



AKKA AUTOMATED OPERATIONS

Technical Overview

The complete overview: architecture, operations, and every deployment model and cloud.

DOC.AKKA.IO · AKKA

Akka Automated Operations (AAO) is an actively maintained Akka region that Akka installs, monitors, patches, and scales for extreme availability and low latency. A region runs on Kubernetes across AWS, Azure, or GCP, hosted either in Akka's cloud (dedicated) or inside your own environment (BYOC or BYOK8s).

With the self-managed models (BYOC and BYOK8s), the region runs inside your own environment and you keep custodial ownership of the infrastructure. Application data stays within your environment; a limited set of platform interactions still crosses to the Akka Federation Plane: deployments, key rotation, the management API, metadata synchronization, and platform observability. The [data flows](#) page lists every flow.

The two planes and the main flows between them:

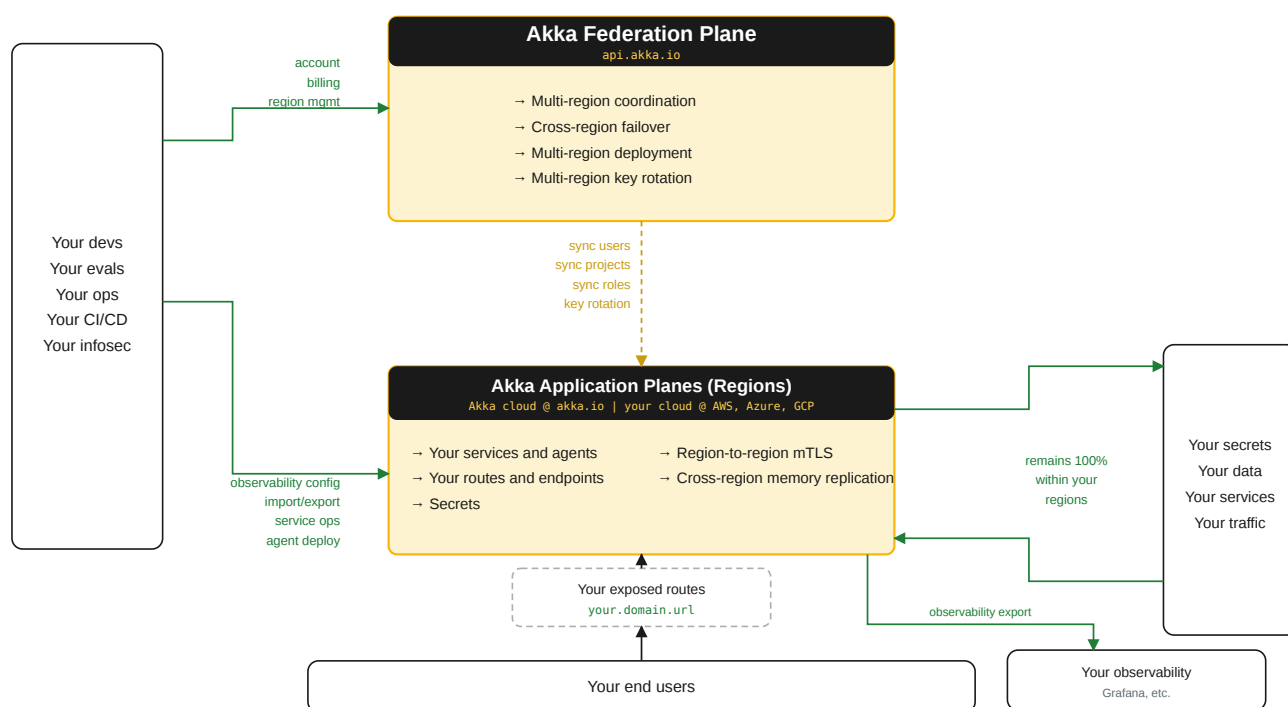


Figure 1. The Federation Plane and one or more per-region Application Planes

Federation Plane and Application Plane

AAO splits into two planes: a global coordination layer run by Akka, and one or more runtime regions that host your workloads.

Federation Plane

The central coordination point for organizations, accounts, billing, user access, key rotation, token issuance, and application deployment across regions. It runs at akka.io and is managed by Akka. It is

distributed and highly resilient. In the extreme exception that it becomes unavailable, running applications continue without interruption.

Application Plane

The runtime that hosts your workloads. Each federated region independently pulls and deploys application images, scales instances, and connects to peer regions. Application data stays within the region. The region exchanges only the platform interactions listed above with the Federation Plane.

What an Application Plane region embeds

- A managed Kubernetes cluster for orchestrating Akka instances as OCI containers.
- An encrypted application data store (append-only, event-sourced, non-queryable journal).
- A set of Akka operators for elasticity, observability, routing, and service management.
- Proxies for traffic steering between regions.
- Physical compute and storage for application execution.
- An image repository caching packed applications available in that region.

Key terminology

Term	Description
Akka Application	An app built with the Akka SDK: APIs, workflows, streaming consumers, timers, and views. Packed into OCI images and deployed as self-clustering service instances that act as their own in-memory, durable database.
Application Plane	Runtime environment hosting Akka apps within one or more regions. Provides compute, storage, I/O, autoscaling, observability, and infrastructure management to meet SLAs.
Federation Plane	Global coordination point federating multiple regions into a single deployment substrate. Runs and is managed at akka.io.
Akka CLI	Interface for developers, operators, and InfoSec teams: build, test, pack, deploy, observe, and manage secrets and accounts.
Data Persistence	Durable, encrypted-at-rest event store using event sourcing. Optimized for Akka's internal processing and not directly queryable.
VPC	A private, isolated network environment in a major cloud provider. A general term, not exclusively AWS's implementation.

Term	Description
Cloud Account	The billing and administrative entity: a subscription (Azure), an account (AWS), or a project (GCP).

What AAO automates

Beyond operations, AAO provides behavioral extensions that would otherwise require custom application code and libraries.

Capability	Description
Runtime Patching	Live-updates JVM and infrastructure runtimes without triggering app versioning or a customer redeployment. No repackaging required.
Rolling Updates	Zero-disruption rolling deployment, even across data-model changes. Old and new versions run side by side during the transition.
Elastic	Cold starts and automatic instance adjustment to traffic. Persistence expands and can shard to more than 1M writes per second at under 20ms write latency.
HA / DR	Cross-region zero-trust with rotating mTLS, multi-AZ database and Kubernetes clustering, continuous point-in-time backups, full region recovery, and CLI-based failover and failback.
App Data Replication	Replicate application data across regions (pinned single-region, write-local, or active-active) with developer-defined replication filters.
Multi-Region Deploy	Deploy a service once across one or more regions, with optional global routes for cross-region traffic.
Multi-Tenancy	Multiple projects and teams share compute and data infrastructure. Dedicated single-tenant regions are available for isolation requirements.
Observability	A Control Tower aggregates traces, spans, metrics, logs, and agentic evaluations across regions (testing use only).

