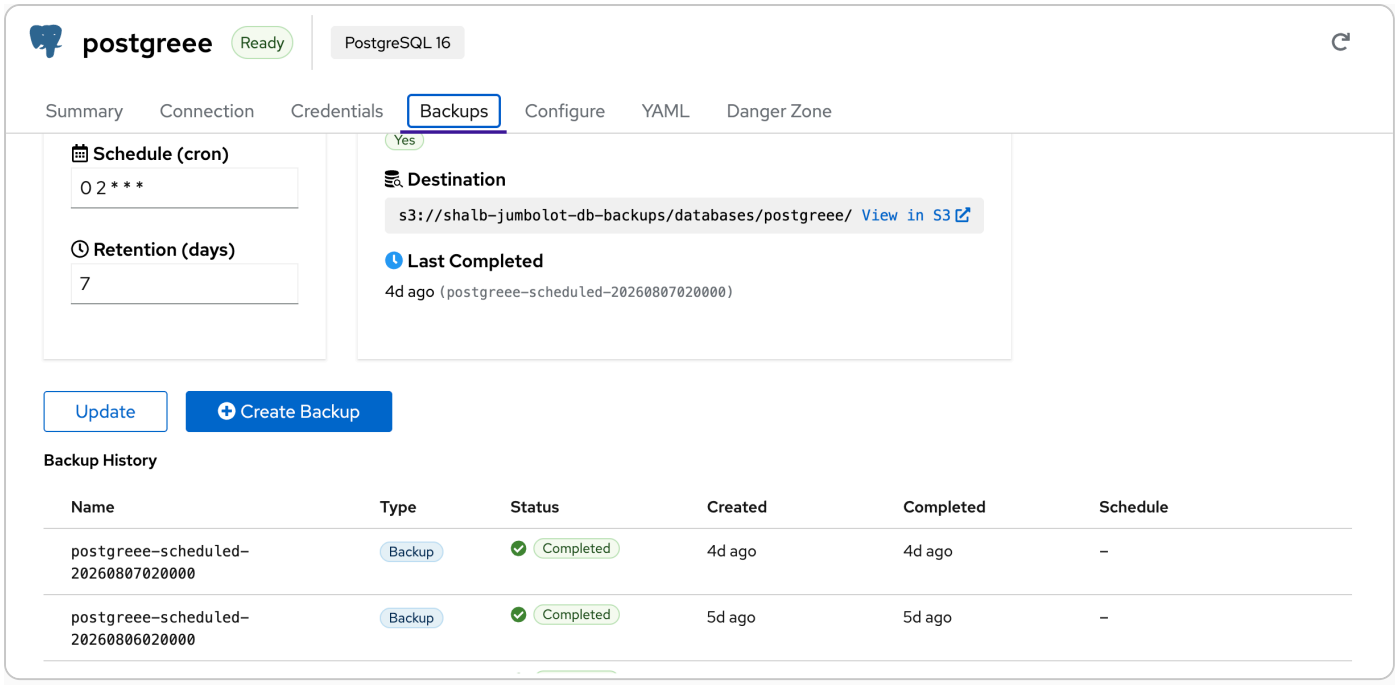
A practical note for multi-replica databases: the resource reports `Ready` when the service is available; check instance readiness before assuming full redundancy after creation or maintenance.

Credentials

- Connection details and passwords are generated by the platform and delivered as Kubernetes Secrets — applications consume them as environment variables or mounted files; nothing is typed or emailed.
- **Credential-rotation policies** rotate static passwords on a schedule and project the current credential into a stable Secret your workloads reference.

Backups

- **Scheduled backups** per database — cron schedule and retention days in the manifest — written to the project's S3-compatible bucket.
- **On-demand snapshots** any time, from the console or by creating a backup resource.
- **Per-database encryption**: reference a project KMS key and backups are envelope-encrypted with a key that never leaves the platform's secrets backend.
- For PostgreSQL, point-in-time recovery is available within the backup window where continuous archiving is enabled.



Database protection is visible and operable from the service itself: schedule, retention, destination, on-demand backup and completion history share one view.

Restore

Two documented paths:

- **Restore into a new database** (recommended): create a new `KdcDatabase` with `spec.restoreFrom` naming the backup — verify the recovered data, then switch applications over. The source database keeps running.
- **In-place restore** (destructive): trigger by annotation on the existing database; the platform re-bootstraps it from the chosen backup. Plan a maintenance window — current data is replaced.

Configuration and lifecycle

- Resources (CPU, memory) and engine parameters adjustable per database; some changes restart instances or trigger failover — applications should reconnect and retry.
- Storage grows online (increase only).
- Version changes are maintenance operations: verify a current backup first, then move.

Responsibilities

Concern	Your platform team	Tenant
Engine operation, replication, failover	(controllers)	–
Backup execution and storage	Controllers execute scheduled backups	schedule/retention choice, on-demand runs
Restore	Mechanism provided	initiates, verifies recovered data
Credentials delivery and rotation		application usage hygiene
Schema, queries, application-level integrity	–	
Off-platform backup copies	S3 endpoints provided	via enterprise backup

Networking and VLAN attachment

Give each project an isolated VPC, with controlled paths to public services and existing datacenter networks.

Kube-DC creates a VPC for each project and connects it to the uplinks, address space, and VLANs managed by your datacenter team. Tenants can expose services without gaining control of the underlying network fabric.

Per-project VPCs

Every project receives its own VPC and private subnet at creation. The software-defined network does not configure private routes between projects by default. A service becomes reachable from another project only when it is deliberately exposed or the operator adds a route. Pods, VMs, and managed-cluster workers in the same project share the project network and can reach one another directly.

