Kube-DC

A multi-tenant private cloud platform: virtual machines, Kubernetes, databases, storage, networking and GPU on your own servers.

Kube-DC runs as a Kubernetes management cluster on standard x86-64 hardware. Its controllers turn platform resources — organizations, projects, machines, clusters, databases, networks, keys — into the virtualization, networking, identity, storage and observability underneath. This datasheet describes the platform's functions, boundaries and interfaces.

TENANCY MODEL
Organizations and projects

WORKLOAD TYPES
VMs, containers, managed services

FOOTPRINT
From three x86-64 servers

CONTENTS

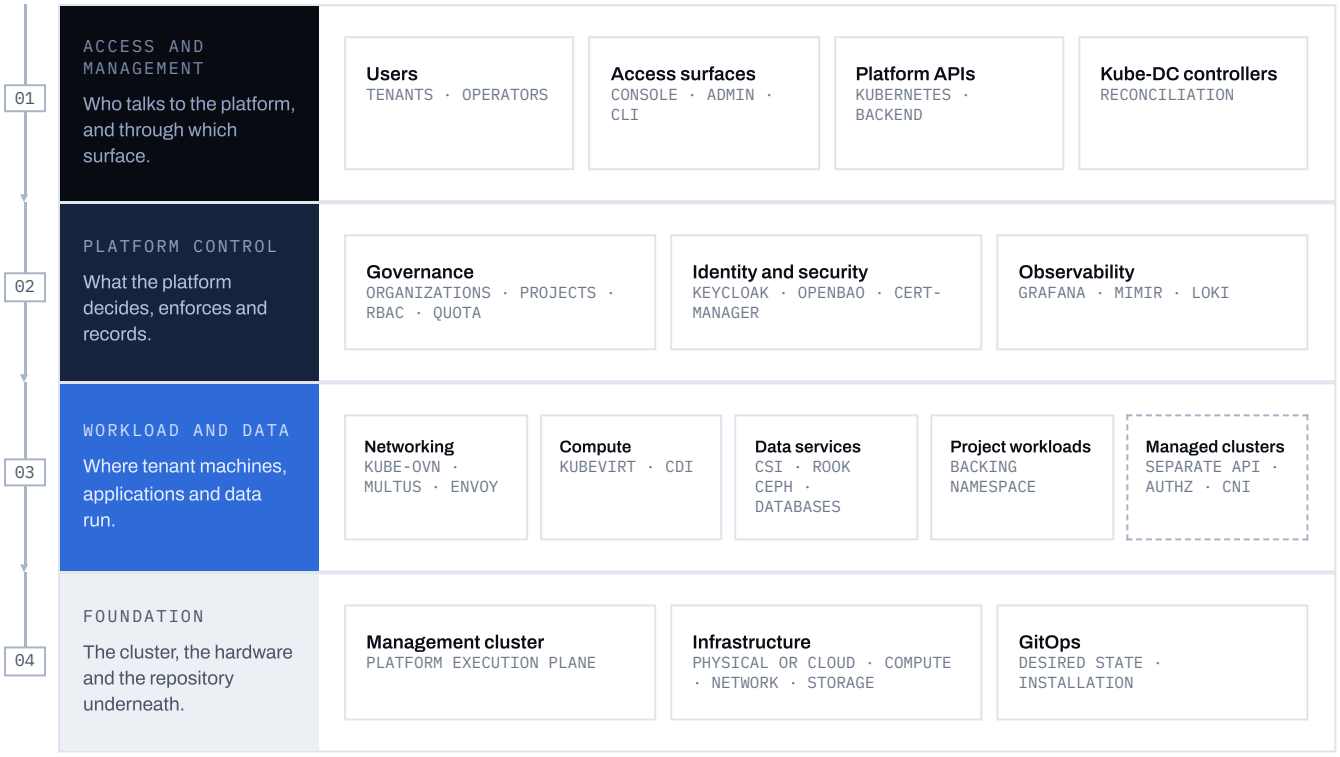
What is in this document

> Every capability in this document is part of one installation. Nothing described here requires a second platform, a separate hypervisor or an add-on product, unless a page states it explicitly.

Architecture

Kube-DC is a Kubernetes management cluster plus a set of controllers. Platform resources are Kubernetes API objects; the controllers reconcile them into networking, virtualization, identity, storage and observability configuration. Components are standard open-source projects, pinned per release.

ARCHITECTURAL LAYERS



ORGANIZATIONS AND PROJECTS GOVERN RESOURCES IN THE MANAGEMENT CLUSTER; MANAGED KUBERNETES CLUSTERS RETAIN SEPARATE API, AUTHORIZATION AND CNI BOUNDARIES.

PLATFORM COMPONENTS

Standard open-source projects, integrated and operated as one product, with versions 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.

KUBERNETES	KUBEVIRT · CDI	KUBE-OVN	ROOK-CEPH	KEYCLOAK
KAMAJI + CAPI	CLOUDNATIVEPG	MARIADB OPERATOR	OPENBAO	ENVOY GATEWAY
CERT-MANAGER	PROMETHEUS	MIMIR · LOKI	GRAFANA	FLUX

04 · MULTI-TENANCY

How multi-tenancy works

Kube-DC has two tenancy objects. An **organization** is a tenant: its own identity realm, members, capacity plan and usage record. A **project** is an isolated environment inside an organization: its own VPC, access rules, quota and resources. Every other boundary in the platform derives from these two.

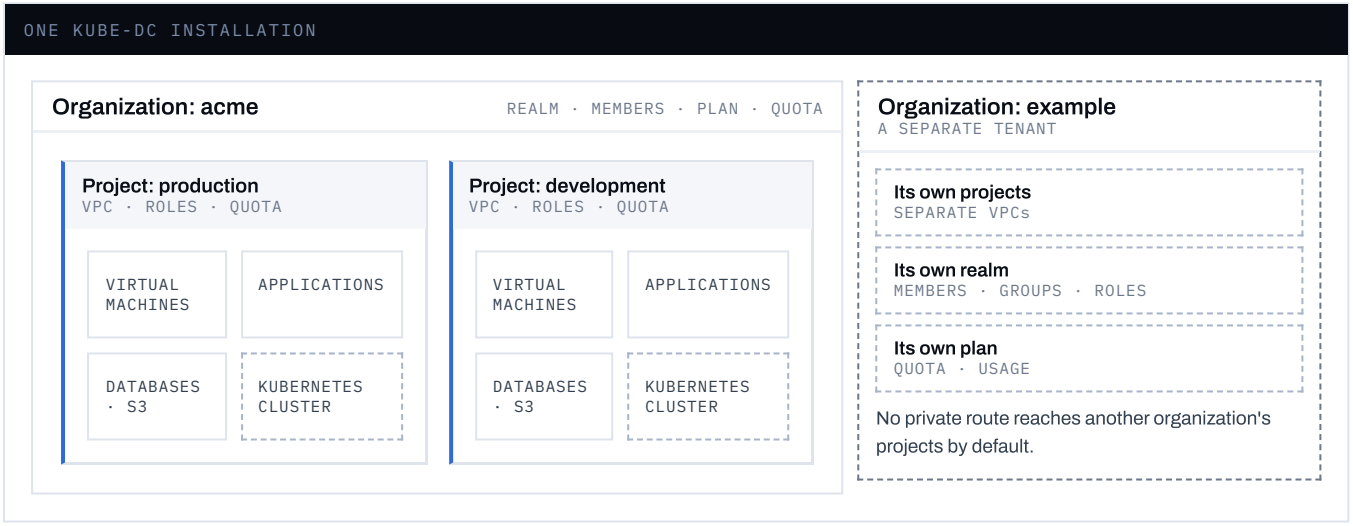
ORGANIZATION · THE TENANT

A dedicated identity realm, its own members and groups, a capacity plan, and a usage record. Two organizations share hardware and nothing else: no shared realm, no shared network, no shared storage credentials, no shared metrics.