What Akka provisions in a region

Networking

Set up using best practices for the specific cloud provider: VPC, subnets, load balancers, NAT.

DNS Zones

Automatically provisioned in your cloud account to resolve names for services deployed in the region.

Persistence

Encrypted-at-rest data store for app data, Akka events, and read-only views. Append-only, lock-free, and sharded. Benchmarked past 1M TPS.

Kubernetes

A managed cluster from your cloud provider orchestrates Akka instances deployed as OCI containers.

Registry

An OCI registry hosting immutable, signed, versioned service assets. You may optionally use your own registry.

Akka Operators

Route management, elasticity automation, metadata sync, rolling deploys, certificate management, key rotation, and multi-region failover and recovery.

Observability split. Akka runs a platform observability stack for internal backend monitoring. Your application logs, metrics, traces, and evaluations stay with your own observability vendor by default. A subset of platform telemetry is sent to the Federation Plane for consolidated cross-region reporting. See [Observability and monitoring](#).

Shared responsibility

System management is shared between you and Akka across provisioning and ongoing operations, and the split shifts with the deployment model. For the authoritative, row-by-row breakdown with the full Responsible / Accountable / Consulted / Informed (RACI) split, see the production-readiness shared-responsibility and RACI pages for your model:

- Dedicated: [Shared responsibility](#) and [RACI](#)
- BYOC: [Shared responsibility](#) and [RACI](#)
- BYOK8s: [Shared responsibility](#) and [RACI](#)

Security and encryption

Application-plane data is isolated from the Federation Plane and stays within the region. It is stored in an encrypted, durable event store that cannot be externally queried.

Data at rest uses the store's default encryption. Keys are provided by Akka or your KMS. Akka adds a further encryption layer for secrets synced to regional execution clusters, for authentication secrets such as TOTP secrets and OpenID refresh tokens, and for secrets you provide through environment variables or your KMS.

Data Encryption Keys (DEKs) encrypt data and are stored alongside it, themselves encrypted by a Key Encryption Key (KEK). KEKs live separately as Kubernetes secrets and rotate periodically. Rotating a KEK re-encrypts DEKs without re-encrypting the underlying data. Inter-region traffic uses mutual TLS with unique per-client and per-service certificates. Communication is established only if both sides mutually authenticate. See [Multi-region infrastructure](#) for the multi-region certificate and ingress details.

Akka adheres to a range of compliance standards. Full certification detail lives in the [Akka Trust Center](#).

Networking and ports

The default network configuration is a VPC with per-region subnets plus load balancers; the exact subnet layout differs by cloud and is described on each cloud page. By default, connectivity between the region and your environment traverses the internet. Private connectivity options are available per cloud.

Each flow below is inbound (ingress) or outbound (egress) relative to the region. All use TLS on port 443, except the infrastructure identity platform, which also uses ports 3023, 3024, and 3026.

Flow	Type	Port(s)
Akka Applications → Akka Control Tower	Egress	443
Akka Federation Plane → Akka Region API	Ingress	443
Data Import / Export	Ingress	443
Container Registry	Ingress	443
Backoffice Functions	Ingress	443
Kubernetes Cluster → Akka Platform Container Registry	Egress	443
Akka Region → Akka Federation Plane and Console	Egress	443
Platform Observability Agent → Akka	Egress	443

Flow	Type	Port(s)
Infrastructure Identity Platform	Egress	443, 3023, 3024, 3026

The Federation Plane exposes the Management API and the Application Plane exposes the Execution API, both secured with TLS and bearer tokens. The full hostname and IP allowlist, and the per-cloud data flows, are on the [data flows](#) page.

Default sizing and scaling

A new region and its services start with these defaults. Each scales automatically with demand and can be tuned with your Akka Success Team.

Persistence

Default is a small store: thousands of TPS, lowest initial cost. Scales and shards to well past 1M writes per second at under 20ms latency.

Compute

Always-available instance pools spread topologically across zones. Default mixes on-demand and spot instances to balance cost and performance.

Service instance

Each deployed service defaults to 3 instances across availability zones with 2,560 MB RAM each. Autoscales on CPU usage.

For forecasting load and choosing an availability level, see [Choosing your environment](#).

Backups and disaster recovery

Single-region Kubernetes resources and databases are backed up and recoverable. The persistence RPO is 5 minutes and the persistence RTO is 24 hours. Kubernetes RTO and RPO are 1 hour. Multi-region support replicates the application across regions for low-latency failover.

For the full recovery procedures and scenario catalogs, see the disaster-recovery pages for your model: [BYOC](#) and [BYOK8s](#).

Data flows

Every network flow between your environment and the Akka platform is documented for security review, including direction, connectivity, and authentication. See [Data flows and security pathways](#).

Support and maintenance

Akka continuously applies security patches and software upgrades to the underlying platform, including the infrastructure upgrades needed to keep it secure and performant. You provide a recurring maintenance window for operations that may cause downtime; in practice, most updates are non-events. Compute nodes are elastic and expand and contract with deployed services. Database storage autoscales, while additional database CPU and memory are applied by the Akka team when workloads exceed utilization thresholds. On request, Akka can pre-warm environments ahead of anticipated spikes.

If issues arise, Akka provides a customer support portal where cases are created and addressed per the Customer Support Policy.

Where to go next

Choose your deployment model and follow its setup. Each setup page ends by letting you choose your cloud.

Model	What Akka does	Start here
Dedicated	Akka hosts and operates the region in Akka's cloud, so there is nothing for you to install or provision. When you prepare to go live, review the Dedicated production readiness .	
BYOC	Akka provisions and operates the region in your own cloud account.	Get started with BYOC
BYOK8s	Akka installs and operates the platform on a cluster you provide.	Get started with BYOK8s

For help deciding between models and sizing for availability and data residency, see [Choosing your environment](#).

Under Bring Your Own Cloud (BYOC), Akka provisions and operates a region inside your own cloud account. You prepare the account with the `akka-bootstrap` utility and grant Akka a deployment identity; Akka then provisions the region, runs smoke tests, and hands it over.

This page lists the cloud-agnostic setup. For your cloud's resource inventory, private connectivity, and `values.hcl` specifics, follow the per-cloud page: [AWS](#), [Azure](#), or [GCP](#).

Installation flow

1. **Understand:** review the [technical overview](#) and receive the `akka-bootstrap` utility from your Akka Success Team, so you can inspect exactly which permissions Akka's deployment identity will require.
2. **Request a region:** submit a region request through the form in the Akka customer portal.
3. **Cloud account:** create a cloud account for Akka. Not strictly required, but a dedicated account simplifies cost management.
4. **Apply permissions:** run `akka-bootstrap` to configure the account, then attach the generated deployment-identity details to your region request.
5. **Provision:** Akka remotely provisions the region and attaches it to a platform organization created for you.
6. **DNS:** create record sets in your DNS zone that map to the region's load-balancer IP.
7. **Smoke tests:** Akka attaches the region to the Federation Plane and runs smoke tests.
8. **Region handover:** register a user at akka.io with a corporate email; Akka makes them your organization's primary owner. A production-labeled region is not fully handed over until you complete the [region-readiness steps](#).

