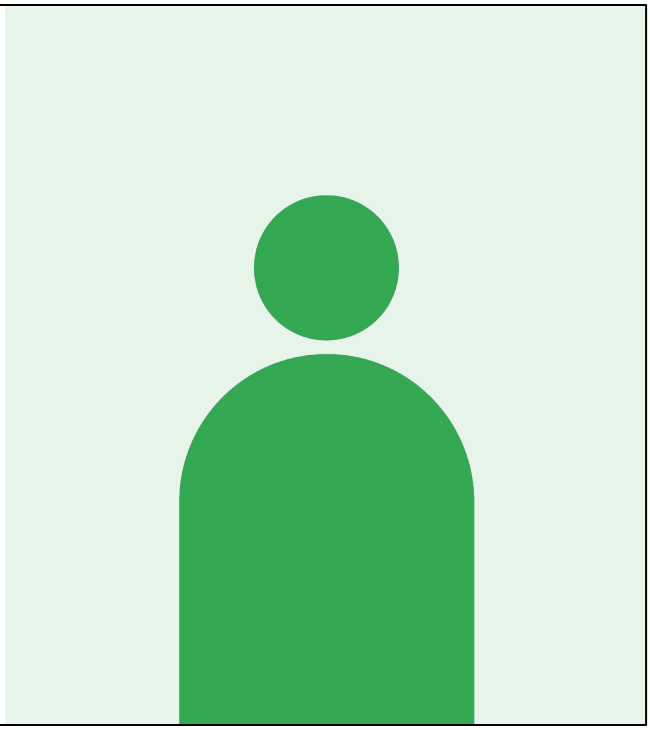
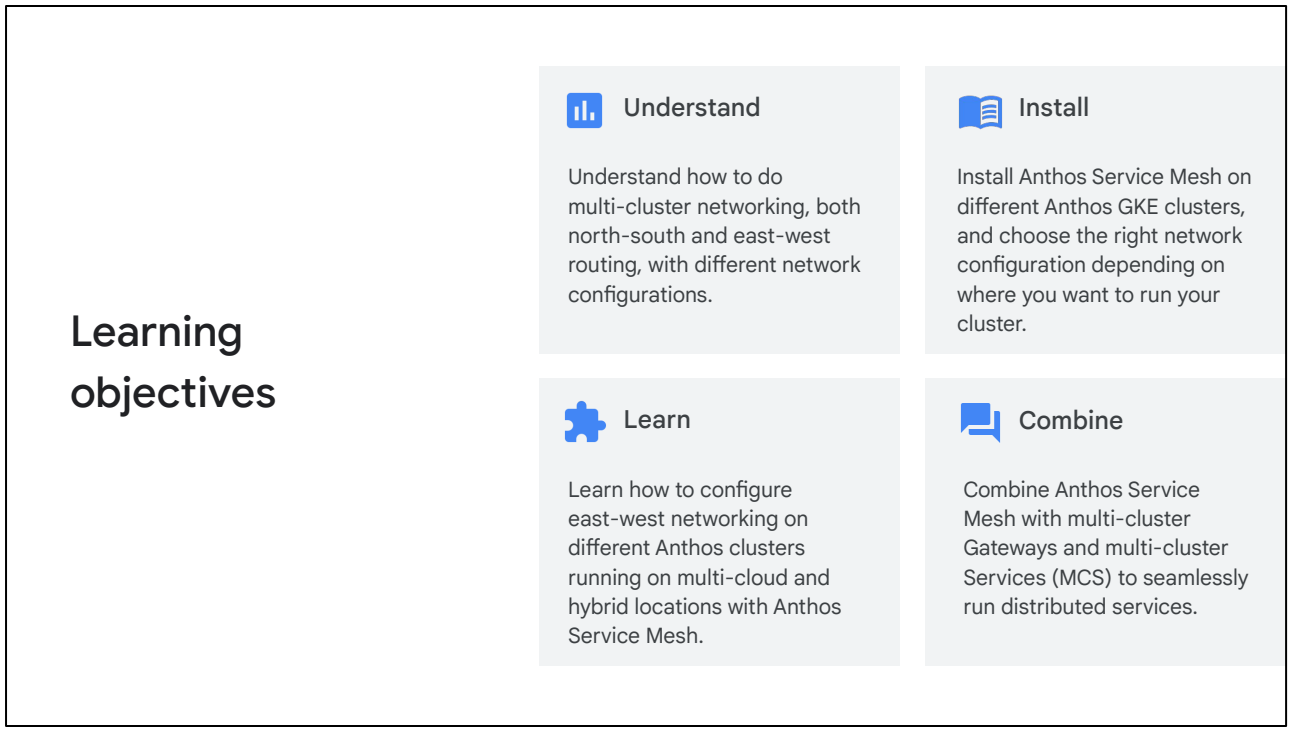
Google Cloud

# Multi-cluster Networking with Anthos Service Mesh

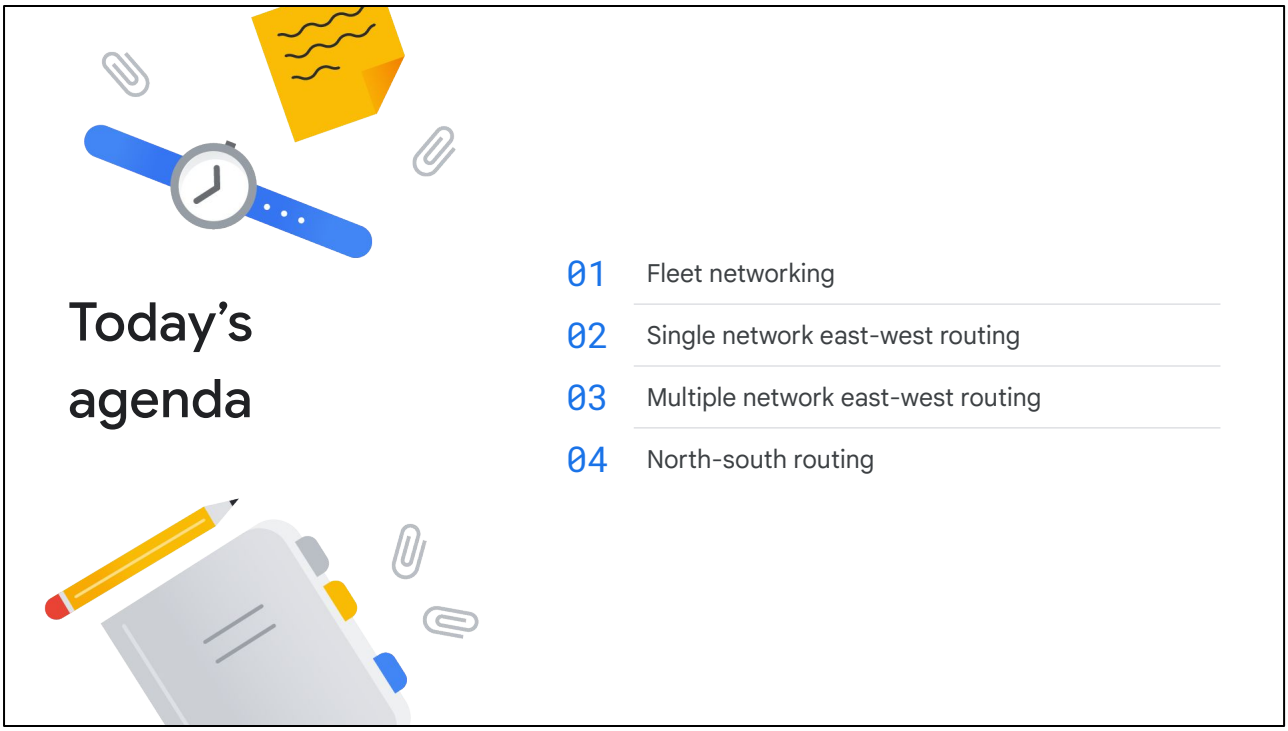


Welcome to Multi-cluster Networking with Anthos Service Mesh.

