

OneK8s

One Kubernetes platform, one application contract for every cloud - built on Azure (AKS) first.

Platform design, version 4.5, built 2026-10-26 from commit [9f1f777](#).

The current version is published at <https://onek8s.github.io>; the changes between versions are in [8.1 Release notes](#).

Contents

1. Start here

- [1.0 Summary](#)
- [1.1 Overview](#)
- [1.2 How the design meets the requirements](#)
- [1.3 Open standards and portability](#)
- [1.4 The application contract](#)
- [1.5 Developer experience: onboarding, migration and operations](#)

2. Inside a cluster

- [2.1 Network layout](#)
- [2.2 Control plane](#)
- [2.3 Node pools](#)
- [2.4 Allowed and blocked flows](#)
- [2.5 Workload identity and secrets](#)
- [2.6 Application multi-tenancy](#)
- [2.7 Policy enforcement](#)

3. The fleet

- [3.1 Cluster types: stateless and stateful](#)
- [3.2 Environments](#)
- [3.3 Keeping clusters in sync](#)
- [3.4 Zero-downtime application upgrades](#)
- [3.5 Zero-downtime cluster upgrades](#)

4. Encryption

- [4.1 Encryption in transit and TLS](#)
- [4.2 Encryption at rest](#)

5. Egress control

- [5.1 Egress control and the company firewall](#)
- [5.2 Traffic flow documentation](#)

6. Operations

- [6.1 AI-driven day 2 operations](#)
- [6.2 Observability with Grafana Cloud](#)
- [6.3 Backup and region recovery](#)

8. Appendix

- [8.1 Release notes](#)

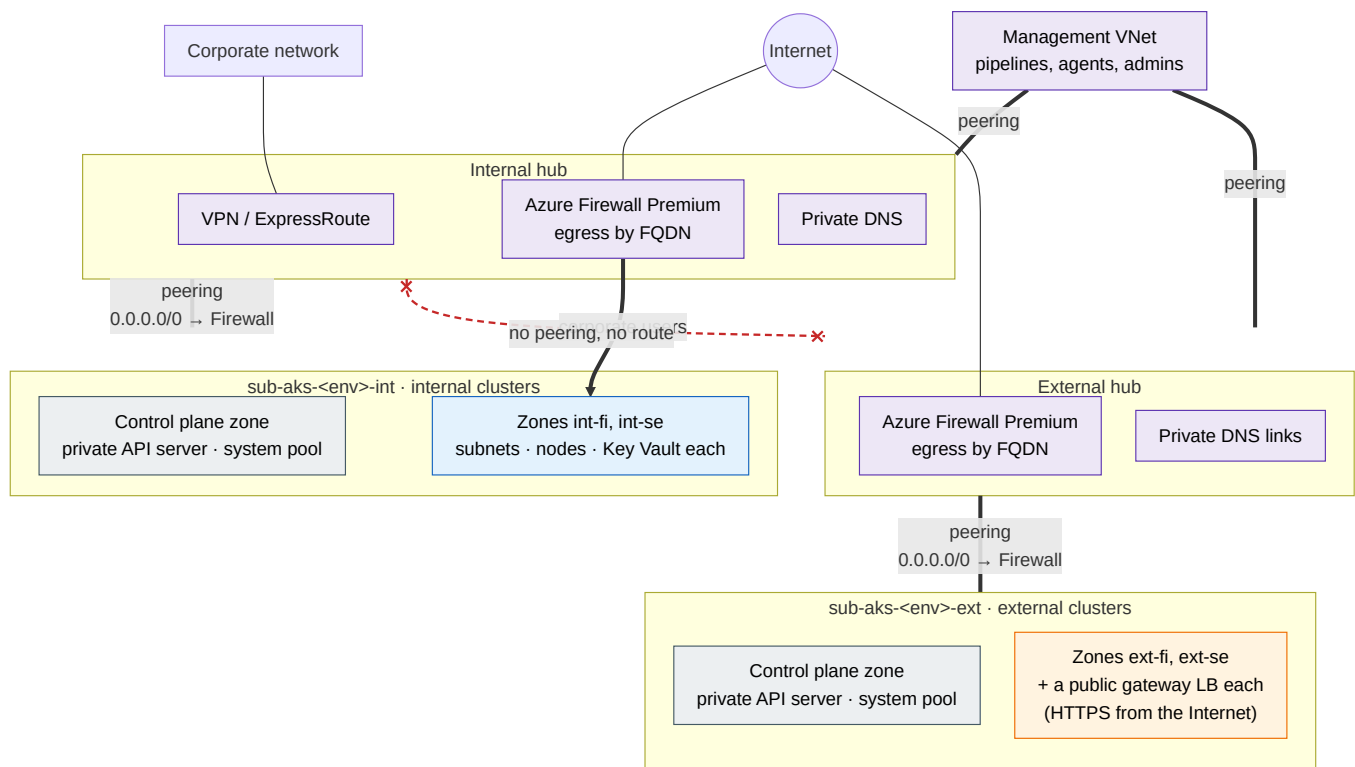
1.0 Summary

Requirements: [R1](#), [R2](#), [R3](#), [R4](#), [R6](#), [R7](#), [R10](#), [R11](#), [R12](#), [R13](#), [R14](#), [R16](#), [R17](#), [R18](#), [R19](#), [R22](#), [R23](#), [R25](#), [R26](#), [R27](#), [R28](#), [R29](#), [R30](#), [R31](#), [R32](#).

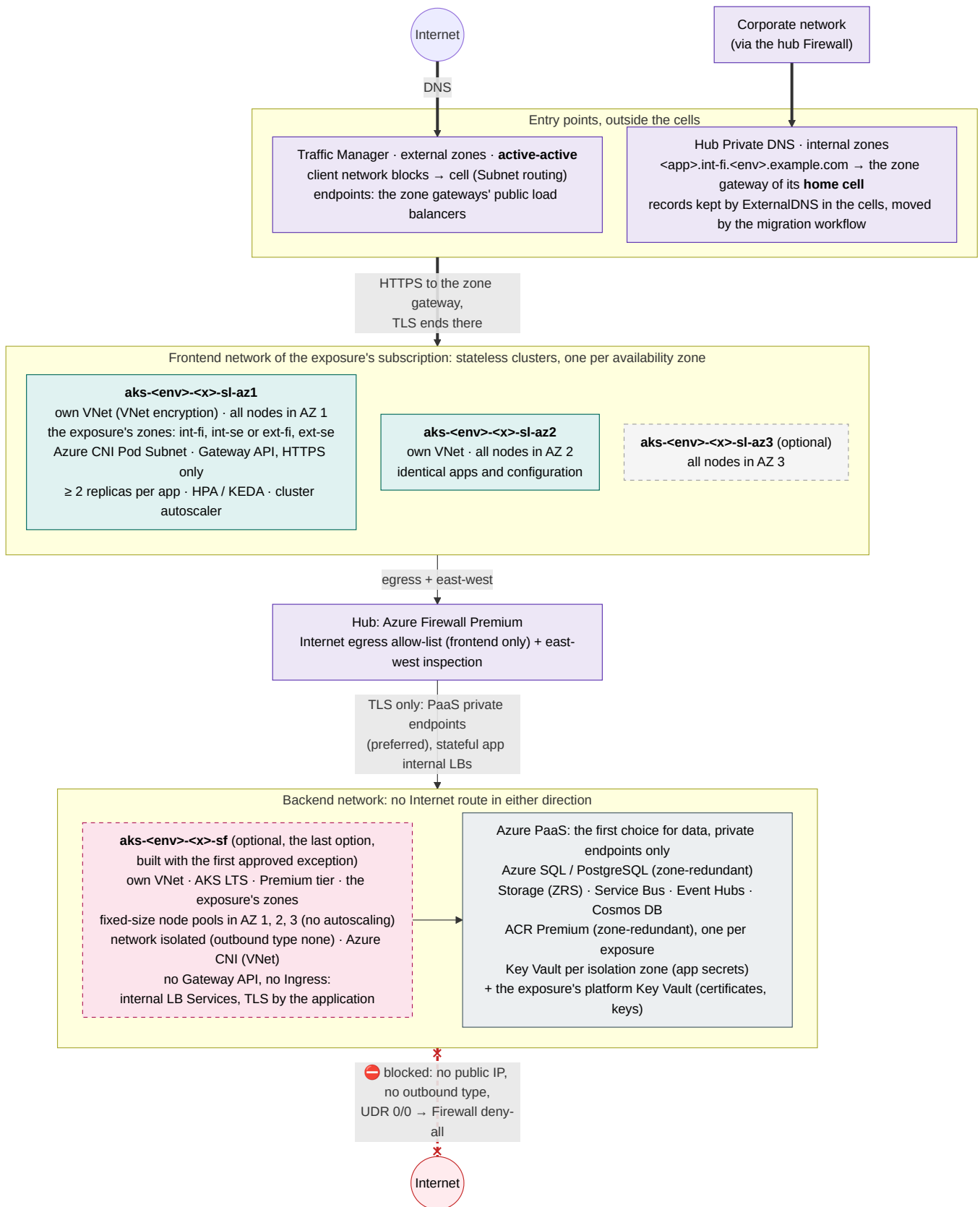
The whole design in a few pages, for the reader who will not read the rest: what the platform is, the choices that shape it, and what it costs. Every line links to the section that holds the detail; this page adds nothing the sections do not say. It opens the [design PDF](#).

What it is

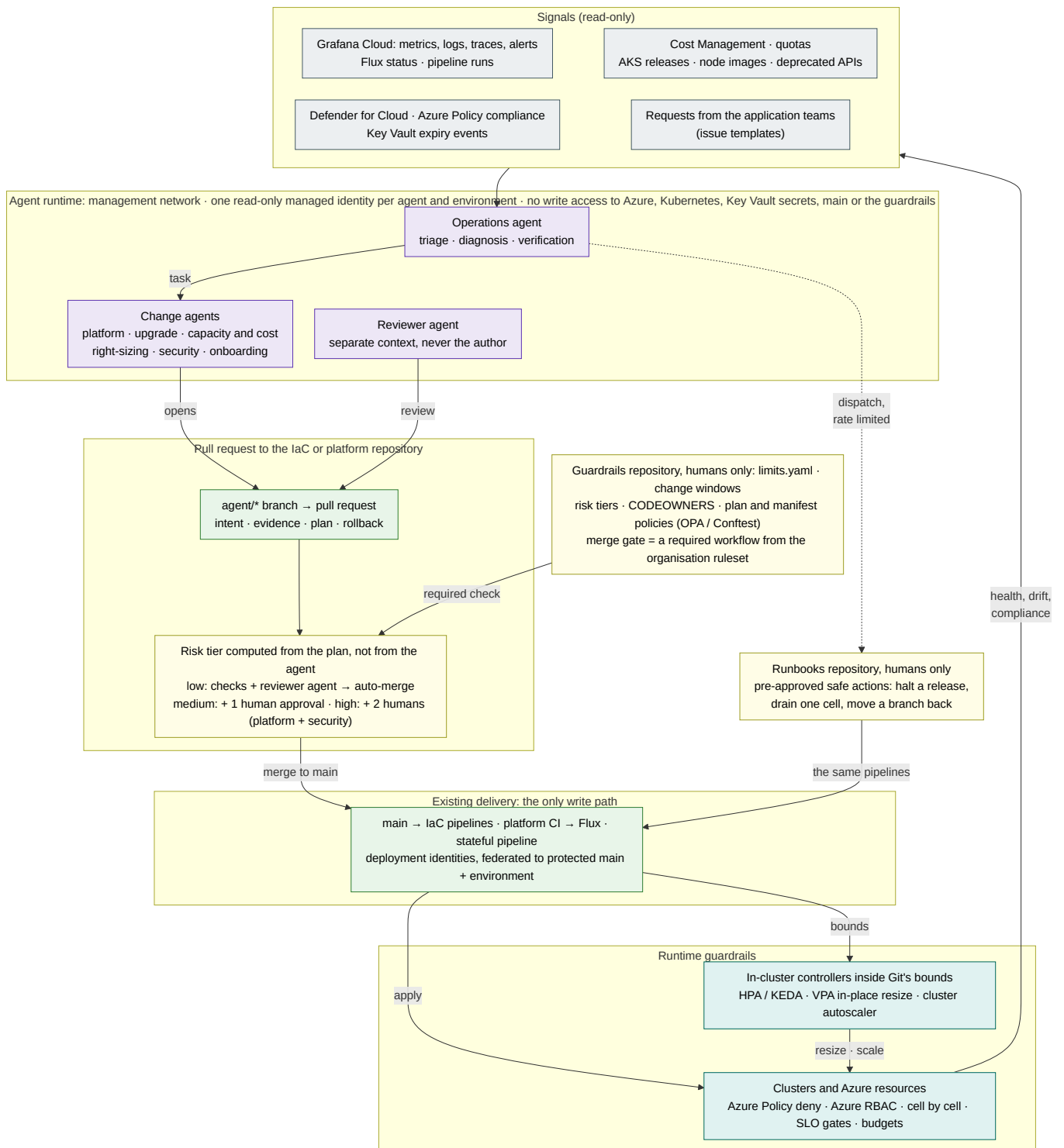
One Kubernetes platform, one application contract for every cloud – built on Azure (AKS) first. Applications, their manifests and the platform's in-cluster configuration use only open standards – Kubernetes, OCI, Flux, the Gateway API, External Secrets, OIDC workload identity, OpenTelemetry – and data lives in PaaS services that every major cloud offers over the same protocol. The contract is portable today ([section 1.3](#), [R16](#)); the platform under it is an AKS platform that leans on what Azure manages, and a second cloud is another implementation of it.



The landing zone is hub-and-spoke with **one hub and one Azure Firewall Premium per exposure** ([section 1.1](#)). Inside a cluster, countries are **isolation zones**: their own nodes, subnets, route tables, Key Vault and policies, with the hub Firewall between them ([section 2.1](#), [section 2.4](#)). Inside a zone, every application has its own namespace with default-deny policies, its own workload identity and its own share of the zone's vault ([section 2.5](#), [section 2.6](#)).



The fleet is **fourteen clusters, twenty with the optional stateful clusters**: per exposure and environment (dev, acc, prd) two or three **stateless cells** that can be drained, upgraded or rebuilt at any time, and a **stateful cluster** only when a workload needs it ([section 3.1](#), [section 3.2](#)). An internal application runs in one home cell and is migrated between cells. Everything in a cluster comes from Git, delivered by Flux to the cells and by a pipeline to the stateful cluster ([section 3.3](#)); releases and cluster upgrades move through the cells one at a time, behind the cell router ([section 3.4](#), [section 3.5](#)).



Day-2 operations are run by **agents that change the platform only by pull request**, behind a merge gate that humans own (section 6.1). Telemetry goes to one Grafana Cloud stack per environment through Private Link (section 6.2); only persistent data is backed up, immutably, into the paired region, and clusters are rebuilt from IaC and Git (section 6.3).

The choices that shape it

- **Cells and zones:** one cluster per cell, the countries as isolation zones inside it; internal and external in separate subscriptions and clusters, with no route between the two hubs (section 3.1).

- **One cell network:** Azure CNI Pod Subnet with Cilium, every pod carrying its zone's address, so every flow between two zones of a cell and every flow out of the cell meets the hub Firewall, which decides egress per application from rule files ([section 2.1](#), [section 5.1](#)); classic node pools with the cluster autoscaler ([section 2.3](#)).
- **Private by default:** no public network access for any platform service, and private API servers over Private Link ([section 2.2](#)).
- **Cells and update speeds:** three cells in prd, two in dev and acc; two update speeds, with node images pinned per cell ([section 3.5](#)).
- **The edge:** Gateway API only, the implementation a per-cluster setting; HTTP/2 and TLS 1.3 only in the whole gateway path, Traefik in every cell ([section 4.1](#)).
- **Secrets and access:** secrets only through External Secrets Operator, separated per namespace by Key Vault ABAC; namespace roles from Entra ID groups with matching vault rights; tenants list nothing ([section 2.5](#)).
- **Keys:** customer-managed HSM keys per control plane, per cluster's node disks and per zone's volumes, with KMS data encryption ([section 4.2](#)).
- **The stateful cluster:** built only with the first approved exception, on an LTS minor ([section 3.1](#)).
- **Agents:** they start with a human on every tier above low and earn trust per agent and environment ([section 6.1](#)).

What it costs

The structure is fixed; the numbers are measured in acceptance ([section 3.2](#)).

- **Fixed per cell:** the system pool, the gateways and their load balancers, private endpoints, the hub peering, the Standard tier – paid per cell and exposure, which is why the third cell in prd costs only its fixed part.
- **Fixed per environment:** the hub share (a Firewall Premium per exposure, about 1,300 USD a month list each), Key Vaults, the registry (Premium only in prd), Traffic Manager, backups, the Grafana Cloud stack, Log Analytics.
- **Variable:** the zone nodes on load (on-demand and spot, amd64 and arm64, chosen per application), PaaS, telemetry volume, Firewall data processing.

Capacity is N+1 per environment: with k cells every cell can carry $1/(k-1)$ of the environment on top of its own share, on on-demand capacity alone.

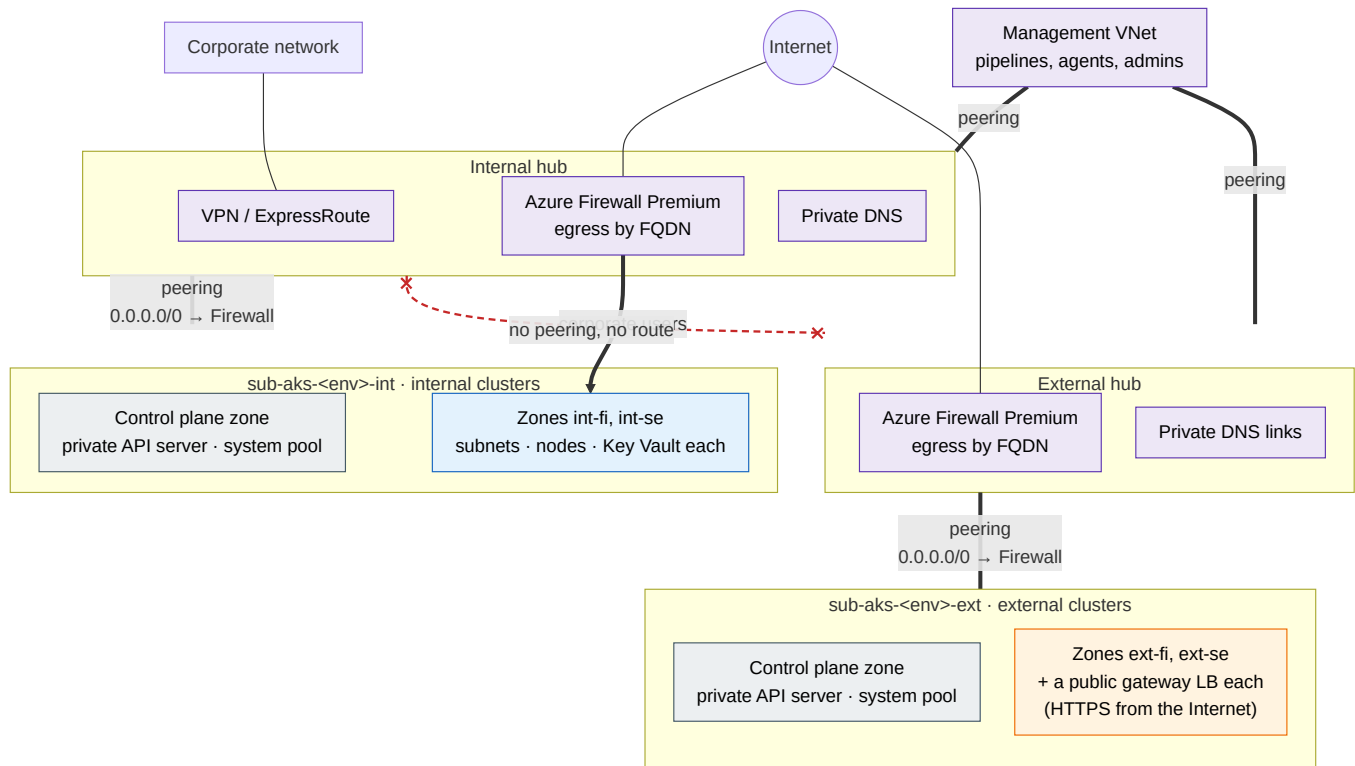
How to read the rest

The side navigation lists every page, and the [front page](#) says who reads what; how humans and AI share the work is [how we work](#). The design is one PDF, every page of it with this summary first ([onek8s-design.pdf](#)).

1. START HERE

1.1 Overview

Requirements: [R1](#), [R2](#), [R4](#), [R7](#), [R19](#).



Hub-and-spoke landing zone with **one hub per exposure** and **internal and external workloads in separate subscriptions and clusters**. What the picture does not show: the internal hub's Firewall inspects all egress and east-west traffic, and the hub also holds Bastion. With no gateway in the external hub and no peering between the hubs, the clusters the Internet reaches have no path to the corporate network at all. The management VNet hosts jump hosts and CI/CD agents; peering is not transitive, so it joins nothing, and it is the only place the Kubernetes API can be reached from. Every cluster of the fleet - stateless or stateful, internal or external - has this same inner layout; how the clusters are placed in the frontend and backend networks is shown in [picture 3.1](#). External zones are reached on a public load balancer of the zone's gateway in every cell, with Azure Traffic Manager choosing the cell by DNS; internal zones directly at the zone's gateway in the cell that hosts the application (private DNS kept up to date by ExternalDNS). In both cases the traffic ends at the zone's gateway in a stateless cluster - the stateful cluster is never reached from outside the platform.

1.2 How the design meets the requirements

Requirements: [R1](#), [R2](#), [R3](#), [R4](#), [R5](#), [R6](#), [R7](#), [R8](#), [R9](#), [R10](#), [R11](#), [R12](#), [R13](#), [R14](#), [R15](#), [R16](#), [R17](#), [R18](#), [R19](#), [R20](#), [R21](#), [R22](#), [R23](#), [R24](#), [R25](#), [R26](#), [R27](#), [R28](#), [R29](#), [R30](#), [R31](#), [R32](#), [R33](#), [R34](#).

The [requirements](#) are owned by humans; this page, like the rest of the design, shows how the design meets each of them ([how we work](#)). The [acceptance criteria](#) prove it: the requirement and isolation tests are tagged with the requirement number they cover and run after every change of a cell.

#	How the design meets it
R1	A subscription per environment and exposure (<code>sub-aks-<env>-int</code> , <code>sub-aks-<env>-ext</code>), each with its own clusters, VNets, Key Vaults, ACR and policy assignments; zones <code>int-*</code> only in internal clusters, <code>ext-*</code> only in external ones (section 3.1)
R2	Separate isolation zones per country (<code>*-fi</code> , <code>*-se</code> , ...) inside the clusters of each exposure
R3	Every zone has its own Key Vault, subnets (NSG + route table) and AKS node pools (section 2.3); certificates are the one shared exception, kept in the per-environment platform Key Vault with a role assignment per certificate
R4	Deny-by-default on three layers: NSG, Azure Firewall (each zone's route table takes zone-to-zone traffic to it, pods included because they carry their zone's addresses), Cilium network policy (picture 2.4); an application reaches another zone's application only as a client of its published entry point, with a Firewall rule like any external API (section 2.1)
R5	A zone is a slot of every cell's address space with the same index in every cell; a new country declares two zones (one per exposure) and gets its subnets, NSGs, node pools, policies and Firewall rules without re-addressing or routing changes (adding a country)
R6	Azure CNI Pod Subnet powered by Cilium in every cluster: Cilium's layer 3 and 4 policies for the zone baseline and namespace isolation, kube-proxy replaced; Advanced Container Networking Services (FQDN filtering, Hubble, eBPF host routing) is not used (section 5.1)
R7	A private cluster over Private Link, the endpoint in the cell's system subnet and no public endpoint; reached only from the management network through the hub Firewall (section 2.2)
R8	Pod labels <code>platform/arch</code> and <code>platform/capacity</code> , turned into node selection by Azure Policy mutation; per zone an on-demand and a tainted spot pool for both architectures with fallback to on-demand; defaults amd64 on-demand; stateful cluster on-demand amd64 only (section 2.3)
R9	Per-application workload identity with Entra ID auth to all Azure services, local auth disabled by Azure Policy; Key Vault secrets only for targets without Entra ID (section 2.5)
R10	Per-application secret reader identity, usable only by External Secrets Operator, with secret-name-prefix conditions (picture 2.5)
R11	One ESO controller per zone reads Key Vault on behalf of the applications; application identities have no Key Vault role, application pods cannot use the reader identity or reach a vault, only ESO may write Secrets in application namespaces, and the Secrets Store CSI driver add-on is disabled; the zone's gateway uses ESO like an application, other platform components may use Key Vault directly with their own narrow roles (section 2.5)

#	How the design meets it
R12	One shared platform Key Vault per environment and exposure (<code>kv-<env>-<exposure>-platform</code>) holds all certificates of that exposure's zones; a renewal job per environment issues one certificate per zone into the vault of the zone's exposure, and External Secrets Operator in every cluster of the exposure reads it with plain Azure RBAC scoped to the individual certificate (section 4.1)
R13	Namespace per application, default-deny policies (picture 2.6); egress out of the cluster only through the hub Firewall's rules for the zone (R29)
R14	Every namespace carries <code>platform/team</code> , <code>platform/cost-center</code> and the annotation <code>platform/contact</code> , set at onboarding and required by Azure Policy; OpenCost splits each cell's cost by namespace and so by team and cost centre, in Grafana Cloud; a monthly budget per environment subscription alerts the owners at 80 % and 100 % (section 2.6)
R15	Azure Policy add-on for AKS in every cluster, assigned per environment subscription; the same Azure Policy also governs the Azure resources (section 2.7)
R16	Applications, their manifests and the platform's in-cluster configuration use only open standards, and data services open protocols; cloud-specific choices live only in the landing zone, the cluster IaC and <code>cluster-settings</code> (section 1.3)
R17	Per exposure, stateless clusters in the frontend network and one optional stateful cluster in the backend network, built only when a workload requires it (picture 3.1)
R18	Placement order stateless cluster → Azure PaaS → stateful cluster; the stateful cluster needs a documented exception (where does a workload run)
R19	Backend spoke with PaaS private endpoints; network isolated AKS (outbound type <code>none</code>), no public IPs, Firewall deny-all for backend prefixes except the two allow-listed destinations (flows)
R20	Fixed-size node pools per AZ 1/2/3 (no autoscaling), ZRS storage, Premium tier with Long Term Support
R21	Azure CNI (VNet, dynamic pod IP allocation) powered by Cilium; applications are published with internal <code>LoadBalancer</code> Services and terminate TLS themselves, TLS 1.3 only like the gateways (section 4.1)
R22	Azure CNI Pod Subnet powered by Cilium, every pod with its zone's address, Azure Virtual Network encryption between nodes, a Gateway API implementation per cluster with Let's Encrypt certificates distributed through Key Vault, HTTPS-only listeners offering HTTP/2 alone and TLS 1.3 alone, and h2c from every gateway to every pod; an application is reached from outside its namespace only through the gateway, also by the zone's other applications, and uses any protocol inside it (section 2.6 , section 4.1)
R23	<code>sl-az1</code> , <code>sl-az2</code> (+ optional <code>sl-az3</code>); external applications in every cell behind a zone-redundant traffic layer, internal applications in one home cell each, migrated to another cell on failure; policy-enforced replicas ≥ 2 for serving applications, non-serving workloads (jobs, scheduled jobs, queue consumers) with the copies their work needs in one home cell per environment (section 3.4), HPA/KEDA, VPA in-place right-sizing (section 2.3), the cluster autoscaler on every zone pool
R24	Readiness and liveness (+ startup) probes enforced by admission policy on every long-running workload, a finite job judged by its exit status (section 2.7); applications behind a gateway answer HTTPS <code>GET /healthz</code> with 200 only while they can serve, probed per cell by Traffic Manager (external zones) and Grafana Cloud synthetic monitoring, which leave out a cell whose probe fails (section 3.1 , section 6.2)
R25	2 exposures $\times$ (2 + 2 + 3) = 14 clusters (20 with the stateful clusters) from the same IaC modules, all in-cluster state from one Git repository: FluxCD in the stateless clusters, CI/CD pipeline in the stateful cluster (picture 3.2 , picture 3.3)

#	How the design meets it
R26	Policy-enforced rollout guardrails, releases one cell at a time with DNS affinity (external zones), zero-downtime migration of internal applications between cells, pipeline-driven stateful rollout, drain-and-upgrade per stateless cluster, PDB-guarded per-AZ upgrade of the stateful cluster (section 3.4 , section 3.5)
R27	Every cluster starts on 1.36, the platform floor. Stateful: created on an LTS minor and kept on it until 6 months before its LTS ends, nodes patched in place where possible. Stateless: <code>s1-az1</code> on the second latest minor with patches after 1 week per environment, <code>s1-az2</code> one minor behind with 3 weeks per environment; node images pinned per cell (update policy)
R28	Keys in the platform vault of the cluster's exposure: per cluster a KMS etcd key and an OS-disk key (AKS supports one disk encryption set per cluster, not per node pool); in the stateful cluster one key per isolation zone for its persistent volumes; encryption at host on every node (section 4.2)
R29	The hub Firewall allows a zone only the destinations of its per-application rule files, proposed by the egress agent from the denied flows and merged by a human (section 5.1); Azure Policy rejects an egress policy in an application namespace that would open anything out of the cluster, so the Firewall is the one decision (section 2.7)
R30	Agents with read-only identities per agent and environment open pull requests; a merge gate owned by humans computes a risk tier from the plan and requires checks and 0–2 human approvals; only the existing deployment identities write; runtime guardrails unchanged (section 6.1)
R31	One Grafana Cloud stack per environment; Grafana Alloy on every node collects node-locally and sends through Azure Private Link, also from the Internet-less stateful cluster; every user of a stack sees all of its data, so nothing the users may not see is sent; Azure-produced logs stay in Log Analytics and are read in place (section 6.2)
R33	The non-negotiable rules are the first page of the onboarding request and are applied by admission, CI and the agents, never by a person: the update policy's timetable, platform-owned base images with the base image agent rebuilding every application from <code>main</code> and the release agent promoting the digest at tier low on the automated gates, an image past its deadline refused at admission, exceptions with an expiry (section 1.5 , patching)
R34	A preview namespace per pull request and a sandbox namespace per team inside the dev cells, deployed from the pull request's branch outside the cell branches, sharing the application's identity, vault prefix, rule file and groups, under every policy of dev, removed with the pull request (section 1.5 , section 3.2)
R32	Every PaaS data store, the stateful cluster's persistent volumes and the secrets that cannot be issued again are backed up into the paired region, in locked immutable vaults and storage accounts of a backup subscription per environment; clusters, Kubernetes objects and images are rebuilt from IaC and Git; region recovery declared by humans and drilled in acc (section 6.3)

Exposure = internal (`int`) or external (`ext`): its own subscription and clusters per environment (R1). **Isolation zone** = one *type* = one combination of exposure × country, inside the clusters of its exposure:

Zone	Exposure (subscription, clusters)	Country	Node pools (stateless)	Node pools (stateful)	Key Vault (one per environment)	Subnets
<code>int-fi</code>	internal (<code>sub-aks-<env>-int</code> , <code>aks-<env>-int-*</code>)	FI	<code>intfi</code> , <code>intfispot</code> , <code>intfiarm</code> , <code>intfiarmspot</code>	<code>intfiz1-z3</code>	<code>kv-<env>-int-fi</code>	<code>snet-int-fi-*</code>

Zone	Exposure (subscription, clusters)	Country	Node pools (stateless)	Node pools (stateful)	Key Vault (one per environment)	Subnets
int-se	internal (sub-aks-<env>-int , aks-<env>-int-*)	SE	intse , intsespot , intsearm , intsearmspot	intsez1 – z3	kv-<env>-int-se	snet-int-se-*
ext-fi	external (sub-aks-<env>-ext , aks-<env>-ext-*)	FI	extfi , extfispot , extfiarm , extfiarmspot	extfiz1 – z3	kv-<env>-ext-fi	snet-ext-fi-*
ext-se	external (sub-aks-<env>-ext , aks-<env>-ext-*)	SE	extse , extsespot , extsearm , extsearmspot	extsez1 – z3	kv-<env>-ext-se	snet-ext-se-*

Adding a country adds two zones (int-xx to the internal clusters, ext-xx to the external ones) following the same pattern, online ([adding a country](#)).

Next to the zone vaults, every environment has exactly **one shared platform Key Vault per exposure**:

Key Vault	Exists	Holds	Access
kv-<env>-<zone>	Once per zone and environment, in the zone's exposure subscription; its private endpoint in the environment's snet-shared-pe	Application secrets of that zone, named <namespace>--<secret>	Read by the zone's External Secrets Operator with one reader identity per namespace, written by the namespace's edit / admin groups and pipelines - all under the namespace's prefix by RBAC + ABAC (section 2.5); nobody lists
kv-<env>-<exposure>-platform	Once per environment and exposure (kv-dev-int-platform , kv-dev-ext-platform , ...)	The gateway certificates of the exposure's zones (<zone>-gateway--wildcard , platform-gateway--wildcard), the platform namespaces' own secrets under their prefixes (flux-system-...), the renewal job's ACME account key, the image-signing certificate and the customer-managed keys of the exposure's clusters (key-<cluster>-kms , key-<cluster>-disk , key-sf-pv-<zone>)	Plain Azure RBAC: the gateway and platform namespaces read their own prefix like any namespace share; the keys and the signing certificate are assigned at single-object scope (section 4.1 , section 4.2)

Where another section says kv-<env>-platform , it means the platform vault of the exposure in question.

Colour legend in all pictures: grey = control plane / platform / PaaS, blue = internal, orange = external, purple = hub / management, green = identity, teal = stateless cluster (cell), pink = stateful cluster, yellow = notes / policy, red dashed = blocked.

Cells and bulkheads

The design uses two isolation patterns at two levels, and this document uses their names:

Term	In this design	Pattern	Limits the impact of
Exposure	Internal or external: a subscription per environment with its own cells, stateful cluster, VNet, vaults, ACR and policies	Hard boundary (subscription and cluster)	A compromise of an Internet-facing cluster or subscription: it cannot reach the internal workloads' control plane, nodes, identities or secrets
Cell	One stateless cluster of an exposure (<code>aks-<env>-<exposure>-sl-az<N></code>) with its own VNet, all of it in one availability zone: an external cell runs every external application, an internal cell the internal applications whose home it is	Cell-based architecture	An availability zone outage, a cluster failure, a bad release or a bad cluster upgrade: changes reach one cell at a time
Cell router	External zones: Azure Traffic Manager answers each client's DNS query with one cell, by the client's network (active-active). Internal zones: the hub Private DNS, kept by ExternalDNS, points each application's name to its home cell (incoming traffic)	Cell-based architecture	– (zone-redundant, outside the cells)
Isolation zone	One exposure × country (<code>int-fi</code> , <code>ext-se</code> , ...) with its own address space, subnets, nodes, Key Vault and policies, present in every cell of its exposure	Bulkhead	A noisy, failing or compromised workload of one zone: it cannot use another zone's nodes, network or secrets
Shared tier	The hubs (one per exposure with its own Firewall; gateways, DNS), Azure PaaS, ACR, Key Vaults, the optional stateful cluster	–	Not split into cells; zone-redundant instead, and changed with extra care

Two differences from the classic cell-based architecture:

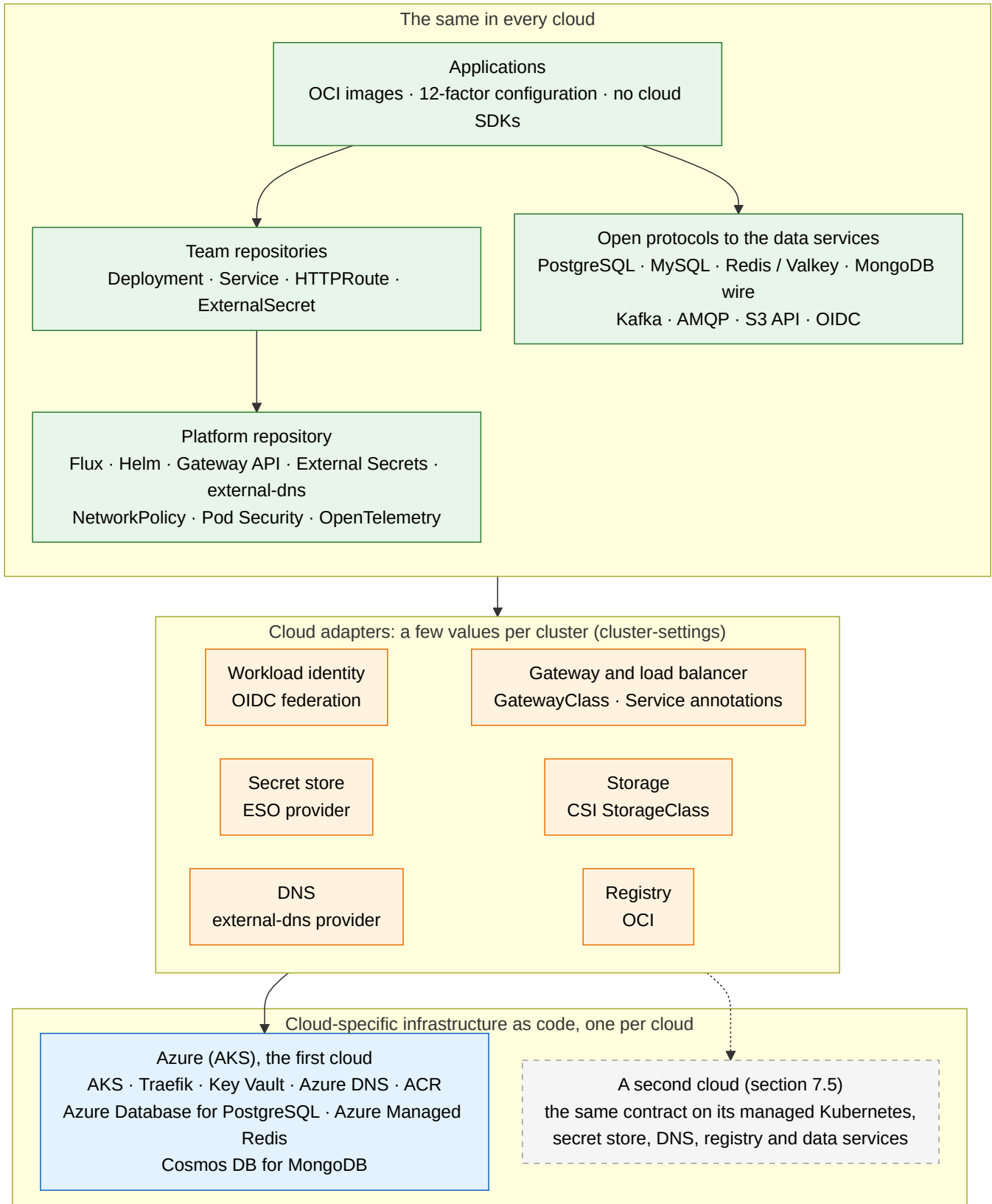
- **Every cell serves every external user.** For external zones the cells are full replicas, aligned with availability zones (as in the AWS guidance for EKS, one cluster per availability zone); users are not partitioned between cells by tenant or customer. The cell router only pins an external user's *session* to a cell ([DNS affinity](#)); internal applications run in one home cell each and are moved between cells by a migration workflow ([internal zones](#)).
- **The data is shared.** All cells use the same PaaS services and stateful cluster, so a cell isolates compute and releases, not data. Data changes therefore follow the [compatibility rule](#).

In this document “zone” means an isolation zone, unless it says availability zone (AZ) or zone-redundant. The canonical names are: a cell is `sl-az<N>` in prose, and its resources follow the cell's name `<env>-<exposure>-stateless-az<N>`: the cluster `aks-<cell>` (`aks-prd-int-stateless-az1`), its VNet `vnet-<cell>`, its NSGs `nsg-<cell>-<subnet>`, its route table `rt-<cell>`; the stateful cluster is `sf` in prose and `aks-<env>-<exposure>-stateful`; exposures are `int` and `ext`; zones are `int-fi`, `ext-se`, and so on. Key Vaults have globally unique names, `kv-<org>-<region>-<zone>` and `kv-<org>-<region>-platform`; where this document writes `kv-<env>-<zone>` it means the zone's vault of that environment. These are example names: an estate

names its resources by its landing zone's own convention, which is not published - a naming policy of a template per resource type over the location, the exposure and the environment, whose defaults are the names above, kept outside Git.

1.3 Open standards and portability

Requirements: [R15](#), [R16](#).



OneK8s is one Kubernetes platform, one application contract for every cloud – built on Azure (AKS) first. Applications run in any public cloud without changes to them or to how they are deployed (requirement [R16](#)). What is portable is the *contract*; the platform under it is an AKS platform that uses what Azure manages, because portability is weighed after security, automation, standard offerings, simplicity and cost. Azure (AKS) is the first cloud implemented.

The rule that makes it work: **applications, their manifests and the platform's in-cluster configuration speak only open standards**; cloud-specific choices live only in the picture's two lower layers, the values per cluster being handed to Flux by the IaC.

What is the same in every cloud

Layer	Open standard / project	In this platform
Container images	OCI image and distribution specs	Every app is an OCI image; every cloud registry serves the same image
Cluster API	Kubernetes, CNCF certified	Manifests use only GA Kubernetes APIs; no cloud CRDs in team repositories
Delivery	Flux (CNCF graduated), Kustomize, Helm	The same Git repositories and Kustomizations reconcile every cluster (section 3.3)
Ingress	Gateway API , the implementation chosen per cluster	Teams write <code>HTTPRoute</code> s; the Gateway, its listeners and TLS are the platform's (section 4.1)
Secrets	External Secrets Operator	Teams write <code>ExternalSecret</code> s against a <code>SecretStore</code> named <code>key-vault</code> ; only the store's provider is cloud-specific (section 2.5)
Identity	OIDC workload identity federation	Pods get cloud tokens through a Kubernetes service account – never stored keys
DNS	external-dns	Publishes listener hostnames in the cloud's DNS – one provider flag
Network policy	Kubernetes <code>NetworkPolicy</code> , Cilium	Deny-by-default between namespaces; egress to named destinations is the cloud's firewall
Workload security	Pod Security Standards, Gatekeeper constraint templates, <code>ValidatingAdmissionPolicy</code> (CEL)	The rules of section 2.7 are Gatekeeper templates and CEL policies; on AKS the Azure Policy add-on delivers them
Telemetry	OpenTelemetry , Prometheus	Apps emit OTLP; the collector's destination is configuration (section 6.2)
Data	PaaS services over open protocols: PostgreSQL, MySQL, Redis, MongoDB, Kafka, the S3 API	An application talks to a data service through its protocol, never a cloud SDK, and uses the common subset of its features (section 3.1, below)

The reference application of [section 0.3](#) shows it end to end: its manifests contain no Azure name at all.