The akka-bootstrap utility

`akka-bootstrap` prepares your cloud account for AAO by creating the Deployment Identity role the Federation Plane uses to create and manage infrastructure. It creates and manages the Deployment Identity operations role only. You create the Editor, Viewer, and SRE identities.

It is a Terragrunt project with a per-cloud module layout:

```
.
├── main.tftpl      # generates provider config + module calls
├── modules
│   ├── aws/       # IAM roles, trust policy
│   ├── azure/     # app registration, service principal, roles
│   └── google/    # service account, IAM bindings
├── README.md
└── terragrunt.hcl # orchestration: reads values.hcl
```

You create a `values.hcl` with your backend configuration and the region details for your cloud (the exact block differs per cloud; see the per-cloud page), then run the Terragrunt lifecycle:

COMMANDLINE |

```
terragrunt init    # initialize and validate the generated Terraform
terragrunt plan    # review the planned changes
terragrunt apply   # provision the identity, roles, and credentials
```

After a successful apply, share the generated deployment-identity details with Akka: the role ARN to assume on AWS, the service-principal credentials on Azure, or the service-account identity to impersonate on GCP.

The utility is the source of truth for the exact permissions Akka’s deployment identity requires. Review its generated output rather than a hand-maintained list, so the permissions you grant always match what Akka runs.

Identity model

AAO uses four roles across all clouds:

Role	Responsibilities	Who creates it
Deployment	Creation and management of infrastructure	akka-bootstrap
Editor	Management of infrastructure without delete permissions	You
Viewer	Read-only permissions for infrastructure	You
SRE	Identical to the Deployment role, for emergency response or upkeep	You

Within Kubernetes, Akka uses least-privilege access scoped to the Akka-managed namespaces and limits its access to customer-owned Kubernetes resources.

Choose your cloud

Continue with your cloud for its specifics:

- [BYOC on AWS](#): VPC and subnets, the IAM identity model, private connectivity, and the AWS `values.hcl`.
- [BYOC on Azure](#): virtual network, the Microsoft Entra ID identity model, resource providers, private connectivity, and the Azure `values.hcl`.

- [BYOC on GCP](#): VPC, the service-account identity model, private connectivity, and the GCP `values.hcl`.

This page covers running Akka Automated Operations (AAO) on AWS. For the architecture and concepts, start with the [technical overview](#).

The default multi-tenant architecture. Akka provisions a region into your VPC with public, private, and database subnets, and reaches it from the Federation Plane for control-plane operations.

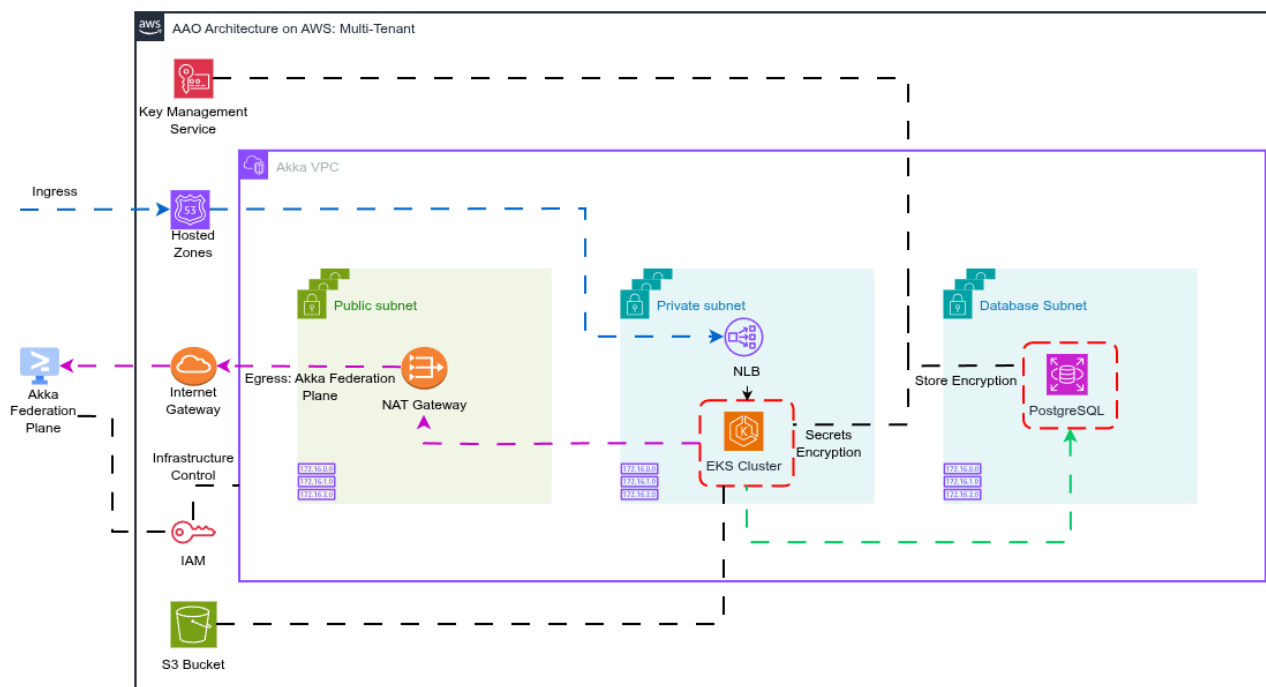


Figure 1. AAO on AWS: multi-tenant architecture (default)

AWS resources

AAO on AWS uses these AWS services:

- AWS Identity and Access Management (IAM)
- AWS RDS for PostgreSQL
- AWS Elastic Kubernetes Service (EKS)
- AWS Virtual Private Cloud (VPC)
- AWS Elastic Load Balancing
- AWS Route 53
- AWS Simple Storage Service (S3)
- AWS Key Management Service (KMS)

- AWS Transit Gateway

Identity model

To begin, create a non-human privileged IAM role with the minimum permissions to set up and manage the Akka environment, plus a trust policy that lets the Federation Plane's identity assume it. This role handles initial infrastructure bootstrapping, release deployments, and ongoing maintenance.

AAO uses four roles:

Role	Responsibilities	Who creates it
Deployment	Creation and management of infrastructure on AWS	akka-bootstrap
Editor	Management of infrastructure without delete permissions	You
Viewer	Read-only permissions for infrastructure	You
SRE	Identical to the Deployment role, for emergency response or upkeep	You

The exact IAM actions for each role are created and managed by `akka-bootstrap` and are the source of truth. Review the permission reference the utility generates rather than a copy here, so the permissions you grant always match what Akka runs. Within Kubernetes, Akka uses least-privilege access scoped to the Akka-managed namespaces and limits its access to customer-owned Kubernetes resources.

