



Solo Enterprise for Istio on vSphere Kubernetes Service on VMware Cloud Foundation

Reference Architecture

Table of Contents

Executive Summary 3

 vSphere Kubernetes Service 3

 Solo Enterprise for Istio 4

Solution Architecture..... 7

 VKS Cluster Configuration 8

 Solo Enterprise Packages 10

Solution Validation.....10

 Install Solo Enterprise for Istio on VKS 10

 Step 1 — Environment..... 10

 Step 2 — Pod Security Admission (VKS-specific).....11

 Step 3 — Gateway API CRDs.....11

 Step 4 — Shared root CA11

 Step 5 — cert-manager issuer and istio-csr (each cluster) 12

 Step 6 — Install Solo Enterprise for Istio with Helm (each cluster) 13

 Step 7 — Check the mesh components (each cluster) ..14

 Step 8 — East-west gateways (each cluster)14

 Step 9 — Peer the clusters.....15

 Step 10 — Bookinfo sample application (each cluster) ..16

 Step 11 — Solo UI16

 Validate Solo Enterprise for Istio on VKS 17

Conclusion 23

Executive Summary

Teams running applications on vSphere Kubernetes Service (VKS) need consistent security, traffic control, and visibility for service-to-service traffic, often across more than one cluster. Solo Enterprise for Istio provides that layer as a supported, hardened distribution of Istio built around ambient mode, a sidecarless data plane that adds mutual TLS, identity-based authorization, and L7 traffic management without changing or restarting application pods.

This reference architecture documents the installation and validation of Solo Enterprise for Istio on VKS clusters running on VMware Cloud Foundation (VCF). It covers:

- Ambient mode on VKS. A per-node ztunnel proxy secures all mesh traffic with mTLS. Optional waypoint proxies add L7 policy only where it is needed.
- Multicluster mesh. Two or more VKS clusters are peered through east-west gateways so services can discover and call each other across clusters, with failover and a shared root of trust.
- Solo UI. A single management view of every cluster in the mesh: service graph, workload health, workloads, and policies.
- Built-in telemetry. OpenTelemetry collectors installed with the UI gather request rate, latency, and error metrics from ztunnel, plus Kubernetes object state, from every cluster. No waypoint proxies or application changes are needed.

The result is a repeatable pattern that VKS platform teams can apply with the same Helm-based workflow across clusters, using VKS for Kubernetes lifecycle and Solo Enterprise for Istio for the service network.

vSphere Kubernetes Service

vSphere Kubernetes Service (VKS) is the Kubernetes runtime built directly into VMware Cloud Foundation (VCF). With CNCF certified Kubernetes, VKS enables platform engineers to deploy and manage Kubernetes clusters while leveraging a comprehensive set of cloud services in VCF. Cloud admins benefit from the support for N-2 Kubernetes versions, enterprise grade security, and simplified lifecycle management for modern apps adoption.

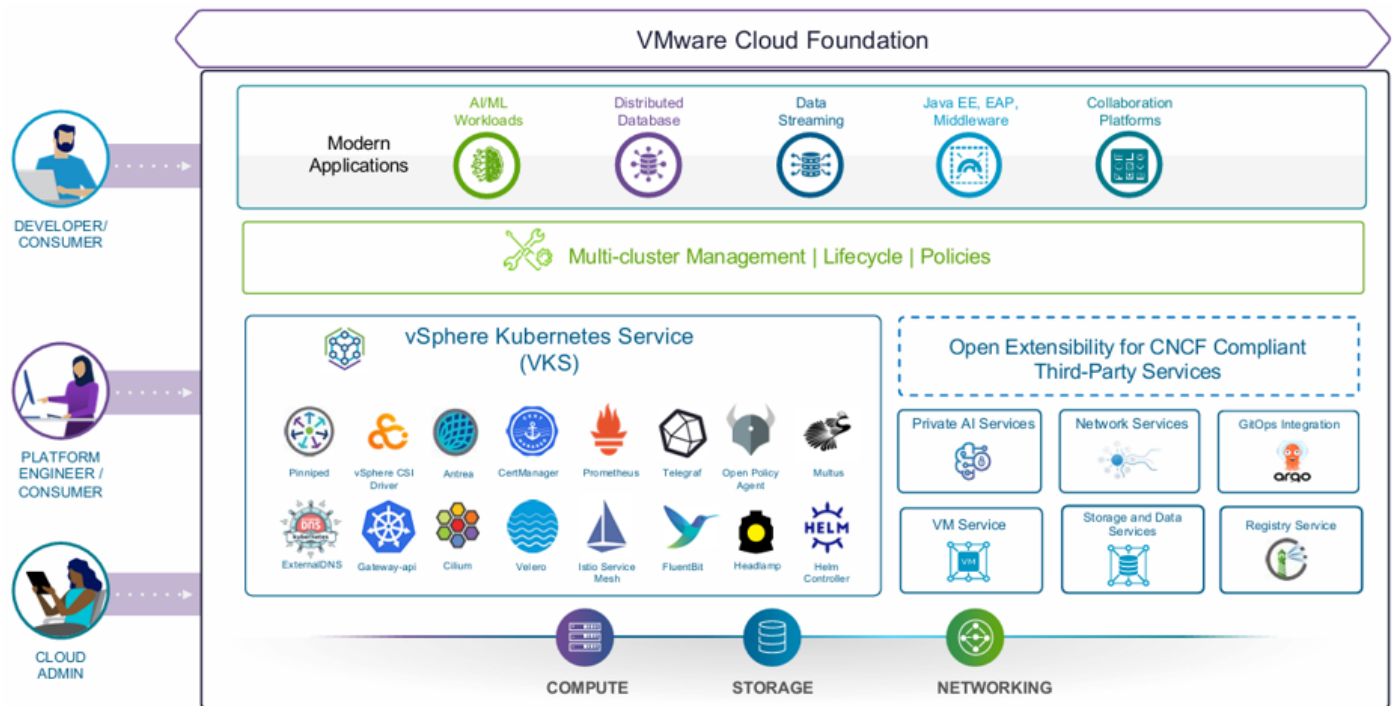


Figure 1: VKS on VCF Ecosystem

Benefits of using VKS with modern applications:

Lower TCO: With VKS organizations can reduce silos, leverage existing tools and skill sets without having to retrain staff and/or change existing processes. Utilizing unified lifecycle management across infrastructure components to stay up to date with the most recent patches and minimizing security risks.

Operational Simplicity: VKS is engineered for unparalleled operational simplicity, leveraging the familiarity of existing vSphere tools, skills, and workflows. This design philosophy significantly reduces the learning curve for IT teams and streamlines management processes. With VKS, organizations benefit from automated cluster provisioning, which accelerates deployment times and minimizes manual configuration errors. Furthermore, its robust capabilities extend to automated upgrades and comprehensive lifecycle management. This integrated approach ensures consistent operations, reduces overhead, and frees up valuable resources to focus on innovation rather than infrastructure maintenance.

Run and Manage Kubernetes at Scale: Effortlessly deploy and manage Kubernetes clusters at scale, leveraging a built-in, Cloud Native Computing Foundation (CNCF) certified Kubernetes distribution. VKS provides fully automated lifecycle management, streamlining operations from initial setup to ongoing maintenance and upgrades. This comprehensive approach ensures that organizations can harness the power of Kubernetes for their containerized applications with unparalleled efficiency and reliability, without the complexities typically associated with large-scale Kubernetes deployments.

Solo Enterprise for Istio

[Solo Enterprise for Istio](#) is Solo.io's enterprise distribution of the open source Istio service mesh. Solo.io is a founding contributor to Istio ambient mode and maintains long-term support builds, FIPS-compliant images, and security patches for each release. The enterprise distribution [adds features](#) not available in upstream Istio, including multicluster peering for ambient mode and the Solo UI.

