PORTAINER.iO

Reference Architecture

# KUBERNETES MANAGEMENT PLATFORM

This Reference Architecture defines how Kubernetes clusters are built, governed, secured, and consumed.

Dec 25

# CONTENTS

The organisation is adopting a unified Kubernetes Management Platform to provide a consistent, governed, and scalable foundation for container operations across data centre, cloud, remote site, and far edge environments. As Kubernetes usage grows, the need for a centralised management plane becomes critical. Fragmented cluster deployments, inconsistent security controls, and varied team practices significantly increase operational risk and reduce the organisation's ability to run Kubernetes in a safe and repeatable manner.

This Reference Architecture defines how Kubernetes environments are built, governed, secured, and consumed. It establishes a clear set of platform principles, architectural standards, and cluster lifecycle expectations designed to support both production and non-production workloads, distinguishing between declarative, secure and auditable production environments, and support for rapid iteration with the appropriate guardrails for non-production environments.

The architecture is structured into five planes, encompassing the infrastructure itself, management platform layer and its presentation layer, cross-cluster and external services, and a landing zone. These are detailed in chapter 5.

This reference architecture aims to deliver a vendor-neutral conceptual model, with a set of guiding principles for designing and operating a platform. As an example, a target-state implementation using Portainer is included, showing how the reference architecture can be implemented in practice in chapter 6.

This architecture ensures Kubernetes remains portable, upstream aligned, and strongly governed. It supports safe, predictable operation at scale while reducing operational burden and providing teams with a consistent and intuitive platform experience. The architecture also acknowledges the unique challenges of edge deployments, where physical access, network trust boundaries, and intermittent connectivity require hardened controls, deterministic system behaviour, and strong policy enforcement.

# 1. Terms of Reference

The organisation intends to establish an enterprise-grade Kubernetes Management Platform to provide a consistent, governed, and scalable foundation for container orchestration across all environments. As Kubernetes adoption accelerates, the absence of a unified operating model introduces operational risk, inconsistent security postures, fragmented deployment workflows, and rising platform management overhead.

This Reference Architecture defines the structural blueprint for how Kubernetes clusters are deployed, governed, secured, operated, and consumed. It establishes the standards, principles, and architectural planes required to support production and non-production workloads with clear separation of concerns and predictable behaviour. The intent is to ensure that future infrastructure investments align with a long-term, coherent, and supportable management-plane design.

**The reference architecture provides:**

- A vendor-neutral conceptual model
- A set of guiding principles for platform governance
- Architectural standards for cluster configuration, workload deployment, and access control
- A formal definition of the multi-plane platform structure
- A target-state implementation using a chosen management platform
- A foundation on which detailed designs, migration plans, and operational procedures can be built
- An overview of pre-requisites defined in the ‘landing zone’ plane detailed below.

The document covers the conceptual, logical, and service planes required for a complete Kubernetes management capability.

# 2. Requirements

## 2.1 Requirements Overview

The requirements described in this section reflect the organisation's dependency on a Kubernetes environment that is predictable, secure, auditable, and efficient to operate. These requirements ensure the platform can support the current and future application landscape, minimise operational burden, and maintain compliance across all clusters regardless of location or underlying distribution.

The requirements are grouped into three key areas: **Platform, Security**, and **Management**. Each requirement informs architectural decisions and feeds into the standards and principles that shape the target platform.

## 2.2 Platform Requirements

The Kubernetes Management Platform must support a multi-cluster, multi-distribution, topology spanning cloud, datacentre, and edge environments. Clusters must be deployed from approved profiles to ensure consistent configuration and behaviour. Without this, operational drift and inconsistency inevitably increase platform fragility.

A declarative configuration model is mandatory for production. All configuration affecting production clusters must originate from a GitOps engine using Git or OCI registries as sources of truth. This ensures reproducibility, auditability, and traceability of change.

For non-production environments, the platform must allow controlled flexibility through ClickOps for rapid experimentation. ClickOps should generate declarative manifests (YAML) wherever possible, ensuring alignment between desired and actual state.

Namespaces must be treated as isolated tenancy constructs. Every project or workload must be deployed into a standardised namespace structure defined by the platform. These namespaces must include predefined quotas, limit ranges, and network policies to prevent resource contention and cross-tenant interference, and ensure compliance with security standards.

All application teams, operators, and platform engineers must interact with Kubernetes through the management plane rather than through ad-hoc tooling or direct API calls to the managed clusters. This ensures a consistent control surface and predictable operational patterns.

## 2.3 Security Requirements

Security must be defined as centrally enforced baseline policy, with **environment-specific overlays** applied where appropriate, such as for cloud, datacentre, or edge environments. Admission controllers must validate workload configuration at deployment time and enforce rules relating to registry access, Pod security, resource constraints, annotations, and compliance requirements.

Only approved registries may be used for workload deployment. Team-specific registries may be permitted if they meet organisational signing and provenance requirements.

Identity integration is essential. Authentication must be delegated to the enterprise identity provider. Authorisation must follow persona-based roles to ensure consistent privilege boundaries. Wherever possible, the platform should use **short-lived, just-in-time credentials** instead of long-lived accounts or tokens.

Auditability is critical. Every change, configuration or operational, must be traceable to a human or automated actor. Production changes must map back to declarative sources of truth, and all operational actions must be logged centrally.

## 2.4 Platform Management Requirements

Multi-cluster operations must be centralised through a single management plane. Each cluster must connect to this plane, and lifecycle operations such as upgrades, configuration distribution, and policy enforcement must be driven centrally. Where organisational or regulatory constraints require segmentation, separate management planes may be used for production and non-production while maintaining consistent governance approaches.

Clusters must be grouped logically by environment, geography, or function. This allows policies, configuration, and access to be applied at an appropriate scope.

Observability must be integrated into the platform. While detailed metrics and traces may remain external, the management plane must provide essential cluster health and log access, and must surface alerts in **machine-readable formats** suitable for aggregation into enterprise monitoring tools.

Routine tasks must not require direct `kubectl` access. Operational controls should be exposed through the management plane to ensure consistency with defined standards and principles.

# 3. Architecture Principles and Guidelines

The principles outlined in this section provide the conceptual foundation for the Kubernetes Management Platform. These principles represent long-lived architectural truths and should not be altered without reassessing the broader platform design. They guide decision-making and ensure consistency across all environments.

## 3.1 Platform Principles

**Declarative production configuration**
All production workloads and cluster configuration must be applied through a declarative model. Production configuration must originate from a GitOps engine, using Git or OCI registries as sources of truth. This ensures production behaves predictably, remains fully auditable, and that configurations are reproducible.

**Controlled flexibility in non production**
Development and test environments require agility to support rapid iteration. The platform must permit ClickOps in non-production, provided such actions generate declarative manifests (YAML) that can be reconciled back into the environment. Guardrails ensure ClickOps cannot introduce unsafe deployments. Audit trails provide visibility, even in non-production.