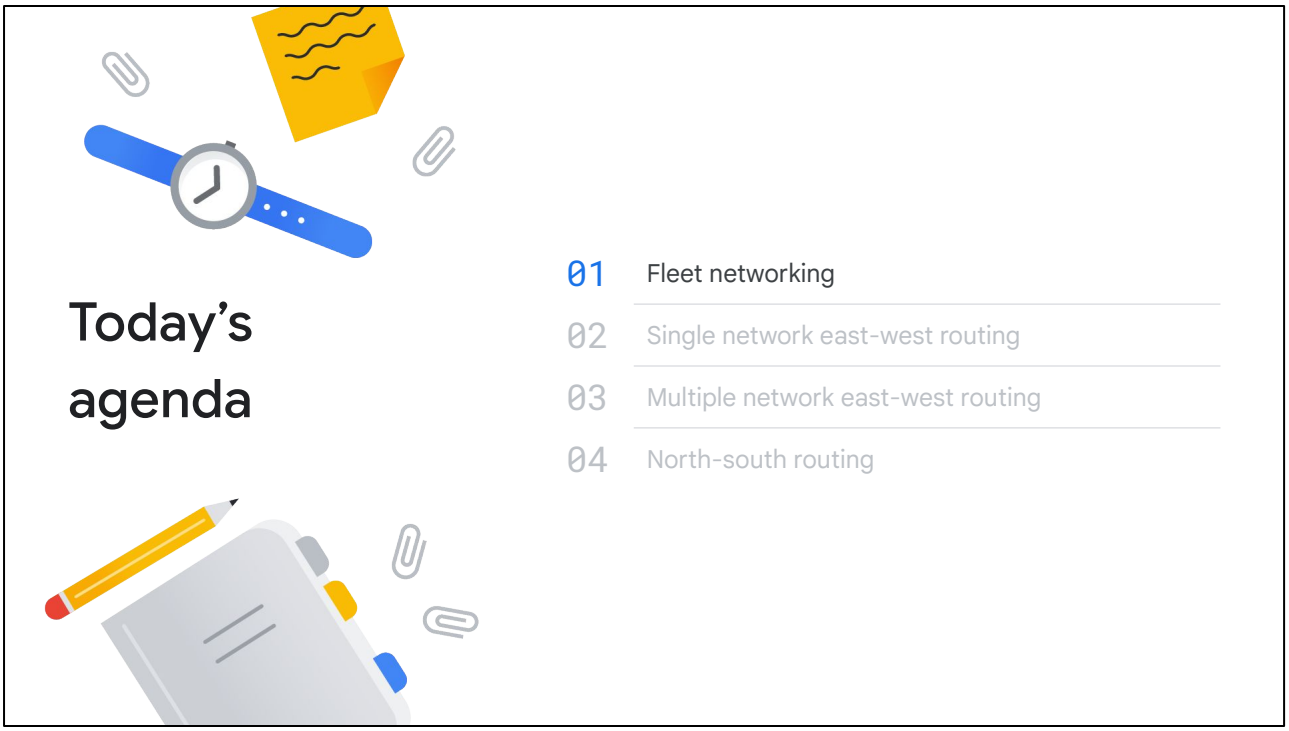



In this module, you:

- Understand how to do multi-cluster networking, both north-south as well as east-west routing with different network configurations.
- Install Anthos Service Mesh on different Anthos GKE clusters and choose the right network configuration depending on where you want to run your cluster.
- Learn how to configure east-west networking on different Anthos clusters running on multi-cloud and hybrid locations with Anthos Service Mesh.
- Combine Anthos Service Mesh with Multi-Cluster Gateways and Multi-Cluster Services to seamlessly run distributed services.




This is our agenda for the module, shown here on the slide. Let's get started.



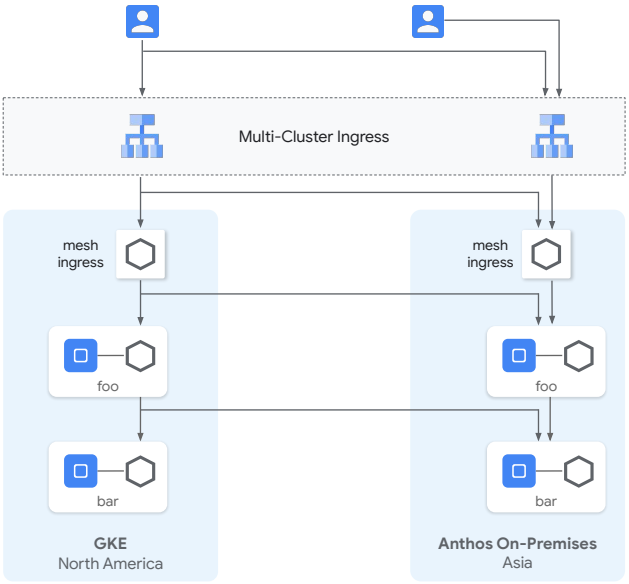

Let's start reviewing concepts on fleet networking.

## Fleet networking

There are two ways of communicating between Anthos clusters:

- **North-south routing:** communication from a load balancer into our clusters
- **East-west routing:** communication between clusters