Installation

Prepare your AWS account with the `akka-bootstrap` utility, then Akka provisions the region. See [Bring Your Own Cloud setup](#) for the akka-bootstrap utility and the shared install flow.

Create a `values.hcl` file with your backend configuration and AWS region details:

```

inputs = {
  backend = {
    type = "s3"
    config = {
      bucket = "your-terraform-state-bucket"
      prefix = "akka-bootstrap"
    }
  }
}
akka_regions = {
  aws = [
    {
      environment      = "prod"          # dev | stage | prod
      akka_region_name = "aws-<tenant>--<tenant-env>--<region>"
      cloud_region_name = "<region>"
      profile           = "<profile>"
    }
  ]
  azure = []
  gcp   = []
}
}

```

Then run the Terragrunt lifecycle:

```

terragrunt init    # initialize and validate the generated Terraform
terragrunt plan    # review the planned changes
terragrunt apply   # provision the IAM role, policies, and credentials

```

After a successful apply, attach the generated role ARN to your region request so the Federation Plane can assume it and manage resources in your account.

Amazon EKS

Managed Kubernetes hosts the Application Plane, with workloads spread across availability zones. Akka deploys operators for route management, elasticity, metadata sync, rolling deployment, certificate management, key rotation, and multi-region failover.

Private connectivity

By default, traffic between the region and your internal networks traverses the internet. To keep it private, AAO on AWS uses an AWS Transit Gateway to route region-to-internal traffic across accounts and VPCs. The per-flow detail is on the [data flows](#) page.

The Transit Gateway pattern:

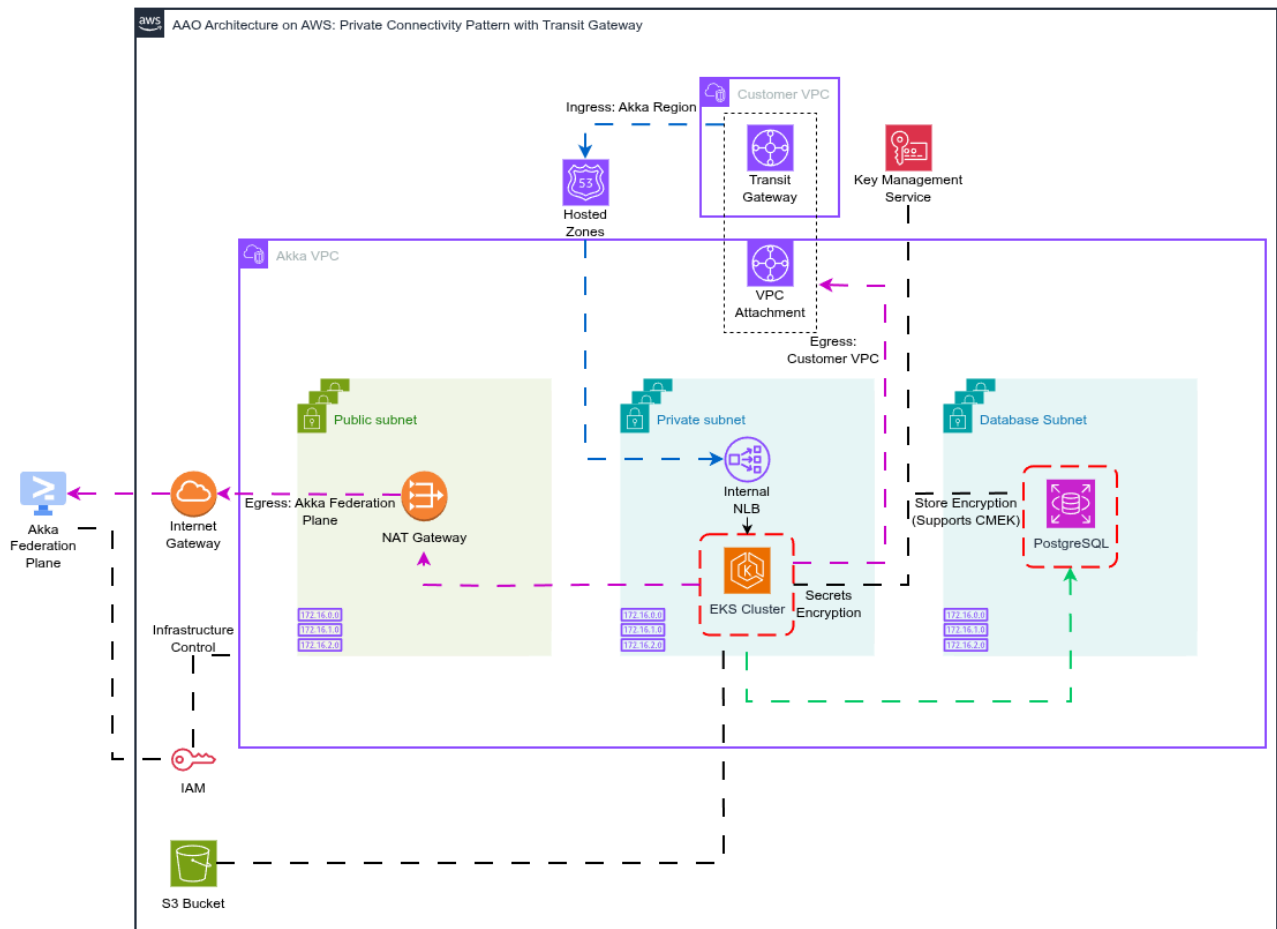


Figure 2. AAO on AWS: private connectivity with a Transit Gateway

A hub-and-spoke topology (private only, with platform images pulled from a container registry bucket):