**Single management plane (with optional segmentation)**
A single management plane should govern configuration, security, and operations across all clusters. Fragmentation of control must be avoided unless explicitly required for regulatory or organisational separation. In such cases, segmented planes (for example, prod vs pre-prod) must follow the same governance and declarative standards.

**Cluster immutability through profiles**
Clusters must be created from predefined profiles that enforce consistent configuration, security, and operational patterns across the fleet. Clusters that diverge from approved profiles introduce risk and elevated management overhead.

**Guardrails and Golden Paths over freedom**
The platform must provide intuitive, safe workflows (Golden Paths) for workload delivery, supported by policy guardrails that prevent unsafe or non-compliant configurations without constraining innovation.

## 3.2 Security Principles

**Security as centrally defined baseline with overlays**
Security posture is defined once as a baseline and propagated to all clusters. Environment-specific overlays may apply additional controls for cloud, datacentre, or edge environments.

**Identity-driven authorisation**
Access must always be derived from the identity provider. Access privileges map directly to persona-based roles and are consistent across all clusters. Short-lived, just-in-time credentials are preferred over long-lived tokens or static kubeconfigs.

**Auditability and transparency**
Every change, whether configuration or operational, must be independently auditable. No undocumented or manual configuration pathways should be used.

## 3.3 Operational Principles

**Namespace isolation**
Workloads must be deployed into purpose-defined namespaces following standard resource constraints and network policies. Only platform components may run in system namespaces.

**Golden Paths and guardrails**
The platform must provide intuitive, opinionated workflows that reduce operational variance while preventing unsafe actions. Guardrails must prevent users from bypassing declared standards.

**Automation by default**
Manual intervention should be the exception. Automated workflows and declarative state reconciliation should underpin both administrator-driven workflows and developer self-service patterns.

# 4. Architecture Standards

Standards represent the enforceable translation of principles into technical behaviour. They define the required configuration and operational expectations for all Kubernetes clusters in the organisation.

## 4.1 Cluster Standards

All clusters must conform to an approved cluster profile. These profiles standardise:

- Operating system and Kubernetes distribution
- Control plane configuration and high-availability mode
- CNI and network policy baseline
- Ingress and Gateway API configuration
- Baseline add-ons (GitOps engine, policy enforcement engine, metrics agents)
- Bootstrap scripts for management-plane enrollment
- CSI block storage and documented storage hardware configuration
- Data protection and snapshot capabilities

Clusters that deviate from approved profiles must not be placed into production.

## 4.2 Deployment Standards

Production deployments must use a GitOps engine sourcing configuration from Git or OCI.

Non production deployments may use ClickOps, Helm workflows, or GitOps, provided actions remain governed and do not bypass standards.

All workloads must follow the organisation’s repository or registry structure (Git or OCI). Manifests must not be deployed directly to production except through the GitOps engine.

## 4.3 Security Standards

- All clusters must run an admission controller in enforcing mode.
- All images must originate from approved or team-scoped registries that meet organisational provenance requirements.
- All workloads must be deployed with defined resource constraints.
- All cluster components must communicate over mutually authenticated channels.
- Security policies must be version-controlled and centrally defined, with environment overlays applied where required.
- The platform must prefer short-lived, just-in-time credentials over long-lived accounts when providing administrative access.

## 4.4 Access & RBAC Standards

- The enterprise identity provider is the single source of identity truth.
- Platform personas map to roles within the management plane.
- Namespace-scoped RBAC is the default access pattern for application teams.
- Direct `cluster-admin` privileges must be limited to platform engineering teams.
- No unmanaged or per-cluster bespoke RBAC objects may be created.

## 4.5 Networking Standards

- The platform must standardise on one or more approved CNIs.
- CIDR ranges must be defined by cluster profile and must not overlap enterprise networks.
- Gateway API with Envoy-class controllers is the preferred ingress/routing model for new deployments.
- Legacy Ingress controllers may be used only where required for compatibility.

## 4.6 Storage & CSI Standards

Clusters must expose a standardised set of StorageClasses (for example, `fast-ssd` , `standard-hdd` , `local-path` ) for cluster-wide persistent storage with documented performance characteristics. Storage solutions are highly-available, distributed to guarantee data resilience and be able to cope with individual node failure; data protection standards below safeguard against other failure modes.

Distributed, shared file systems (such as NFS/SMB) may be used for workloads requiring POSIX semantics but must not be used for databases or stateful transactional workloads.

## 4.7 Data Protection Standards

Clusters must support:

- CSI volume snapshots
- Application-consistent backup workflows requiring quiescing or coordination for databases
- Export of backup artefacts to external storage (S3, Azure Blob, GCS, NFS)
- Periodic restore testing
- Backup portability across clusters and regions

Backup copies and software are a starting point for creating additional copies from production data into non-production environments.

## 4.8 Observability and Alerting Standards

The management plane must surface:

- Node, Pod, and control-plane metrics
- Baseline log forwarding
- Health checks for cluster components
- Alerts for structural failures (control-plane, CNI, admission control, PV failures, certificate expiry)
- Alerts for GitOps reconciliation failure or prolonged drift

Alerts must be output in open, machine-readable formats for aggregation into enterprise monitoring systems.

UI-only alerts are insufficient, as this relies on interactive logins for alerts to be noticed. Out-of-band alerting should be the defacto (eg via slack/teams and ideally via an escalation system eg PagerDuty).

## 4.9 Break-Glass and Recovery Standards

A break-glass access mechanism must exist for emergency scenarios where the management plane or its agents become unavailable. Such access must be tightly controlled, audited, using short-lived credentials and used only under approved procedures.

In the event of a management-plane outage, emergency access to clusters must be available via:

- A documented break-glass kubeconfig
- Time-limited credentials
- A strictly controlled and audited workflow
- Four-eyes principle

Use of break-glass procedures must be minimised and periodically reviewed.

Break-glass access should not be used for any 3rd party application integrations (eg legacy CI/CD). Any use of these tools must route through the Kubernetes Management Platform.

## 4.10 Registry Governance Standards

Registry allow listing must:

- Restrict images to approved sources
- Support team-specific registries if they meet signing/provenance requirements
- Provide flexible patterns for cloud registries (for example, `.azurecr.io` where appropriate)

Supply chain validation should be integrated with external tools where required.

# 5. Reference Architecture

## 5.1 Overview

The Kubernetes Management Platform (KMP) is structured into five architectural planes, each with clearly defined responsibilities. This approach ensures that clusters are built consistently, governed centrally, operated predictably, and integrated cleanly with broader enterprise systems.

While these planes build on one another conceptually, several capabilities, particularly those in the Service Plane, operate as cross-cluster service planes rather than tightly stacked layers. This separation encourages service-oriented thinking and reduces unnecessary coupling.

