TECHNICAL GUIDE:
April 2024

VMware® Avi™ Load Balancer Ingress Design Guide

vmware® by Broadcom

Table of Contents

Document Overview

The VMware Avi Load Balancer Design Guide for Container Ingress application services use cases/solutions for Kubernetes/Openshift container clusters provides a brief description of all the Avi solution components and their design considerations.

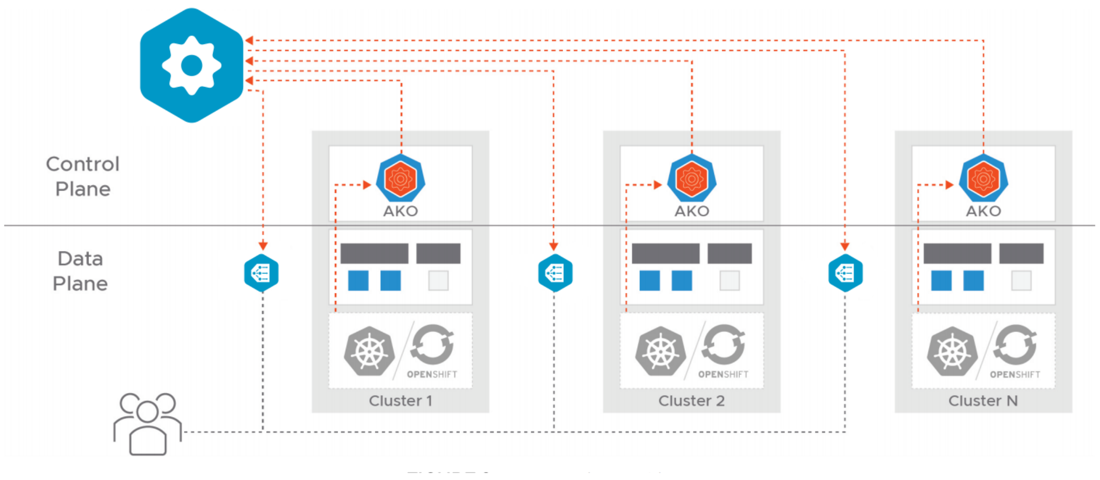
The Design Guide is intended for Infrastructure and Network Architects who want to leverage VMware Avi Load Balancer solution for providing advanced application services like Reverse Proxy, Service Load Balancer (SLB), Global Server Load Balancer (GSLB), Web Application Firewall (WAF) etc. for deploying Container Ingress (L7) or Load Balancer (L4) services in Kubernetes/Openshift container clusters.

1.0 Ingress Load Balancing

Avi Load Balancer integrates with Kubernetes / OpenShift container environments using an Ingress Controller called Avi Kubernetes Operator (AKO) to provide L4/L7 services.

The Avi Deployment in Kubernetes for AKO comprises of the following main components:

- The Avi Controller
- The Service Engines (SE)
- The Avi Kubernetes Operator (AKO)



1.1 Avi Controller

The Avi Controller which is the central component of the Avi architecture is responsible for the following Control plane functionality like the:

- Infrastructure orchestration
- Centralized management
- Analytics dashboard
- Integration with the underlying ecosystem for managing the lifecycle of the data plane (Service Engines).

In Container environments, the Avi Controller is deployed outside the Kubernetes cluster, typically in the native type of the underlying infrastructure. However, it can be deployed anywhere as long as connectivity and latency requirements are satisfied. The Avi Controller uses the Avi Cloud configuration to manage the SEs. This Avi Cloud is usually of the underlying infrastructure type, for ex. VMware vCenter Cloud, Azure Cloud, Linux Server Cloud etc. A single Avi Cloud can be used for integration with multiple Kubernetes clusters, with each cluster running its own instance of AKO. Clusters are separated on SEs in the DataPlane by using VRF Contexts. Each Kubernetes cluster must be deployed in a separate VRF to prevent overlap of IP addresses etc. The IPAM and DNS functionality is handled by the Avi Controller via the Avi cloud configuration.

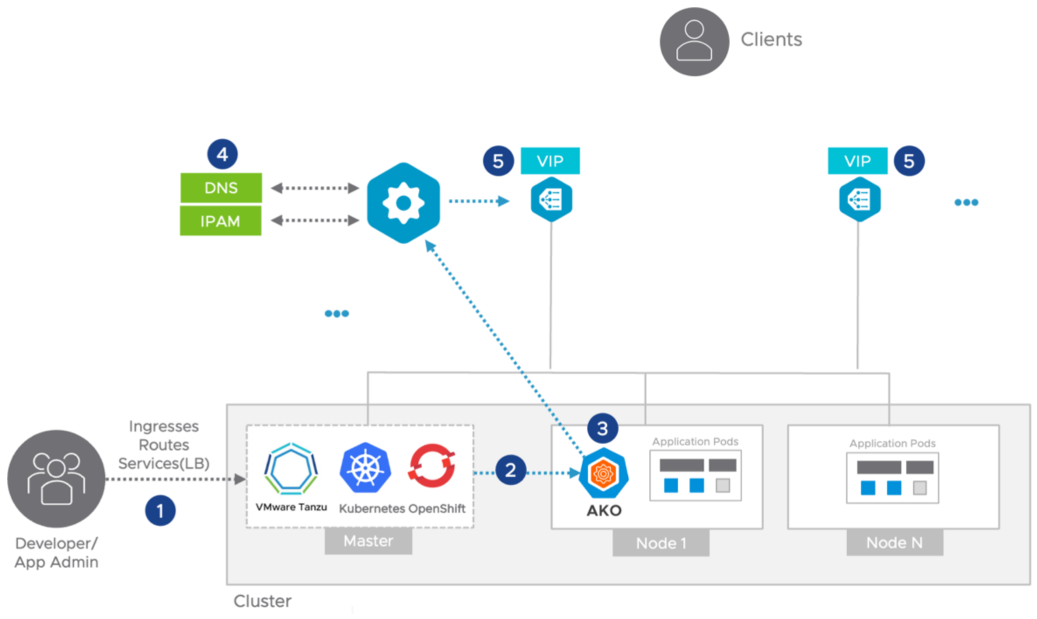
1.2 Avi Service Engine

The Avi Service Engines are the data-plane engines that implement the virtual services for Container ingresses and provides application services like load balancing, traffic management, Web Application Firewall, DNS/IPAM, GSLB etc. These SEs handle all the data plane responsibilities in the platform. The SEs are deployed external to the container cluster and typically in the native type of the underlying infrastructure.

1.3 Avi Kubernetes Operator (AKO)

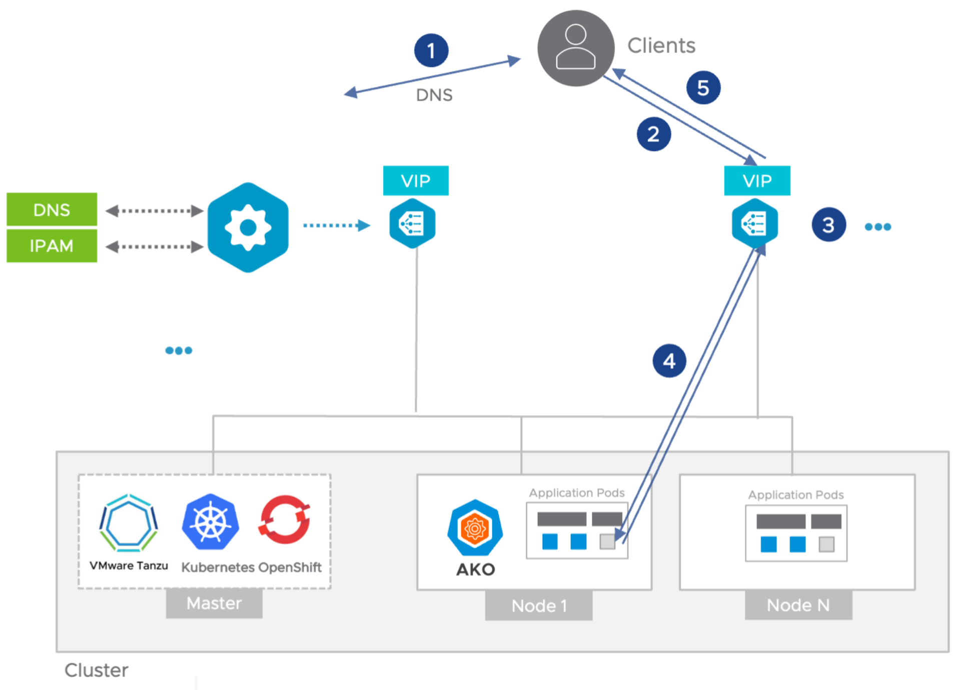
AKO works as an ingress controller and performs Avi specific functions/integration in a Kubernetes/Openshift environment. It runs as a pod in the cluster and translates the required Kubernetes/Openshift objects to Avi objects and automates the implementation of ingresses/routes/services on the Service Engines (SE) via the Avi Controller. The AKO specific configuration parameters are defined in the values.yaml manifest file. AKO is deployed as a pod via Helm using the values.yaml file. The configuration parameters for AKO is detailed in values.yaml. For multi-cluster deployments, AKO is installed in each cluster using Helm. Once deployed, AKO establishes connectivity with the Controller using the credentials mentioned in the values.yaml manifest file. AKO needs HTTPS access to the Avi controller for configuring virtual service objects using API calls.