What stays cloud-specific

Piece	Where	On Azure
Tenant, accounts / subscriptions, policy, hub network	The landing zone, out of this design's scope: Microsoft's Azure landing zones with the hub-and-spoke of section 1.1	Azure landing zone accelerator
Clusters and what an environment's clusters share: network, secret store, registry, DNS, global traffic manager	Cluster IaC	Terraform modules per environment and cluster type (section 3.2)
The few facts the cluster needs about its cloud: cluster name and exposure, tenant ID, secret store URLs and client IDs per namespace, DNS zones and resource groups, the gateway class per gateway, the registry and image-signing identity	<code>cluster-settings</code> of each cluster, filled in by the IaC	A ConfigMap per cluster in the platform repository (section 3.3)
Providers of a few platform components	Platform repository, per cloud overlay	External Secrets <code>azurekv</code> , <code>external-dns azure</code> , the Gateway API implementation (<code>approuting-istio</code> or <code>traefik-<gateway></code>)
Day-2 tools that read the cloud	The agents' tools (section 6.1)	Azure tools behind a cloud-neutral tool interface
How the admission policies reach the cluster	Cluster IaC (policy assignments)	Azure Policy add-on (Gatekeeper managed by AKS), assigned per environment subscription
The egress firewall and its per-application rule files	The hub	Azure Firewall Premium per exposure (section 5.1)
Human access to the clusters	Landing zone and cluster IaC	Entra ID groups, PIM, Azure RBAC for Kubernetes

Data in PaaS services that every cloud offers

Data belongs in managed PaaS services ([section 3.1](#)). To stay portable, an application **talks to a data service through its open protocol, never through a cloud SDK**, and gets its endpoint and credentials from a Kubernetes `Secret` (synced by External Secrets, or – preferably – an identity-based login with workload identity). The platform offers only services that **every major cloud runs as a managed service with the same wire protocol**:

Service	Protocol / API	Azure	AWS	Google Cloud
PostgreSQL	PostgreSQL wire protocol	Azure Database for PostgreSQL – Flexible Server	Amazon RDS for PostgreSQL, Aurora PostgreSQL	Cloud SQL for PostgreSQL, AlloyDB
MySQL	MySQL protocol	Azure Database for MySQL – Flexible Server	Amazon RDS for MySQL, Aurora MySQL	Cloud SQL for MySQL
Redis / Valkey	RESP (Redis protocol)	Azure Managed Redis	Amazon ElastiCache (Valkey, Redis OSS), MemoryDB	Memorystore for Valkey / Redis
MongoDB	MongoDB wire protocol	Azure Cosmos DB for MongoDB (vCore)	Amazon DocumentDB (MongoDB compatible)	Firestore with MongoDB compatibility
Kafka	Kafka protocol	Azure Event Hubs (Kafka endpoint)	Amazon MSK	Google Cloud Managed Service for Apache Kafka
Object storage	S3 API (on Azure through a gateway)	Azure Blob Storage	Amazon S3	Cloud Storage (S3-compatible XML API)
Secrets	External Secrets provider	Azure Key Vault	AWS Secrets Manager	Google Secret Manager
Container registry	OCI distribution	Azure Container Registry	Amazon ECR	Artifact Registry

Every cloud's offer is *compatible*, not identical: version support, extensions (for example PostgreSQL extensions), MongoDB feature coverage and Redis modules differ. So:

- An application uses **the common subset**: core PostgreSQL features and widely available extensions, standard Redis commands without modules, the MongoDB features DocumentDB, Cosmos DB and Firestore all support. A team that needs more records it as an exception in its onboarding, because that exception pins it to one cloud.
- **Vendor-neutral managed offers** that run in all three clouds – MongoDB Atlas, Confluent Cloud, Aiven, Redis Cloud – are acceptable alternatives where the cloud-native service falls short, ordered through the marketplace so billing and networking (Private Link / PrivateLink / Private Service Connect) stay in the cloud.
- Services **without** a cross-cloud equivalent with the same protocol (for example Cosmos DB's NoSQL API, DynamoDB, Spanner, Service Bus) are not offered as platform services; where the Azure sections name them, they describe the Azure implementation of a documented cloud-pinning exception.

1.4 The application contract

Requirements: [R8](#), [R9](#), [R11](#), [R13](#), [R14](#), [R15](#), [R16](#), [R18](#), [R21](#), [R22](#), [R23](#), [R24](#), [R26](#), [R29](#), [R31](#), [R33](#).

What an application team has to do, and nothing more, to run on the platform in any cloud. Everything here is enforced by admission policy, a CI check or the onboarding, and each line links to the section that explains it; the reference application of [section 0.3](#) meets all of it and contains no Azure name.

Manifests

Plain Kubernetes, in the team's own repository, one directory per application and one per cell for what differs there ([section 3.3](#)). Only GA Kubernetes APIs, the Gateway API, External Secrets and Cilium network policy objects; no cloud CRDs ([section 1.3](#)).

The application	Why	Enforced by
Runs as a <code>Deployment</code> with <code>replicas</code> or <code>HPA minReplicas</code> of at least 2 (or 0 when switched off), <code>topologySpreadConstraints</code> over nodes, <code>RollingUpdate</code> with <code>maxUnavailable: 0</code> , and its own <code>PodDisruptionBudget</code> with <code>maxUnavailable</code> of at least 1	A release or a node drain never takes it down (section 3.4)	Admission; the PDB by CI
Runs work that serves no request as a <code>Job</code> or <code>CronJob</code> (<code>backoffLimit</code> , <code>activeDeadlineSeconds</code> , <code>ttlSecondsAfterFinished</code> , <code>concurrencyPolicy: Forbid</code>) or, for a queue consumer, as a <code>Deployment</code> with at least 1 replica scaled on queue length and a graceful shutdown longer than its longest message; such a workload has no route and lives in the application's home cell in both exposures	Work that must not repeat runs once per environment; a cell drain finishes or restarts it (non-serving workloads)	Admission (the rollout rules apply to serving workloads only); the placement
Declares readiness and liveness (and startup) probes and resource requests; limits stay in Git, requests are resized in place by the platform's VPA unless the pod carries <code>platform/rightsizing: off</code>	HPA, the autoscaler and the N+1 sizing work from them (section 2.3)	Admission; the VPA is platform-maintained
Shuts down gracefully: a <code>preStop</code> delay and <code>terminationGracePeriodSeconds</code> longer than the gateway's drain	Connections finish during a rollout or drain (section 3.4)	Review
Uses an image from the environment's registry, by digest once promotion verifies signatures; builds arm64 or multi-architecture images when it chooses those nodes	No image reaches acc or prd unsigned (section 3.2 , section 2.7)	Admission (allowed images, Image Integrity); CI checks the image index against the labels
Builds its image from the platform's base image for its runtime with the platform's build workflow, from a <code>main</code> that always builds and passes its tests; accepts that the platform rebuilds and promotes it when the base is patched	Every application is patched on the platform's timetable, gated by the team's tests alone (section 1.5)	The build workflow; the registry refuses an image past its deadline

The application	Why	Enforced by
Meets the <i>restricted</i> pod security standard, with <code>readOnlyRootFilesystem: true</code> (writable paths as <code>emptyDir</code>), <code>automountServiceAccountToken: false</code> , no <code>hostNetwork</code> , no privileged containers, no <code>PersistentVolumeClaim</code> in a stateless cell	A compromised pod stays a pod (section 2.7)	Admission
Puts the application's <code>Deployment</code> , <code>Service</code> , <code>HTTPRoute</code> and <code>ExternalSecret</code> s in its namespace; writes no <code>Ingress</code> , <code>Gateway</code> , <code>LoadBalancer</code> , <code>NodePort</code> or egress policy out of the cluster	The zone's gateway is the platform's (section 4.1)	Admission

Traffic in and out

The application	Why	Enforced by
Publishes itself with an <code>HTTPRoute</code> on the zone's <code>Gateway</code> , with hostnames in the zone's domain (<code><name>.<zone>.<env>.<domain></code>); TLS, the certificate and DNS are the platform's	HTTPS only, HTTP/2 only and TLS 1.3 only at the edge (section 4.1)	Admission (Gateway rules)
Serves HTTP/2 cleartext (h2c) on its route port , and declares <code>appProtocol: kubernetes.io/h2c</code> on that <code>Service</code> port; keeps HTTP/1.1 for the kubelet's probes until Kubernetes 1.38	No HTTP/1.1 between a gateway and a pod (section 4.1)	Admission (<code>h2c-upstream</code>) and CI; the onboarding checks the image with <code>curl --http2-prior-knowledge</code>
Answers <code>GET /healthz</code> with 200 only while it can serve, behind the gateway with its host name	Traffic Manager and the synthetic probes leave out a cell whose probe fails (R24)	The probes
Expects no egress by default . A new external destination is a request: the egress agent proposes the Firewall rule from the denied flows and the team confirms it, or the team opens the pull request itself (section 5.1); the application writes no egress policy	Egress only to named destinations (R29)	The hub Firewall; admission (<code>app-egress</code>)
Listens from outside its namespace only through the zone's gateway, HTTPS on 443 to its route; between its own pods it uses any protocol and port	One way in for every client, also the zone's other applications (R22)	Cilium default deny (inbound only from the namespace and the zone's gateway); admission (no <code>LoadBalancer</code> , <code>NodePort</code> or <code>Ingress</code>)
Reaches other applications, of its zone or another, only through their published names with a rule in its own rule file; another zone's pods or nodes, a Key Vault, the instance metadata service and the stateful cluster of another zone never	Isolation (section 2.4 , section 2.6)	Cilium default deny, the Firewall, the NSGs

Identity, secrets and data

The application	Why	Enforced by
Authenticates to every Azure service with its workload identity (<code>id-<zone>-<app></code> , the ServiceAccount's <code>azure.workload.identity/client-id</code> annotation); its configuration holds endpoints and client IDs only	No keys, connection strings or SAS tokens; local auth is disabled on the services (section 2.5)	Azure Policy on the services; the onboarding creates the identity and roles
Takes a secret only where the target cannot use Entra ID, as an <code>ExternalSecret</code> against the namespace's <code>SecretStore</code> <code>key-vault</code> , naming a secret under its own prefix <code><namespace>-</code> ; mounts it as a volume; never calls Key Vault itself	Secrets reach a pod only through External Secrets Operator, separated per namespace (section 2.5)	Admission (ESO rules); Key Vault ABAC; no vault role on the application's identity
Keeps its data in a PaaS service over its open protocol (PostgreSQL, MySQL, Redis, MongoDB, Kafka, S3-compatible storage) and the common subset of its features; a cloud-specific service or extension is a documented exception at onboarding	Portability of the contract (section 1.3)	Onboarding review
Changes its data and APIs backwards compatibly - <i>expand</i> → <i>migrate</i> → <i>contract</i> over separate releases	Two versions run at once during a release (section 3.4)	Review; the release train's SLO gates
Asks for the stateful cluster only with a documented exception (why neither a stateless cell nor a PaaS service fits, owner, exit plan); there it terminates TLS 1.3 itself on an internal load balancer	The stateful cluster is the last option (section 3.1)	The exception label on the namespace; admission

Choices the application makes with labels

Pod label	Values	Effect
<code>platform/arch</code>	<code>amd64</code> (default), <code>arm64</code> , <code>multi</code>	Which nodes; <code>multi</code> lets the platform pick the cheapest (section 2.3)
<code>platform/capacity</code>	<code>on-demand</code> (default), <code>spot</code>	Spot nodes when available, on-demand otherwise; only for workloads that tolerate losing replicas with 30 seconds' notice
<code>platform/rightsizing</code>	<code>off</code>	The VPA only recommends; the baseline in Git still moves by pull request

The platform turns the labels into node selection; an application that sets selectors, tolerations or affinities for nodes itself is refused.

Observability

The application	Why	Enforced by
Logs JSON to stdout; emits OpenTelemetry (OTLP) to the endpoint the platform injects (<code>OTEL_EXPORTER_OTLP_ENDPOINT</code>) and propagates W3C <code>traceparent</code> ; logs carry <code>trace_id</code>	One Grafana Cloud stack per environment, node-local collection (section 6.2)	The injected variables; Beyla instruments without code
Logs no secret, token or personal data that the environment's users may not see	Every user of a stack sees all of its data (section 6.2)	Alloy's redaction is a safety net only
Keeps its SLO objectives, dashboards and alerts in <code>apps/<app>/observability/</code> next to its manifests	Default SLOs come from the gateway; the team sets the targets	The IaC pipeline applies them

Onboarding

An application starts with the request template (section 6.1): the zone, the team's Entra ID groups for `view` , `edit` and `admin` , the owner labels `platform/team` and `platform/cost-center` and the contact annotation (section 2.6), the Azure services it needs and whether it needs a secret, and, for an internal application, its home cell (section 3.1). The onboarding agent turns it into the namespace, the identities, the roles and the Flux configuration; the team never touches cluster-scoped objects. The journey around the contract - onboarding, migrating an existing application and operating it in production - is section 1.5. Cost is reported per team from the labels, and a budget alerts the owners.

1.5 Developer experience: onboarding, migration and operations

Requirements: [R8](#), [R9](#), [R11](#), [R13](#), [R14](#), [R16](#), [R18](#), [R23](#), [R24](#), [R25](#), [R26](#), [R29](#), [R30](#), [R31](#), [R33](#), [R34](#), [R22](#).

The platform as an application team meets it: three journeys - **onboarding** a new application, **migrating** one that runs elsewhere, and **operating** the ones in production - told as use cases. The [application contract](#) says *what* a team has to deliver; this page says what happens around it, from the first request to the day the application is retired. It names no tool: the sections it links to do.

Four rules shape every journey:

1. **The team owns its application and its repository; the platform owns everything cluster-scoped.** A team writes plain Kubernetes for its namespace and nothing else ([section 1.4](#)); the namespace, identities, roles, routes' gateway, DNS, certificates and firewall rules are made for it.
2. **Every interaction is a pull request or a request.** A change to the application is a pull request in the team's repository; a change the platform must make for the team (a namespace, an identity, a firewall rule, a secret share) is a request that an agent turns into the platform's pull request, approved by its risk tier ([section 6.1](#)). Nobody, human or agent, changes a cluster by hand.
3. **The platform answers before the team asks.** Admission policy and the CI checks tell a team what the contract misses before anything runs; denied flows become proposed rules; the right-sizing, release and operations agents open the pull requests and issues a team used to write itself.
4. **The same experience in every environment and every cloud.** dev, acc and prd differ only in who approves and how fast ([section 3.2](#), [R25](#)); a second cloud is the same contract under another implementation ([section 1.3](#), [R16](#)).

Running it with a small platform team: the non-negotiable rules

The platform is run by a few people for many applications, with the agents of [section 6.1](#) doing the routine. It stays small only if the platform team does **no work per application and negotiates nothing per application**: every rule is a policy, a timetable or a limit that applies to every application alike and is applied by a check, an agent or admission, never by a person. The rules below are the ones a team accepts at onboarding ([R33](#)); a team that cannot meet one is not onboarded, or carries an exception with an owner and an expiry date ([exemptions expire](#)). The platform's time goes into the rules and the exceptions, not into the applications.

Rule	What the team accepts	If it is not met
The platform's patch timetable applies to every application. Cells are upgraded on the update policy 's timetable, and every application image is rebuilt and deployed when its base image is patched (patching)	No application blocks a cell upgrade or postpones a patch; the team's answer to both is the contract's rollout rules and its automated tests	A removed API or a failed rebuild is an issue to the team with a date; after it the upgrade goes ahead, and an image past its deadline is refused at admission, so the application cannot scale or restart until the team fixes it
Automated tests are the only gate. The contract check, the team's own tests in CI and the smoke tests on the per-cell names decide what moves; no release waits for a manual acceptance	The team keeps tests that prove its routes work, and <code>main</code> always releasable, because the platform rebuilds and promotes from it without asking	An application without tests that pass is not onboarded; a rebuild whose tests fail stops at dev and is the team's issue
One release train per environment , with its change windows and freezes (section 6.1)	No per-team release calendar; an urgent fix takes the train with a shorter soak	-
The contract's rollout rules and backwards compatibility (section 1.4 , section 3.4)	Two replicas, probes, a disruption budget, graceful shutdown, <i>expand</i> → <i>migrate</i> → <i>contract</i>	Refused at admission; a breaking change is rolled back by the SLO gates
No egress, no secrets, no cluster-scoped objects by default (section 1.4)	Every destination, secret and identity role is a request or a confirmed proposal	Denied by the Firewall or admission, logged
Capacity inside the limits (<code>limits.yaml</code> , section 6.1)	Requests and bounds move by pull request within the limits; a limit change is a request to the platform, not a purchase	Over a soft limit one human approves, over a hard limit the pull request is rejected
Every application has an owner who answers (section 2.6)	The contact and the groups are kept current; the team is on call for its application	A namespace whose contact does not answer and whose groups are empty is retired after a notice
Exceptions expire. The stateful cluster, a cloud-pinned service, a policy exemption	An exception has an owner, a reason and a date, at most six months; renewal is a new request	The exception lapses and its rule applies

The rules are the first page of the onboarding request: the team accepts them before anything is created for it.

What a team has, and what it never has

The team has	The team never has
One repository with one directory per application and one per cell (section 3.3)	Write access to the platform repository or the IaC

The team has	The team never has
One namespace per application and zone, with its quota, default-deny policies and the roles of its own three groups (section 2.6)	A cluster role, a second namespace's objects, the list of other tenants
One workload identity per application and zone, with the roles its onboarding named (section 2.5)	A key, connection string or certificate it keeps itself
Its own prefix in the zone's vault, written by its <code>edit</code> and <code>admin</code> members (section 2.5)	A pod that reads the vault; the vault's list of names
One DNS name per environment in the zone's domain, published from its route (section 3.1)	A listener, a certificate, a load balancer or a public IP of its own
A rule file of its named egress destinations, proposed from what the Firewall denied (section 5.1)	An egress policy of its own, or a destination by address
Its SLOs, dashboards and alerts next to its manifests, and every log, metric and trace of the environment (section 6.2)	Telemetry of another environment; a secret in the telemetry
Its share of the cell's cost by team and cost centre (section 2.6)	A bill for the platform's fixed part
Read access to its namespace in every environment, a kubeconfig for that namespace alone	<code>kubectl apply</code> in acc or prd; break-glass, which is the platform's

Onboarding a new application

The journey from an idea to the first request served in production. Each step is a use case; the right column says who or what does the work.

Step	What the team does	What the platform does
1. Place it	Walks down the placement list (where does a workload run): a stateless cell for the code, PaaS services for the data, the stateful cluster only with a documented exception (R18). Picks the zone (its country, its exposure) and, for an internal application, a home cell - the slow-speed cell by default	Refuses what the list does not allow: a <code>PersistentVolumeClaim</code> in a cell, a cloud-pinned service without an exception, a zone that does not match the exposure
2. Request it	Accepts the non-negotiable rules and fills the request template (onboarding agent): application and zone, the three Entra ID groups, team, cost centre and contact, the Azure services it needs and whether any of them needs a secret, its data stores and who owns their restore (section 6.3), arm64 or spot if wanted	The onboarding agent turns the request into pull requests: namespace with its labels and quota, workload identity with its roles, vault prefix and secret store, the Flux configuration of the team's repository, the budget line, the backup posture. A new application is at least a medium tier, so one human approves; the team gets back the names it will use

Step	What the team does	What the platform does
3. Start from the reference	Copies the reference application of section 0.3 , builds its own image from the platform's base image for its runtime with the platform's build workflow, and adds the tests that will be its only gate. Keeps what the contract asks for: two replicas, probes, a <code>PodDisruptionBudget</code> , graceful shutdown, h2c on the route port, the placement labels (section 1.4)	CI runs the contract check on every pull request: manifests against the same rules as admission, the image for h2c and its architectures, the egress names against the zone's allow-list (section 3.3). A team learns what is missing before anything is deployed
4. First run in dev	Merges. Checks the first-run list: the route answers on its name over TLS, <code>/healthz</code> is green behind the gateway, logs and traces appear in the environment's dashboards with the trace id	Flux deploys within minutes; the mutations place the pods on the zone's nodes of the chosen architecture and capacity (section 2.3); ExternalDNS publishes the name; the gateway probes <code>/healthz</code>
5. Open the egress it needs	Expects nothing to be reachable. Reads the egress agent's proposal - each denied destination with the pod that asked for it and the evidence - and confirms the ones the application needs, or opens the pull request itself	The Firewall denies and logs; the egress agent proposes one rule file change per namespace, a human merges it (tier high); the traffic flow pages show the application's picture (section 5.2)
6. Connect its data	Reaches its PaaS services by identity over their open protocols; takes a secret only where the target cannot use Entra ID, as an <code>ExternalSecret</code> under its prefix, written to the vault by an <code>admin</code> member (section 2.5)	The onboarding created the services' private endpoints, the identity's roles and the Firewall rules for them; the policies refuse a pod that reads the vault itself (R11)
7. Promote	Reviews the promotion pull request to acc, then to prd: the digests that ran in dev, the release notes and the risk summary the release agent wrote (section 3.3). Sets its SLO targets and alert routes in Git	Signatures verified and images imported at promotion; the release train carries the change through the cells one at a time behind SLO gates (section 3.4); the first application in a prd namespace is a medium tier

The onboarding is done when the application serves its name in prd, its SLOs are measured and its owner is on the cost report. The team never created a cluster-scoped object, a DNS record, a certificate, a role assignment or a Firewall rule - it asked for them, by template or by confirming a proposal.

Adding a team member is a change to the team's Entra ID groups, not to the platform; **adding a second application** of the same team repeats steps 1 to 7 with a new namespace in the same repository.

Trying it before dev: previews and sandboxes

A team tests a change before it reaches dev's `main`, and tries the platform before its first onboarding, in dev itself ([R34](#)) - never in a separate sandbox cluster, which would be a fourth environment that drifts and a place where the contract is not enforced. Both kinds of namespace meet every policy of dev, so what works in them works in prd.

	Preview	Sandbox
What	A namespace <code><app>-pr-<n></code> next to the application's dev namespace, in the same zone, for one pull request	A namespace <code><team>-sandbox</code> in a zone of dev, long-lived, with the reference application preloaded
Who asks	The team, by marking its pull request for preview; nothing else to request	The onboarding request, also before the team's first application
Deployed from	The pull request's branch, read by the dev cells outside the cell branches and the release train, like the placement (section 3.3)	The branch the team names, the same way
Shares	The application's identity, vault prefix, Firewall rule file and groups in dev, so a preview reaches exactly what the application reaches and nothing more; the application's dev data stores, so a schema change in a preview follows <i>expand</i> → <i>migrate</i> → <i>contract</i> like a release	Its own identity without roles, its own vault prefix and rule file - a sandbox is an application of its own, and a destination or a service it needs is a request like any other
Published as	<code><app>-pr-<n>.<zone>.dev.<domain></code> on the zone's gateway, published only in dev	<code><team>-sandbox.<zone>.dev.<domain></code>
Observability and cost	Labelled as a preview: left out of the application's SLOs and alerts, in its logs and traces, charged to the team	Charged to the team like an application
Lifetime	Removed when the pull request closes, or after seven days at most; a team runs at most a few at once (its quota)	Until the team asks for its retirement, or when its groups are empty and its contact does not answer
Needs a human	Never: no identity, rule or role is created for a preview	The onboarding's tier, once

A preview is not a release: it never reaches acc or prd, is never promoted, and carries no `main` commit. It gives a team the real gateway, policies, egress control and telemetry for a branch, which is what a local environment cannot.

Migrating an existing application

An application that runs today on a virtual machine, another Kubernetes cluster or another cloud meets the contract for the first time. The journey adds an assessment and a cut-over to the onboarding; the onboarding itself does not change.

1. Assess against the contract

The team lists what the application does today, and every finding falls into one of four classes. The classes decide the effort and the approvals, not the tools:

Class	Meaning	Examples
Fits as is	The contract is met	Stateless HTTP service with probes, configuration from the environment, data in a PostgreSQL or MySQL service; a scheduled job or a queue consumer (non-serving workloads)
Change the application	A code or manifest change the team makes before onboarding	Serves HTTP/1.1 only → serves h2c on the route port (section 4.1); an Ingress → an HTTPRoute; a key or connection string → workload identity (R9); reads a vault with an SDK → an ExternalSecret (R11); session state in memory → in a cache service, so any copy may answer; nodeSelector s and tolerations → the placement labels (R8); a cloud SDK for data → the service's open protocol (section 1.3); logs to files → JSON to stdout; a single copy of a serving application → two, with a leader election where only one may work (R23); a scheduled job that may not run twice → a CronJob in the home cell
Change where it runs	The architecture does not fit a stateless cell	Data on local disks or a PersistentVolumeClaim → a PaaS service, or the stateful cluster with an approved exception (R18); a product shipped only as an operator → the stateful cluster; a cloud-specific service → an exception that pins the application to one cloud (section 1.3)
Does not fit	The platform does not offer it; the application stays where it is or is redesigned	A listener for other namespaces that is not HTTPS on 443 through the gateway (a database protocol served to other applications, a message broker's port; inside its own namespace an application uses any protocol, R22); a public entry point for an internal workload; a flow into another zone's pods; a privileged container or host network (section 2.7)

The assessment ends as a list of findings with their class, kept with the application, and the same CI check that onboarding uses turns the second class into failing checks the team works down. Every “change where it runs” finding and every exception goes through the onboarding request, where a human approves it.

2. Prepare the data

Data moves first and separately from the code, because two versions run during any cut-over ([compatibility rule](#)): the target PaaS service is created by the onboarding, the data is copied or replicated into it with the service's own migration tooling while the old application keeps running, and the application is made to work against the new store before it moves. The schema follows *expand* → *migrate* → *contract*; a store that cannot be replicated is moved in a planned window, which is the one downtime the journey allows and the team's call. Where the data stays in the stateful cluster, its exception, backup posture and restore owner are part of the request ([section 6.3](#)).

3. Onboard, then cut over

The application is onboarded like a new one (steps 1 to 7 above), in dev first with test data, then in acc and prd next to the running old one, under its platform name. The cut-over moves the clients, and follows the same shape as the [migration between cells](#):

1. **Deploy** the application in prd on the platform; nothing points to it yet.
2. **Test** it on its platform name over the real TLS path, with the migrated data.
3. **Switch** the clients: the old name becomes an alias of the platform name, or the clients are reconfigured, in the order the team chooses (a share of the clients first where the clients allow it).

4. **Drain** the old deployment: wait until it sees no requests, keep it ready to take the name back.
5. **Remove** the old deployment and its data after the soak; it was the rollback until then.

Rollback at any step before the last is moving the name back; the data store is already the new one, so the old application must be able to read it, which is what step 2 of the preparation made sure of.

4. Moves inside the platform

Once onboarded, an application moves without a second migration project:

Move	How
To another cell (internal zones)	A pull request to the team's placement; the migration workflow does the rest without downtime (section 3.1)
To another zone	A new onboarding in the target zone and a retirement in the source: the zone is the namespace, the identity, the vault prefix and the country of the data, none of which moves
From internal to external exposure, or back	The same, in the other exposure's clusters and subscription (section 3.1); an internal application that an external one must call is published like any other public name
To another cloud	The same repository and manifests against another implementation of the contract (section 1.3); the data moves with the service's own tooling, the exceptions list says what does not

Operating applications in production

What a team does with an application once it is in prd, and what the platform does for it. The platform's own operations - upgrades, capacity, policy, incidents of the platform - are the agents' ([section 6.1](#)); here is what the team sees of them.

Releasing a change

A merged pull request in the team's repository is a release candidate; the promotion pull requests carry it to acc and prd, and the release train moves it through the cells one at a time behind SLO gates ([releases cell by cell](#)). The team sees the train on its pull request: which cell runs the new version, the test results on the per-cell names, the soak, and a rollback with its reason when a gate fails. Consequences the team plans for:

- **A change rides a train** with other teams' changes; a release is not instantaneous. An urgent fix takes the same path with a shorter soak, and a team that needs to stop a rollout asks for the halt - a safe action that needs no pull request ([section 6.1](#)).
- **Two versions run at once** during the train, so every change is backwards compatible with the previous one; the compatibility rule is the team's, not the platform's to check.
- **Switching an application off** is `replicas: 0` in Git; it is switched on the same way. Nothing in between is allowed ([R23](#)).

Patching: base images and dependencies

Patching is the one change that must reach every application without anyone asking every team, so the platform drives it and the team's tests gate it:

- **Base images are the platform's.** For every supported runtime and version (a .NET LTS, a Java LTS, Node LTS, Python, a static base for Go and the like) the platform keeps one base image in the environment's registry, built from the vendor's image, scanned and signed. Every application image is built from one of them by the platform's build workflow in the team's repository, and records its base, so the registry knows every image per base.
- **A vendor patch rebuilds the fleet.** When the vendor publishes a patch, the platform rebuilds the base, then the base image agent rebuilds every application on it from the current commit of its `main`: the same source, a new base layer, the team's tests and the contract check in CI, a signed image. Each rebuild is a pull request in the team's repository that sets the new digest for dev and merges itself when the tests pass (tier low); the release agent's promotion pull requests carry the digest to acc and prd the same way (tier low: a digest that passed every gate in acc). The patch train runs on the automated gates alone - the tests, the smoke tests on the per-cell names, the SLO gates and a shorter soak - inside the change windows; a breach rolls the cell back like any release ([section 3.4](#)).
- **The timetable** is the update policy's: a routine patch rides the next train in each environment; a fix for an actively exploited vulnerability takes the emergency fast track of [section 3.5](#) - the waiting times skipped with the platform owner's approval, still dev → acc → prd and one cell at a time. A runtime version that reaches its end of life is retired on a date; the team moves to the next supported base before it, under the same deadline rule.
- **Dependencies inside the image** (packages the application brings) are the team's to update; the registry's scan reports them with the same deadlines, and a critical one past its deadline is treated like a stale base.
- **What the team sees:** a pull request it did not write that merges itself, an issue with a date when a rebuild or its tests fail, and the share of its images on the newest base in the compliance report. It does nothing else.
- **What the team owes:** an image built from the platform's base with the platform's workflow, a `main` that always builds and passes its tests, and tests that prove its routes - because nothing else stands between a rebuilt image and prd.

Knowing that it works

The team's SLOs, dashboards and alerts live next to its manifests and are applied with them ([section 1.4](#)). The default SLOs - availability and latency at the zone's gateway - exist from the first run; the team sets the targets and adds what the gateway cannot see (queue lag, business events). Alerts are routed by the namespace's team label ([section 2.6](#)), and every member of the environment's Grafana stack sees everything in it ([R31](#)), so a team debugs a call across applications with one trace id, and keeps secrets and personal data out of what it logs ([what must not be sent](#)). The same place shows the application's cost per month by team and cost centre, and its requests against its use.

Being on call

An alert on the application reaches the team; an alert on the platform reaches the platform. The line between the two is the contract: the gateway, nodes, network, DNS, certificates, Firewall, the vault and the cluster are the

platform's, the pods and their behaviour are the team's. The operations agent triages every alert first, correlates it with releases, platform changes and cell health, and files an issue with the evidence - to the team when the cause is in its application, to the platform otherwise ([section 6.1](#)), so a team on call starts from a diagnosis, not from a dashboard. In prd a team reads its namespace and changes it only through Git; there is no shell into a pod of prd, and break-glass is the platform's, followed by a pull request that brings Git back in line.

Changing the application's resources, access and reach

Every day-2 change of an application is a pull request, and most of them are proposed before the team asks:

The team wants to	The team does	Tier
Scale on load	Sets HPA bounds in Git; the VPA resizes requests in place, and the right-sizing agent proposes the baseline (pod right-sizing)	Low
Use spot or arm64 nodes	Changes the pod labels (R8)	Low
Reach a new destination	Confirms the egress agent's proposal, or opens the rule file pull request (section 5.1)	High (a Firewall rule)
Add or rotate a secret	Writes it to its vault prefix as an <code>admin</code> member; the <code>ExternalSecret</code> follows (section 2.5)	None: no pull request to the platform
Give a person access	Changes its Entra ID groups (section 2.6)	None
Use a new Azure service	A request: the identity's roles, the private endpoint, the Firewall rule	Medium or high
Call another application of its zone	Its published name in the caller's rule file, like any destination (section 2.6)	High (a Firewall rule)
Publish a second name	A second <code>HTTPRoute</code> in the zone's domain; DNS follows (section 1.4)	Low
Move to another cell	A pull request to its placement (section 3.1)	Low

Living with the platform's changes

The platform changes under the application without asking it, and the team notices only what concerns it:

- **Cluster upgrades, node images and base images** are invisible: a cell is drained, upgraded and returned; an internal application is migrated out of the cell and back ([section 3.5](#)). What the team owes is the contract's rollout rules, which make the drain harmless.
- **A removed Kubernetes API or a new policy** reaches the team as an issue from the upgrade or compliance agent with the manifests concerned and a date ([section 6.1](#)); a policy starts in audit, so the team sees its finding before it is denied ([section 2.7](#)).
- **A cell failure** fails the internal applications of that cell over to another cell and leaves a drained external cell out of the answers; the team is told, and does nothing ([section 3.1](#)). **A region loss** is the platform's recovery; the team's part is the restore of its own data stores, named at onboarding ([section 6.3](#)).

- **Cost** is reported, not negotiated: the capacity and cost agent files an issue when an application's requests are far from its use or its spot share could grow; the team answers with a pull request or a reason.

Retiring an application

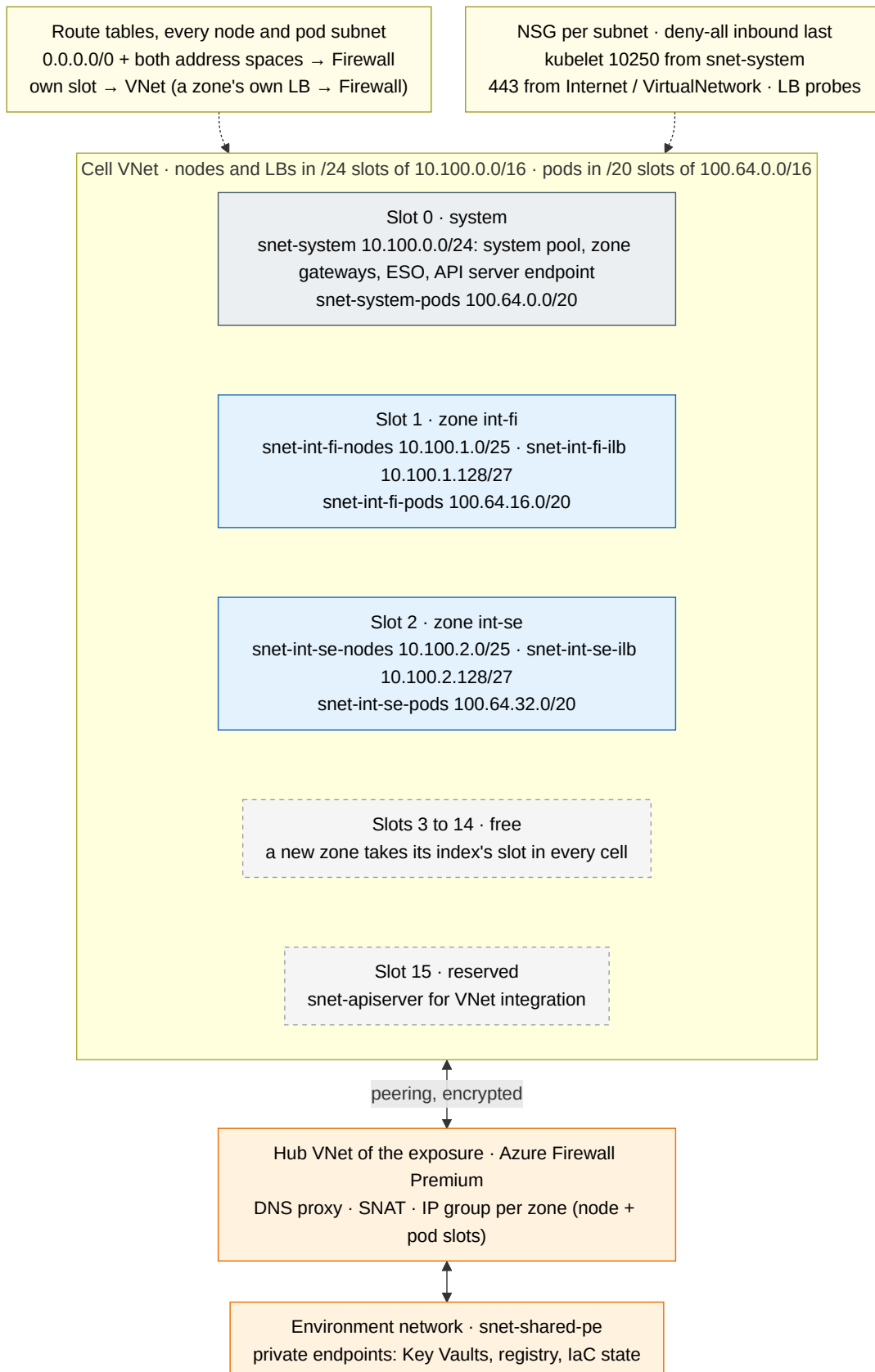
The last use case is the first in reverse: the team removes the application's directory and its cell directories, and asks for the retirement. The request removes, in order, the DNS name, the Flux configuration, the namespace and its roles, the Firewall rules and the traffic flow page, the identity and its roles, the vault prefix (recovered for the vault's retention, then purged) and the budget line; the backups of its data stores stay for their retention and are the last thing to go. The cost report shows the team's share going to zero.

The same journey in every environment

	dev	acc	prd
Previews and sandboxes	Yes, in the dev cells (previews and sandboxes)	No	No
Who deploys	Flux from the team's merge, within minutes	The promotion pull request, merged by tier	The promotion pull request, merged by tier, inside a change window
Who approves a new application	The tier: one human	One human	One human; a Firewall rule or an identity change two
Human access to the namespace	Read, and the roles of the team's groups	Read	Read
The data	The team's test data	The team's test data	Production
What differs	Nothing in the contract: smaller pools, a Basic registry on its public endpoint (section 3.2)	Nothing	Three cells instead of two

2.1 Network layout

Requirements: [R2](#), [R3](#), [R4](#), [R5](#), [R6](#), [R22](#), [R29](#).



Every cell has its own VNet; the picture shows its slots, subnets and routes for an internal cell. A zone's **index**, and so its slot, is the same in every cell of the exposure, so a new zone takes a free slot in every cell and nothing existing is re-addressed. The pods' address space is a /16 of 100.64.0.0/10 : about 4,000 pods per zone, handed to the nodes in batches of sixteen (an example is in the [address plan](#)). Private endpoints are not in the cells at all: they are reached through the hub firewall.

	Stateless cell	Stateful cluster (designed)
CNI	Azure CNI Pod Subnet with dynamic IP allocation, powered by Cilium; classic node pools with the cluster autoscaler (section 2.3)	The same CNI
Pod IPs	From the zone's <code>snet-<zone>-pods</code> - routable in the VNet, kept up to the hub firewall (AKS's masquerading agent translates nothing); the firewall SNATs them with the other private ranges	From <code>snet-<zone>-pods</code> - routable in the VNet
Which application may leave the cluster	The hub Firewall decides per zone from per-application rule files; its log names the pod (section 5.1)	The same Firewall, between the backend spoke and the PaaS private endpoints (flows)
<code>snet-<zone>-ilb</code>	Internal load balancer of the zone's gateway (internal zones; an external zone's gateway has a public load balancer, section 3.1)	Internal load balancers of the applications' LoadBalancer Services
Traffic to another zone of the same cell	Everything meets the hub Firewall: pods, nodes and load balancers are in the zone's slots, and the zone's route table sends the cell's address spaces to the Firewall. Never a direct connection: an application calls another zone's application through that zone's published name, with an application rule in its own rule file	Same

Zone isolation rests on every zone having its own nodes in its own subnets: a packet from a zone that leaves the cluster always carries an address of that zone's slots. Every subnet has its own **NSG** (`nsg-<cell>-<subnet>`) and every node and pod subnet a **route table** (`rt-<cell>`, `rt-<cell>-<zone>`) with BGP propagation off, **of the same shape for the system pool and for every zone** - five routes, the Firewall being the exposure's hub Firewall. The default route alone would not do: a packet to another slot of the cell matches the VNet's own system route, which is more specific than `0.0.0.0/0`, and would be delivered inside the VNet without any firewall seeing it. A user-defined route for the whole address space takes precedence over the system route for the same prefix, so what one slot sends to another leaves for the Firewall, which decides by rule and logs it: nothing is allowed between zones ([R4, section 2.4](#)), and between a zone and the system pool only the Kubernetes flows ① to ④ of section 2.4 - the kubelet, node metrics, Cilium's health and the gateways from the system pool to a zone, cluster DNS from a zone to the system pool - as network rules of the exposure's Firewall Policy. The own-slot exceptions are needed because Azure applies a route to the traffic within a subnet as well: the nodes and pods of one pool talk to each other directly. Two details keep the paths symmetric, which a stateful firewall needs. The API server's private endpoint stays outside the Firewall's view: a private endpoint ignores its subnet's route table, and every subnet of the VNet carries a /32 system route to it that wins over the user-defined routes, so a zone's nodes reach the API server directly and it answers them directly (making the endpoint honour the system subnet's table was tried and withdrawn: its answers to the system pool's own nodes and pods then left for the Firewall while their requests had stayed inside the subnet, and the system node's kubelet lost the API server). And a zone's own load balancer is reached through the Firewall because a request delivered inside the VNet would be handed to the

gateway by the load balancer while the gateway's answer, from the system pool, left for the Firewall. A new zone changes no table. So inside a cell three layers separate the zones, and the zones from the platform, in the order a packet meets them: the Firewall (L3-L7, logged), the destination subnet's NSG and Cilium - and the Firewall is also the layer for everything that leaves the cell.

An application that must call an application of another zone - or its own zone's published name - is a client of that zone's entry point like any other, never a neighbour: its rule file names the published name (`<app>.
<zone>.<env>.example.com` , resolved by the hub's Private DNS to the zone's gateway load balancer), exactly as it names an API on the Internet ([section 5.1](#)). The Firewall's application rule proxies the flow from its own address, so the gateway - whose pods run on the system pool - answers through the hub by the system subnets' default route, the path is symmetric, and the Firewall's log names the calling pod. A network rule between two zones' slots is never written: it would bypass the entry point. Applications of one zone call each other through their Kubernetes Services, inside the cluster, where Cilium decides ([section 2.6](#)).

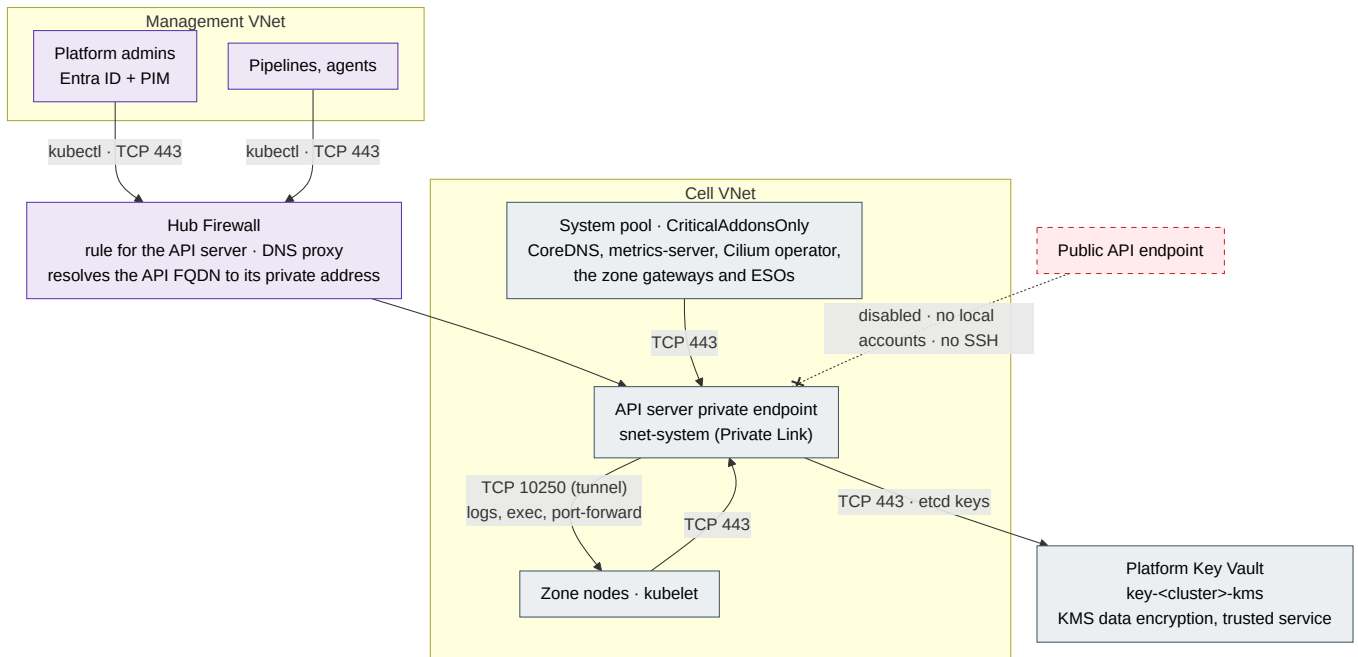
What the NSGs allow on a node subnet, and from where:

- the kubelet port (TCP 10250) from the cell's system subnet (the AKS tunnel); Cilium health (TCP 4240), ICMP and DNS (53) between the cell's node subnets; the cell's pod subnets between the node subnets; load balancer probes from `AzureLoadBalancer` ; ICMP from the hub firewall's address; traffic within the subnet. A pod subnet gets the rules of its pool's node subnet;
- TCP 443 from the **Internet** on the system subnet (every zone's gateway pods run there) and on an external zone's node subnet; TCP 443 from `VirtualNetwork` on an internal zone's node and load balancer subnets and on the system subnet - an NSG rule takes one service tag, and the corporate ranges arrive through the hub as `VirtualNetwork` .