The KMP provides the essential governance, security, and operational foundations required for Kubernetes at scale, while allowing specialist tools to provide deeper observability, analytics, supply chain validation, and on-call alert routing.

The five planes are:

**Cluster immutability through profiles**
Defines the technical composition of Kubernetes clusters, including operating system, Kubernetes distribution, networking, storage, baseline observability, admission control, and data protection.

**Platform Control Plane**
Provides centralised management and governance for all clusters, including GitOps orchestration, policy distribution, RBAC mapping, access brokering, environment-aware behaviour, and alert routing.

**Service Plane**
The Service layer is where the Kubernetes infrastructure and platform integrate with the broader ecosystem of enterprise systems, including artifact registries, secrets management, certificate management, metrics monitoring, alerting, logging, SIEM, SOC, data protection and more.

**Presentation Plane**
Offers a unified portal and API ensuring consistent access, and auditability, including full visibility of applications, clusters, policies, and audit events. It tailors the user experience to personas and reinforces environment-specific governance (such as declarative-only production).

**Landing Zone**
A fifth, foundational, Landing Zone is assumed as a prerequisite. This includes identity integration, approved regions, security policy baselines, compliance controls, and the creation of core pipelines and repositories. Without this baseline, consistent platform behaviour cannot be guaranteed.

Each plane is described in detail in the following pages.

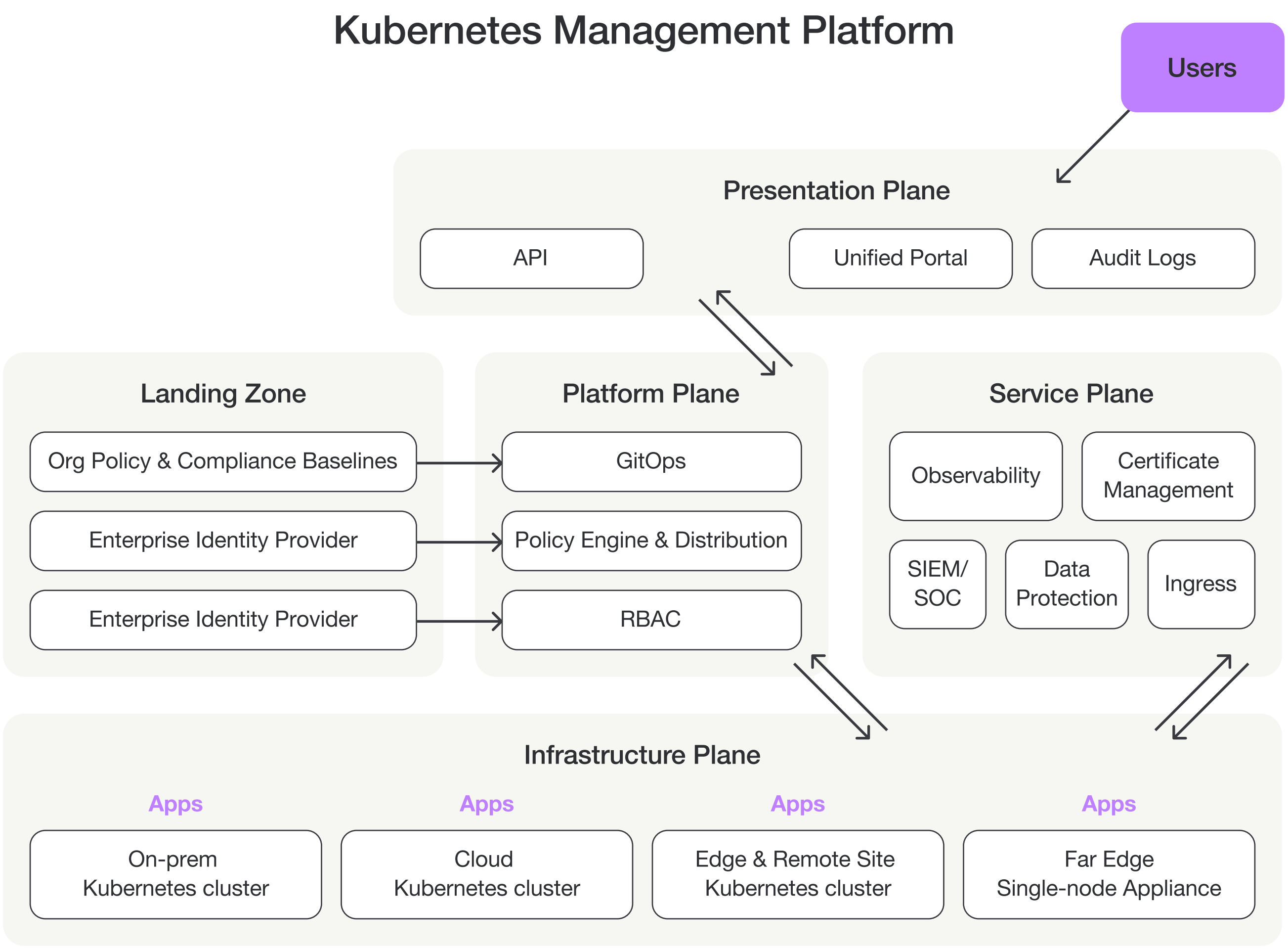
Kubernetes Management Platform
Users
Presentation Plane
API
Unified Portal
Audit Logs
Landing Zone
Org Policy & Compliance Baselines
Enterprise Identity Provider
Enterprise Identity Provider
Platform Plane
GitOps
Policy Engine & Distribution
RBAC
Service Plane
Observability
Certificate Management
SIEM/ SOC
Data Protection
Ingress
Infrastructure Plane
Apps
On-prem Kubernetes cluster
Apps
Cloud Kubernetes cluster
Apps
Edge & Remote Site Kubernetes cluster
Apps
Far Edge Single-node Appliance

**Figure 1** – Conceptual Architecture Using the Planes Model

## 5.2 Infrastructure Plane

The Infrastructure Plane forms the execution environment for all Kubernetes workloads. It encompasses the node operating system, Kubernetes distribution, networking stack, storage layer, admission controllers, baseline observability, and now data protection. All clusters must be created from cluster profiles that define approved configurations for each environment type and prevent drift or inconsistency.

### 5.2.1 Node Operating System

The choice of node operating system directly affects reliability, manageability, and security posture. Two OS families are approved within the KMP.

| OS Class | Description | Typical Use |
|---|---|---|
| **Enterprise Linux** | Traditional Linux distributions with established governance, patching frameworks, operational tooling, and enterprise support models. | Datacentre and cloud clusters requiring alignment with existing enterprise practices and tooling ecosystems. |
| **Kubernetes-Specific OS** | Immutable, image-based OS designed solely to run Kubernetes, removing configuration drift and enabling deterministic node behaviour. | Edge and remote clusters where consistency, reduced attack surface, and low-maintenance node lifecycle are critical. |

All OS images must be produced as golden images, versioned, and maintained centrally.