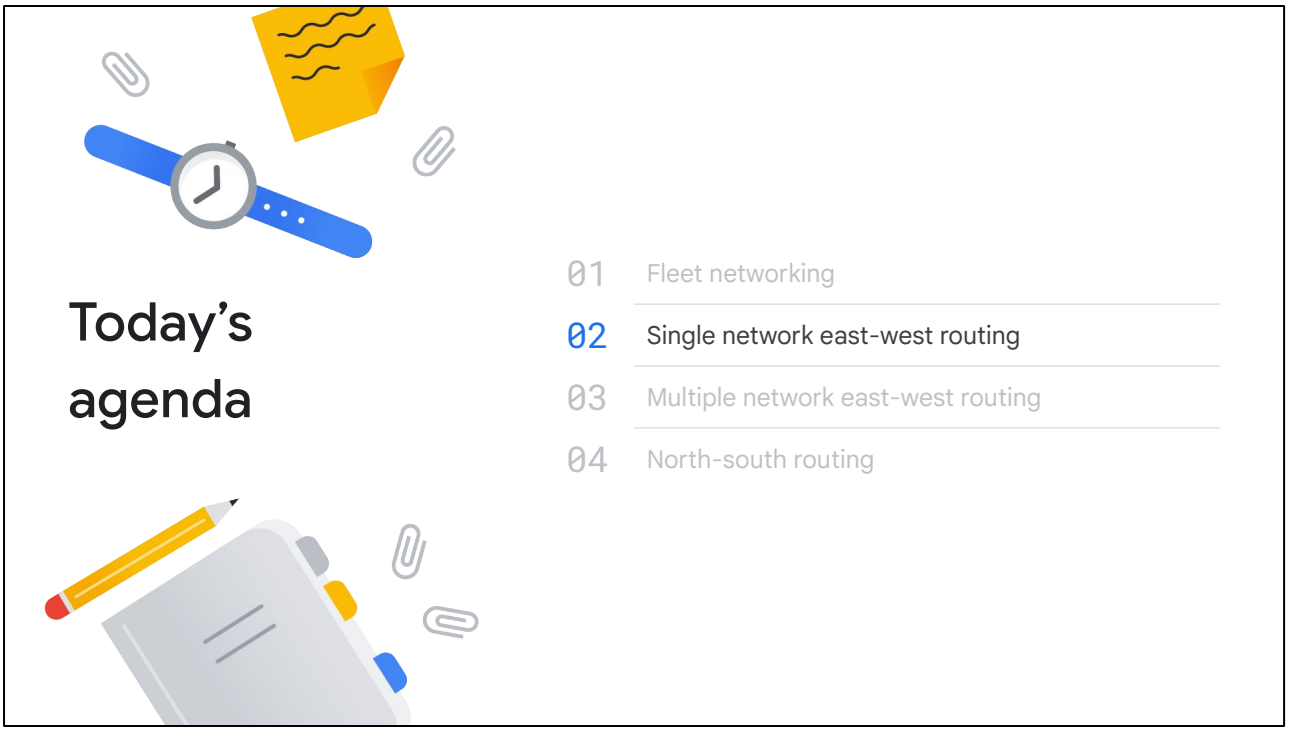
This module focuses primarily on east-west routing routing. We discuss north-south in depth in the Multi-Cloud, Multi-Cluster with Anthos training day.




There are two ways of communicating between Anthos clusters, and in most cases, you will be using both approaches.

- North-South routing handles the communication from a load balancer into our clusters. That load balancer can be receiving traffic from the internet or from some internal applications.
- East-West routing handles the communication between clusters. This is important for use cases, such as having dependencies in other clusters, implementing fall-back strategies, or performing blue-green deployments.

In this module, we focus primarily on East-West routing routing. North-South is covered in depth in the Multi-Cloud, Multi-Cluster with Anthos course.




Let's start by discussing East-West routing

## Multi-cluster mesh networking

- Multi-cluster on GKE (one project, one network, on subnet)
  - Variant 1: Multiple subnets
  - Variant 2: Multiple VPCs peered
  - Variant 3: Multiple projects with a shared VPC
  - Variant 4: Private clusters

In multi-cluster mesh networking there are different ways to configure Anthos Service Mesh. The main differences are dependent on the network and the type of cluster. First, we discuss a single network with clusters running on Google Cloud with Google Kubernetes Engine.

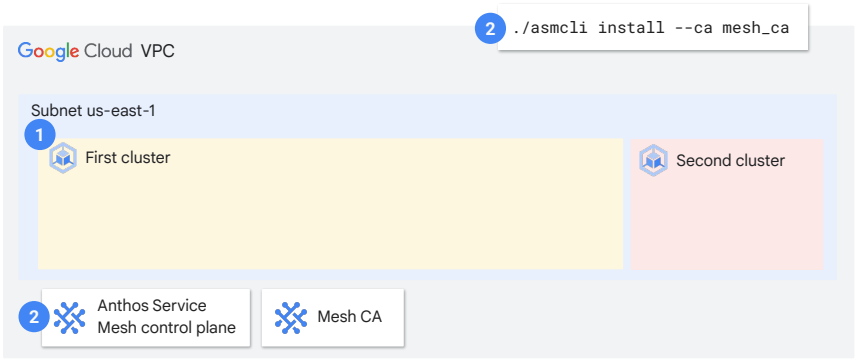
# Set up a multi-cluster mesh on GKE using a single subnet in a single VPC network


Let’s set up a multi-cluster mesh on GKE using a single subnet in a single VPC network.

# 1. Create and register your GKE clusters to Anthos Connect Hub

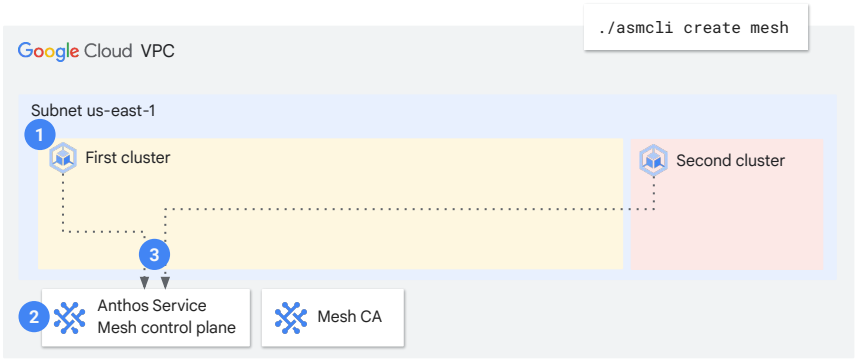

First, create and register your GKE clusters to Anthos Connect Hub.

## 2. Install Anthos Service Mesh with Mesh CA on both clusters



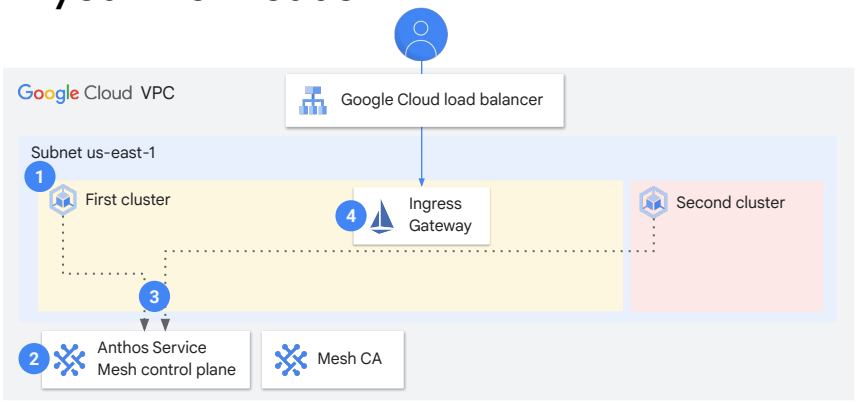

Install Anthos Service Mesh with Mesh CA on both clusters.