PROJECT · THE ENVIRONMENT

A backing namespace, a dedicated VPC and subnet, project-scoped roles, an optional quota cap, and the resources created inside it. Production and development are separate projects, not naming conventions.

TENANCY HIERARCHY



WHAT EACH BOUNDARY ENFORCES

	ORGANIZATION	PROJECT
Identity	Dedicated realm, members, groups, OIDC single sign-on	Project-scoped roles bound to groups
Network	No shared address space between organizations	Dedicated VPC, subnet and router; explicit exits only
Compute and storage	Capacity plan across all its projects	Backing namespace, storage classes, object-storage credentials
Quota	CPU, memory, storage, pods, load balancers, GPU entitlements	Optional per-project cap inside the organization plan
Observability	Own dashboards, metrics space and log stream	Filtered views scoped to the project's workloads
Audit and billing	Audit log, recorded usage, optional subscription billing	Actions attributed to project and actor

05 · DEPLOYMENT PROFILES

Different solutions from one installation

Every installation carries the full function set described in this document. A deployment profile is not a product edition and not a separate install — it is which of those functions a given organization actually uses. A tenant that starts with one profile can use another the same day, without a platform change.

DEVELOPMENT PLATFORM

Application environments per team

- **Project namespace**
- **Managed PostgreSQL**
- **S3 bucket**
- **HTTPS endpoint**

An application, its database, its object storage and its published URL are declared in one set of manifests against the project's own credentials.

AI PLATFORM

GPU capacity as a governed resource

- **Shared GPU slices**
- **Passthrough GPU VMs**
- **Object storage for datasets**
- **Per-model entitlements**

GPU entitlements are enforced as quota per organization and per model, with optional reserved capacity for tenants that cannot queue.

PRIVATE CLOUD · IAAS

Self-service virtual machine estate

- **Linux and Windows VMs**
- **Per-project VPCs**
- **Floating IPs**
- **Snapshots and live migration**

Tenants create, size and rebuild their own machines; hosts are drained for maintenance while machines keep running.

MANAGED KUBERNETES

Clusters delivered as a service

- **Hosted control planes**
- **Autoscaling worker pools**
- **Staged upgrades**
- **Cluster backup and restore**

The platform operates the control plane and the worker lifecycle; the tenant holds cluster-admin credentials for its own API.

WHICH PLATFORM FUNCTIONS EACH PROFILE USES

FUNCTION	DEV	AI	IAAS	K8S
Organizations, projects, RBAC, quota	Yes	Yes	Yes	Yes
Per-project VPC and networking	Yes	Yes	Yes	Yes
Container workloads in the project	Primary	Primary	Optional	Optional
Virtual machines	Optional	Optional	Primary	Workers
Managed databases	Primary	Optional	Optional	Optional
GPU slices and passthrough	Optional	Primary	Optional	Optional
Managed Kubernetes clusters	Optional	Optional	Optional	Primary
Observability, audit, backups	Yes	Yes	Yes	Yes

PRIMARY = THE PROFILE'S MAIN WORKLOAD TYPE. OPTIONAL = AVAILABLE IN THE SAME PROJECT, UNDER THE SAME QUOTA, WITHOUT ADDITIONAL INSTALLATION.

TYPICAL TENANT ORGANIZATIONS

Public sector
One organization per directorate or agency, with data residency fixed by where the servers are.

Enterprise IT
Existing VM estate and new container workloads in the same projects and address space.

Research and education
Faculties, labs and course groups as organizations with individually capped plans.

AI teams and labs
Shared GPU pools with per-model entitlements and datasets that stay on-premises.

What a project contains

Everything below is created inside a project, by holders of a project role, within the project's quota. Each item is a Kubernetes API object, so the console, the CLI and any pipeline create it the same way.

Virtual machines

Linux and Windows guests from a catalog or an imported image, with sizing, disks, cloud-init, snapshots, a browser console and live migration. Detailed on page 07.

Container workloads

Deployed into the project's backing namespace with a standard kubeconfig, or into a dedicated managed cluster with its own API. Detailed on page 08.

A dedicated VPC and subnet

The project's own address range and router. No route reaches another project unless it is configured. Detailed on page 09.

Load balancers and published endpoints

Floating IPs, L4 load balancers, gateway routes and TLS certificates issued for the project's hostnames. Detailed on page 10.

Block volumes and object storage

Persistent volumes from platform storage classes, plus S3-compatible buckets and access keys scoped to the project. Detailed on page 11.

Managed databases

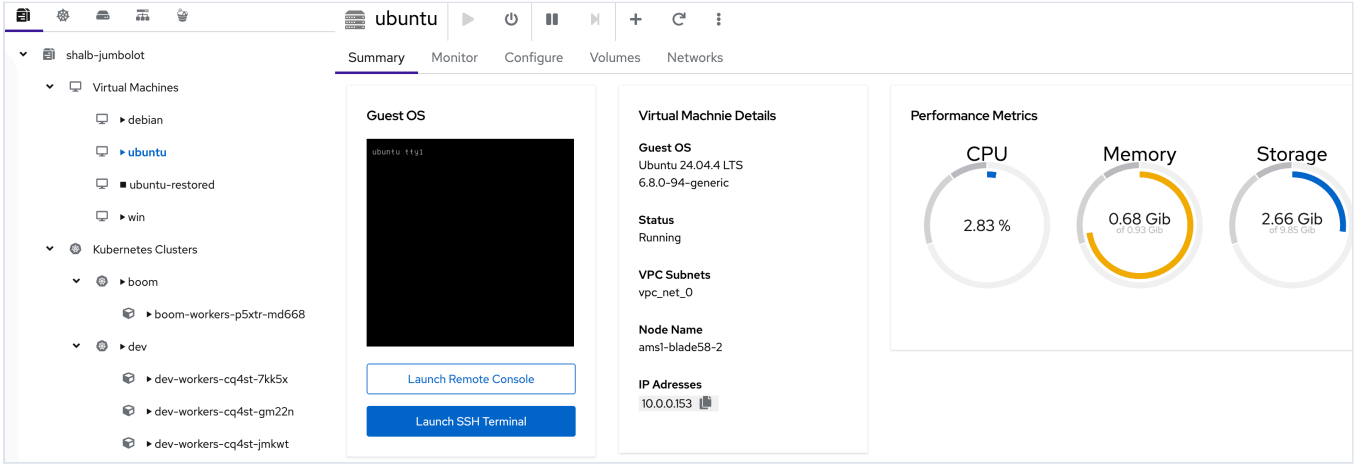
PostgreSQL and MariaDB instances with replication, connection credentials, scheduled backups and point-in-time recovery. Detailed on page 12.

GPU resources

Shared slices of a physical GPU for containers, or a whole device attached to one machine by passthrough. Detailed on page 13.

Secrets, keys and certificates

Purpose-scoped credentials and encryption keys held in the platform secrets manager and synced into the project. Detailed on page 16.



ONE PROJECT TREE HOLDS BOTH VIRTUAL MACHINES AND KUBERNETES CLUSTERS; SELECTING A MACHINE GIVES ITS GUEST OS, NETWORK, PLACEMENT, LIVE UTILIZATION AND A CONSOLE SESSION.