Why subnets and not separate VNets? AKS requires all node pool subnets and the API server subnet of one cluster to be in the same VNet. Within one cluster, "network per country" is therefore implemented as *a subnet slot per zone + NSG + Cilium, with the hub firewall between the slots and for everything that leaves the cell*, which the security team accepts for countries. Internal and external are not separated this way: they are separate clusters in separate subscriptions behind separate hubs, so the picture shows an internal cell; an external cell's VNet (`vnet-prd-ext-stateless-az1`) is the same with `ext-fi` and `ext-se` in its slots. Should a hard VNet boundary per country ever be required, the same step is taken once more – **one cluster per zone** – and the address plan stays the same.

2.2 Control plane

Requirements: [R7](#).



The API server is **private**, over **Private Link**. Its public FQDN resolves to the private address

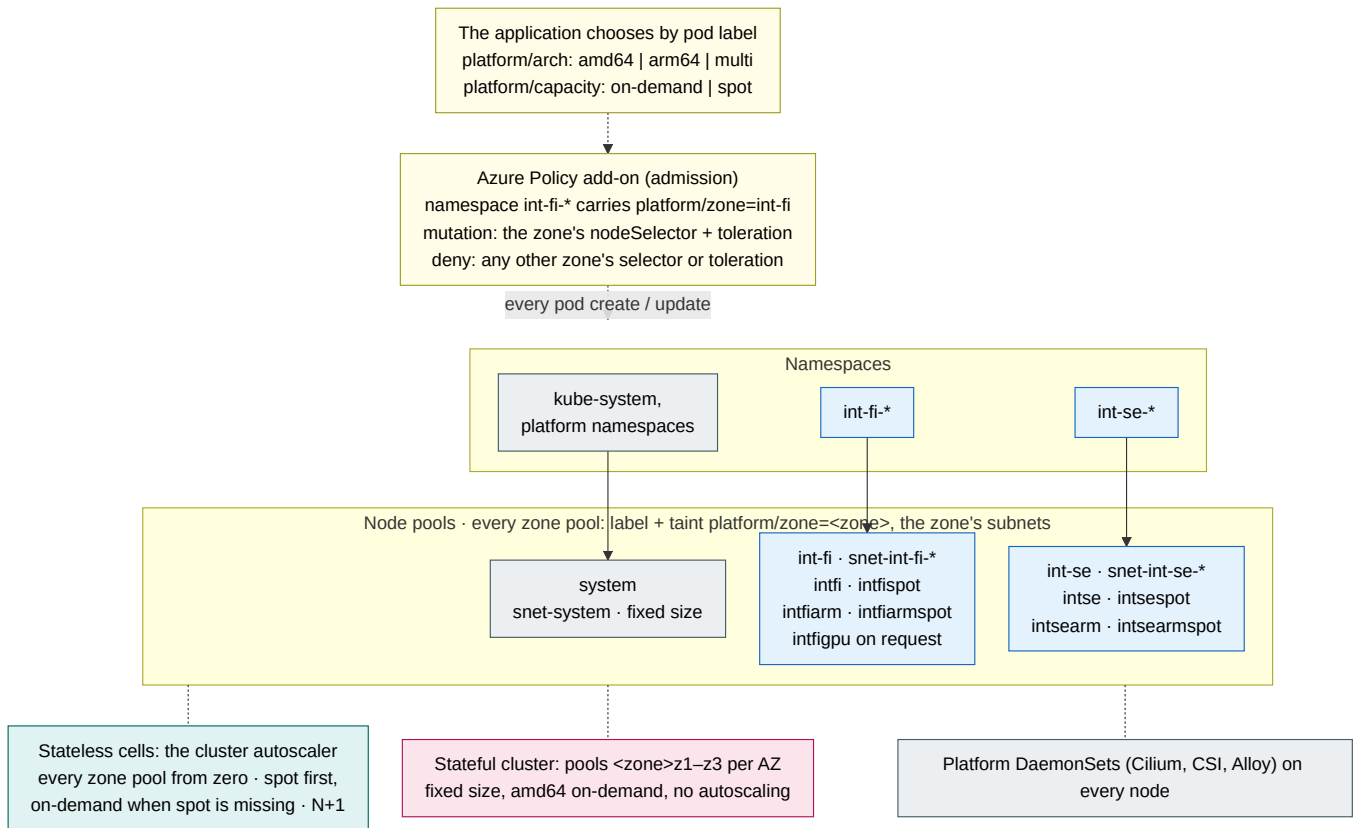
(`private_dns_zone_mode = none`), so no AKS-managed private DNS zone has to be linked to every network.

Admins are authorised by Azure RBAC for Kubernetes. The system pool keeps a fixed node count and runs only platform components.

- **No local accounts** (`--disable-local-accounts`): the certificate-based `clusterAdmin` credential would be a back door around Entra ID, PIM and the audit log. Every human, pipeline and agent signs in with Entra ID; break-glass is a PIM-activated Entra group, not a local credential. Azure Policy denies clusters with local accounts enabled.
- **Private Link, not VNet integration: API Server VNet Integration** (the API server as an internal load balancer in a delegated `snet-apiserver`) is refused by Azure on an encrypted virtual network with v4 and newer VM sizes, and the cells' networks are encrypted ([section 4.1](#)); every cell keeps a slot for its subnet. The cells use KMS data encryption, which reaches the private vault as a trusted service ([section 4.2](#)).
- **No SSH to nodes** (`--ssh-access disabled` on every node pool): nodes are replaced, never logged into. Node-level diagnosis uses `kubectl debug node/...` or a kernel-level tracing tool, through break-glass only ([section 6.2](#)).

2.3 Node pools

Requirements: [R3](#), [R8](#), [R20](#), [R23](#).



A zone pool's taint is `platform/zone=<zone>:NoSchedule`. Both cluster types run classic AKS node pools (VM scale sets); they differ in how the pools scale, because they run different kinds of workloads:

	Stateless clusters	Stateful cluster
Zone nodes	Per zone one pool per capacity type and architecture: <zone> (on-demand, amd64), <zone>spot, <zone>arm, <zone>armspot (intse, intsespot, ...: the zone without the dash)	One pool per zone and AZ (intfiz1, intfiz2, intfiz3, ...)
Subnet	--vnet-subnet-id snet-<zone>-nodes, --pod-subnet-id snet-<zone>-pods	The same
Availability zone	The cell's AZ	--zones 1 / 2 / 3, one pool per AZ
VM sizes	One fixed size per pool from the VNet-encryption-capable families, amd64 and arm64	One fixed size per pool, chosen when the workload's exception is onboarded; amd64
Capacity type	On-demand and spot , chosen per application (below)	On-demand only

	Stateless clusters	Stateful cluster
Scaling	The cluster autoscaler , every zone pool from zero : a zone without application pods has no nodes, because the zone's gateway and ESO run on the system pool (below); the on-demand pools' maximum is sized for N+1	No autoscaling . Fixed node count per pool, set in IaC; capacity changes are planned changes
Node replacement	Kubernetes and node image upgrades per pool while the cell is out of traffic, surge and PDBs respected (section 3.5)	Surge upgrade per pool, one AZ at a time (section 3.5)
Defined by	Cluster IaC: a new zone in a cell is an IaC change, never a hand-made pool	Cluster IaC
Disk encryption	Cluster's disk encryption set (<code>key-<cluster>-disk</code>) on the ephemeral OS disks; <code>--enable-encryption-at-host</code> on every pool	Cluster's disk encryption set on the OS disks, <code>--enable-encryption-at-host</code> ; persistent volumes with the zone's own key (section 4.2)
System pool	Classic node pool <code>system</code> , fixed node count, pinned to the cell's AZ	Classic node pool <code>system</code> , fixed node count, spread over AZ 1–3

Why two scaling models. Stateless workloads scale on load all day and can be moved at any time, so their pools follow the pending pods and give unused nodes back. The stateful cluster runs a few approved StatefulSets and operators that are sized when they are onboarded and do not scale on load; every node change moves a replica and re-attaches its disks. Fixed-size node pools keep its capacity predictable, and its nodes change only in planned upgrades – never because an autoscaler decided to scale down. Node auto provisioning is not offered until it supports the pod subnet.

Zone nodes only for applications

In the stateless clusters a zone's nodes exist only while the zone has application pods. The zone's platform components – its gateway (`<zone>-gateway` , [section 4.1](#)) and its External Secrets Operator (`<zone>-secrets` , [section 2.5](#)) – run on the system pool next to the platform's own controllers, so they never hold a zone node. A zone that no application uses yet, or no longer, costs no VMs: the cluster autoscaler adds the first node when an application pod of the zone is pending and removes the last one once it is empty. The zone's separation does not depend on where these two components run: the gateway's load balancer is still the zone's own, in the zone's subnet, its routes are still limited to the zone's namespaces, the ESO still reaches only the zone's vault, and Cilium policies still separate both from every other zone. What the system pool loses is node-level separation between the zones' gateways, which share the platform's nodes; it is sized for them (at least two Envoy replicas per zone and cell, the minimum of App Routing's HPA). The gateways also share one control plane per cell, App Routing's istiod ([section 4.1](#)): the data plane is per zone, the control plane is not.

Zone pools in the stateless clusters:

- All nodes run **Azure Linux 3.0** (`--os-sku AzureLinux` on every pool): the smaller image with fewer packages to patch.
- **LocalDNS is stated, not inherited.** From 1.37 AKS turns its node-level DNS proxy on by itself on every pool “where the networking supports it” (a cell running Cilium network policies is left out today, but that list is

AKS's to change per minor). Every pool therefore carries an explicit `LocalDNS mode - Disabled` - so a minor upgrade never changes how pods resolve names.

- The autoscaler removes a node only after it has been empty or under-used for the scale-down delay, and drains it through the PDBs; all applications have ≥ 2 replicas and a PDB, so a scale-down never takes an application down.
- Node pools are cluster IaC, created only by the deployment identity; the VM sizes are limited to families with VNet encryption and encryption at host.

CPU architecture and spot, chosen by the application

To lower cost, the stateless clusters offer **amd64** and **arm64** nodes and **on-demand** and **spot** nodes in every zone. The platform supports all four combinations; the team that builds an application decides what the application supports and declares it with two labels in the pod template. Without labels an application gets the safe default, amd64 on on-demand nodes, so nothing changes for applications that do not opt in.

Pod label	Values	Effect (injected by Azure Policy mutation)
platform/arch	amd64 (default)	nodeSelector <code>kubernetes.io/arch: amd64</code>
	arm64	nodeSelector <code>kubernetes.io/arch: arm64</code>
	multi	no architecture constraint: the scheduler may use any free node of either architecture
platform/capacity	on-demand (default)	nothing – the pod cannot tolerate the spot taint, so it runs only on on-demand nodes
	spot	tolerations of <code>platform/capacity=spot:NoSchedule</code> and of AKS's own <code>kubernetes.azure.com/scalesetpriority=spot:NoSchedule</code> , and a <i>preferred</i> node affinity for <code>kubernetes.azure.com/scalesetpriority: spot</code> : the pod runs on spot nodes when there are any, and on on-demand nodes when spot capacity is not available

How the platform handles them:

- **Four pools per zone:** the spot pools are tainted `platform/capacity=spot:NoSchedule` next to the taint AKS puts on every spot pool. The cluster autoscaler's **priority expander** (its ConfigMap from the platform repository) grows them first; when Azure has no spot capacity in the cell's AZ it backs off that pool and grows the on-demand one - a slower fallback, the same contract for applications. The spot taint is what keeps every application that did not choose spot away from spot nodes, also from spot nodes that already exist and have room.
- **VM sizes per architecture:** amd64 families (e.g. Dsv5/Dsv6, Esv5/Esv6, Daldsv6) and arm64 families (e.g. Dpsv6/Epsv6, Cobalt 100) – only those that support VNet encryption, because it is the only encryption between nodes ([section 4.1](#)), and encryption at host, which every node has ([section 4.2](#)).

- **Capacity and N+1:** the on-demand pools' maximum alone is sized for N+1 (one cell carrying the whole environment), because spot capacity can disappear exactly when a cell is pre-scaled. The spot pools' maximum caps how much of a zone can run on spot.
- **Spot evictions** come with about 30 seconds' notice and do **not** respect PDBs; Azure can evict many spot nodes at once. The autoscaler replaces evicted capacity – with spot if it is available, otherwise on-demand. An application may choose `spot` only if it tolerates losing replicas suddenly: graceful shutdown within 30 seconds, no long-running requests that cannot be retried, and work that is safe to repeat (queue consumers, batch jobs, stateless APIs with enough replicas). Latency-critical entry points and singletons-by-design stay on-demand.
- **Images:** `arm64` needs an arm64 image, `multi` a multi-architecture image (OCI image index with `linux/amd64` and `linux/arm64`). Admission cannot inspect images, so the build pipeline checks the platforms of the image index in ACR against the labels in the rendered manifests before an image is promoted, and the application's tests run on every architecture it declares. All platform DaemonSets (Cilium, CSI drivers, monitoring agent) are multi-architecture.
- **Validation:** applications set only the labels; Azure Policy rejects unknown label values, a `kubernetes.io/arch` selector that disagrees with the label (or any with `multi`) and any required affinity on it, any selector or affinity on `kubernetes.azure.com/scalesetpriority`, and pods that tolerate `platform/capacity` without the `spot` label (section 2.7).
- **System pool and stateful cluster:** the system pool stays amd64 on-demand. The stateful cluster runs only on-demand nodes – a spot eviction would move a replica and re-attach its disks without a PDB – and only amd64 pools; in the stateful cluster any label value other than the defaults is rejected. An arm64 pool set (one per zone and AZ) can be added when an onboarded workload needs it.

Choosing is a trade-off the application team owns: `multi` + `spot` is the cheapest, `amd64` + `on-demand` the most predictable. The choice can be changed in any release, because it is only a label in the pod template.

Pod right-sizing: vertical scaling in place

The autoscaler adds nodes and the N+1 headroom is sized from the pods' **requests**, so requests far from real use cost money (too high) or reliability (too low). Since Kubernetes 1.35 a running container's CPU and memory can be changed **in place**, and since 1.36 the pod-level budget too ([Kubernetes: resize pod resources in place](#)); the AKS [Vertical Pod Autoscaler add-on](#) actuates it. The platform uses it in two loops:

Loop	Who	How fast	Writes
Online: resize running pods to their usage	The VPA add-on in mode <code>InPlaceOrRecreate</code> – deterministic, in the cluster	Minutes	The pods' <code>resize</code> subresource only, within the bounds of the application's <code>VerticalPodAutoscaler</code>
Policy: which applications may be resized, the bounds, and the requests new pods start from	Right-sizing agent (section 6.1)	Daily, and on OOMKill / throttling / stuck-resize alerts	Pull requests to the fleet repository, tiered by the merge gate

The agent never resizes a pod itself: an agent with write access to pods would break *Git is the only write path*, and a language model does not belong in a loop that must react in minutes - the same split as HPA / KEDA and the cluster autoscaler, which act online inside bounds that are in Git.

- **Every stateless Deployment is right-sized by default:** the onboarding agent creates a `VerticalPodAutoscaler` next to the application's manifests with `updateMode: InPlaceOrRecreate` and safe starting bounds from the declared requests (memory `minAllowed` = the declared request, CPU `minAllowed` a quarter of it, `maxAllowed` twice the declared values); the right-sizing agent widens them after 14 days of usage. The requests new pods start from are a patch next to it. CI checks that every Deployment has its VPA.
- **Opt-out:** the pod label `platform/rightsizing: off` keeps the VPA in `updateMode: "Off"` – recommendations only, which the agent still uses to move the baseline in Git. The agent falls back to `Off` by itself for an application whose resizes misbehave, and says so in an issue.
- **Bounds:** `controlledValues: RequestsOnly` and a `maxAllowed` for every controlled resource, checked by the merge gate against `limits.yaml`; a pull request that moves each value at most $\times 2$ is tier low. **Only requests move, limits stay in Git**, so a runtime that sizes its heap from the limit at start is not affected; `resizePolicy NotRequired` keeps a resize from restarting the container.
- **HPA / KEDA:** an HPA on CPU utilisation computes against the CPU request, so the VPA then controls only memory (and the reverse); an application whose HPA scales on both stays `off`.
- **Stateful cluster:** VPA objects only in `off` – every size change is a planned change through the pipeline.
- **What keeps it safe:** the kubelet admits a resize only if it fits the node (`PodResizePending` , deferred or infeasible); the autoscaler adds no node for a deferred resize, the VPA then evicts the pod (PDB respected) and the pending pod brings the node. `maxAllowed` must therefore fit the zone pools' VM size and the namespace quota, and admission policies do not see the `resize` subresource. A resize changes the pod, not the Deployment, so Flux sees no drift; the agent keeps Git close to the VPA's target so a rebuilt cell does not start from stale numbers. Memory QoS (1.37) makes a too-low request reserve memory, which the first 1.37 cell soaks for.

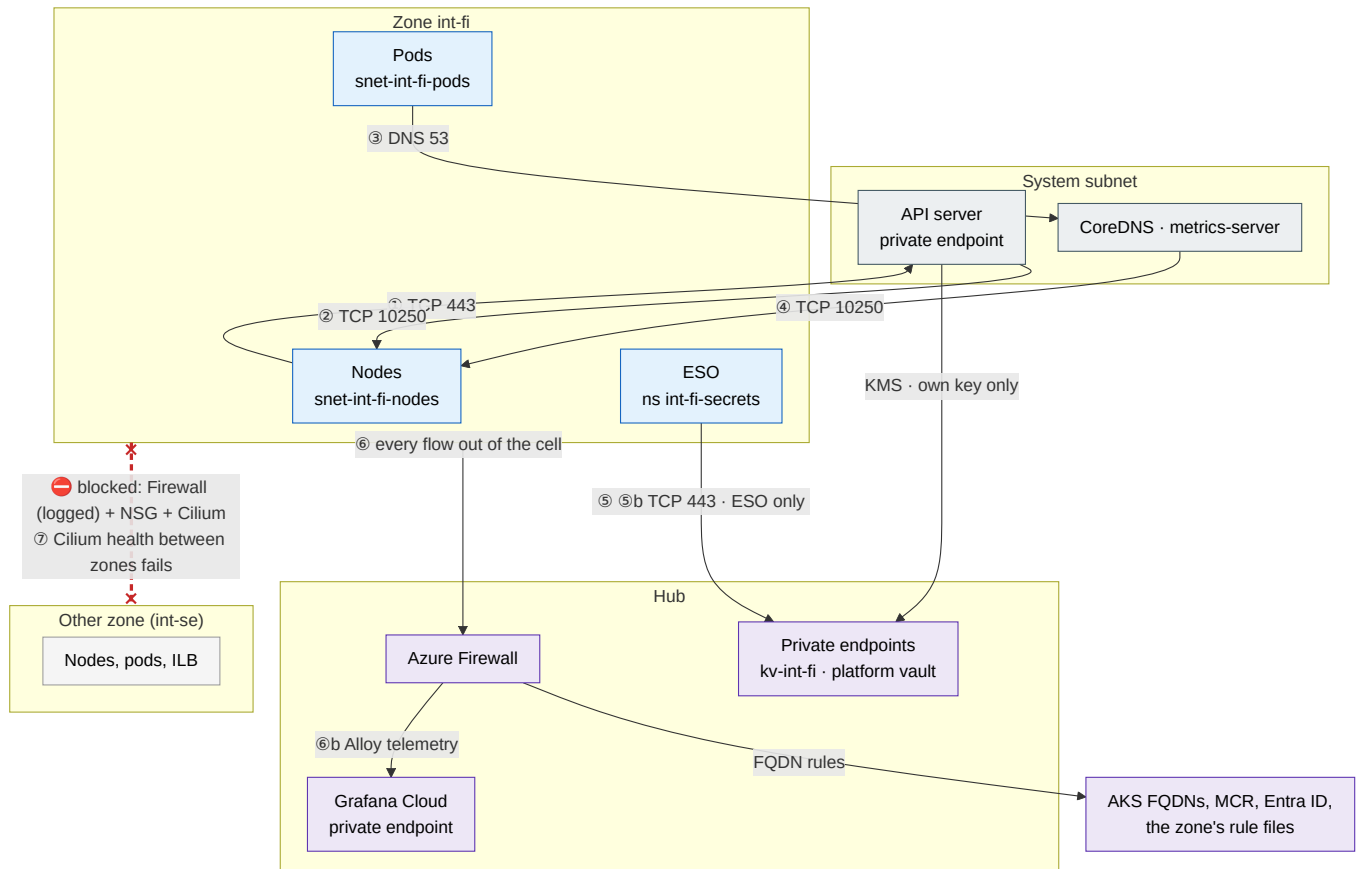
Zone pinning

Application namespaces are named `<zone>-<app>`; the picture shows the admission. There is one mutation definition per zone, matched by the namespace prefix `<zone>-*`. The checks use the namespace *name*, not a lookup of the namespace's labels, because custom Azure Policy definitions cannot use Gatekeeper data replication; the same `zone-placement` rule makes sure the `platform/zone` label of a namespace matches its name prefix.

Note: the kubelet identity is cluster-wide in AKS, so it is **never** granted access to Key Vaults – workloads use workload identity only (picture 2.5).

2.4 Allowed and blocked flows

Requirements: [R2](#), [R3](#), [R4](#), [R7](#), [R10](#), [R11](#), [R13](#), [R29](#).



Only the flows Kubernetes needs to function are allowed between a workload zone and the rest of the platform:

#	Source	Destination	Port	Purpose
①	Zone nodes, system pool	The API server's private endpoint in <code>snet-system</code>	TCP 443	kubelet / pods → API server over Private Link
②	<code>snet-system</code> (the AKS tunnel)	Zone nodes	TCP 10250	logs, exec, port-forward; the NSG allows 10250 from the system subnet only
③	Zone pods	CoreDNS (system pods)	UDP/TCP 53	name resolution
④	metrics-server (system pods)	Zone nodes	TCP 10250	resource metrics
⑤	The zone's External Secrets Operator (<code><zone>-secrets</code> pods)	Own zone Key Vault, through the hub Firewall to its private endpoint in the environment's <code>snet-shared-pe</code>	TCP 443	application secrets; application pods never reach a Key Vault (section 2.5); the Firewall's rule takes the cells' system subnets as source

#	Source	Destination	Port	Purpose
⑤b	The zone's External Secrets Operator (<zone>-secrets pods)	Platform Key Vault, the same way	TCP 443	certificates for the zone gateways and TLS-terminating apps; Cilium policy allows only these pods, RBAC only their zone's certificates
⑥	Zone nodes + pods	The exposure's Azure Firewall	per FQDN	AKS required FQDNs, MCR, Entra ID (workload identity token exchange); pods only to the FQDNs of their zone's per-application rule files
⑥b	Zone nodes (Grafana Alloy of the node)	Grafana Cloud, through the Firewall	TCP 443	Metrics, logs and traces of the node's own pods, to a private endpoint in the hub so that nothing crosses the Internet (section 6.2)
⑦	Zone nodes	The zone's own nodes and the system pool	TCP 4240, ICMP	Cilium health (optional); between two zones the Firewall drops it, so <code>cilium-health</code> lists the other zones' nodes as unreachable
–	API server	Platform Key Vault	TCP 443	etcd encryption with the cluster's KMS key: KMS data encryption reaches the private vault as a trusted service (section 4.2)

Everything else between zones is blocked – pod-to-pod, pod-to-other-zone Key Vault and direct Internet. The Firewall sees it because every node and pod subnet routes the cell's address spaces to it, its own slot excepted. The flows ②, ③ and ④ cross the Firewall too, as network rules of the exposure's Firewall Policy (TCP and ICMP from the system pool to a zone, only cluster DNS and Cilium's health back), so the platform's own pods - the other zones' gateways and External Secrets Operators among them - are behind the same three layers as another zone. Flow ① stays on the VNet's /32 route to the endpoint, and an application calls another zone only as a client of its published entry point ([section 2.1](#)).

Inside the allowed flows, a pod reaches a destination outside the cluster only if a hub Firewall rule names it by **FQDN**, written per application ([section 5.1](#)).

Isolation matrix

What each principal can reach, per isolation zone. "Own" means the zone, namespace or application the principal belongs to; everything not listed is blocked, and the negative tests of [section 0.3](#) try every blocked cell of this table after every change.

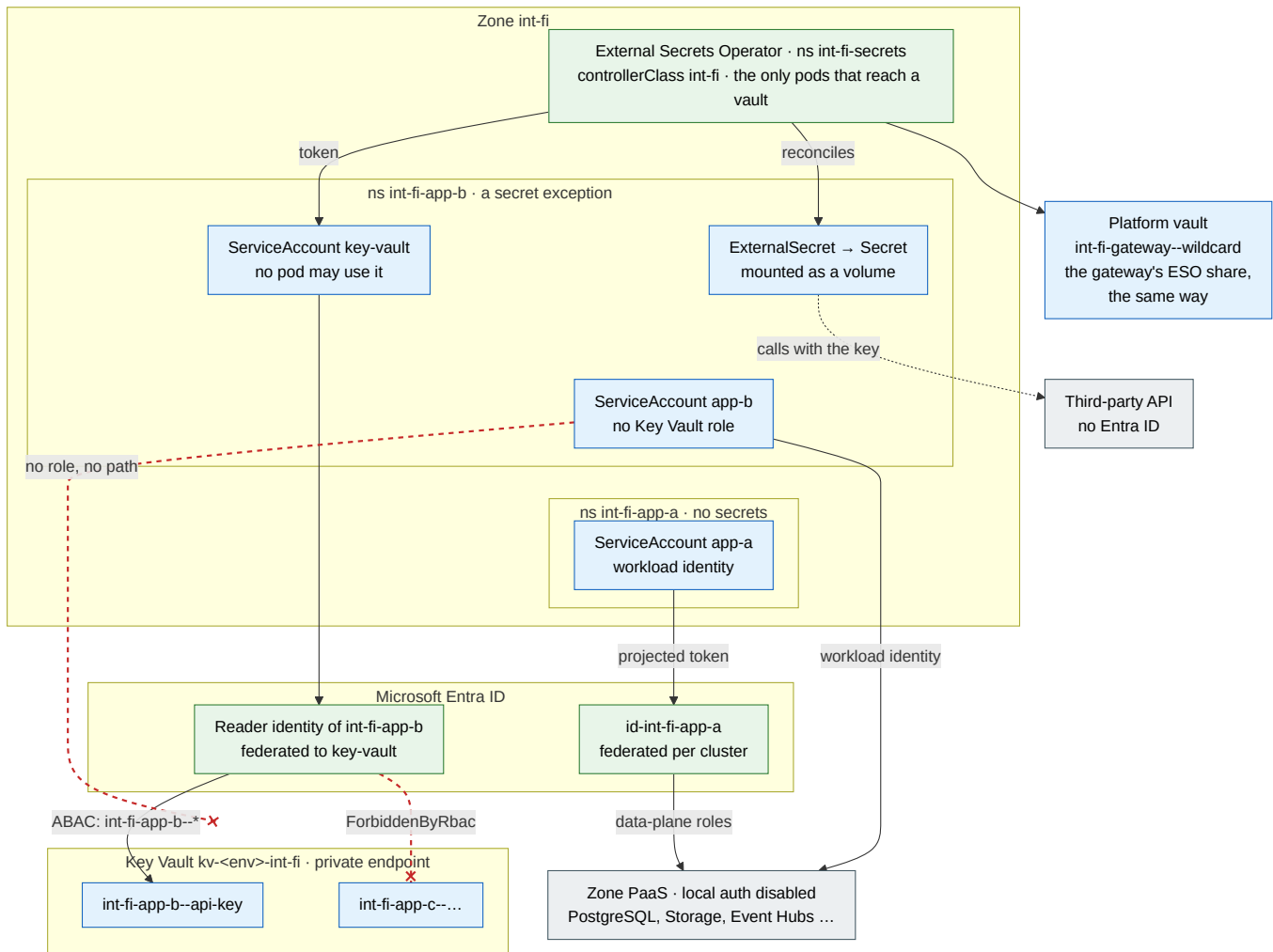
Principal	Kubernetes API	Network	Azure identities	Secrets	Certificates	Application data
Application pod	None: automountServiceAccountToken: false, no API access	Own namespace, the zone's gateway (ingress only), CoreDNS, the node's Alloy, declared FQDNs of the own zone (the published names of its own zone's and another zone's applications among them); never IMDS, another zone's pods, nodes or load balancer directly, a vault or the stateful cluster of another zone	Own workload identity (id- <zone>- <app>), roles only on the zone's own resources	Only the Kubernetes Secret s ESO wrote into its namespace	The zone's wildcard (or its own) as a Secret , if onboarded with one	Own PaaS resources through the identity
Application team (humans)	Own namespaces only (Entra ID group), read-only in acc and prd	–	None; pipelines write secrets to the zone vault under the application's prefix only	Own prefix in the zone vault (pipeline)	None	Through the application only
Zone ESO controller (<zone>-secrets)	Secret writes and serviceaccounts/token in the zone's namespaces	The zone vault and the platform vault, through the hub Firewall	None of its own; uses each namespace's secrets-reader token	Every application's reader identity of the zone, each limited by ABAC to its prefix	The zone's certificates, by per-object RBAC	None
Zone gateway (<zone>-gateway)	None with App Routing (istiod configures it); read-only Gateway API access with Traefik	Own zone's application pods, istiod; receives from Internet or corporate network	None	None	The zone's wildcard, through ESO	None

Principal	Kubernetes API	Network	Azure identities	Secrets	Certificates	Application data
Other zone, same cell	Blocked (RBAC by namespace)	Blocked by the Firewall (logged), NSG and Cilium; its published entry point only with an application rule in the caller's rule file	Different identities	Different vault	Different certificate	Different PaaS resources, private endpoints in the other zone's subnet
Stateful cluster, same zone	Separate cluster	Only from the same zone of the stateless clusters, on the application's TLS port	Same pattern, federated per cluster	Same zone vault	Same zone certificate	Zone's volumes under the zone's own key
Platform team	Read-only everywhere; writes only through Git	Management VNet → API servers	None standing; PIM	None standing; PIM on the vaults, alerted	None standing	None
AI agents	AKS RBAC Reader (no Secret s)	Management VNet → API servers, model endpoint, Git	Read-only Azure roles	Metadata only (security agent)	None	None
Deployment identities (IaC, Flux, stateful pipeline)	Write, scoped to platform and application namespaces; Flux cluster-wide in its cell	ACR private endpoint	Create and assign identities (IaC only)	Assign roles (IaC only), never read values	Create placeholders (IaC), issue (renewal job)	None
Break-glass (PIM)	Cluster admin for the activation, logged and alerted	Management VNet	Owner for the activation	Read with alert	Read with alert	Through the resources, with alert

A compromised application pod therefore reaches its own namespace, its own zone's declared destinations and its own PaaS resources, and nothing else; a compromised ESO controller reaches one zone's secrets; a compromised cluster administrator reaches one cell. Country separation inside one cluster rests on these controls, not on a separate control plane, which the security team accepts for countries.

2.5 Workload identity first, Key Vault secrets only when needed

Requirements: [R3](#), [R9](#), [R10](#), [R11](#), [R12](#).



Rule: an application authenticates with its workload identity wherever the target supports Microsoft Entra ID; a secret is used only when there is no other way. No passwords, connection strings with keys, SAS tokens or storage account keys for Azure services.

Workload identity per application

- Each application has, per isolation zone and environment, its own user-assigned managed identity (`id-<zone>-<app>`), created by IaC together with the namespace. Its Kubernetes ServiceAccount carries the `azure.workload.identity/client-id` annotation; pods get a short-lived projected token that is exchanged for an Entra ID token – nothing long-lived is stored anywhere.
- Federated credentials per cluster:** every cluster has its own OIDC issuer, so the identity has one federated credential per cluster of the environment (`sl-az1` , `sl-az2` , (`sl-az3`), `sf`) for subject

`system:serviceaccount:<namespace>:<app>` . A rebuilt stateless cluster gets a new issuer; the IaC that creates the cluster also updates the federated credentials (a managed identity allows at most 20). When an identity would need more - a platform identity federated to every cluster of a large fleet - the path is AKS's [identity bindings](#) (GA since September 2026): one federated credential per identity, the clusters bound to it by AKS; the applications' identities, one per namespace and environment, stay far below the limit and keep the plain credentials.

- The identity gets **data-plane roles directly on the Azure resources of its zone**, never on other zones:

Target	How the application authenticates	Local auth disabled with
Azure SQL Database	Entra token (<code>Active Directory Default</code> in the driver), contained database user for the identity	Entra-only authentication
Azure Database for PostgreSQL	Entra token as password, role mapped to the identity	Entra-only authentication
Storage (Blob, Queue, Table, Files REST)	<code>Storage Blob Data ... roles</code>	<code>allowSharedKeyAccess: false</code>
Service Bus, Event Hubs	<code>... Data Sender / Receiver roles</code>	<code>disableLocalAuth: true</code>
Cosmos DB	Cosmos DB data-plane RBAC	<code>disableLocalAuth: true</code>
Azure Cache for Redis	Entra authentication, access policy for the identity	access keys disabled
Key Vault	No role – applications never call Key Vault; their secrets are delivered by External Secrets Operator (below)	RBAC permission model
Other Azure APIs (App Configuration, Azure OpenAI, ...)	Azure RBAC roles	<code>disableLocalAuth</code> where available

“Local auth disabled” is enforced with Azure Policy on the resources, so a key-based fallback cannot be switched on later. Application configuration contains only endpoints and client IDs, which are not secrets. The table lists the Azure services; the ones without an equivalent in every cloud (Azure SQL, Service Bus, Cosmos DB's NoSQL API) are available only as a documented cloud-pinning exception ([section 1.3](#)).

- **Platform components use workload identity too:** Flux (`provider: github` on its Git sources: a GitHub App's short-lived installation token, no stored key), the stateful deployment pipeline (workload identity federation, no client secret), External Secrets Operator ([below](#)), external-dns. Unlike applications, platform components may use Key Vault directly with their own identity ([below](#)). Images are pulled with the kubelet identity, which has only `AcrPull` on the environment's ACR – no `imagePullSecrets` .

Secrets only when needed

A secret is allowed only when the target cannot use Entra ID: third-party APIs and SaaS keys, partner systems and legacy protocols. TLS certificates are **not** in the zone vaults; they live in the environment's platform Key Vault

([section 4.1](#)). Such a secret always lives in the zone’s **Key Vault** – never in Git, Helm values or a Kubernetes `Secret` created by hand.

Each zone has its **own Key Vault per environment** (`kv-<env>-<zone>` , RBAC permission model, public access disabled), shared by all clusters of the environment, with one private endpoint in the environment’s shared `snet-shared-pe` , reached from the cells through the hub Firewall ([section 3.1](#)). The tenant of a vault is the **namespace**: every secret a namespace owns is named `<namespace>--<secret>` (`int-fi-example--api-key`), and a namespace name may not contain `--` . Inside a zone’s vault, applications share the vault but are separated with [Azure ABAC conditions](#):

- The namespace’s **secret reader identity** `id-<region>-<env>-shared-kv-<namespace>` (see below – not the application’s own identity) gets **Key Vault Secrets User** on the zone vault with condition `@Resource[Microsoft.KeyVault/vaults/secrets:name] StringStartsWith '<namespace>--'` .
- Application pipelines (`writers`) get **Key Vault Secrets Officer** under the namespace’s write condition - the resource name for existing secrets (`get` , `update` , `delete` , `recover` , `purge`), the request name for new ones (`setSecret` , `restore`); rotation is owned by the application team (expiry dates set, Event Grid `SecretNearExpiry` alerts).
- The namespace’s Entra ID groups get what their Kubernetes role gets on `Secret` s, under the same prefix conditions: `admin` Key Vault Secrets Officer, `edit` the custom Key Vault Secrets Editor role (writes and reads attributes, never a value), `view` nothing ([section 2.6](#)).
- The **platform viewers** ([section 2.6](#)) list every secret’s name and attributes, never a value.

Key Vault ABAC is **preview** and conditions only on the vault and secret name, so the naming convention `<namespace>--<secret>` is mandatory and enforced by the platform. ABAC is not the only boundary between two applications of a zone: each has its own reader identity and its own `SecretStore` , and admission rejects an `ExternalSecret` that names another application’s prefix ([section 2.7](#)), so a change in the preview cannot open a path on its own; the fallback, one vault per application, is a per-application flag.

Soft multi-tenancy

A zone’s applications share one vault, and the separation between them is in the data plane, not in the resource: this is **soft multi-tenancy**. What each tenant – a namespace, its identity and its groups – can do:

	Own secrets (<code><namespace>--*</code>)	Another namespace’s secrets	Secret names of the vault
Namespace identity (External Secrets Operator)	Read value and metadata, by name	Refused	Refused (no listing)
<code>admin</code> group, pipelines	Create, read, change, delete, recover, purge, by name	Refused	Refused (no listing)
<code>edit</code> group	Create and change, read attributes - never a value, no delete, recover or purge	Refused	Refused (no listing)
<code>view</code> group	–	–	–

	Own secrets (<namespace>-- *)	Another namespace's secrets	Secret names of the vault
Platform viewers	– (no value)	– (no value)	Listed, with attributes

The conditions cover `readMetadata` too, and Key Vault evaluates a condition on a list call against the whole collection, which has no name, so **no tenant can list the vault**: each works with its own secrets by name, which is all External Secrets Operator does, and nobody can create a secret outside the naming convention (the condition on a new secret is on the name in the request). Tools that browse the vault – the Azure portal among them – are for the platform viewers, whose role lists every name; so names must never carry secrets. The isolation tests sign in as a member of each group and as the namespace's identity and check every row of the table ([section 0.3](#)). The hard alternative – one vault per namespace – stays a per-namespace option.

The same pattern applies to other shared PaaS services: one instance per zone (or environment), a data-plane role per namespace scoped below the instance - a database per namespace on a PostgreSQL or SQL server, a Cosmos DB database, a storage container, a Service Bus queue or topic - the namespace's workload identity and groups as the only principals, and the names of other tenants' objects visible at most to the platform viewers. Where a service cannot scope its data plane below the instance, or a tenant needs its own keys, network or performance isolation, the namespace gets its own instance – hard multi-tenancy, a decision per onboarding.

Delivered only by External Secrets Operator

Rule: no application reads Key Vault itself. Secrets and certificates reach application pods only as Kubernetes `Secret`s written by [External Secrets Operator](#) (ESO). The rule applies to application namespaces; platform components are the exception ([below](#)). The **Secrets Store CSI driver is not used**: the AKS Key Vault secrets provider add-on (`azure-keyvault-secrets-provider`) is disabled by the cluster IaC on every cluster, so its CRDs are not installed, and the stateless storage rule refuses inline `csi` volumes ([section 2.7](#)); an Azure Policy denies enabling the add-on and an admission rule refuses `SecretProviderClass` objects.

- **One ESO controller per isolation zone**, in the platform namespace `<zone>-secrets` (a reserved application name, like `<zone>-gateway`), on the system pool – so that a zone without applications has no nodes ([section 2.3](#)) – and deployed by Flux (stateless) or the pipeline (stateful). Each instance has its own `controllerClass` (`<zone>`) and reconciles only the stores of its zone; the CRDs and the conversion webhook are installed once per cluster. Its path to Key Vault is the zone's own flow ⑤ (⑤b to the platform vault, [section 2.4](#)), so a controller can never reach another zone's vault and a compromised controller exposes one zone at most. It writes `Secret`s and requests ServiceAccount tokens only through RoleBindings in its own zone's namespaces, not through the chart's cluster-wide role ([CIS benchmark](#)). The CRDs, the conversion webhook and the certificate controller are installed once per cluster by the **platform release** in the namespace `external-secrets` (`controllerClass: platform`), which also serves the platform namespaces' own stores.
- **A secret reader identity per namespace** (`id-<region>-<env>-shared-kv-<namespace>` , created by IaC with every managed namespace), federated with one credential `aks-<cluster>` per cluster, as in the picture. Application pods may not run as `key-vault` (Azure Policy). The teams' `edit` and `admin` roles do include `serviceaccounts/token` , so their members could obtain its token – which gives them nothing they

do not already hold, since those groups manage the namespace's secrets in the vault themselves; no workload can.

- **One SecretStore named key-vault per application namespace**, generated from the onboarding by a platform preset: provider `azurekv`, `vaultUrl` of the own zone vault (the gateway and platform namespaces point theirs at the platform vault, [section 4.1](#)), `authType: WorkloadIdentity` with `serviceAccountRef: key-vault`, `controller: <zone>`. Azure Policy rejects any other store configuration. Applications write only **ExternalSecret**s that reference their own namespace's store; `ClusterSecretStore`, `ClusterExternalSecret` and `PushSecret` are platform only. An application can therefore only ever pull with its own reader identity, and Key Vault ABAC still decides which secret names that is – application A's `ExternalSecret` for `b-...` fails with `403`.
- **No other way in.** Application identities have no Key Vault role at all; no Cilium policy of an application namespace reaches a vault (only `<zone>-secrets` and the platform namespaces below do), the Firewall's rule for the vaults' private endpoints takes the cells' system subnets as its source, and Azure Policy rejects an application policy that opens egress out of the cluster ([section 2.7](#)). In application namespaces, admission accepts `Secret`s only from the zone's ESO controller (plus Helm release Secrets), so a secret in Git, Helm values or created with `kubectl` is rejected as before. Only the environment IaC module assigns Key Vault data-plane roles; an acceptance test and the security agent ([section 6.1](#)) check that no application identity holds one.
- **Consumption and rotation.** Applications mount the resulting `Secret` as a **volume** (preferred: the kubelet updates the files after a rotation; no `subPath`) or as environment variables (picked up only on the next rollout). `refreshInterval` is 1 hour by default, shorter where a rotation must take effect faster; the `SecretSynced` condition and ESO's metrics alert the platform and the application team when a sync fails.

Platform components may use Key Vault directly

Platform components in platform namespaces (`<zone>-secrets`, `external-secrets`, `flux-system`, `infra-observability`) are owned, reviewed and upgraded by the platform team, so they are not forced through ESO. Such a component may read Key Vault directly with **its own workload identity**, with a data-plane role scoped as narrowly as Key Vault allows (a single certificate or secret where possible), declared in the IaC module and reviewed with the platform change, and with the vault in its namespace's platform-owned egress policy. The Secrets Store CSI driver stays disabled for them too.