Key capabilities validated in this architecture:

Capability	What it does on VKS
Ambient data plane	ztunnel runs as a DaemonSet on every VKS node and handles L4 traffic: mTLS, workload identity (SPIFFE), L4 authorization, and telemetry. Applications join the mesh with a namespace label; no sidecar injection.
Waypoint proxies	Optional per-namespace or per-service proxies for L7 features: HTTP routing, retries, fault injection, and L7 authorization. Deployed only where needed.
Multicloud peering	East-west gateways connect VKS clusters. Services marked as global are discoverable from every peered cluster, and traffic between clusters stays mTLS-encrypted end to end.
cert-manager integration	istio-csr lets cert-manager issue all mesh certificates, so the mesh uses the same PKI and issuers as the rest of the platform. VKS provides cert-manager as an add-on.
Solo UI	Multicloud dashboard for mesh topology, workloads, and traffic. Runs on one cluster with a lightweight relay on each other cluster.
Telemetry pipeline	OpenTelemetry collectors in each cluster forward mesh metrics and Kubernetes resource changes to a ClickHouse store on the management cluster. Distributed tracing can be added with waypoints and application trace headers; it was not part of this validation.
Enterprise support	Long-term support releases, -solo images, and 24x7 support from Istio maintainers. [VERIFY: LTS/FIPS wording with product marketing]