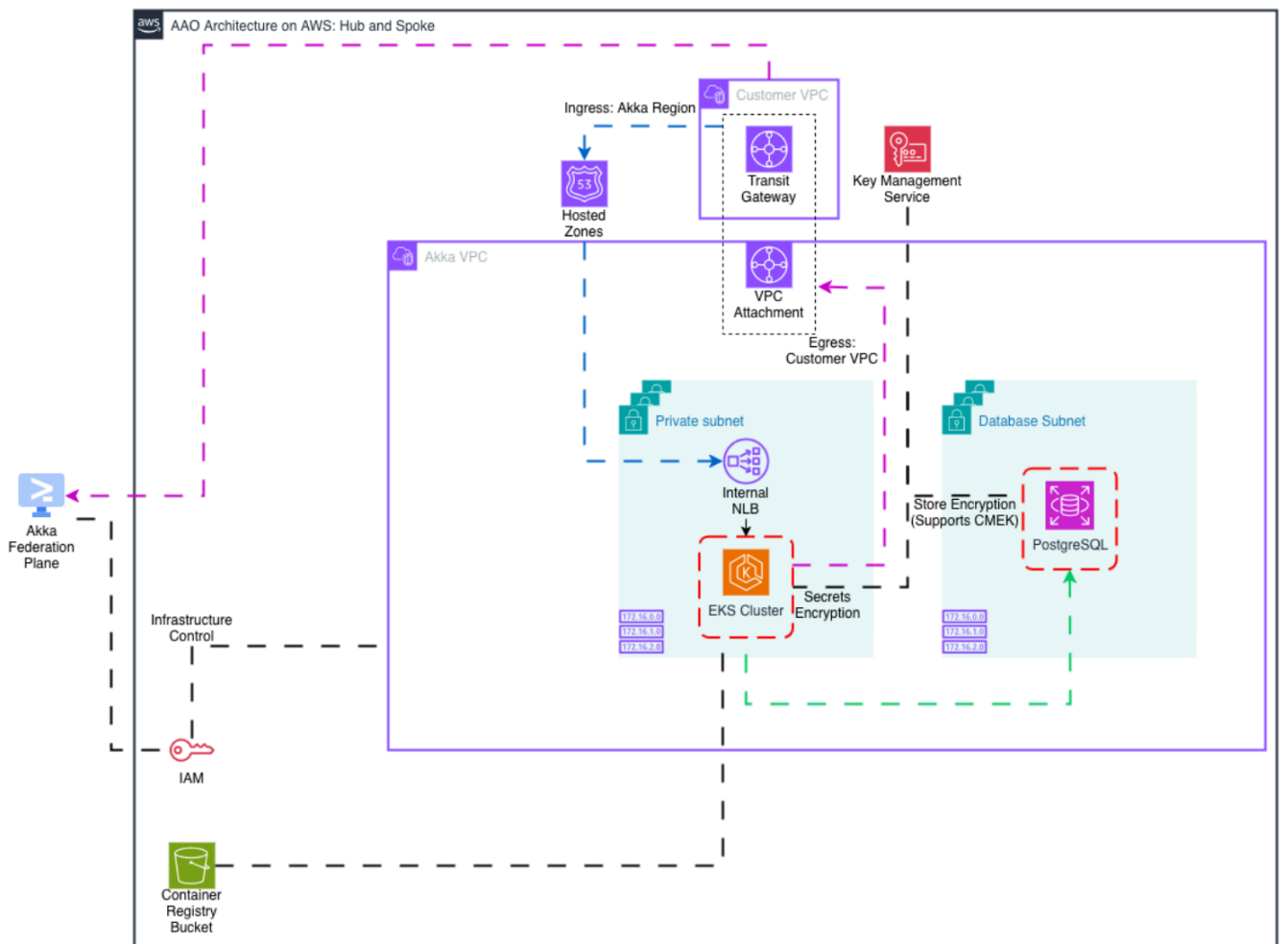


Figure 3. AAO on AWS: hub-and-spoke topology

Bringing your own EKS cluster (BYOK8s)

If you provision the cluster yourself instead of having Akka provision into your account, see [Akka Automated Operations on AWS with your own Kubernetes](#) for the complete, self-contained guide (the shared requirements plus the AWS specifics).

See also

- [Technical overview](#)
- [BYOK8s requirements](#)
- [Data flows](#)
- [BYOC disaster recovery](#)

This page covers running Akka Automated Operations (AAO) on Azure. For the architecture and concepts, start with the [technical overview](#).

The default architecture. Akka provisions a region into your virtual network with private and database subnets, using Microsoft Entra ID for the deployment identity.

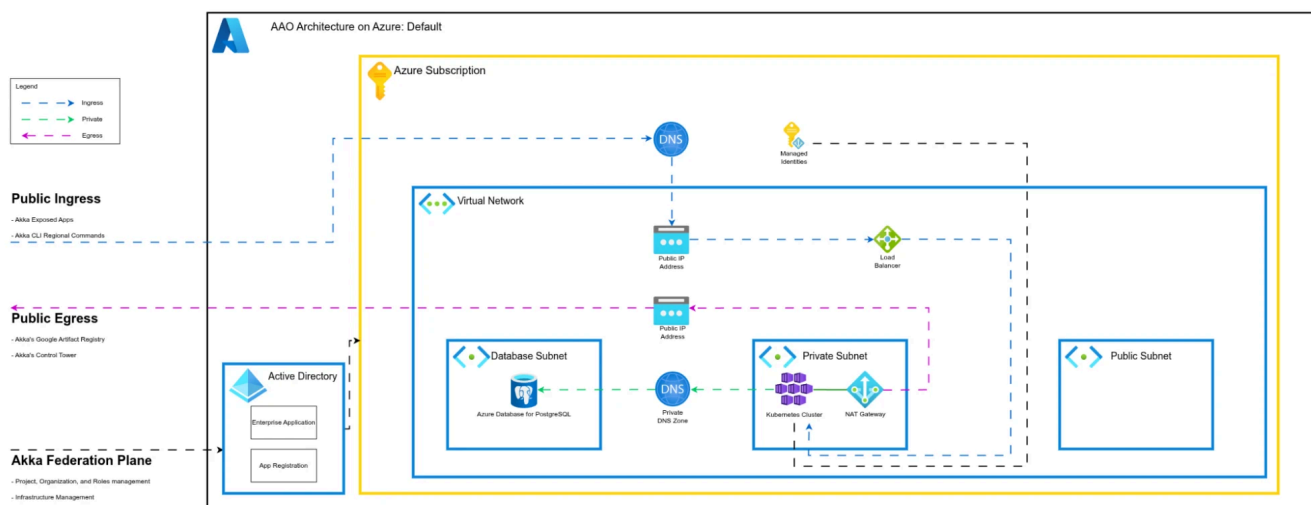


Figure 1. AAO on Azure: default architecture

Azure resources

AAO on Azure uses these Azure services:

- Microsoft Entra ID (applications, app registrations, service principals)
- Microsoft Azure Resource Manager (subscription, resource group, roles and role assignments)
- Networking: virtual network, subnet, public IP address, NAT gateway, load balancer, private and public DNS zones
- Compute: AKS cluster, node pools (virtual machine scale sets), network security groups
- Data: Azure Database for PostgreSQL flexible server
- Storage: storage accounts and blob containers
- Identity: managed identities and federated credentials

Identity model

An enterprise application is registered in Microsoft Entra ID to create its identity, and a service principal is created for it. A non-human privileged role with the minimum permissions is configured and used by the Federation Plane to securely bootstrap and manage the environment.

AAO uses four roles: Deployment (created by `akka-bootstrap`), and Editor, Viewer, and SRE (created by you). The exact permissions are created and managed by `akka-bootstrap` and are the source of truth; review the generated permission reference rather than a copy here. The Editor role cannot delete resources and cannot read or write federated identity credentials.