The application deployment workflow is as shown below:



- User configures Ingresses/Routes and CRDs
- AKO gets these objects from the K8s/OC Master
- AKO translates Ingresses/Routes and calls Avi APIs
- Avi Controller allocates an IP address from IPAM and publishes FQDN to DNS
- Avi Service Engines host Virtual Services for Ingresses/Routes

AKO does not come in the data-path. The data-path traffic flow is as shown below:



- Client resolves app FQDN to Avi VIP via DNS (Avi or external)
- Client sends request to Avi VIP
- Avi Service Engine chooses a backend pod for load-balancing.
- Request is sent to the backend Pod IP
- App responds back to the Client via Avi Service Engine

1.4 Network Considerations

With AKO, the service engines are deployed outside the cluster. To be able to load balance requests directly to the pods, the pod CIDR must be routable from the SE. The reachability of the pod CIDRs from external networks depends on the CNI used.

AKO supports the following CNI:

- Calico
- Antrea
- OpenShift SDN
- Flannel
- NCP
- OVN-Kubernetes CNI in OpenShift
- Cilium CNI in Kubernetes

Depending on the routability of the Pod CNI used in the cluster, AKO can route using the following options:

- Pods are Not Externally Routable – Static routing

  For CNIs like Canal, Calico, Antrea, Flannel etc., the pod subnet is not externally routable. In these cases the CNI assigns a pod CIDR to each node in the Kuberntes cluster. The pods on a node get IP assigned from the CIDR allocated for that node and is routable from within the node. In this scenario, the pod reachability depends on where the SE is placed.

  If SE is placed on the same network as the Kubernetes nodes, you can turn on static route programming in AKO. With this, AKO syncs the pod CIDR for each Kubernetes node and programs static route on the Avi Controller for each Pod CIDR with the Kubernetes node IP as the next hop. Static routing per cluster uses a label-based routing scheme.

- Pods are Not Externally Routable – NodePort

  In cases where direct load-balancing to the pods is not possible, NodePort based services can be used as the pool members in the Avi virtual service as end points. For this functionality, configure the services referenced by Ingresses/Routes as type *NodePort* and set the `configs.serviceType` parameter to enable NodePort based Routes/Ingresses. The `nodeSelectorLabels.key` and `nodeSelectorLabels.value` parameters are specified during the AKO installation to select the required Nodes from the cluster for load balancing. The required nodes in the cluster need to be labelled with the configured key and value pair.

- Pod Subnet is Routable

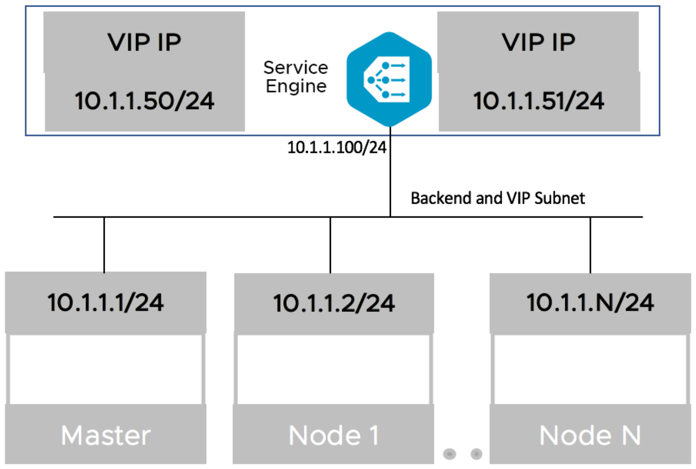
  For CNIs like NSX-T CNI, AWS CNI (in EKS), Azure CNI (in AKS) etc., the pod subnet is externally routable. In this case no additional configuration is required to allow SEs to reach the Pod Ips. Static Route Programming is set to Off in the AKO configuration. Ses can be placed on any network and will be able to route the pods.

1.5 Avi SE Deployment modes

Avi SE supports the following deployment modes:

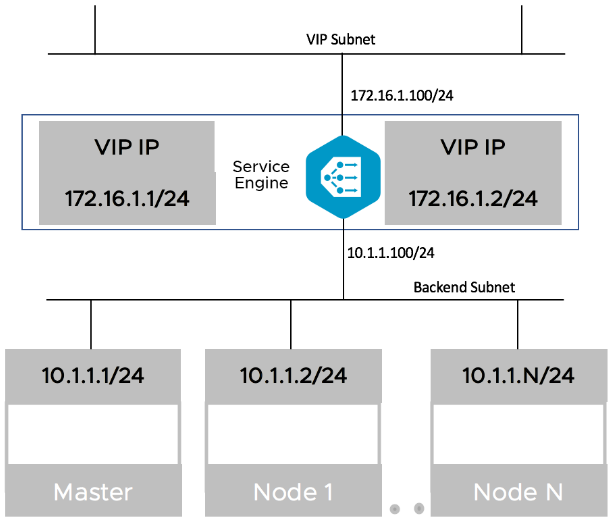
- Single Arm Deployment

  The deployment in which the virtual IP (VIP) address and the Kubernetes/ OpenShift cluster are in the same network subnet is called a *Single Arm Deployment*. NSX-T Write access cloud requires single arm SE deployment using NodePort or NodePortLocal with Antrea CNI.



- Two-Arm Deployment

  When the virtual IP (VIP) address and the Kubernetes/ OpenShift cluster are in different network subnets, then the deployment is a Two-Arm deployment.



AKO supports both Single-Arm and Two-Arm deployments with vCenter Cloud in write-access mode.

1.6 Tenancy

This feature allows AKO to map each kubernetes / OpenShift cluster uniquely to a tenant in Avi. To enable this feature, set the field ControllerSettings.tenantsPerCluster to true. Create the required roles with appropriate privileges to the ako user in the admin or the specific tenant mapped to the container cluster. Avi non admin tenants primarily operate in 2 modes, provider context and tenant context.

- Provider Context

  Service Engine Groups are shared with admin tenant. All the other objects like virtual services and pools are created within the tenant. This requires `config_settings.se_in_provider_context` flag to be set to *True* when creating tenant.

- Tenant Context

  Service Engines are isolated from admin tenant. A new Default group is created within the tenant. All the objects including Service Engines are created in tenant context. Requires config_settings.se_in_provider_context flag to be set to False when creating tenant.