Address model

Object	What it does
Shared egress IP	Every project gets outbound connectivity through a shared external address
External IP (Eip)	A dedicated external address, allocated automatically per LoadBalancer service or explicitly for a purpose
Floating IP (Fip)	Maps an external address to an individual VM — inbound and outbound — without a load balancer

Address pools are defined by the platform team from your ranges; tenants consume them within quota.

Exposing services

Two documented paths, chosen per service:

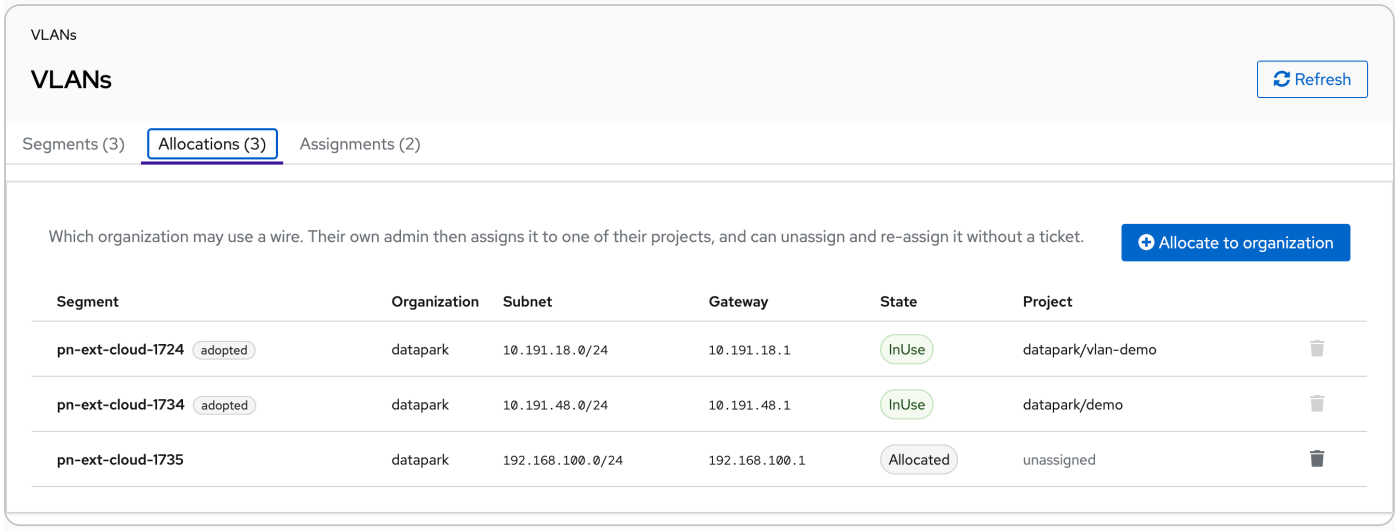
- **HTTPS route (recommended for web/API/gRPC):** one annotation on a `LoadBalancer` Service publishes it at a hostname with a TLS certificate issued by the platform's configured issuer — no ingress controller for tenants to run, no certificate handling. Uses the platform's gateway, your routable address pools and DNS.
- **Direct LoadBalancer (any TCP/UDP):** the Service receives a dedicated external IP for protocols and ports beyond HTTP.

Managed-cluster API endpoints follow the same model: private in-VPC by default, public HTTPS through the platform gateway on request.

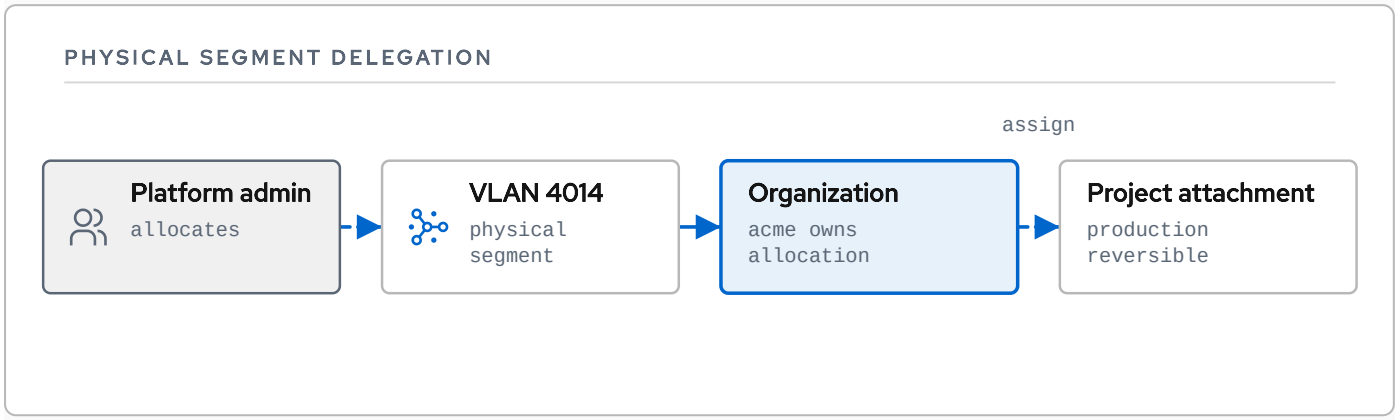
VLAN attachment: layer-2 into your datacenter

Where the datacenter fabric trunks delegated VLANs to the platform nodes and the operator configures the bridge and VLAN pools, a project's network can be bridged onto an existing datacenter VLAN.