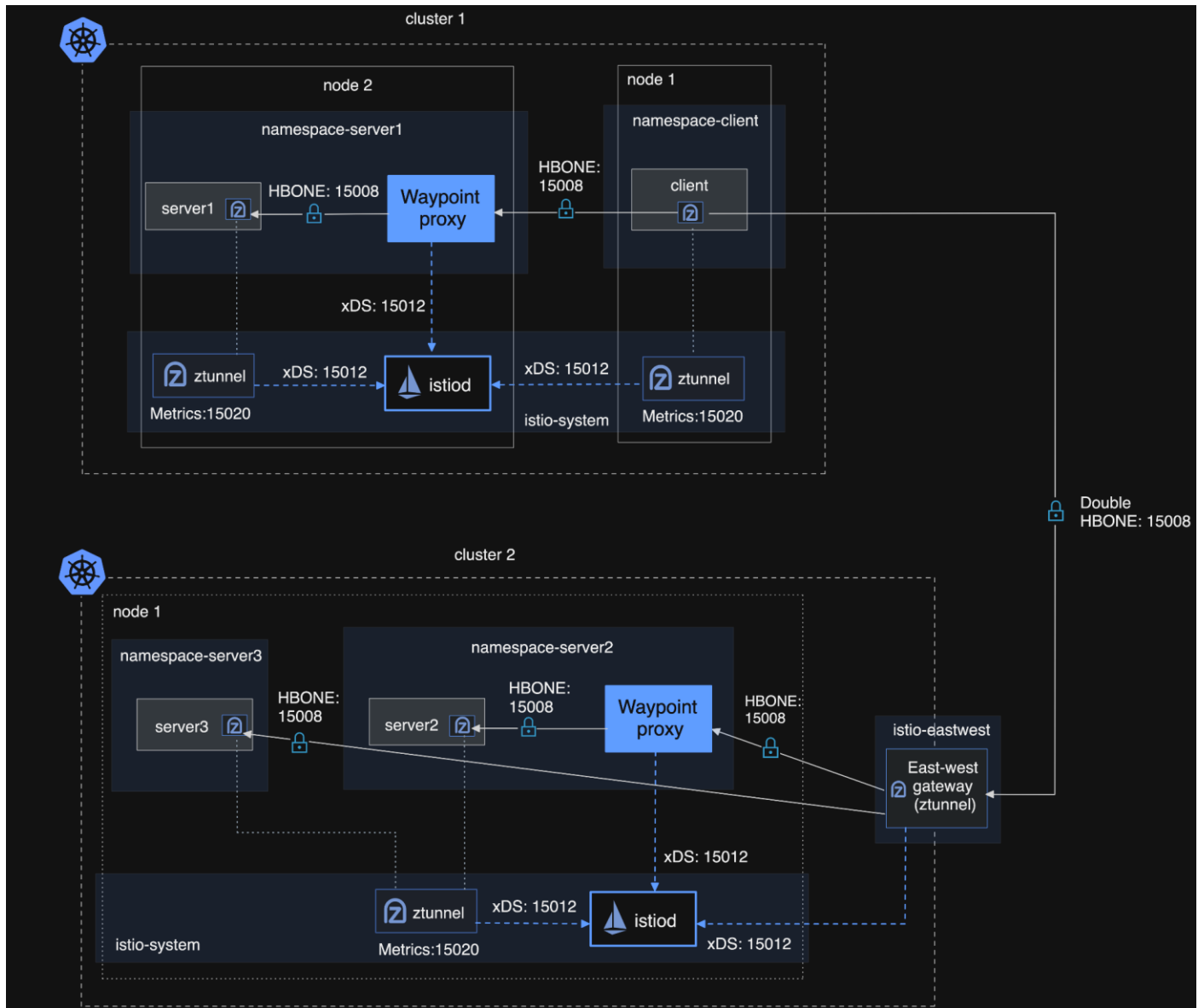


Figure 2: Solo Enterprise for Istio ambient mode architecture

Solution Architecture

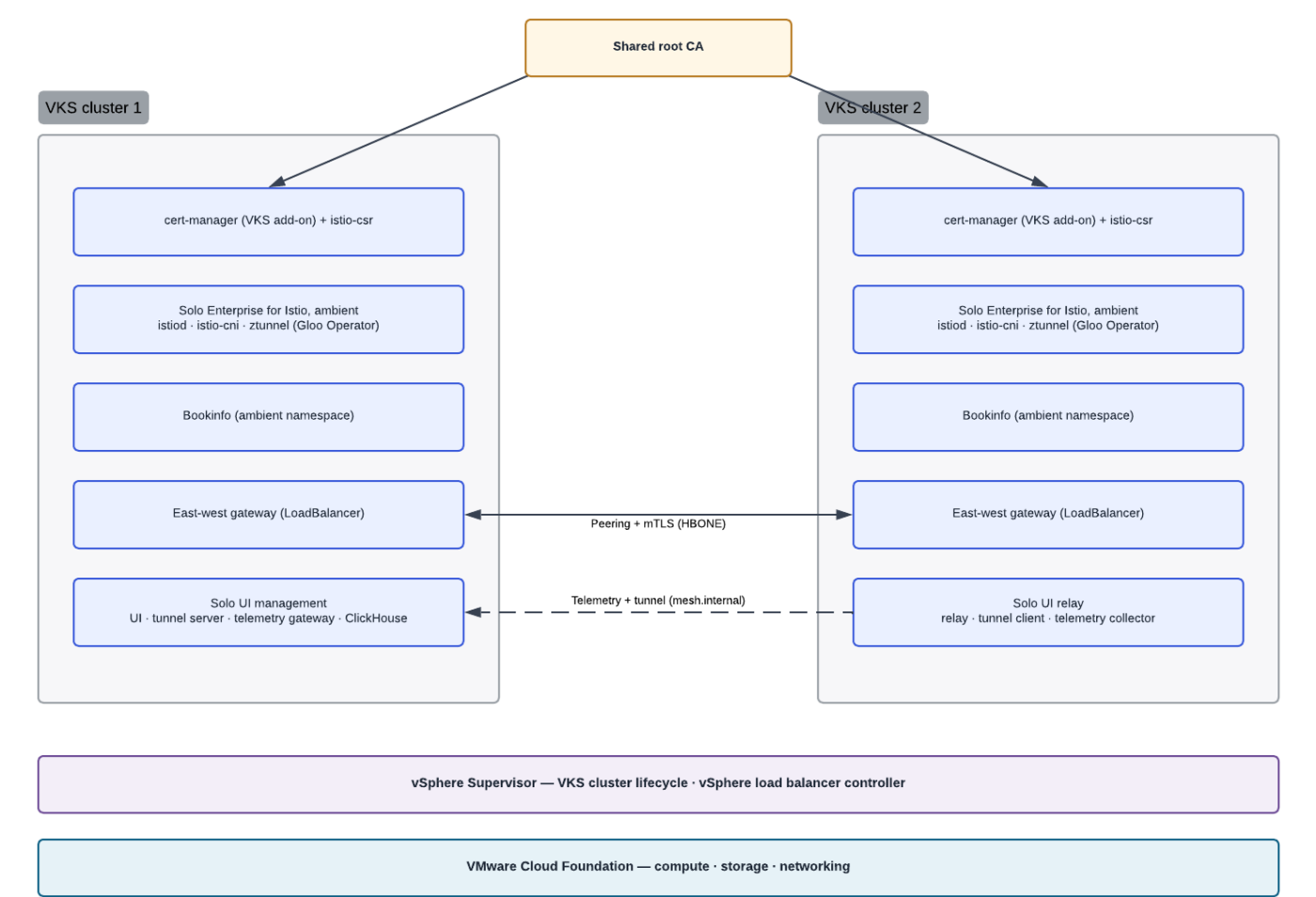


Figure 3: Solo Enterprise for Istio multicluster ambient mesh on VKS

At its core, the Supervisor control plane provides essential services, such as the Harbor container registry, the Contour ingress controller, and the Argo CD GitOps utility. For more information on vSphere Supervisor services and installation, visit: <https://techdocs.broadcom.com/us/en/vmware-cis/vcf/vsphere-supervisor-services-and-standalone-components/latest.html>

This control plane is responsible for the automated deployment and lifecycle management of CNCF-compliant Kubernetes clusters via the vSphere Kubernetes Service (VKS) via Cluster API. The specific version of VKS used in this white paper is 3.6.

Table 1: Installed Supervisor Components

Component	Version	Notes
vSphere Kubernetes Service	3.6	Allows for CNCF K8s clusters

The validated topology uses two VKS clusters, **cluster1** and **cluster2**, provisioned from the same Supervisor. On each cluster, Helm installs the Solo Enterprise for Istio control plane (istiod) and ambient data plane (istio-cni and ztunnel). The clusters are joined into one mesh

through east-west gateways, exposed as **LoadBalancer** services that the Supervisor's standard vSphere load balancer controller provides. Each cluster holds a static peering entry for the other cluster's east-west gateway address.

Note: This reference architecture validates a two-cluster deployment and demonstrates Solo's multi-cluster capabilities, but the same components run on a single VKS cluster, and teams can add clusters to the mesh later by deploying an east-west gateway and peering, without reinstalling the clusters already in place.

Mesh certificates are issued by cert-manager rather than by istiod. On each cluster, istio-csr receives certificate signing requests from istiod and ztunnel and passes them to a cert-manager **ClusterIssuer** backed by a root CA shared by both clusters. Because both clusters chain to the same root and use the same trust domain (**cluster.local**), workloads in either cluster can authenticate each other over mTLS.

cluster1 also hosts the Solo UI management plane: the UI, a tunnel server, a telemetry gateway, and a ClickHouse database. **cluster2** runs a relay that connects to the management plane through the mesh, using the management services' global ***.mesh.internal** hostnames, and forwards telemetry and Kubernetes object snapshots. The UI also shows the vSphere zone of each cluster (for example, **domain-c10**), which the vSphere cloud provider sets on the nodes.



Figure 4: Mesh certificate issuance with cert-manager and istio-csr

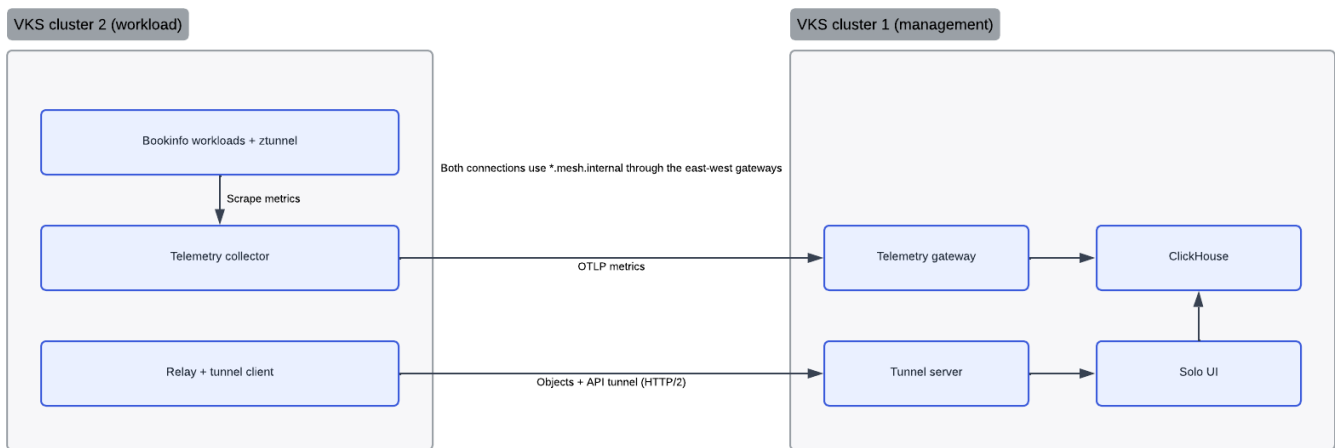


Figure 5: Solo UI telemetry and management data flow

VKS Cluster Configuration

To showcase Solo Enterprise for Istio on VKS, we deployed two VKS clusters, each cluster consisted of one control plane node and three worker nodes. Storage was provided by vSAN

Express Storage Architecture (ESA), using the default RAID 5 storage policy. Transparently, the vSphere CSI driver exposed this policy to an adjacent Kubernetes storage class, ***vsan-esa-default-policy-raid5***.

Worker node resources were configured using a *best-effort-large* profile, whilst control plane nodes utilized the *best-effort-medium* profile.

Table 2: Kubernetes Cluster Configuration Summary

Component	Configuration	Notes
Kubernetes Control Plane	1 Replica — 2x vCPU 8GB RAM vSAN RAID 5	VM Class: “best-effort-medium” Storage Class: “vsan-esa-default-policy-raid5”
Kubernetes Worker Nodes	3 Replicas — 4x vCPU 16GB RAM vSAN RAID 5	VM Class: “best-effort-large” Storage Class: “vsan-esa-default-policy-raid5”
Kubernetes Release	v1.35.5	
OS Image	Ubuntu 24.04	Kernel version 6.8.0-90-generic

The VKS cluster configuration is defined declaratively in the YAML file below. As previously outlined, applying this manifest to the Supervisor triggers the creation of the cluster, which is then managed and reconciled by the Cluster API. Both VKS clusters are using the same configuration and the YAML is defined below:

```
# vks-cluster.yaml

apiVersion: cluster.x-k8s.io/v1beta1
kind: Cluster
metadata:
  name: vks-cluster
  namespace: supervisor-namespace
spec:
  clusterNetwork:
    pods:
      cidrBlocks:
        - 192.168.144.0/20
    services:
      cidrBlocks:
        - 10.96.0.0/12
      serviceDomain: cluster.local
  topology:
    class: builtin-generic-v3.6.0
    classNamespace: vmware-system-vks-public
    version: v1.35.5--vmware.1-vkr.1
    variables:
      - name: kubernetes
        value:
          certificateRotation:
            enabled: true
            renewalDaysBeforeExpiry: 90
      - name: vmClass
        value: best-effort-medium
      - name: storageClass
        value: vsan-esa-default-policy-raid5
```

```
controlPlane:
  replicas: 1
  metadata:
    annotations:
      run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu,os-version=24.04
workers:
  machineDeployments:
    - class: node-pool
      name: worker-pool
      replicas: 3
      metadata:
        annotations:
          run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu,os-version=24.04
      variables:
        overrides:
          - name: vmClass
            value: best-effort-large
```

Solo Enterprise Packages

Table 3: Solo Enterprise Packages

Package	Version	Notes
Solo Enterprise for Istio	1.31.1-solo	Solo.io Helm charts: base, istiod, cni, ztunnel, peering
Solo Enterprise UI	0.5.8	Solo.io add-on

Solution Validation

Install Solo Enterprise for Istio on VKS

All steps below are in `install-multicluster.sh`. Each step can be run on its own, for example `./install-multicluster.sh peering ui`.