# 3. Discover endpoints on both clusters



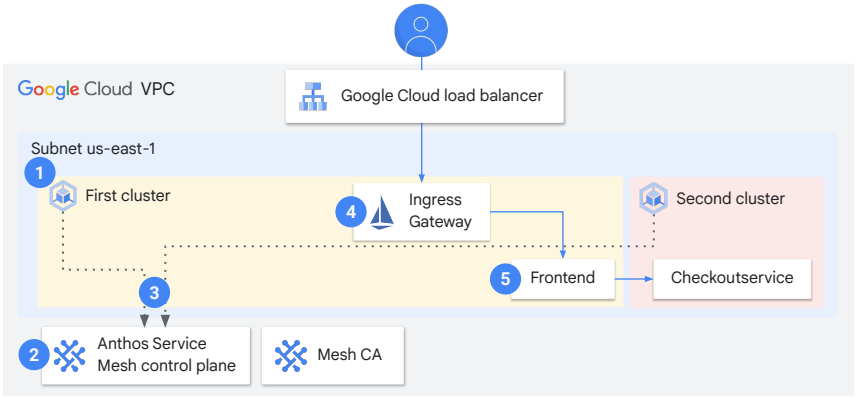

In order for each cluster to route requests to services in the other cluster, the clusters need to have a list of endpoints from the other cluster. To discover the endpoints, clusters must be registered to the same fleet, and must be able to access the other cluster's API server for endpoint enumeration. The “asmcli create mesh” command registers your clusters and creates a secret with the other cluster’s API server.

# 4. [Optional] Create an ingress gateway to access your workloads



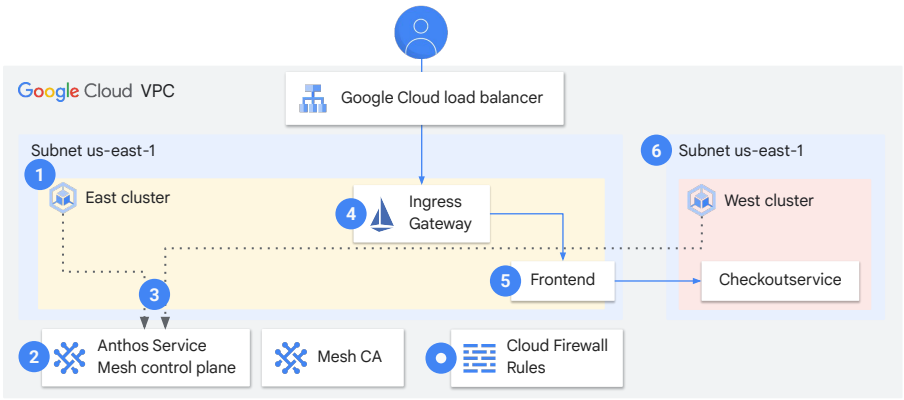

If you want to access your services from outside the mesh, you might want to create an ingress gateway. Ingress routing into the cluster is called north-south routing. In this diagram, the ingress gateway only appears in one cluster, but you could also have one ingress gateway per cluster. Then, you could use Multi-cluster Gateways and Multi-cluster Services to balance requests from Google's load balancer into one of the gateways. We discuss this option later in the module.

# 5. Deploy your application across clusters and you are done!



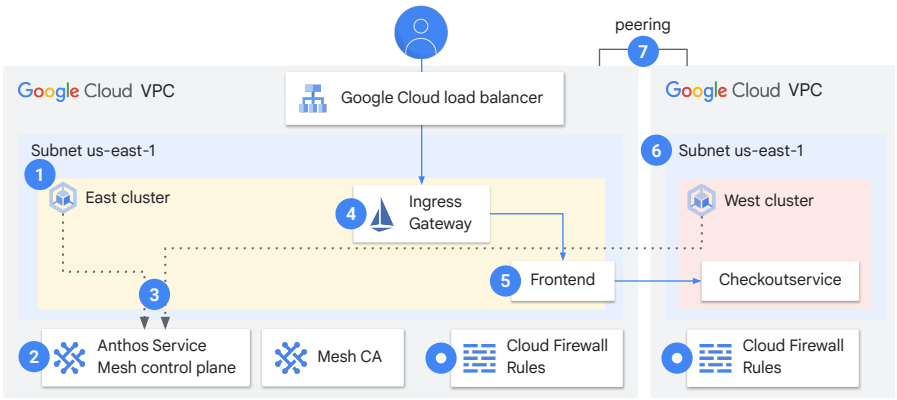

That's it - label namespaces for automatic sidecar injection and deploy your application across clusters. Services in one cluster will automatically be able to communicate with services deployed on the other cluster.

## [Variant 1] Cross-subnet traffic: additional firewall rules



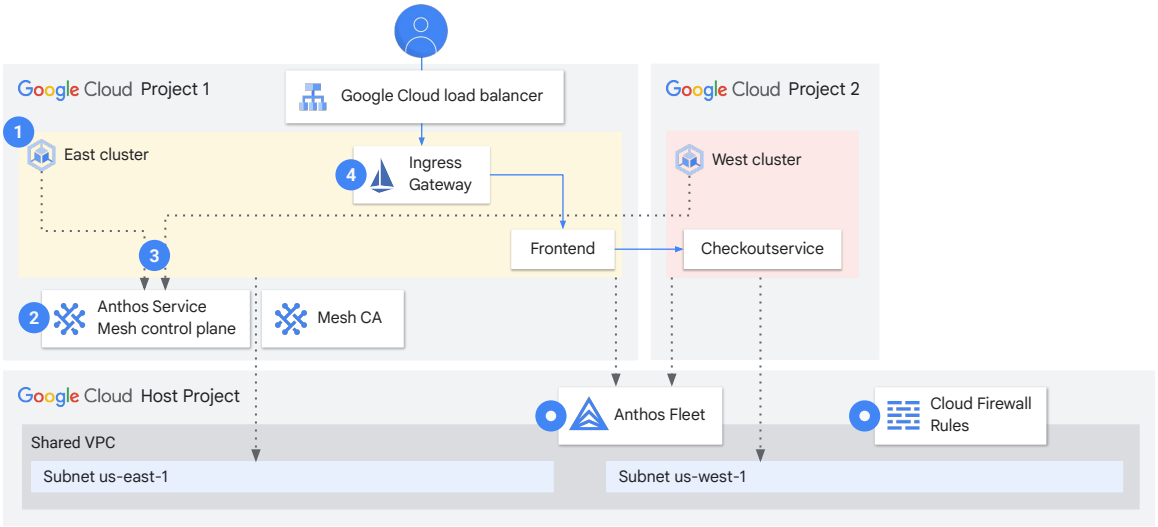

The second variant uses multiple projects to host your infrastructure. This is a common approach for large teams that want to have better controls on permissions and cost. To keep the communication secure, you should set up peering between both networks, which allows communication using private IPs. This configuration is the same as the first variant, since peered networks act as different subnets of the same network. Therefore, you would need to create firewall rules on both projects allowing communication between all CIDR blocks in both clusters.