### 5.2.2 Kubernetes Distribution

The Kubernetes distribution determines how clusters are installed, upgraded, and operated. The reference architecture supports three distribution categories, provided they are CNCF-conformant and upstream-aligned. Clusters must operate within supported version boundaries.

| Distribution Type | Upstream-Aligned Kubernetes |
|---|---|
| **Upstream-Aligned Kubernetes** | Built via kubeadm, Cluster API, Talos APIs, or equivalent lifecycle tooling. Predictable behaviour, no lock-in, and consistent across all environments. |
| **Cloud Managed Kubernetes** | EKS, AKS, GKE. Control-plane management is offloaded to the cloud provider while still supporting KMP governance patterns. |
| **Vendor-Augmented Kubernete** | Enterprise Kubernetes distributions layered on top of upstream Kubernetes. Note that these distributions are often not slimmed down; their optionality poses a potential security issue (due to many more binaries included) and a larger attach surface (due to many more services running) if not adequately managed. |

## 5.2.3 Networking

Networking defines workload connectivity, isolation boundaries, and overall cluster behaviour. To maintain consistency across environments, the reference architecture standardises the networking stack.

- **CNI Options:** Calico or Cilium for datacentre/cloud; Flannel or minimal Cilium for edge.
- **IP Address Management:** Pod and Service CIDRs are defined per cluster profile and must not overlap enterprise networks.

### Ingress and Routing

The **Gateway API** with Envoy-class implementations is preferred for modern routing and multi-tenant isolation.

Legacy Ingress controllers (NGINX-class, Traefik-class) may still be used where required.

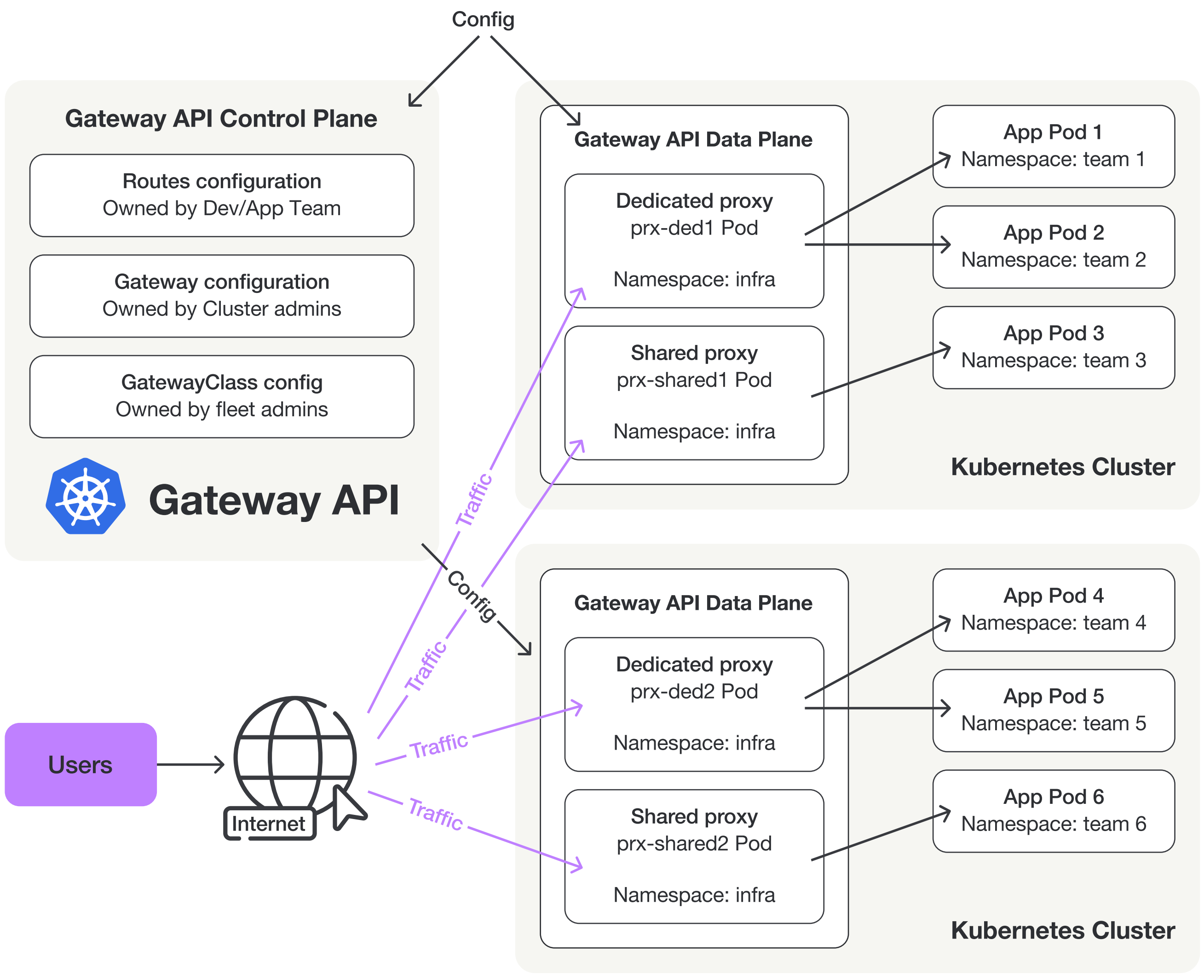

**Figure 2** – Gateway API Logical Routing Model

### 5.2.4 Load Balancing and Ingress

North–south traffic routing must be consistent across clusters.

| Type | Usage |
|---|---|
| Cloud | Native cloud provider load balancers. |
| Datacentre | MetalLB or Kube-Vip, defined by cluster profile. |
| Gateway / Ingress | Gateway API controllers with Envoy recommended. Must integrate with cert-manager and define a GatewayClass. |

### 5.2.5 Storage and CSI

Storage varies across environments but must follow consistent patterns.

| Type | Usage |
|---|---|
| Block Storage | Cloud block, SAN CSI, NVMe CSI for primary workloads. Must support snapshots and expansion. |
| Shared File Storage | NFS/SMB CSI for POSIX workloads (not suitable for databases). |
| Local Storage | Local-path CSI for edge and single-node clusters (with reduced resiliency). |

StorageClasses must be standardised, and performance characteristics clearly documented.

### 5.2.6 Admission Control

Admission controllers provide the enforcement layer preventing non-compliant workloads.

- **OPA Gatekeeper or Kyverno** validates policies at admission time.
- **Policy Bundles** define restrictions, compliance requirements, registry rules, Pod security, resource limits, and required metadata.