- **Per-organization VLAN pools**: the platform team delegates ranges of VLAN IDs to organizations; tenants allocate segments from their pool self-service, without tickets and without being able to touch another organization's VLANs.
- Workloads — VMs and pods — gain a leg on the physical segment: lab instruments, storage networks, legacy systems, partner links, anything that expects layer-2 adjacency.
- Addressing on the attached segment is yours: static, your DHCP, or platform-managed.



Platform operators delegate physical segments to an Organization; its administrators can then assign or reassign those allocations to Projects without changing the underlying fabric.



Platform administrators allocate a physical segment to an Organization; Organization administrators then attach, detach, and later reattach that allocation to one of their Projects.

Traffic control

- Egress restrictions and allowlists are operator-configured platform controls — tenants request exceptions; the platform team owns the policy.
- Network boundaries are a property of the VPC layer: they are not tenant-configurable objects inside a project.

- North-south exposure happens through the objects above — an address or route a tenant deliberately created.

Responsibilities

Concern	Your platform team	Tenant
Fabric, uplinks, address ranges, VLAN delegation	□	–
VPC creation and isolation	□ automatic	–
Egress policy and allowlists	□	requests exceptions
Service exposure (LBs, routes, FIPs)	Mechanisms provided	□ configures per service
VLAN segment allocation from the org pool	Pool delegation	□ self-service within pool
DNS for published hostnames	Platform zone provided	□ own domains via CNAME

Storage and object storage

Provide block, shared, and S3-compatible storage through standard Kubernetes APIs.

Kube-DC uses Ceph on your hardware for its storage layer. The platform team defines storage classes, topology, and replication. Tenants request storage through Kubernetes APIs within their assigned quotas.

Block storage

- PersistentVolumes for containers and disks for VMs from a replicated Ceph RBD pool.
- Online volume expansion.
- Storage classes are operator-defined: you decide which tiers exist (replicated, node-local, performance) and what backs them — the platform runs on the disks and failure domains you give it.

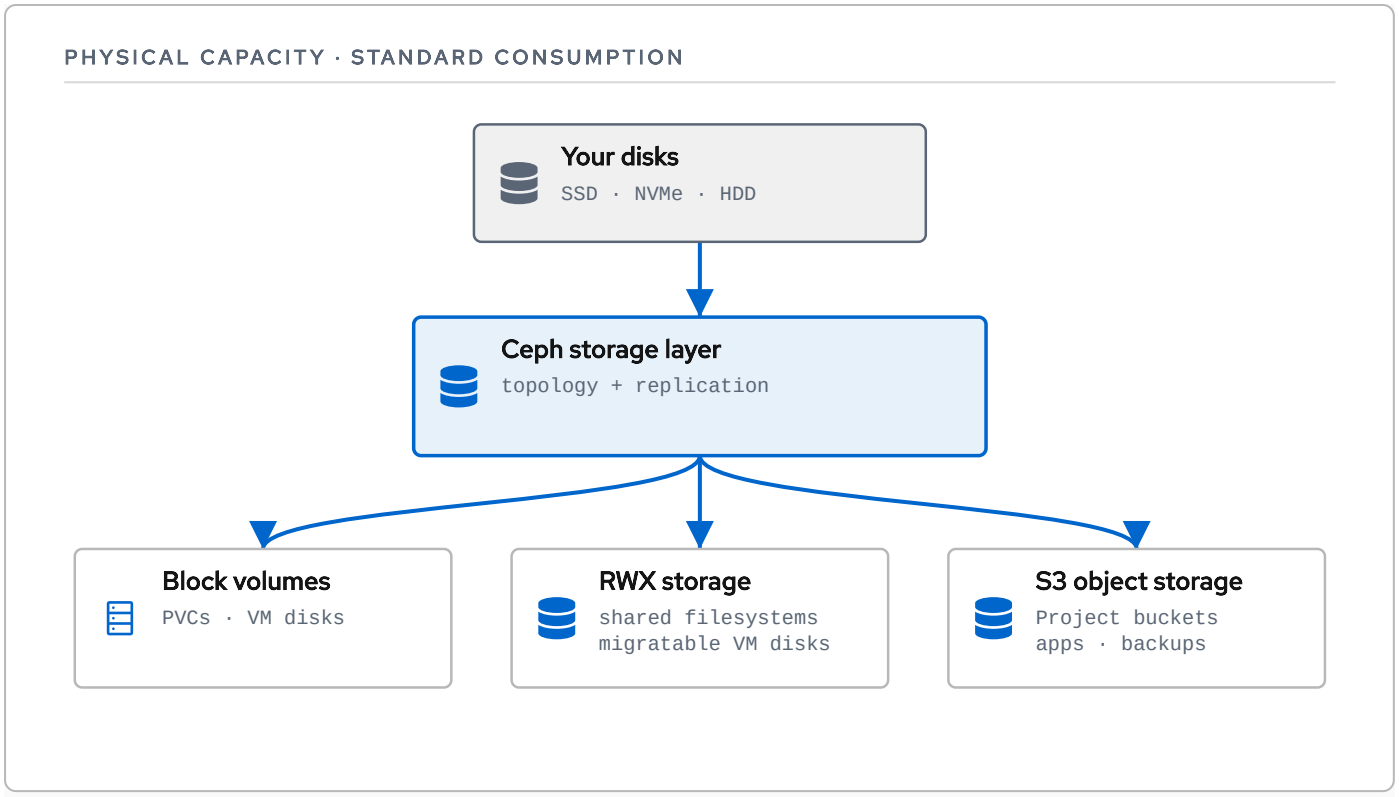
Shared (read-write-many) storage

Operator-defined RWX storage classes serve the cases single-writer volumes can't — each with the access mode and backend appropriate to its semantics:

- **Multi-pod shared filesystems** — several pods writing the same volume (content stores, shared uploads, legacy apps that expect a shared disk).
- **Migration-eligible VM disks** — a VM whose disk is on the shared block class is eligible for live migration between hosts (see the Virtual Machines datasheet).