In AKO manifest file, configure the following:

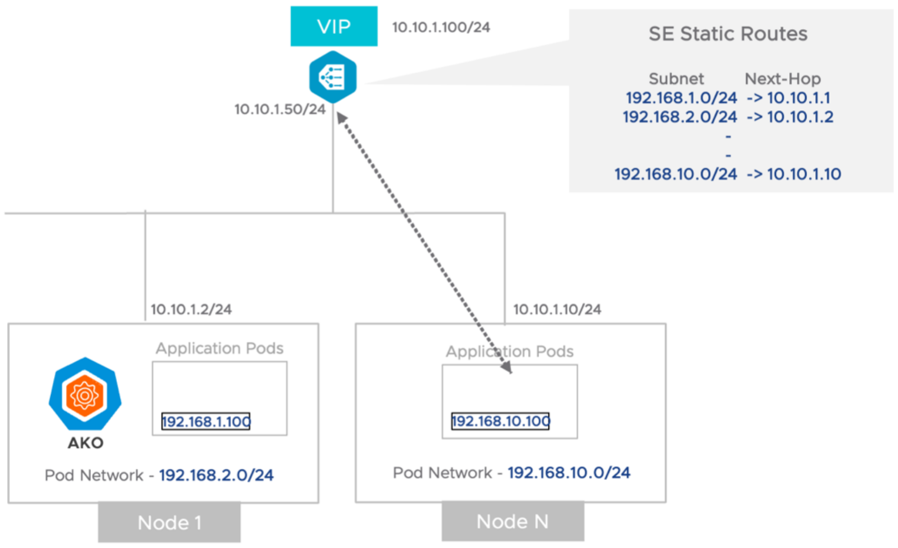
- Set the `ControllerSettings.tenant` Name to the tenant created in the earlier steps.
- The `avicredentials.username` and `avicredentials.password` to the user credentials created above.

1.7 AKO Deployment modes

AKO could be deployed in any of the following modes as configured in ako values.yaml manifest file:

- CusterIP mode

  In ClusterIP mode of AKO deployment, AKO automates the configuration of static routes on Avi mapping the pod CIDRs to the corresponding node IPs as next hop. AKO configures the backend pools with pod IPs as the pool members. The SEs must have a vNIC directly connected to the node network to route the backend traffic destined for the pod IPs via the respective next hops configured as shown in the example below:



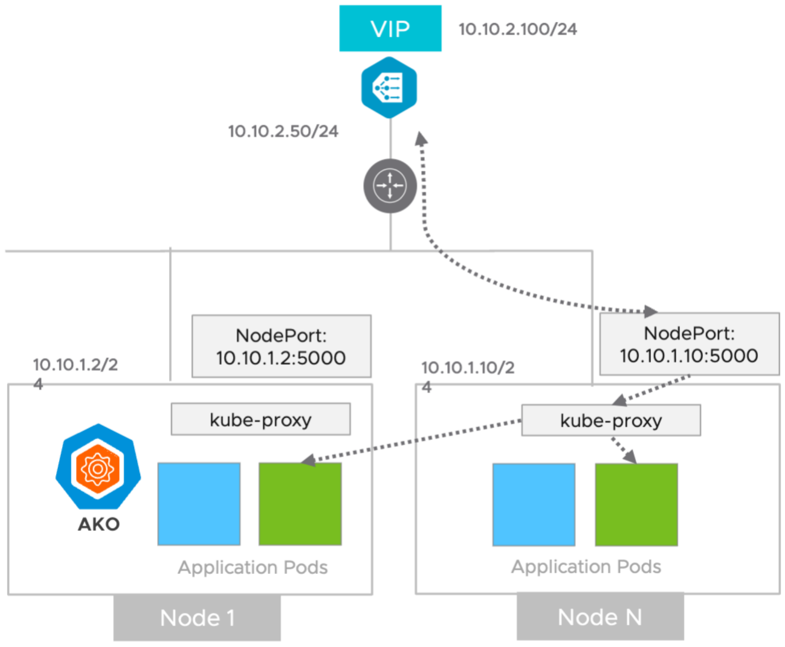
Some of the key design requirements include:

- Separate/one SE group per cluster because static routes are labeled per cluster
- SE data vNic must be directly connected to node network
- Ingress backend service must be ClusterIP which is the default

This mode of deployment allows SEs to health monitor the pods directly as well as implement any session persistence to pod IPs based on the use cases.

- NodePort mode

In NodePort mode of AKO deployment, the backend services are exposed via the node IPs and Port. Configuration of static routes are not required as all the node IPs are routable. AKO configures the backend pools with node IPs and port as the pool members. The traffic landing on the nodes on specific port are proxied to the corresponding backend pods by the kube-proxy as shown in the example below:

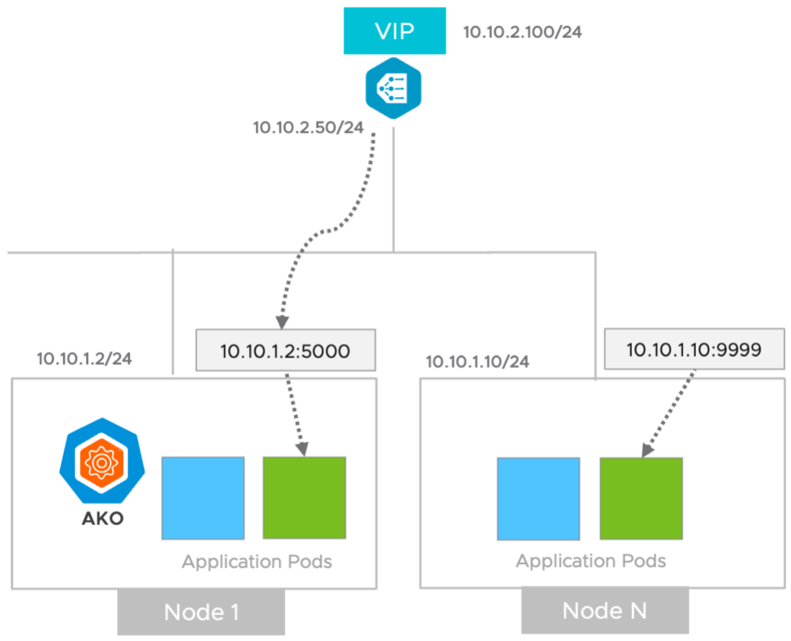


Some of the key design requirements include:

- Ingress backend service must be of type NodePort
- SE groups can be shared across multiple clusters
- AKO NodePort mode is agnostic of the CNIs on the cluster, but it is recommended to use only the tested/verified CNIs published by VMware.

- NodePortLocal mode

In NodePortLocal mode of AKO deployment, each Pod is individually exposed as NodePort. Pods belonging to the same service can be exposed using different Ports on different nodes. AKO configures the backend pools with node IPs and NodePorts corresponding to the backend pods. No static routes are required as Node IPs are routable.



Some of the key design requirements include:

- CNI must be Antrea with NodePortLocal feature enabled
- Ingress backend service must be ClusterIP which is the default
- SE groups can be shared across clusters
- SE vNic can be on any network which can route to the Kubernetes/Openshift nodes
- SEs can health monitor individual Pods
- Connections can be persisted to individual Pod IPs

The following table summarises the comparison across AKO deployment modes:

AKO Deployment	ClusterIP	NodePort	NodePortLocal
SE-Pod Connectivity	Direct to Pod IP	To Node IP and Port	Direct to Pod exposed as Node IP and Port
Network requirements	SE must be directly connected to Node network	SE can be connected to any routable subnet	SE can be connected to any routable subnet
SE Group scope	Dedicated SE group per cluster	SE group can be shared across clusters	SE group can be shared across clusters
Health monitor	To Pod	To service	To Pod
Persistence	Supported	Not supported	Supported
Ingress backend service type	ClusterIP	NodePort	ClusterIP
CNI	Any supported	Any supported	Antrea 0.13+

1.8 VS Sharding

AKO follows hostname based sharding to sync multiple ingresses with same hostname to a single virtual service. When an Ingress object is created with multiple hostnames, AKO generates an md5 hash using the hostname and the Shard VS number. This uniquely maps an FQDN to a given Shared VS and avoids DNS conflicts. During initial clean bootup, if the Shared virtual service does not exist in Avi, AKO creates the same and then patches the ingress FQDN to it either in the form of a pool (for insecure routes) or in the form of an SNI child virtual service (in case of secure routes).