07 · VIRTUAL MACHINES

Virtual machines

Machines run as KubeVirt virtual machines inside the project's backing namespace, on the project's VPC, under the project's quota. A VM and a container workload in the same project share the same address space, the same storage classes and the same access model.

IMAGES AND PROVISIONING

- **Image catalog per installation.** Prepared Linux and Windows images are published centrally; tenants select from the catalog rather than uploading their own each time.
- **Bring your own images.** Standard disk images are imported through the platform's data-import path; guest compatibility is validated on import.
- **Clone-based creation.** New machines are created from a local golden copy rather than a fresh download, so provisioning time does not grow with image size or repeat use.
- **Cloud-init and sysprep.** Hostname, users, SSH keys, packages and first-boot scripts are declared with the machine, for both Linux and Windows guests.

LIFECYCLE AND AVAILABILITY

Tenant-driven operations

Start, stop, restart, resize, attach and detach disks, rebuild from image, rename and delete — all inside the project role, with no platform-team involvement.

Live migration

Machines whose disks are on shared storage move between hosts while running. This is what makes draining a node a routine maintenance operation instead of a scheduled outage.

Snapshots and clones

Point-in-time volume snapshots taken on demand or on a schedule, restorable in place or as a new machine. Clones create a working copy without a full restore.

Placement and scheduling

Node selection, anti-affinity and dedicated node pools are set by the platform team; tenants see the resulting placement but do not choose hosts.

GUEST ACCESS AND NETWORKING

AVAILABLE TO EVERY MACHINE IN A PROJECT

Browser console
VNC · SERIAL
Graphical and serial console without SSH or a jump host

Project addresses
VPC SUBNET · STATIC OR POOL
Same subnet as the project's container workloads

Published access
FLOATING IP · LOAD BALANCER
Identical mechanism to any other project service

Additional interfaces
MULTUS · VLAN ATTACHMENT
Direct attachment to an existing datacenter VLAN

Guest agent
STATUS · FILESYSTEM · CLEAN SHUTDOWN
Reports guest OS state and enables consistent snapshots

Metrics
CPU · MEMORY · DISK · NETWORK
In the tenant's own dashboards from creation

CONSTRAINTS WORTH KNOWING

Live migration requires shared storage	Machines on node-local volumes can be moved only by shutting them down
GPU passthrough pins a machine	A VM holding a passthrough device is not live-migratable
Sizing is bounded by quota	A machine cannot be created or resized beyond the project's remaining CPU, memory and storage allowance
Windows licensing is yours	The platform runs Windows guests; guest OS licences remain your responsibility

Managed Kubernetes clusters

A tenant can deploy directly into its project's namespace, or provision a dedicated cluster with its own API server and its own authorization boundary. Dedicated clusters have hosted control planes: the platform runs the control plane as managed workloads, so there are no master nodes for anyone to operate.

WHAT A CLUSTER CONSISTS OF

OPERATED BY THE PLATFORM · OWNED BY THE TENANT

When a cluster is created

SELF-SERVICE

A tenant submits a web console form, or applies a cluster manifest in its project. The platform then builds and maintains every element below.

PRODUCES

Kubernetes API endpoint
TENANT-ADMINISTERED
Private in the project's VPC by default; public endpoint on request

Hosted control plane
PLATFORM-OPERATED
Runs as managed pods and is right-sized by the platform

Cluster datastore
SNAPSHOTTED
On platform storage, backed up on a schedule

Worker machines
AUTOSCALED POOLS
VMs in the project that join the cluster automatically

THE TENANT HOLDS CLUSTER-ADMIN CREDENTIALS; THE PLATFORM CARRIES THE CONTROL PLANE, THE DATASTORE AND THE WORKER LIFECYCLE. THE CLUSTER KEEPS ITS OWN API, AUTHORIZATION AND CNI BOUNDARY.

WORKER POOLS

- **Independent per pool.** Each pool has its own machine size, image, node labels and autoscaling range, so general-purpose, memory-heavy and GPU nodes coexist in one cluster.
- **Autoscaling within quota.** Pools grow and shrink with scheduling pressure; the upper bound is the pool's maximum and the project's remaining quota, whichever is reached first.
- **Replaced, not patched.** Node image changes roll out as new machines joining and old machines draining, so worker state stays reproducible.

UPGRADES AND RECOVERY

Staged upgrades

Control plane and worker pools upgrade separately, in steps the tenant triggers. A platform upgrade does not force a tenant cluster version change.

Version window

The platform publishes the supported Kubernetes versions per release; clusters must stay inside that window to remain supported.

Cluster-state backups

Scheduled datastore snapshots with optional encryption, retained per policy and restorable by the tenant. Full scope on page 17.

Adoption of existing clusters

An existing conformant cluster can be brought under platform management through the CLI, rather than rebuilt.

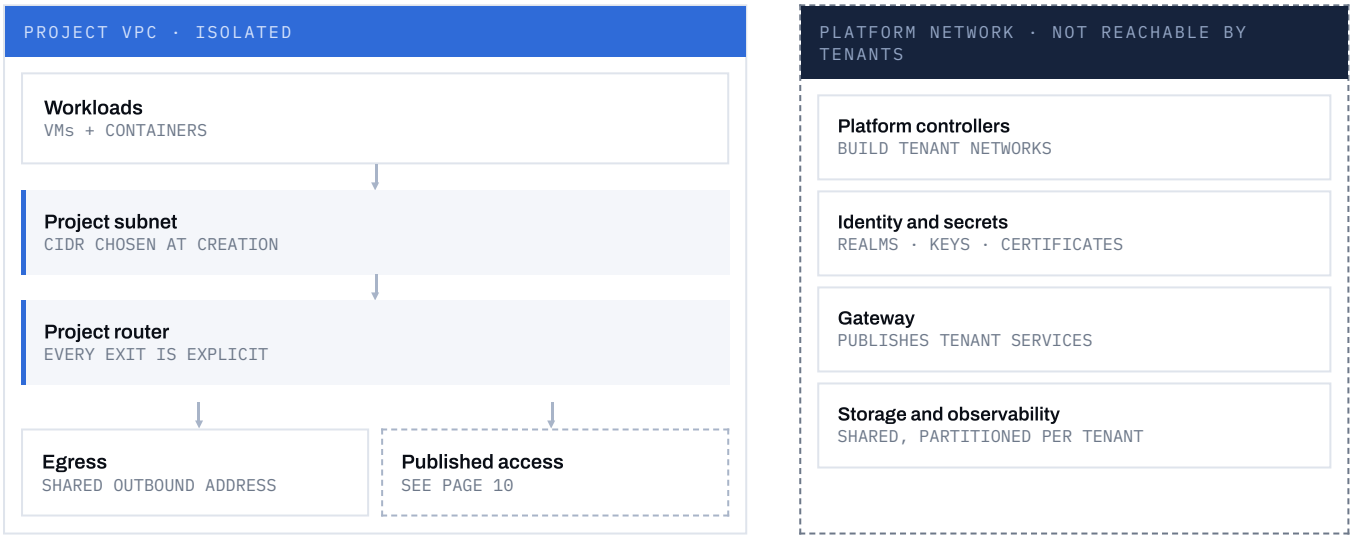
Deploying into the project namespace and provisioning a dedicated cluster are both available in the same project. The namespace path needs no cluster lifecycle at all; the dedicated path adds a separate API and authorization boundary for tenants that need cluster-level control.