Resource providers to register

These register automatically through `akka-bootstrap`. Verify they are enabled on the subscription:

- `Microsoft.Authorization`
- `Microsoft.ContainerService`
- `Microsoft.DBforPostgreSQL`
- `Microsoft.ManagedIdentity`
- `Microsoft.Resources`
- `Microsoft.Network`
- `Microsoft.Storage`

The subscription must also have EncryptionAtHost enabled:

```
az feature register --name EncryptionAtHost --namespace Microsoft.Compute  
az provider register -n Microsoft.Compute
```

COMMAND LINE |

Installation

Prepare your subscription with the `akka-bootstrap` utility, then Akka provisions the region. Create a `values.hcl` with your backend configuration and Azure region details:

```

inputs = {
  backend = {
    type = "azurerm"
    config = {
      bucket = "your-terraform-state-bucket"
      prefix = "akka-bootstrap"
    }
  }
}

akka_regions = {
  azure = [
    {
      environment      = "prod"      # dev | stage | prod
      akka_region_name = "az-<tenant>--<region>"
      tenant_id        = "azure-tenant-id"
      subscription_id   = "azure-subscription-id"
    }
  ]
  aws = []
  gcp = []
}

```

Then run `terragrunt init`, `terragrunt plan`, and `terragrunt apply`. After a successful apply, share the generated service-principal credentials with Akka so the Federation Plane can authenticate and manage resources in your subscription.

Private connectivity

By default, traffic between the region and your internal networks traverses the internet. To keep it private, AAO on Azure uses Virtual Network Peering with a user-defined NAT gateway. The per-flow detail is on the [data flows](#) page.

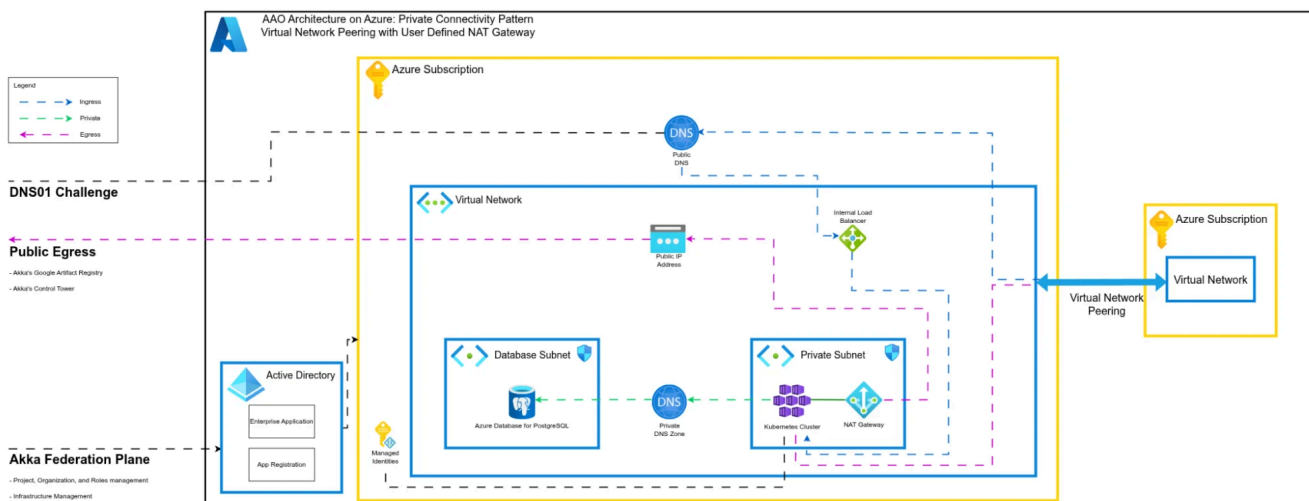


Figure 2. AAO on Azure: private connectivity with Virtual Network Peering and a user-defined NAT gateway

Bringing your own AKS cluster (BYOK8s)

If you provision the cluster yourself, see [Akka Automated Operations on Azure with your own Kubernetes](#) for the complete, self-contained guide.

See also

- [Technical overview](#)
- [BYOK8s requirements](#)
- [Data flows](#)

This page covers running Akka Automated Operations (AAO) on GCP. For the architecture and concepts, start with the [technical overview](#).

The architecture Akka provisions into your GCP project, with a VPC network, GKE, Cloud SQL, and the supporting services below.

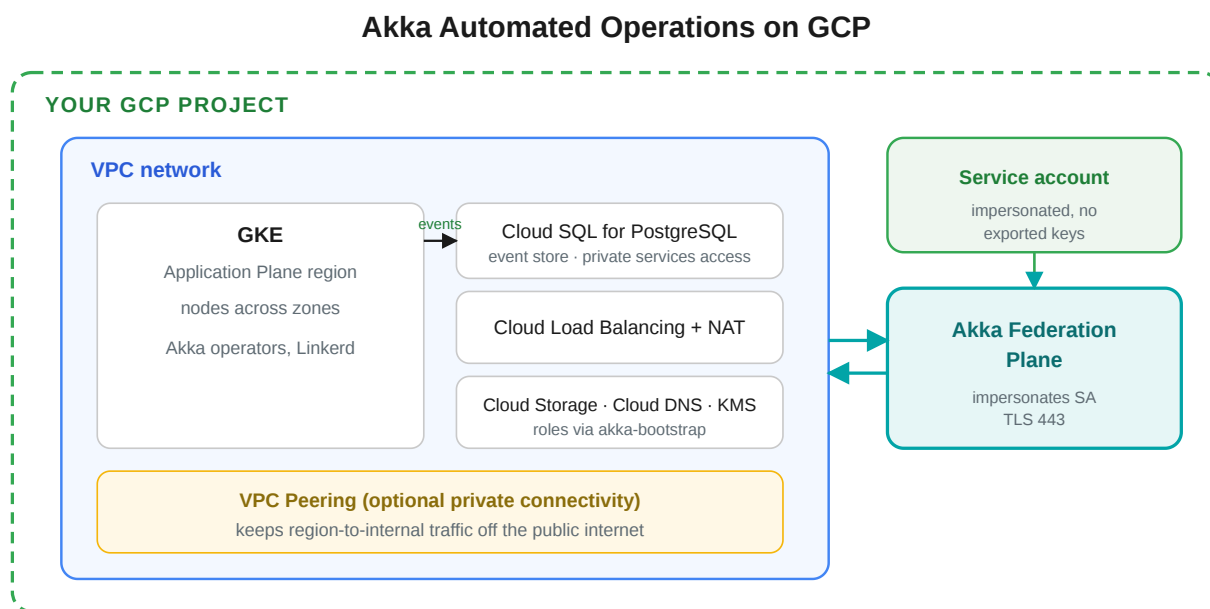


Figure 1. AAO on GCP: architecture

GCP resources

AAO on GCP uses these Google Cloud services (from the `akka-bootstrap` `modules/google` module):

- Cloud IAM and Cloud Resource Manager (service account, custom roles, and bindings)
- Compute Engine networking (VPC, subnets, Cloud NAT, external IP, Cloud Load Balancing)
- Google Kubernetes Engine (GKE)
- Cloud SQL for PostgreSQL
- Service Networking (private services access for the database)
- Cloud DNS (public and private zones)
- Cloud KMS
- Cloud Storage

Identity model

The cloud account is a GCP project. A dedicated Google service account is created for the region and assigned a non-human privileged role with the minimum permissions to set up and manage it. The service account is then configured so the Federation Plane's identity can impersonate it to bootstrap and manage the region.