S3-compatible object storage

- **Per-project buckets** provisioned through a standard `ObjectBucketClaim` — credentials arrive as a Secret and ConfigMap the workload mounts; no manual account management.
- The platform's own services use the same storage: database backups and managed-cluster snapshots land in per-project buckets.
- Typical tenant uses: application assets, exports, log archives, backup targets.



The operator's disk topology feeds Ceph, which publishes block, shared and S3 services to Project workloads through standard Kubernetes storage interfaces.

Snapshots, clones and images

- **Volume snapshots** through the standard Kubernetes snapshot API — point-in-time copies of workload and VM volumes.
- **Instant clones**: new volumes from snapshots without duplicating data — the mechanism behind the prepared-OS-image catalog and VM provisioning without image downloads.
- Prepared images are published per project as snapshots.

Capacity and quota

Storage consumption counts against organization plans and project quotas like CPU and memory. Object-storage capacity is governed per organization and requires its own capacity planning. How many disks, which tiers, what replication — these are platform-team decisions made against your hardware; capacity and durability are properties of the topology your team designs, validated in the architecture review.

Data protection posture

Storage replication protects against component failure within the platform; it is not a backup. The protection stack is layered:

Layer	Mechanism	Owner
Component failure	Ceph replication per your topology	Platform team
Point-in-time copies	Scheduled jobs run by platform controllers; on-demand snapshots and restores initiated by tenants	Shared
Site/storage loss	Copied by your configured enterprise backup integration via S3 endpoints	Your backup team

Responsibilities

Concern	Your platform team	Tenant
Disks, tiers, replication, failure domains		–
Storage classes and quotas	defines	consumes within quota
Volume/bucket provisioning	Automatic via API	requests
Snapshots and clones	APIs provided	initiates
Backup of application data	S3 + snapshot APIs provided	owns

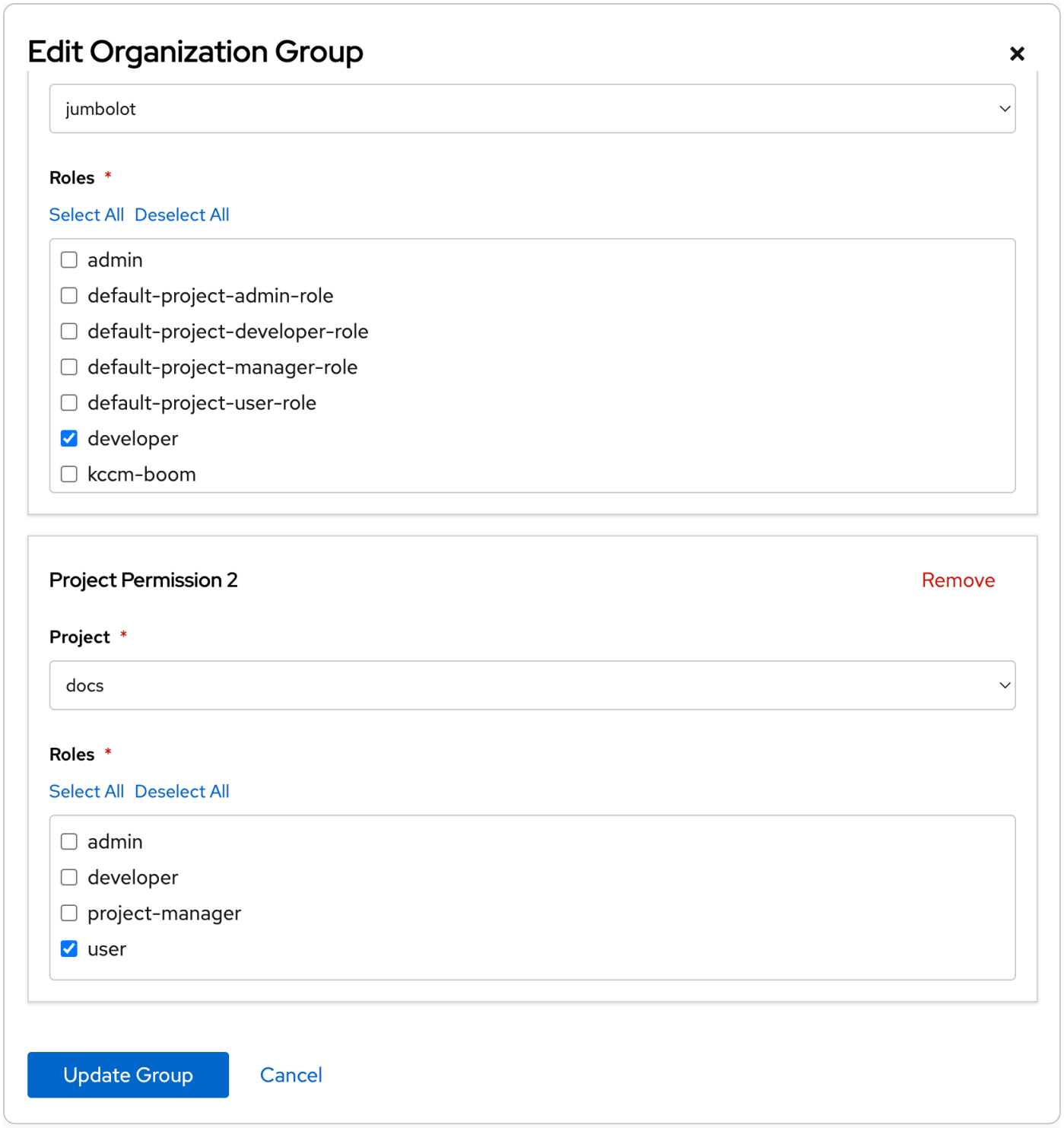
Security, identity, and keys

Separate tenant identity, permissions, networks, keys, and secrets at the boundaries where they are used.