For ingresses created with an insecure host/path combination, AKO creates a corresponding Avi pool object and patches the pool on one of the existing shard virtual services. The shard virtual service has a DataScript associated with it that reads the host/path of the incoming requests and appropriately selects a pool by matching it with the priority label specified for each pool member (corresponding to a host/path combination).

For secure ingresses, an SNI virtual service is created which although is a dedicated virtual service, does not have any IP addresses associated with it. The SNI virtual service is a child to a parent virtual service and is created based on the secret object specified in the ingress file against the host/path that is meant to be accessed securely.

In the current AKO model, the Shard VS size is an enum. It allows 3 pre-fixed sets of values:

- LARGE (8 virtual services)
- MEDIUM (4 virtual services)
- SMALL (1 virtual service)
- DEDICATED (Shard Mode is disabled, dedicated virtual services would be created per hostname)

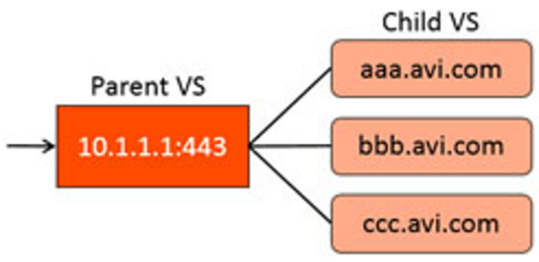
The decision of selecting one of these sizes for Shard virtual service depends on the size of the Kubernetes cluster's ingress requirements. It is recommended to always go with the highest possible Shard virtual service number – LARGE - to account for expansion in the future.

1.9 Virtual Hosting

Virtual hosting enables multiple domain names to be hosted on a SSL enabled VSVIP. Avi used the concept of parent and child virtual services for virtual hosting. AKO supports SNI based virtual hosting by default as well as EVH based virtual hosting. The parent virtual service governs the networking properties used to negotiate TCP and SSL with the client.

- SNI virtual hosting

SNI based virtual hosting is applicable only to TLS enabled virtual services. When client connects to the parent VSVIP via SSL, Avi inspects the domain field in the TLS Hello packet to select a child virtual service hosting the FQDN (as shown below) and the connection in bound to the proper child VS and appropriate SSL Certificate is presented to the client by the child VS to terminate the TLS connection.



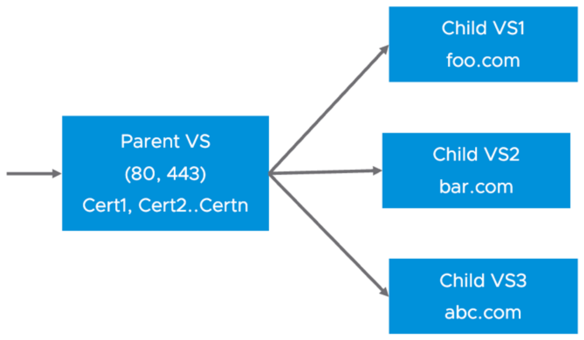
The parent virtual service will receive all new client TCP connection handshakes, which will be reflected in the statistics. The parent delegates to the child during the SNI phase of the TLS handshake and once a child virtual service is selected, the connection is internally handed off to a child virtual service, so subsequent metrics such as packets, concurrent connections, throughput, requests, logs and other statistics will only be recorded on the child virtual service. Similarly the child virtual service will not have logs for the initial TCP or SSL handshakes, such as the SSL version mismatch errors, which are recorded at the parent virtual service.

The child virtual service does not have an IP address or service port. Instead, it points to a parent virtual service, which must be created first. The domain name field is a fully qualified name requested by the SNI-enabled client within the SSL handshake. The parent matches the client request with the child's configured domain name. It does not match against the configured SSL certificate. The child may use a wildcard or domain specific certificate.

If switch to child virtual service did not happen, then the connection/request is handled on the parent virtual service, and hence the SSL and L7 state of the parent gets applied. The default certificate on the parent is presented to the client and the default pool is used for LB.

- Enhanced virtual hosting

  Virtual service placement for EVH service follow the same conditions as SNI parent child. In AKO deployment using EVH, each child VS is mapped to different hostnames. In the child virtual service, FQDN field is used to specify the domain for which the virtual service should be selected. Connection is always handled by the parent virtual service, but the requests will be handled by the selected child virtual service based on matching host header configured under child virtual service as shown below.



  SNI can only handle HTTPS traffic whereas EVH can handle both HTTP and HTTPS traffic. With EVH enabled, host rule CRD's could be applied to insecure ingress as well.

1.10 AKO Objects

AKO listens to the events in the control plane and automatically translates Kubernetes/Openshift objects to equivalent Avi objects and configures them on the controller using the API calls. The object translation is as mentioned below:

- Service type: LoadBalancer

  AKO creates a Layer 4 virtualservice object in Avi corresponding to a service of type loadbalancer in Kubernetes/Openshift. AKO creates a dedicated virtual service for this object in Avi controller reserving a virtual IP for it using the IPAM. AKO also supports the `spec.loadBalancerIP` field usage where-in the corresponding Layer 4 virtualservice objects are created with the user provided IP. For latest Kubernets versions. AKO has introduced a new annotation `ako.vmware.com/load-balancer-ip` to L4 service that will be used to specify preferred IP for Kubernetes versions 1.24 or later.

The layer 4 virtual service uses a pool section logic based on the ports configured on the service of type loadbalancer. AKO selects the pods associated with this service as pool servers associated with the virtualservice. AKO `autoFQDN` setting in values.yaml could be configured to add FQDNs for Service of type Loadbalancer if the DNS IPAM profile is configured. AKO also supports the external-dns format for specifying layer 4 FQDNs using the annotation `external-dns.alpha.kubernetes.io/hostname` on the Loadbalancer object.

- Insecure ingress

For insecure host/path combinations, AKO uses a Sharded VS logic where based on the hostname value (myhost.avi.internal), a pool object is created on a Shared VS when shard VS size is LARGE or MEDIUM or SMALL. A shared VS typically denotes a virtualservice in Avi that is shared across multiple ingresses. A priority label is associated on the poolgroup against it's member pool (that is created as a part of this ingress). An associated datascript object with this shared virtual service is used to interpret the host fqdn/path combination of the incoming request and the corresponding pool is chosen based on the priority label as mentioned above.

With DEDICATED shard VS size, AKO will create a normal VS (no virtual hosting enabled) for each unique host for secure/insecure ingress/route. AKO will apply all host rule specific settings, SSL profile, SSL KeyandCertificate on VS. Redirecting traffic to appropriate pool will be done using a httppolicyset object attached to VS. For secure ingress, there will httppolicyset attached to VS which will redirect traffic from port 80 to 443.

- Secure ingress

When shard VS size is LARGE or MEDIUM or SMALL, AKO creates an SNI child VS to a parent shared VS for the secure hostname. The SNI VS is used to bind the hostname to an sslkeycert object. The sslkeycert object is used to terminate the secure traffic on Avi's service engine. AKO parses the attached secret object and appropriately creates the sslkeycert object in Avi.

On the SNI VS, AKO creates httppolicyset rules to route the terminated (insecure) traffic to the appropriate pool object using the host/path specified in the rules section of this ingress object. Additionally - for these hostnames, AKO creates a redirect policy on the shared VS (parent to the SNI child) for this specific secure hostname. This allows the client to automatically redirect the http requests to https if they are accessed on the insecure port (80).

1.11 AKO Sync

AKO Namespace Sync feature allows the user to sync ingresses/routes from specific namespace/s with Avi controller. To use this feature, set the value of the following parameters to a non-empty string.

Parameter	Description
`AKOSettings.namespaceSelector.labelKey`	Key used as a label based selection for the namespaces.
`AKOSettings.namespaceSelector.labelValue`	Value used as a label based selection for the namespaces.

This key-value pair represent a label that is used by AKO to filter out namespace/s. If either of the above values is left empty then AKO would sync objects from all namespaces with Avi controller. Any changes in values of these parameters will require AKO reboot. Once user boots up AKO with this setting, user has to label a namespace with same key:value pair mentioned in values of labelKey and labelValues.