## [Variant 2] Peered VPC network: additional firewall rules



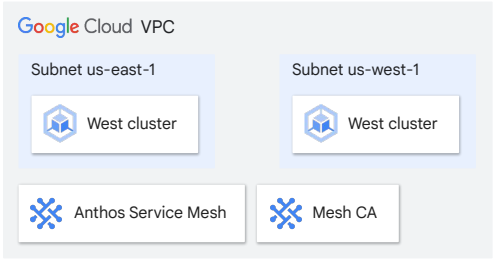

Let's look at some variants. In the first variant, we move from clusters in the same region to clusters in multiple regions, which improves availability and reduces latencies for end customers. When clusters were in the same region, they were also in the same subnet and traffic could flow between the nodes because GKE automatically adds firewall rules on creation. However, when you put GKE clusters in different regions, and therefore different subnets, you must explicitly set up the firewall rules to allow cross-subnet traffic. The firewall rules should include all CIDR ranges in the cluster, including nodes, services, and pods.

# [Variant 3] Shared VPC traffic: additional firewall rules



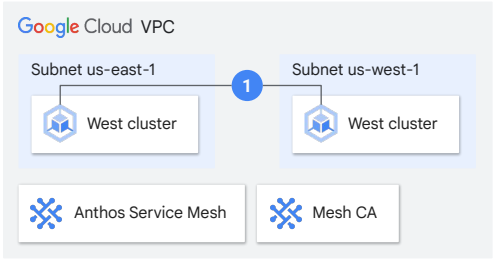

The third variant expands on the controls that we introduced in variant 2. In addition to separating the infrastructure into different projects, you centralize all network configuration on a Host project, and create a Shared VPC so that service teams in other projects can launch their infrastructure using that common network. In this configuration, a centralized networking or security team must create firewall rules on the host project that enable communication between all CIDR ranges in both clusters. Also, make sure that all clusters are registered to the same Anthos Fleet in the host project.

# [Variant 4] Private clusters in different subnets




Variant 4 introduces private clusters, a security best practice that limits the Kubernetes API server accessibility to services on the same VPC network. However, this brings some challenges because clusters are no longer able to communicate with each other. To make that communication possible, we need to perform three actions.

## [Variant 4] Private clusters in different subnets



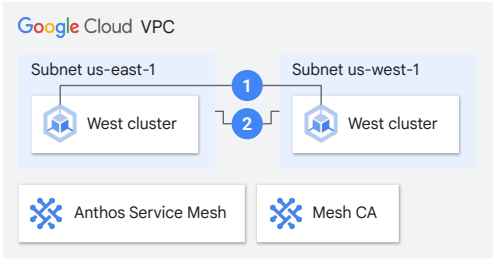

1 Configure endpoint discovery:

```
istioctl x create-remote-secret \
--context=$C1 --name=$C1 --server=$IP1 \
> $C1.secret
```

```
kubectl apply -f $C1.secret
--context=$C2
```

First, configure endpoint discovery. Previously, we used the “asmcli create mesh” command, which created a secret in each other’s cluster with the public Kubernetes API server IP. Now, we must get the private Kubernetes API server IP from each cluster and create a secret pointing to that IP in the other cluster. That way, clusters are able to read the pod endpoints registered in the remote cluster.

## [Variant 4] Private clusters in different subnets




1 Configure endpoint discovery:

```
istioctl x create-remote-secret \
--context=$C1 --name=$C1 --server=$IP1 \
> $C1.secret
```

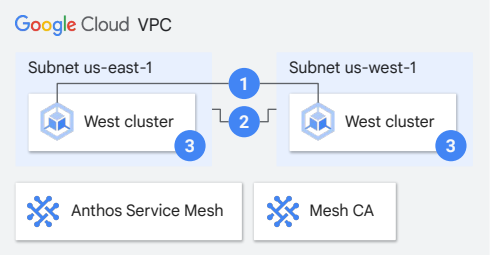
```
kubectl apply -f $C1.secret
--context=$C2
```

2 Configure authorized networks:

```
gcloud container clusters update $C1 \
--enable-master-authorized-networks \
--master-authorized-networks $C1P,$C2P
```

Second, the Anthos Service Mesh control plane in each cluster needs to call the GKE control plane of the remote clusters. To allow traffic, you need to add the pod address range in the calling cluster to the authorized networks of the remote clusters.

## [Variant 4] Private clusters in different subnets




1 Configure endpoint discovery:

```
istioctl x create-remote-secret \
--context=$C1 --name=$C1 --server=$IP1 \
> $C1.secret
```

```
kubectl apply -f $C1.secret
--context=$C2
```

2 Configure authorized networks:

```
gcloud container clusters update $C1 \
--enable-master-authorized-networks \
--master-authorized-networks $C1P,$C2P
```

3 Enable control plane global access:

```
gcloud container clusters update $C1 \
 --enable-master-global-access
```

Third, enable control plane global access to allow the Anthos Service Mesh control plane in each cluster to call the GKE control plane of the remote clusters.

# Today's agenda

01 Fleet networking

02 Single network east-west routing

03 Multiple network east-west routing

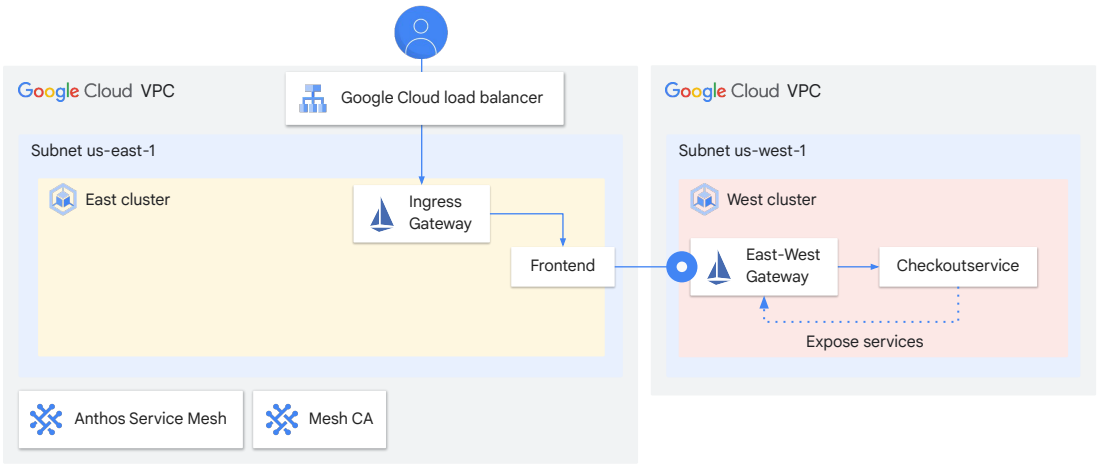
04 North-south routing

## Multi-cluster mesh networking

- Multi-cluster on GKE (one project, one network, on subnet)
  - Variant 1: Multiple subnets
  - Variant 2: Multiple VPCs peered
  - Variant 3: Multiple projects with a shared VPC
  - Variant 4: Private clusters
- Multi-cluster on GKE, different networks
  - Variant 1: Hybrid and Multi-Cloud with Anthos on VMware, bare metal, AWS
  - Variant 2: Anthos attached clusters

We covered how to configure multiple GKE clusters in the same network. Next, let's cover the additional steps needed to make the east-west communication work on different networks without a VPN connection, which might take place when we deploy clusters in multi-cloud or hybrid environments.