Per-project networks and routing

Creating a project creates its network. Each project gets a dedicated VPC, a subnet with an address range chosen at creation, and its own router. Projects are unreachable from each other unless a route or a published service is configured deliberately.

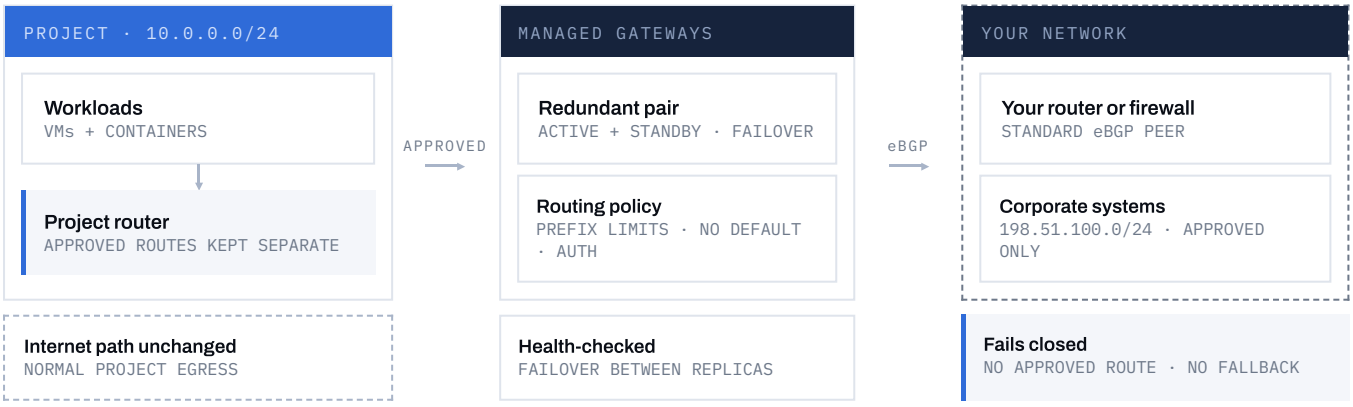
- **Isolation by default.** A project's VPC has no route to another project's, and none to the platform's own services. Nothing is implicitly shared between tenants.
- **Attachment to existing VLANs.** Where the datacenter fabric allows it, a project network attaches directly to a provider VLAN through Multus — for lab hardware, legacy systems and dedicated links that cannot be moved.
- **eBGP peering into your network.** A whole project can be routed into an existing corporate routing domain over standard eBGP. Only the project's prefixes are advertised, and only prefixes the platform team approves are accepted.
- **Controlled egress.** Outbound paths, address pools and allowlists are configured by the platform team as policy, not requested per workload.

PROJECT NETWORK AND THE PLATFORM'S OWN



TENANTS SHARE NO NETWORK WITH EACH OTHER OR WITH THE PLATFORM'S OWN SERVICES; EGRESS AND PUBLISHED PATHS ARE SEPARATE, DELIBERATE DECISIONS.

ROUTING A PROJECT INTO AN EXISTING CORPORATE NETWORK

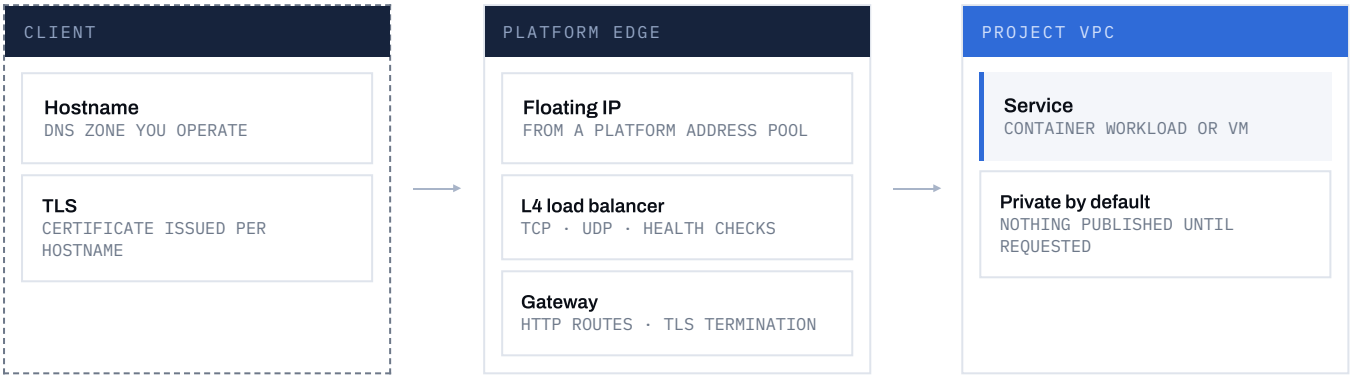


CORPORATE DESTINATIONS TAKE THE MANAGED GATEWAY PAIR; EVERYTHING ELSE KEEPS THE PROJECT'S NORMAL EGRESS. NO ROUTE APPEARS THAT THE PLATFORM TEAM DID NOT APPROVE.

Publishing a service

Publishing is a tenant action inside resources the platform team owns. Tenants create load balancers, claim floating IPs and request certificates; the address pools, the DNS zones, the gateway and the issuing authority remain platform configuration.

THE PATH FROM A CLIENT TO A WORKLOAD



THE TENANT DECLARES THE SERVICE AND THE HOSTNAME; ADDRESSES, DNS AND CERTIFICATE ISSUANCE COME FROM PLATFORM-OWNED CONFIGURATION.

WHO CONTROLS WHAT

ELEMENT	OWNER	NOTES
Address pools	PLATFORM	Public and internal ranges defined per installation; tenants draw from them within quota
Floating IPs	TENANT	Claimed, attached and moved between the project's services and machines
Load balancers	TENANT	Created as standard Kubernetes services; counted against the project's load-balancer quota
HTTP routes and gateway	SHARED	Tenants attach routes to a gateway the platform team operates and sizes
TLS certificates	PLATFORM	Issued and renewed automatically by cert-manager from an issuer you configure
DNS zones	PLATFORM	Delegated zones per installation or per organization; records created for published services
Egress addressing	PLATFORM	Shared or dedicated outbound addresses per project, set as policy

> Publishing a service does not require a network change request, and it cannot exceed what the platform team allocated: a tenant can only publish onto addresses, hostnames and gateways that already exist in platform configuration.

Storage

One storage layer serves virtual machines and container workloads. Block volumes are provisioned through CSI from storage classes the platform team publishes; object storage is S3-compatible and scoped per project. Software-defined storage and external enterprise arrays can back the same platform storage classes.

STORAGE TYPES AVAILABLE IN A PROJECT

TYPE	ACCESS	USED FOR
Replicated block	SINGLE WRITER	Default class for VM disks and stateful workloads; survives losing a node
Shared block	MULTI-ATTACH	Workloads needing the same volume on several nodes, and live-migratable VMs
Node-local	HOST-BOUND	Throughput-sensitive data with its own replication; excludes live migration
Object storage	S3 API	Application data, datasets, database and cluster backups, image artifacts
External array	VENDOR CSI	Existing enterprise storage exposed as an additional platform storage class

BLOCK VOLUMES

- **Provisioned on demand.** Volumes are created by declaring them; the tenant chooses the storage class and size within the project's storage quota.
- **Online expansion.** Volumes grow without detaching, subject to the class and the guest filesystem supporting it. Shrinking is not supported.
- **Snapshots and clones.** Point-in-time snapshots on demand or on a schedule; a clone creates an independent writable volume from a snapshot without a full copy-back.
- **Encryption at rest.** Available per storage class, with keys held in the platform secrets manager rather than on the node.