Production clusters must run admission controllers in enforcing mode.

Enforcement occurs **in-cluster**, while governance occurs **centrally** in the Platform Control Plane.

### 5.2.7 Baseline Observability and Minimal Alerting

Clusters must include baseline observability components providing visibility into cluster health, not full observability stacks.

**Baseline Includes:**

- Node, Pod, and control-plane metrics
- Basic log forwarding
- Health checking for critical services

**Alerting Philosophy:**

Only structural, high-severity alerts should be raised by the KMP.

**Examples:**

- Control-plane degradation
- Node pressure or unschedulable nodes
- PV provisioning or attach failure
- Admission controller malfunction
- CNI failure
- Certificate expiry risk
- GitOps reconciliation failure
- Prolonged drift between desired and actual state

Alerts must be emitted in **machine-readable, open formats** suitable for aggregation into central enterprise monitoring tools.

Alerts should be configured for up-stream forwarding to notification engines (such as Slack/Teams/PagerDuty)

UI-only surfacing of alerts is insufficient due to the reliance on interactive logins for alerts to be noticed.

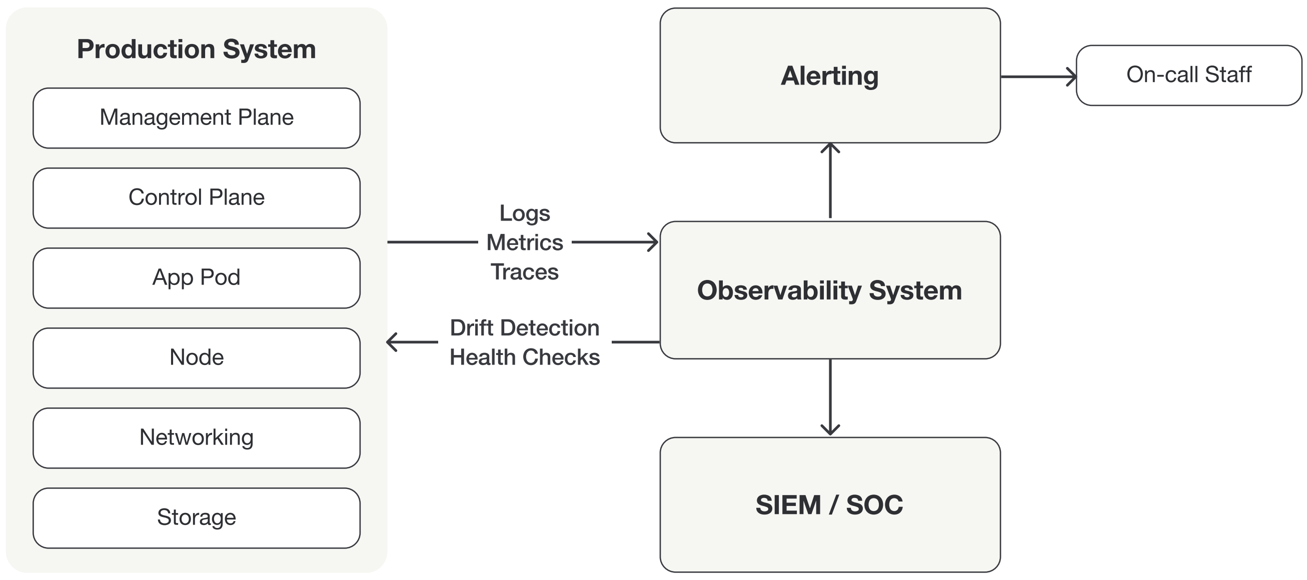

**Figure 3** – Alert Export and Aggregation Pipeline

### 5.2.8 Data Protection and Recovery

Data protection is a first-class concern of the Infrastructure Plane.

**The platform must support:**

- CSI snapshot capability for persistent volumes
- Scheduled and on-demand snapshots
- Export of backup data to external profiles (S3, Azure Blob, GCS, NFS)
- Application-consistent backups, including workload quiescing for databases
- Periodic restore testing as part of governance
- Portability of backup artefacts across clusters and regions

Snapshots alone do not guarantee recoverability for transactional systems.

Backups act as the version-control mechanism for persistent state, complementing declarative configuration.