- **The zone's gateway uses ESO exactly like an application.** `<zone>-gateway` has the standard `key-vault` `ServiceAccount`, federated to its reader identity `id-<region>-<env>-shared-kv-<zone>-gateway`, which has **Key Vault Secrets User** on the platform vault under the prefix `<zone>-gateway--` – the zone's certificate is the secret `<zone>-gateway--wildcard`, the gateway namespace's share like any other ([section 4.1](#)). The gateway pods have no Azure identity at all (with App Routing istiod hands them the Secret, Traefik reads it itself; App Routing's own Key Vault integration through the Secrets Store CSI driver is not used), their egress policy names no vault, and the ESO rules of [section 2.7](#) apply to `<zone>-gateway` as to any application namespace.
- **External Secrets Operator** itself, Flux and the monitoring components follow the rules above: own identity, own narrow roles, egress policy owned by the platform.
- **Grafana Alloy** cannot read Key Vault, so its Grafana Cloud credential is delivered by the platform ESO controller in `external-secrets` through `flux-system`'s share of the platform vault (the secret `flux-`

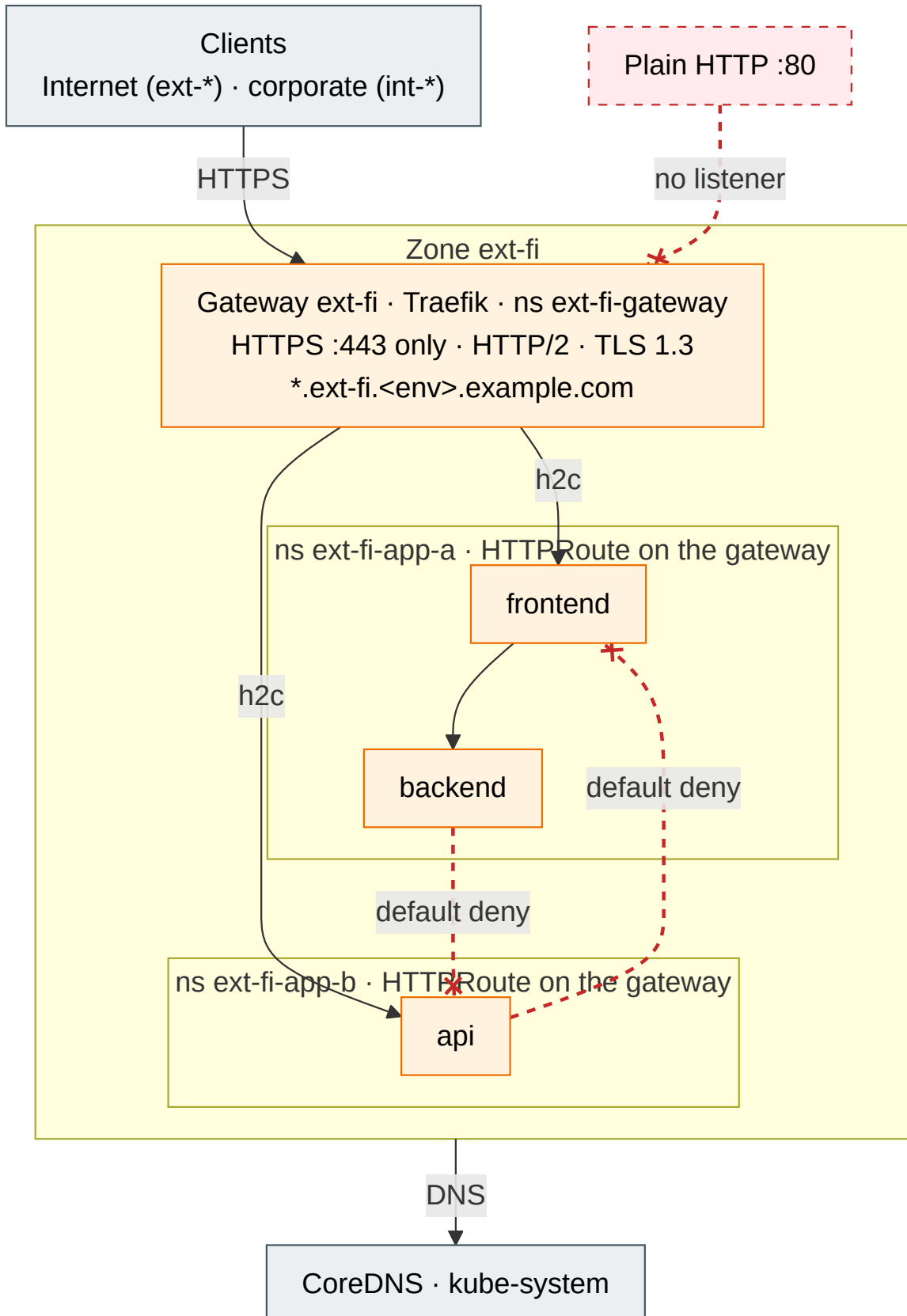
system--grafana-cloud , read by flux-system 's reader identity under its prefix, [section 6.2](#)).

Trade-offs of delivering secrets as Kubernetes Secret s instead of CSI file mounts:

- The values are **stored in etcd**, encrypted with the cluster's customer-managed KMS key ([section 4.2](#)), and readable by anyone with get secrets in the namespace: only the application team in its own namespaces, the platform team through PIM; the AI agents' Azure Kubernetes Service RBAC Reader role does not include Secret s.
- **ESO becomes a critical platform component** (one controller per zone and cluster, upgraded by the platform). In return a Key Vault or Entra ID outage no longer stops pods from starting – the Secret is already in the cluster, whereas a CSI mount fails at pod start.

2.6 Application multi-tenancy inside a zone

Requirements: [R10](#), [R13](#), [R14](#), [R22](#).



Each application gets its own namespace(s) inside a zone, with a baseline policy set applied automatically: the picture shows default deny, DNS to CoreDNS, traffic within the namespace and inbound only from the zone's gateway pods (label `gateway.networking.k8s.io/gateway-name: <zone>` in `<zone>-gateway`). Two applications of a zone talk to each other only through their published names on the zone's gateway, HTTPS on 443, each call allowed by a rule in the caller's rule file ([section 5.1](#)); nothing listens to another namespace directly, and inside its own namespace an application uses any protocol ([R22](#)). Beyond the picture:

- in the stateful cluster, inbound comes only from the node subnets of the same zone in the stateless clusters (Cilium CIDR policy; the applications' `LoadBalancer` Services use `externalTrafficPolicy: Local` so the client IP is preserved);
- no egress out of the cluster by default (the zone's `CiliumClusterwideNetworkPolicy` baseline; the namespace's own `NetworkPolicy` only closes its ingress): the zone's platform policy lets application namespaces out to the hub Firewall, which allows only the destinations named in the application's rule file – Entra ID, PaaS private endpoints, stateful cluster services and approved Internet names ([section 5.1](#)); an application writes no egress policy of its own, and Azure Policy rejects one that would open CIDRs or names out of the cluster ([section 2.7](#)). Key Vault is never on that list: secrets and certificates arrive as Kubernetes `Secret`s from the zone's External Secrets Operator ([section 2.5](#)).

Every namespace also has a ResourceQuota and a LimitRange.

Who may work in a namespace

The onboarding request names up to three Entra ID groups per namespace, one for each of Kubernetes' built-in user-facing roles:

Role	Group's members may	They may not	In the zone's Key Vault
<code>view</code>	Read most objects in the namespace	Read <code>Secret</code> s, roles or role bindings; write anything	Nothing
<code>edit</code>	<code>view</code> , and write most objects, <code>Secret</code> s included where policy lets anyone (in a zone's namespaces only External Secrets Operator does)	Change roles or role bindings	Custom Key Vault Secrets Editor on <code><namespace>--*</code> only, by name: create and change secrets and read their attributes, never a value; no delete; no listing
<code>admin</code>	<code>edit</code> , and roles and role bindings inside the namespace	Change the namespace, its quota or its default network policies	Key Vault Secrets Officer on <code><namespace>--*</code> only, by name: also read values, delete, recover and purge; no listing

None of them reaches another namespace or anything cluster-wide. The cluster IaC gives each group a `RoleBinding` `platform-<role>` in the namespace, applied by Flux through a configuration `<ns>-access` of its own (one Kustomization per role), and the Azure role `Azure Kubernetes Service Namespace User` on the managed namespace, so that its members can fetch a kubeconfig for that namespace alone (`az aks namespace get-credentials`). On the namespace's share of the zone vault the same groups get the matching rights ([section](#)

2.5): since application secrets reach the cluster only through External Secrets Operator, managing them in the vault is how a team with `edit` changes its `Secret` s. A role without a group gets no binding and no vault role. The isolation tests (section 0.3, R13) compare the bindings with the IaC and check each role's rights – in the cluster and in the vault – by impersonating a member of its group and by signing in as a test application that is one.

None of these groups can list the cluster's namespaces or the vault's secrets: other tenants' names stay hidden from them – **soft multi-tenancy**, the separation of tenants sharing one cluster and one vault in their data planes (section 2.5, which also applies the pattern to other PaaS services).

Platform viewers

Some tools need to see what the tenants may not: the Azure portal's Kubernetes views (*Workloads, Services and ingresses, ...*) start from the cluster's list of namespaces, and a vault browser from the list of secrets. The **platform viewers** are the Entra ID groups that may list both – names and attributes only, nothing inside a namespace and no secret value. A team member who is also a platform viewer picks the namespace in the portal and sees there what the namespace's group allows. The environment IaC grants them one custom role, **defined and assignable at the environment's resource group** – nothing at the subscription, so it fits a hardened corporate landing zone that keeps subscription-level role definitions to its platform team:

Permission	For
<code>Microsoft.ContainerService/managedClusters/read</code> , <code>.../listClusterUserCredential/action</code>	The clusters in the portal, a kubeconfig (Entra ID)
Data action <code>Microsoft.ContainerService/managedClusters/namespaces/read</code>	Azure RBAC for Kubernetes: get, list and watch namespaces
<code>Microsoft.KeyVault/vaults/read</code> , data action <code>Microsoft.KeyVault/vaults/secrets/readMetadata/action</code>	The vaults in the portal, every secret's name and attributes
<code>Microsoft.Resources/subscriptions/resourceGroups/read</code>	The resource group in the portal

The isolation tests check that the role is assignable at the resource group only, and sign in as a test member: it lists the namespaces and every vault's secret names, and reads nothing in a namespace, no secret value, and writes nothing.

Ownership and cost

Every namespace says who owns what runs in it and who pays for it (R14):

Metadata	Example	Used by
Label <code>platform/team</code>	<code>payments</code>	Cost and usage per team (OpenCost, Grafana), alert routing

Metadata	Example	Used by
Label <code>platform/cost-center</code>	<code>cc-4711</code>	Charging the team's share of the cells
Annotation <code>platform/contact</code>	<code>payments-team@example.com</code>	Whoever finds something wrong with the namespace

The onboarding request names them, and the IaC sets them on the namespace; the platform's own namespaces are `platform`. Azure Policy (`namespace-ownership` , [section 2.7](#)) requires them on every namespace but the ones AKS creates. **OpenCost** ([section 6.2](#)) prices every node from Azure's retail prices and splits its cost over the namespaces by what their pods request and use, so a team's share of the cell is a query by these labels. A monthly budget on each environment subscription alerts the owners at 80 % and 100 % of the actual cost and at a forecast of 100 % ([section 3.2](#)).

2.7 Policy enforcement with the Azure Policy add-on

Requirements: [R3](#), [R8](#), [R11](#), [R14](#), [R15](#), [R22](#), [R24](#), [R29](#).

The AKS-native **Azure Policy add-on** is the single policy engine, enabled by the cluster IaC in every cluster. It runs Gatekeeper, but AKS installs and upgrades it, and the policies are ordinary Azure Policy definitions and assignments – the same tool that governs the Azure resources (cluster settings, allowed VM sizes, VNet encryption, local auth disabled, private endpoints only). No Gatekeeper is installed by Flux.

- **Assignments per environment and exposure subscription** (`sub-aks-dev-int` , `sub-aks-dev-ext` , ... `sub-aks-prd-ext`), so a new or rebuilt cluster gets every policy automatically; the zone mutations and validations of a subscription are made for its exposure's zones only. Compliance of all clusters is visible in one place (Azure Policy, and the company's security tool - not Microsoft Defender).
- **Built-in definitions first:** the Kubernetes pod security *restricted* initiative, allowed images, required probes and resource requests, and [deployment safeguards](#).
- **Custom definitions** (`Microsoft.Kubernetes.Data` mode) for the platform's own rules, with the constraint or mutation template embedded as Base64 in the definition. Templates are kept in the IaC repository and tested in CI with `gator verify` . The templates are Rego; CEL-based definitions (generated as `Kubernetes ValidatingAdmissionPolicy` , evaluated in-process) are the choice for new validations once Azure Policy supports them for the rule in question.
- **Rollout dev → acc → prd:** definitions are versioned; a new or changed definition is assigned with effect `audit` in dev, then `deny` , then promoted to acc and prd with the rest of the platform change.
- **Exemptions expire.** An Azure Policy exemption always has an owner, a reason, a ticket and an `expiresOn` date (at most 6 months). The security agent reports exemptions that expire within 30 days, and renewing one is a new pull request in the high tier ([section 6.1](#)). Exceptions should stay visible and a little uncomfortable, not pile up silently.
- **Continuous evidence, not a one-time hardening.** The compliance state of the assignments is the drift signal. Next to it, the CIS Kubernetes benchmark of the nodes (kube-bench, run as one read-only Job per node, weekly in acc and prd) and the security tool's posture score per environment go to Grafana Cloud. A *drop* in either raises an alert to the security agent, which matters more than the absolute score. The CIS AKS benchmark and the NSA/CISA Kubernetes hardening guidance are the reference controls, and the checklist is repeated with every Kubernetes minor upgrade, because new minors can change defaults. The full benchmark – nodes (kube-bench), Kubernetes policies (Kubescape) and the cluster's Azure settings – also runs as a pipeline stage after every cell change, with every accepted failure listed with its reason ([section 0.3](#)).
- **Platform namespaces:** `kube-system` , `gatekeeper-system` and `azure-arc` are excluded by the definitions' defaults; the assignments exclude the platform's own namespaces (`flux-system` , `external-secrets` , `external-dns` , `platform-gateway` , `infra-observability` , `aks-command` , the AKS add-on namespaces, `<zone>-gateway` and `<zone>-secrets`) from the application rules through the namespace exclusion parameters; the secrets rules keep `<zone>-gateway` in. The zones' own platform namespaces, `<zone>-gateway` (the zone's gateway) and `<zone>-secrets` (External Secrets Operator), run on the

system pool so that a zone without applications has no nodes ([section 2.3](#)); they are excluded from zone pinning and the other application rules, and their policies are the platform's own. The `<zone>-gateway` namespaces enforce the Pod Security Standard *baseline* and only audit and warn on *restricted*: App Routing's gateway pods set no seccomp profile, and their `securityContext` cannot be customised.

- **Mutations fill in, validations decide.** A mutation only adds what a pod leaves out, so a pod that states its placement differently is not changed but reported (audit) or refused (deny) by the matching validation. A recorded exception is visible the same way: a pod labelled `platform/zone-placement: "off"` is skipped by the zone mutation and stays non-compliant in the zone validation.
- **Which clusters a rule reaches** is set by the assignment, not the definition: a `Microsoft.Kubernetes.Data` rule can test only a cluster's type, location and tags. The stateless rules leave out the stateful cluster with `notScopes`, and the stateful rules leave out the stateless cells. Custom mutations cannot be parameterized, so the zone mutations are one definition per zone, generated by the IaC.

Limits of the add-on and how the design handles them:

Limit	Consequence in this design
Needs <code>data.policy.core.windows.net</code> , <code>store.policy.core.windows.net</code> , <code>dc.services.visualstudio.com</code> (+ Entra ID); no Private Link	Firewall application rules for these FQDNs, also from the stateful cluster (exception to "no Internet", see flows)
Custom definitions cannot use Gatekeeper data replication (no lookups of other objects)	Zone rules match the namespace name prefix <code><zone>-*</code> ; "PDB exists for every Deployment / StatefulSet" is checked in CI on the rendered manifests, because Flux and the stateful pipeline only apply validated commits
Policies are synced from Azure every 15 minutes	The stateless cluster bootstrap keeps the Flux <code>apps</code> Kustomization suspended until all assigned constraint templates and constraints are present in the cluster; the stateful pipeline checks the same before every run
Gatekeeper config cannot be changed	No custom Gatekeeper settings; defaults are fine for this design
Max 10 000 pods per cluster	Well above the expected size; monitored
The built-in deployment safeguards initiative has a warn-only member that reports every pod a mutation changed ("resource was mutated by platform mutation")	The platform's mutations change every application pod by design (zone placement from the namespace, architecture, spot), so the assignment disables that member: it is a note, not a control, and the validations that check what the mutations wrote stay. Placement cannot be injected without a mutation unless a second admission engine is added, which R15 rules out

Day-1 set. The first cluster of any environment is accepted only with these rules in `deny` (they are the acceptance checklist of a new cell): zone placement from the namespace (mutation and validation), `platform/arch` and `platform/capacity` mutations, images by digest from the environment's ACR, Secrets only through ESO and no Secrets Store CSI, no application egress policy out of the cluster, `PersistentVolumeClaim` rejected in stateless cells, Gateway and HTTPS-only listener rules, h2c on every routed Service port, `LoadBalancer` and `NodePort` only in `<zone>-gateway`, no `Service.spec.externalIPs` (the built-in "allowed external IPs" rule with an empty list: deprecated in Kubernetes 1.36, removal planned for 1.43, and the oldest service hijack in the book - CVE-

2020-8554), requests and probes set, the *restricted* pod security standard, and the cluster-level rules (Cilium, Azure Linux, local accounts and SSH off, CMK). Everything else in the table starts in `audit` and is promoted rule by rule.

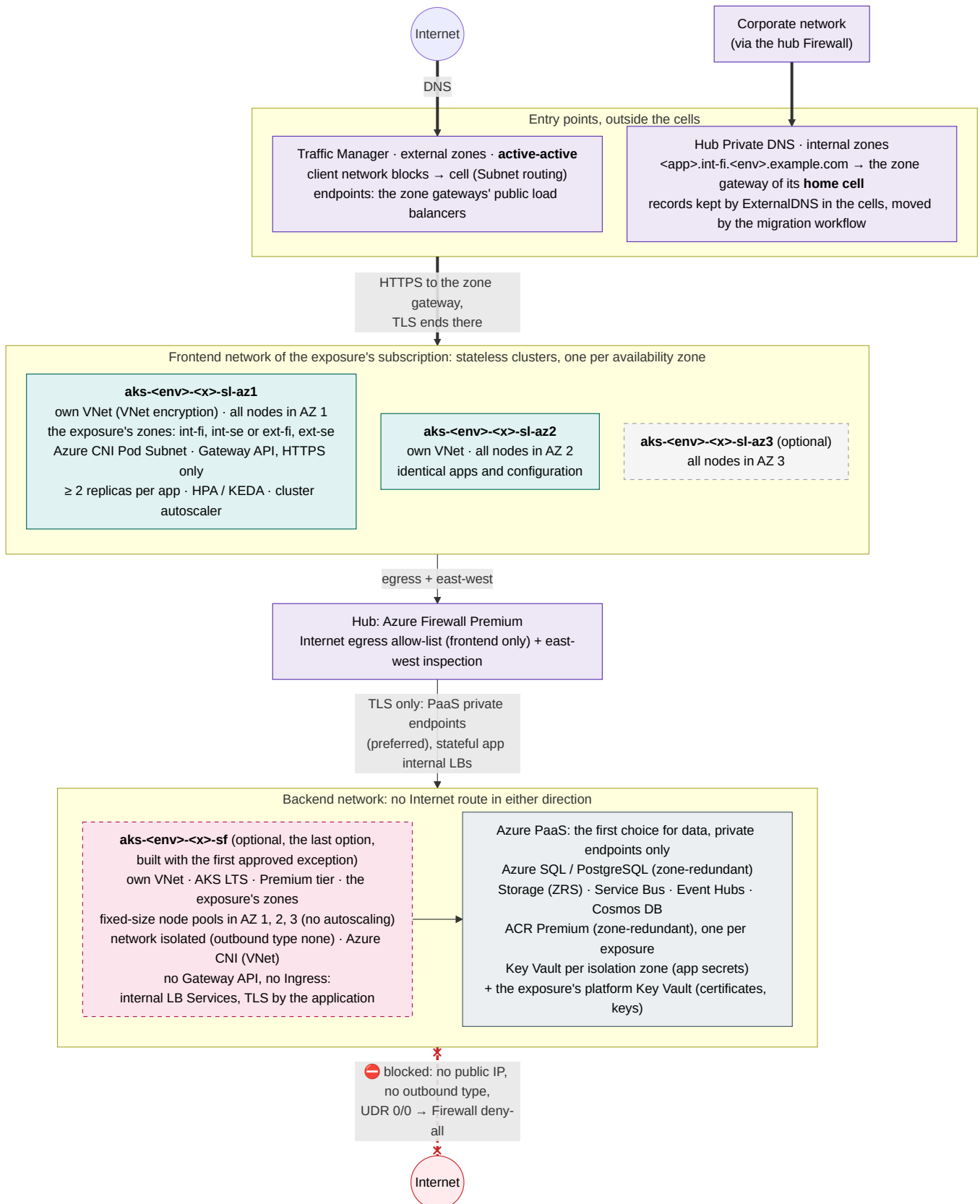
Policy	All clusters	Stateless	Stateful
Namespace <code><zone>-*</code> has label <code>platform/zone=<zone></code> ; pod nodeSelector / toleration match the prefix, no other zone's and not the system pool (mutation + validation)	✓		
<code>platform/arch</code> / <code>platform/capacity</code> labels: inject the architecture nodeSelector and the spot toleration + preferred affinity (mutation); reject unknown values, own <code>kubernetes.io/arch</code> / <code>kubernetes.azure.com/scalesetpriority</code> selectors and spot tolerations without the label (section 2.3)	✓	all values	defaults only (<code>amd64</code> , <code>on-demand</code>)
Namespace carries <code>platform/team</code> , <code>platform/cost-center</code> and the annotation <code>platform/contact</code> (all but AKS's own namespaces; section 2.6)	✓		
Images by digest from the environment's ACR only	✓		
Images signed by the platform's CI (a Notation signature with a Key Vault key, stored next to the image in the ACR). Enforced at promotion : the promotion pipeline verifies the signature before it imports an image into the next environment's ACR, and only that pipeline can write to the acc and prd registries, so an unsigned image never reaches them. Audited at admission with Image Integrity (Ratify on Gatekeeper), which today is preview and supports only the <code>audit</code> effect; it moves to <code>deny</code> when that is supported. Ratify trusts the platform's certificate by its subject and reads it from <code>kv-<env>-platform</code> , and the signatures from the ACR, as one shared identity per environment	✓		
ServiceAccount used by application pods is not <code>key-vault</code> ; no <code>imagePullSecrets</code> on pods or ServiceAccounts (that the ServiceAccount carries the workload identity client ID is a CI check: a custom definition sees only the object under review)	✓		
Secrets only through External Secrets Operator (section 2.5): in application namespaces and <code><zone>-gateway</code> , <code>Secret</code> s may be created or changed only by the zone's ESO controller ServiceAccount (<code>system:serviceaccount:<zone>-secrets:external-secrets-<zone></code> , checked on the admission request's user) – Helm release Secrets, the AKS Flux extension's agents and the KMS rewrite (<code>aksService</code>) excepted; the Secrets Store CSI add-on is disabled by the cluster IaC, inline <code>csi</code> volumes are refused by the storage rule and <code>SecretProviderClass</code> objects by their own rule	✓		
ESO objects in application namespaces and <code><zone>-gateway</code> : <code>SecretStore</code> named <code>key-vault</code> , provider <code>azurekv</code> with the own zone vault or the platform vault, <code>authType: WorkloadIdentity</code> , <code>serviceAccountRef: key-vault</code> in the own namespace, <code>controller: <zone></code> ; <code>ExternalSecret</code> only against a <code>SecretStore</code> (not <code>ClusterSecretStore</code>), <code>remoteRef.key</code> / <code>dataFrom</code> names under the namespace's <code><namespace>--</code> prefix (no <code>find</code> , no <code>sourceRef</code>); <code>ClusterSecretStore</code> , <code>ClusterExternalSecret</code> , <code>PushSecret</code> platform only	✓		
Requests set, probes set, no privileged / <code>hostNetwork</code> / <code>hostPath</code> for applications	✓		

Policy	All clusters	Stateless	Stateful
Beyond the <i>restricted</i> pod security standard: <code>readOnlyRootFilesystem: true</code> (writable paths only as <code>emptyDir</code>), <code>automountServiceAccountToken: false</code> for application pods (they do not call the Kubernetes API; workload identity uses its own projected token)	✓		
<code>VerticalPodAutoscaler: updateMode Off / Initial / InPlaceOrRecreate</code> , <code>controlledValues: RequestsOnly</code> , <code>maxAllowed</code> set for every controlled resource (pod right-sizing); also checked by the merge gate against <code>limits.yaml</code> . Every Deployment has one, in <code>InPlaceOrRecreate</code> unless its pods carry <code>platform/rightsizing: off</code> (CI check on the rendered manifests)	✓	all three modes	<code>off</code> only
Application policies stay inside the cluster (<code>app-egress</code>): a <code>CiliumNetworkPolicy</code> or <code>NetworkPolicy</code> in an application namespace may allow egress only to <code>toEndpoints</code> and in-cluster <code>toServices</code> with a <code>k8sService / k8sServiceSelector</code> (the policy pairs of section 2.6); <code>toCIDR</code> , <code>toCIDRSet</code> , <code>toEntities</code> , <code>toFQDNs</code> and <code>ipBlock</code> are rejected, because the hub Firewall is the one decision out of the cluster (section 5.1); <code>CiliumClusterwideNetworkPolicy</code> and <code>Cilium CIDR / egress gateway</code> objects platform only; <code>dnsPolicy: ClusterFirst</code>	✓		
Cluster: Azure CNI powered by Cilium, encryption at host, customer-managed disk keys, private cluster, workload identity, the Azure Policy add-on, local accounts and SSH disabled (built-in definitions on the cluster resource, section 2.2); Azure Linux 3.0 on every node pool; ACNS and the Key Vault secrets provider add-on off	✓		
Replica and rollout guardrails (PDB existence: CI check) of section 3.4 : <code>replicas / HPA minReplicas</code> , <code>topologySpreadConstraints</code> , <code>RollingUpdate</code> with <code>maxUnavailable: 0</code> , <code>PodDisruptionBudgets</code> with <code>maxUnavailable ≥ 1</code> (never <code>minAvailable</code> , which can block every drain); <code>replicas: 0</code> allowed for an application switched off; <code>Job s</code> , <code>CronJob s</code> and queue consumers are exempt from the replica, spread, budget and rollout rules (non-serving workloads)	✓	≥ 2 replicas, spread over nodes	≥ 3 replicas, zone spread
No persistent storage: <code>StorageClass</code> , <code>PersistentVolume</code> , <code>PersistentVolumeClaim</code> , inline <code>csi</code> and <code>ephemeral</code> volumes rejected (always deny, kube-system excepted)		✓	
Reject <code>Ingress</code> ; only <code>HTTPS Gateway</code> listeners; <code>HTTPRoute</code> only to the zone's <code>HTTPS</code> listener and zone domain		✓	
Every port of every <code>Service</code> in an application namespace declares <code>appProtocol: kubernetes.io/h2c</code> - the gateway speaks HTTP/2 to the pod, never HTTP/1.1 (<code>h2c-upstream</code> , section 4.1 ; a custom definition cannot see which Services an <code>HTTPRoute</code> references, so every port of the namespace's Services is held to it); also a CI check on the rendered manifests		✓	
The gateway's TLS offers ALPN <code>h2</code> only and TLS 1.3 only: Traefik's default TLS options, a file-provider <code>ConfigMap traefik-dynamic</code> in every <code><zone>-gateway</code> namespace, which the rule <code>gateway-tls</code> reads back - <code>alpnProtocols</code> exactly <code>[h2]</code> , <code>minVersion</code> exactly <code>VersionTLS13</code> ; the isolation tests prove the handshakes		✓	

Policy	All clusters	Stateless	Stateful
Reject <code>LoadBalancer</code> / <code>NodePort</code> Services outside <code><zone>-gateway</code>		✓	
Reject <code>Ingress</code> , <code>Gateway</code> , <code>HTTPRoute</code> ; <code>LoadBalancer</code> only internal, in the zone's ILB subnet, <code>externalTrafficPolicy: Local</code> ; no <code>NodePort</code>			✓
Namespace has <code>platform/stateful-exception</code>			✓

3.1 Cluster types: stateless and stateful

Requirements: [R1](#), [R5](#), [R17](#), [R18](#), [R19](#), [R20](#), [R21](#), [R23](#), [R24](#).



Every environment has two kinds of clusters. The split follows one rule: **a cluster that holds no data can be taken out of traffic, upgraded or even rebuilt at any time; a cluster that holds data cannot, so it is made zone-redundant and changed as rarely as possible.**

Where does a workload run

Data does not belong in Kubernetes by default. Every workload is placed by going down this list and stopping at the first option that fits:

- 1. **Stateless cluster** – all application code. Its state lives in the options below.
- 2. **Azure PaaS service in the backend network** (PostgreSQL, MySQL, Managed Redis, Cosmos DB for MongoDB, Event Hubs with Kafka, Blob Storage, ...) reached through private endpoints from the stateless clusters. Zone redundancy, backups, patching and upgrades are Microsoft’s job; the immutable copies in the paired region are the platform’s (section 6.3). Services with no equivalent in the other clouds (Azure SQL, Service Bus, Cosmos DB’s NoSQL API) are allowed only as a documented cloud-pinning exception at onboarding (section 1.3); where this document names them, it describes the Azure implementation of such an exception.
- 3. **Stateful cluster – last option**, only when
 - no Azure PaaS service provides the capability (e.g. a product that is only shipped as a container or operator), or
 - a special use case requires it (e.g. a licence, a latency requirement to data that only the cluster can meet, a protocol the PaaS service does not offer).

A workload in the stateful cluster needs a documented exception (why 1 and 2 do not fit, owner, exit plan) approved by the platform team, and the stateful cluster itself is only built when the first exception is approved (optional); the exception is the label `platform/stateful-exception=<ticket>` on its namespace, required by a policy. The exceptions are reviewed when new PaaS services become available. The fewer workloads the stateful cluster runs, the smaller its blast radius and upgrade risk.

Comparison

	Stateless (aks- <code><env></code> -sl-az<N>)	Stateful (aks- <code><env></code> -sf)
Runs	Frontends, APIs, workers – anything that can be killed and recreated; state in Azure PaaS	Only approved exceptions: workloads that own data on disks (StatefulSets, operators) for which no PaaS service fits
Count per environment	Per exposure (int , ext) one per availability zone: sl-az1 , sl-az2 in dev and acc; sl-az1 , sl-az2 , sl-az3 in prd (section 3.2)	Zero or one per exposure – built only when the first approved workload of that exposure needs it

	Stateless (aks- <code><env>-sl-az<N></code>)	Stateful (aks- <code><env>-sf</code>)
Availability zones	All nodes of a cluster in its AZ (every pool <code>--zones <N></code>); the cluster is the unit of failure	Every isolation zone has one node pool per AZ (<code>intfiz1</code> , <code>intfiz2</code> , <code>intfiz3</code> , ...); system pool spread over AZ 1–3
Nodes	Classic node pools per zone, capacity type and architecture with the cluster autoscaler, from zero (section 2.3)	Classic AKS node pools with a fixed VM size and node count
Network	Frontend spoke <code>vnet-<code><env>-<code><exposure>-sl-az<N></code></code></code> in the exposure's subscription, peered to the exposure's hub	Backend spoke <code>vnet-<code><env>-<code><exposure>-sf</code></code></code> in the exposure's subscription, peered to the exposure's hub, together with the PaaS private endpoints
Internet	Egress only via Azure Firewall FQDN allow-list (outbound type <code>userDefinedRouting</code>); inbound only via the traffic layer	None. Network isolated cluster (outbound type <code>none</code> , bootstrap artifacts from the private ACR cache), no public IPs, UDR <code>0.0.0.0/0</code> → Firewall which denies all Internet for backend prefixes
CNI	Azure CNI Pod Subnet with dynamic IP allocation, powered by Cilium - every pod with its zone's address (section 2.1)	The same
Incoming HTTP(S)	Gateway API , the implementation chosen per cluster (Traefik), one <code>Gateway</code> per isolation zone, HTTPS only; the Kubernetes Ingress API is not used	No Gateway API implementation and no Ingress. Applications are published with internal <code>LoadBalancer</code> <code>Services</code> (L4)
TLS	Terminated by the zone's gateway with Let's Encrypt certificates for internal DNS names; applications serve plain HTTP inside the cluster	Terminated by the application itself – a hard onboarding requirement
Node-to-node encryption	Azure Virtual Network encryption – no service mesh, no mTLS	Application TLS (VNet encryption may be enabled as defence in depth but is not relied on)
Kubernetes version	Standard support, two speeds: <code>sl-az1</code> on the second latest GA minor (N-1), <code>sl-az2</code> (and <code>sl-az3</code>) one minor behind (N-2), never older than 1.36 (update policy)	Long Term Support (<code>--tier premium --k8s-support-plan AKSLongTermSupport</code>): created on the second latest GA minor (N-1), 1.37 at the earliest (the first minor whose claims report when they were last used), then kept on that minor until 6 months before its LTS ends
Tier	Standard	Premium (required for LTS; billed at the Premium rate only once the minor has left community support - AKS, April 2026)
Scaling	≥ 2 replicas per app, HPA / KEDA on load, VPA resizes pods in place, the cluster autoscaler adds and removes nodes; each cluster sized to carry 100 % of the load alone (the zone pools' maximum)	≥ 3 replicas per StatefulSet, one per AZ; no autoscaling – fixed node count per pool, sized when an exception is onboarded and changed as a planned IaC change

	Stateless (aks-<env>-sl-az<N>)	Stateful (aks-<env>-sf)
Storage	None – admission policy rejects PersistentVolumeClaims; ephemeral OS disks	Azure Disk Premium_ZRS / StandardSSD_ZRS , Azure Files ZRS (over NFS with encryption in transit, mounted with the namespace's workload identity instead of a storage account key - both GA in the Azure Files CSI driver since August 2026; the alternatives are weighed per exception), through per-zone StorageClasses encrypted with the zone's own key (section 4.2); prefer PaaS for databases. Every claim carries Kubernetes' Unused condition (1.37): the storage agent proposes what to do with volumes no pod has used for longer than the zone's retention, and nothing deletes them without the owning team
Upgrade model	Drain from traffic, upgrade or rebuild, return (picture 3.5)	In place, one AZ at a time, PDB-protected (picture 3.5)

Isolation zones are kept in both cluster types. Each stateless and stateful cluster has the nodes, subnets, Key Vaults and policies of the zones of its exposure – `int-fi` and `int-se` in an internal cluster, `ext-fi` and `ext-se` in an external one – exactly as in pictures 2.1–2.6; the data in the stateful cluster is what needs the country separation most. Its applications are reachable only from the same zone of the stateless clusters.

Internal and external: separate subscriptions and clusters

Internal and external workloads never share a cluster or a subscription (the security team's decision). Countries are separated one level lower, inside a cluster, by node pool and network:

Separation	Internal ↔ external (exposure)	Country ↔ country (fi , se , ...)
Boundary	Subscription and cluster: <code>sub-aks-<env>-int</code> and <code>sub-aks-<env>-ext</code> , each with its own cells and optional stateful cluster	Isolation zone inside the clusters of one exposure
Control plane, system pool, admission, Flux, the gateway control plane	Separate	Shared by the exposure's zones
Nodes and network	Separate spoke VNets peered to separate hubs with an Azure Firewall each , no peering and no route between them	Own node pools, own address space, subnets, NSGs and route tables
Key Vaults	Separate platform vaults (<code>kv-<env>-int-platform</code> , <code>kv-<env>-ext-platform</code> : certificates, the clusters' keys) and zone vaults	Own zone vault (<code>kv-<env>-<zone></code>)
Azure RBAC, Azure Policy, budget, quota	Per subscription	Per zone where it matters (namespaces, ABAC, policy mutations)

Separation	Internal ↔ external (exposure)	Country ↔ country (fi , se , ...)
ACR	One per exposure subscription, fed by the same promotion pipeline	Shared
Entry points	External: Traffic Manager and public load balancers, only in the external subscription. Internal: hub Private DNS and internal load balancers, only in the internal one	One profile / Private DNS zone per zone

So an external cluster – the one the Internet reaches – has no internal zone, no internal identity, no role on an internal vault and no route to an internal subnet at all: its hub has neither a peering to the internal hub nor a gateway to the corporate network; a compromise of its control plane, nodes, subscription or Firewall does not reach the internal workloads. Traffic between the two exposures is like traffic between two zones (R4): denied. An internal application that must call an external one goes to its public entry point like any Internet client (an FQDN rule on the internal Firewall); an external application has no path to an internal one, and giving it one is a design change, not a Firewall rule. What the two keep in common is the source – the same IaC modules with the exposure as a parameter, the same fleet repository with a directory per cluster – and the management VNet, Entra ID, GitHub and the environment's Grafana Cloud stack.

Incoming traffic: from outside and from inside

Clients never connect to a pod or node directly, and the stateful cluster is never reached from outside the platform. The two kinds of zones use the cells differently, because their availability requirements differ:

- **External zones (ext - *):** every application runs in every cell, active-active. A **traffic layer** (Azure Traffic Manager, the cell router) outside the cells answers each client's DNS query with one cell, chosen by the client's network, and leaves a failed or drained cell out of its answers.
- **Internal zones (int - *):** every application runs in one cell, its home cell. The application team chooses the cell; its DNS name points straight to the zone's gateway in that cell, maintained by ExternalDNS. A **migration workflow** moves an application to another cell without downtime – on request, around cluster upgrades, and after a cell failure. A cell failure therefore costs the internal applications of that cell a short outage until they are migrated (recovery time target *TBD*, minutes) – accepted because internal applications have lower availability requirements than external ones. There is no WAF and no load balancer in front of internal zones.

Source	Entry point	Path to the cells
Internet (external zones ext - *)	Azure Traffic Manager: public DNS name of the application → CNAME to the zone's Traffic Manager profile, which answers with one cell	The public load balancer of the zone's gateway in every cell: one public IP per external zone and cell, HTTPS (443) only
Corporate network (internal zones int - *)	The zone's gateway in the application's home cell: <app>.int-fi.prd.example.com resolves in the hub Private DNS to the gateway's internal load balancer IP of zone int-fi in that cell	On-premises → ExpressRoute/VPN → the internal hub's Firewall → the gateway's internal load balancer of the zone in the home cell

Source	Entry point	Path to the cells
Other Azure workloads (other spokes)	Same as the corporate network	Same as above; a Firewall rule per source
Applications in the same zone and cell	The application's DNS name, like any client; never its Service or pods, which only its own namespace and the zone's gateway reach (R22)	The hub's Firewall (a rule in the caller's rule file) → the zone's gateway in the cell; never through the traffic layer or to another cell
Applications in the same internal zone, other cell	The application's DNS name, like any internal client	The internal hub's Firewall → the zone's gateway in the other cell (one Firewall rule per zone: zone prefix → zone prefix)
Applications in another zone	Never directly (R4). If a business flow needs it, the caller is treated like any other client: the application's DNS name, resolved to the target zone's gateway, with an application rule in the caller's rule file (section 2.1)	The hub's Firewall (which proxies the flow from its own address) → the target zone's gateway in the home cell

No component in front of a cell keeps state about the client: sessions and caches belong in Azure PaaS (e.g. Azure Cache for Redis) or in the token, so a client can be moved to another cell at any time without losing anything. What the cell choice controls is **which application version** a client sees while a release moves through the cells one by one ([releases cell by cell](#)).

External zones: active-active with DNS affinity

The traffic layer is DNS: the client connects straight to the cell Traffic Manager names, and **its TLS session ends at the zone's gateway** there - nothing in between decrypts it. Every client is mapped to one cell **by its network**, so it keeps using the same cell across requests, TLS sessions and reconnects – also clients that do not support cookies (API clients, mobile apps, other systems):

- **The affinity exists for consistent versions, not for state.** It makes sure that a client sees **one application version at a time** during a release.
- **A failed or drained cell is left out of the answers**, and its clients move to another cell as soon as their cached DNS answer expires (the record's TTL, 30 s); they come back when it returns.
- **All cells always prove that they work.** A cluster that only receives traffic in a failure is a cold standby with cold caches and untested capacity. With active-active, a cell failure only affects the clients of that cell (half or a third).
- **Long-lived connections** (WebSockets, Server-Sent Events, gRPC streaming) stay on one cell for the life of the connection – DNS is only asked again when the client reconnects. Applications that use them must reconnect with back-off, because draining a cell closes them after the drain timeout.

How Traffic Manager maps a client to a cell ([Traffic Manager routing methods](#)): per zone and environment one **parent** profile with the *Subnet* routing method maps the address space of the clients' resolvers, in blocks assigned by measured traffic, to one **child** profile per cell with the *Priority* method, which lists the cell's own gateway first and the other cells after it in a fixed order, so the clients of a failed cell all move to the same cell. The health probe is `HTTPS GET /healthz` of an application of the zone through each cell's gateway every 30 s, a cell is left out after 2

failures, and the TTL is 30 s, so a drain or a failover reaches most clients within a minute; a cell is taken out by disabling its endpoints and returned a quarter of its blocks at a time. The platform answers with A records only.

Limits of DNS affinity, accepted in this design: clients behind a large shared resolver land on the same cell, so the load is less even than per-request routing (the cells are sized $N+1$, the blocks are rebalanced outside releases, and an alert fires when a cell carries more than its $N+1$ share for an hour); a resolver that ignores the TTL keeps using a drained cell longer, which the drain timeout allows for; a mobile client that changes networks may change cells and meet the other version, which the [compatibility rule](#) covers. **There is no WAF in the traffic layer** and none in the gateway: the design launches with DDoS protection, the 443-only NSG, the gateway's TLS termination, Cilium ingress rules and the applications' own input validation as the controls.

Real client IP: the zone's gateway `Service` uses `externalTrafficPolicy: Local` (set through the Gateway's `parametersRef` `ConfigMap`), so the gateway's Envoy sees the client's own address; there is no proxy whose `X-Forwarded-For` it would have to trust.

Every external application also has a **per-cell host name** (e.g. `<app>-<cluster>.ext-fi.prd.example.com`, an A record of that cell's gateway kept by ExternalDNS). The release and upgrade pipelines run their smoke and synthetic tests on it, through the cell's real TLS path, before the cell gets user traffic ([releases](#), [section 3.5](#)); the same names are the Traffic Manager endpoints.

The public addresses of the platform are the external zones' gateways - one Azure Standard public load balancer IP per external zone and cell, created by the gateway's `Service` in `<zone>-gateway`. Besides them only the platform gateway of a cell that takes Flux's GitHub webhook (one listener for the receiver, the platform's one HTTP/1.1 path); every cell's Flux also polls Git every minute. An internal cell has no public load balancer at all - one cannot answer behind the hub firewall. All public IPs are protected by one Azure DDoS Network Protection plan for the whole tenant, attached to every cell VNet. The NSGs allow TCP 443 from the Internet on the system subnet (where every zone's gateway pods run) and on an external zone's node subnet, and nothing else from outside the VNet. This traffic does not pass the hub Firewall; the NSG, the zone's gateway and the application's Cilium ingress (only from the zone's gateway) are its controls.

Internal zones: one home cell per application, moved by migration

Placement. Every internal application has exactly one **home cell** per environment; the default is the slow-speed cell `sl-az2`, which runs the more proven Kubernetes version ([update policy](#)). The team chooses it in Git (`placement.yaml`, e.g. `cell: sl-az2`), which every cell reads from `main` through a Flux Kustomization of its own (path `placement/`, interval 1 min) – environment state, not part of the release train ([section 3.3](#)). It creates the application's Flux Kustomization only in its home cell – and during a migration also in the target cell; the application's manifests themselves still come from the cell's branch.