OBJECT STORAGE

Buckets and keys per project

Buckets are project resources; access keys are issued per bucket and stored as project secrets. One project's credentials do not read another's data.

S3-compatible API

Standard S3 clients, SDKs and backup tools work unchanged, which is how database and cluster backups are written and read.

Quota-aware

Object capacity counts against the organization's storage allowance and is visible per project alongside block usage.

A destination, not an archive

Object storage inside the installation shares its failure domain. Copies that must survive the site belong off-platform — see page 17.

USING STORAGE YOU ALREADY OWN

SNAPSHOT, CLONE AND EXPANSION BEHAVIOUR ON AN EXTERNAL CLASS IS WHATEVER ITS CSI DRIVER SUPPORTS; VALIDATE PER ARRAY IN THE ARCHITECTURE REVIEW.

Managed databases

Databases are project resources with their own lifecycle: a tenant declares an instance and gets an endpoint, credentials, replication, monitoring and a backup policy it controls. Operators run the engines; nobody in the tenant team has to become the database team.

ENGINES AND TOPOLOGIES

ENGINE	TOPOLOGY	BEHAVIOUR
PostgreSQL	Single instance	Development and non-critical workloads; restart on failure, no automatic failover
PostgreSQL	Primary + replicas	Streaming replication; an eligible replica is promoted automatically after primary failure
MariaDB	Single instance	Development and non-critical workloads; restart on failure
MariaDB	Replicated	Replication with managed failover, subject to topology and spare capacity

FAILOVER OUTCOME DEPENDS ON REPLICA HEALTH, PLACEMENT AND AVAILABLE CAPACITY — IT IS AUTOMATED, NOT UNCONDITIONAL.

WHAT A TENANT CONTROLS

Instance shape

Engine version, CPU and memory, storage class and size, replica count — declared on creation and changeable afterwards.

Credentials and delivery

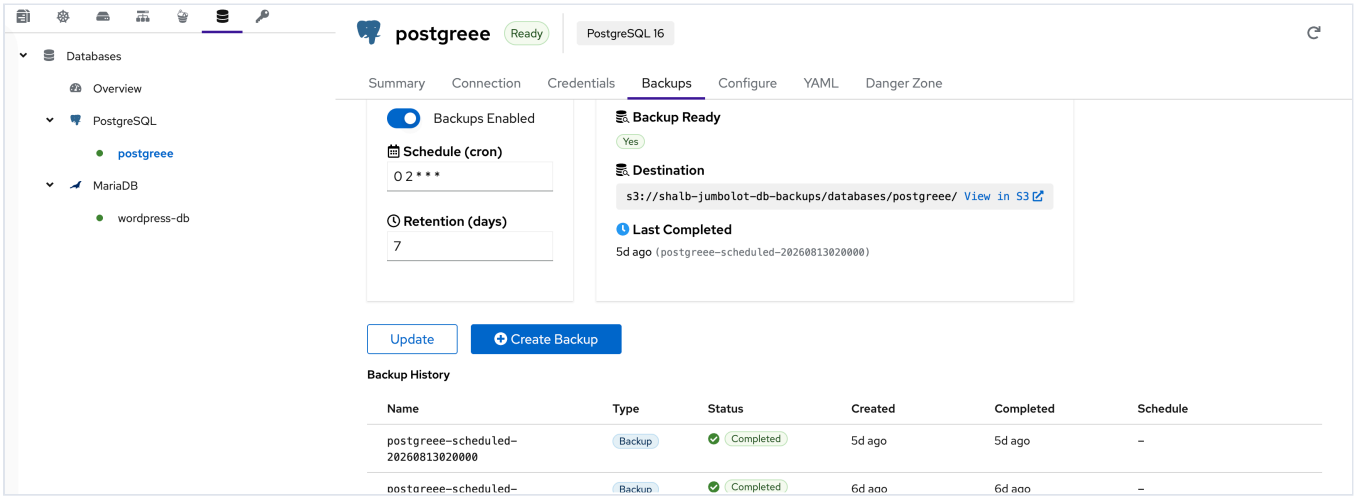
Connection details are written into a project secret, so an application consumes them without anyone copying a password. Rotation is available.

Reachability

Private inside the project VPC by default. Exposure beyond the project uses the same publishing path as any other service.

Backup policy

Schedule, retention and destination bucket are set by the team that owns the database — backup policy stops being a central ticket.



BACKUPS ENABLED, CRON SCHEDULE, RETENTION, PROJECT S3 DESTINATION AND RUN HISTORY — ALL EDITABLE BY THE PROJECT'S OWN ROLE HOLDERS.

RECOVERY OPTIONS

Restore in place

SAME INSTANCE

Roll the existing database back to a chosen backup

Restore to a new instance

SIDE-BY-SIDE

Verify or extract data without touching production

Point-in-time recovery

BASE BACKUP + WAL

Recover to a timestamp inside the retention window

GPU services

GPUs are allocated in two modes: a share of a physical device for container workloads, or a whole device attached to one virtual machine. Both are quota-governed per organization and per GPU model, so expensive cards are distributed by policy rather than by whoever asks first.

ALLOCATION MODES

SHARED SLICES · CONTAINERS

Several workloads on one physical GPU

Slices come from a catalogue of fixed profiles published per installation. Device memory is partitioned and enforced per slice; compute is shared cooperatively between the workloads on the device.

SUITED TO
Inference endpoints, notebooks, development, small training runs

NOT A PERFORMANCE GUARANTEE
A density mechanism: neighbours on the same device affect throughput

PASSTHROUGH · WHOLE DEVICE

One device, one virtual machine

The GPU is assigned to a single VM, which sees the physical device directly. No sharing, no neighbours, and a stronger isolation boundary between tenants.

SUITED TO
Sustained training, benchmarking, workloads with strict isolation requirements

CONSTRAINT
A machine holding a passthrough device is not live-migratable

GOVERNANCE AND SCHEDULING

- **Entitlements are quota.** An organization's plan states how many devices or slices it may hold per GPU model. A request beyond the entitlement is rejected the same way a CPU request beyond quota is.
- **Reserved capacity where queuing is unacceptable.** Specific devices or node pools can be reserved for an organization, so a critical tenant is not competing for scheduling with everyone else.
- **Node pools are labelled and targeted.** GPU nodes form their own pools; container workloads and Kubernetes worker pools request them explicitly rather than landing on them by accident.
- **Usage is recorded per tenant.** GPU consumption appears in the organization's usage record alongside CPU, memory and storage, which is what makes internal chargeback for accelerators possible.

DRIVER AND FIRMWARE QUALIFICATION

QUALIFIED TOGETHER, RELEASED TOGETHER

Host kernel	GPU driver	Device plugin	Container toolkit
PINNED PER RELEASE	VERSION-MATCHED	SLICE PROFILES	RUNTIME INTEGRATION

A RELEASE IS NOT ALLOWED ONTO GPU NODES UNTIL THE WHOLE SET IS QUALIFIED AS ONE COMBINATION. SUPPORTED MODELS ARE CONFIRMED AGAINST YOUR BILL OF MATERIALS BEFORE PURCHASE.

Quotas and capacity

Quota is the mechanism that makes self-service safe. An organization holds a capacity plan; its projects consume against that plan and can be individually capped inside it. Enforcement happens at admission — a request beyond the limit is refused, not throttled after the fact.

HOW ALLOCATION FLOWS

INSTALLATION CAPACITY

The physical total: CPU, memory, storage and GPUs across all nodes, minus what the platform itself reserves.