Similarly It might not be desirable to have all the nodes of a kubernetes cluster to participate in becoming server pool members. Hence key/value (nodeSelectorLabels.key and nodeSelectorLabels.value) is used as a label based selection on the nodes in Kubernetes/openshift to participate in NodePort. If key/value are not specified then all nodes are selected.

1.12 AKO HA

AKO Active/Standby mode supported from AKO 1.9.1 allows the user to run two instances of AKO in a Kubernetes/OpenShift cluster. Active and passive modes are assigned automatically by performing a leadership election among the AKO pods. HA enabled by changing the `replicaCount` in values.yaml to two.

The leader AKO performs all Avi functions like any other standalone AKO pod. Leader election is done via Kube API server:

- It begins with the creation of a lease object in the avi-system namespace, where the leader periodically renews/locks the lease at regular intervals defined by `renewTime` as a way of informing other replicas regarding its leadership
- The follower(s) poll the lease object to check if leader has the lease/lock, if not they try to acquire the lease and become the leader. Lease/Poll interval is small enough to make sure that the follower takes over before leader reboots and tries to acquire the lease.
- If the leader pod is shutdown, it gives up the lease while shutting down
- The follower acquires the lease on the next poll.

The follower AKO performs the following functions:

- Polls the lease object in the avi-system namespace.
- Reads the objects in Kubernetes/OpenShift cluster and populates the cache.
- Reads the Avi objects configured by Active AKO and builds the cache.
- Takes over as leader if the lease is not locked

1.13 AKO CRDs

AKO Custom Resource Definitions (CRDs) are used to extend the Kubernetes APIs server with additional schemas.

AKO categorizes the CRDs into:

- Layer 7

  These CRD objects are used to express layer 7 traffic routing rules. The following Layer7 CRDs are currently available with AKO:

  - HostRule CRD:

    The CRD is used to express additional virtual host properties. The virtual host FQDN is matched from either the Kubernetes ingress or OpenShift route-based objects. The HostRule CRD could be used to enable/disable virtual host, express httppolicyset object ref, WAF policy ref, custom application profile ref, custom analytics profile ref, Datascript ref, TLS configuration, configure GSLB FQDN, analytics policy, TCP settings etc.

    HostRule CRD Ref

  - HTTPRule CRD

The HTTPRule CRD is primarily targetted for the developers. While the path matching rules in the Ingress/Route objects would define traffic routing rules to the microservices, the HTTPRule CRD can be used as a complimentary object to control additional layer 7 properties like: algorithm, hash, tls re-encrypt use cases.

HTTPRule CRD Ref

- Infrastructure: AviInfraSetting CRD

  These CRD objects are used to control Avi's infrastructure components like Ingress class, SE group properties etc. AviInfraSetting provides a way to segregate Layer-4/Layer-7 VirtualServices to have properties based on different underlying infrastructure components, like ServiceEngineGroup, intended VIP Network etc.

  AviInfraSetting CRD Ref

1.14 AKO compatibility

Refer to the latest AKO version compatibility matrix.

1.15 AKO Sizing Recommendations

The Controller and SE sizing are done based on the latest sizing recommendations for the specific Avi versions, features enabled/consumed and also as per scale parameters wrt SSL/application traffic and number of config objects like virtual service, HTTP policies etc.

Sizing Ref:

- Controller sizing
- SE sizing
- System limits: Configmax

1.16 AKO Design Decisions

The following table lists the design decisions for the Avi AKO design.

Design Decision ID	Design Decision	Design Justification	Design Implication
NSX ALB- AKO-001	Decide on Avi Essentials, Avi Basic or Avi Enterprise licensing tiers	Full featured Enterprise licensing tier is recommended for production deployments	Avi controller could be configured with only any one of the licensing Tiers. Enterprise license tier is needed to support most of the use cases in PROD deployments.
NSX ALB- AKO-002	Leverage existing Avi controller cluster or deploy new Avi controller cluster	New controller cluster could be considered depending on the licensing tier, sizing and scalability requirements	Additional resources to host new controller cluster.
NSX ALB- AKO-003	IaaS Platform where Avi controller cluster would be deployed	Deployment environment for Avi controllers and Service Engines	Same controller cluster could be used for centralized management across multiple clouds
NSX ALB- AKO-004	IaaS Platform where Avi SE would be deployed	Decide on supported / recommended cloud types based on customer infrastructure / use case.	Pre-requisites and Restrictions around different cloud types. Example – NSX-T Write cloud supports only Single Arm deployment. Also NSX-T T1 gateway has to be configured in the AKO values.yaml file.

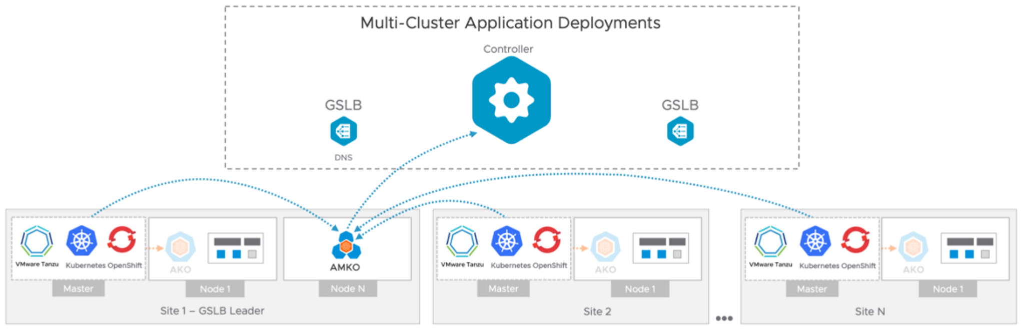
NSX ALB- AKO-005	CNI type – routable v/s non-routable pod CIDRs	Decides backend routing options like routing directly to the pods using AKO ClusterIP / NodePortLocal or routing directly to NodePort.	Direct health monitoring of pods and session persistence cannot be implemented for NodePort mode deployment. Performance overheads with kube-proxy. Turn off AKO static route sync for routable CNI's
NSX ALB- AKO-006	SE single arm or dual arm	Depending on customer use case/deployment. ClusterIP needs direct SE connectivity to node network and needs dual arm to route between separate VIP network and node network. NSX-T Write cloud supports only Single Arm deployment. vCenter Write cloud can support dual arm for ClusterIP or NodePort or NodePortLocal.	Need static routing to backend networks via the VIP network gateway to force single arm deployment in case of vCenter Write cloud. Also enable "Prefer static route v/s directly connected networks" in the cloud configurations for VIP/pool placements based on static routing.
NSX ALB- AKO-007	AKO deployment mode – ClusteIP or NodePort or NodePortLocal	Depending on customer requirements / use cases / cloud type / POD network reachability. NSX-T Write cloud supports NodePort or NodePortLocal - with Antrea CNI. NodePort or NodePortLocal is recommended for dual arm deployments	ClusterIP and NodePortLocal needs the backend service to be exposed as ClusterIP. NodePort needs the backend services to be exposed as NodePort. NodePortLocal is only supported with Anrea CNI and needs it to be enabled in the CNI. ClusterIP needs separate SE group per cluster in multi-cluster deployment leading to additional resource / license implications.

NSX ALB- AKO-008	Tenancy	Depending on customer requirement / use case. Provider mode - AKO objects created in tenant context, but shared SE's in Admin tenant. Tenant mode – SE's created in tenant context and AKO objects created in tenant context for complete resource isolation. Provider mode with all AKO objects created in Admin tenant but access restricted using granular RBAC controls	Additional resources and license implications for Tenant mode deployment.
NSX ALB- AKO-009	AKO HA	Depending on the scale of each cluster.	AKO runs a single instance of a stateless pod. On restart AKO needs to build its caches from kube API server as well as from the controller. Large environments could take up to 10-12 minutes.
NSX ALB- AKO-010	AKO sync scope	Depending on customer requirement the scope of AKO sync could be restricted to selective name spaces or nodes using labels.	Lesser objects to be synced. Optimizes the resource utilization and scale.
NSX ALB- AKO-011	AKO shard size (LARGE / MEDIUM / SMALL / DEDICATED)	Depending on customer requirement / scale. Recommend shard size of LARGE for large scale deployments and/or to accommodate future expansion plans.	Preserves public IP addresses. Example - In shard size of LARGE all ingresses would be shared across 8 x VSVIPs and for the shard size of SMALL all ingresses would be hosted on 1 x VSVIP. Ease of operations with fewer subnets/NICs.