DNS by ExternalDNS. Each internal zone has a Private DNS zone (`int-fi.prd.example.com`) in the exposure's environment resource group, linked to every cell VNet and to the hub, so the hub firewall's DNS proxy - and the corporate network behind it - resolves it. Every cell runs **ExternalDNS** in the platform namespace `external-dns`, one instance per exposure of names:

- `external-dns-int` : provider `azure-private-dns`, a workload identity with *Private DNS Zone Contributor* on the internal zones' Private DNS zones only, `management.azure.com` on the Firewall allow-list; source

gateway-httproute limited to Gateways labelled `platform/exposure=int`, so the A record of each hostname of an `HTTPRoute` points to the zone's gateway load balancer in that cell;

- `external-dns-ext` (external cells): provider `azure` on the public zone, Gateways labelled `platform/exposure=ext`, the per-cell host names and the global names that Traffic Manager fronts;
- **the home cell:** every internal route carries the annotation `platform/home-cell: <cluster>`, and each cell's `external-dns-int` publishes only the routes that name it (`--annotation-filter=platform/home-cell=<cluster>`), so exactly one cell publishes each name. The value comes from the team's placement through Flux substitution, the same in every cell; Azure Policy rejects the annotation set by hand in application manifests;
- one shared owner ID per environment with `upsert-only`, so the target cell of a migration can take over a record from a source cell that is gone; record TTL 60 s, sync interval 1 min.

Migration workflow – a pipeline in the runbooks repository; the same steps serve every reason to move:

1. **Deploy to the target:** the placement adds the target cell without `platform/dns-publish`; Flux deploys the application there (the autoscaler adds nodes if needed). Wait until its Kustomization is `Ready`.
2. **Test:** smoke tests against the target cell's gateway IP with the application's real host name (`curl --resolve`), over the real TLS path.
3. **Switch DNS:** the placement moves `platform/dns-publish` to the target cell. Its ExternalDNS overwrites the A record within a minute.
4. **Drain the source:** wait for the TTL and until the source cell's gateway sees no more requests for the application (or a maximum, e.g. 15 min, for clients that cache DNS too long); long-lived connections are closed and reconnect to the target.
5. **Remove from the source:** the placement drops the source cell; Flux prunes the application there.

Until step 5, the migration is rolled back by moving `platform/dns-publish` back. Planned migrations are zero-downtime. Teams start one by a pull request that changes their `placement.yaml`.

Cell failure. An alert on the cell's health (gateway health routes probed by Grafana Cloud synthetic monitoring, AKS and VM health) dispatches the **failover**: the migration workflow for all internal applications of the failed cell, with step 4 skipped and step 5 deferred until the cell is back (a safe action, [section 6.1](#)). Their recovery time is alert + Flux reconcile + node provisioning and pod start in the target + ExternalDNS sync + TTL. The failed-over applications stay in their new cell; the team or the platform moves them back by a planned migration.

Consequences:

- **TLS ends at the zone's gateway.** Internal clients connect to it directly with the zone's certificate ([section 4.1](#)); there is no WAF in front of internal zones.
- **Real client IP:** the gateway sees it directly, because its `Service` uses `externalTrafficPolicy: Local` and the internal hub's Firewall allows the flow with network rules (no SNAT to private destinations).
- **One version at a time.** An internal application runs in one cell, so its clients see one version, apart from the few minutes of a rolling update, which the [compatibility rule](#) covers anyway.
- **Capacity:** every cell must still be able to take all internal applications of the other cell(s) – for upgrades and failures – so the N+1 sizing does not change. The zone pools' maximum is the same in every cell.

- **Firewall:** corporate network and the pipeline agents → the zone's prefixes (its node and load balancer subnets in every cell, one IP group per zone) on TCP 443, so a migration never needs a Firewall change.

VNets and address plan

One spoke VNet per cluster, one address space per cell, one slot per isolation zone - the slot layout of [section 2.1](#), stated per cell in the environment's shared configuration: an address space for nodes and load balancers (/24 slots by the zone's index) and a pod address space (a /16 of the RFC 6598 range 100.64.0.0/10 with a /20 per slot index; every node pool needs a pod subnet, the system pool's is snet-system-pods). An example cell with zone int-se at index 2:

Cell	Address space	sn et- sys tem	A zone's slot	snet - <zone>- nodes	sne t- <z on e>- ilb	Pod subnets
dev- int- statel ess- az1	10.100.0.0/16 (snet-system 10.100.0.0/24 , int-se 10.100.2.0/24 , its load balancers 10.100.2.128/27)	/2 4	/24	/25 - about 120 nodes	/2 7	Pod address space 100.64.0.0/16 , a /20 per slot index: snet-system-pods 100.64.0.0/20 , snet-int-fi-pods 100.64.16.0/20 , snet-int-se-pods 100.64.32.0/20

Because a zone has the same index in every cell, the zone is two prefixes per cell, its node slot and its pod slot, and the Firewall's IP group for the zone lists them (int-se in the example: 10.100.2.0/24 , 100.64.32.0/20): a rule written for a zone does not change when a cell is added or rebuilt, and a rebuilt cell takes its own address spaces again (drained first, so the old VNet is gone before the new one exists). The Firewall's SNAT private ranges include 100.64.0.0/10 with the RFC 1918 ranges, so no pod address ever leaves the landing zone.

The private endpoints are not in the cells. The zone Key Vaults, the platform Key Vault, the registry (prd), the Terraform state and the AI model endpoint have their private endpoints in one subnet (snet-shared-pe) of the environment's shared network (vnet-aks-<env>), registered in the hub's privatelink.* zones; a cell reaches them through the hub Firewall, whose rule for the vaults takes the cells' system subnets - where the External Secrets Operators run - as its source. Zone separation of vault access is RBAC and ABAC plus that source group, not a subnet per zone. The same shared network holds the stateful cluster when one is built.

Why not one common VNet for all cells?

	One VNet per cell (chosen)	One VNet per environment	One frontend VNet for all stateless clusters + backend VNet
Frontend/backend separation (R19)	Separate VNets, only reachable through the exposure's hub Firewall	Lost – the Internet-less backend shares a VNet with the frontend, separated only by NSG/UDR	Kept
Change one cluster at a time (R26)	Yes – a VNet change (address space, DNS servers, VNet encryption, peering) affects one cluster, and a stateless cluster can be rebuilt with a fresh VNet	No – every VNet change hits all clusters at once	No for the stateless clusters – a VNet change hits all of them at once
VNet settings per cluster type	VNet encryption on the cells only, other DNS/egress settings for the backend	One setting for all	Yes
One prefix per zone	One per zone <i>and cell</i> , with the same slot index everywhere; the Firewall's IP group per zone joins them	Yes	Yes
Extra cost	One hub peering per cell; a new zone is one IP group entry per cell	–	–

Sizing. `/24` slots give every zone about 120 nodes, enough for a small environment; a production cell takes larger slots (a `/16` cut into sixteen gives each slot a `/20`), which is a per-cell value, so cells of different sizes coexist and a cell is resized by a rebuild. The corporate IPAM therefore reserves **one prefix per cell**, not one per zone across cells. If the corporate plan wants one summarisable prefix per country or zone instead, the same indices give such a plan (a `/12` per environment, a `/16` per country, a `/17` per zone) without changing the cells.

IPv6: the platform is IPv4 only and plans no IPv6.

Adding a country online

A new country is two new zones, one per exposure. A zone is a slot of every cell's address space, so nothing is re-addressed, no peering changes and no route is added; it is added in the order `dev` → `acc` → `prd`, and the existing zones only get new Firewall rules:

1. **Declare the zone** in the environment's shared configuration (its exposure and its index, a free slot in every cell of the exposure, which the IaC checks), its Key Vault name and, for an internal zone, its Private DNS zone; the environment module creates the vault with its private endpoint and the DNS zone.
2. **Subnets and NSGs** (`snet-<zone>-nodes` , `snet-<zone>-ilb`) appear in every cell of the exposure from the slot, and `snet-<zone>-pods` from the pod address space's slot), and the cluster identity gets *Network Contributor* on them and on the zone's route table (`rt-<cell>-<zone>` : the default route, the cell's address spaces to the Firewall - its own load balancer subnet included -, the zone's own node subnet and pod slot kept in the VNet); the other tables, the system subnets' included, already route its slot, because they route

the whole address space, so none of them changes. The Firewall's Kubernetes-flow rules name the zone's IP group.

3. **Firewall:** the zone's IP group (its node and pod slots in every cell), its allow-list and its application rule collections; the existing rules do not change.
4. **Certificates and identities:** the zone's gateway certificate in the platform Key Vault (the renewal job), the zone's External Secrets reader identities and role assignments; in the stateful cluster also the zone's persistent volume keys, disk encryption sets, storage accounts and StorageClasses ([section 4.2](#)).
5. **Nodes:** in the stateless cells, the zone's four pools (from zero, [section 2.3](#)); in the stateful cluster, node pools `<zone>z1 – z3` (fixed node count). Neither restarts existing nodes.
6. **Policies and Git:** the zone's Azure Policy mutation and validation definitions (one per zone), and the zone's `Gateway`, External Secrets Operator, Cilium baseline and namespaces through Flux (the platform repository's `infrastructure/zones`, applied once per zone as `zone-<zone>`) and the stateful pipeline.
7. **Entry points:** ExternalDNS's role on the internal zone's Private DNS zone; the external zone's Traffic Manager profiles and the public DNS CNAMEs of its applications. Existing zones' entry points do not change.

A country can be switched off in a cell without giving up its slot: its zone stays defined (vault, DNS zone, certificate) and runs no nodes there.

Flows between frontend and backend

Source	Destination	Port	Rule
Stateless zone X pods (their own addresses)	Stateful zone X application internal LBs	Application TLS port (e.g. TCP 443)	Same isolation zone only (<code>int-fi</code> → <code>int-fi</code>), via Firewall; TLS terminated by the application
Stateless zone X pods	Zone X PaaS private endpoints	TCP 443, 1433, 5432, 5671	Same isolation zone only, via Firewall; TLS enforced by the PaaS service
All nodes of the exposure	The exposure's ACR private endpoint	TCP 443	Images, Helm charts
Internal clusters	External clusters, and the reverse	–	Blocked: no peering and no route between the two hubs. An internal caller reaches an external application through its public entry point like any Internet client; the reverse has no path
Stateful cluster	Frontend networks	–	Blocked – the backend never initiates connections to the frontend
Stateful cluster	Internet	–	Blocked – no route, no outbound IP, Firewall deny-all
Management VNet pipeline agents	Stateful API server (Private Link)	TCP 443	Deployments to the stateful cluster (section 3.3)

Source	Destination	Port	Rule
All nodes of the environment (Grafana Alloy)	Grafana Cloud private endpoint in the hub	TCP 443	Telemetry through Azure Private Link, so the stateful cluster still has no Internet (section 6.2)
Stateful nodes / pods	Microsoft Entra ID (AzureActiveDirectory service tag)	TCP 443	Exception, needed for workload identity token exchange because Entra ID has no Private Link for sign-in
Stateful nodes (Azure Policy add-on)	data.policy.core.windows.net , store.policy.core.windows.net , dc.services.visualstudio.com	TCP 443	Exception, Firewall application rules for these FQDNs only; the add-on has no Private Link (section 2.7)

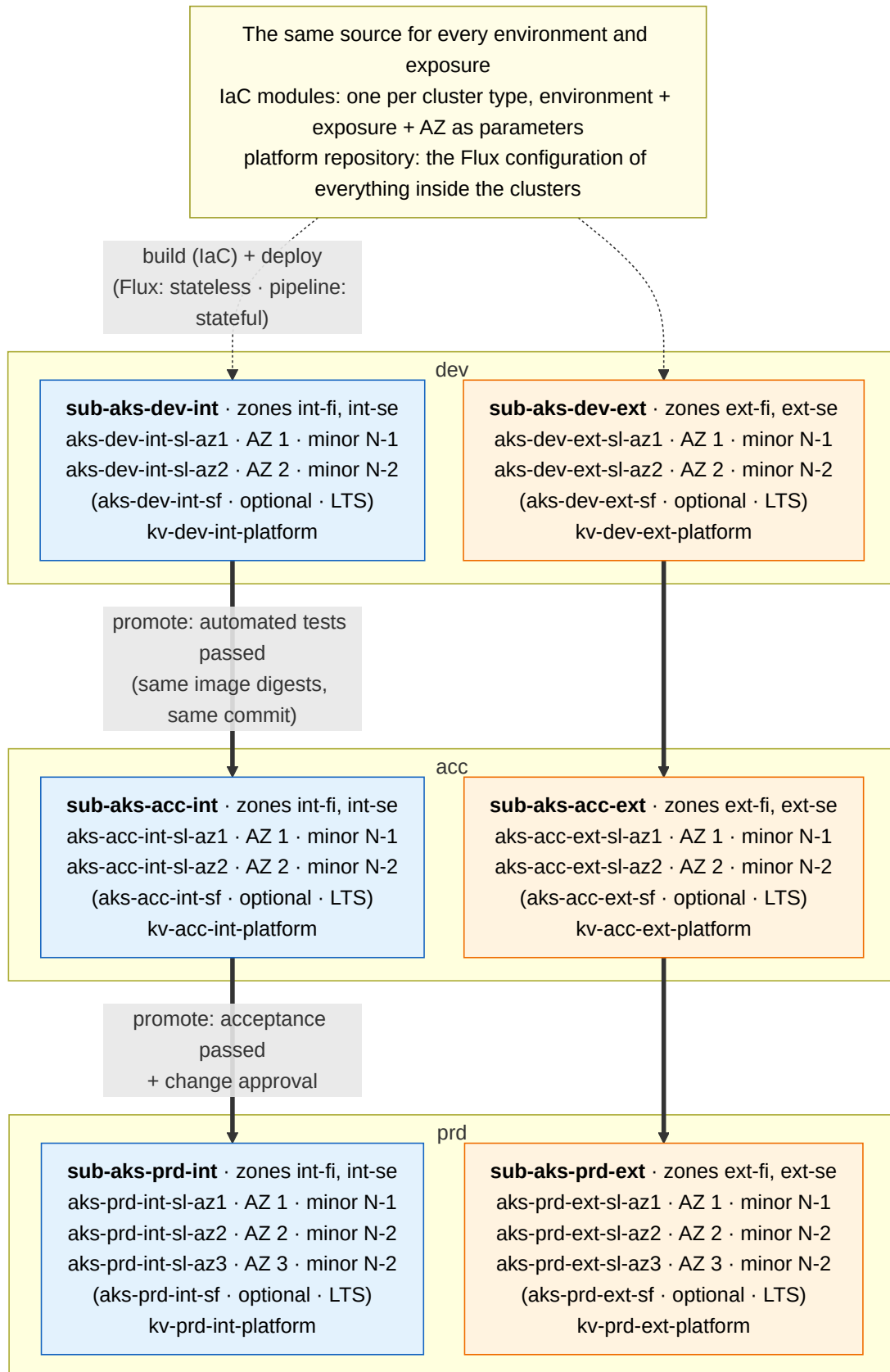
Telemetry goes to Grafana Cloud through Private Link; the AKS monitoring add-on is not enabled ([section 6.2](#)). Other AKS add-ons that need public Azure endpoints are not enabled in the stateful cluster; the Azure Policy add-on is the one accepted exception ([section 2.7](#)). These two destinations (Entra ID, Azure Policy) are the whole list of public endpoints the stateful cluster may reach; the Firewall application rules are the inventory, and the isolation tests prove from inside the cluster that nothing else answers.

Shared tier failure domains

Cells isolate compute and releases. The shared tier - Traffic Manager, the hub Firewalls, Private DNS, Entra ID, the Azure Policy endpoints, Key Vault, the registry, the PaaS data services, the stateful cluster, Grafana Cloud and GitHub - is not split into cells, so each of its parts is a failure domain of its own: zone-redundant where Azure offers it, and with a known effect when it is lost anyway. What each failure does to running traffic, and how it recovers, is the fault table of [section 0.3](#), which the resilience tests inject one by one.

3.2 Environments (fourteen to twenty clusters)

Requirements: [R1](#), [R14](#), [R17](#), [R18](#), [R23](#), [R25](#), [R34](#).



Every environment has **two subscriptions, one per exposure** (section 3.1), each with its own cells and optional stateful cluster: **fourteen clusters**, twenty with every stateful cluster. Where this document says `aks-<env>-sl-az<N>` or `aks-<env>-sf` without an exposure, it means the cell or stateful cluster of either exposure: everything per cell is the same in both. dev and acc must have the *same topology* as prd per exposure (at least two stateless clusters, and a three-AZ stateful cluster if prd has one), otherwise the upgrade procedures cannot be rehearsed there; they can use lower zone pool limits and smaller stateful node pools.

Why three cells in prd: with two cells an AZ loss *while* the other cell is drained for an upgrade leaves nothing; with three, each carries a third and takes half ($N+1 = 50\%$ headroom), and the third cell costs only its fixed part. Dev and acc rehearse the same procedures with two cells. One service differs by environment on purpose: prd's container registry is Premium with a private endpoint; dev's and acc's are Basic on their public endpoint, Entra ID only.

Previews and sandboxes inside dev

dev is also where a team tests before `main` (R34): a preview namespace per pull request next to the application's dev namespace, and a long-lived sandbox namespace per team, both deployed from a branch outside the cell branches and the release train (section 3.3), under every policy, quota and Firewall rule of dev, labelled and charged to the team, and removed with the pull request (section 1.5). There is no sandbox cluster and nothing of the kind in acc or prd: dev's cells carry the previews on their zone pools, which scale from zero, and a quota per team caps how many run at once, so previews change the variable cost of dev and nothing else.

Cost and capacity model

The cost of an environment has three parts. The numbers are measured in acceptance at idle and at prd peak (section 0.3) and recorded per environment and isolation zone in Grafana Cloud; this page only fixes the structure.

Part	What is in it	Scales with	Where it is capped
Fixed per cell	System pool (two nodes), the gateway control plane where App Routing runs and the zones' gateway pods, the zone gateways' load balancers and public IPs, private endpoints (zone vaults, platform vault, ACR), hub peering, AKS Standard tier	Number of cells and zones	IaC
Fixed per environment	Hub share (the exposure's hub and Firewall; Private DNS, Bastion, the DDoS plan), Key Vaults, the container registry (Basic in dev and acc, Premium with its replica in prd), Traffic Manager profiles, backup vaults and storage, Grafana Cloud stack, Log Analytics, the stateful cluster's node pools and control plane if built	Number of environments, zones and stateful workloads	IaC
Variable	Zone nodes (on-demand and spot), PaaS data services, Grafana Cloud ingest and retention, Firewall data processing	Load, right-sizing, spot share, telemetry volume	<code>limits.yaml</code> : zone pool limits, <code>maxSpotShare</code> , retention; budgets and quota alerts to the capacity agent

Capacity rules that the cost model must leave room for, all of them checked in acceptance:

- **N+1 per environment:** with `k` cells, every cell's on-demand pool limits, vCPU quota and node subnets allow it to carry $1/(k-1)$ of the environment on top of its own share (100 % extra with two cells, 50 % with

three). Spot capacity is not counted.

- **Quota headroom** for the surge of a cell upgrade (a surge node per pool before an old one goes) and of the stateful cluster's `maxSurge: 1` per pool; the capacity agent watches quota use against the sum of the pool limits and files the raise before it is needed.
- **Reservations and savings plans** cover only the steady baseline (system pools, stateful pools, the on-demand share that always runs), proposed by the cost agent as purchases for the platform owner.
- **Who pays:** OpenCost splits each cell's cost by namespace, and so by the namespaces' `platform/team` and `platform/cost-center` labels (section 2.6); a monthly budget per environment subscription alerts its owners at 80 % and 100 % of actual cost and at a 100 % forecast.

The stateful cluster is optional. It is not built until the first workload has an approved stateful-cluster exception ([where does a workload run](#)); until then every environment runs only the stateless clusters and Azure PaaS. The backend network itself exists from day one, because it holds the PaaS private endpoints and the ACR. Everything the stateful cluster needs later is already prepared, so adding it is an ordinary IaC change with no impact on the running platform: its `/18` per zone is reserved in the address plan, the stateful cluster module and pipeline are kept with the IaC, and the Firewall rules for “stateless zone X → stateful zone X” are only created with it. The same exception review also lets the platform team remove the stateful cluster again when its last workload moves to a PaaS service.

- **Clusters are cattle, built by IaC.** An environment module, applied once per environment *and exposure* in that exposure's subscription, creates what all clusters of the exposure share and what must survive a cluster rebuild: the Key Vaults per zone (with the application secrets), the platform Key Vault (with the environment's certificates and their role assignments), the application managed identities, the ACR and the Azure Policy assignments – and, in the backup subscription, the zones' immutable backup vaults and backup storage accounts in the paired region (section 6.3). Two cluster modules (stateless, stateful) with `env` and `az` as parameters create the cluster's customer-managed keys in the platform Key Vault, its control plane identity and disk encryption set (section 4.2), the VNet, cluster, system pool and zone node pools, private endpoints to the shared zone and platform Key Vaults, the federated credentials and – in stateless clusters – the Flux bootstrap. The IaC also hands Flux a `cluster-settings` ConfigMap in `flux-system` for post-build substitution - `cluster_name`, `cluster_exposure`, the vault URLs and client IDs per namespace, the DNS zones and their resource group, `gateway_implementation` and `gateway_class_name_<gateway>`, the registry's login server - and the stateful pipeline sets the same variables itself.
- **Versions are the one intended difference.** Environments and cells run the same configuration but not always the same Kubernetes version or node image: patches move through dev → acc → prd with a delay, and `sl-az1` and `sl-az2` run different minors by design ([update policy](#)).
- **Everything inside a cluster comes from Git** – via Flux in the stateless clusters and via the deployment pipeline in the stateful cluster; no manual `kubectl apply`. Human access in acc and prd is read-only Kubernetes RBAC plus break-glass via PIM.
- **Every environment has its own ACR per exposure** (in the exposure's backend spoke); the promotion pipeline imports the exact image digests from the previous environment's ACR (`az acr import`) into the ACR of each exposure that runs the image, so prd never pulls from the Internet and no cluster pulls across the exposures. CI signs every image it builds; promotion first verifies the signature, then copies it with the

image (an OCI referrer, for example with `oras cp -r`). An unsigned image therefore never reaches the acc or prd registry ([section 2.7](#)).

3.3 Keeping clusters in sync: Flux for stateless, pipelines for stateful

Requirements: [R25](#), [R26](#), [R34](#).

The **platform repository** holds what every cluster runs, **one repository per application team** that team's applications; the cluster IaC hands both to Flux:

```
<platform repository>/
├─ clusters/<env>-<exposure>-stateless-az<N>/ # entry point of one cell, e.g. prd-int-
stateless-az1
|   └─ cluster-settings.yaml # ConfigMap whose ${...} values the
cluster IaC fills in (below)
|   └─ infrastructure.yaml # Kustomizations: infra-controllers →
infra-configs → tenants, external-dns, observability, ...
|   └─ zones.yaml # zone-<zone> per enabled isolation zone
|   └─ tenants.yaml
├─ infrastructure/
|   └─ controllers/, configs/ # External Secrets Operator, Cilium
baseline, the platform namespaces
|   └─ zones/ # applied once per zone: Gateway, ESO,
certificate, Cilium policies; gateway-implementations/{traefik,approuting-istio}, firewall-
egress
|   └─ autoscaler-priority/ # the cluster autoscaler's priority
expander: spot pools first
|   └─ platform-gateway/, flux-webhook/ # only in a cell with a public entry point
that takes GitHub's webhooks
|   └─ external-dns/, flux-system-key-vault/, observability/, image-verification/,
storage/, namespace-access/{view,edit,admin}
├─ tenants/<namespace>/ # a namespace's SecretStore and key-vault
ServiceAccount (and its webhook Receiver)
└─ policy/egress-allow-lists.yaml # the zones' egress allow-lists, read by
the CI check

<team repository>/ # one repository per team
├─ apps/<app>/ # Deployment, Service, HTTPRoute, VPA -
cloud-neutral
├─ apps/<app>-home/, <app>-global/ # the home-cell route (internal zones) and
the Traffic Manager route (external zones)
└─ <cell directory>/ # what one cell syncs: image tag,
replicas, patches (stateless/, firewall-egress/, ...)
```

The stateful cluster's `infrastructure/stateful` (storage classes, operators, no Gateway API) and the applications' `stateful/<env>` directories are part of the same layout, added with the first approved stateful workload.

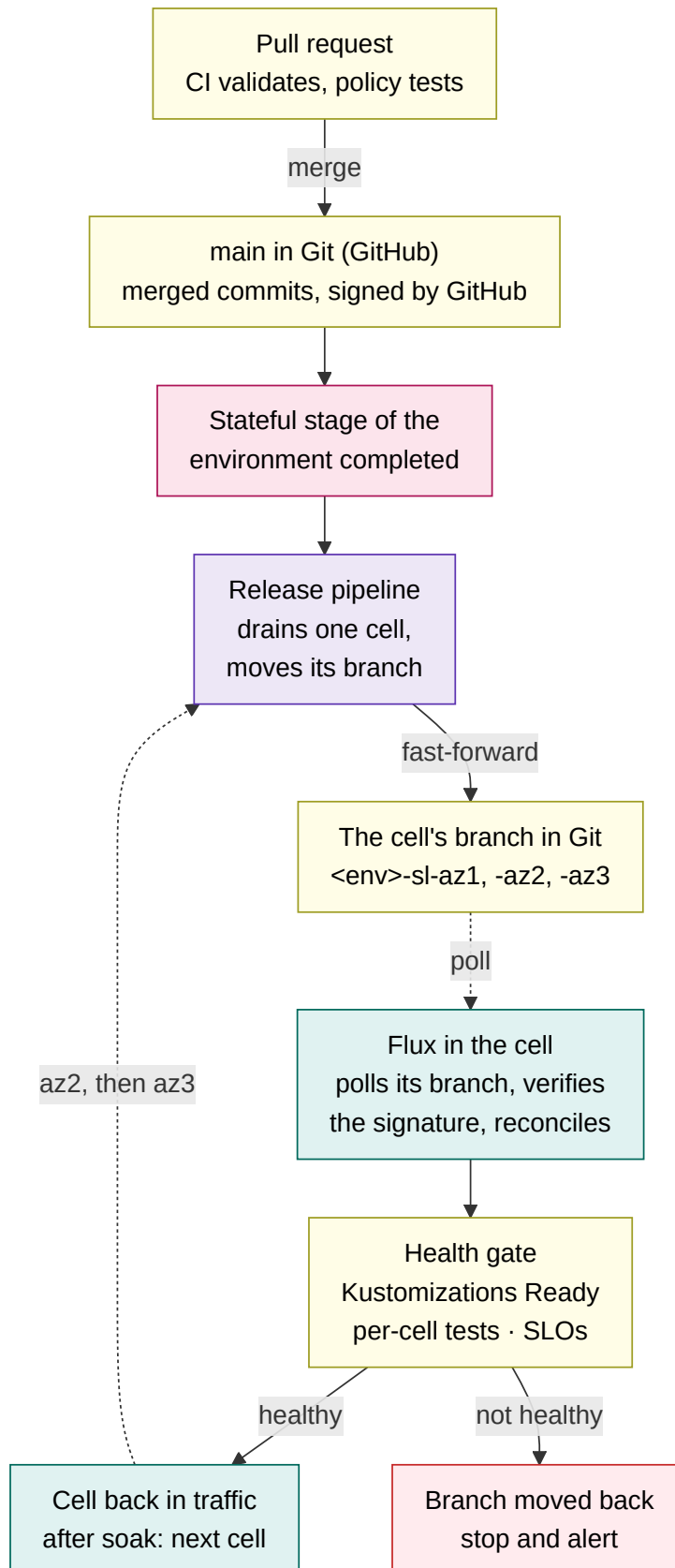
Why two mechanisms. The stateless clusters are many, identical and rebuilt often: a pull-based reconcile loop brings a fresh cluster to the desired state without anyone pushing to it, and corrects drift continuously. The stateful

cluster is a single, long-lived cluster whose changes must be **ordered, gated and observed** (StatefulSet rollouts, operator upgrades, schema steps) – a pipeline with explicit stages, approvals and a visible run history fits better, and the Internet-less cluster runs one component less.

Common: Git is the source, a pull request the only way in

- **Policies are not in the fleet repository:** they are Azure Policy definitions and assignments in the IaC repository ([section 2.7](#)); CI still tests the rendered fleet manifests against the same constraint templates.
- **The merged commit is the release.** CI validates every pull request (`kustomize build` , `kubeconform` against every Kubernetes minor that runs in the fleet – both stateless speeds and the stateful LTS version, see [update policy](#) – the egress names against the zone's allow-list and policy tests); the branches take no other push than the merge GitHub makes, which GitHub signs with its own key, and Flux and the stateful pipeline verify that signature before they apply a commit (`verify` on the `GitRepository`).
- **A change enters an environment by pull request.** What differs per environment is in the environment's cell directories: the image digest ([section 2.7](#)), replicas and patches. Promotion `dev` → `acc` → `prd` is a pull request that copies the tested values from the `dev-*` cell directories to the `acc-*` ones, then to `prd-*` (generated by the release agent, merged by the risk tiers of [section 6.1](#)); the images themselves are imported between the environments' registries by the promotion pipeline ([section 3.2](#)). What is shared – `apps/<app>/` , `infrastructure/` – needs no copy: it reaches an environment when the environment's cells take the commit (next heading).

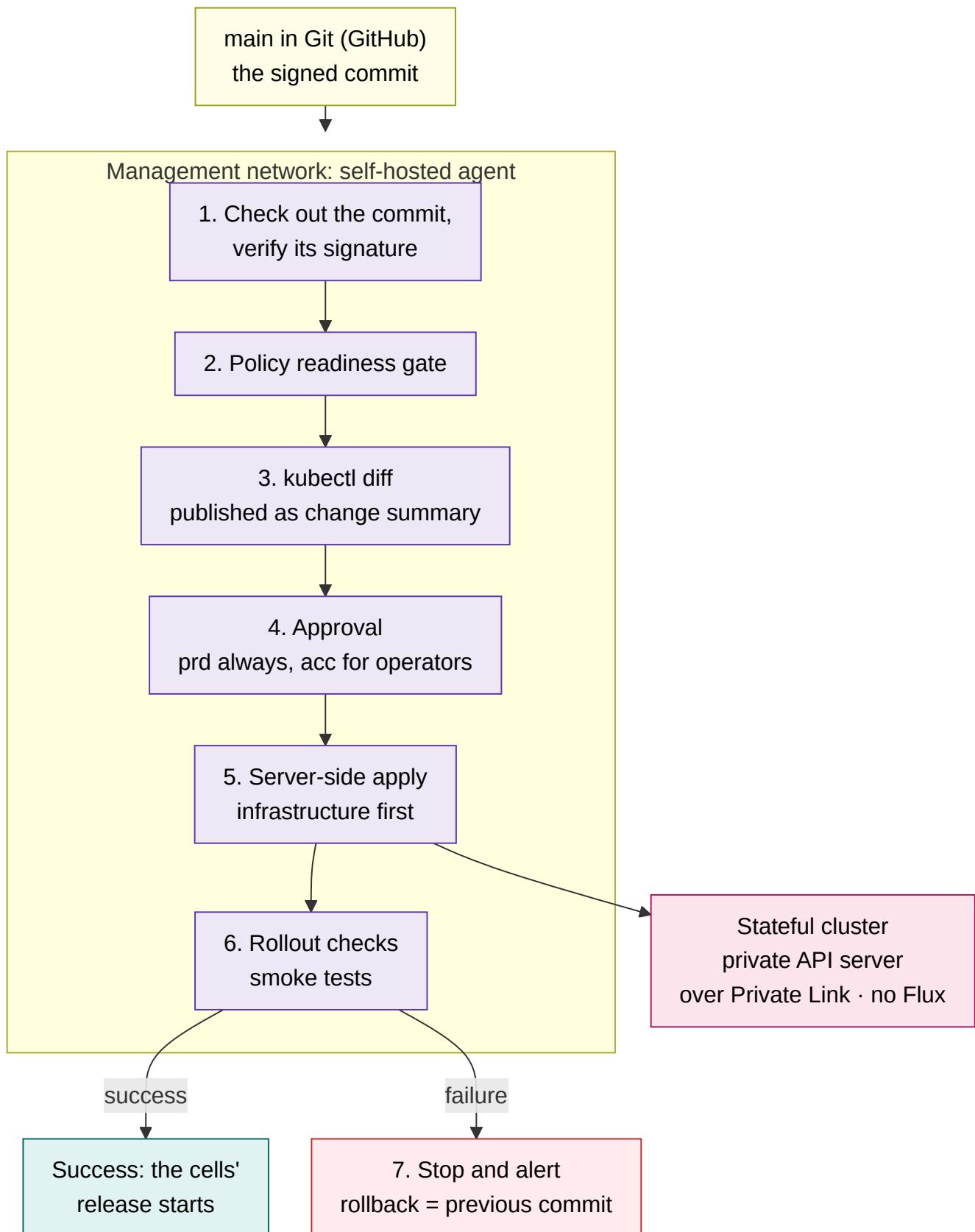
Stateless clusters: FluxCD



- **Flux installation:** the AKS GitOps extension (`microsoft.flux` , release train `Stable` , minor versions auto-upgraded, tolerating the system pool's taint), installed by the cluster IaC, so Microsoft manages its lifecycle like the Azure Policy add-on. The stateless clusters are not network isolated; the cluster extension FQDNs (`<region>.dp.kubernetesconfiguration.azure.com`) are on their Firewall allow-list. (The extension is **not supported on network isolated clusters**, which no longer matters because the stateful cluster does not run Flux.)

- **Sources are Flux configurations the cluster IaC defines** (`GitRepository` + `Kustomization`s, both managed by the extension). The cluster-scope configuration `platform` follows the platform repository (the cell's branch, path `./clusters/<cluster>`, `prune: true`, source read every minute in dev and every 5 minutes by default). Each managed namespace gets a **namespace-scoped configuration of its own** (the cluster IaC names the team's repository and path), applied as the namespace's `flux-applier` with the post-build variables `cluster_name`, `zone`, `zone_domain` and `home_cell`; the platform repository holds for a tenant only its `SecretStore` and `key-vault` `ServiceAccount` (`tenants/<ns>/`). Every source follows the cell's branch (below), verifies the commit signature and reads GitHub with a GitHub App's short-lived installation token (`provider: github`), never a stored key.
- **Order inside a cluster:** `dependsOn chains`, `prune: true` to remove anything deleted from Git. `infra-controllers` → `infra-configs` (Cilium baseline, platform namespaces) → `tenants`; per zone `node-pools-<zone>` and `infra-controllers` → `zone-<zone>` → `firewall-egress-<zone>`; `infra-controllers` → `flux-system-key-vault` → `observability`; `external-dns` → `external-dns-internal`; in a cell that takes GitHub's webhooks `platform-gateway` → `flux-webhook`. The applications' configurations are outside this chain: a route simply waits for its gateway. Nothing waits for a zone or a team, so one broken zone or tenant never blocks the others.
- **One branch per cell**, in the platform repository and in every team repository: `sl-az1` follows branch `<env>-sl-az1`, `sl-az2` follows `<env>-sl-az2`, and so on. The release pipeline's move is a fast-forward, and a ruleset lets only the pipeline's identity push these branches (**releases cell by cell**), so a bad change never reaches every cell at once.
- **Placement of internal applications** is not part of the cell branches: the team's `placement.yaml` is read from `main` by a Kustomization of its own in every cell (path `placement/`, interval 1 min), so a migration takes effect within minutes and independently of the release train; the application's route carries the home cell as `platform/home-cell`, and the internal ExternalDNS of a cell publishes only the routes that name its own cluster (**internal zones**). External applications are in every cell.
- **Previews and sandboxes** (dev only, [R34](#)) are read the same way, outside the cell branches: a pull request marked for preview is deployed by the dev cells from its branch into `<app>-pr-<n>`, with the application's own configuration, and pruned when it closes; a team's sandbox follows the branch the team names. Neither moves with the release train, and neither reaches acc or prd ([section 1.5](#)).
- **Drift** is corrected on every reconcile: the Kustomizations every 10 minutes (retry after 1 minute), the sources every minute in dev. A cell with a public entry point may also take GitHub's push webhooks: a `Receiver` behind its platform gateway, the platform's one HTTP/1.1 path ([section 4.1](#)), reconciles at once. The internal cells have no public entry point, so a merge reaches them within the source interval. Flux's notification-controller alerts the platform channel when a reconcile fails.

Stateful cluster: CI/CD pipeline (built with the first approved stateful workload)



- **Where it runs:** self-hosted agents in the management VNet ([picture 2.2](#)); the agents reach GitHub, the cluster never does, and it needs no Git, no reconcile controller and no extra egress.
- **Identity:** workload identity federation from the pipeline to a managed identity per environment (no client secret). Azure RBAC for Kubernetes gives it write access only to the stateful cluster, scoped to the platform and application namespaces; humans stay read-only.

- **Stages** of one run, for a given commit:
 1. check out the commit and verify its signature – exactly the content that Flux deploys;
 2. check that all assigned Azure Policy constraints are present in the cluster (policy readiness gate);
 3. `kubectl diff` of `infrastructure/stateful`, then `apps/*/stateful/<env>`, published as the run's change summary;
 4. approval (prd always, acc for operator or CRD changes);
 5. server-side apply with pruning (`kubectl apply --server-side --prune --applyset=...`), infrastructure first;
 6. wait for StatefulSet / operator rollouts (`kubectl rollout status` , operator health), optionally partitioned canary first; run smoke tests against the applications' internal load balancers;
 7. on failure stop and alert; rollback = re-run the pipeline with the previous commit (data migrations are forward-only, see the compatibility rule in [section 3.4](#)).
- **Order inside an environment:** the stateful stage runs first – providers before consumers. Stateful changes must be backwards compatible with the running stateless version anyway (expand → migrate → contract).
- **Platform changes that touch both cluster types stay in lock-step** because there is one commit: a new Cilium policy template, a shared CRD or an operator version in `infrastructure/` is in the commit that the stateful stage applies first and the cells then follow branch by branch. The stateful pipeline refuses a commit whose shared `infrastructure/` differs from the one the cells of the environment run, and a cell's branch cannot move to a commit whose stateful stage has not completed, so the two never diverge by more than one release train.
- **Drift:** there is no reconcile loop, so a scheduled run (nightly) does steps 1–3 only and alerts on any difference. With read-only human access, drift can only come from break-glass actions.

3.4 Zero-downtime application upgrades

Requirements: [R23](#), [R24](#), [R26](#).

The same rules apply in every cluster and are **enforced by admission policy** (Azure Policy add-on), so no application can be deployed in a way that breaks a zero-downtime rollout or a node drain:

Guardrail	Stateless	Stateful
Replicas	<code>replicas / HPA minReplicas ≥ 2</code> , or 0 (switched off; nothing in between)	<code>≥ 3</code> , one per AZ
Spread	<code>topologySpreadConstraints</code> over nodes	<code>topologySpreadConstraints</code> over <code>topology.kubernetes.io/zone</code>
PodDisruptionBudget	Mandatory, <code>maxUnavailable ≥ 1</code>	Mandatory, <code>maxUnavailable: 1</code>
Rollout strategy	RollingUpdate, <code>maxUnavailable: 0</code> , any <code>maxSurge ≥ 1</code>	RollingUpdate (optionally <code>partition</code> for canary) or operator-managed
Probes	readiness + liveness (+ startup) required	readiness gated on replication / quorum
Graceful shutdown	<code>preStop</code> delay + <code>terminationGracePeriodSeconds</code> longer than the gateway / traffic layer drain time	same, plus clean leader hand-over
Resources	requests required (HPA and the cluster autoscaler depend on them); resized in place by the VPA within Git bounds (section 2.3)	requests = limits for memory; VPA recommendations only
Images	From the environment's ACR only, by digest (section 2.7)	same
Non-serving workloads	See below : the replica, spread, budget and rollout rows apply to serving workloads only	same

- **Scaling on load** (stateless): HPA on CPU/memory (KEDA on queue length / request rate when a team needs it); the cluster autoscaler adds nodes for pods that do not fit and removes them again. `maxReplicas` and the zone pools' maximum are sized so that one cluster can take the full load. Vertically, the VPA resizes running pods **in place** – no new pod, usually no restart – within the bounds in Git ([pod right-sizing](#)), so following the load needs no release; only changing the bounds does.
- **No scaling on load in the stateful cluster**: replica counts and node pool sizes are fixed and changed as planned changes through the pipeline and IaC.
- **Compatibility rule**: during a release two versions run at the same time (in different cells, and for a few minutes inside a cell during the rolling update), so API and message changes must be backwards compatible and database changes follow *expand* → *migrate* → *contract* over separate releases. DNS affinity (external zones) and the single home cell (internal zones) reduce what *clients* see of this, but clients that change networks, rollbacks and the data shared by all cells still meet both versions.

Non-serving workloads

Jobs, scheduled jobs and queue consumers serve no request, so the rules above that keep a request flowing do not apply to them ([R23](#), [R24](#)); what applies is that their work completes exactly as often as intended while cells are drained and released:

Workload	Runs as	The rules
A finite job	<code>Job</code> with <code>backoffLimit</code> , <code>activeDeadlineSeconds</code> and <code>ttlSecondsAfterFinished</code> , judged by its exit status	Idempotent, so a restart after a drain repeats nothing twice; resource requests; the same image, security and placement rules as a pod of the application
A scheduled job	<code>CronJob</code> with <code>concurrencyPolicy: Forbid</code> and a <code>startingDeadlineSeconds</code>	Runs in the application's home cell - in both exposures, an external application's non-serving workloads are placed in one cell like an internal application and migrated the same way (internal zones) - so a schedule fires once per environment, never once per cell
A queue consumer	<code>Deployment</code> with at least 1 replica, scaled on queue length (KEDA), readiness and liveness probes, a graceful shutdown longer than its longest message, <code>RollingUpdate</code>	May run in every cell, because the queue serialises the work; below 2 replicas it needs no disruption budget and a drain costs it a pause, not data

During a cell drain a running job is given the drain timeout to finish; a job still running then is stopped and restarted in the target cell by its `CronJob` or owner, which is why it must be idempotent. A release moves a non-serving workload with the cell's branch like any other manifest; the SLO gates do not see it, so a team sets an alert on its own measure of success (last run, queue lag) in its observability directory.

Releases cell by cell

A release – a signed commit of `main` with any number of application changes ([section 3.3](#)) – moves through the cells of an environment **one cell at a time**, with the same *drain* → *change* → *test* → *return* procedure as a cluster upgrade ([section 3.5](#)). External applications run in every cell behind Traffic Manager's DNS affinity; internal applications run only in their home cell ([incoming traffic](#)). Either way, every interactive user switches from the old to the new version **exactly once and never back**:

1. **Stateful stage first** (only if the environment has a stateful cluster): providers before consumers ([section 3.3](#)).
2. **Drain cell 1**: disable its Traffic Manager endpoints and wait for the DNS answers to expire and the cell's connections to drain. Its external users move to the other cells, which still run the old version, so nobody sees a change yet. Internal applications whose home is cell 1 keep serving.
3. **Deploy**: the pipeline moves the cell's branch (`<env>-s1-az1`) to the commit; Flux rolls it out and the pipeline waits until all Kustomizations are `Ready`. No external traffic reaches the cell during the rolling

update; the internal applications of the cell are updated **in place** by their zero-downtime rolling update – this is the moment their users switch to the new version.

4. **Test** through the per-cell test host names (external zones) and against the cell's gateway IPs with the applications' real host names (internal zones): smoke and synthetic tests over the real TLS path.
5. **Return** the cell to Traffic Manager: its external clients come back in steps, a quarter of its address blocks at a time, and this is the moment they switch to the new version. The pipeline watches the error-rate and latency SLOs per cell in Grafana Cloud ([SLO gates](#)); on a breach it drains the cell again – its external clients fall back to the old version in the other cells – and moves the branch back, which also rolls back the cell's internal applications.
6. **Soak**, then repeat for the next cell. After the last cell all clients are on the new version.