Rather than long-lived exported keys, the Federation Plane impersonates the dedicated service account, which keeps bootstrap and management credential-free on your side.

AAO uses four roles: Deployment (created by `akka-bootstrap`), and Editor, Viewer, and SRE (created by you). The exact roles and bindings are created and managed by `akka-bootstrap` and are the source of truth; review the generated permission reference rather than a copy here.

Google GKE

Managed Kubernetes hosts the Application Plane, with instances spread topologically across zones. The event store is Cloud SQL for PostgreSQL.

Installation

Prepare your project with the `akka-bootstrap` utility. In `values.hcl`, use a `gcs` backend and populate the `gcp` block of `akka_regions`, then run `terragrunt init`, `terragrunt plan`, and `terragrunt apply`. After a successful apply, share the generated service-account details with Akka so the Federation Plane can impersonate it. Bootstrap is driven by the `modules/google` Terraform module.

Private connectivity

By default, traffic between the region and your internal networks traverses the internet. To keep it private, AAO on GCP uses VPC Peering between the region VPC and your internal networks. The per-flow detail is on the [data flows](#) page.

Bringing your own GKE cluster (BYOK8s)

If you provision the cluster yourself, see [Akka Automated Operations on GCP with your own Kubernetes](#) for the complete, self-contained guide.

See also

- [Technical overview](#)

- [BYOK8s requirements](#)
- [Data flows](#)

Under Bring Your Own Kubernetes (BYOK8s), you provision the cluster to Akka's specification and install Teleport to grant access. Akka then configures the cluster as an application-plane region, runs smoke tests, and hands it over. This model runs in your cloud or your own data center.

This page lists the cloud-agnostic requirements. For a complete, self-contained setup guide for your cloud, follow the per-cloud page: [BYOK8s on AWS](#), [BYOK8s on Azure](#), or [BYOK8s on GCP](#). Each of those includes everything below plus its cloud specifics.

Once the region is running, do not change any provisioned infrastructure without first informing Akka. It could break the Akka region.

Installation flow

1. **Understand:** review the [technical overview](#) and contact your Akka Success Team for a BYOK8s package.
2. **Capacity planning:** size the environment with your Akka Success Team.
3. **Infrastructure requirements:** agree on DNS subdomain zones, certificate provider, and related decisions.
4. **Infrastructure provisioning:** provision the network, cluster, and database, and set up the egress domain allowlist.
5. **Customer smoke tests:** run the checklist below before handing the cluster to Akka.
6. **Teleport:** install the Akka-provided Teleport Helm chart. Deploy it promptly: the join token has a 3-hour TTL.
7. **Installation:** Akka configures the cluster as an application-plane region and sets up platform observability. You seed database credentials into the namespaces Akka provides.
8. **Akka smoke tests:** Akka attaches the region to the Federation Plane and runs smoke tests.
9. **Region handover:** Akka assigns the region to your organization. A production-labeled region is not fully handed over until you complete the [region-readiness steps](#).

Infrastructure requirements

Requirement	Specification
Kubernetes	Version 1.34 at minimum. Cilium or Calico is mandatory for network-policy enforcement. Configure it as an overlay network only when the cloud-native CNI is unfeasible due to IPAM constraints.

Requirement	Specification
Cluster CIDRs	<code>/17</code> for the pod network and <code>/17</code> for the service network.
Network	Minimum <code>/21</code> CIDR block from your IPAM allocation, subdivided into three database subnets (<code>/27</code> each) and three private subnets (<code>/24</code> each).
Nodes	16 vCPU / 64 GB RAM instances (for example AWS <code>m5.4xlarge</code> , GCP <code>n2d-standard-16</code> , Azure <code>Standard_D16s_v6</code>). Minimum one node. Kubernetes autoscales the node count with demand.
Service Mesh	Akka deploys Linkerd, currently the only supported service mesh.
Load Balancer	Public Layer 4 by default. An internal load balancer is supported, and a subnet or IP can be specified on supported clouds.
DNS and Certificates	Two subdomain DNS zones, one for platform APIs and one for deployed services (for example <code>akka.acme.com</code> , <code>apps.akka.acme.com</code>), plus cert-manager <code>ClusterIssuer</code> objects for automatic certificate generation on both.
PostgreSQL	Version 17 or later, provisioned and configured by you, highly available across zones (RDS on AWS, CloudSQL on GCP, Azure Database for PostgreSQL flexible server).
CSI Driver	The cloud-specific secrets-store CSI (Container Storage Interface) driver provider.
Registry	Platform images can be pulled through your Artifactory, using Akka's container registry as a mirror.

Database credentials

Seed the connection credentials (username, password, host, database name) as Kubernetes Secrets into both the `kalix-system` and `kalix-management-system` namespaces. You create these namespaces if needed, but do not manage them: AAO imports them into its management scope and reconciles them. Akka recommends the [external-secrets operator](#) to mirror credentials from your cloud secret store.

Access

Within a customer-provisioned cluster, Akka uses least-privilege permissions scoped to the Akka-managed namespaces and limits its access to customer-owned Kubernetes resources.

Capacity planning

Database size is driven by total application data operations per second at peak load. Work with your Akka Success Team to determine sizing.

DB Size	Data ops/sec	CPU	Memory (GB)	Storage (GB)	Max Connections
XSmall	1,000	1	4	100	200
Small	3,000	2	16	200	500
Medium	6,000	4	32	400	1,000
Large	12,000	8	64	800	1,500
XLarge	24,000	16	128	1,600	2,000

Egress allowlist and Teleport

Akka reaches the cluster through Teleport, which uses ports 3023, 3024, and 3026 in addition to 443. Teleport is a certificate authority and identity-aware proxy; the non-standard ports segment traffic types, and the security boundary is mutual TLS rather than the port. The full egress hostname and IP allowlist (Teleport, console, Federation Plane, Control Tower, observability, and container registry endpoints) is on the [data flows](#) page.

Customer smoke-testing checklist

- Check connectivity between the database instance and the Kubernetes cluster.
- Ensure the `ClusterIssuer` is ready and in a good state, and verify all tokens it uses.
- In multi-region configurations, ensure traffic flows in both directions between regions.