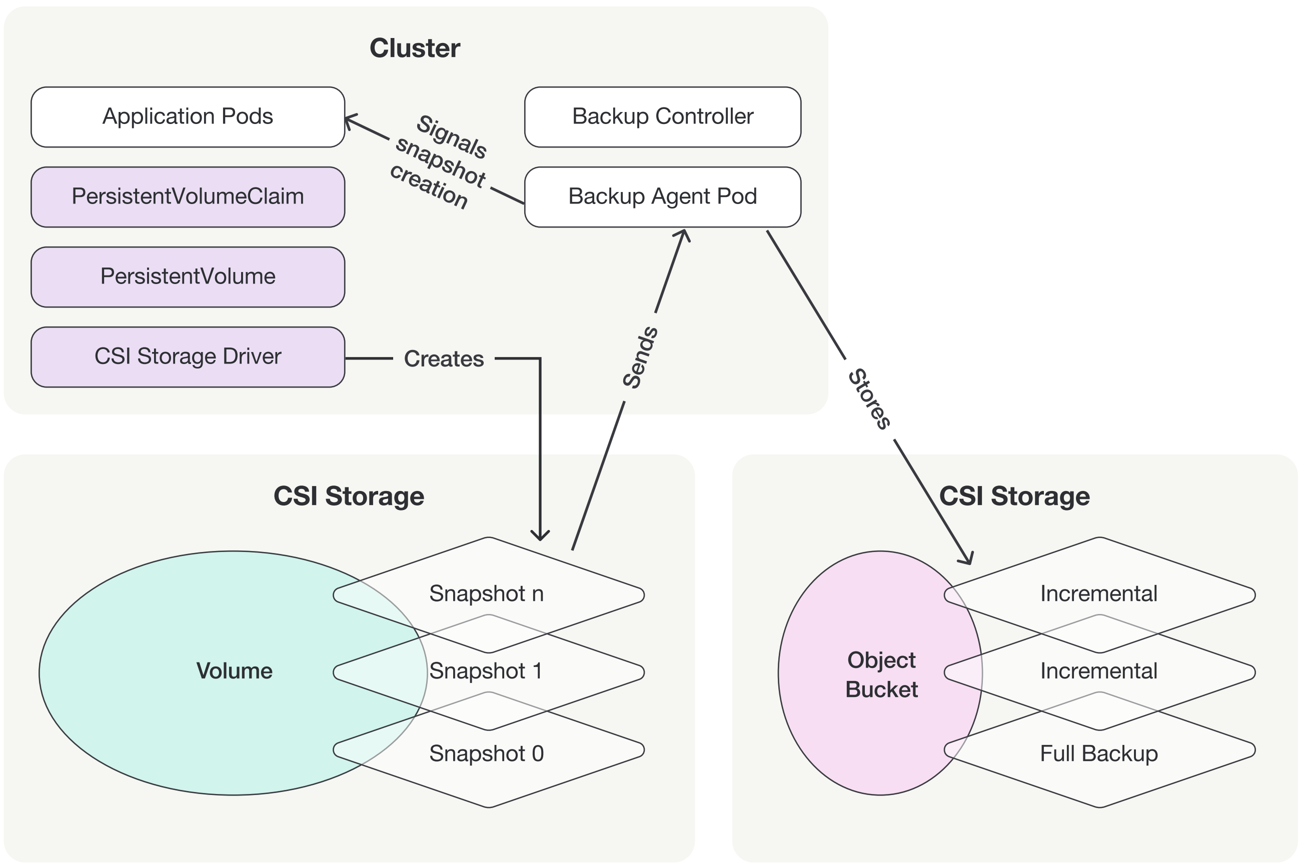

**Figure 4** – Application-Consistent Backup Flow

### 5.2.9 Cluster Profiles

Cluster profiles encapsulate all Infrastructure Plane standards in a repeatable, enforceable template.

Profiles include:

- Production
- Non-production
- Edge / far-edge

Clusters must not diverge from their assigned profile.

## 5.3 Platform Control Plane

The Platform Control Plane provides cross-cluster governance and orchestration. All human and system actions flow through this plane to ensure consistent behaviour, auditability, and policy compliance.

### Platform Control Responsibilities

**Cluster Registry**
Maintains metadata, lifecycle state, environment classification, region, and profile for every cluster.

**Cluster Access Broker**
Intermediates all operational actions. Validates identity, authorisation, policy compatibility, and logs all actions. Users do not interact with the Kubernetes API directly.

A break-glass pathway exists but must be controlled and auditable.

**GitOps Orchestration**
Monitors declarative sources (Git or OCI) and reconciles changes into clusters.
Production is strict-declarative; non production may use mixed-mode with ClickOps generating manifests.

**Policy Distribution Engine**
Stores and distributes policy bundles to admission controllers across all clusters.

**RBAC Mapper**
Aligns identity provider groups to platform personas and Kubernetes roles.

**Environment Policy Engine**
Enforces environment-specific behaviour, such as declarative-only production.

**Alert Routing**
Normalises high-severity alerts and forwards them to enterprise monitoring systems.

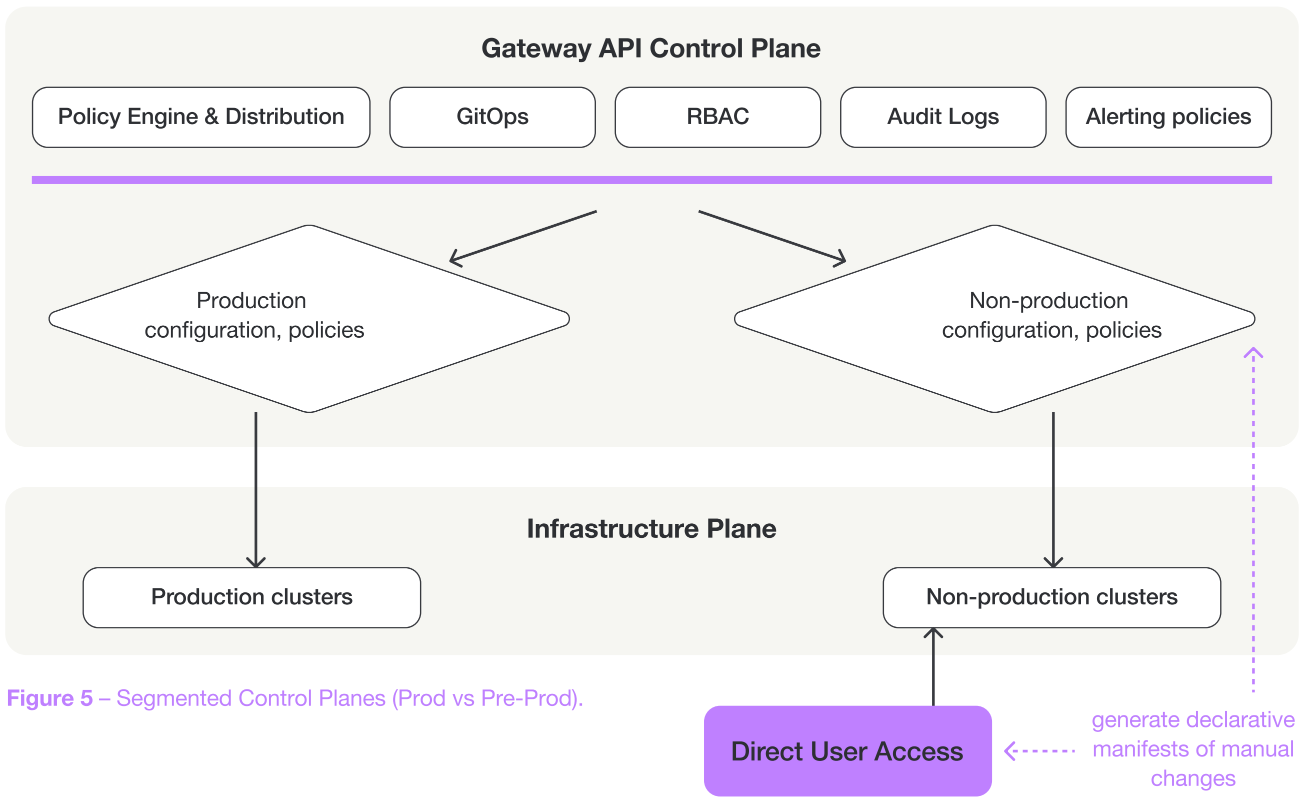
Gateway API Control Plane
Policy Engine & Distribution
GitOps
RBAC
Audit Logs
Alerting policies
Production configuration, policies
Non-production configuration, policies
Infrastructure Plane
Production clusters
Non-production clusters
Direct User Access
generate declarative manifests of manual changes

**Figure 5** – Segmented Control Planes (Prod vs Pre-Prod).

## 5.4 Service Plane

The Service Plane provides external services used through-out the organisation, providing central services including:

**Logging**
Provides baseline logs across clusters, with long-term retention and deep querying delegated to external tools.

**Metrics**
Exposes baseline metrics, while deep analytics and APM remain external.

**Alert Routing**
Handles critical platform alerts, forwarding them to enterprise alert aggregation tools.

**Registry Controls**
Manages allow-listed registries, provenance requirements, and team-specific registry rules.

**Certificates**
Integrates with cert-manager and external PKI systems for certificate issuance and rotation.

**Backup Integration**
Coordinates snapshot and backup workflows, while full DR orchestration remains with specialist tools.