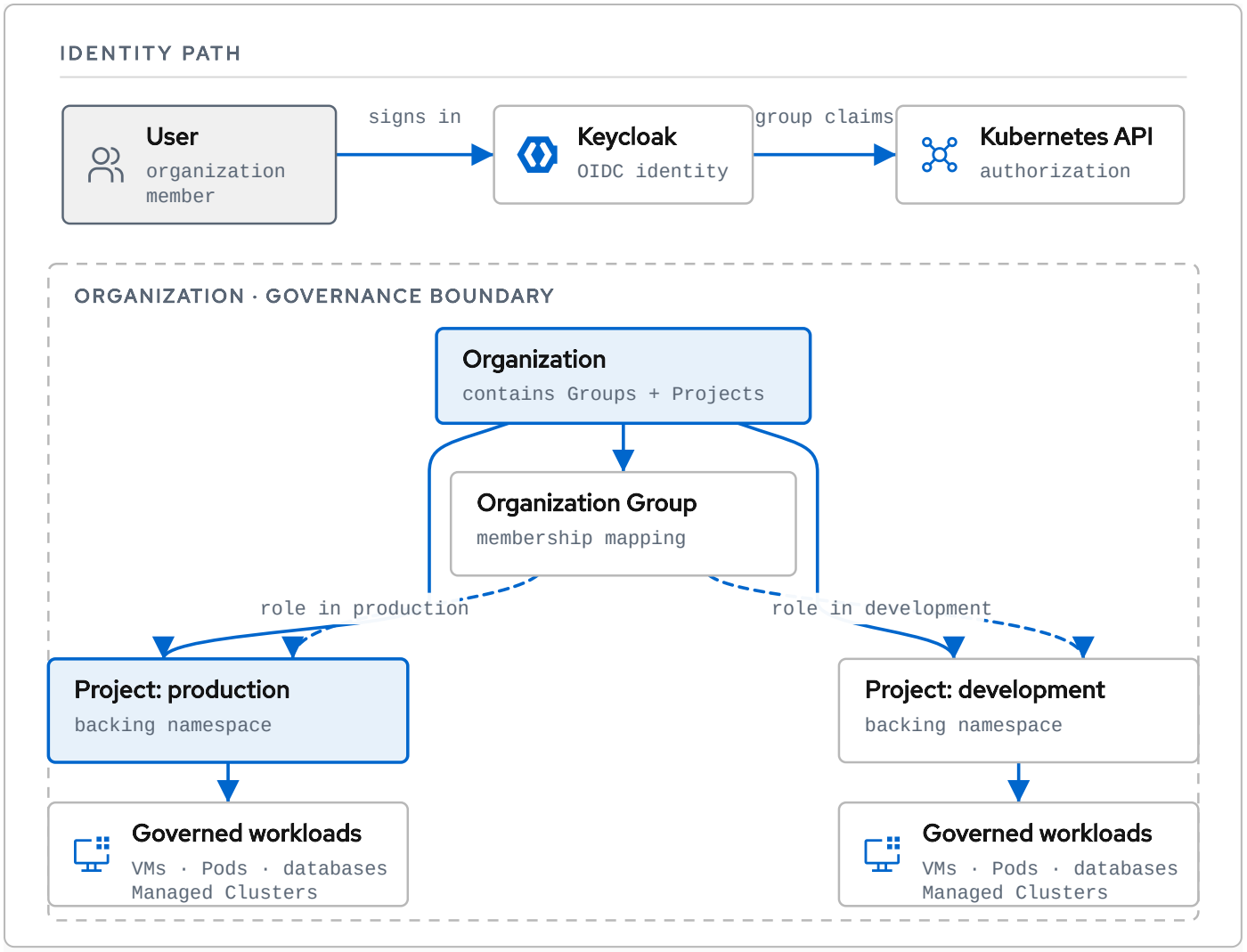
Kube-DC combines organization identity, project RBAC, VPC boundaries, and project-scoped keys and secrets. A different subsystem enforces each boundary, so tenant separation does not depend on a single control.

Identity and single sign-on

- **An identity realm per organization** (Keycloak): each tenant organization manages its own users, groups and login policies — isolated from every other organization's.
- **Single sign-on across every surface**: console, Kubernetes API and CLI authenticate against the same realm via OIDC. Platform-project kubeconfigs use organization OIDC credentials; managed-cluster admin/break-glass kubeconfigs use separate per-cluster credentials.
- **Group-based access**: organization groups map to project roles declaratively, so joiners/leavers are handled in the directory, not in per-project role edits.



Organization Groups map identity roles to Project-specific permissions, keeping access assignments explicit at each governed workload boundary.



Identity claims establish API access, while Organization Groups map roles independently into each Project; workloads and Managed Clusters remain governed by their authorized Project.