ALLOCATED →

ORGANIZATION PLAN

A named plan per tenant. The sum of plans may be deliberately below or above installation capacity, depending on how you choose to oversubscribe.

CAPPED →

PROJECT QUOTA

Optional per-project caps inside the plan. Without a cap, projects compete for the organization's own allowance.

A RUNAWAY PROJECT EXHAUSTS ITS OWN CAP OR ITS ORGANIZATION'S PLAN — NEVER ANOTHER TENANT'S ALLOWANCE AND NEVER THE INSTALLATION.

WHAT IS METERED

CPU
REQUESTS AND LIMITS
Counted across container workloads and virtual machines alike

Memory
REQUESTS AND LIMITS
A VM's assigned memory occupies quota while it is running

Storage
BLOCK + OBJECT
Provisioned volume capacity plus object-storage consumption

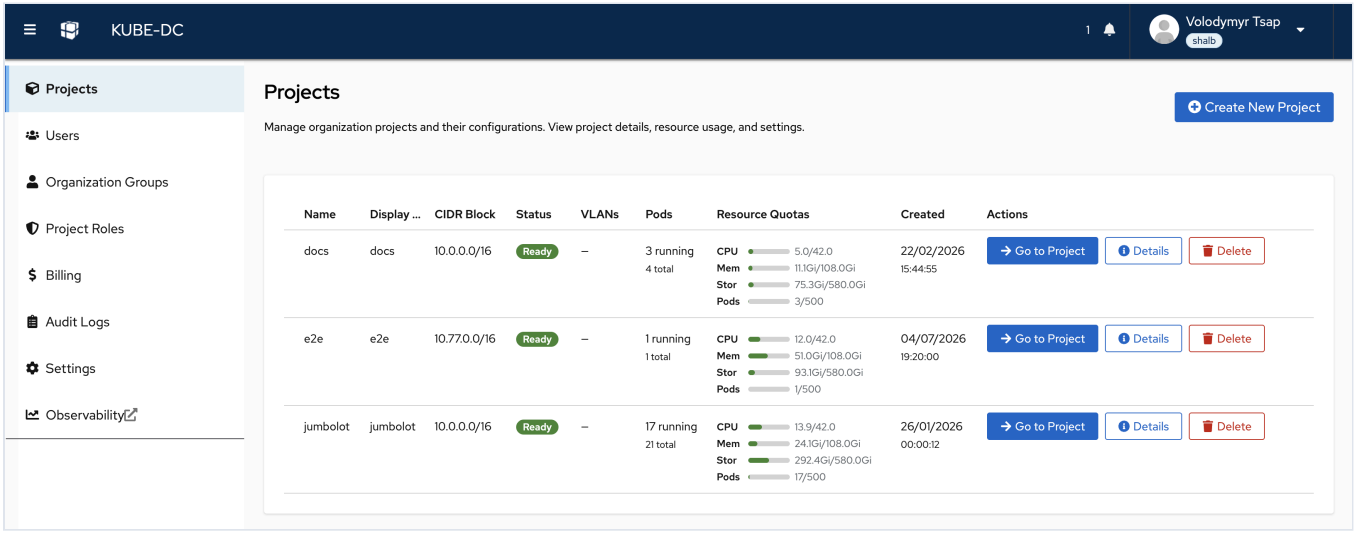
Pods
OBJECT COUNT
Bounds runaway scaling independently of CPU and memory

Load balancers
SERVICE COUNT
Limits how much of a shared address pool one tenant can take

GPU entitlements
PER MODEL
Devices and slices held per GPU model, per organization

Managed clusters
COUNT + WORKERS
Worker machines consume the same project quota as any VM

Addresses
FLOATING IPs
Public address consumption per project



CONSUMPTION AGAINST QUOTA IS VISIBLE PER PROJECT IN THE ORGANIZATION'S OWN CONSOLE — CPU, MEMORY, STORAGE AND POD COUNT, ALONGSIDE THE PROJECT'S NETWORK ALLOCATION AND STATE.

WHERE LIMITS ARE SET AND SEEN

Platform administrator	Creates plans, assigns them to organizations, sets reserved capacity and oversubscription policy
Organization administrator	Distributes the plan across projects, sets or removes per-project caps, sees every project's consumption
Project role holder	Sees the project's own usage and remaining headroom; cannot raise its own cap
API and automation	Quota objects are readable through the Kubernetes API, so pipelines can check headroom before deploying

Identity and access

Each organization gets its own identity realm. People sign in once and reach the console, the platform API and the CLI with the same identity; authorization is granted to groups, per project, and never to individuals directly.

HOW A PERSON GETS ACCESS

A GROUP IS GRANTED ROLES INDEPENDENTLY IN EACH PROJECT, SO JOINING A TEAM IS THE ONLY ONBOARDING STEP AND REMOVING SOMEONE FROM A GROUP REVOKES EVERYTHING AT ONCE.

IDENTITY SOURCES AND SESSIONS

Federated or local accounts

A realm federates to your corporate directory or identity provider, or holds local accounts where a tenant has no directory of its own.

One session, three surfaces

The tenant console, the platform API and the CLI accept the same OIDC identity; the CLI issues a kubeconfig bound to it.

Multi-factor and password policy

Enforced by the realm, so requirements follow whatever your identity provider already mandates rather than being re-implemented.

Machine identities

Pipelines and automation use project-scoped service accounts and tokens, separate from human accounts and revocable independently.

ROLE SCOPES

SCOPE	TYPICAL HOLDER	CAN DO
Platform administrator	The operating team	Create organizations, assign plans, operate the installation. Never part of a project role.
Organization administrator	A tenant's own IT owner	Manage members and groups, create projects, distribute quota, view all projects
Project administrator	A team lead	Full control of resources inside one project, including grants to groups in that project
Project member	Engineers	Create and operate workloads, machines, databases and storage within quota
Read-only	Auditors, support, finance	Inspect state, usage and dashboards without any mutating permission

ROLES ARE IMPLEMENTED AS KUBERNETES RBAC OVER HIERARCHICAL NAMESPACES; PLATFORM-WIDE ADMINISTRATION IS NEVER REACHABLE FROM A PROJECT ROLE.

Security, audit and encryption

Three controls make tenant self-service reviewable: every action is recorded, every credential is issued from a secrets manager rather than pasted into a manifest, and traffic and data are encrypted with keys the installation owns.

AUDIT LOG

- **Every mutating action is recorded.** Who acted, in which organization and project, on which resource, when, and whether the request was allowed — including actions taken through the API and the CLI, not only the console.
- **Scoped to the tenant.** An organization administrator reads its own audit log in the tenant console; the platform operator sees the estate-wide record.
- **Includes privilege use.** Elevated access — a platform operator acting inside a tenant — is recorded as an explicit, time-bounded elevation rather than an anonymous superuser action.
- **Exportable.** Records can be shipped to the log store and on to your SIEM, so retention beyond the platform's own window is your policy to set.

SECRETS AND KEY MANAGEMENT

HELD IN THE PLATFORM SECRETS MANAGER · SYNCED INTO THE PROJECT

Database credentials ISSUED PER INSTANCE Delivered as a project secret; rotation available	**Object storage keys** PER BUCKET Scoped so one project cannot read another's data	**Encryption keys** PURPOSE-SCOPED Volume, backup and cluster keys held separately	**Certificates** ISSUED AND RENEWED Automatic renewal from an issuer you configure

TENANTS CONSUME CREDENTIALS BY REFERENCE; THE PLATFORM ISSUES, STORES AND ROTATES THEM. NOTHING REQUIRES A LONG-LIVED SECRET IN A REPOSITORY.