The generic instructions for installing Solo Enterprise for Istio in a multi-cluster configuration using Helm can be found [here](#).

Prerequisites

- Two VKS clusters, with a kubeconfig containing a context for each. In this validation the two cluster kubeconfigs were merged into one file separated by kube contexts.
- `LoadBalancer` service support on both clusters (east-west gateways).
- cert-manager on both clusters, installed as the VKS cert-manager add-on.
- A Solo Enterprise for Istio license key with the multicluster entitlement.
- kubectl, helm 3, openssl.

Step 1 — Environment

```
export SOLO_ISTIO_LICENSE_KEY=<license-key>
export ISTIO_VERSION=1.31.1
export ISTIO_IMAGE=${ISTIO_VERSION}-solo
export HELM_REPO=us-docker.pkg.dev/soloio-img/istio-helm

export cluster1=cluster1 context1=<cluster1-context>
```

```
export cluster2=cluster2 context2=<cluster2-context>

bash <(curl -sSfL https://raw.githubusercontent.com/solo-io/doc-examples/main/istio/install-istioctl.sh)
export PATH=${HOME}/.istioctl/bin:${PATH}
```

Step 2 — Pod Security Admission (VKS-specific)

VKS enforces Pod Security Admission. The istio-cni and ztunnel DaemonSets, the east-west gateway, the Solo UI, and the Bookinfo sample were each blocked until their namespace was labeled. Label all of them before installing:

```
for ctx in ${context1} ${context2}; do
  for ns in istio-system istio-eastwest solo-enterprise bookinfo; do
    kubectl --context ${ctx} create namespace ${ns} --dry-run=client -o yaml | kubectl
--context ${ctx} apply -f -
    kubectl --context ${ctx} label --overwrite namespace ${ns} \
      pod-security.kubernetes.io/enforce=privileged \
      pod-security.kubernetes.io/enforce-version=latest
  done
done
```

If a namespace is labeled after its workloads were created, restart them (for example `kubectl rollout restart daemonset -n istio-system ztunnel`).

Step 3 — Gateway API CRDs

```
for ctx in ${context1} ${context2}; do
  kubectl --context ${ctx} apply -f https://github.com/kubernetes-sigs/gateway-
api/releases/download/v1.4.0/standard-install.yaml
done
```

Step 4 — Shared root CA

A self-signed root CA is generated once and installed in both clusters. `pathlen:1` allows an intermediate CA per cluster in production.

```
mkdir -p certs
cat > certs/root-openssl.cnf <<'EOF'
[ req ]
prompt = no
distinguished_name = dn
x509_extensions = v3_ca

[ dn ]
O = Solo Enterprise for Istio on VKS
CN = root-cert

[ v3_ca ]
basicConstraints = critical, CA:TRUE, pathlen:1
keyUsage = critical, keyCertSign, cRLSign
subjectKeyIdentifier = hash
EOF

openssl req -x509 -sha256 -nodes -days 3650 -newkey rsa:2048 \
-keyout certs/root-key.pem -out certs/root-cert.pem \
```

```
-config certs/root-openssl.cnf -extensions v3_ca
```

Production note for the doc: in this validation the root CA key is stored in each cluster and signs workload certificates directly. In production, keep the root offline and give each cluster's `ClusterIssuer` an intermediate CA from the enterprise PKI (for example, Vault or a cloud private CA).

Step 5 — cert-manager issuer and istio-csr (each cluster)

```
helm repo add jetstack https://charts.jetstack.io --force-update

for i in 1 2 do
  cluster=$(eval echo \${cluster$i}); ctx=$(eval echo \${context$i})

  kubectl --context ${ctx} create secret tls istio-ca -n cert-manager \
    --cert=certs/root-cert.pem --key=certs/root-key.pem
  kubectl --context ${ctx} create secret generic istio-root-ca -n cert-manager \
    --from-file=ca.pem=certs/root-cert.pem

  kubectl --context ${ctx} apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: istio-ca
spec:
  ca:
    secretName: istio-ca
EOF

  helm upgrade -i cert-manager-istio-csr jetstack/cert-manager-istio-csr \
    --kube-context ${ctx} \
    --version 0.18.0 \
    --namespace cert-manager \
    --wait \
    --set "app.certmanager.issuer.name=istio-ca"
    --set "app.certmanager.issuer.kind=ClusterIssuer"
    --set "app.certmanager.issuer.group=cert-manager.io"
    --set "app.tls.rootCAFile=/var/run/secrets/istio-csr/ca.pem"
    --set "app.tls.certificateDNSNames={cert-manager-istio-csr.cert-manager.svc,istio-csr.cert-manager.svc}"
    --set "app.server.caTrustedNodeAccounts=istio-system/ztunnel"
    --set "app.server.clusterID=${cluster}"
    --set "volumeMounts[0].name=root-ca"
    --set "volumeMounts[0].mountPath=/var/run/secrets/istio-csr"
    --set "volumes[0].name=root-ca"
    --set "volumes[0].secret.secretName=istio-root-ca"
done
```

Two settings differ from the istio-csr chart defaults and are required for an ambient multicluster mesh:

Setting	Why
<code>app.server.clusterID=<cluster></code>	istio-csr rejects CSRs from a cluster ID other than its own. The chart default is <code>Kubernetes</code> ; it must match <code>global.multiCluster.clusterName</code> in the istiod values.

Setting	Why
app.server.caTrustedNodeAccounts=istio-system/ztunnel	Lets ztunnel request certificates for the workloads on its node, which ambient mode requires.

Quote the `--set` values in bash. Unquoted, bash expands `{...}` (as in `certificateDNSNames`) into separate arguments.

Step 6 — Install Solo Enterprise for Istio with Helm (each cluster)

Four charts per cluster: `base` (CRDs), `istiod` (control plane), `cni`, and `ztunnel`. The `istiod` and `ztunnel` values turn off the built-in `istiod` CA and send certificate requests to `istio-csr`, set the cluster name and network used for peering, and set `IPV6_ENABLED=false` on `ztunnel`. The VKS clusters are IPv4-only; without that setting, `ztunnel` returns IPv6 addresses for `*.mesh.internal` services and the Solo UI relay on cluster2 cannot send telemetry to cluster1 (`dial tcp [2001:2::3]:4316: connect: network is unreachable`).

```
export REPO=us-docker.pkg.dev/soloio-img/istio

for i in 1 2; do
  cluster=$(eval echo \${cluster$i}); ctx=$(eval echo \${context$i})
  kubectl --context ${ctx} label --overwrite namespace istio-system
  topology.istio.io/network=${cluster}

  helm upgrade --install istio-base oci://${HELM_REPO}/base \
    --kube-context ${ctx} -n istio-system --version ${ISTIO_IMAGE} --wait -f - <<EOF
defaultRevision: ""
profile: ambient
EOF

  helm upgrade --install istiod oci://${HELM_REPO}/istiod \
    --kube-context ${ctx} -n istio-system --version ${ISTIO_IMAGE} --wait -f - <<EOF
global:
  hub: ${REPO}
  tag: ${ISTIO_IMAGE}
  caAddress: cert-manager-istio-csr.cert-manager.svc:443
  multiCluster:
    clusterName: ${cluster}
    network: ${cluster}
  proxy:
    clusterDomain: cluster.local
meshConfig:
  accessLogFile: /dev/stdout
  trustDomain: cluster.local
  defaultConfig:
    proxyMetadata:
      ISTIO_META_DNS_CAPTURE: "true"
env:
  ENABLE_CA_SERVER: "false"
  PILOT_ENABLE_IP_AUTOALLOCATE: "true"
  PILOT_SKIP_VALIDATE_TRUST_DOMAIN: "true"
  DISABLE_LEGACY_MULTICLUSTER: "true"
pilot:
  cni:
    namespace: istio-system
    enabled: true
platforms:
```

```

    peering:
      enabled: true
profile: ambient
license:
  value: ${SOLO_ISTIO_LICENSE_KEY}
EOF

helm upgrade --install istio-cni oci://${HELM_REPO}/cni \
  --kube-context ${ctx} -n istio-system --version ${ISTIO_IMAGE} --wait -f - <<EOF
ambient:
  dnsCapture: true
excludeNamespaces:
  - istio-system
  - kube-system
global:
  hub: ${REPO}
  tag: ${ISTIO_IMAGE}
profile: ambient
EOF

helm upgrade --install ztunnel oci://${HELM_REPO}/ztunnel \
  --kube-context ${ctx} -n istio-system --version ${ISTIO_IMAGE} --wait -f - <<EOF
hub: ${REPO}
tag: ${ISTIO_IMAGE}
variant: distroless
profile: ambient
caAddress: cert-manager-istio-csr.cert-manager.svc:443
multiCluster:
  clusterName: ${cluster}
network: ${cluster}
istioNamespace: istio-system
namespace: istio-system
configValidation: true
proxy:
  clusterDomain: cluster.local
terminationGracePeriodSeconds: 29
env:
  L7_ENABLED: "true"
  IPV6_ENABLED: "false"
EOF
done