Project authorization

- **A curated, supported role set** per project — scoped to deploying and operating workloads. Cluster-scoped resources, CRDs and admission configuration are excluded by construction.
- **Interactive `exec` / `attach` into pods is excluded from the supported project roles** and denied by an admission policy at the API. One-off tasks run as auditable Jobs instead. Scope stated plainly: users permitted to create or modify workloads can still run code through those workloads, and the restriction does not apply inside tenant-administered managed clusters.
- Custom roles remain under platform-team control; the default roles are a baseline, not a ceiling.

Key management (KMS)

- **Per-project, purpose-scoped keys** (`application`, `backup`, …) backed by non-exportable keys in the platform's secrets backend (OpenBao Transit): encrypt/decrypt happens as an API operation — key material never leaves the backend.
- Used by the platform itself where it matters: managed-database backups and managed-cluster snapshot encryption reference tenant KMS keys.
- Envelope encryption is the documented pattern for large payloads.

Secrets management

- **A managed secret service per project**: store API tokens, credentials and configuration secrets outside Git, with lifecycle managed through the CLI and console.
- **Sync into Kubernetes Secrets**: selected values project into standard Secrets that workloads consume — applications stay platform-agnostic while the source of truth stays managed.

Certificates

- **Managed certificates** as a project resource: public trust via ACME or issuance from your organization's private CA — the resulting TLS Secrets are consumed by workloads directly.
- HTTPS routes get their certificates automatically (see the Networking datasheet); managed-cluster public endpoints likewise.

Platform-level enforcement

- Admission policies enforce tenant boundaries at the Kubernetes API — including the pod-access block and resource-placement rules.
- Network isolation is a property of the SDN (per-project VPCs), not of tenant-managed policy.
- Egress control and allowlists are operated centrally by the platform team.
- Platform configuration can be declaratively reconciled from Git.

Responsibilities

Concern	Your platform team	Tenant
Identity realms, SSO, login policy defaults	provisions	manages own users/groups
Project roles and admission policies		assigns members to roles
KMS backend and key custody		creates/uses keys
Secrets service	operates	owns contents
Certificates	Issuance machinery	requests/consumes
Workload and guest-OS security	—	

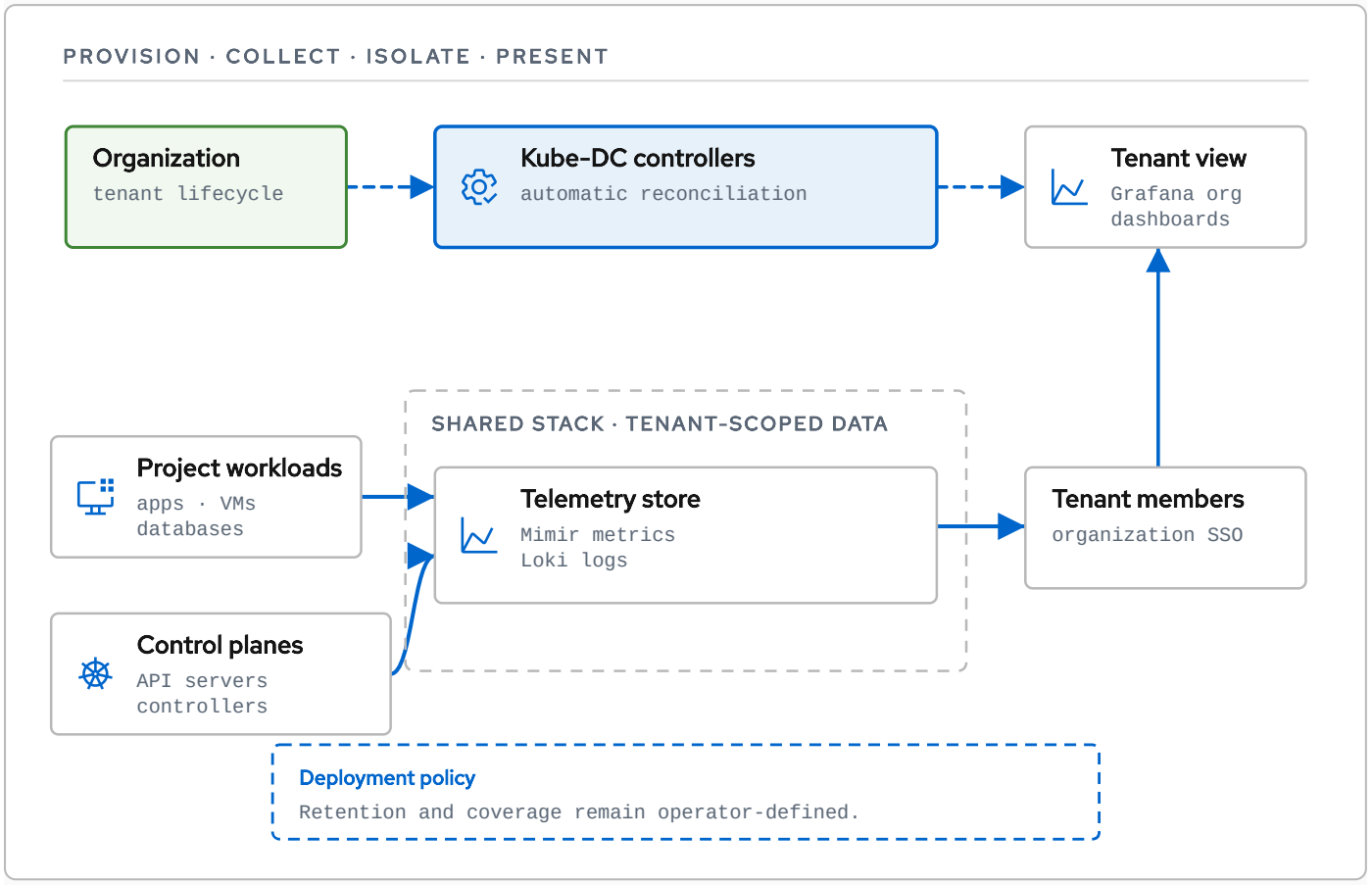
Observability

Each organization receives its own view of metrics, logs, and dashboards.