# Multi-cluster on GKE, different networks

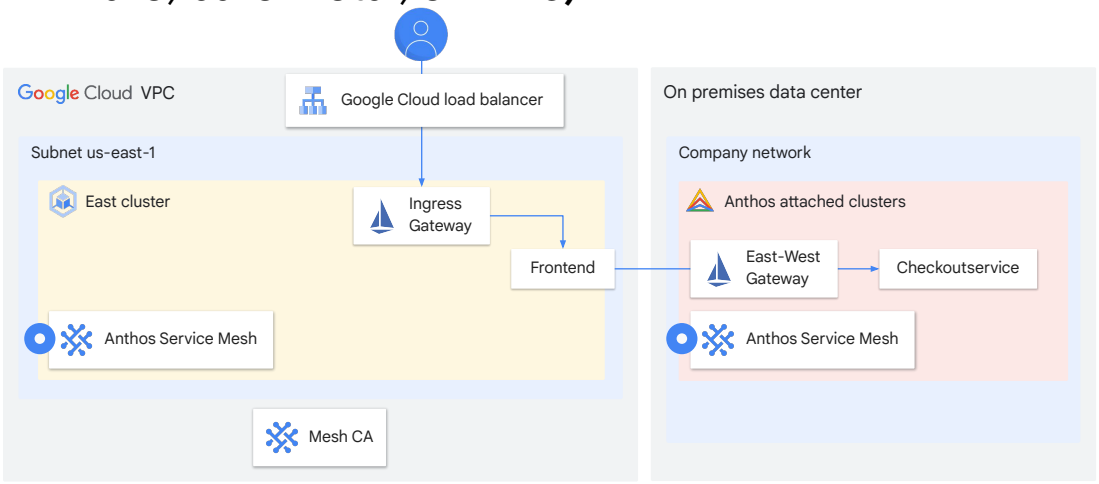



First, let's take a look at this diagram. We have two clusters in two different projects, and therefore, two VPC networks which are not peered. That means, if you want to communicate between clusters, you will have to create public IPs.

Do you see any new component in the diagram that we did not have before?

The East-West Gateway is the new component in the diagram. This gateway proxies all communication to services on another cluster and makes the communication transparent, so that you don't need to worry where your services are located. To make this possible, you must expose all services on the East-West Gateway. While this gateway is public on the internet, services behind it can only be accessed by services with a trusted mTLS certificate and workload ID, just as if they were on the same network. However, production systems may require additional access restrictions, for example, via firewall rules, to prevent external attacks.

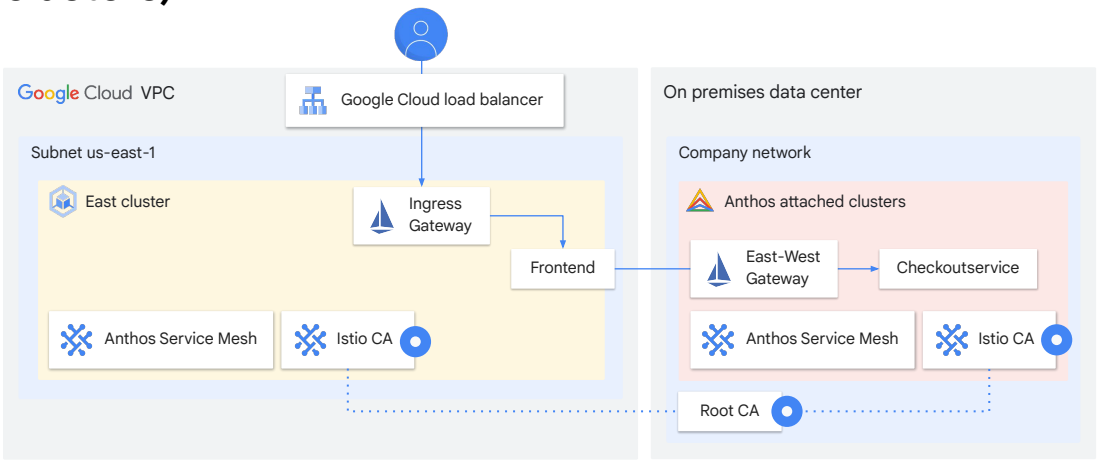
## Variant 1: Hybrid and Multi-Cloud (Anthos clusters on VMware, bare metal, or AWS)




The first variant of the multi-cluster mesh on separate networks shows the way to deploy a mesh across clusters located both inside and outside of Google Cloud. In this example, we see an Anthos GKE cluster on Google Cloud and an Anthos cluster on VMware. The installation commands vary but the steps are the same. Refer to the documentation for the specific commands.

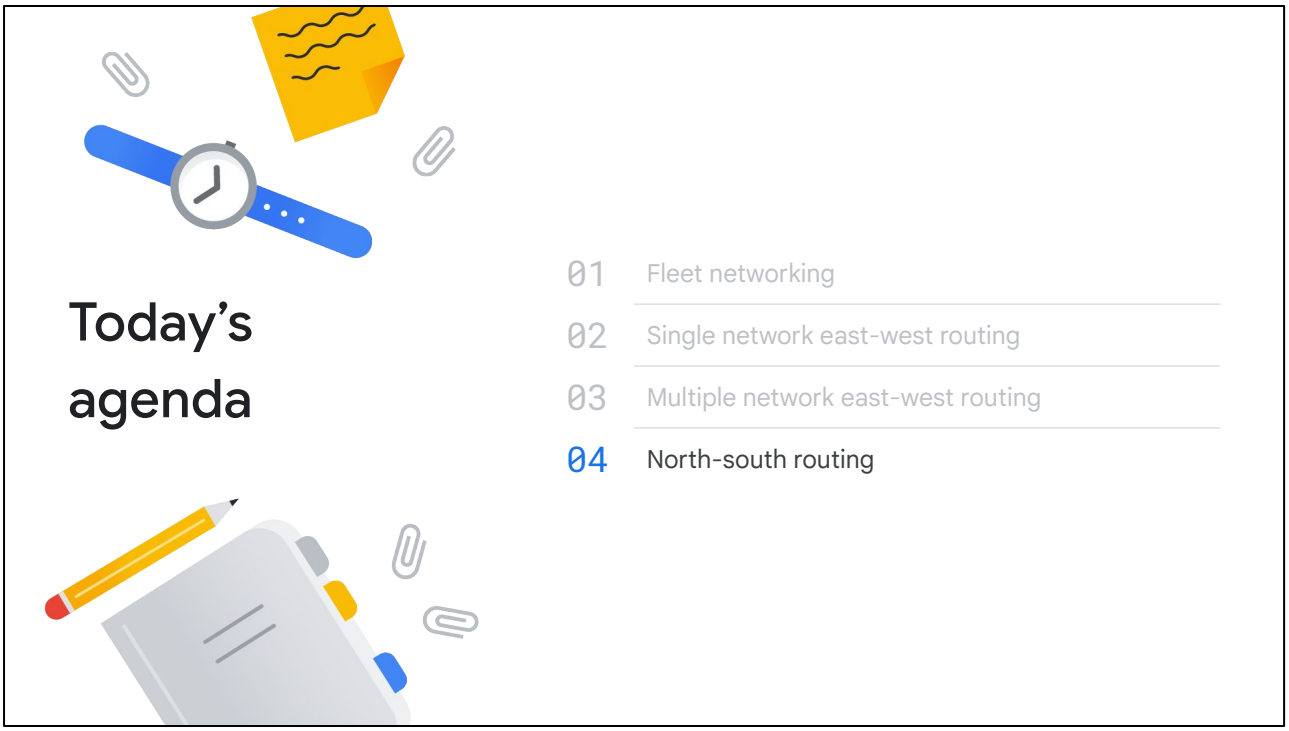
The major difference lies in the control plane, as we can no longer use the Google managed control plane and must default to the in-cluster control plane with istiod. Also, Mesh CA can be used but it does not work with CA Service as the root CA.