Choose your cloud

Each page is a complete, self-contained setup: the requirements above plus that cloud's specifics.

- [BYOK8s on AWS](#): Route 53, IRSA, EKS, RDS for PostgreSQL, and external-secrets.
- [BYOK8s on Azure](#): resource providers, EncryptionAtHost, and Azure Database for PostgreSQL.
- [BYOK8s on GCP](#): Cloud SQL, private services access, and Cloud DNS.

AWS specifics

Apply these AWS additions to the requirements above.

Route 53 records: create these in your hosted zone (public or private):

Record	Type	Target
akka.[zone]	A	Network Load Balancer (platform APIs)
*.akka.[zone]	CNAME	akka.[zone]
services.akka.[zone]	A	Network Load Balancer (user services and app support APIs)
*.services.akka.[zone]	CNAME	services.akka.[zone]

- **Certificates:** cert-manager `ClusterIssuer` with the Route 53 `dns01` solver, authorized through IRSA (IAM Roles for Service Accounts) to write validation TXT records.
- **EKS cluster:** node groups (one On-Demand, optionally one Spot) at 16 vCPU / 64 GB (for example `m5.4xlarge`); AMI `AL2023_x86_64_STANDARD`; managed addons `coredns`, `aws-ebs-csi-driver`, `vpc-cni`, `kube-proxy`, and the AWS secrets-store CSI driver provider; Calico in policy-only mode.
- **RDS for PostgreSQL:** PostgreSQL 17, Multi-AZ, automatic minor upgrades and deletion protection; parameter group `rds.force_ssl = 1`. Because SSL is required, install the RDS CA bundle (`https://truststore.pki.rds.amazonaws.com/<region>/<region>-bundle.pem`) into the `kalix-system` and `kalix-management-system` namespaces.
- **Database credentials:** use the external-secrets operator to mirror them from AWS Secrets Manager.

Azure specifics

Apply these Azure additions to the requirements above.

- **Resource providers:** register the resource providers and enable EncryptionAtHost as described on the [BYOC on Azure page](#).
- **Database:** use Azure Database for PostgreSQL flexible server.
- **Secrets:** use the Azure secrets-store CSI driver provider.

GCP specifics

Apply these GCP additions to the requirements above.

- **Database:** use Cloud SQL for PostgreSQL, reachable over private services access.
- **DNS:** use Cloud DNS for the two subdomain zones.
- **Secrets:** use the GCP secrets-store CSI driver provider.

This page documents the individual network flows between your environment and the Akka platform. Each flow lists the data carried, its source and destination, the connectivity used, and how it is authenticated. "Your ..." denotes a resource in your own environment.

Connectivity terms used below:

- **Peering:** a private network path between your environment and the Akka region.
- **Firewall, Federation Plane allowlisted:** a customer-initiated outbound flow to the Federation Plane over the internet, from an allowlisted egress path.
- **Akka NAT gateway:** a region-initiated egress flow. Akka can work with you to route these through your firewall instead.
- **Pre-authorized IP ranges:** Federation-Plane-to-region flows over the internet. A private-connectivity option is available.

Your end users

Flow	From	To	Connectivity	Auth
End user accessing an Akka service from outside your environment	End user	Akka Region	Public (through your firewall)	Your keys, secrets, and tokens
End user accessing an Akka service from inside your environment	End user	Akka Region	Private (peering)	Your keys, secrets, and tokens

Your deployed Akka services

Flow	From	To	Connectivity	Auth
Access to your private internal systems	Your service	Your APIs	Private (peering)	Your keys, secrets, and tokens
Pushing observability metrics for your services to your observability	Akka Region	Your observability	Private (peering)	Your keys, secrets, and tokens

Your operators, admins, and developers

Flow	From	To	Connectivity	Auth
Akka CLI or console authentication initiation	CLI or console	Federation Plane	Public (firewall, Federation Plane allowlisted)	Akka authentication
OpenID authentication	Console	Your IAM	Public	OpenID
Add a region to a project	CLI	Federation Plane	Public (firewall, Federation Plane allowlisted)	Akka authenticated, valid access token
Initiate multi-region failover	CLI	Akka Region	Private (peering)	Akka authenticated, valid access token
List services and projects	CLI	Akka Region	Private (peering)	Akka authenticated, valid access token
Create a project	CLI	Federation Plane	Public (firewall, Federation Plane allowlisted)	Akka authenticated, valid access token
Add a user, role, or permission	CLI	Federation Plane	Public (firewall, Federation Plane allowlisted)	Akka authenticated, valid access token
Inspect service components in a region	CLI	Akka Region	Private (peering)	Akka authenticated, valid access token
Deploy a service: initiate deployment	CI/CD or developer	Akka Region	Private (peering)	Akka authenticated, valid access token
Deploy a service: region pulls the image	Akka Region	Your image registry	Private (peering)	Akka authenticated, valid access token

Akka SRE

Flow	From	To	Connectivity	Auth
Troubleshoot a region (cloud console or kubectl)	Your VDI, when needed	Akka Region	Private (your remote desktop)	Your cloud credentials, granted through Teleport

Akka Region

Flow	From	To	Connectivity	Auth
Region observability metadata to the Federation Plane	Akka Region	Federation Plane	Public (Akka NAT gateway)	Kubernetes RBAC and Federation Plane observability token
Region requests the public key set	Akka Region	Federation Plane	Public (Akka NAT gateway)	Internal Akka JWT authentication
Certificate expiration monitoring (can be disabled)	Akka Region	Akka SRE	Public (Akka NAT gateway)	Kubernetes RBAC
Replication to another Akka Region	Akka Region	Akka Region	Private (peering)	Mutual TLS and ACLs

Akka Federation Plane

All flows below use the bootstrap credential you create and share with Akka.

Flow	From	To	Connectivity
Bootstrap a new Akka Region	Federation Plane	Akka Region	Public (pre-authorized IP ranges)
Install a new version of Akka into a region	Federation Plane	Akka Region	Public (pre-authorized IP ranges)

Flow	From	To	Connectivity
Maintain infrastructure (drift detection)	Federation Plane	Akka Region	Public (pre-authorized IP ranges)
Upgrade infrastructure (Kubernetes, database)	Federation Plane	Akka Region	Public (pre-authorized IP ranges)
Multi-region root CA management for cross-region mTLS	Federation Plane	Akka Region	Public (pre-authorized IP ranges)
Synchronize project, user, role, organization, and region metadata	Federation Plane	Akka Region	Public (pre-authorized IP ranges)

Private connectivity by cloud

Under BYOC, Akka can set up a private-connectivity option per cloud to keep region-to-internal traffic off the public internet: Transit Gateway on AWS, VPC Peering on GCP, and VNet Peering with Azure Firewall on Azure. See the [BYOC on AWS](#), [BYOC on Azure](#), and [BYOC on GCP](#) pages.