Kube-DC includes a multi-tenant observability stack based on Grafana, Mimir, and Loki. When an organization is created, platform controllers provision a Grafana organization, dashboards, a metrics tenant, and log routing. Teams can open Grafana and inspect their workloads without installing a separate monitoring stack.

What every organization gets

- **A Grafana organization of its own** – members sign in with the same SSO they use everywhere else and land in their organization's view.
- **Pre-provisioned dashboards** for the organization's projects and workloads – created and maintained by the platform's controllers, not hand-built per team.
- **Metrics** collected from the organization's workloads and served from the multi-tenant metrics store, scoped per tenant.
- **Logs** from the organization's workloads, routed per tenant and queryable from the same Grafana.
- **Managed-cluster control-plane telemetry**: the logs and events of an organization's managed Kubernetes clusters surface in its Grafana – API servers and controllers are watchable without filing a ticket.



Organization reconciliation provisions Grafana, metrics tenancy and log routing; workloads and managed control planes feed the shared data layer, while tenant-scoped queries return only that organization's data.

Tenant separation

The stack is shared; the data is not. Metrics and logs are separated per tenant at the data layer — queries from one organization's Grafana reach only that organization's data. The platform operates one observability system for all tenants instead of one stack per team, which is what makes day-one monitoring economically automatic.

The operator's view

The platform team monitors the platform itself from the same tooling:

- Cluster health, capacity, networking and component status, with alerting for platform operators.
- **Storage health**: Ceph dashboards and alerts for the storage layer, surfaced in the operator's monitoring and the administration console.
- Coverage and retention are operator-defined — the bundled defaults are a starting point, tuned to your capacity in the architecture review.

Responsibilities

Concern	Your platform team	Tenant
Observability stack operation and capacity		–
Per-organization provisioning (Grafana org, dashboards, tenants, routing)	automatic, by controllers	–
Retention and coverage configuration		–
Watching workload dashboards, acting on application signals	–	
Custom dashboards within the organization's Grafana	–	
Platform alerting and response		reports suspected platform issues

GPU services

Offer fixed GPU shares to containers or attach a complete GPU to a virtual machine.

Kube-DC presents GPUs as catalogued products with per-model entitlements and capacity reservations. It also documents the security and supply-chain controls required by the privileged components that run on GPU nodes.

Before enabling GPU services, the platform team qualifies the hardware, kernel, driver, and operator versions together. Shared GPUs and dedicated GPU VMs have separate controls, so an installation can offer either or both.

Shared GPU for containers

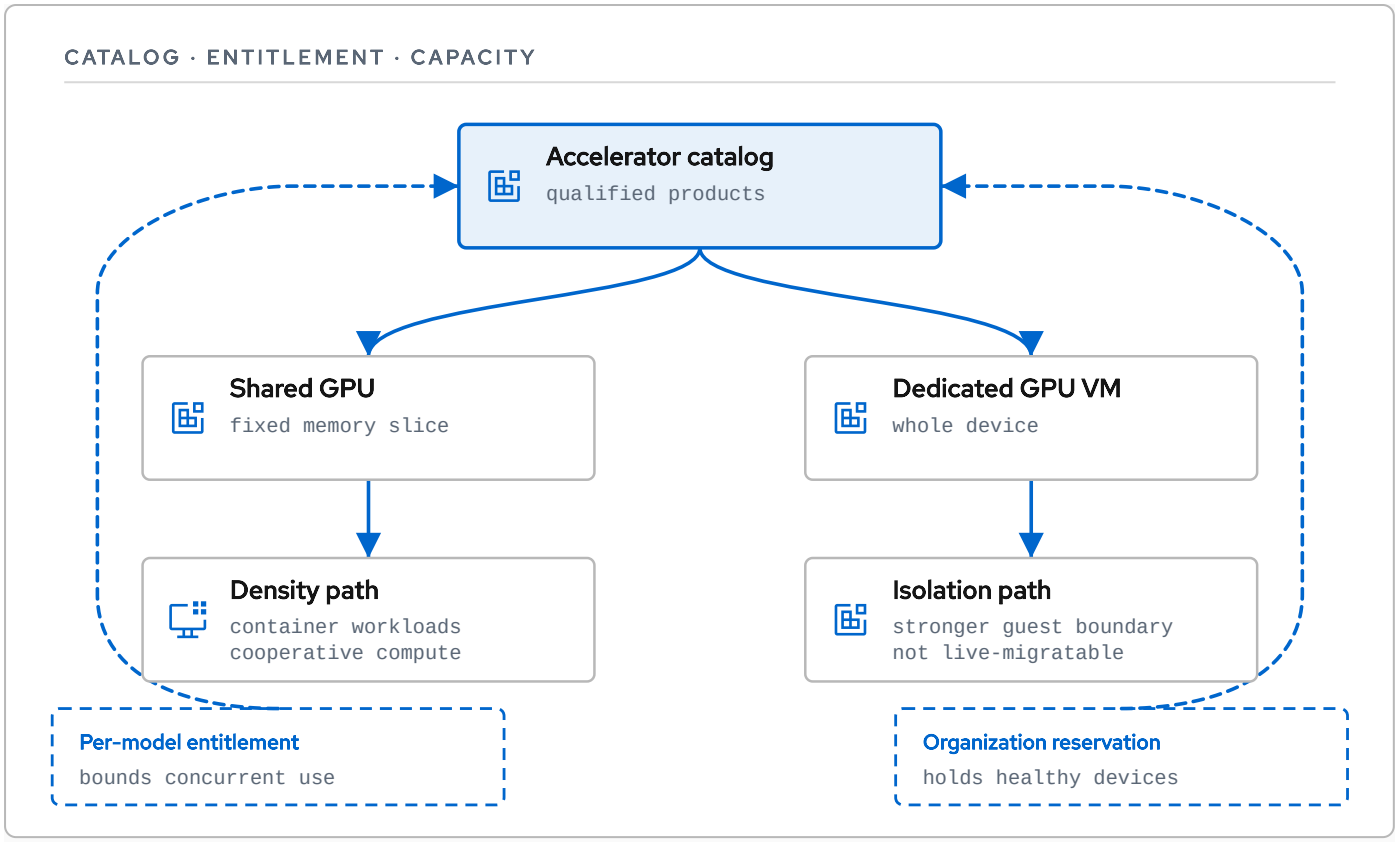
Several container workloads run on one physical GPU at the same time, each holding a fixed fraction of the device — the right fit for inference, notebooks, small training runs and batch scoring, where a whole card is waste.