ENCRYPTION

WHERE	DEFAULT	MECHANISM
Published endpoints	ON	TLS terminated at the gateway with certificates issued per hostname
Platform APIs and console	ON	TLS on every management interface, including the CLI and cluster API endpoints
Tenant traffic between nodes	OPTIONAL	Encrypted overlay for east-west traffic, enabled per installation
Block volumes	PER CLASS	Encryption at rest with keys from the secrets manager, not from the node
Backups	OPTIONAL	Database and cluster-state backups encrypted before they leave the service
Platform secrets	ON	Encrypted at rest in the secrets manager with sealed unlock keys

PLATFORM POLICY CONTROLS

No interactive container access
Attaching a shell to a running container is blocked platform-wide; one-off work runs as an auditable job instead.

Time-bounded elevation
Operator access into a tenant is granted for a window, recorded, and visible on the administration dashboard while active.

Configuration as reviewed commits
Platform settings and component versions are reconciled from Git, so a change has an author, a review and a revert path.

Backup, recovery and data protection

There is no single platform-wide backup job. Each service protects its own data with a mechanism appropriate to it, on a schedule the resource owner sets, into a destination the project controls. Off-site protection is a separate layer, and deliberately so.

WHAT IS PROTECTED, AND HOW

RESOURCE	MECHANISM	DESTINATION	RESTORE PATH
Managed databases	Scheduled and on-demand backups, continuous WAL archiving	Project object storage	In place, into a new instance, or to a timestamp (PITR) — by the tenant
Kubernetes clusters	Cluster-state datastore snapshots, optionally encrypted	Platform storage or object storage	Restore cluster state to a chosen snapshot — by the tenant
Virtual machines	Volume snapshots, on demand or scheduled; guest-agent quiesce	Storage pool	Roll back in place, or clone the snapshot into a new machine
Block volumes	CSI snapshots and clones	Storage pool	Restore to the same volume or provision a new one from the snapshot
Object storage	Replication inside the pool; versioning where enabled	Storage pool	Object-level recovery through the S3 API
Platform configuration	Desired state in Git, reconciled continuously	Your repository	Revert a commit and let reconciliation restore configuration
Identity and secrets	Realm and secrets-manager state backups	Platform storage	Operator-driven restore as part of platform recovery

TWO LAYERS, TWO FAILURE DOMAINS

Inside the platform
PROJECT OBJECT STORAGE + STORAGE POOL

Recovers from deletion, corruption, a bad deploy or a lost node. Fast, tenant-driven, no ticket. Shares the installation's failure domain.

COPIED →

Your backup system
OFF-SITE · SEPARATE FAILURE DOMAIN

Recovers from losing the site or the storage system. Integrates through the S3 endpoints and snapshot APIs.

KUBE-DC DOES NOT REPLACE AN ENTERPRISE BACKUP PRODUCT. ONE INSTALLATION IS ONE SITE; MULTI-SITE PROTECTION IS DESIGNED IN THE ARCHITECTURE REVIEW.

WHO SETS AND WHO RUNS IT

The resource owner sets the policy

Schedule, retention and destination for a database or a cluster are edited by the project's own role holders, in the console or in the manifest.

The platform enforces the floor

Minimum retention, mandatory encryption and permitted destinations can be set as platform policy so a tenant cannot opt out of protection entirely.

Restores are self-service

A tenant restores its own database, volume, machine or cluster state without opening a request with the platform team.

Backups are visible, not implied

Last successful run, duration and history are shown on the resource, so a silently failing backup is a visible state rather than a discovery during an incident.

What is not a backup: the Git repository holds desired configuration, not tenant data. Replication inside the storage pool protects against hardware failure, not against deletion. Neither substitutes for the backup policies above.

Observability, provisioned automatically

Creating an organization also creates its observability: dashboards, a metrics space, log routing and an entry point in the console. Nothing is installed by the tenant, and no per-tenant monitoring stack is built by the platform team.

WHAT EXISTS THE MOMENT A TENANT IS CREATED

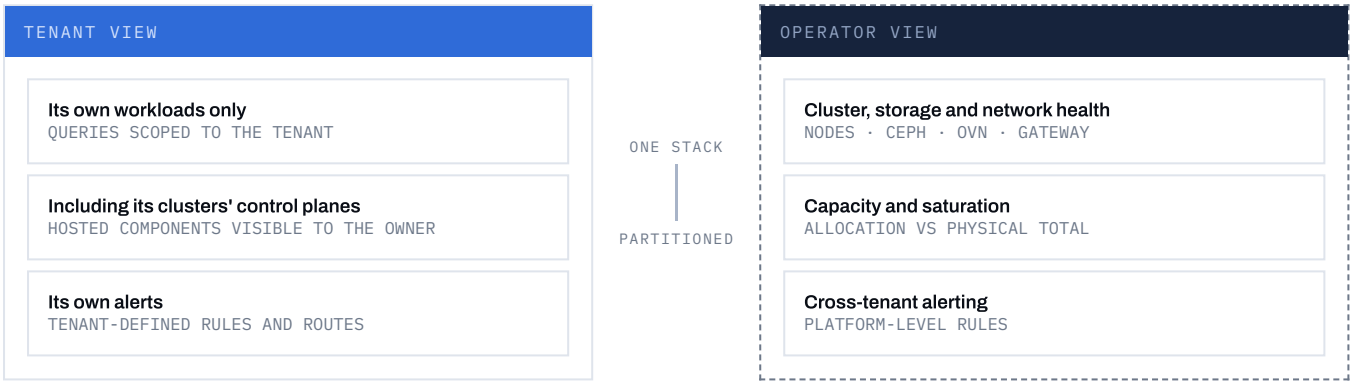
Dashboards
PRE-BUILT · PER TENANT
Workloads, machines, databases, clusters and storage use

Metrics space
ISOLATED TENANT
Scraping and long-term storage without tenant configuration

Log routing
LABELLED BY PROJECT
Container and system logs queryable by project and workload

Console entry point
SAME SIGN-IN
Reached from the tenant console with the project identity

SEPARATION AT THE DATA LAYER



ONE OBSERVABILITY STACK SERVES BOTH VIEWS. A TENANT CANNOT QUERY ANOTHER TENANT'S METRICS OR LOGS; THE OPERATOR SEES THE INFRASTRUCTURE THE TENANTS RUN ON.

COVERAGE AND CONTROLS

Virtual machines are first-class

CPU, memory, disk and network for VMs sit in the same dashboards as container workloads, rather than in a separate hypervisor tool.

Managed services report themselves

Database replication state, backup success, cluster control-plane health and storage use appear without the tenant instrumenting anything.