**Audit Streaming**
Streams all user actions, RBAC changes, policy evaluations, and GitOps events to SIEM.

## 5.5 Presentation Plane

The Presentation Plane exposes KMP capabilities through a unified UI and API.

Responsibilities include:

- Unified portal showing clusters, namespaces, applications, GitOps state, policies, and health
- Central API enabling CI/CD and automation under the same governance model
- Audit visibility, including GitOps history and policy violations
- Persona-based views (Developer, Ops, Platform Engineer, Edge Operator)
- Environment-aware UI controls—limited in production, flexible in non-production
- Offline-first and asynchronous workflows for far-edge environments

# 6. Target State Architecture Using the Portainer Suite of Products

## 6.1 Overview

The target state architecture is implemented through the Portainer suite, which combines the following products:

- Portainer Business as the central management control plane including GitOps, policy governance, identity integration, and fleet visibility.
- Talos OS as a secure and deterministic host operating system for upstream Kubernetes; supports the Production Cluster Profile with consistent lifecycle management.
- KubeSolo as a lightweight Kubernetes runtime for far-edge environments requiring single-node configurations on vendor-supplied Linux operating systems.
- Portainer Agents and Edge Agents provide secure, outbound-only connectivity from clusters to the control plane, enabling consistent governance across remote, intermittent, or untrusted network environments.

The suite aligns directly with the plane model defined in Section 5. Each component contributes to a specific plane while maintaining predictability, upstream compatibility, and operational simplicity. This alignment provides a coherent approach to governance, lifecycle management, security, and workload operations across the entire fleet.

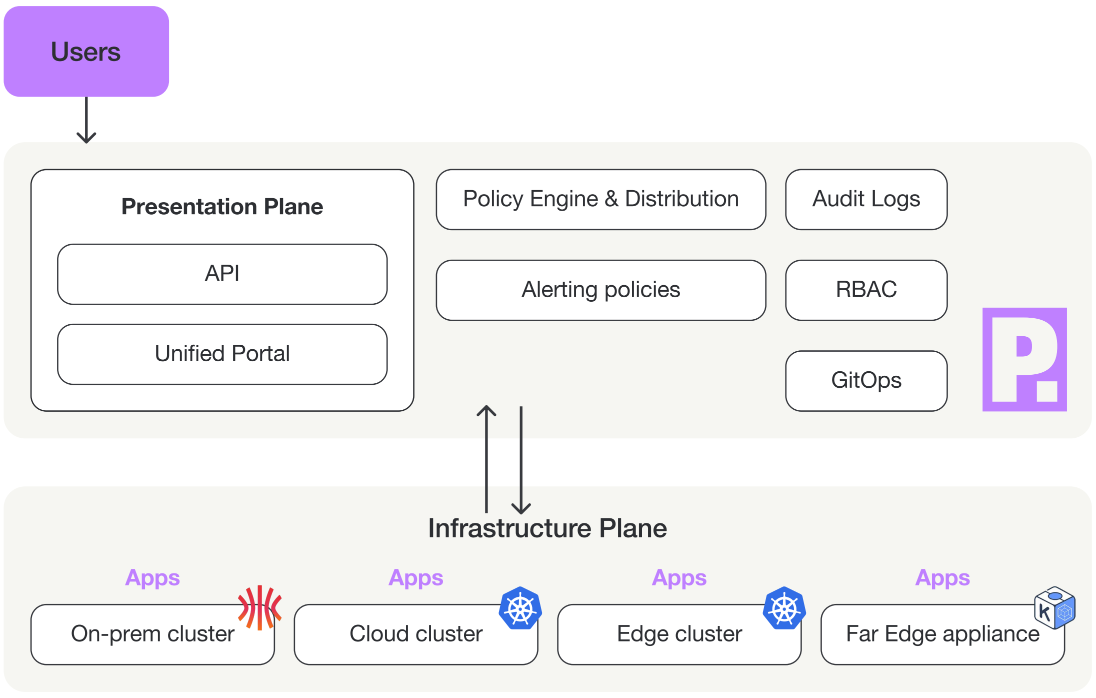

**Figure 6** – Portainer Suite Logical Architecture

## 6.2 Infrastructure Plane Alignment

The Infrastructure Plane describes how clusters are constructed, connected, and secured. The Portainer Suite implements this plane through upstream Kubernetes, Talos OS for host lifecycle, KubeSolo for far-edge environments, and Portainer Agents for controlled connectivity.

### Node Operating System and Lifecycle

**Talos OS**
Used where a secure, immutable, API-driven OS is required. Talos installs and maintains upstream Kubernetes and does not function as a distribution. Its determinism and reduced configuration drift make it suitable for the Production Cluster Profile and for edge environments where physical access or tampering risks are higher.

**KubeSolo**
Used for far-edge environments where the underlying OS is supplied by the hardware vendor and cannot be replaced. KubeSolo runs as a single-node Kubernetes runtime and is fully governed through Portainer.

### Cluster Bootstrap and Connectivity

Clusters connect to Portainer using one of three agent models:

| Component | Use Case |
|---|---|
| Portainer Agent | Standard connectivity for datacentre and cloud clusters. Relies on the network security afforded by a dedicated “Out of Band” management network. |
| Portainer Edge Agent | Secure, outbound-only connectivity where direct inbound access is not possible, or where administrative commands would transit insecure networks. |
| Portainer Async Edge Agent | Designed for highly distributed or intermittently connected far-edge sites. |

All agents communicate **outbound only,** reducing exposure and removing the need to open inbound network paths. This supports remote industrial networks, cellular links, and OT environments where trust boundaries are strict.

### Compatibility with Networking, Storage, and Ingress

Portainer does not impose specific CNIs, CSIs, or ingress controllers. Clusters rely on the networking and storage standards defined in Section 5. This ensures compatibility with:

- Calico, Cilium, Flannel
- NGINX-class, Traefik-class, Envoy, and **Gateway API** implementations
- Block, file, and local-path CSI drivers
- Edge-appropriate CSI patterns

This separation preserves the reference architecture requirement that governance must remain independent from the underlying data plane technologies.

### Admission Control and Policy Enforcement

Portainer governs the configuration of the admission control layer. It natively supports OPA Gatekeeper and will deploy it to clusters as defined in the cluster profile. Enforcement occurs within the cluster via OPA, while governance (policy bundles, versions, visibility) is managed centrally in the Platform Control Plane.

This satisfies the architectural requirement that enforcement be decentralised but governed centrally.

## 6.3 Platform Control Plane Alignment

Portainer Business provides the full capability set required by the Platform Control Plane. It governs all clusters through a consistent, unified control surface that enforces identity, policy, declarative configuration, and controlled ClickOps workflows.