## Variant 2: Hybrid and Multi-Cloud (Anthos attached clusters)



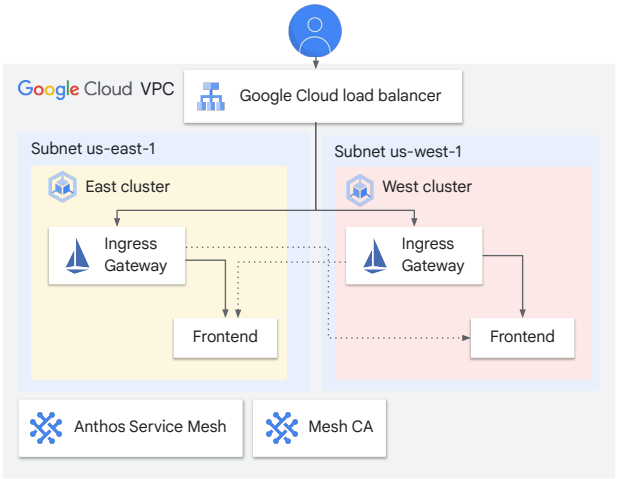

The second variant shows a diagram of a mesh spanning across an Anthos GKE cluster and an Attached Anthos cluster running on a customer's location. Remember that attached clusters are Anthos compatible clusters that are managed by you and have been registered to Anthos Connect Hub. Examples of these clusters include Amazon EKS, Microsoft AKS, Red Hat Openshift, both OKE and OCP, KIND, K3s, and K3d.

The support for these clusters varies depending on the attached cluster you are connecting and you might need to downgrade the version of Anthos Service Mesh. The biggest change with respect to the first variant is the usage of an in-cluster Istio CA instead of Mesh CA. Since you will have multiple Certificate Authorities, you need to create a Root CA that will establish trust between the two clusters.



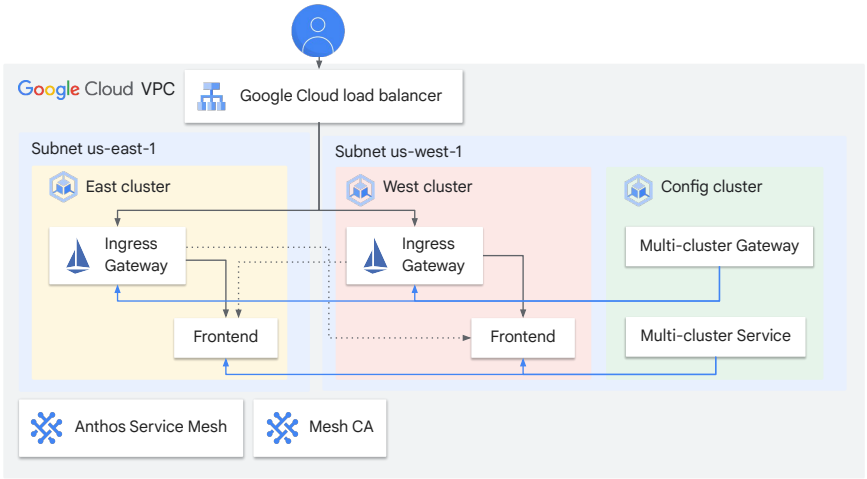

Finally, let's cover north-south routing. Remember this means distributing the traffic from the load balancer into the clusters. We only cover the basics here, if you want a deep dive, refer to the Fleet Networking module in the Multi-Cluster, Multi-Cloud with Anthos course.

## How can you distribute traffic from the load balancer into multiple clusters?



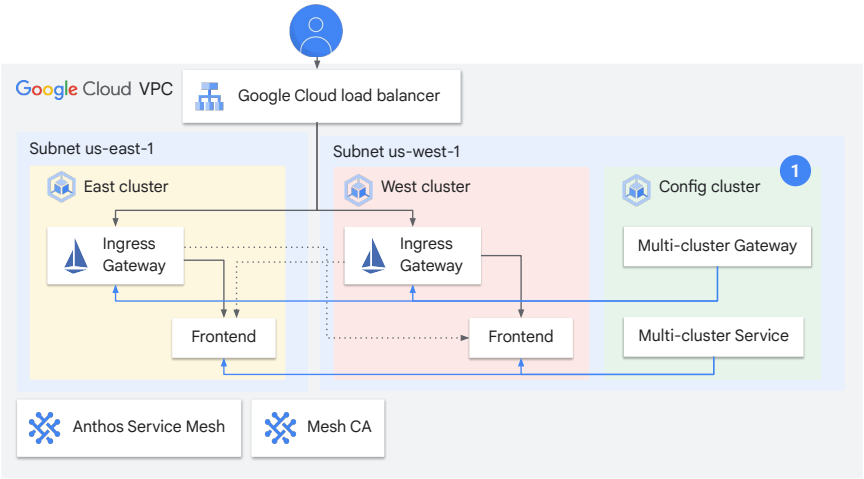

How can we distribute traffic from the load balancer into multiple clusters?