- **Fixed-product profiles.** A shared product is a defined fraction of a specific GPU model (for example, an 8 GiB slice of an NVIDIA V100) published in the platform's accelerator catalog. Tenants request a product; they do not freely tune memory and compute.
- **The memory slice is enforced.** Inside the container, the GPU reports the slice size, not the physical card.
- **The compute share is cooperative.** Workloads are steered toward their share in steady state; startup and some library paths can briefly exceed it. Stated plainly: this is a density mechanism, **not a performance guarantee and not a security boundary** — for a hard boundary, use a dedicated GPU VM.
- **Standard Kubernetes consumption.** Workloads request GPU products through Kubernetes' Dynamic Resource Allocation — a manifest applied with `kubectl`, like any other workload. Shared GPU runs tenant workloads in production on the reference deployment today.

Dedicated GPU VMs

One whole physical GPU attached to one virtual machine, for workloads that need a stronger boundary than software sharing: an isolated guest with the full device.

- Whole-device passthrough to a Linux or Windows guest, from the deployment's qualified image and driver matrix.
- **Not live-migratable** — planned maintenance means shutdown and restart, and a later start can wait for capacity or receive a different physical device.
- Guest images and driver packages are qualified per deployment: the platform team publishes an approved package version and checksum before enabling GPU VM creation. The published matrix names the qualified guests; anything outside it is qualified separately.



The accelerator catalog offers a density-oriented shared slice for containers and a whole-device VM path; model-specific quota bounds use, while a separate Organization reservation holds healthy capacity.

Entitlements, quota and reservations

Two different things, deliberately distinguished:

- **Entitlement (quota)** – each GPU model is metered independently in organization and project quota, so a team's V100 allowance never consumes another model's headroom. An entitlement bounds concurrent use; it does **not** reserve a physical device, so a valid workload can legitimately queue when all matching GPUs are busy.
- **Capacity reservation** – an operator-held amount of healthy physical capacity set aside for one organization, tracked in a separate capacity ledger. This is what a department buys when queuing is unacceptable. Reservations are fungible: the guarantee is the number of healthy devices, not a specific card.

Assigning a GPU add-on does not by itself create a reservation – the distinction is enforced in the product, not just in the documentation.

Node modes and day-2 operations

A GPU node runs in one mode at a time – shared-container or dedicated-VM passthrough – because the two require different host configuration.

- **Holder-safe mode transitions**: the `kube-dc` CLI performs the day-2 switch between modes with an explicit safety contract – creation gates closed, GPU holders inventoried, node cordoned, fleet state and live state in agreement before anything moves.

- **Upgrade gating**: GPU nodes couple the PCI device, host kernel, Kubernetes distribution, driver, GPU operator and monitoring exporter into **one tuple** that must be validated together — upstream support for each component separately is not sufficient. The CLI provides a read-only pre-upgrade gate; the documented sequence qualifies a canary node and proves rollback before a fleet-wide plan.

Security and supply chain

GPU drivers, device plugins, schedulers and monitoring agents run privileged enough to compromise a node, so version selection and image provenance are part of the security boundary — not routine application upgrades.

- **A published threat model** covers shared containers, dedicated VMs, catalog and discovery, quota, billing, installation, node-mode transitions, monitoring and the user interfaces — including the invariants the design preserves (a tenant cannot mint GPU entitlement by editing ordinary resources).
- **A qualified installer tuple**: an enabled installation accepts one indivisible, version-pinned component set, with runtime audit and a promotion gate — not "latest".
- Admission policies force tenant GPU requests into the exact catalog product, and per-model quota bounds the count, so entitlement cannot be exceeded by hand-written manifests.

Responsibilities

Concern	Your platform team	Tenant
Hardware, drivers, GPU operator, qualification of the component tuple		–
Enabling GPU capabilities per cluster and per project	(gated, after qualification)	–
Catalog products (which slices and models exist)	defines	requests
Entitlements and reservations	grants	consumes within quota
Node-mode transitions and GPU node upgrades		–
Workload design, drivers inside VM guests, performance tuning	–	