```

Step 7 — Check the mesh components (each cluster)

```

for ctx in ${context1} ${context2}; do
  helm --kube-context ${ctx} list -n istio-system
  kubectl --context ${ctx} rollout status deployment/istiod -n istio-system
  kubectl --context ${ctx} rollout status daemonset/istio-cni-node -n istio-system
  kubectl --context ${ctx} rollout status daemonset/ztunnel -n istio-system
done

```

If istio-csr is reinstalled or reconfigured after the mesh is running, restart ztunnel so it requests new certificates: `kubectl rollout restart daemonset -n istio-system ztunnel`.

Step 8 — East-west gateways (each cluster)

```
for i in 1 2; do
```

```

cluster=$(eval echo \${cluster$i}); ctx=$(eval echo \${context$i})
helm upgrade -i peering-eastwest oci://${HELM_REPO}/peering \
  --kube-context ${ctx} \
  --version ${ISTIO_IMAGE} \
  --namespace istio-eastwest \
  -f - <<EOF
eastwest:
  create: true
  cluster: ${cluster}
  network: ${cluster}
  deployment: {}
EOF
done

kubectl --context ${context1} get svc -n istio-eastwest istio-eastwest
kubectl --context ${context2} get svc -n istio-eastwest istio-eastwest

```

Step 9 — Peer the clusters

Each cluster gets a remote peer entry that points at the other cluster's east-west gateway address.

```

export CLUSTER1_EW_ADDRESS=$(kubectl --context ${context1} get svc -n istio-eastwest
istio-eastwest \
  --jsonpath="{.status.loadBalancer.ingress[0]['ip','hostname']}") # 10.139.11.137
export CLUSTER2_EW_ADDRESS=$(kubectl --context ${context2} get svc -n istio-eastwest
istio-eastwest \
  --jsonpath="{.status.loadBalancer.ingress[0]['ip','hostname']}") # 10.138.250.104

helm upgrade -i peering-remote oci://${HELM_REPO}/peering \
  --kube-context ${context1} --version ${ISTIO_IMAGE} --namespace istio-eastwest \
  -f - <<EOF
remote:
  create: true
  items:
    - name: istio-remote-peer-${cluster2}
      cluster: ${cluster2}
      network: ${cluster2}
      addressType: IPAddress
      address: ${CLUSTER2_EW_ADDRESS}
      preferredDataplaneServiceType: loadbalancer
      trustDomain: "cluster.local"
EOF

helm upgrade -i peering-remote oci://${HELM_REPO}/peering \
  --kube-context ${context2} --version ${ISTIO_IMAGE} --namespace istio-eastwest \
  -f - <<EOF
remote:
  create: true
  items:
    - name: istio-remote-peer-${cluster1}
      cluster: ${cluster1}
      network: ${cluster1}
      addressType: IPAddress
      address: ${CLUSTER1_EW_ADDRESS}
      preferredDataplaneServiceType: loadbalancer
      trustDomain: "cluster.local"
EOF

```

Step 10 — Bookinfo sample application (each cluster)

```
for ctx in ${context1} ${context2}; do
    kubectl --context ${ctx} label --overwrite namespace bookinfo istio.io/dataplane-
mode=ambient
    kubectl --context ${ctx} -n bookinfo apply -f
https://raw.githubusercontent.com/istio/istio/${ISTIO_VERSION}/samples/bookinfo/platfor
m/kube/bookinfo.yaml -l 'account'
    kubectl --context ${ctx} -n bookinfo apply -f
https://raw.githubusercontent.com/istio/istio/${ISTIO_VERSION}/samples/bookinfo/platfor
m/kube/bookinfo.yaml -l 'app'
    kubectl --context ${ctx} -n bookinfo apply -f https://raw.githubusercontent.com/solo-
io/doc-examples/main/istio/sample-apps/productpage-with-curl.yaml
done
```

Apply `productpage-with-curl.yaml` last. It replaces `productpage-v1` with a version that adds `curl` and `netcat` containers for testing. If `bookinfo.yaml` is applied after it, `kubectl apply` removes those containers. Run test commands in the `curl` container: `kubectl exec deploy/productpage-v1 -c curl -- curl ....`

The namespace label is the only change needed to add the application to the mesh. The Bookinfo pods do not meet the VKS default Pod Security profile, which is why `bookinfo` was labeled in Step 2. That label is for the sample app and is not needed for ambient mode.

Step 11 — Solo UI

```
export SOLO_UI_VERSION=0.5.8

# Management plane on cluster1
helm upgrade -i solo-management \
oci://us-docker.pkg.dev/solo-public/solo-enterprise-helm/charts/management \
--kube-context ${context1} -n solo-enterprise \
--version ${SOLO_UI_VERSION} \
--set cluster=${cluster1} \
--set licensing.licenseKey=${SOLO_ISTIO_LICENSE_KEY} \
--set oidc.issuer="" \
--set products.mesh.enabled=true \
--set clickhouse.resources.requests.cpu=200m

# Relay on cluster2
helm upgrade -i solo-relay \
oci://us-docker.pkg.dev/solo-public/solo-enterprise-helm/charts/relay \
--kube-context ${context2} -n solo-enterprise \
--version ${SOLO_UI_VERSION} \
--set cluster=${cluster2} \
--set products.mesh.enabled=true \
--set tunnel.fqdn=solo-enterprise-tunnel.solo-enterprise.mesh.internal \
--set telemetry.fqdn=solo-enterprise-telemetry-gateway.solo-enterprise.mesh.internal

# Open the UI through its LoadBalancer address (10.139.11.140 in this validation) or a
port-forward
kubectl --context ${context1} get svc -n solo-enterprise solo-enterprise-ui
kubectl --context ${context1} port-forward -n solo-enterprise svc/solo-enterprise-ui
8080:80
```

The relay connects to the management plane's tunnel and telemetry services through their *.mesh.internal hostnames, so its traffic crosses the east-west gateways over mTLS. The solo-enterprise namespace does not need the ambient label for this. istioctl multicluster check on cluster1 reports these as globally shared services.

```
holuser@console:~/Desktop$ istioctl multicluster check
✓ Incompatible Environment Variable Check: all relevant environment variables are valid
✓ License Check: license is valid for multicluster
✓ CNI DNS Capture Check: AMBIENT_DNS_CAPTURE is enabled
✓ Pod Check (istiod): all pods healthy
✓ Pod Check (ztunnel): all pods healthy
✓ Pod Check (eastwest gateway istio-eastwest/istio-eastwest): all pods healthy
✓ Gateway Check: all eastwest gateways programmed
  ✓ istio-eastwest/istio-eastwest available at 10.139.11.137
✓ Peers Check: all clusters connected
  ✓ Connected to cluster2 via 10.138.250.104
✓ Duplicate Remote Cluster Check: no remote gateways share a cluster ID
✓ Shared Services Check: no globally shared services found
✓ Service Conflicts Check: no conflicting service aliases or hostnames found
✓ Peer Segments Check: no rejected peer segments found
=====
✓ Intermediate Certs Compatibility Check: root certificate SHA256 sum: 1c5a4ca38c4d85aa83e42e9a707c4144b9e548a222b941f0c595611bc08188a0
✓ Network Configuration Check: all network configurations are valid
✓ Duplicate Cluster ID Check: skipped (requires multiple cluster inputs)
✓ Stale Workloads Check: skipped (requires contexts for multiple clusters)
✓ Service Account Consistency Check: skipped (requires contexts for multiple clusters)
```

Figure 6: istioctl multicluster check on cluster1 — peering, license, and shared root certificate verified

Notes for the doc:

ClickHouse sizing. The management chart requests 2 CPU and 3 GiB memory for ClickHouse by default. The CPU request was lowered to 200m due to resource constraint. Production management clusters should keep the default request.