An external client only moves when its own cell is drained or returned. While its cell is drained it uses another cell, which runs either the old version or – if that cell was already released – the new one; when its cell returns, it gets the new version. An internal client always uses its application's home cell. So every client switches from the old to the new version **once and never back**:

Client	What it sees during a release
Internal client (corporate network, other Azure workloads)	One version at a time, except during the few minutes of the rolling update in the home cell; switches once, old → new
External client with a stable IP address (browser, mobile app, API client, other system)	One version at a time; switches once, old → new
External client whose IP address changes (e.g. a phone moving between networks)	May change cells with its address and meet the other version – covered by the compatibility rule
Long-lived connection (WebSocket, SSE)	Reconnects when its cell is drained or returned, or its pod is replaced, then follows the rules above

Consequences:

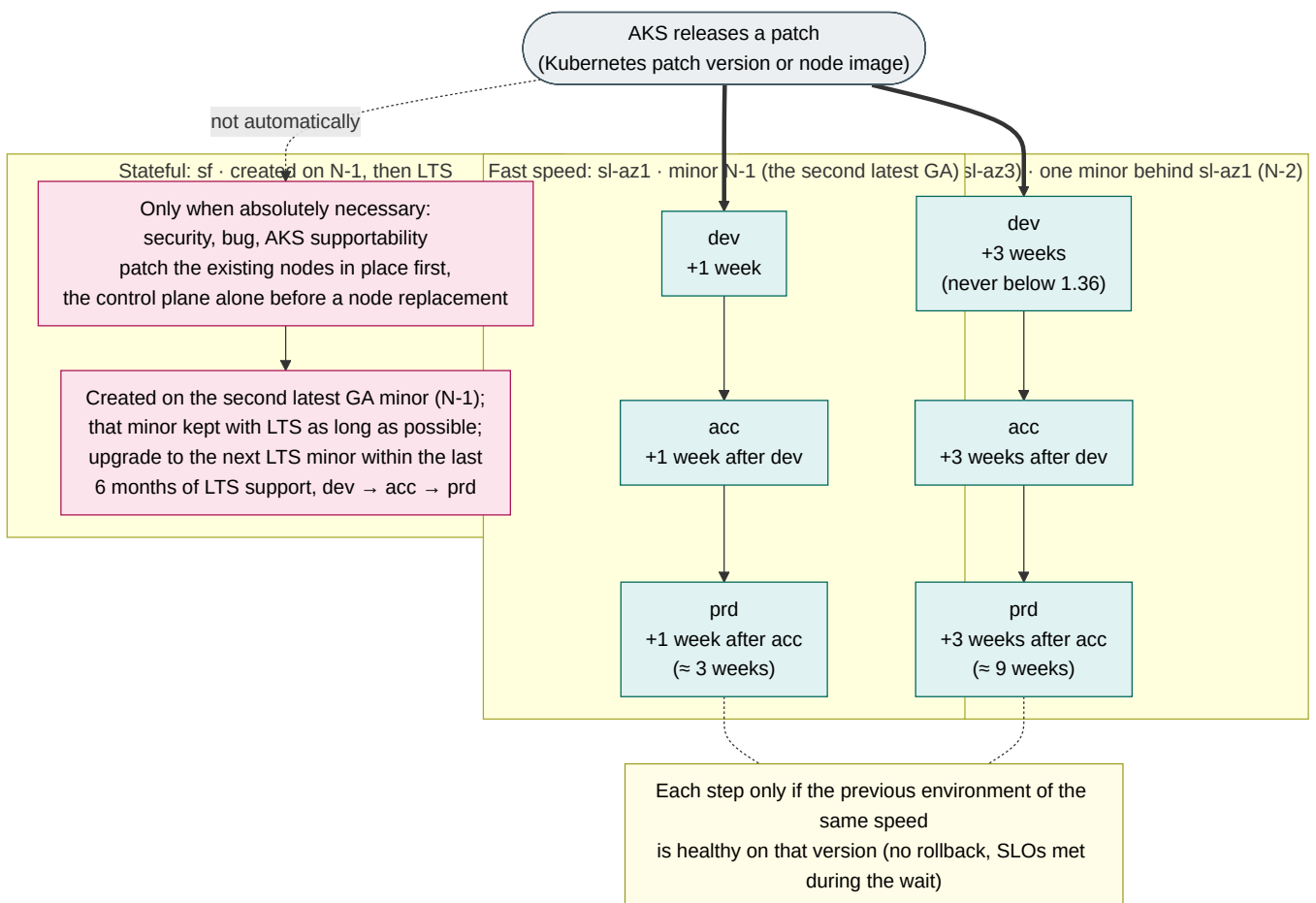
- **Releases are trains.** Each cell costs one drain time plus rollout, tests and the return, so many changes ride in one release instead of one release per application change. An urgent fix takes the same path with a shorter soak.
- **N+1 capacity** is needed for releases too: while one cell is drained, the others carry its load (the same sizing as for cluster upgrades).
- **The first cell is the canary** for external users; for internal applications the canary is the first cell that is home to some of them, so placing a few low-risk internal applications in every cell makes sense. In-cell progressive delivery (Flagger with `HTTPRoute` weights on the zone's gateway) is not used for interactive applications: it splits requests inside the cell, so their clients would jump between versions again. It stays an option for back-end APIs that are strictly backwards compatible.

3.5 Zero-downtime cluster upgrades

Requirements: [R26](#), [R27](#).

Cluster upgrades are triggered by pipelines in a fixed order: **dev** → **acc** → **prd**, with the waiting times of the [update policy](#) below, and within an environment as described per cluster type. Nothing is upgraded by hand.

Update policy



The two cluster types are updated at very different speeds, for the same reason they are built differently: a stateless cell can be drained and replaced at any time, the stateful cluster holds data and is changed as rarely as possible. The difference is the upgrade *cycle*, not the starting point: the stateful cluster starts on the same second latest minor as `sl-az1`, and then moves slowly because the workloads it runs (databases, brokers, operators with data on disks) are harder to move and test.

	Stateless, fast speed: <code>sl-az1</code>	Stateless, slow speed: <code>sl-az2</code> (and <code>sl-az3</code>)	Stateful: <code>sf</code>
Kubernetes minor	N-1 : the second latest GA minor in AKS	One minor behind <code>sl-az1</code> (N-2) – always a minor that <code>sl-az1</code> has already run in <code>prd</code>	N-1 when the cluster is created (the same minor as <code>sl-az1</code>), then kept on that minor as long as it has Long Term Support
Minor upgrades	When a new minor becomes GA, to the new N-1, with the same 1-week steps	To the minor <code>sl-az1</code> leaves, with the same 3-week steps	Within the last 6 months of LTS support of the current minor: <code>dev</code> first, then <code>acc</code> , then <code>prd</code>
How nodes are updated	Cell drained, control plane and every pool upgraded with surge (below)	Same	Patched in place where possible; nodes replaced only when a change requires it (below)
Node image of new nodes	Pinned per cell: a node created during scale-out takes its pool's node image, which changes only when the version pipeline upgrades the pool while the cell is drained	Same	The pool's node image version
AKS auto-upgrade channels	<code>none</code> – the version pipeline decides	<code>none</code>	Cluster <code>none</code> ; node OS channel for in-place patches (below)

Starting point: Kubernetes 1.36. The platform is new, so every cluster – both stateless speeds and the stateful cluster – is created on **1.36** (AKS GA June 2026, also an LTS minor), and 1.36 is the **floor**: no cluster runs an older minor. The slow speed's "one minor behind `sl-az1`" therefore starts to apply only when `sl-az1` moves past 1.36; until then `sl-az2` runs 1.36 too, with its slower patch timetable. The floor gives every cell in-place pod resize for containers (GA) and pod-level resources (beta), which [pod right-sizing](#) relies on. The stateful cluster is the one exception: it is created on **1.37 or later** (AKS GA October 2026, also an LTS minor), because 1.37 is the first minor in which every `PersistentVolumeClaim` reports when it was last used - the `Unused` condition the storage agent works from. Since the cluster is built only with the first approved exception, this costs nothing today; should that exception come before 1.37 is the second latest minor, the cluster takes 1.37 as the newer of the two LTS minors rather than waiting. None of 1.36's and 1.37's deprecations touch the platform: it runs no kube-proxy (Cilium), cgroups v2 (Azure Linux 3.0), CoreDNS, no static pods of its own and no `gitRepo` volumes; `Service.spec.externalIPs`, deprecated in 1.36, is denied by policy ([section 2.7](#)). AKS's own backstop stays far behind the policy: from June 2027 it upgrades by itself any cluster still on an unsupported minor 60 days after platform support ended, and its end-of-support notifications (Resource Graph, Event Grid; GA May 2026) are one of the upgrade agent's triggers ([section 6.1](#)).

Why two speeds in the stateless clusters. At any moment one cell runs a version that is newer and less proven and the other cell(s) a version that `sl-az1` has already run in production – so a regression in a Kubernetes patch, a node image or a minor version reaches only the fast cell first, while the slow cell(s) keep serving ([releases cell by cell](#) uses the same idea for applications). Removed or changed Kubernetes APIs show up in `sl-az1` (`dev` first) a

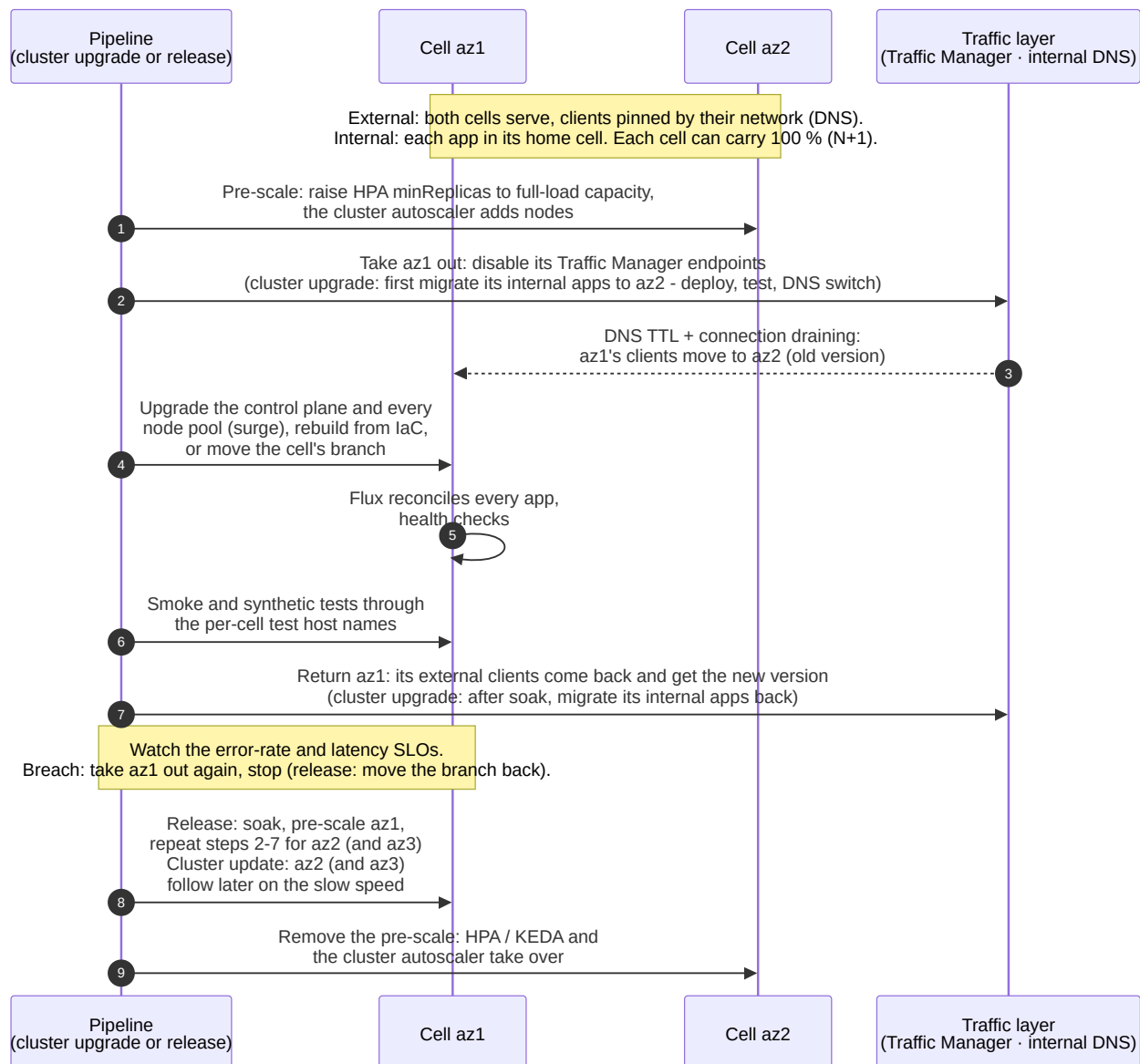
full minor cycle before they reach `s1-az2`. Applications and fleet manifests must therefore work on both minors; CI validates the rendered manifests against both (and against the stateful LTS version).

How the stateless timetable is run. A scheduled **version pipeline** reads the Kubernetes versions and node images AKS offers in the region and the date each was released, computes the target version and node image of every stateless cluster from the table above, and the upgrade agent ([section 6.1](#)) opens it as a pull request to the cluster IaC with a compatibility report. The cluster upgrade pipeline then applies it cell by cell with the drain procedure below; dev and acc run without approval, prd needs an approval only for minor upgrades. Rules:

- Every patch has its own timeline counted from its AKS release date; if a newer patch is already due, it replaces an older one that has not been promoted yet (patches are not applied one by one).
- `s1-az2` runs the oldest minor in AKS standard support. Its minor upgrade is scheduled so that prd is upgraded before that minor's end of support in the [AKS Kubernetes release calendar](#); if the calendar leaves less than 3 weeks per step, the steps are shortened.
- **Why the slow speed is 3 weeks and not a month:** a month per step would put prd's slow cell on a node image that is three months old before it changes; three weeks keeps it at about nine weeks, and still a full release cycle behind `s1-az1`.
- **Emergency fast track:** a fix for an actively exploited vulnerability can skip the waiting times for both speeds with the platform owner's approval – still dev → acc → prd and still one cell at a time.
- `s1-az3` follows the slow speed so that two of three cells run the proven version.

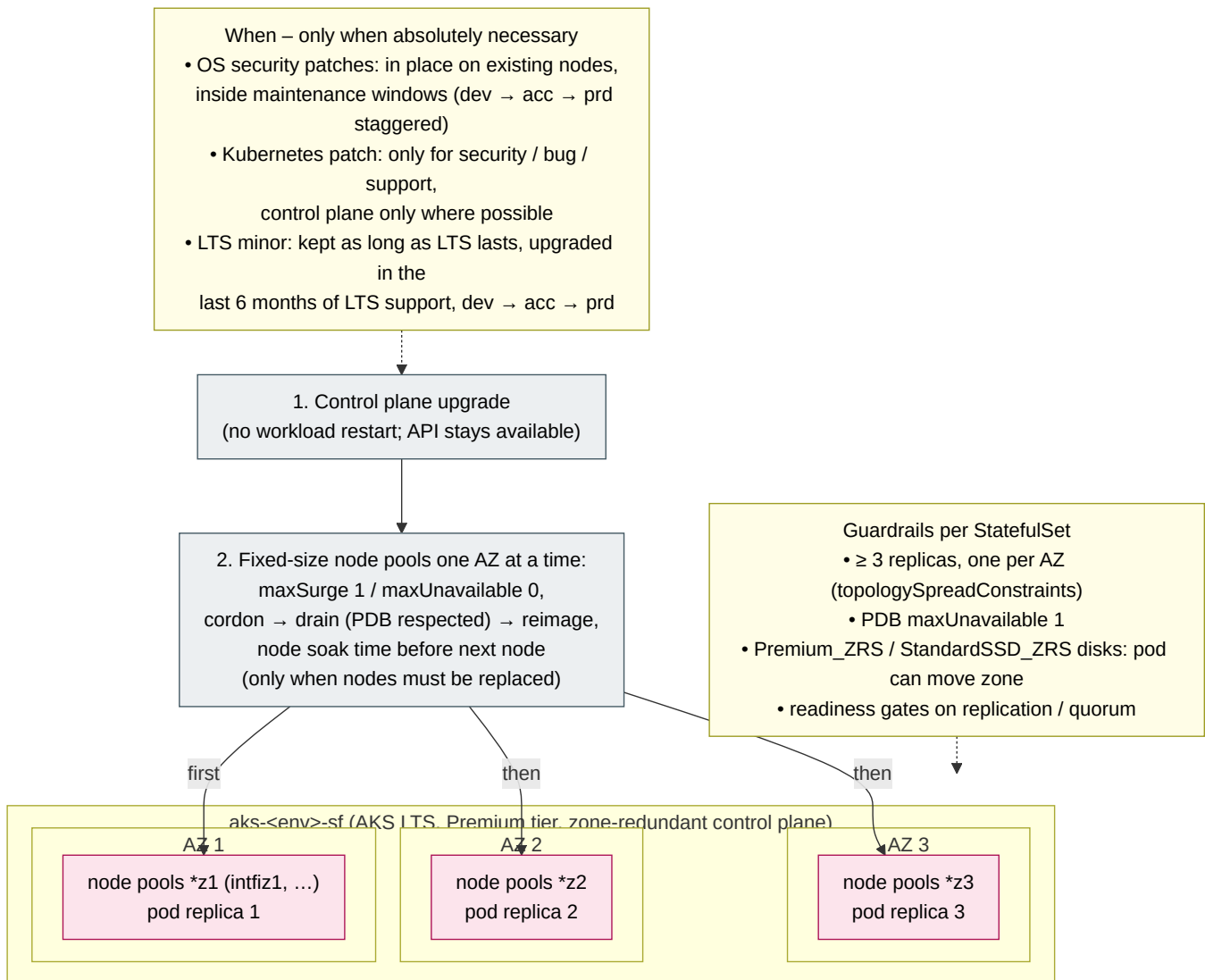
Stateful cluster: only when absolutely necessary. Of the picture's reasons, security means a vulnerability in the node OS, kubelet, container runtime or Kubernetes that cannot be mitigated otherwise, and a bug one that affects the stateful workloads; the 6-month LTS window holds dev, acc and prd with soak times between them. The smallest change is chosen: an in-place OS security patch (`SecurityPatch` channel, [below](#)) before a node image upgrade, a control-plane-only patch upgrade (`az aks upgrade --control-plane-only`; kubelets may lag behind the API server within the Kubernetes version skew policy) before replacing nodes, and node replacement only when the change cannot be made otherwise (e.g. a new Kubernetes minor).

Stateless clusters



Users never hit a cell that is being changed. The procedure is the one of an application release ([releases cell by cell](#)), and the two never run in the same cell at the same time. The internal applications move with the **migration workflow** ([internal zones](#)), the tests use the per-cell host names ([incoming traffic](#)), and Traffic Manager returns the external clients in steps of address blocks. The pools upgrade with surge, PDBs respected; a large change (new VNet, CNI, OS SKU) **creates a fresh cluster** from IaC instead and lets Flux bootstrap it (blue/green at cluster level). Node images and OS security updates follow the same timetable and procedure; the AKS auto-upgrade and node OS channels are off, so nothing changes a cell outside it. Prerequisite for all of this is **N+1 capacity**: the zone pools' maximum, vCPU quota and node subnet size must allow one cluster to carry the whole environment.

Stateful cluster



There is only one stateful cluster, so it is upgraded **in place** and protected by zone redundancy:

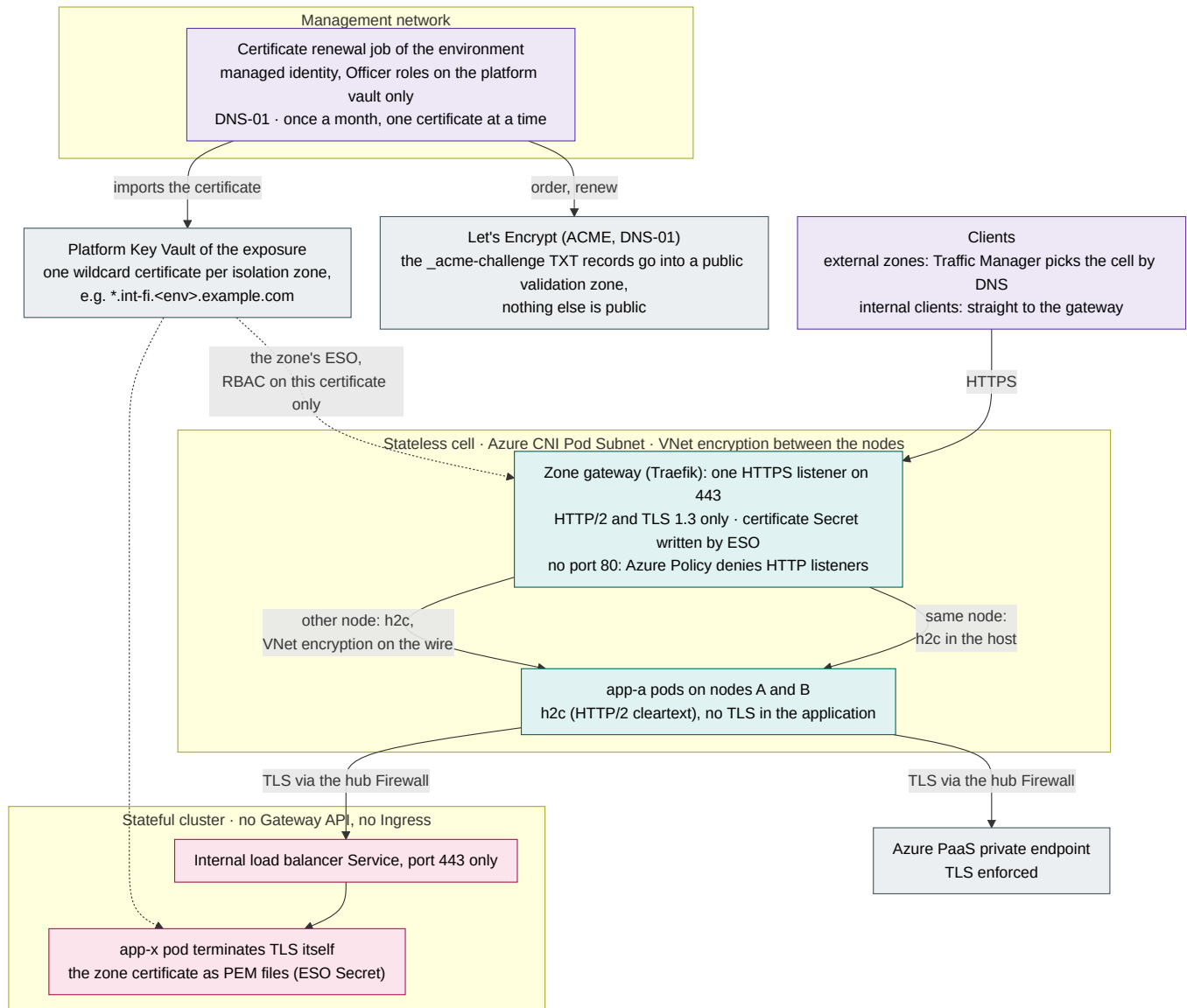
- **Version policy (update policy):** cluster auto-upgrade channel `none`; the pipeline applies Kubernetes patch versions dev → acc → prd, preferably control plane only.
- **OS patches in place:** the node OS channel `SecurityPatch` (AKS node OS upgrades) patches the existing nodes and reimages only for kernel-level patches, inside the maintenance window (dev a week before acc, two weeks before prd), respecting the surge settings and the PDBs - AKS-managed, no package mirror and no reboot daemon in the Internet-less cluster; if AKS refuses it on a network isolated cluster, `NodeImage` inside the same window. **Image Cleaner** runs weekly on every cluster and removes unused, vulnerable images from these long-lived nodes.
- **Node pools:** one fixed-size pool per isolation zone *and* AZ, so a pool upgrade touches only one AZ, with a drain timeout besides the picture's settings; at most one replica of each StatefulSet is down at any moment. The surge node is created in the same AZ, so zonal disks re-attach. The vCPU quota must leave room for the surge nodes, because the pools do not autoscale.
- **LTS minor upgrades** (rare, once per LTS cycle, within the last 6 months of LTS support): the same procedure, rehearsed in dev and acc first. Alternative for risky jumps: add new node pools on the new

version, cordon and drain the old pools AZ by AZ, delete them.

- **The way back is [node pool rollback](#):** a pool that misbehaves after a Kubernetes or node image step goes back one step, one pool - one AZ - at a time, with the same surge and PDB rules, which is why the pipeline moves the stateful pools one version at a time and rehearses the rollback in dev. The stateless cells need none of this: their way back is the previous cell behind the router.
- **Settings changes that reimage nodes** (SSH access, the IMDS restriction, the outbound type) are planned like a node image step, held to the window by AKS's Node Disruption Policy; a control-plane-only upgrade to the next LTS minor is allowed within the version skew, which separates the control plane step from the pools'.

4.1 Encryption in transit and TLS

Requirements: [R12](#), [R21](#), [R22](#).



Every hop is encrypted, but each cluster type does it in the way that costs the applications least. Encryption at rest with customer-managed keys is in [section 4.2](#).

Stateless clusters: VNet encryption + TLS-only zone gateway

- **No service mesh, no mTLS, no TLS configuration in the applications.** Applications listen on plain HTTP inside the cluster. Traffic between pods on the same node never leaves the host; traffic between nodes is encrypted by **Azure Virtual Network encryption**, enabled on every stateless spoke VNet (and the peering

to its hub; Azure Firewall Premium takes part in it, Standard does not – one reason the Firewalls are Premium).

- Requires node VM sizes that support VNet encryption (accelerated networking); because the only enforcement mode is `AllowUnencrypted`, an unsupported VM size would silently send clear text. The allowed VM sizes are therefore enforced with Azure Policy on the node pools' VM sizes, and VNet encryption on the VNets.
- VNet encryption covers VM-to-VM traffic in the VNet and peered VNets. Everything that leaves the stateless cluster (to Azure Firewall, the stateful cluster, PaaS, Internet) is TLS anyway – see below.
- **Gateway API only, no Ingress.** HTTP traffic into a cell is configured exclusively with the Kubernetes [Gateway API](#) (`GatewayClass`, `Gateway`, `HTTPRoute`; `GRPCRoute` where needed), standard channel only. The Kubernetes Ingress API is not used anywhere: no Ingress controller is installed (the managed NGINX of App Routing is not enabled), and Azure Policy rejects `Ingress` objects.
- **HTTP/2 only and TLS 1.3 only, from day 1: at the edge and between a gateway and a pod.** HTTP/1.1 cannot be made safe against request smuggling, and the dangerous hop is the one a reverse proxy shares between users - its pool of connections to the back ends ([PortSwigger, "HTTP/1.1 must die", 2025](#)); accepting HTTP/2 at the edge and downgrading upstream is worse than HTTP/1.1 end to end. So:
 - At the edge the gateway offers **HTTP/2 only** (Traefik's default TLS options list `h2` as the one ALPN protocol) and **TLS 1.3 only** as the minimum version: no TLS 1.2 client to keep, no cipher-suite choices, and Go's defaults add the hybrid post-quantum key exchange X25519MLKEM768. The policy `gateway-tls` reads both settings back in every gateway namespace so neither can drift ([section 2.7](#)). The residual: a client that offers only `http/1.1` still gets HTTP/1.1 from Go's server (golang/go#46310). App Routing's Istio exposes neither setting on a listener, so Traefik is the implementation in every environment until it can.
 - Between a gateway and a pod the platform speaks **HTTP/2 cleartext with prior knowledge (h2c)**, chosen by one field on the application's Service port, `appProtocol: kubernetes.io/h2c`, which Azure Policy requires on every routed Service port ([section 2.7](#)). The application contract gains one line: **an application serves h2c on its route port** (most frameworks do; the onboarding agent checks the image with one `curl --http2-prior-knowledge`). The kubelet's probes stay HTTP/1.1 until Kubernetes 1.38 lets a probe speak HTTP/2 ([KEP-5999](#)) - harmless, because the vulnerability is the proxy's reused upstream connection.
 - Nothing that downgrades may stand in front of a gateway: Application Gateway v2 and Front Door speak HTTP/1.1 to their back ends.
 - One exception: Flux's webhook receiver and GitHub's deliveries are HTTP/1.1, behind the platform gateway alone, in cells with a public entry point.
- **Two Gateway API implementations, chosen per cluster: Traefik Proxy in every environment**, because App Routing cannot yet offer HTTP/2 alone and TLS 1.3 alone at the edge, and **AKS App Routing's Gateway API implementation**
 - the Azure standard offering - when it can. App Routing runs a sidecar-less Istio control plane per cell that serves the `GatewayClass` `approuting-istio` and an Envoy Deployment per `Gateway`; it is **not a service mesh**, AKS upgrades it with the cluster, and it cannot run together with the Istio service mesh add-on.
 - One `Gateway` per isolation zone in the platform namespace `<zone>-gateway` (a reserved application name), its Envoy pods on the system pool (so that a zone without applications has no nodes, [section 2.3](#)), behind an internal load balancer in `snet-<zone>-ilb` (an external zone's: a

public load balancer, see [incoming traffic](#)). The platform owns the `GatewayClass` and one `Gateway` per zone; application teams own only `HTTPRoute`s in their namespaces – the Gateway API role split between cluster operator and application developer. `allowedRoutes` selects namespaces with `platform/zone=<zone>`, so a route can never attach to another zone's gateway.

- With App Routing the data plane is per zone but the control plane is the cell's one istiod: if it fails, the gateways keep serving their last configuration until it is back - the cell is the bulkhead, not the zone. Neither implementation offers rate limits, request size limits or an in-gateway WAF ([section 3.1](#)). App Routing's own Key Vault and DNS integration is not used: the certificate comes through ESO ([below](#)) and the DNS records from ExternalDNS.
- **Traefik Proxy** (a per-cluster flag of the cluster IaC) also costs less than istiod's requests (1 CPU and 4 GiB per cell, not tunable – [Azure/AKS#5939](#)), a second system node in a small cell. With Traefik each Gateway gets its own Traefik Deployment and `GatewayClass traefik-<gateway>` on the system pool – no shared control plane, so a zone's gateway depends on nothing of another zone's – reading its configuration and certificate from the API server (two replicas, a PDB, `externalTrafficPolicy: Local`, a Cilium policy to the API server). The Gateway, its listener on 443, its certificate, the load balancer and the Cilium policies are the same for both, and so are the teams' `HTTPRoute`s; the implementation-specific part is a kustomize component of the platform repository, and a change of class re-creates the Gateways (a few minutes without gateways, so the cell is drained first).
- **The platform gateway, only with the Flux webhook.** A cell with a public entry point may run a Gateway `platform` in the namespace `platform-gateway` with one HTTPS listener `flux-webhook-<cell>.<dns zone>` for the Flux webhook receivers (a per-cell flag of the cluster IaC brings it). GitHub delivers webhooks over HTTP/1.1 and the receiver speaks nothing else - the platform's one HTTP/1.1 path, which takes only deliveries signed with the token in the platform vault. An internal cell has none - a public load balancer cannot answer behind the hub Firewall - and polls Git every minute. The public addresses are the external zones' gateways and, where the webhook is on, the platform gateway.
- **DNS names with Let's Encrypt certificates.** Each zone has its own DNS name space, e.g. `*.int-fi.prd.example.com`; an internal zone's names are resolved only by its Private DNS zone in the hub (split horizon: the name has no public A record), an external zone's are public. The zone's gateway serves the zone's wildcard certificate from the platform Key Vault ([below](#)). Because the certificate is publicly trusted, Internet clients and clients inside the corporate network need no private CA.
- **No non-TLS traffic.** The zone's `Gateway` has only an HTTPS listener on 443; port 80 is not exposed at all (no HTTP → HTTPS redirect listener either), the internal load balancer has only port 443, and the NSG on `snet-<zone>-ilb` allows only TCP 443. Azure Policy rejects `Gateway` listeners with protocol `HTTP`, `HTTPRoute`s that do not attach to the HTTPS listener, hostnames outside the zone's domain, `Ingress` objects, and `LoadBalancer / NodePort Services` outside `<zone>-gateway`. The gateway does not add HSTS by itself: applications add the `Strict-Transport-Security` header with a `ResponseHeaderModifier` filter on their `HTTPRoute` (the platform may later require it with Azure Policy).

Stateful cluster: the application terminates TLS

- **No Gateway API implementation and no Ingress** – the CRDs are not installed and Azure Policy rejects `Ingress`, `Gateway` and `HTTPRoute` objects. Fewer moving parts in the cluster that is hardest to upgrade.

- **Azure CNI (VNet-integrated):** pod IPs are routable in the backend spoke, so NSGs, the Firewall and Cilium policies see real addresses.
- Each application is published with a `LoadBalancer` Service that must be internal (`service.beta.kubernetes.io/azure-load-balancer-internal: "true"`), placed in its zone's `snet-<zone>-ilb` (`...-internal-subnet` annotation) and use `externalTrafficPolicy: Local`; `NodePort` and public load balancers are rejected by Azure Policy.
- **The application handles TLS itself** (TLS 1.3 only, the same rule as the gateways'), which is an onboarding requirement for the stateful cluster: it terminates TLS on its listener and reloads the certificate on rotation. Most products that end up here (databases, brokers, search engines) support this natively.
- The application uses the zone's gateway certificate ([below](#)) and is reached under a name in the zone's domain, e.g. `app-x.int-fi.prd.example.com`.
- Clients in the stateless clusters connect with TLS and verify the name; plain-text ports are not allowed by the Firewall rules between frontend and backend.

Certificates issued centrally, distributed through the platform Key Vault

No cluster issues certificates – there is no cert-manager or other ACME client in any cluster, so no cluster needs Internet access to Let's Encrypt or write access to DNS.

- **One platform Key Vault per environment and exposure.** `kv-<env>-<exposure>-platform` – `kv-<env>-platform` for short, [section 1.2](#); the real names are `kv-<org>-<region>-platform` – (RBAC permission model, public access disabled, purge protection) exists once per environment and exposure, is created by the environment IaC module and is shared by all clusters of that exposure. It holds **all certificates of the exposure's zones** – the zone vaults hold none – plus the renewal job's ACME account key, and the customer-managed keys of the exposure's clusters ([section 4.2](#)). It has one private endpoint per environment in the shared subnet `snet-shared-pe`, reached from every cell through the hub ([picture 2.1](#)).
- **Certificates per environment.** Each environment gets its own certificates in its own vault; nothing is copied between environments, and a dev or acc identity can never read a prd key:

Environment	Platform Key Vaults	Certificates (one wildcard per isolation zone, plus the platform gateway's)
dev	<code>kv-dev-int-platform</code> , <code>kv-dev-ext-platform</code>	<code>int-fi-gateway--wildcard</code> (<code>*.int-fi.dev.example.com</code>) and <code>int-se-gateway--wildcard</code> in the internal vault; <code>ext-fi-gateway--wildcard</code> , <code>ext-se-gateway--wildcard</code> and <code>platform-gateway--wildcard</code> in the external one
acc	<code>kv-acc-int-platform</code> , <code>kv-acc-ext-platform</code>	the same names for <code>*.<zone>.acc.example.com</code>
prd	<code>kv-prd-int-platform</code> , <code>kv-prd-ext-platform</code>	the same names for <code>*.<zone>.prd.example.com</code>

The certificate *names* are identical in every environment, so the fleet manifests (`SecretStore` , `ExternalSecret`) need only the vault URL of the cluster's environment and exposure, substituted from `cluster-settings` . The names follow the vault's `<namespace>--<secret>` convention (section 2.5): the zone's gateway namespace `<zone>-gateway` owns `<zone>-gateway--wildcard` . An application that must not share the zone key gets its own certificate `<namespace>--<name>` in the same vault.

- **One renewal job per environment** (a pipeline or a Container Apps job): its identity holds **Key Vault Certificates Officer and Key Vault Secrets Officer on that environment's platform vault** and DNS Zone Contributor on the validation zone, to which the zone names are delegated by CNAME, so it cannot change any other record. The ACME account key is the secret `letsencrypt-account-key` , outside every namespace's prefix. It renews when less than 30 days are left, and Key Vault `CertificateNearExpiry` events alert the platform team if renewal fails. The jobs are rolled out dev → acc → prd like any other platform change.
- **All clusters of an environment use the same certificate:** the zone gateways in `sl-az1` , `sl-az2` , `sl-az3` and the TLS-terminating applications in `sf` . A rebuilt cluster needs no new certificate, and Let's Encrypt rate limits are never an issue (a handful of certificates per environment).
- **Access: the same prefix share as every other secret.** The gateway namespace `<zone>-gateway` is an ordinary Key Vault share of the platform vault (section 2.5): its reader identity `id-<region>-<env>-shared-kv-<zone>-gateway` , federated with the namespace's `key-vault` ServiceAccount, holds **Key Vault Secrets User with the ABAC condition on the `<zone>-gateway--` prefix**, so it reads the zone's wildcard and nothing else; only External Secrets Operator uses it. Stateful applications and any application with a client or server certificate get the same kind of share on the platform vault. Key Vault exposes a certificate together with its private key as a secret of the same name, which is what ESO reads. Neither the gateway nor any application calls Key Vault itself. No identity has a data-plane role on the vault scope except the renewal job and the platform team (PIM); the clusters' control plane identities have a scoped role on their own key, and Key Vault Contributor (management plane only) where KMS needs it (section 4.2). The shares are created by the environment IaC module, so a certificate can be added before it exists; until the job's first run the gateway reports `InvalidCertificateRef` and answers nothing.
- **Delivery and rotation** with the zone's External Secrets Operator (the Secrets Store CSI driver is disabled): an `ExternalSecret` reads the certificate's secret object (PKCS#12) through the namespace's platform-vault `SecretStore` , and its template unpacks it (`pkcs12cert` , `pkcs12key`) into a `kubernetes.io/tls` Secret `gateway-tls` (`tls.crt` with the chain, `tls.key`). The refresh interval is 1 hour – a renewed certificate is in every cluster well within the 30-day renewal margin.
 - The zone's gateway: the `ExternalSecret` in `<zone>-gateway` writes the Secret that the zone's `Gateway` listener references; istiod hands the changed Secret to the gateway's Envoy pods, and Traefik watches it itself – both reload without a restart.
 - Applications: the Secret is mounted as PEM files (no `subPath`) and the application reloads them on change.
- Things to be aware of:
 - The parent domain must be a **registered public domain** – suffixes like `.internal` , `.local` or `.corp` cannot get Let's Encrypt certificates.
 - Certificates appear in public Certificate Transparency logs; the per-zone wildcard keeps application names out of them.

- The wildcard private key is shared by everything in the zone that needs a certificate. Its blast radius is one zone of one environment, and it is renewed every 60 days.
- The platform vault is shared by all zones of an environment, so zone separation for certificates rests on the prefix conditions (and Cilium egress policy) rather than on a separate vault – the same model that separates the applications' secrets inside a zone vault.
- Egress `acme-v02.api.letsencrypt.org` is allowed only for the renewal job, not for any cluster.

Service mesh or not

The stateless clusters run **no service mesh** (R22). The question comes back with every new Istio or AKS feature, so this is the reasoning.

What a mesh offers, and what the platform does instead:

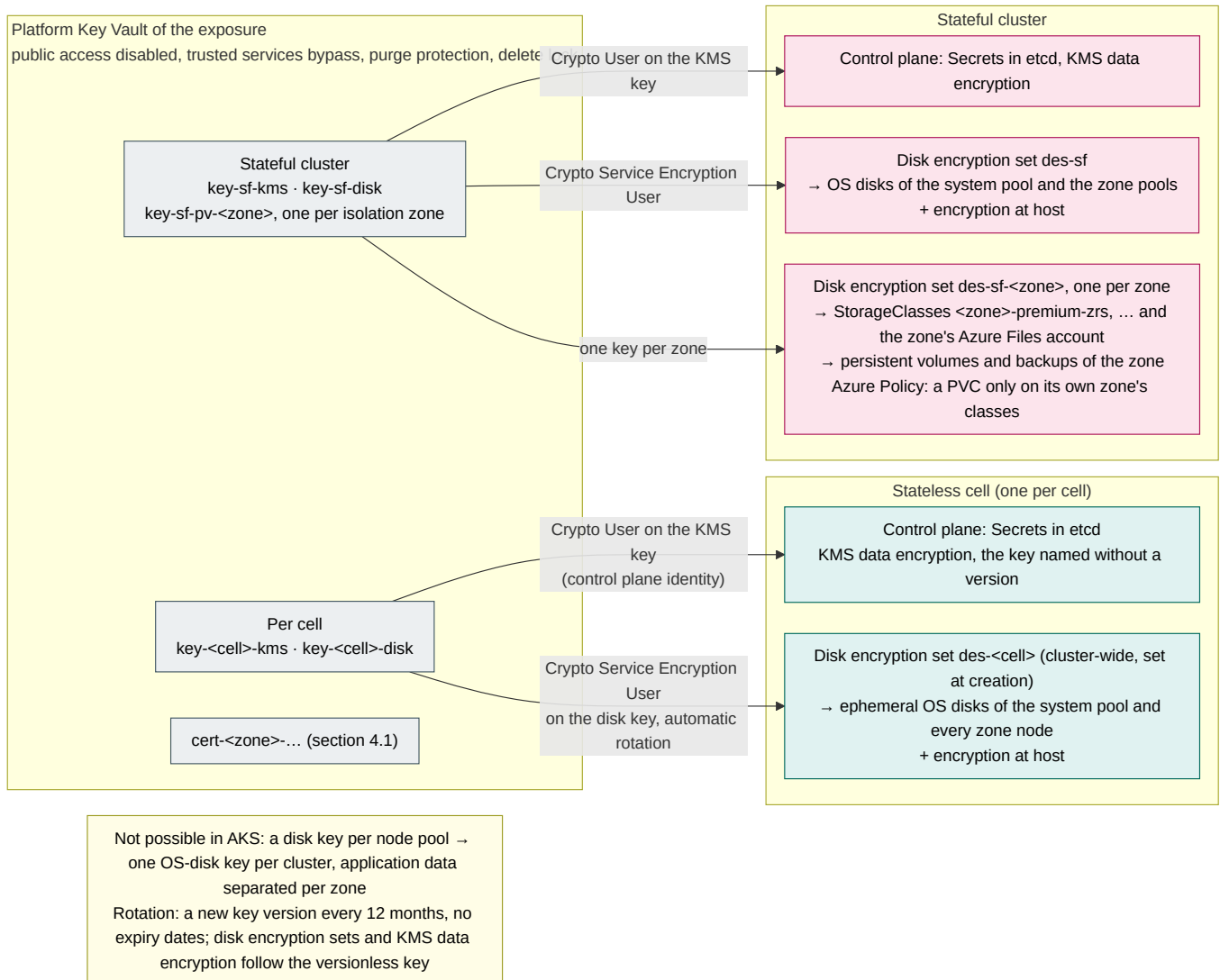
A mesh gives	The platform does instead	What is missing without the mesh
Encryption between services (mTLS)	Azure Virtual Network encryption between nodes; traffic between pods of one node never leaves the host	Nothing in transit: every hop is encrypted either way. What mTLS adds is a workload <i>identity</i> on the wire, which the next row covers. Since Kubernetes 1.37 the building blocks for that identity are in-core and GA - <code>PodCertificateRequest</code> (the kubelet obtains an X.509 certificate per pod from a signer) and <code>ClusterTrustBundle</code> (the CA bundle every pod can mount) - but AKS ships no signer, so using them means running one
Identity-based authorisation (which service may call which)	Cilium policies by namespace and label decide which pods may connect; applications validate Entra ID tokens of their callers for <i>who</i> may call <i>what</i>	Authorisation by network identity without code in the application. Here it lives in the application, which already has to validate the caller's token for its own data
Layer 7 traffic management : retries, timeouts, circuit breaking, traffic shifting, fault injection	Releases move cell by cell behind the cell router (section 3.4); retries and timeouts are the client library's	In-cell canaries by request share - deliberately not used for interactive applications
Telemetry without code : per-request metrics and traces between services	The zone's gateway gives the first span and the default SLOs; Beyla's eBPF instrumentation in Grafana Alloy gives per-service RED metrics without code (section 6.2); applications emit OpenTelemetry	Nothing of substance since Beyla
Egress control by name	The hub Firewall per zone, from per-application rule files (section 5.1)	–

What a mesh costs: a data plane in every pod's path (sidecars, or ambient's per-node ztunnel and waypoints) that the platform runs, upgrades and debugs; a control plane that cannot run next to App Routing's istiod; a second policy language next to Cilium policies and HTTPRoute s; and the largest cloud-specific component in the cluster.