Retention is your policy

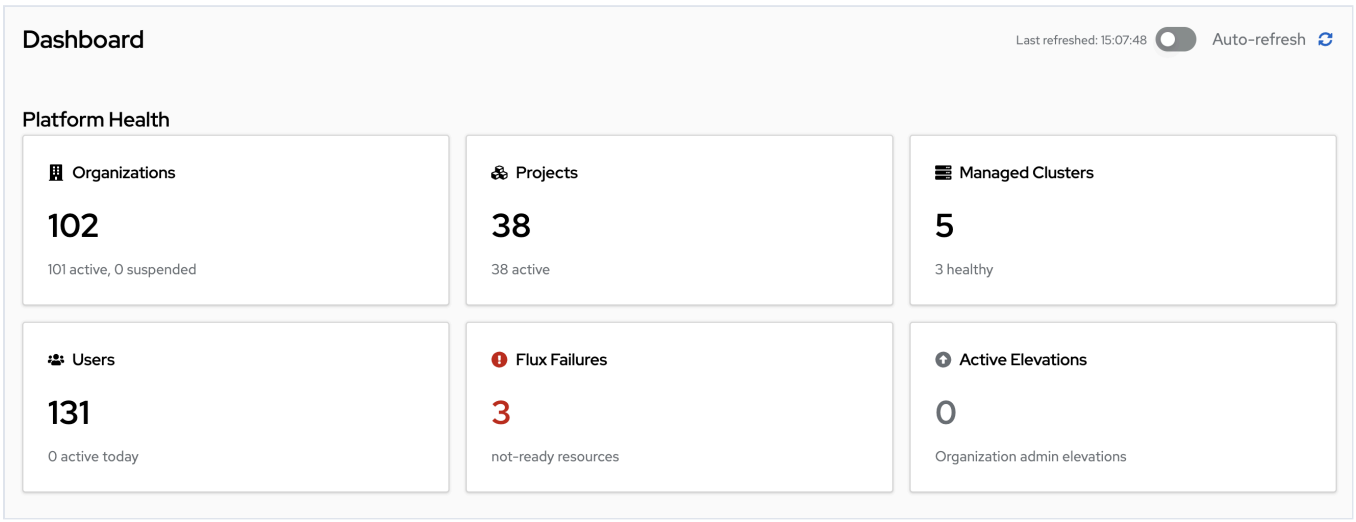
Metric and log retention windows are set per installation, sized against the storage you allocate to observability.

Exportable

Metrics and logs can be forwarded to an existing observability platform or SIEM; the built-in stack does not have to be the only destination.

Global platform administration

A separate administration surface for whoever operates the installation. It covers the whole estate — every organization, project, cluster and user — and is reachable only with platform-administrator rights, never from a project role.



ONE DASHBOARD COVERS THE WHOLE TENANT ESTATE: WHAT EXISTS, WHAT IS FAILING RECONCILIATION, AND WHO HOLDS ELEVATED ACCESS RIGHT NOW.

WHAT THE ADMINISTRATION SURFACE COVERS

Tenants and plans
CREATE · ASSIGN · SUSPEND
Organizations, their capacity plans and their project inventory

Users and elevation
ESTATE-WIDE
Accounts across realms, plus active time-bounded operator elevations

Reconciliation state
FAILURES SURFACED
Platform resources that did not converge, with the reason

Capacity
ALLOCATED VS PHYSICAL
How much of the installation is committed to plans, and how much is in use

Storage and network health
POOLS · GATEWAYS · ADDRESS POOLS
The shared infrastructure every tenant depends on

Usage and billing
PER ORGANIZATION
Recorded consumption, and subscription billing where it is enabled

DAY-2 OPERATIONS

Upgrades as reviewed commits

Component versions and platform settings live in Git and are reconciled continuously, so a change has an author, a diff and a revert path instead of being reconstructed from console history.

Node maintenance

Draining a host live-migrates eligible machines and reschedules container workloads in one operation, on the platform team's schedule.

Scaling the installation

Capacity grows by adding nodes; existing plans are unaffected until you choose to raise them.

Registries you control

Installation and upgrades pull component images and charts from registries you operate. Fully air-gapped operation is scoped in the architecture review.

Interfaces and automation

Every platform resource is a Kubernetes API object. The console, the CLI and any pipeline act on the same objects with the same authorization, so there are no console-only features and no separate SDK to adopt.

ACCESS SURFACES

SURFACE	USED BY	FOR
Tenant web console	Project members	Creating and operating machines, clusters, databases, storage and published services through forms
Administration console	Platform operators	The estate-wide view described on page 19
Kubernetes API	Tooling and pipelines	Every platform resource, with a project-scoped kubeconfig and standard RBAC
Command line	Both	Login and context switching, day-to-day operations, installation, status and cluster adoption
Git repository	Platform operators	Desired platform state: component versions, settings, tenant definitions

AUTOMATION PATHS

Terraform
KUBERNETES PROVIDER
Platform resources are managed as ordinary Kubernetes manifests — no dedicated provider required

Ansible
K8S MODULES
Existing playbooks address the platform the same way they address any cluster

CI pipelines
PROJECT SERVICE ACCOUNTS
Machine identities scoped to a project, revocable independently of people

GitOps
FLUX OR YOUR OWN
Tenants can reconcile their own project contents from their own repositories

AGENT-DRIVEN WORKFLOWS

AN AI AGENT IS A FIRST-CLASS ACTOR — SCOPED, DOCUMENTED, AUDITED

The platform publishes documented workflows an AI coding assistant can drive on a team's behalf. An agent authenticates as a project-scoped identity, so it operates inside the same roles, the same quota and the same audit log as a human member — and because every platform operation is an API object, an agent needs no screen-scraping and no privileged escape hatch.

CREATE A PROJECT | DEPLOY AN APP | PROVISION A MACHINE | REQUEST A DATABASE | PUBLISH AN ENDPOINT

Scoped credentials
No agent holds platform-wide rights; a token is revocable on its own

Documented steps
Workflows are explicit, not inferred from the UI

Fully attributed
Every action lands in the tenant's audit log, agent or person

QUOTA AND ROLE CHECKS APPLY IDENTICALLY TO AGENTS; AN AGENT CANNOT EXCEED WHAT ITS PROJECT MEMBERSHIP ALLOWS.

Because the API is the platform, an exit path exists by construction: workloads remain standard Kubernetes objects and standard VM images, on hardware you own, with platform configuration in your own repository.

Deployment baseline

One starting configuration, subject to architecture validation. It is the smallest sensible production shape rather than a lab minimum; sizing above it depends on workload profile, storage topology and the failure domains you need to survive.

REFERENCE SIZING

	STARTING POINT PER NODE
Server nodes	3 or more; capacity scales by adding nodes
CPU per node	16 cores or more, x86-64 with virtualization extensions
RAM per node	64 GB or more
Storage per node	Per your capacity plan; dedicated NVMe or SSD disks for the storage pool
Operating system	Ubuntu 26.04 LTS
Network	Redundant NICs; separate management, cloud and provider networks
GPU nodes	Optional, in dedicated pools; models qualified against your bill of materials

PLATFORM SERVICES THEMSELVES CONSUME 12 CPU AND 64 GB RAM ACROSS THE CLUSTER; EVERYTHING BEYOND THAT CARRIES TENANT WORKLOADS.

HOW AVAILABILITY IS ACHIEVED

Management cluster
Replicated control plane across three or more nodes

Storage
Replicated pool; a node loss does not lose a volume

Machines
Live migration off a host you need to service

Gateways and databases
Redundant pairs with health-checked failover

ONE INSTALLATION IS ONE SITE. MULTI-SITE AND DISASTER-RECOVERY TOPOLOGIES ARE DESIGNED PER DEPLOYMENT.

WHAT IS INCLUDED, AND WHAT IS ENABLED ON REQUEST

PART OF THE PLATFORM

Tenancy and RBAC, virtual machines, Kubernetes clusters, networking, storage, managed databases, quotas, audit, backups and observability.

ENABLED WHEN NEEDED

GPU services, subscription billing and payment providers, eBGP peering and VLAN attachment, external storage arrays, encrypted overlay networking.

FURTHER DOCUMENTATION

Each function in this document has its own detailed datasheet.

Compute, networking, storage, data services, GPU, security and operations are documented individually, including API references, configuration options and validated topologies.