NSX ALB- AKO-012	SNI or EVH based virtual hosting	EVH based virtual hosting is recommended for additional flexibility and control over SNI based virtual hosting (Default).	Needs `AKOSettings.enableEVH` to be set to true in AKO values.yaml SNI can only handle HTTPS traffic whereas EVH can handle both HTTP and HTTPS traffic. With EVH enabled, host rule CRD's could be applied to insecure ingress as well
NSX ALB- AKO-013	Multiple AKO instances	Run multiple instances of AKO per cluster to create namespace-based isolation.	Deploying only single AKO instance in K8s Cluster with large number of Ingress / Service deployment could lead AKO to take longer time to sync Ingress / Service state to the Controller.
NSX ALB- AKO-014	VIP Per Namespace	Parent VS per Namespace in EVH mode	All Ingresses across namespace will share a set of VIP as per Shard Size
NSX ALB- AKO-016	AKOSetttings.layer7Only	AKO will only do layer 7 loadbalancing	AKO sync both Ingress and L4 Service to the Controller.
NSX ALB- AKO-017	Share a single L4 VIP across multiple LoadBalancer services using - AKOSetttings.servicesAPI	Use this flag to enable AKO to watch over Gateway API CRDs i.e. GatewayClasses and Gateways. AKO only supports Gateway APIs with Layer 4 Services.	Setting servicesAPI to true would enable users to configure GatewayClass and Gateway CRDs to aggregate multiple Layer 4 Services and create one VirtualService per Gateway Object. If servicesAPI is not set to true, AKO would create separate VS for each L4 by default.
NSX ALB- AKO-018	Share a single L4 VIP across multiple LoadBalancer services using Annotations for Openshift clusters.	Sharing of single VIP among multiple services is achieved by providing the annotation (ako.vmware.com/enable-shared-vip: "vip-name") to multiple LoadBalancer Services, where VIP sharing is intended.	AKO would create separate VS for each L4 by default unless the (ako.vmware.com/enable-shared-vip: "vip-name") annotation is configured in the services definition.

2.0 Ingress Global Server Load Balancing

The Avi Multi-Cluster Kubernetes Operator (AMKO) facilitates application deployments across multiple OpenShift/Kubernetes clusters. AMKO is a Kubernetes pod that runs in a namespace called `avi-system`.



AMKO is aware of the following object types:

- Kubernetes:
  – Ingress
  – Service type load balancer

- OpenShift:
  – Routes
  – Service type load balancer

Installation of AMKO is done via Helm which applies the following permissions via a Kubernetes service account for GSLBConfig and GlobalDeploymentPolicy objects:

- CREATE
- GET
- LIST
- READ
- UPDATE

AMKO also requires permissions to read the ingress and service objects for all the member clusters. For this, a separate kubeconfig file is created with all the required permissions from all the clusters and is passed to AMKO via an OpenShift/Kubernetes secret object. To access a kubernetes/openshift cluster via a kubeconfig file, the cluster context must be defined. The cluster context defines the access parameters for a user on how they can access a cluster. Each cluster context is a combination of:

- Cluster: This section contains a list of kubernetes/openshift clusters.
  – cluster.certificate-authority-data: CA cert of the server
  – cluster.server: URL of the kubernetes/openshift API server
  – name: name of this cluster that it should be identified with.

- User: This section defines a list of users and their credentials. The user credentials determine what resources and verbs a user has access to in a given cluster.
  – user.client-certificate-data: client certificate for the user.
  – user.client-key-data: client key for the user.
  – name: name of this user that it should be identified with.

- Context: This section defines the cluster accesses by associating a cluster, a user and a namespace. This cluster context is then used to access the resources of a kubernetes/openshift cluster.
  - context.cluster: Name of the cluster as defined in the clusters section.
  - context.user: Name of the user as defined in the users section.
  - name: name of this context that it should be identified with.

AMKO assumes that it has connectivity to all the member clusters' OpenShift/Kubernetes API servers. Without this, AMKO will not be able to watch over the Kubernetes and OpenShift resources in the member clusters.

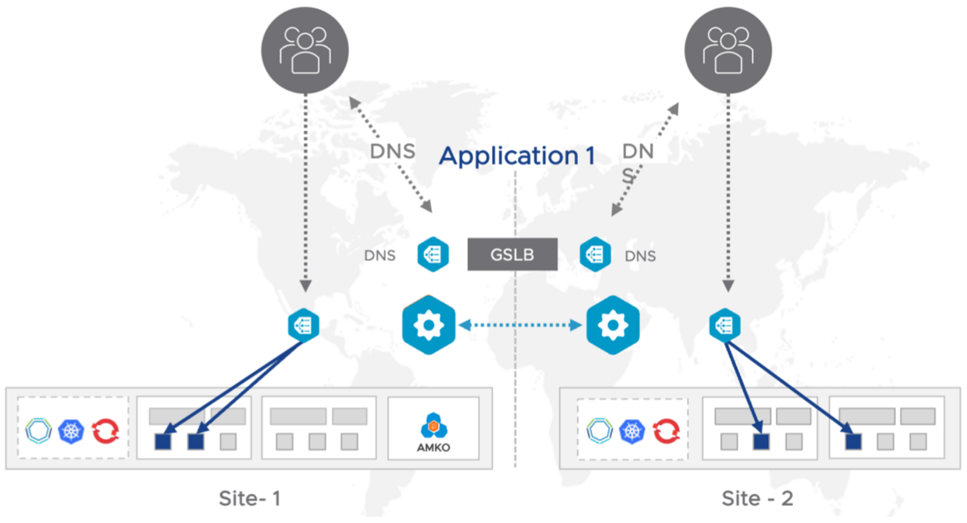
Before deploying AMKO, one of the clusters have to be designated as the leader and rest of the clusters as followers. AMKO has to be deployed on all clusters (wherever federation is required). This is to ensure that the leader cluster's AMKO would federate the AMKO config objects like GSLBConfig and GlobalDeploymentPolicy objects to all follower clusters.

AMKO is configured via two CRDs - GSLBConfig and GlobalDeploymentPolicy. The Helm based installation procedure will automatically create these two in the specified namespaces. During AMKO installation, set the required parameters via values.yaml. These parameters are translated to the GSLBConfig and the GlobalDeploymentPolicy objects.

values.yaml - AMKO parameters ref

With the AMKO installation based on the parameters configured above, Helm creates the objects GSLBConfig and GlobalDeploymentPolicy. If appSelector and namespaceSelector were not specified via the Helm installation (values.yaml), the respective fields will be empty in the resulting GlobalDeploymentPolicy object. This means that by default, no objects are selected.
The GlobalDeploymentPolicy object could be edited to add the required criteria for app or cluster selection. The GDP object can be edited at runtime to change the application selection parameters, traffic split and the applicable clusters. AMKO will recognize these changes and will update the GSLBServices accordingly.

The below diagram depicts the multi-cluster GSLB deployment with AMKO:



1. Avi GSLB provides DNS-based Global Load-Balancing for multi-cluster environments
2. AMKO automates the configuration and management of Avi GSLB by acting as a Kubernetes operator for multi-cluster operations.
3. AMKO runs in a cluster with connectivity to all the Kubernetes clusters and Avi Controllers.
4. Avi AKO provides service discovery for the underlying ingresses/routes/services deployed through Avi and GSLB leverages this.

2.1 AMKO Federation

AMKO uses federation to replicate AMKO configuration to a set of member clusters. This ensures a seamless recovery of AMKO configuration during disasters. The set of clusters registered on AMKO is considered as a federation set. One of these clusters in the federation set must be designated as the **leader**. And, all the other clusters are designated as the **followers**.