4. ENCRYPTION

4.2 Encryption at rest with customer-managed keys

Requirements: [R28](#).



All data at rest in the clusters is encrypted with **customer-managed keys (CMK)**, not Microsoft-managed keys, kept in the platform Key Vault `kv-<env>-<exposure>-platform` – `kv-<env>-platform` for short, [section 1.2](#) – next to the certificates ([section 4.1](#)).

Keys

Key (in <code>kv-<env>-platform</code>)	Encrypts	Used by	Mechanism
<code>key-<cluster>-kms</code> (<code>key-aks-prd-int-stateless-az1-kms</code>)	Kubernetes Secrets in etcd (control plane)	The cluster's user-assigned control plane identity	KMS data encryption with a private key vault (below)
<code>key-<cluster>-disk</code>	OS disks of every node in the cluster	Disk encryption set <code>des-<cluster></code>	Disk encryption set on the cluster (<code>--node-osdisk-diskencryptionset-id</code>)
<code>key-sf-pv-<zone></code> (target, with the stateful cluster)	Persistent volumes (Azure Disk) and the Azure Files storage account of one isolation zone in the stateful cluster	Disk encryption set <code>des-<env>-sf-<zone></code> ; the zone's storage account	StorageClass <code>diskEncryptionSetID</code> ; storage account CMK

Nothing is shared between environments, because each environment has its own platform vault.

Why not one key per node pool. The requirement was a key per node pool, but AKS does not support it: node pools added later use the cluster's disk encryption set too. The closest the design can get is:

- **OS disks: one key per cluster.** Node OS disks contain only the node image, container images and ephemeral pod data, never application data. In the stateless clusters they are ephemeral.
- **Application data: one key per isolation zone.** In the stateful cluster, every zone's node pools (`<zone>z1 – z3`) write their data to persistent volumes, and those use the zone's own key. Disabling `key-sf-pv-int-fi` makes the `int-fi` data unreadable and leaves the other zones alone.

If AKS later supports a disk encryption set per node pool, the OS-disk key can be split per zone the same way.

Control plane: KMS etcd encryption

- Every cluster runs with KMS against the platform vault and its own `key-<cluster>-kms`, the vault reached as `Private` (no public network access).
- **KMS data encryption** (preview). It names a **versionless** key and reaches a vault without public network access through the vault's trusted services exception (`bypass AzureServices`) – no private endpoint of AKS's own and no API Server VNet Integration, which Azure refuses on an encrypted VNet with v4+ VM sizes ([section 2.2](#)). Once on, it cannot be turned off; AKS rewrites every Secret as `aksService` when it is switched on, which the `secrets-via-eso` policy exempts.
- **The alternative, classic KMS** (GA), reaches a private vault only through API Server VNet Integration; a classic-KMS cluster switches to KMS data encryption in two updates.
- KMS requires a **user-assigned** control plane identity (`id-<region>-<env>-aks-<cell>`), created by the cluster module before the cluster. It gets **Key Vault Crypto User scoped to its own KMS key**, Reader on

its disk encryption set and – with KMS data encryption, AKS’s documented prerequisite because it reads the vault’s network settings – **Key Vault Contributor on the platform vault**. See the trade-off below.

- Everything that ends up in etcd is encrypted with the cluster’s key: the application secrets and certificates delivered as Kubernetes Secrets by External Secrets Operator ([section 2.5](#)), the zone gateways’ certificates, Flux and platform Secrets.

Nodes: disk encryption set per cluster

- The cluster module creates `des-<cluster>` (system-assigned identity with **Key Vault Crypto Service Encryption User** **scoped to the cluster’s disk key**, automatic key rotation enabled, versionless key URL) before the cluster and passes it as `--node-osdisk-diskencryptionset-id`. The cluster identity gets **Reader** on the disk encryption set.
- The disk encryption set reaches the private vault through the **trusted Microsoft services** exception. The platform vault therefore keeps `publicNetworkAccess: Disabled` and adds `bypass: AzureServices`. The trusted services bypass still applies when public access is disabled, so the vault gets no public network path.
- **Encryption at host** is enabled on every node (`--enable-encryption-at-host` on every node pool), so temp disks and disk caches on the VM host are encrypted as well. Azure Policy enforces it like the VM size allow-list. The allow-list may therefore contain only SKUs that support **both** VNet encryption and encryption at host ([section 2.3](#)).
- The disk encryption set can only be set when a cluster is created, so **every cluster is created with CMK from the start** by the cluster module. The cluster module must never create a cluster without it, because adding it later would mean rebuilding the cluster.

Stateful cluster: a key per isolation zone for persistent volumes

- The zone’s disk encryption set `des-<env>-sf-<zone>` has access to its own `key-sf-pv-<zone>` only, and its StorageClasses (`<zone>-premium-zrs`, `<zone>-standardssd-zrs`) name it as `diskEncryptionSetID`. The cluster identity has Reader on all of them. The built-in StorageClasses are not used, and only the platform can create StorageClasses.
- **Azure Files**: the CSI driver’s dynamically created storage accounts would use Microsoft-managed keys. Each zone therefore has one storage account created by IaC (ZRS, private endpoint in `snet-<zone>-pe`, CMK with `key-sf-pv-<zone>`), and the zone’s Azure Files StorageClass points at it (`storageAccount` parameter).
- **Backups** (Azure Backup for AKS) are disk snapshots that keep the disk’s encryption set, so a zone’s snapshots are under the same key as its data. The vault-tier copies that protect against a region loss are restored into volumes under the paired region’s new zone keys ([section 6.3](#)).
- A disk encryption set on a StorageClass applies only to **new** volumes. The per-zone classes therefore exist before the first workload is onboarded. Adding a country adds the zones’ keys, disk encryption sets and StorageClasses ([adding a country](#)).

Rotation and lifecycle

Key	Rotation	What happens
Disk keys (*-disk , key-sf- pv-*)	Key Vault rotation policy, new version every 12 months	Disk encryption sets with automatic key rotation pick up the new version within an hour. Managed disks are re-wrapped online; ephemeral OS disks get it when the node is recreated (node image upgrade, cell rebuild)
KMS keys (*- kms)	Same rotation policy	KMS data encryption names the versionless key and follows the rotation by itself, nothing to re-write. A classic-KMS cluster would take the new version as --azure-keyvault-kms-key-id on the next apply (on Kubernetes 1.37 AKS accepts only a versionless key ID, so a classic-KMS cluster moves to the versionless form before that minor) and re-write all Secrets (kubectl get secrets -A -o json \ kubectl replace -f -); KMS keeps the previous version cached, so the older one is disabled only after the next rotation

- **HSM-protected keys.** The production platform vault is Key Vault **Premium** and every customer-managed key is an RSA-HSM key (FIPS 140-3 Level 3 validated HSMS), the Azure security baseline for customer-managed keys at a cost of cents per key; a Managed HSM pool is not needed for this assurance level. The SKU is a per-environment setting: dev and acc may run Standard vaults with software keys, prd runs Premium.
- **No expiry date on any key version.** An expired or deleted KMS key stops the API server, and the encrypted Secrets cannot be recovered; an expired disk key stops VMs from starting. Rotation policies only create new versions. Old versions are disabled by hand once nothing uses them. Since Kubernetes 1.37 (beta, KEP-3926) an administrator can at least *delete* objects the API server can no longer decrypt through the API, so a cluster whose KMS key version is gone can be brought back to a state where Git recreates every Secret through External Secrets Operator - instead of an etcd restore.
- **Deleting a key destroys its cluster's data.** The platform vault has purge protection, a `CanNotDelete` resource lock, Azure Policy denying key expiry dates in the platform vaults, and activity and diagnostic log alerts on any key disable, delete or permission change.
- **Rebuilt stateless clusters** reuse their key names. The cluster module creates a new key version for the new cluster, and the old versions are disabled once the old cluster is deleted.
- **Revocation (crypto-shredding)** is the reason to use CMK: disabling a key makes the data under it unreadable. It takes the platform team's PIM role on the vault and is part of the incident runbook. It is rehearsed in dev.

Trade-off: cluster identities on the shared platform vault

Until now, no identity had a role on the platform vault scope except the renewal job and the platform team. KMS data encryption with a private vault needs **Key Vault Contributor on the vault** for each cluster's control plane identity (AKS reads the vault's network settings; classic KMS with a private vault needs it to create its private link). That role is management plane only: under the RBAC permission model it cannot read certificates or keys, or assign roles. It can, however, change the vault's network settings and delete the vault. The design keeps the keys in the platform vault as decided, and limits the impact:

- Purge protection on the vault, a `CanNotDelete` lock and Azure Policy denying `publicNetworkAccess` other than `Disabled` ; in production Azure Policy also requires purge protection on every vault.
- An activity log alert on every management-plane write to the platform vaults made by anything other than the environment IaC identity.
- Each cluster identity's data-plane role (Crypto User) is scoped to its own KMS key, so a cluster can never use another cluster's key.

5.1 Egress control and the company firewall

Requirements: [R4](#), [R6](#), [R29](#).

Who decides what an application may reach outside the landing zone: **the hub Firewall, in every cell**. Every cell runs Azure CNI Pod Subnet with the Cilium dataplane, so every pod keeps its own VNet address up to the Firewall ([section 2.1](#)), which decides every flow that leaves the cluster from rules written per application. Cilium decides only inside the cluster. This page states the model, why Cilium's per-application FQDN policies are not used, and how the platform works with a company-wide firewall that another team owns.

The hub Firewall decides, Cilium inside the cluster

The Firewall sees a zone's application pods by their **own addresses**, in the zone's pod subnet, and the platform's own components (gateways, ESO, Flux, external-dns) by the system subnets. Its rules name subnets or IP groups, so it opens a destination to a zone, not to one application - but its log names the pod, so every flow can be attributed. Every pod keeps its own address up to the hub Firewall (the platform configures AKS's ip-masq-agent to translate nothing), and no egress policy exists in the application repositories:

- **Cilium decides only inside the cluster**, at layers 3 and 4: the zone baseline, namespace isolation and the platform namespaces' policies ([section 2.6](#)); the baseline's DNS rule is plain port 53 to kube-dns, and a platform policy per zone (`<zone>-firewall-egress` , Flux Kustomization `firewall-egress-<zone>`) lets the zone's application namespaces out to the world. Azure Policy's `app-egress` rule ([section 2.7](#)) keeps an application's own policies inside the cluster, so the Firewall is the one decision out of it.
- **The isolation zone is the egress boundary**. The Firewall decides everything that leaves the cluster from rules per application (one rule file per application namespace: the owner, the reason, the review) whose source is the zone's IP group (its node and pod subnets in every cell): an allowed destination is open to every application of the zone, the rule is removed with the application that asked for it, and the Firewall's log names the pod. A destination is any name the application calls as a client - an API on the Internet, a PaaS endpoint, or the published name of an application in **another zone**, whose gateway the same rule reaches through the Firewall ([section 2.1](#)); zones are never connected directly.
- **An AI egress agent proposes the rules** ([section 6.1](#)): it reads the flows the Firewall denied (Log Analytics), finds the pod behind each source address, checks the destination against the application's configuration, and opens one pull request per application with the evidence. Humans of the platform and security merge it; nothing is allowed automatically.
- **Nothing in 100.64.0.0/10 leaves the landing zone**: the Firewall Policies' SNAT private ranges are the landing zone's own (the RFC 1918 ranges and 100.64.0.0/10), so every allowed flow to the Internet or the corporate network is translated to the Firewall's address.
- **Premium features on every flow**: IDPS, threat intelligence, TLS inspection and URL filtering where agreed, FQDN tags; one log for security, in Log Analytics; one enforcement point for every flow that leaves the landing zone, from any compute, that nothing inside a cluster can change.

Making the shared destinations visible. The Firewall opens a destination to the whole zone - but the per-application rule files say which application asked for it, why, and the evidence. [Section 5.2](#) generates from them,

and from the applications' own configuration, a picture per application and a picture per zone (the union the Firewall opens, and who brought each name in) - the review a zone-wide rule needs. **An application that must not share its destinations** with the rest of its zone - a regulated integration, a third-party API key that should only ever leave from one place - gets its **own zone** ([adding a country](#) adds a zone the same way).

Why not Cilium's FQDN policies

AKS offers per-application egress by name through [Advanced Container Networking Services](#) (ACNS):

`CiliumNetworkPolicy` objects with `toFQDNs`, resolved by a DNS proxy, enforced at the source node; the same add-on carries Hubble flow logs and eBPF host routing. The design does not use it:

- **Cost.** ACNS is billed per node and hour, so it scales with exactly the zone nodes the autoscaler creates on load. The hub Firewall is paid once per exposure whether the cells use it or not and is already the enforcement point for everything that is not a pod; letting it decide for the pods too adds data processing per GB and no fixed cost.
- **What it would add is small here.** Per-application granularity, which the per-application rule files and the traffic flow pages give at review time, and which a zone of its own gives at runtime; Hubble's per-flow history, where the Firewall's log already names the pod; and eBPF host routing's few percent of node CPU, which Cilium's kube-proxy replacement and a pod subnet that masquerades nothing have mostly taken already.
- **What it would cost beyond the fee.** A DNS proxy in every pod's path and a policy per application for every team to keep (a mistake shows up as a broken application); no IDPS, threat intelligence or TLS inspection in that layer; incompatibility with LocalDNS; and, for host routing, a rebuild per cell and the loss of confidential VMs, pod sandboxing, Windows nodes and the Static Egress Gateway while it is on.

The dataplane baseline of the acceptance tests ([section 0.3](#)) is checked against the thresholds at which eBPF host routing would pay off: nodes spending more than 10 % of CPU in softirq, several hundred thousand packets per second per node, or a gateway-to-pod p99 budget the baseline misses - thresholds a cell of fewer than about 100 zone nodes does not reach.

Working with a company-wide firewall

In many companies the network or security team owns the firewalls, and the platform team does not. The platform's egress rules are then **requests to that team**, in a form that team can take without learning Kubernetes: a rule file per application namespace (the zone, a contact, and per destination the FQDN, protocol, port, reason and evidence) plus the source **IP groups** the platform maintains (one per zone and one for the exposure's system subnets). Nothing in a rule refers to pods or namespaces, and the egress agent's pull requests and every human change go through the same review. The rules reach the company's firewall as a child Firewall Policy under the company's parent, on the platform's own Firewall per exposure.

The Firewall's log and its summaries

The hub Firewall writes its log to the hub's own Log Analytics workspace: one line per connection with the source address, the destination (name or address) and port, the protocol, the action and the rule that decided it, or the default action. Denials are always logged, allowed flows only while an environment learns its rules. Two kinds of consumer read it:

- **The egress agent reads the raw log** ([section 6.1](#)): fresh within minutes, one flow at a time, and it finds the pod behind a source address at the time of the flow.
- **Everything else reads summaries**: the denial detection of [section 6.2](#), the evidence of an egress review, the rule reviews and the trends per zone. Azure Monitor's [summary rules](#) aggregate the raw log on a schedule into tables in the same workspace, so no consumer scans the raw log for the same questions again and the summaries outlive its retention.

Every rule attributes a source address to its **environment and zone** from the prefixes the Firewall Policies' IP groups hold: a zone's node and pod slots in every cell, a cell's system and shared slots as `system` and `shared`, the management, VPN and hub networks by name, anything else `other`. The IaC generates the mapping with the IP groups, so a new zone or cell is attributed from its first flow. The summaries keep the source address; they never name the pod.

Rule	Bin	Table	Grouped by	Measures	Read by
<code>firewall-denied-flows</code>	1 hour	<code>FirewallDeniedFlows_CL</code>	<code>Environment</code> , <code>Zone</code> , <code>SourceIp</code> , <code>Category</code> , <code>Protocol</code> , <code>Destination</code> (the FQDN, else the address), <code>DestinationPort</code> , <code>Policy</code> , <code>RuleCollectionGroup</code> , <code>RuleCollection</code> , <code>Rule</code> , <code>ActionReason</code>	<code>Count</code> , <code>FirstSeen</code> , <code>LastSeen</code>	The denial detection per zone; an egress review's evidence (which source asked for what, and when); the zone isolation's evidence
<code>firewall-rule-hits</code>	1 hour	<code>FirewallRuleHits_CL</code>	<code>Environment</code> , <code>Zone</code> , <code>Action</code> , <code>Category</code> , <code>Policy</code> , <code>RuleCollectionGroup</code> , <code>RuleCollection</code> , <code>Rule</code> , <code>ActionReason</code>	<code>Count</code> , <code>Sources</code> , <code>Destinations</code> (distinct)	Rule hygiene: which rules each zone uses and how much; an application rule no zone has hit for 30 days is proposed for removal; which deny rules and default actions fire
<code>firewall-zone-traffic</code>	1 day	<code>FirewallZoneTraffic_CL</code>	<code>Environment</code> , <code>Zone</code> , <code>Action</code> , <code>Category</code> , <code>Protocol</code> , <code>DestinationPort</code>	<code>Count</code> , <code>Sources</code> , <code>Destinations</code>	The trend per zone and day: capacity, the Firewall's data processing cost, the weekly baseline of the denial detection, the SOC's dashboards

Retention (example, to be agreed, like [section 6.2](#)): the raw log with the SOC's retention, the hourly tables 90 days, the daily table 13 months like the metrics.

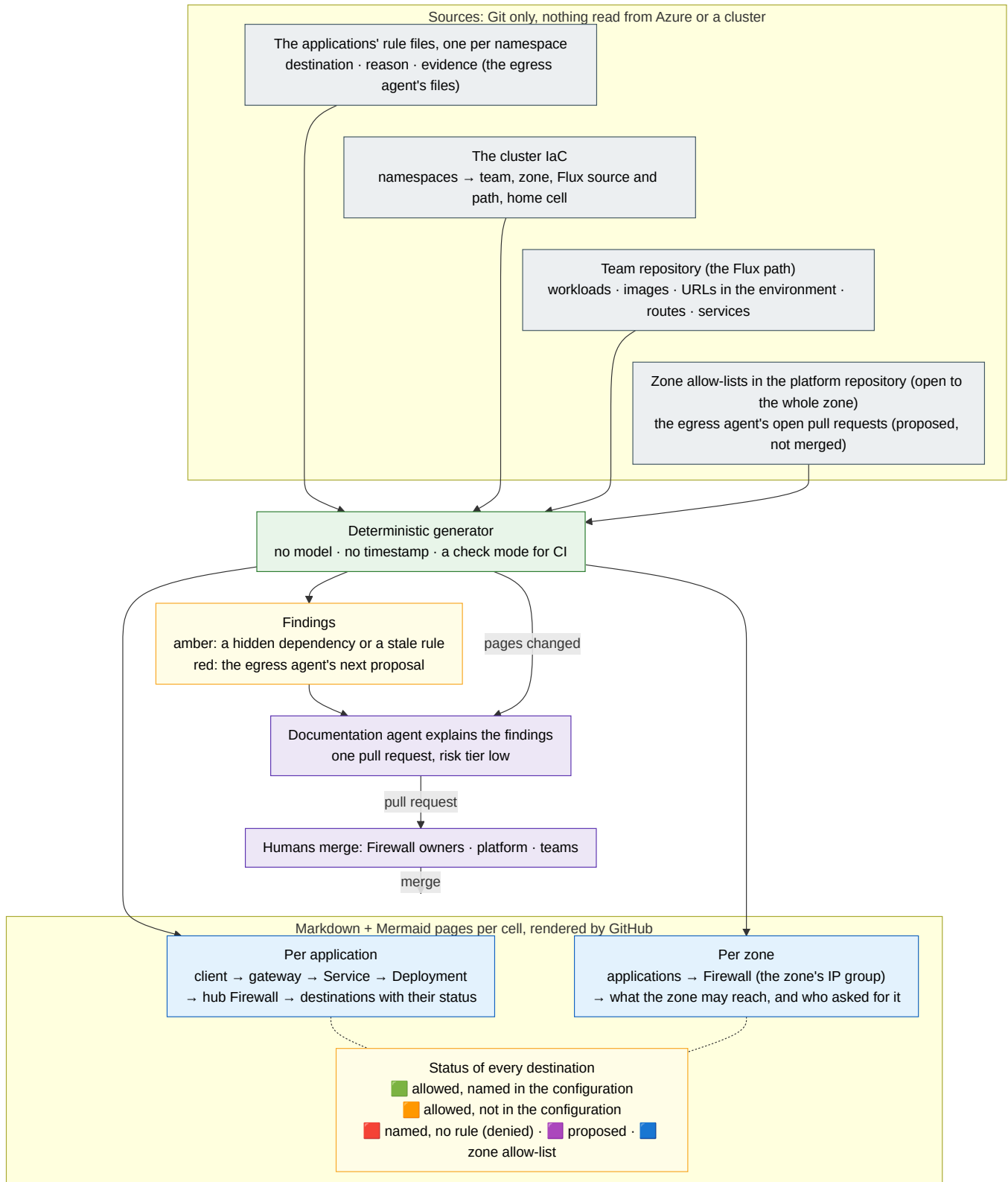
The rules are monitored like a collector. The workspace's diagnostic setting sends the rules' run log to the workspace itself, one entry per bin: started, succeeded or failed. Azure Monitor retries a failed bin ten times within

eight hours and then skips it; after eight consecutive failed bins it suspends the rule. A failed bin is a platform alert, so a gap in a summary is never read as a quiet zone.

What shapes the queries. A rule's query has no time filter (the bin is its range, and a bin delay covers the log's ingestion latency) and reads only its own workspace (no cross-workspace query), so every hub workspace, in every environment and exposure, has the same three rules. The results must be a tiny fraction of the source; they are billed as Analytics ingestion, and on the Analytics plan the query itself is free.

5.2 Traffic flow documentation: a picture per application and per zone

Requirements: [R29](#).



[Section 5.1](#) chose the hub Firewall as the egress decision of the default stateless cell. Its price is granularity: the Firewall's rules take a zone's IP group as their source, so a destination one application asked for is open to every application of its zone, and the Firewall itself can never say **which application uses a rule** - its log shows pod addresses, never a namespace. Azure has no view that answers "what does application X talk to, and why is this rule here?".

The platform can answer it, because its rules are not written at the Firewall. Every rule has a **source**: the egress agent ([section 6.1](#)) writes one file per application namespace with the destination, the reason it found in the application's configuration and the evidence (the denied flows, the pod, the manifest line), and a human merges it. Join those files with the application's own configuration in Git and the picture draws itself. That is what the **documentation agent** does.

What is documented, and from where

Everything comes from Git. Nothing is read from a cluster, the Firewall or Azure, so the documentation is the *declared* state - what is merged - and can be regenerated at any commit:

Source	Where	What it gives
The applications' rule files, one per application namespace	Next to the Firewall rules (section 5.1)	Per application: destination, port, protocol, reason , evidence , the zone, the contact, and the Firewall rule it is rendered into
The cells' and the environment's configuration	The cluster IaC	Which namespaces run in which cells, their team and contact, where their configuration is (Flux source and path), the zone and its domain, the home cell
The application's Flux Kustomization at that path	The team's repository (section 3.3)	Workloads, replicas, images, every URL in the pods' environment and arguments, routes (hostnames, gateway, backends), services - the rendered manifests
The zone allow-lists	The platform repository	Names open to every application of the zone (the Firewall's zone rule collection; the same list the pull request check reads)
The egress agent's open pull requests	The rule files' repository (optional)	Destinations proposed but not merged yet

The pictures

Two pictures, both [Mermaid](#) in Markdown pages, because GitHub renders Mermaid inline: no rendering step, no binary files in Git, a readable diff when a rule changes, and the same source the design uses.

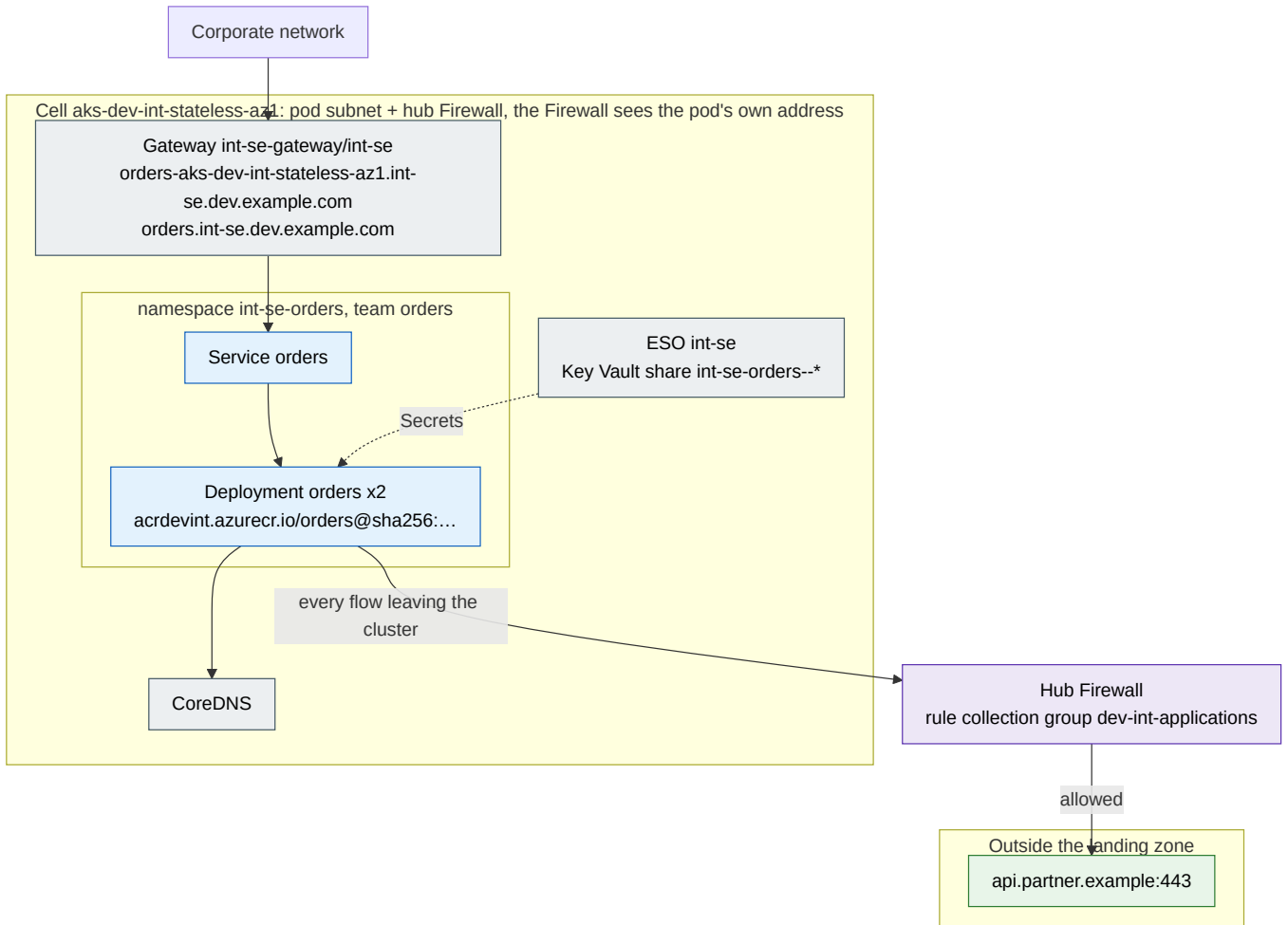
Per application (`docs/traffic-flows/<spoke>/<namespace>.md` , example below), from the client (Traffic Manager and the Internet for an external zone) to every destination outside the landing zone, with the hostnames' Flux variables substituted per cell. Each destination carries a **status**, the one fact the Firewall cannot give:

Status	Colour	Meaning	What to do
allowed	green	A rule in the application's file, and the application's configuration names the destination (a URL in its environment or arguments, or the name in its manifests)	Nothing: the rule has a source
allowed, not in the configuration	amber	A rule in the application's file that nothing in its configuration names	A hidden dependency (a library, a base image, a sidecar) to document - or a rule nobody needs any more, to remove. The agent reports it as a finding
denied: no rule	red, dashed	The configuration names a destination that no rule and no zone allow-list allows	The Firewall denies it; the egress agent proposes the rule from the denied flows, a human merges it
proposed	purple, dashed	In an open pull request of the egress agent	Review the pull request
zone allow-list	teal	On the zone's allow-list, open to every application of the zone. On an application's page only when its configuration names it; otherwise it is the zone page's business	Nothing
platform rule	grey	A rule of the platform for every cell of the exposure (the registry, Grafana Cloud). Likewise only when named	Nothing

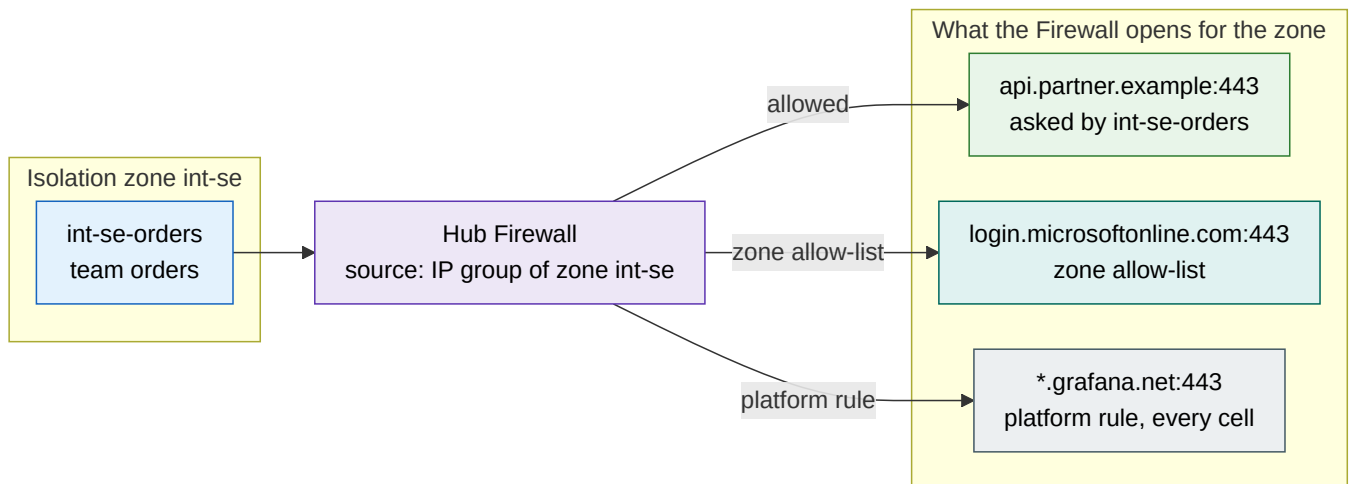
Below the picture, the page lists the same destinations with the rule's reason, where the configuration names the destination, and the rule's evidence; then the routes (hostnames per cell, gateway, backends), the workloads and notes (a cell that syncs another overlay of the same repository, a namespace whose configuration was not available).

Per zone (`docs/traffic-flows/<spoke>/zone-<zone>.md`): **the union of what the Firewall opens for the zone**, and the names applications declare that no rule allows yet. The zone page also notes when the platform repository's allow-list and the Firewall's allow-list of the zone differ - the Firewall enforces one, the pull request check reads the other. The shared names live here on purpose: an application's page shows only what the application itself asked for or names, so a reviewer sees its own footprint without the zone's noise. This is the picture section 5.1's "applications of one zone share the zone's destinations" needs: a reviewer of the next egress pull request sees what the zone already reaches and who brought each name in, and the Firewall's owners see the zone's whole exposure on one page. An index page lists the zones and applications of the spoke.

An example of each picture, for an application `int-se-orders` with one destination of its own in a cell where the Firewall decides:



The zone `int-se` - the application's rule, the zone allow-list and the platform's rules:



The documentation agent

The agent follows the rules of [section 6.1](#): read-only sources, Git as its only write path, one pull request per run, tier low (documentation). As with the egress agent, **the model is the expensive part and does only what needs judgement**:

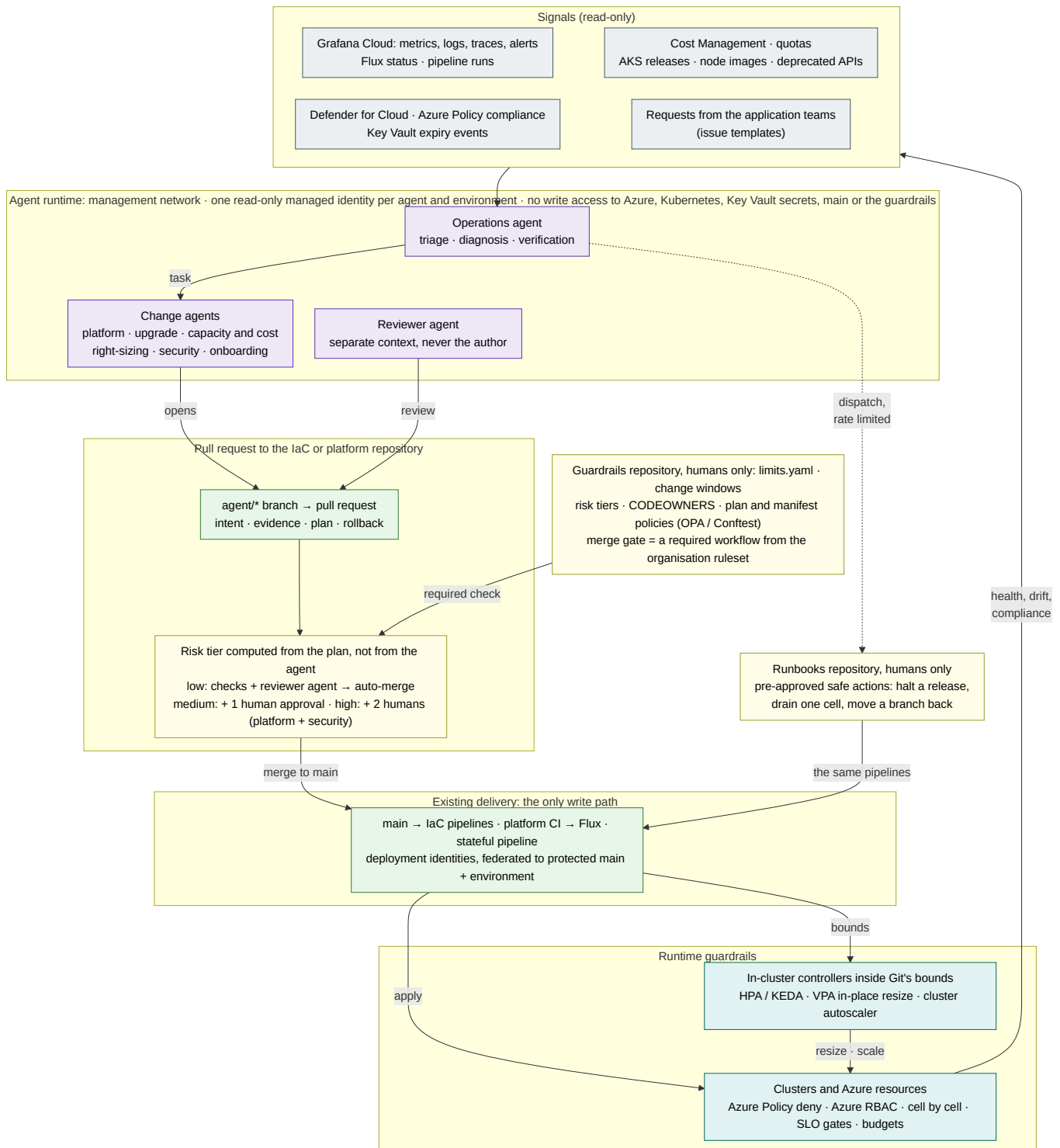
1. **Generation is code**, not a model. Without a timestamp a run that changes nothing writes nothing, and the check mode is the pull request check of the rule files' repository, so the pages never fall behind the rules.
2. **The model explains the findings**: when a page changed, the documentation agent runs the generator into its workspace, looks briefly for the explanation of each amber or red destination in the application's repository, the rule's evidence and the open issues, and opens the pull request with the pages and its findings. It changes nothing but `docs/traffic-flows/` ; its grants hold no rule file, no allow-list, no application repository. Proposing a rule stays the egress agent's job, merging it a human's.
3. **Triggers**: a merged pull request to the rule files or to an application's configuration, and a weekly schedule. The generation needs no credential and no cluster; the model step reaches the model endpoint over its private endpoint.

The pages live **next to the rules they document**, in `docs/traffic-flows/<spoke>/` of the rule files' repository: that is where the egress agent's pull requests arrive and where the Firewall's owners review ([section 5.1](#)). A company that keeps its firewall rules in the firewall team's repository keeps the pages there too; the generator takes the repositories as parameters.

The amber status is the review the pictures are for: a rule that no configuration explains is either a hidden dependency to document or a platform flow in an application's file (a telemetry agent's usage statistics reported from every node, for example), which moves to the platform's rules of the exposure. Such a removal is a pull request the documentation agent cannot open itself: a rule's removal is the egress files' owners' decision.

6.1 AI-driven day 2 operations

Requirements: [R30](#), [R33](#).



This design was written by AI, and the platform it describes is built to be operated mostly by AI too: upgrades, capacity, cost, pod right-sizing, compliance, drift, incident triage and application onboarding are run by **agents**. Agents may change every environment, prd included, but **only the way a human platform engineer would: with a pull request to the IaC or fleet repository**. Humans no longer make most of the changes; they own the **guardrails** that decide which changes can be merged and by whom.

Rules

1. **Git is the only write path:** the existing delivery ([section 3.3](#)) applies every agent change.
2. **Agents observe with read-only identities, act with pull requests,** each with the least read access its task needs and no access to application data.
3. **Guardrails are deterministic and owned by humans.** What may be merged is decided by checks, limits and approval rules in a separate guardrails repository that agents can read but not change, not by an agent's judgement – an agent cannot label its own change as low risk.
4. **Risk decides the approvals, not the environment alone.** A small change inside the limits merges automatically, also in prd; anything that touches identity, network, policy, data or the guardrails needs one or two humans.
5. **The design documents are the specification.** Agents read this repository as their context and cite the section a change follows; a change the design does not cover is a design change, made here first, by pull request.
6. **The runtime guardrails stay the last line.** Azure Policy, the admission policies of [section 2.7](#), cell-by-cell rollouts with SLO gates and automatic rollback stop a wrong change that got merged, whoever wrote it.

Agents

One agent per responsibility, each with its own identity, tool allow-list and instructions, so that a confused or manipulated agent can only do what its role allows. Agent instructions (prompts, tools, allowed repositories and paths) are kept in the guardrails repository, versioned like code.

Agent	Responsibility	Triggered by	Output
Operations agent (on-call)	Triage alerts, correlate metrics, logs, Flux and pipeline status, Azure Resource Health; diagnose; verify that a fix worked; draft the incident report	Grafana Cloud alerts (IRM webhook), Flux notifications, failed pipeline runs, schedule (daily health report)	Incident issue with diagnosis and evidence, task for a change agent, dispatch of a safe runbook action
Platform change agent	General IaC and fleet changes: fixes from incidents, platform component versions (KEDA, operators, External Secrets Operator; not the App Routing gateway, which AKS upgrades with the cluster), configuration	Task from the operations agent, issue assigned by a human	Pull request
Upgrade agent	Runs the update policy : takes the target versions the version pipeline computes (the timetable stays deterministic), reads the release notes and reports what in them changes the platform, checks the fleet manifests for removed APIs on the target minor, follows the cell-by-cell rollout and stops it on failed health	Version pipeline output, AKS release events, AKS's end-of-support notifications (Event Grid)	Version pull requests per environment and speed with a compatibility report; halt + incident on failed health

Agent	Responsibility	Triggered by	Output
Release agent	Prepares the application promotion pull requests dev → acc → prd (section 3.3), summarises the change set, watches the cell gates	Successful release in the previous environment	Promotion pull request with release notes and risk summary
Capacity and cost agent	Zone pool sizes and limits, spot share, amd64 / arm64 fit, system pool and stateful node pool sizes, quota headroom, idle resources; request and allocation efficiency (section 6.2); reservations or savings plans for the steady baseline (system pools, stateful node pools, the on-demand share that always runs), while the autoscaler and spot cover the variable part	Schedule (weekly), quota and budget alerts	Pull requests within <code>limits.yaml</code> , issues to application teams; reservation proposals as issues to the platform owner (they are purchases, not code)
Storage agent (stateful cluster only, built with it)	Finds the persistent volumes nothing uses from the <code>Unused</code> condition Kubernetes 1.37 puts on every <code>PersistentVolumeClaim</code> (<code>status: "True"</code> , reason <code>NoPodsUsingPVC</code> ; its <code>lastTransitionTime</code> is the moment the last pod left - pending pods count as use, finished Jobs do not), checks each idle claim against the owning <code>StatefulSet</code> or operator, the namespace's stateful exception and the zone's backup recovery points (section 6.3), and proposes what to do with the ones idle longer than the zone's retention: delete after a confirmed backup, keep with a reason, or move to a cheaper tier. Kubernetes deletes nothing on its own and neither does the agent; an idle claim is a signal, the owning team decides. Later input: the CSI volume health in <code>PersistentVolumeClaim.status</code> (alpha again in 1.37)	Schedule (weekly), the stateful cluster's cost alerts	Issues to the application team with the evidence (idle since, size, tier, last recovery point); a pull request removing the claim's manifest only after the team confirms - tier medium (data)
Right-sizing agent	Pod CPU and memory (pod right-sizing): the policy of the VPA that resizes pods in place, online – which applications (on by default, team opt-out), modes and <code>minAllowed</code> / <code>maxAllowed</code> bounds, HPA interplay, quota and N+1 fit – and the requests in Git new pods start from; watches resize health (<code>PodResizePending</code> , <code>OOMKills</code> , throttling). Never resizes a pod itself	Schedule (daily), task issues from the operations agent	Pull requests to the applications' <code>vpa.yaml</code> / <code>resources.yaml</code> within <code>limits.yaml</code> , issues to application teams and the capacity agent
Security and compliance agent	Azure Policy non-compliance, the security tool's recommendations and posture score trend, CIS benchmark scan results, expiring policy exemptions, image CVEs of platform components, certificate and secret expiry, policy <code>audit</code> → <code>deny</code> promotion (section 2.7), stateful-cluster drift report	Schedule (daily), security tool / Key Vault events	Pull requests for platform findings, issues to application teams for theirs
Onboarding agent	Turns an application team's request (new application, zone, identity, Key Vault secret exception, arch / capacity choice) into the IaC and fleet changes: namespace, managed identity, federated credentials, role assignments, overlays	Issue from a request template	Pull requests – always tier medium or high, because they create identities

Agent	Responsibility	Triggered by	Output
Base image agent	Rebuilds every application image on a patched platform base image from the current commit of its <code>main</code> (patching), runs its tests and the contract check, and sets the new digest for dev; reports the images and dependencies past their deadline	A new base image in the registry (a vendor patch), the registry's scan results	One pull request per application repository (its dev digest) - tier low when the tests pass; issues to the team with a date when they fail
Egress agent	Proposes each application's hub Firewall rules from the flows the Firewall denied (the raw log, not its summaries): finds the pod behind each source address, checks the destination against the application's configuration, and adds it to the application's rule file with the reason and the evidence (section 5.1)	Schedule (hourly), Firewall deny alerts	One pull request per application namespace to its rule file - tier high (a Firewall rule); never a rule it merges itself
Documentation agent (section 5.2)	Keeps the traffic flow pages current: a picture per application and per isolation zone, generated from Git by code; the model only explains the findings	Merged rule or application changes, schedule (weekly)	One pull request with the generated pages (<code>docs/traffic-flows/</code>) - tier low; never a rule
Reviewer agent	Reviews every agent pull request against this design and the evidence: does the change do what its intent says, is the rollback plan real, does it widen access; its review is a required check, never a human approval	Every pull request	Review with findings

The **reviewer agent never reviews its own work**: it runs with a separate context, its own instructions and, where possible, a different model than the author, and it only sees the pull request, the plan output and the design documents. Human pull requests get the same review.