- **Authentication.** `oidc.issuer=""` enables the built-in identity provider for testing only. Production deployments should connect the UI to the organization's OIDC provider.

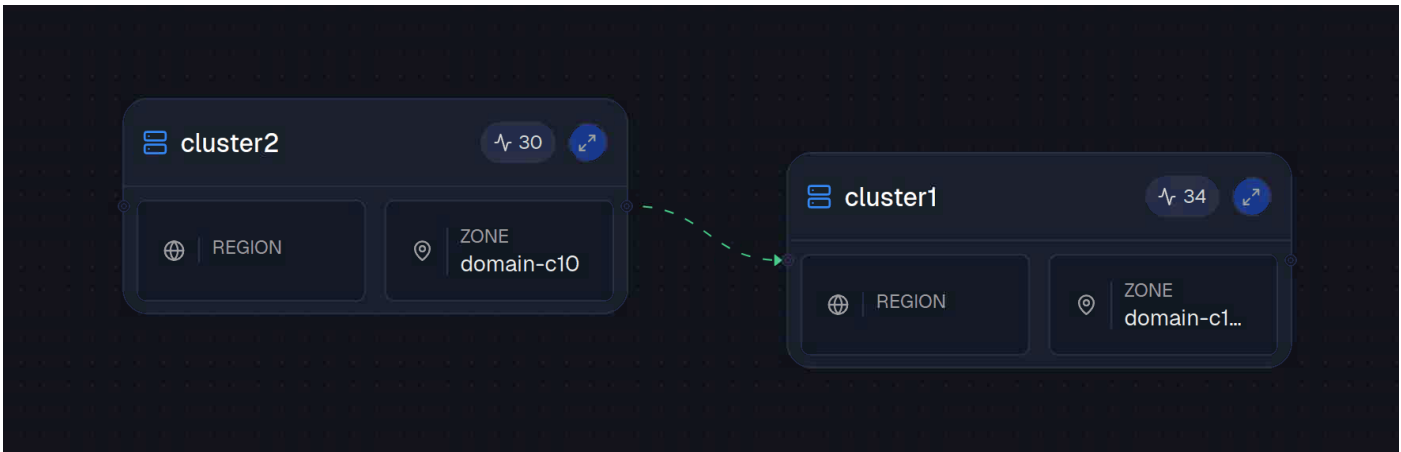


Figure 7: Solo UI showing both VKS clusters in the mesh, with vSphere zones

Validate Solo Enterprise for Istio on VKS

#	Check	Command / action	Result in this validation	Figure
1	Mesh version	istioctl version	Control plane and data plane report 1.31.1-solo on both clusters	—

#	Check	Command / action	Result in this validation	Figure
2	Mesh components running	kubectl get pods -n istio-system -o wide	istiod running; istio-cni-node and ztunnel on all 4 nodes (control plane included) on both clusters	—
3	Certificates issued by cert-manager	bookinfo-multicloud.sh certs (reads istioctl ztunnel-config certificate)	Bookinfo workloads on both clusters hold 1-hour leaf certificates for spiffe://cluster.local/ns/bookinfo/sa/..., issued by the shared root CA. Both clusters show the same root serial (2a71866a...). istiod's certificate request is Approved/Ready from issuer istio-ca.	Output A
4	Clusters peered	istioctl multicloud check --contexts="{context1},{context2}"	Both clusters: license valid for multicloud, east-west gateway programmed, peer connected (cluster1 to 10.138.250.104, cluster2 to 10.139.11.137), unique cluster IDs, gateway service accounts match	fig-multicloud-check.png
5	Both clusters in the UI	Solo UI > cluster graph	cluster1 and cluster2 shown with vSphere zones, relay link from cluster2	fig-ui-cluster-graph.png
6	Solo UI relay over the mesh	Relay (cluster2) to management plane (cluster1) via *.mesh.internal	Relay connected; cluster2 objects and metrics visible in the UI after setting IPV6_ENABLED=false on ztunnel. cluster1 exports 3 Solo UI global services	fig-ui-cluster-graph.png
7	Sample app in the mesh on both clusters	Solo UI > namespace and workload views	bookinfo on both clusters, six workloads each	fig-ui-namespace-graph.png, fig-ui-workload-graph.png
8	Global services	bookinfo-multicloud.sh global call: label Bookinfo services solo.io/service-scope=global, call reviews.bookinfo.mesh.internal 20 times from cluster1	cluster1 exports 7 global services (4 Bookinfo, 3 Solo UI) and cluster2 exports 4. With healthy local endpoints, all 20 calls were answered by cluster1.	Output B
9	Cross-cluster failover	bookinfo-multicloud.sh failover: scale reviews to 0 on cluster1, repeat the 20 calls, restore	All 20 calls answered by cluster2, none failed. The UI shows productpage-v1 on cluster1 calling reviews v1-v3 on cluster2, which call ratings locally.	Output B, fig-ui-crosscluster-failover.png
10	Telemetry without application changes	bookinfo-multicloud.sh traffic, then open the global service and workload views	ratings.bookinfo.mesh.internal (global service view): per-cluster sources, request rate, error rate, P99 latency, and traffic by vSphere zone. productpage-v1 (workload panel): outbound calls to details and reviews v1-v3 at about 6 req/s, 15 ms P90	fig-ui-global-service.png, fig-ui-workload-metrics.png

#	Check	Command / action	Result in this validation	Figure
			latency, and 0% errors. The two views report different latency percentiles. No waypoints or application instrumentation were deployed.	

Note on `istioctl multicluster check`: with both contexts, the check reports a warning, "Intermediate Certs Compatibility Check: cacerts secret not found", and exits non-zero. This is expected with istio-csr: the check looks for the `cacerts` secret used by istiod's built-in CA, and this install does not use one. Run against a single cluster, the check passes and shows the shared root certificate fingerprint (Figure 6).