### Cluster Registry and Metadata

Portainer maintains an inventory of all clusters as managed environments. Metadata includes region, profile, version, environment type, and connectivity state. Environment Groups map clusters into logical fleet segments such as production, non-production, geographic, or far-edge estates.

### Cluster Access Broker

Portainer acts as an access broker for all cluster interactions. It validates the identity of the user or system actor, authorisation scope, and policy compatibility, and logs all operations. Users do not interact directly with Kubernetes APIs.

A break-glass access pathway exists for emergencies but must be tightly controlled and auditable.

### GitOps Orchestration

Portainer includes a built-in GitOps engine supporting both Git and OCI as declarative sources. It reconciles changes into clusters at:

- Namespace scope
- Application scope
- Full cluster configuration scope

Production clusters operate in strict declarative mode. Non-production environments may use hybrid modes (ClickOps + declarative manifests).

### Policy Distribution

Portainer manages the storage and distribution of policy bundles consumed by OPA. It provides visibility into compliance, drift, and enforcement outcomes.

### Identity and Role Mapping

Portainer integrates with enterprise identity providers through OIDC or LDAP. Teams and Roles map to persona-based access boundaries spanning clusters, namespaces, and environment groups.

Short-lived credentials and time-bounded access sessions are encouraged.

### Environment-Aware Behaviour

Portainer enforces environment-specific deployment rules such as:

- Declarative-only production
- Controlled ClickOps in non-production
- Mixed connectivity and asynchronous operation in far-edge deployments

### Alert Routing

Portainer monitors for high-severity platform alerts (control-plane issues, CNI failure, certificate expiration, GitOps drift, etc.) and forwards them to enterprise monitoring systems. Portainer is not intended to replace specialist alerting platforms.

## 6.4 Service Plane Alignment

The Service Plane provides cross-cluster capabilities that complement governance and operational workflows. Portainer contributes governance-focused services, while deep analytics remain external.

### Logging and Metrics Integration

Portainer provides baseline visibility into node health, Pod health, resource usage, and cluster logs. Integration points allow redirection to enterprise observability platforms

### Registry Governance

Portainer manages allow-listed registries and image provenance rules. This includes support for:

- Central registries
- Team-specific registries
- Cloud registries (for example, wildcard allowlisting such as `.azurecr.io` )

Vulnerability scanning and supply chain analytics remain external.

### Certificate and Secret Management

Portainer integrates with cert-manager and external PKI systems to manage certificate issuance and rotation. Secret distribution aligns with Kubernetes native mechanisms or external secret managers.

### Audit Streaming to SIEM

Portainer emits a unified audit log including:

- User activity
- RBAC changes
- Policy events
- GitOps synchronisation
- Operational actions

These logs can be streamed to SIEM for correlation and long-term analysis.

### Backup and Recovery Integration

Portainer integrates with CSI-compatible tools for backup orchestration. While DR orchestration remains the responsibility of specialist tools, Portainer ensures that declarative configuration and policy definitions remain portable and recoverable.

## 6.5 Presentation Plane Alignment

Portainer Business provides the portal and API endpoint serving as the Presentation Plane for the KMP.

### Responsibilities

- Unified portal displaying clusters, namespaces, applications, policies, GitOps state, and edge connectivity
- Persona-based visibility and action scoping
- Central API for automation frameworks
- Environment-aware UI behaviour, with restricted operations in production
- Asynchronous, offline-first workflows for far-edge and intermittently connected sites
- Clear activity logs and event histories with SIEM export

The Presentation Plane ensures that all operational and deployment activities occur within the governance boundaries defined by the platform.

## 6.6 End-to-End Operational Integration

### High Availability and Metadata Management

Portainer supports high availability through load-balanced instances backed by an external database. This ensures continuity of governance functions.

### Cluster Lifecycle with Talos OS

Portainer integrates with the Talos API for node creation, upgrades, remediation, and replacement. Kubernetes remains upstream and CNCF conformant during all lifecycle operations.

### Far-Edge Lifecycle with KubeSolo

KubeSolo operates on top of vendor-supplied Linux OS images commonly found in industrial devices. Portainer governs these environments through Edge Agents or Async Edge Agents, providing policy, deployment, and configuration management even across unstable networks.

### Drift Minimisation

Drift is minimised through:

- Talos host immutability
- Portainer GitOps reconciliation
- Policy enforcement at admission control

### Operation in Restricted Networks

Portainer's outbound-only connectivity model supports:

- Strict firewalls
- Network isolation
- Cellular and satellite networks
- OT segmentation
- Airgapped environments (with intermittent sync through Async Edge Agents)

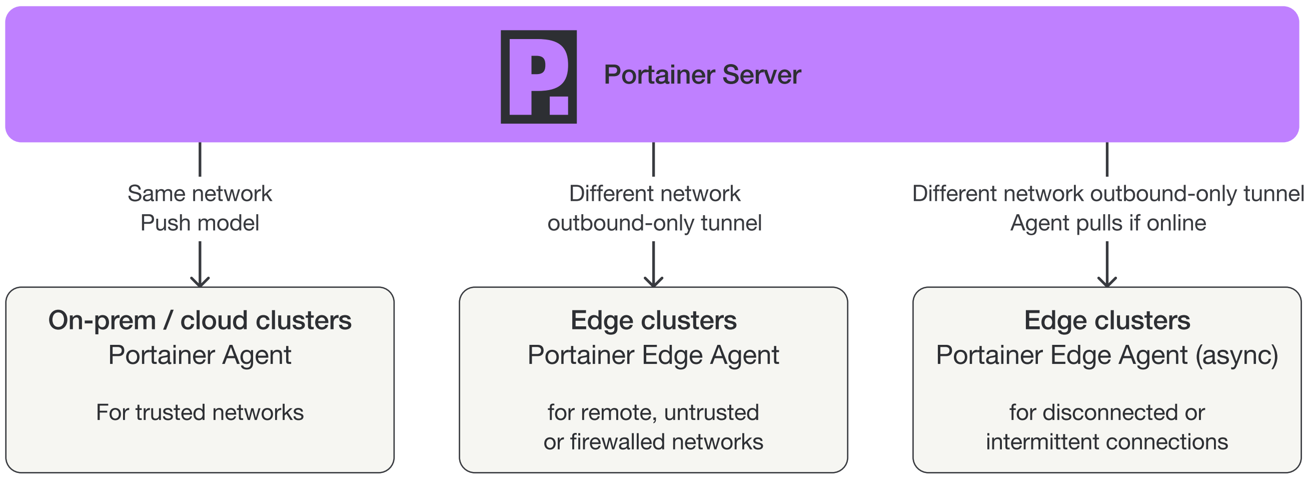

**Figure 7** – Edge Connectivity and Offline Operations Diagram.

## Appendix – Diagram Summary

The following diagrams are referenced throughout the document and should be created to support the architecture narrative:

PORTAINER.iO