Every agent pull request has the same structure, enforced by the merge gate: **intent** (the issue or alert it answers), **evidence** (queries and links to metrics), the **design section** it follows, the **plan** (IaC plan / what-if and rendered manifest diff, attached by CI, not written by the agent), the **rollback** and the **risk tier** computed by the gate. Commits carry trailers `Agent: <name>` and `Agent-Run: <id>` , so every line in Git leads back to the agent run and its transcript.

Repositories and the merge gate

Repository	Content	Agents	Humans
IaC repository	Environment and cluster modules, Azure Policy, identities, network, target versions	Pull requests from <code>agent/*</code> branches	Approve by tier; merge to <code>main</code> only through the merge gate

Repository	Content	Agents	Humans
Fleet repository (the platform repository and the team repositories)	In-cluster state (section 3.3)	Pull requests from <code>agent/*</code> branches	Approve by tier
Guardrails repository	<code>limits.yaml</code> , risk tier rules, plan and manifest policies (OPA / Conftest), the merge gate workflow, CODEOWNERS of the other repositories, agent instructions and tool allow-lists, the IaC of the agent identities	Read only	Two approvals (platform + security) for every change
Runbooks repository	Pre-approved safe actions as parameterised pipelines	May only dispatch them	Own and change them
This design repository	The specification	Pull requests (design changes)	Approve every change

The examples use GitHub; Azure DevOps offers the same with branch policies, required reviewers and build validation.

- **main is protected by an organisation ruleset** that no repository admin or app can bypass: pull request required, linear history, signed commits, stale approvals dismissed on push, and the most recent push approved by someone other than its author.
- **The merge gate is a required workflow defined in the guardrails repository**, so an agent cannot change the check by changing the repository it is checked in. It:
 1. waits for CI to attach the **plan** – `terraform plan` / Bicep what-if as JSON for the IaC repository, rendered manifests per cluster and environment for the fleet repository;
 2. evaluates the plan, not the diff, against the policies and `limits.yaml` (below);
 3. computes the **risk tier** from what the plan does (resource types, actions, environments), never from a label set by the author;
 4. checks the approvals the tier requires: the reviewer agent, and humans from the CODEOWNERS teams. Agents' apps are never members of those teams, so an agent approval can never count as a human one;
 5. enables auto-merge for tier low; for the others it waits for the approvals.
- **Change windows and freezes** are part of `limits.yaml`; outside a window a prd pull request waits, unless it is marked as an incident fix by a human.
- **Rate limits per agent**: open pull requests, merges per day and environment, and **one prd cell per pull request** – the gate rejects a pull request that changes more than one cell's tag or version at a time, so the cell-by-cell rule of [section 3.4](#) also holds for agents.

Risk tiers

Tier	Examples	Required to merge
Low	Patch version or node image per the update timetable; promotion of image digests that run in acc and passed all its gates; an application rebuilt on a patched base image with its tests green; HPA / KEDA bounds, VPA modes and bounds and resource-only request changes, spot share and node pool sizes and limits inside <code>limits.yaml</code> ; alert rules and dashboards; platform component patch versions; documentation	Plan, policy and CI checks green + reviewer agent; auto-merge (prd only inside a change window)
Medium	Kubernetes minor upgrade of a stateless cell; platform component minor versions; new application in an existing zone; Azure Policy definition rollout <code>audit</code> → <code>deny</code> ; changes over a <code>limits.yaml</code> soft limit; any change the reviewer agent flags	Low + 1 human from the platform team
High	Role assignments and identities (outside the onboarding pattern), federated credentials, NSG / Firewall / route / address plan, new country, Key Vault access, deletion of anything that holds data (Key Vault, ACR, storage, databases, the stateful cluster), stateful cluster version changes, Azure Policy exemptions, changes to CI / pipeline definitions	Low + 2 humans : platform and security
Not allowed for agents	Changes in the guardrails or runbooks repositories, the agents' own identities and permissions, rulesets, break-glass, disabling Azure Policy assignments	Humans only, in the guardrails repository

`limits.yaml` holds the numbers the gate checks, for example:

```

environments:
  prd:
    changeWindows: ["Mon-Thu 08:00-15:00 Europe/Helsinki"]
    freezes: ["2026-12-18/2027-01-06"]
    maxMergesPerDay: { upgrade-agent: 2, capacity-agent: 3, platform-agent: 5 }
    nodePools:
      maxNodesPerPool: { soft: 30, hard: 60 }      # over soft = medium, over hard =
rejected
      maxCpuPerZone: { soft: 960, hard: 1920 }
      maxSpotShare: 0.5
      allowedVmSizes: [Standard_D4ads_v6, Standard_D8ads_v6, Standard_D4pds_v6,
Standard_D8pds_v6]
    rightSizing:                                # VPA objects and resource-only workload
changes
    updateModes: ["Off", Initial, InPlaceOrRecreate] # stateful cluster: "Off" only
    controlledValues: [RequestsOnly]
    maxAllowed: { cpu: { soft: 4, hard: 16 }, memory: { soft: 16Gi, hard: 64Gi } } # per
container
    maxStepFactor: 2                            # a request or limit moves at most ×2
per pull request
  kubernetes:
    allowedMinors: { stateless: ["1.36"], stateful: ["1.36"] } # 1.36 is the floor; newer
minors added by the upgrade agent's PR, approved per tier
  forbidden:
    - action: delete
      types: [Microsoft.KeyVault/vaults, Microsoft.ContainerRegistry/registries,
Microsoft.Storage/storageAccounts, Microsoft.Sql/servers]
    - action: roleAssignment
      roles: [Owner, User Access Administrator, Role Based Access Control Administrator]

```

Policies in the gate cover what Azure Policy cannot see before deployment: deletions and replacements in the plan (a “replace” of a stateful resource is treated as a delete), blast radius (number of resources, number of cells), address spaces outside the IPAM allocation, Firewall rules to `0.0.0.0/0` or `*`, role assignments wider than a zone, and the rendered manifests against the same constraint templates as the clusters.

Identities and access rights

Agents run as container jobs in the **management VNet** (management subscription), the same network as the pipeline agents; their only egress is the model endpoint (private endpoint), the Git platform and an allow-list of documentation sites, through the internal hub's Firewall (the management VNet is peered to both hubs). Each agent has, **per environment, its own user-assigned managed identity** with workload identity federation (no client secrets) and its **own GitHub App** with short-lived installation tokens. The identities and their role assignments are IaC in the guardrails repository.

Agents reach Azure and the clusters through **tools, not shells**: read-only wrappers around `az` and `kubectl` with a read-verb allow-list, or a tool server such as the [AKS-MCP server](#) run with its read-only access level and the agent's own identity, next to tools for Grafana Cloud and Git. The tool server is a convenience and an audit point,

not the security boundary: the read-only roles below are, so a tool server configured with a write level could still change nothing.

Azure (per environment subscription, read only)

Agent	Azure roles	Never
All agents	<code>Reader</code> on the environment subscription; <code>Monitoring Reader</code> ; a Grafana Cloud <code>Viewer</code> service account per agent and environment, which sees all telemetry of the stack like every Grafana user (section 6.2)	Any write role, <code>Contributor</code> , <code>Owner</code> , <code>User Access Administrator</code> , PIM eligibility; any role on the backup subscription (section 6.3)
Operations, platform, upgrade, release, right-sizing, security	<code>Azure Kubernetes Service RBAC Reader</code> on the clusters (does not include <code>Secrets</code>); read access to the Flux and pipeline status	<code>Azure Kubernetes Service RBAC Writer / Cluster Admin</code> , <code>listClusterAdminCredential</code>
Capacity and cost	<code>Cost Management Reader</code> ; quota read	—
Security and compliance	<code>Security Reader</code> ; <code>Key Vault Reader</code> on the zone and platform vaults (metadata and expiry dates, not values)	<code>Key Vault Secrets User / Officer</code> , <code>Key Vault Certificate User</code>
Onboarding, reviewer	<code>Reader</code> only	—

No agent has a data-plane role on the PaaS services or the ACR: they do not need application data to run the platform, and a prompt injection through a log line or an issue then cannot exfiltrate it. Agents do read application logs and traces in Grafana Cloud, and the [rule of what must not be sent](#) keeps secrets out of them; their outbound access stays limited to the model provider and GitHub. Only the guardrails pipeline may create role assignments for agent identities; any other role assignment to one of them triggers an Activity Log alert, and a daily comparison against the guardrails IaC reports and removes it.

Git platform

Agent	GitHub App permissions	Repositories
Change agents (platform, upgrade, release, capacity, right-sizing, security, onboarding)	<code>contents: write</code> only on <code>agent/<name>/*</code> <code>branches (ruleset)</code> , <code>pull_requests: write</code> , <code>issues: write</code>	IaC, fleet, design
Operations agent	<code>issues: write</code> ; <code>actions: write</code>	Issues in IaC and fleet; dispatch in the runbooks repository only
Reviewer agent	<code>pull_requests: write</code> (reviews), <code>checks: write</code>	IaC, fleet, design
All	<code>contents: read</code>	Guardrails, runbooks

Deployment identities (unchanged, not agents): the existing IaC pipeline, Flux and stateful pipeline identities are the only ones with write access. Their federated credentials accept only tokens for the protected `main` branch and

the GitHub environment of the target (`repo:<org>/<repo>:environment:prd`), so a pull request branch – an agent's or a human's – can never deploy.

Humans: platform and security engineers approve pull requests and own the guardrails; their Azure access stays read-only plus break-glass via PIM ([section 3.2](#)). A break-glass action is followed by a pull request that brings Git back in line, opened by the operations agent from the drift it detects.

Safe actions without a pull request

Some incidents cannot wait for a pull request, but all of them are in the **safe direction**: stopping, or moving traffic away from a failing cell, not starting. They are parameterised pipelines in the runbooks repository, run with the existing deployment identities; the operations agent may only dispatch them:

Action	Limits enforced by the runbook
Halt a running release or upgrade	Always allowed
Drain one stateless cell (traffic weight 0 at the cell router)	Never the last cell in traffic, at most one cell per environment, alert to the platform on-call
Fail over the internal applications of a failed cell (also dispatched by the cell health alert)	Only when the cell is unhealthy, only to a cell in traffic, through the migration workflow (the only change it commits without a pull request is the placement of the affected applications), alert to the platform on-call
Roll a cell back to its previous commit	Only to the commit its branch had before the current release
Suspend a Flux Kustomization of an application	Application namespaces only, never <code>infra-*</code>

Anything else, including undoing these actions, is a pull request. Every dispatch pages the human on-call, who can always reverse it.

Guardrails during deployment and at runtime

The pull request is the first guardrail, not the only one:

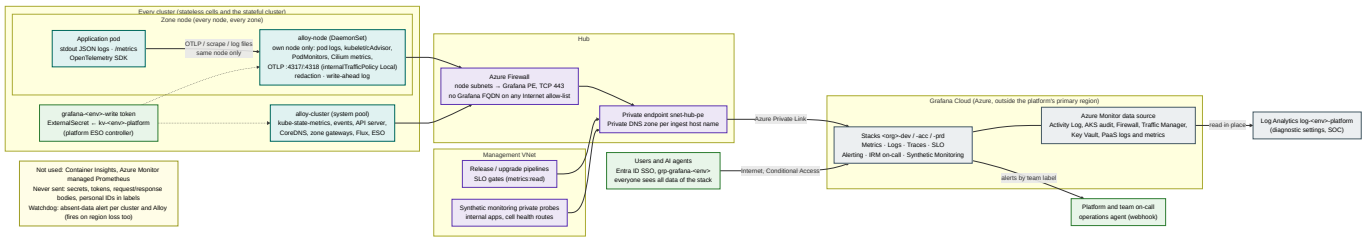
- **Deployment:** the cell-by-cell releases, SLO gates, soak times and automatic rollback of [section 3.4](#) and [section 3.5](#) apply to every merged change; the stateful pipeline keeps its `kubectl diff` and approval stage.
- **Feedback:** the operations agent verifies every merged change against the evidence in its pull request (did the alert clear, did cost drop) and reports the result on the pull request, so that humans can judge the agents by outcome.

Audit and kill switch

- Agent runs (instructions version, inputs, tool calls, outputs) are stored in an immutable storage account in the management subscription for a year; Git, the GitHub audit log and the Azure Activity Log link to them through the run ID.
- **Kill switch:** suspending an agent's GitHub App installation stops its pull requests; `agents.enabled: false` in the guardrails repository makes the merge gate reject every agent pull request; removing the federated credentials stops its Azure reads. The platform keeps running – it is just changed by humans again.
- **Start small:** agents begin with a human on every tier above low in dev and above medium in acc and prd (`minimumTier` in the guardrails), and tiers are relaxed per agent and environment based on the measured outcome (merged without rework, rollbacks, incidents caused).

6.2 Observability with Grafana Cloud

Requirements: [R14](#), [R24](#), [R31](#).



All telemetry goes to [Grafana Cloud](#); the platform runs only the collector, [Grafana Alloy](#).

Access rule: anyone who may sign in to a stack may see all of its data. Data inside a stack is not separated by isolation zone, country or application. This is a deliberate decision. The isolation zones ([R1–R4](#)) still separate the workloads, networks, nodes and secrets. Telemetry is not separated, because a cross-zone view is what makes diagnosing problems fast. The access rule decides who gets into a stack, and therefore what goes into it ([below](#)).

Stacks and access

Stack	Holds	Who can sign in
<code><org>-dev</code> , <code><org>-acc</code> , <code><org>-prd</code>	All telemetry of that environment: every cluster, cell, zone and application, plus the Azure Monitor data source of the environment subscription	Members of <code>grp-grafana-<env></code> (platform team, application teams, SOC, AI agents' service accounts). There are no per-team folders with restricted data and no label-based access policies

- **Sign-in with Entra ID** (SAML single sign-on) through one enterprise application per stack. A user's Grafana role comes from a group: `Viewer` for everyone in `grp-grafana-<env>` , `Editor` for the platform team in dev and acc, and `Admin` only through a PIM-activated group. Local Grafana users and basic auth are disabled. Grafana Cloud organisation administration (stacks, billing, access policies) also goes through PIM.
- **Machine access uses access policies** with the narrowest scopes:

Token	Scopes	Stored in	Used by
<code>grafana-<env>-write</code>	Fleet Management, <code>metrics:write</code> , <code>logs:write</code> , <code>traces:write</code> on its own stack	<code>kv-<env>-platform</code>	Alloy in every cluster of the environment
<code>grafana-<env>-opencost</code>	<code>metrics:read</code> , limited by a label-based access policy to the cluster's own series	<code>kv-<env>-platform</code>	OpenCost in every cluster of the environment

Token	Scopes	Stored in	Used by
<code>grafana- <env>-read- <agent></code>	Viewer service account	The agent's own secret store	One per AI agent and environment (section 6.1)
<code>grafana- <env>-gate</code>	<code>metrics:read</code>	GitHub environment secret	Release and upgrade pipelines (SLO gates)
<code>grafana- <env>-iac</code>	Stack administration	GitHub environment secret, <code>main</code> only	The Terraform pipeline that applies dashboards, alerts and SLOs

Tokens expire after 90 days and are rotated by the IaC pipeline. The only read token in a cluster is OpenCost's, limited by a label-based access policy to the cluster's own series (`cluster="<cluster>"`), so a compromised cluster cannot read another cluster's telemetry through it.

- **Everything is code.** Data sources, folders, dashboards, alert rules, contact points, notification policies, SLOs, synthetic checks and IRM schedules are managed by the `grafana` Terraform provider. The platform's resources live in the IaC repository. An application team's dashboards, alerts and SLOs live in `apps/<app>/observability/` in the fleet repository, and the same pipeline applies them, `dev` → `acc` → `prd`. Changes made in the UI are allowed in `dev` only, and the next apply overwrites them.

Collection: Grafana Alloy, node-local

The `k8s-monitoring` Helm chart installs Alloy in the platform namespace `infra-observability`, configured from the stack with **Grafana Cloud Fleet Management** (remote configuration): the chart's values only start the collectors and point them at Fleet Management; their pipelines – what is collected, how it is labelled and where it goes – are managed there by the platform team. Flux installs it in the stateless clusters and the pipeline in the stateful cluster, from the platform repository ([section 3.3](#)). **Every collector works only on its own node**, so telemetry never crosses from one isolation zone to another inside the cluster. A zone's data leaves the cluster from that zone's own nodes, through the Firewall rule of that zone:

Alloy instance	Runs on	Collects
DaemonSet <code>alloy-daemon</code> (the chart's one collector, clustered)	Every node (tolerates every taint, including <code>spot</code> ; <code>system-node-critical</code> , requests 100m / 256Mi)	What its Fleet Management pipelines say: pod logs from <code>/var/log/pods</code> (<code>read-only hostPath</code>); node journal; kubelet and cAdvisor metrics of its node; <code>PodMonitor</code> / <code>ServiceMonitor</code> targets on its own node only (discovery filtered by <code>spec.nodeName</code>); Cilium metrics; OTLP traces, metrics and logs pushed by the pods of its node
The cluster-level targets, taken by one member of the cluster	System pool	kube-state-metrics, node-exporter and Beyla's Kubernetes cache (separate Deployments of the chart), Kubernetes events, API server and CoreDNS metrics, Flux and External Secrets Operator metrics, the zone gateways' metrics

- **OTLP endpoint for applications:** Service `alloy-otlp.infra-observability.svc`, ports 4317 (gRPC) and 4318 (HTTP), with `internalTrafficPolicy: Local`, so a pod always reaches the Alloy on its own node. An Azure Policy mutation injects `OTEL_EXPORTER_OTLP_ENDPOINT` and `OTEL_RESOURCE_ATTRIBUTES` (`deployment.environment`, `k8s.cluster.name`, `platform.zone`) into every application container that does not set them itself. The platform's Cilium baseline allows every application pod to reach `alloy-daemon` on these two ports, and `alloy-daemon` to reach the declared metrics port of the pods on its node, the same way it allows DNS. An application's own egress policy needs no entry for telemetry.
- **Common labels** on every series, log line and span: `env`, `cluster`, `cell` (`sl-az<N>` or `sf`), `zone` (from the namespace's `platform/zone` label), `namespace`, `app` and `team` (from the namespace labels set at onboarding). Alerts are routed and costs are attributed by these labels, not by access.
- **Platform rules for Alloy:** Alloy and node-exporter run as Grafana's Fleet Management presets set them – privileged, on the host network and PID namespace, with the node's logs, cgroups and storage – so Alloy can instrument applications with eBPF (Beyla) and read the node. Being on the host network, they are outside Cilium's pod policies; the namespace's other pods (kube-state-metrics, Beyla's cache, OpenCost) keep a platform-owned policy. `infra-observability` is a platform namespace with a platform-owned egress policy `observability` (the Grafana Cloud ingest and Fleet Management host names, `prices.azure.com` for OpenCost, the API server's name and the kubelets; layer 4 to `world`, the Firewall deciding by name).
- **Write token:** an `ExternalSecret` `grafana-cloud` in `flux-system` reads the secret `flux-system--grafana-cloud` (the Fleet Management URL, user and token as one JSON) from the platform vault through `flux-system`'s own prefix share, served by the platform's External Secrets Operator in `external-secrets` (section 2.5); it lives in `flux-system` because the HelmRelease's `valuesFrom` reads Secrets of its own namespace only.
- **Buffering:** Alloy keeps a write-ahead log on the node (an `emptyDir`), so a short Grafana Cloud or Private Link outage loses no data. A longer outage drops the oldest data first, and it never blocks the applications.
- **OpenCost,** part of the same chart, prices every node from Azure's retail prices (spot nodes at the spot price) and splits each cell's cost by namespace and pod; kube-state-metrics exports the namespaces' `platform/team`, `platform/cost-center` and `platform/zone` labels, so cost is reported per team and cost centre (section 2.6). OpenCost reads the cluster's usage back from the stack's Prometheus API with a token of its own that has only `metrics:read`.
- **Fleet Management matches its pipelines to the collector's `cluster` attribute,** the cluster's name, so a renamed or rebuilt cell keeps its name or gets its matcher updated with it; otherwise it gets only Grafana's default pipelines.
- **Not used:** Container Insights, the AKS cost analysis add-on and Azure Monitor managed Prometheus, so nothing is collected twice. The company's security tool is security, not observability, and keeps its own agents.

Network: Grafana Cloud through Private Link

Telemetry never goes over the Internet. This matters most for the stateful cluster, which has no Internet at all (R19).

- The stacks are hosted on Azure, and Grafana Cloud accepts metrics, logs, traces and OTLP through [Azure Private Link](#). The ingest host names are shared by all stacks in the same Grafana Cloud region, so there is

one private endpoint per hub (`snet-hub-pe` of the internal and of the external hub) for all environments of that exposure. The stacks stay apart by tenant ID and token.

- **DNS:** for each ingest host name, the hub has a Private DNS zone named after the full host name (for example `prometheus-prod-<n>-prod-<region>.grafana.net`), with an apex A record pointing to the private endpoint. Only the ingest and query endpoints resolve privately. The Grafana UI (`<org>-<env>.grafana.net`) still resolves publicly.
- **Firewall:** one network rule per environment and exposure on the exposure's Firewall, and the same for the management VNet.
- **The stacks live outside the platform's region:** they are hosted in an Azure region other than the primary one (preferably the paired region of [section 6.3](#)). The private endpoint reaches them across regions, so losing the primary region does not take observability, its history or the alerting with it.
- **Users** sign in only with Entra ID, under a Conditional Access policy (compliant device, MFA) on each stack's enterprise application.

What Grafana Cloud does

Capability	Grafana Cloud feature	Design rule
Metrics	Grafana Cloud Metrics (Mimir), Prometheus API	Platform dashboards per environment, cell, zone and application, the same in every environment. Cardinality limits per application, enforced by Alloy relabelling (labels such as <code>user_id</code> or a request path with IDs are dropped)
Logs	Grafana Cloud Logs (Loki)	Applications log to stdout as JSON. The zone gateways' Envoy access logs are on by default; their format cannot be customised. Kubernetes audit logs stay in Log Analytics (below)
Traces	Grafana Cloud Traces (Tempo), OpenTelemetry	Applications use the OpenTelemetry SDK with W3C <code>traceparent</code> . The zone's gateway (Envoy, which supports W3C <code>traceparent</code>) creates the first span of every request. Head sampling is set in the SDK: 100 % in dev and acc, a default of 10 % in prd that applications may raise. Errors are always kept, and logs carry <code>trace_id</code> so that logs, metrics (exemplars) and traces link to each other
SLOs	Grafana SLO	Every <code>HTTPRoute</code> gets two default SLOs from the gateway's Envoy metrics (Istio standard metrics <code>istio_requests_total</code> and request duration), without any application code: availability (share of non-5xx responses) and latency (share of requests under the application's threshold). Teams set the objectives in <code>apps/<app>/observability/slo.yaml</code> and may add their own SLIs
Alerting and on-call	Grafana Alerting, Grafana IRM	Multi-window, multi-burn-rate alerts on every SLO. Notification policies route by <code>team</code> : platform alerts go to the platform on-call, application alerts to the application team's escalation chain. Every alert links to its runbook and dashboard
Synthetic monitoring	Grafana Synthetic Monitoring	Public probes for every external application's Traffic Manager host name and its per-cell host names. Private probes run in the management VNet against every internal application and against the cells' gateway health routes; they drive the cell health alert (cell failure)

Capability	Grafana Cloud feature	Design rule
Cost and usage	Usage attribution by label, Adaptive Metrics and Adaptive Logs	Usage is reported per <code>team</code> and <code>zone</code> . The capacity and cost agent (section 6.1) proposes dropping unused metrics. Compute efficiency is shown per zone and team with two ratios: request efficiency (usage ÷ requests) and allocation efficiency (requests ÷ node allocatable). Either one alone hides waste: right-sized pods can still pack badly onto the chosen VM sizes

SLO gates. Releases ([section 3.4](#)) and cluster upgrades ([section 3.5](#)) are gated on these SLOs, per cell. The pipeline queries the burn rate of the cell's applications from the stack's Prometheus API, with the `grafana-<env>-gate` token, through the private endpoint. A missing signal counts as a failed gate: no data means no promotion.

Watchdog. The absent-data alert pages the platform on-call, so a broken collector is never mistaken for a quiet system.

Azure Monitor data, read in place

Some data is produced by Azure, not by the clusters. It stays where the SOC and its security tool expect it, in the environment's Log Analytics workspace (`log-<env>-platform`), through diagnostic settings that Azure Policy enforces:

- Activity Log; AKS resource logs (`kube-audit-admin` and `guard` , more categories as a setting of the cluster IaC) into the environment's workspace `log-<env>-platform` (1 GB a day, no public access); Traffic Manager, Key Vault, private endpoint and PaaS diagnostic logs; Azure Policy compliance. The Firewall's log goes to the hub's own workspace: the egress agent reads it raw, every other consumer its [summaries](#).
- Grafana reads them, and Azure Monitor platform metrics (Firewall, Traffic Manager endpoint health, PaaS), through the **Azure Monitor data source**. It uses an app registration per environment with `Monitoring Reader` and `Log Analytics Reader` on the environment subscription, a certificate credential, and rotation by IaC. The access rule applies here too: every stack user can query these logs from Grafana.
- The security tool's cloud and runtime alerts go to the SOC tooling as before, and are also shown in Grafana.

Security detections and forensics

Collecting everything is not the same as seeing what matters. Next to the SLO alerts, the platform defines what must **never happen** in it. Each detection has an owner who answers it:

Detection (source)	Owner
A role assignment, PIM activation or federated credential change outside the IaC, guardrails and onboarding pipelines (Activity Log, Entra ID audit log)	Security team
Any write to a production resource by an identity that is not a deployment identity, or a human sign-in to a cluster outside break-glass (Activity Log, <code>kube-audit-admin</code>)	Platform on-call

Detection (source)	Owner
A key, vault, backup vault or Resource Guard change (section 4.2 , section 6.3)	Security team
A workload identity used from an unexpected cluster or IP, or reading resources outside its zone (Entra ID sign-in logs)	Security team, application team informed
Rising Cilium policy drops in a cell, or Firewall denials from a zone (the hourly denied-flows summary), in acc / prd (section 5.1)	Application team, platform on-call
The security tool's runtime alerts (reverse shell, crypto mining, unexpected binaries)	SOC, platform on-call

The detections are Grafana alert rules on the Log Analytics data source and Sentinel analytics rules in the SOC tooling, kept as code. A detection that keeps firing without action is fixed or removed, so that the remaining ones are trusted.

Forensics. Metrics, logs and flow logs show symptoms; during an incident the platform team sometimes needs the kernel's view: which process opened a socket, what a container executed. A kernel-level tracing tool ([Inspektor Gadget](#), eBPF-based and Kubernetes-aware) is deployed **only during an incident**, through break-glass, as a privileged DaemonSet in a platform namespace. It is not installed by default, because a privileged, cluster-wide tracer is itself a high-value target.

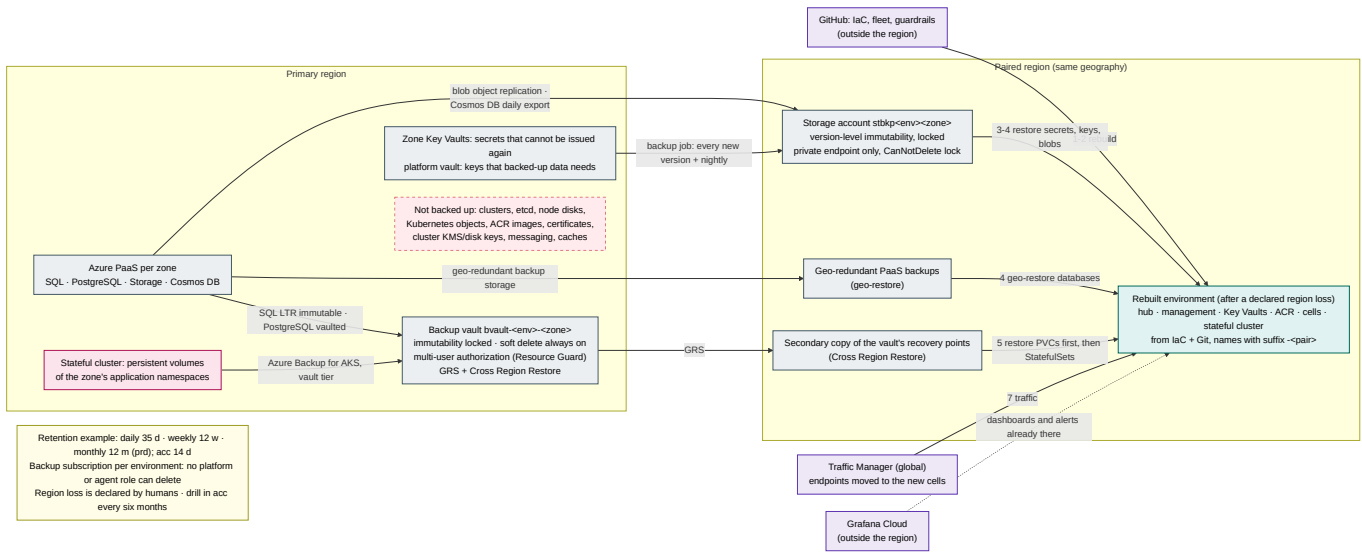
What must not be sent

Because every stack user sees all data, **nothing may be sent that a stack user is not allowed to see**:

- **No secrets, tokens or credentials** in logs, span attributes or metric labels. Alloy drops `Authorization`, `Cookie` and `Set-Cookie` headers and redacts known token patterns (JWTs, connection strings, Key Vault secret values) before sending. This is a safety net; the application is responsible for not logging them.
- **Personal data only where the stack's users may see it**: no request or response bodies, and no personal identifiers in metric labels. Logs may hold the personal data that the operations staff of every country are allowed to process.
- **Retention** (example, to be agreed): metrics 13 months; logs 30 days in prd and 14 days in dev and acc; traces 14 days. Kubernetes audit logs and Firewall logs stay in Log Analytics with the SOC's retention; the Firewall log's [summaries](#) are kept longer.

6.3 Backup and region recovery

Requirements: [R32](#).



The platform survives the loss of an availability zone with its cells and zone-redundant services ([section 3.1](#)). To survive the **loss of the whole Azure region, every piece of persistent data is backed up immutably into the region's Azure paired region**. Nobody can change or delete those copies before their retention ends: not the platform team, not an `Owner`, not a compromised pipeline or agent.

Only persistent data is backed up, never clusters ([section 3.2: clusters are cattle](#)).

Backup is built with the stateful cluster and stays open until then on purpose: a stateless cell holds no data and is rebuilt from IaC and Git in a disaster, and the PaaS pattern below is applied with the first data store an application onboards.

What is backed up and what is rebuilt

	Backed up (immutable, paired region)	Rebuilt from IaC and Git
Application data in Azure PaaS	Azure SQL, PostgreSQL, Storage (blobs), Cosmos DB (below)	—
Stateful cluster	Persistent volumes only (Azure Backup for AKS, vault tier)	The cluster, its node pools, StorageClasses, operators and every manifest (pipeline from the fleet repository)
Stateless clusters	— (they hold no data)	Everything

	Backed up (immutable, paired region)	Rebuilt from IaC and Git
Zone Key Vaults kv- <env>-<zone>	Secrets that cannot be issued again (third-party API keys, partner credentials)	The vaults, their role assignments and ABAC conditions; secrets the platform can issue again
Platform Key Vault kv- <env>-platform	Only customer-managed keys that backed-up data depends on (for example a PaaS service's TDE key, if one is used)	Certificates (issued again by the renewal job), the cluster KMS and disk keys (new clusters get new keys, section 4.2), the Grafana tokens
ACR	prd: the registry is geo-replicated to the paired region (ACR Premium, prd only), so every deployed digest stays valid and image builds leave the critical path of the prd recovery	dev and acc (Basic): images are built again by CI from the Git commits that the fleet repository deploys
Messaging (Service Bus, Event Hubs) and caches (Redis)	– (in-flight data and caches, not records)	Namespaces, queues, topics and caches
Hub, management VNet, Firewall, Traffic Manager, DNS, identities, policies	–	Everything (IaC repository and guardrails repository)
Git, GitHub environments, Grafana Cloud	Not in the region (section 6.2)	–

Every data store is named at onboarding. The onboarding request lists each data store of the application with its type, the backup mechanism from the table below, the RPO and RTO the business expects of it and the team that owns its restore; the onboarding agent turns the list into the Azure Policy-checked backup posture and a row in the environment's dependency inventory in Grafana. An application team that keeps data anywhere else must name it there too. That data is either moved into one of the data stores below or backed up by an export job into the zone's immutable backup account.

Where the backups go

Per environment there is a **backup subscription** sub-bkp-<env>. It sits in its own management group, and only the backup administrators group (not the platform team) has roles on it. Inside it, everything is **per isolation zone**, so the copies of different countries stay apart like the data itself ([R2](#)):

Resource (per zone)	Region	Settings
Backup vault bvault-<env>-<zone>	Primary (Azure Backup requires the vault in the data source's region)	Geo-redundant storage with Cross Region Restore , so every recovery point is also available in the paired region; immutability enabled and locked (irreversible); soft delete always on, 14 days; multi-user authorization through a Resource Guard in the security team's subscription

Resource (per zone)	Region	Settings
Storage account stbkp<env> <zone>	Paired region	Blob versioning; version-level immutability with a locked time-based retention policy ; no shared key access; public network access disabled, private endpoint only; lifecycle rule deletes versions after their retention ends; CanNotDelete lock

- **Immutable and locked** means a recovery point or blob version can be neither changed nor deleted before its retention ends, and the policy itself can no longer be weakened. Lowering a retention, disabling soft delete or removing protection on a vault also needs a second person's approval through the Resource Guard.
- **No platform or agent identity can delete.** The vaults' managed identities get the roles Azure Backup needs on each data source they protect, and nothing else. The backup subscription grants the platform's IaC identity only Backup operator (start a backup or restore, never delete), and grants the AI agents nothing (section 6.1).
- **Retention** (the picture's example, to be agreed per data class): acc keeps prd's layout so the procedures can be rehearsed there; dev has none, because it holds synthetic data only.
- **Keys of the backups:** the backup vaults and the backup storage accounts use **Microsoft-managed keys with infrastructure encryption**, not customer-managed keys, so a Cross Region Restore never depends on a key whose region is lost; the data they hold is CMK-encrypted at its source.

Per data store

Data store	How it is protected	Immutable copy in the paired region	RPO (region loss)
Azure SQL Database	Built-in backups on geo-redundant backup storage (point-in-time restore, 35 days), plus long-term retention (weekly, monthly)	LTR backups with time-based immutability, locked (backup immutability) ; geo-restore of the point-in-time backups to the paired region	About 1 h through geo-restore; one week if only the immutable LTR copy is left
Azure Database for PostgreSQL flexible server	Built-in geo-redundant backup (geo-restore to the paired region), plus Azure Backup vaulted backup into the zone's backup vault (daily)	The vault's recovery points (locked immutable vault, Cross Region Restore), restored as files and loaded with pg_restore	About 1 h through geo-restore; 24 h from the vault
Azure Storage (blobs)	Object replication from every application storage account into a container of the zone's stbkp<env> <zone>	The destination versions, under locked version-level immutability: a delete or overwrite at the source never reaches the copy	Minutes (asynchronous replication)
Azure Cosmos DB	Continuous backup (point-in-time restore, 30 days) for mistakes inside the region, plus a daily export by the application's backup job into stbkp<env><zone>	The exports in the immutable account	24 h from the export

Data store	How it is protected	Immutable copy in the paired region	RPO (region loss)
Stateful cluster: persistent volumes	Azure Backup for AKS with the vault tier , one backup instance per isolation zone, scoped to the zone's application namespaces and their volumes . Cluster-scoped resources are excluded; everything else comes from Git	Vault tier recovery points in the locked immutable vault, with Cross Region Restore	Up to 36 h (one daily vault-tier point, plus up to 12 h of geo-replication)
Application-consistent database backups in the stateful cluster	The operator's own backup (for example a database operator's base backup and WAL archive) writes to the zone's <code>stbkp<env><zone></code>	Immutable blobs	The operator's archive interval
Key Vault secrets and keys	A backup job (<code>az keyvault secret backup / key backup</code>) on every new version (Event Grid <code>SecretNewVersionCreated</code>) and nightly in full	Encrypted backup blobs in <code>stbkp<env><zone></code> . They can be restored only into a vault in the same subscription and the same Azure geography , which the paired region must therefore be (true for the regions considered here)	Minutes; 24 h at worst

- **Volume snapshots are not enough.** AKS volume snapshots and the operational tier stay in the primary region and may be deleted with it, so only the vault tier counts for region recovery. AKS backups are crash-consistent, so a database inside the stateful cluster also needs its operator's own application-consistent backup.
- **Customer-managed keys.** A restored disk or database is encrypted again under the paired region's new keys ([section 4.2](#)). If backed-up data is encrypted under a key that the restore needs (for example SQL TDE with a customer-managed key), that key is in the Key Vault backup above and is restored first.
- **Azure Policy enforces** the backup posture: every SQL server, PostgreSQL server and Cosmos DB account has geo-redundant backup storage and its LTR / vault policy; every application storage account has versioning and an object replication policy to its zone's backup account; every backup vault is locked immutable with Cross Region Restore. A data store without protection is non-compliant and blocks the promotion of its application to prd.
- **Monitoring:** failed backup jobs, replication lag and the age of the newest recovery point per data store are alerts in Grafana Cloud ([section 6.2](#)).

Region recovery

A region loss is **declared by humans**: the platform lead together with the business owner, following the incident runbook. No agent and no automation may start a region recovery ([safe actions](#)). The recovery rebuilds the environment in the paired region, prd first:

1. **Bootstrap:** GitHub-hosted runners with OIDC federation to the environment subscriptions run the IaC with `region=<pair>` . They create the two hubs, the management VNet with its pipeline agents, and the

corporate connectivity. For prd the ExpressRoute circuit (or VPN) in the paired region is **provisioned in advance** and only connected in the recovery, because ordering a circuit takes days that no prd RTO can absorb; dev and acc build theirs in the recovery. Once the agents are up, the pipelines run from the management VNet as before.

2. **Environment module:** the zone and platform Key Vaults, managed identities, Azure Policy assignments, backend spoke, ACR and private endpoints. Names that must be globally unique (Key Vaults, storage accounts, ACR, SQL and PostgreSQL servers) get the region suffix `-<pair>`, because the originals still exist in the lost region - the naming policy's `{location_code}` does that for the platform's resources once its templates carry it. Applications get their names from the overlays and `cluster-settings`, so this is a variable change, not a code change.
 3. **Secrets and keys:** restore the Key Vault backups into the new zone vaults, and the needed keys into the new platform vault. The renewal job issues the certificates again.
 4. **Data**, per zone, in parallel: geo-restore (or restore from the immutable copy) the SQL and PostgreSQL servers, copy the blobs from `stbkp<env><zone>` into new application storage accounts, and restore Cosmos DB.
 5. **Clusters:** create the stateless cells (Flux bootstrap) and, if one exists, the stateful cluster. For the stateful cluster, **restore the persistent volumes first** (Cross Region Restore of the namespaces' volumes, restoring only `PersistentVolumeClaims` and their volumes, with the new zone `StorageClasses`). The pipeline then deploys the `StatefulSets`, which bind the restored claims by name.
 6. **Images:** prd's ACR replica in the paired region already holds every deployed digest, so the cells start their applications unchanged. In dev and acc, CI builds the images of the commits that the fleet repository deploys, pushes them to the new ACR, and the release agent opens the pull request that moves the digests.
 7. **Traffic:** Traffic Manager is global and survives; the IaC points its endpoints at the new cells' external gateways. The internal Private DNS zones are created in the new internal hub, and ExternalDNS fills them.
- **RTO** is measured, not assumed. It is the time of the rebuild drill below, and it is dominated by the database restores and the image builds. The target is set with the business.
 - **Address plan:** the paired region reuses the environment's own prefixes ([address plan](#)), so Firewall rules, NSGs and on-premises routes keep their meaning. That works only because the primary region's spokes are gone; a partial outage is handled by waiting for the region, not by running both at once.
 - **Failback** to the primary region is a planned migration later, done the same way in the other direction. It is not part of the recovery.

Tests

Added to the acceptance tests ([section 0.3](#)) and repeated **every six months in acc**:

- **Immutability:** as `owner` of the backup subscription, try to delete a recovery point, delete a blob version, shorten a retention and disable soft delete. Every attempt is denied or blocked by multi-user authorization.
- **Region recovery drill:** treat the primary region as lost (no access to its resources at all), rebuild acc in the paired region from IaC, Git and the immutable copies only, and run the reference application's tests. Record the RPO of every data store and the RTO end to end.

- **Stateful volumes:** Cross Region Restore of one zone's persistent volumes into a rebuilt stateful cluster; data consistent, and the StatefulSets bind the restored claims.

8.1 Release notes

Requirements: none - the version history of the design.

The design is versioned, one minor version per pull request that changes it and a major version only when a human says so, newest first; every version is tagged `v<version>` in every repository. Only the current major version is listed here: the older versions are in the Git tags, the last before v4.0 being `v3.8`. The version is shown in the site's header and on the covers of the [PDFs](#) ([how to add an entry](#)).

v4.5 - 2026-10-21

- Page 1.5 Developer experience tells the three journeys of an application team as use cases - onboarding a new application, migrating an existing one and operating the ones in production ([#119](#)).
- Requirements R33 (the non-negotiable rules: the patch timetable binds every application, automated tests are the only gate) and R34 (previews per pull request and sandboxes per team inside dev), with their acceptance criteria, R23 and R24 allow jobs, scheduled jobs and queue consumers, and R22 says that an application is reached from outside its namespace only as HTTPS on 443 through the gateway and uses any protocol inside it; the design states platform-owned base images, the non-serving workloads of 3.4 rebuilt and promoted by the base image agent, and the preview and sandbox namespaces of dev ([#119](#)).

v4.4 - 2026-10-20

- The requirements table is ordered by the design's reading order and numbered R1 to R32 without sub-letters, so the requirements a page cites sit together; every reference follows the new numbers ([#118](#)).

v4.3 - 2026-10-20

- The site has a Lab examples section beside the requirements and the design, generated by the documentation agent and in neither PDF ([#114](#)).

v4.2 - 2026-10-20

- Page 3.3 shows the stateless rollout with Flux and the stateful pipeline in two separate, readable pictures ([#112](#)).

v4.1 - 2026-10-20

- How we work is the front page, shorter and without what the requirements already say; the requirements PDF holds only the requirements and the acceptance criteria (#110).

v4.0 - 2026-10-20

- Requirements-driven collaboration: the requirements and acceptance criteria are in `requirements/`, owned by humans (part 0, page 0.1 How we work), and the design is in `design/`, owned by AI; a required Approvals check asks a human to approve a change to `requirements/` and the AI to review a change to `design/` (#109).
- The requirement texts of 1.2 are page 0.2 and design 1.2 shows how the design meets each of them; the acceptance tests of 7.1 are page 0.3 (#109).
- What no requirement names moved to the unpublished `archive/`: the pages 7.1 Implementation project and acceptance testing, 7.4 Outlook and the options kept open and 7.5 Other clouds, sovereignty and exit (its data services joined 1.3), and sections of 1.0, 1.2, 2.3 (GPUs), 4.1, 5.1, 5.2, 6.2, 7.2 and 7.3 (#109).
- Every design page names the requirements it serves under its title, and `check.sh` checks it; every link to a requirement jumps to its row in 0.2 (#109).
- The pages with a picture no longer repeat it in prose; the text keeps what the picture cannot show (reasons, decisions, numbers, exceptions) (#109).
- Two PDFs: the design with every design page, the summary first, and the requirements (#109).
- The decision log (7.2) and the open questions (7.3) left the design for `archive/`: the design states only what is chosen, without decision, sign-off or verification numbers (#109).