# Use multi-cluster Gateway and multi-cluster Service (MCS) for multi-cluster load balancing



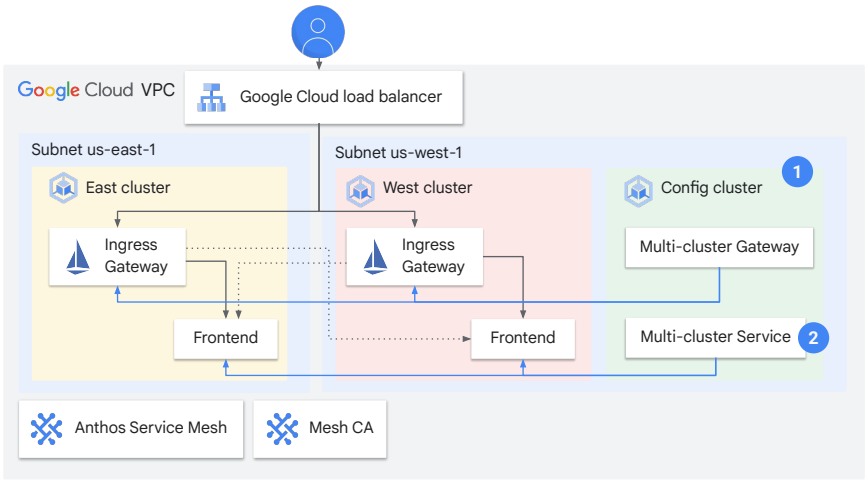

You can use Multi-Cluster Gateway and Multi-Cluster Service (MCS) for multi-cluster load balancing. Let's see how it works.

# 1. Create and register a configuration cluster




First, create a new GKE cluster and register it into Anthos Hub. This cluster is used to host all cross-cluster configurations.

## 2. Create a multi-cluster Service and link your clusters




2. Create a multi-cluster Service in the configuration cluster and link the clusters you want to distribute the load to.

## 2. Multi-cluster Service (MCS) creates Kubernetes Services in linked clusters

```
kind: MultiClusterService
apiVersion: networking.gke.io/v1aplpha1
metadata:
  name: foo
spec:
  template:
    spec:
      selector:
        app: ingressgateway
      ports:
      - name: frontend
        port: 80
clusters:
  - link: “us-east-1/gke-east”
  - link: “us-west-1/gke-west”
```

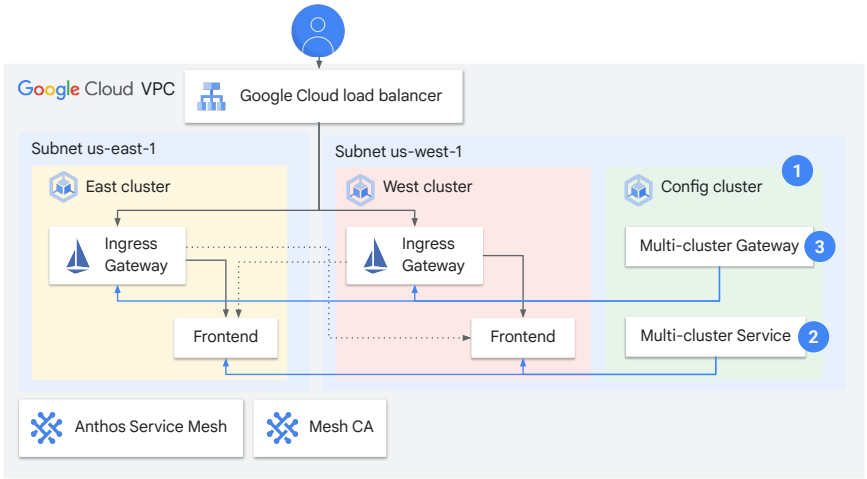
```
kind: Service
metadata:
  annotations:
    cloud.google.com/neg: ‘{{..}}’
    cloud.google.com/neg-status: ‘{{..}}’
    networking.gke.io/mc-parent: ‘{{..}}’
  name: frontend
spec:
...
```

A MultiClusterService, or MCS, is a custom resource that is a logical representation of a Service across multiple clusters. An MCS is similar to, but substantially different from, the core Service type. An MCS exists only in the config cluster and generates derived Services in the target clusters. An MCS does not route anything like a ClusterIP, LoadBalancer, or NodePort Service does.

Like a Service, an MCS is a selector for pods but it is also capable of selecting labels *and* clusters. The pool of clusters that it selects across are called member clusters, and these are all the clusters registered to the fleet. This MCS deploys a derived Service in all member clusters with the selector app: foo. If app: foo pods exist in that cluster, then those pod IPs will be added as backends for the MCI.

The following service is a derived Service that the MCS generated in one of the target clusters. This Service creates a Network Endpoint Group, or NEG, which tracks pod endpoints for all pods that match the specified label selector in this cluster. A derived Service and NEG will exist in every target cluster for every MCS (unless using cluster selectors). If no matching pods exist in a target cluster, then the Service and NEG will be empty. The derived Services are managed fully by the MCS and are not managed by users directly.

## 3. Create a multi-cluster Gateway to route traffic from the load balancer to your distributed services




3. Create a multi-cluster Gateway to route traffic from the load balancer to your distributed services

## 3. Create a multi-cluster Gateway to route traffic from the load balancer to your distributed services

- Choose the Gateway controller with the Gateway Class Name:
  - Internal: gke-l7-rilb-mc
  - External: gke-l7-gxlb-mc
- Bind the Gateway and the HTTPRoute using a selector.
- Reference the TLS credentials to accept HTTP(S) requests from your clients.

```yaml
kind: Gateway
apiVersion: networking.x-k8s.io/v1alpha1
metadata:
  name: ingressgateway
spec:
  gatewayClassName: gke-l7-gxlb-mc
  listeners:
  - protocol: HTTPS
    port: 443
    routes:
      kind: HTTPRoute
      selector:
        matchLabels:
          gateway: ingressgateway
    tls:
      mode: Terminate
      options:
        .../pre-shared-certs: cert
```

To configure Google Cloud load balancer, first you need to use the Gateway CRD.

In the metadata, specify the name of the resource and the namespace where you want to launch it.

In the spec, use the gatewayClassName to specify which controller you want to use. To deploy an internal load balancer, use RILB, while to deploy an external, use GXLB instead.

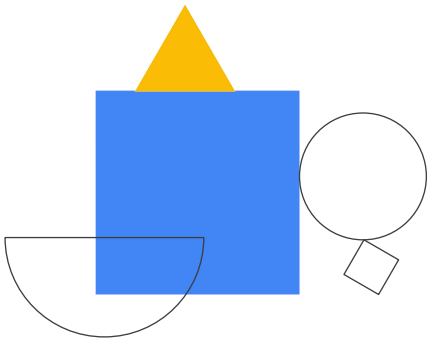
Routes can be created inside the Gateway or in a separate resource. In this case, we specify the HTTPRoute inside the Gateway and then link it with the Gateway using a selector.

Notice that we are specifying the file for the credentials that will be used to establish TLS encryption on the requests.

# Lab intro

30 min

Configuring a multi-cluster mesh



In this lab exercise, you build a service mesh encompassing two clusters, west and east. You deploy an application comprised of services, some running on west and some on east. You test the application to make sure that services can communicate across clusters without problem.

In this lab, you will perform the following tasks:

- Prepare to install Anthos Service Mesh, verifying that the clusters have been created and registered, setting up some environment variables, installing the amscli command line utility, etc.
- Install Anthos Service Mesh on both clusters.
- Configure the mesh to span both clusters.
- Review Service Mesh control planes.
- Deploy the Online Boutique application across multiple clusters.
- Evaluate your multi-cluster application.
- Distribute one service across both clusters.