Output A — workload certificates from ztunnel (subject fields trimmed):

```

CERTIFICATE NAME                                TYPE    STATUS    VALID
CERT  SERIAL NUMBER                            NOT AFTER    NOT BEFORE
spiffe://cluster.local/ns/bookinfo/sa/bookinfo-productpage  Leaf    Available true
56732014f1ee7762398750bd23626d1d680c5748  2026-10-06T03:21:07Z  2026-10-06T02:21:07Z
spiffe://cluster.local/ns/bookinfo/sa/bookinfo-productpage  Root    Available true
2a71866a8fa77aa45f45866361c659c94c9214b1  2036-10-02T20:49:03Z  2026-10-05T20:49:03Z
spiffe://cluster.local/ns/bookinfo/sa/bookinfo-reviews      Leaf    Available true
78f5d349304ff6cc663c93e0697403723d8f06b3  2026-10-06T03:21:08Z  2026-10-06T02:21:08Z
spiffe://cluster.local/ns/bookinfo/sa/bookinfo-reviews      Root    Available true
2a71866a8fa77aa45f45866361c659c94c9214b1  2036-10-02T20:49:03Z  2026-10-05T20:49:03Z

issuer=... CN = root-cert
X509v3 Subject Alternative Name: critical
  URI:spiffe://cluster.local/ns/bookinfo/sa/bookinfo-productpage

```

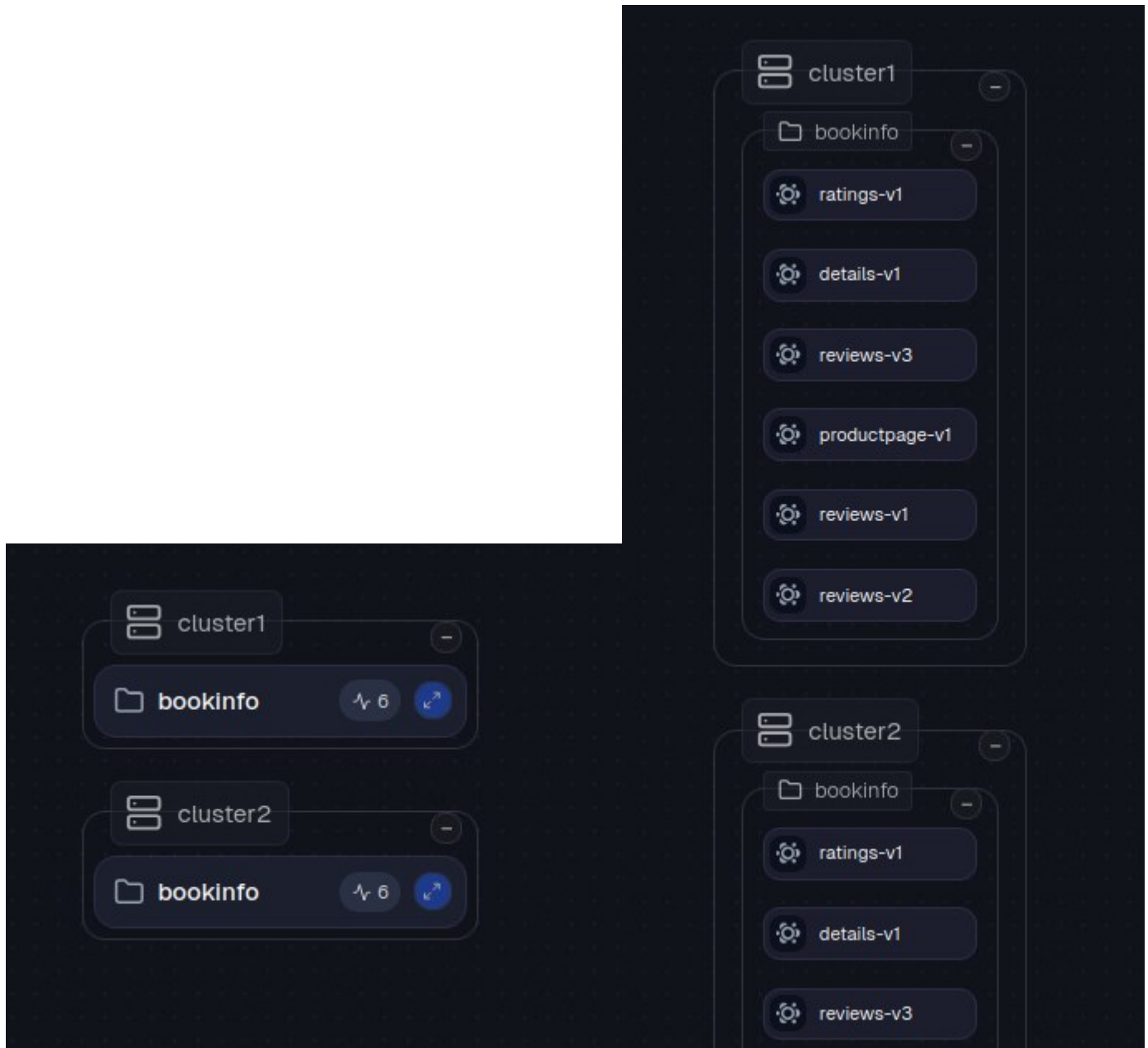


Figure 8: Bookinfo workloads in the ambient mesh on both VKS clusters

Output B — global services, local preference, and failover:

```
=== Cluster: cluster1 ===
☒ Shared Services Check: 7 globally shared service(s)
=== Cluster: cluster2 ===
☒ Shared Services Check: 4 globally shared service(s)

==> Calling reviews.bookinfo.mesh.internal from cluster1 (20 requests)
cluster1: 20
cluster2: 0
failed: 0

==> Failover: scaling reviews to 0 on cluster1
==> Calling reviews.bookinfo.mesh.internal with no local reviews pods (20 requests)
cluster1: 0
```

```
cluster2: 20  
failed: 0
```

With reviews running in both clusters, every call to the global hostname was served by cluster1, so traffic stays in the local cluster while it has healthy endpoints. After the reviews deployments on cluster1 were scaled to zero, the same calls were served by cluster2 through the east-west gateways, with no failed requests and no change to the client.

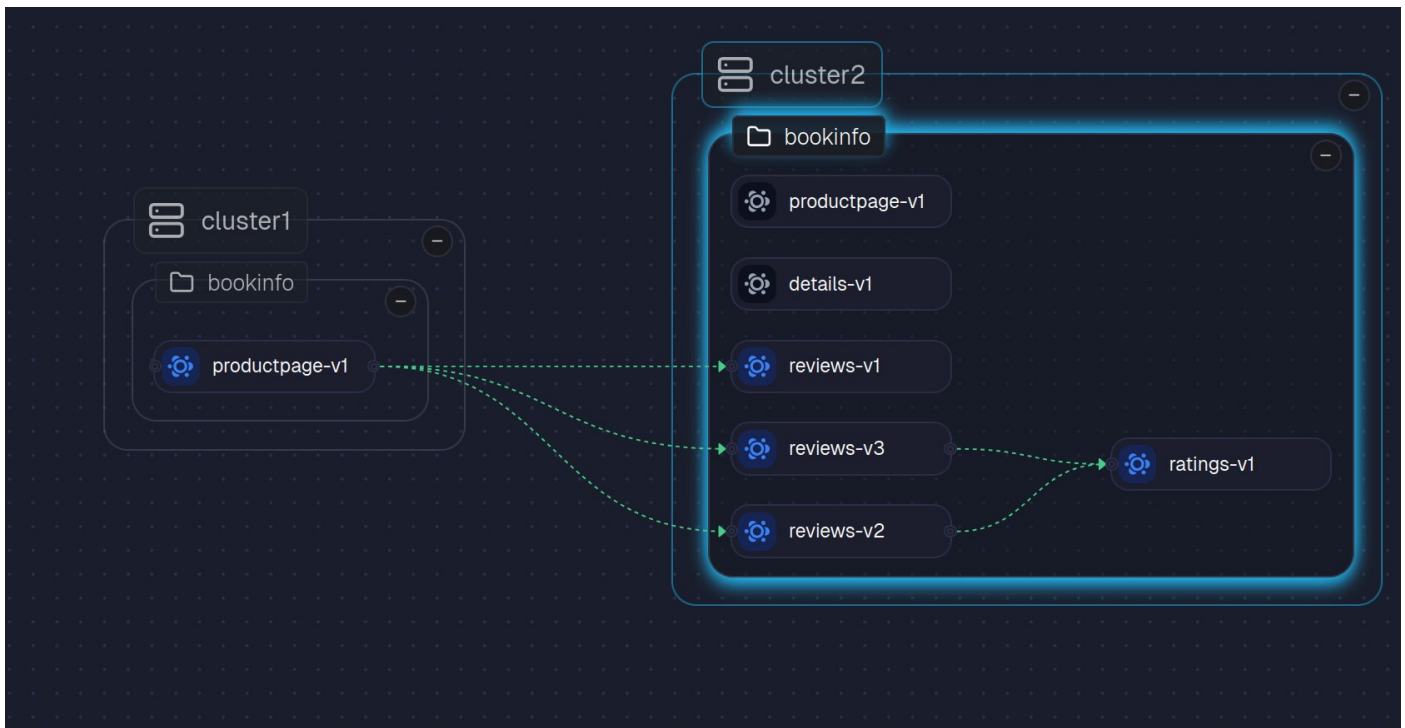


Figure 9: Solo UI during failover — productpage on cluster1 served by reviews on cluster2