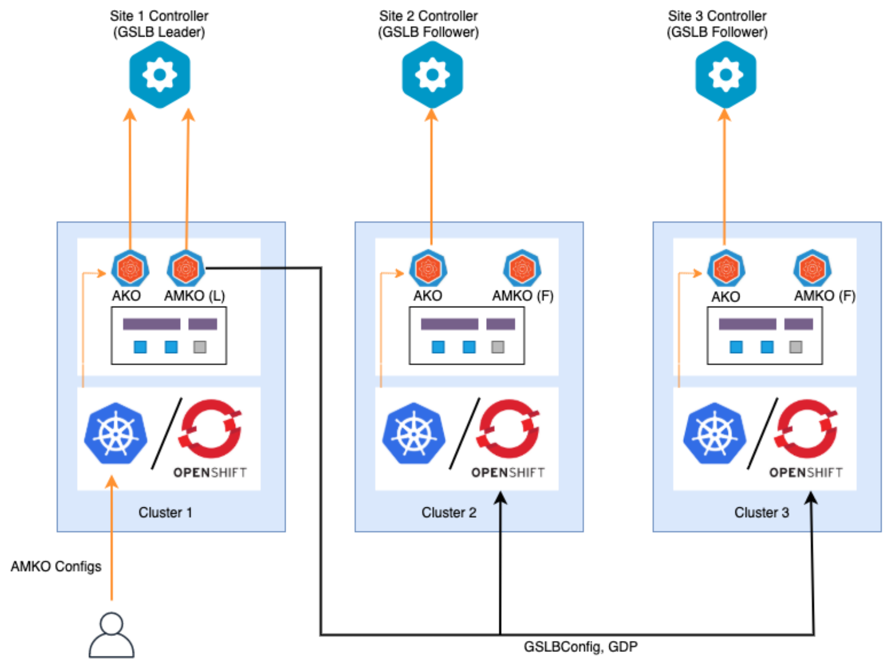
- **Leader Cluster**: Is responsible for distributing the AMKO configuration to all the follower clusters in the federation set. AMKO in the leader cluster is also responsible to create/update GSLB Services on the Avi Controller.
- **Follower Cluster**: AMKO in a **follower** cluster runs passively and does not carry out federation activities or GslbService sync operations unlike the leader.

Post a disaster, the user could manually pick any of the erstwhile follower clusters and designate that as the new leader. The user must ensure that only a single AMKO is designated as the **leader** at any given point in time.

The following objects are federated from the leader cluster to the follower:

- GSLBConfig
- GlobalDeploymentPolicy or GDP

Add/Update/Delete events of these objects are federated to the follower clusters. The below diagram depicts the multi-cluster federated GSLB deployment with AMKO:



Each site has an Avi Controller deployed. The site 1's Avi Controller is chosen as the GSLB Leader and all the other sites are GSLB followers. AMKO is deployed on all three sites. For cluster 1 (in site 1), it is marked as the leader. Clusters 2 and 3, are marked as followers.

On adding or updating the GSLBConfig and GDP objects in cluster 1, AMKO's federator in cluster 1 federates the changes to these objects to all the follower clusters. Federation is currently a one way communication from the leader AMKO to the follower AMKOs. AMKO federator on the leader cluster reacts to the create/update/delete operations on the GSLBConfig and GDP objects on the leader cluster. Modification of these objects on the follower cluster will not prompt the federator on the leader to update these objects.

During a cluster down event on the leader AMKO, the federation of config objects will stop. However, at this point, all other clusters participating in federation would be synced with up to date configuration of the erstwhile GSLB leader. Hence, switching to a new AMKO leader does not require any manual steps of recovering the AMKO config objects.

2.2 AMKO CRDs

AMKO uses the following CRDs for custom deployments:

- GlobalDeploymentPolicy (GDP) CRD

A CRD called GlobalDeploymentPolicy allows users to select kubernetes/openshift objects based on certain rules. The selection policy applies to all the clusters which are mentioned in the GDP object. Only one GDP object is allowed. If using helm install, a GDP object is created by picking up values from values.yaml file. User can then edit this GDP object to modify their selection of objects.

A combination of appSelector and namespaceSelector will decide which objects will be selected for GSLB service consideration.

- appSelector: Selection criteria only for applications:
  – label: will be used to match the ingress/service type load balancer labels (key:value pair).
- namespaceSelector: Selection criteria only for namespaces:

– label: will be used to match the namespace labels (key:value pair).

The behavior is as detailed in the table below:

appSelector	namespaceSelector	Result
yes	yes	Select all objects satisfying appSelector and from the namespaces satisfying the namespaceSelector
no	yes	Select all objects from the selected namespaces (satisfying namespaceSelector)
yes	no	Select all objects satisfying the appSelector criteria from all namespaces
no	no	No objects selected (default action)

The GDP CRD could be used to express global settings for custom traffic split ratio to route percentage of traffic to a particular cluster, health monitor ref, site persistence profile ref, ttl, pool algorithm etc. The GDP object can be edited at runtime to change the application selection parameters, traffic split and the applicable clusters. AMKO will recognize these changes and will update the GSLBServices accordingly.

GDP CRD Ref

- GSLBConfig CRD

  GSLBConfig CRD has been provided to customize the GSLB configuration. Only one GSLBConfig object is allowed. If using helm install, a GSLBConfig object is created by picking up values from the values.yaml file. This CRD defines the GSLB leader controller IP, credentials, member clusters etc.

  GSLBConfig CRD Ref

- AMKOCluster CRD

  A CRD called AMKOCluster governs the federation. If Helm is used to deploy AMKO, this Custom Resource will be installed, based on the values provided via values.yaml.

  AMKO requires a CRD called AMKOCluster to federate the following objects to a list of member clusters:

  – GSLBConfig object
  – GlobalDeploymentPolicy object (GDP)

The CRD specified if the AMKO in the current cluster is leader. Default value is false. If set to false, AMKO won't sync any objects to the Avi Controller, and the AMKO federator won't federate the objects to the member clusters. The CRD also defines the member cluster list on which federation will be performed.

AMKOCluster CRD Ref

- GSLBHostRule CRD

The GSLBHostRule CRD allows users to override certain properties of a specific GslbService object on the Avi Controller created by AMKO. The CRD could be used to express specific GSLB service settings for custom traffic split ratio to route percentage of traffic to a particular cluster, health monitor ref, site persistence profile ref, ttl, pool algorithm etc. overriding the default settings defined in the global GDP CRD.

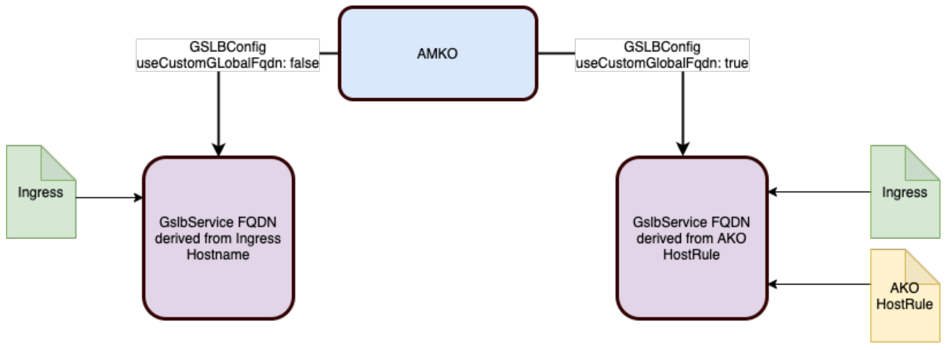
GSLBHostRule CRD Ref

Certain Gslb Service properties can be set and modified at runtime. If these are set through the GDP CRD object, they are applied to all the Gslb Services. GSLBHostRule CRD object could be used to override specific properties for a GslbService.

Properties	Configured via
TTL	GDP, GSLBHostRule
Site Persistence	GDP, GSLBHostRule
Custom Health Monitors	GDP, GSLBHostRule
Third party members	GSLBHostRule
Traffic Split	GDP, GSLBHostRule
Pool Algorithm Settings	GDP, GSLBHostRule

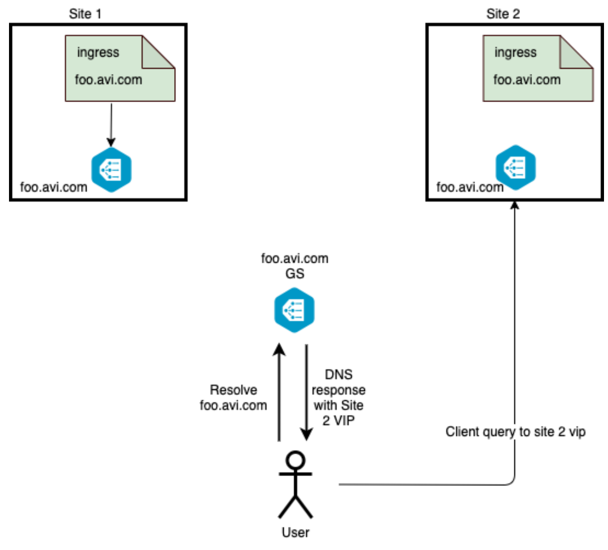
2.3 GSLB Service FQDN

AMKO decides the fully qualified domain name (FQDN) for the GSLB services based on two modes.


GSLBConfig
useCustomGLobalFqdn: false
AMKO
GSLBConfig
useCustomGlobalFqdn: true
Ingress
GslbService FQDN derived from Ingress Hostname
GslbService FQDN derived from AKO HostRule
Ingress
AKO HostRule

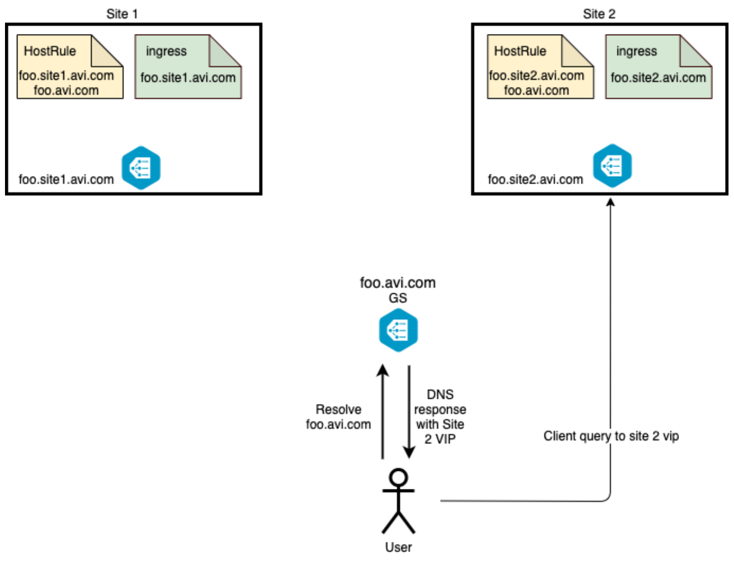
- Default Mode

  In this mode, the hostname(s) field in the status of the Ingress/Route/Service of type Loadbalancer object is used to determine the hostname of the GSLB Service. Each hostname uniquely maps to a GS FQDN automatically. For common hostname across clusters, a single GSLB Service is created with pool members from each cluster that share the hostname. When you have two instances of an application deployed on two different clusters which have the same FQDNs, and you want to expose all these application as GslbServices in one shot, use the Default Mode.



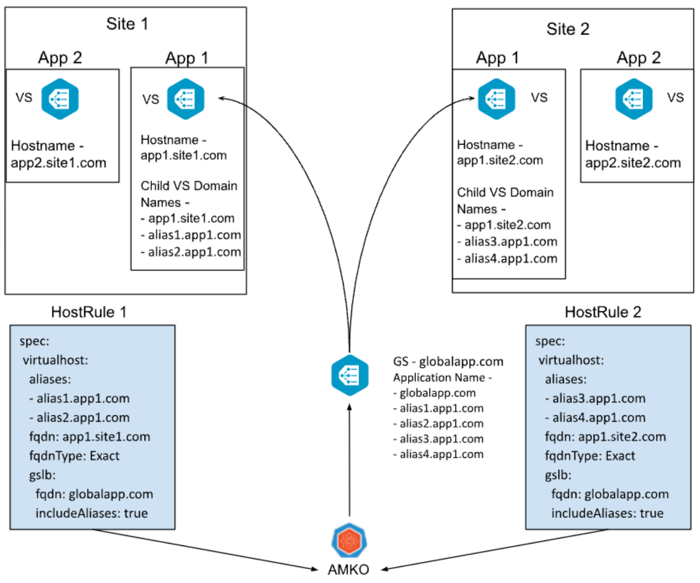
- Custom Global FQDN Mode

  In this mode, AMKO checks the AKO HostRules to figure out the GslbService FQDN. To expose an application via GSLB, the user must provide a mapping between the local FQDN and the global FQDN via AKO's HostRule object. If the user wishes to map two application instances (which have different FQDNs) in two different clusters, they have to create HostRules in both clusters specifying the mapping.



Currently when using local to global FQDN mapping (useCustomGlobalFqdn: true), AMKO creates a GSLB service for ingress/route/service only if there is a HostRule with local to global FQDN mapping.

The number of GSLB services can be reduced by grouping related (FQDN and its aliases) together into one GSLB Service using the field called includeAliases introduced in the HostRule CRD. On setting includeAliases to *True*, AMKO creates a GSLB Service with name equal global-fqdn. All the aliases mentioned in the hostrule will be added as Application Names in the GSLB Service as shown in the example below.



```
spec:
 virtualhost:
  aliases:
  - alias1.app1.com
  - alias2.app1.com
  fqdn: app1.site1.com
  fqdnType: Exact
  gslb:
   fqdn: globalapp.com
   includeAliases: true
```

```
spec:
 virtualhost:
  aliases:
  - alias3.app1.com
  - alias4.app1.com
  fqdn: app1.site2.com
  fqdnType: Exact
  gslb:
   fqdn: globalapp.com
   includeAliases: true
```

2.4 AMKO Sizing Recommendations

The AMKO deployment sizing recommendations are similar to GSLB sizing recommendations.

GSLB Sizing Ref

2.5 AMKO Compatibility

Refer to the latest AMKO version compatibility matrix.

2.6 AMKO Design Decisions

All the design decisions listed in the GSLB use case section are applicable for AMKO deployment as well. The following table lists the design decisions specific to the Avi AMKO design.

Design Decision ID	Design Decision	Design Justification	Design Implication
NSX ALB- AMKO-001	Leader site	Select the most resilient site to configure as AMKO Leader site.	Needs manual intervention to promote the AMKO follower to leader when current AMKO leader is unavailable.
NSX ALB- AMKO-002	GSLB scope	Select clusters participating in GSLB in multi-cluster/site deployment to perform GSLB across multiple instances of the same applications deployed across these clusters/sites.	AMKO also requires permissions to read the ingress and service objects for all the member clusters participating in the GSLB as defined in the cluster contexts in the kubeonfig file.

NSX ALB- AMKO-003	Federation scope	Select clusters where AMKO federation needs to be enabled.	AMKO needs to be deployed in all clusters/sites part federation. AMKO objects will not be replicated/federated to the sites not part of AMKOCluster CRD (derived from AMKO values.yaml manifest file).
NSX ALB- AMKO-004	AMKO sync scope	A combination of appSelector and namespaceSelector will decide which objects will be selected for GSLB service consideration.	No objects selected if no labels are defined for both appSelector and namespaceSelector (default action)
NSX ALB- AMKO-005	GSLB Service FQDN	Default mode or Custom Global FQDN mode as per customer requirement / use case. Enable Aliases if needed.	Defines the global GSLB application FQDN published to the users.

References

For more information, see the following supplemental configuration and administration guides, white papers and best practices documents.

Release Notes:

- AKO Release Notes
- AMKO Release Notes

Sizing Guidelines:

- Avi Controller Sizing
- Avi Service Engines Sizing
- VMware configmax tool

Additional resources:

- Product Interop tool
- Avi Documentation
- AKO Overview
- AMKO Overview
- GSLB Overview
- WAF Overview