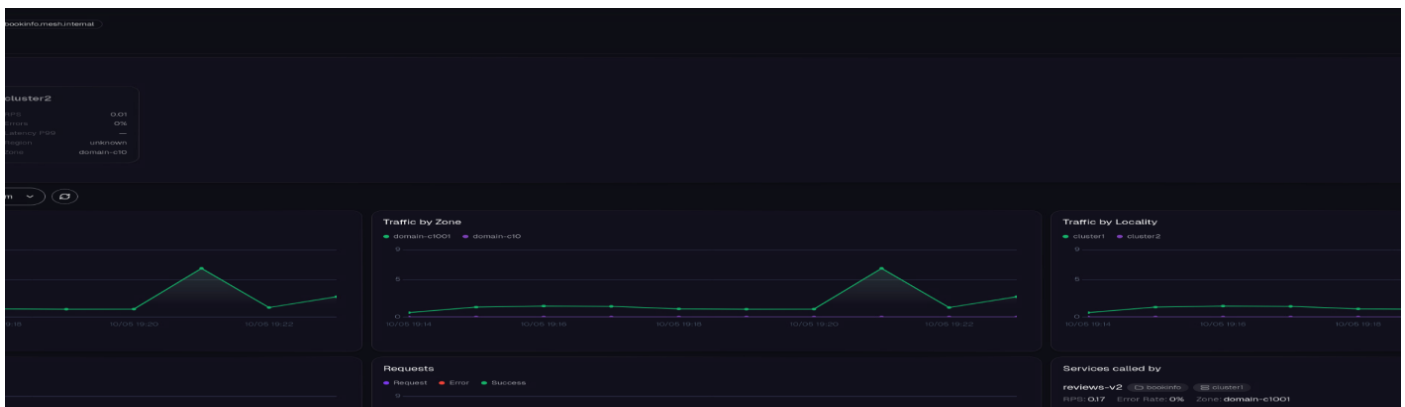


Figure 10: Global service view for ratings.bookinfo.mesh.internal — traffic per cluster and per vSphere zone

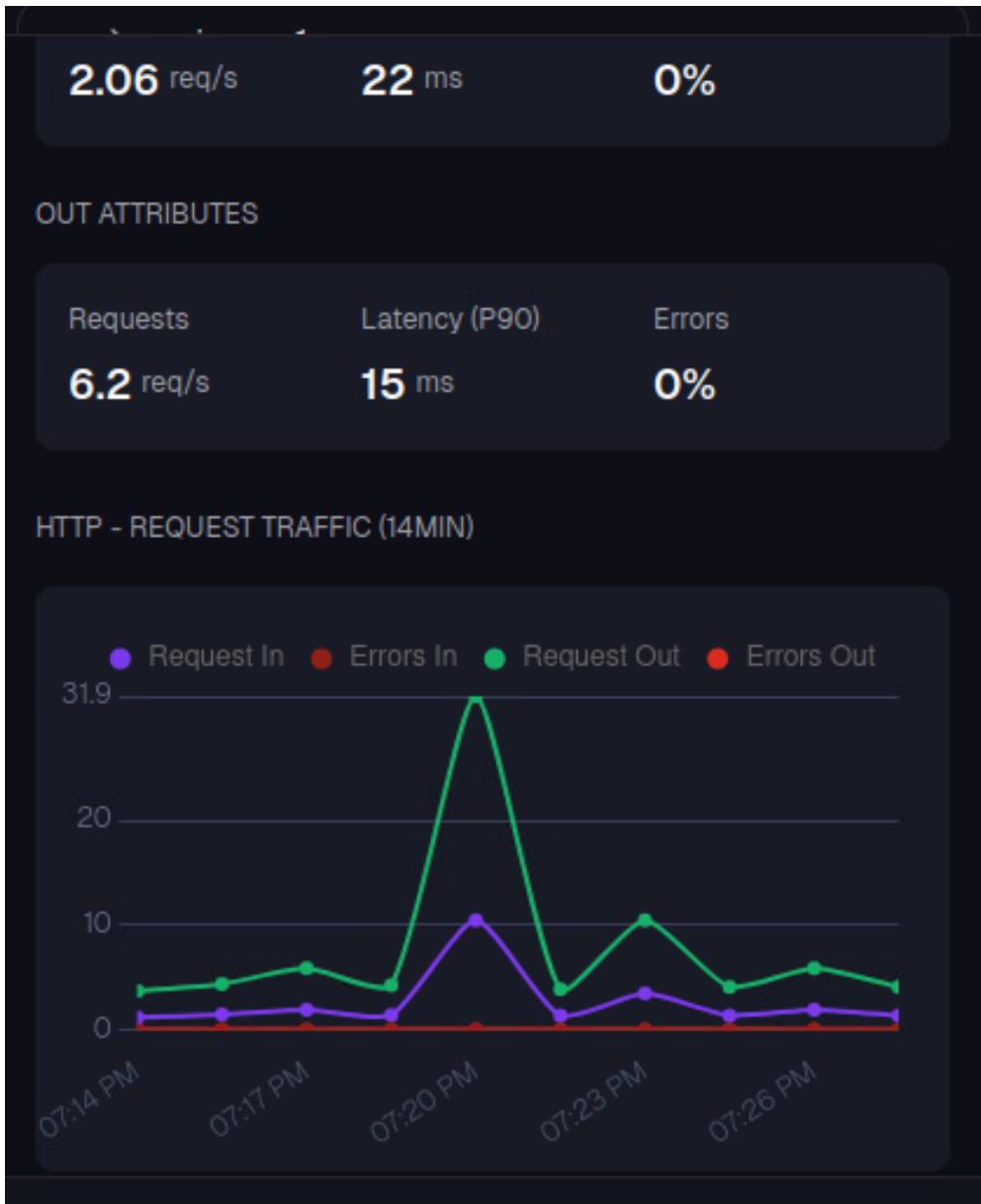


Figure 11: Request rate, latency, and errors for productpage-v1, collected by ztunnel without application changes

Conclusion

This reference architecture shows Solo Enterprise for Istio running in ambient mode on two vSphere Kubernetes Service clusters, installed with the Solo Enterprise for Istio Helm charts. The VKS-specific steps were small: Pod Security Admission labels on the namespaces that run mesh components, disabling IPv6 in ztunnel on the IPv4-only clusters, and a lower ClickHouse CPU request on the small evaluation nodes. The clusters were peered through east-west gateways backed by VKS LoadBalancer services, and mesh certificates were issued by cert-manager through istio-csr from a shared root CA, so the mesh uses the same certificate tooling as the rest of the platform. The Solo UI showed both clusters and their workloads from a single place, with telemetry collected without application changes.

For VKS platform teams, responsibilities split cleanly: VKS and the Supervisor handle cluster provisioning, upgrades, and add-ons such as cert-manager; Solo Enterprise for Istio handles workload identity, encryption, traffic policy, and connectivity between clusters. Teams can start with one cluster in ambient mode and add clusters to the mesh as their VCF footprint grows.



Copyright © 2026 Broadcom. All rights reserved.
The term “Broadcom” refers to Broadcom Inc. and/or its subsidiaries. For more information, go to www.broadcom.com. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies. Broadcom reserves the right to make changes without further notice to any products or data herein to improve reliability, function, or design. Information furnished by Broadcom is believed to be accurate and reliable. However, Broadcom does not assume any liability arising out of the application or use of this information, nor the application or use of any product or circuit described herein, neither does it convey any license under its patent rights nor the rights of others.
Item No: vmw-bc-wp-tech-uslet-word-2026; Aug-26