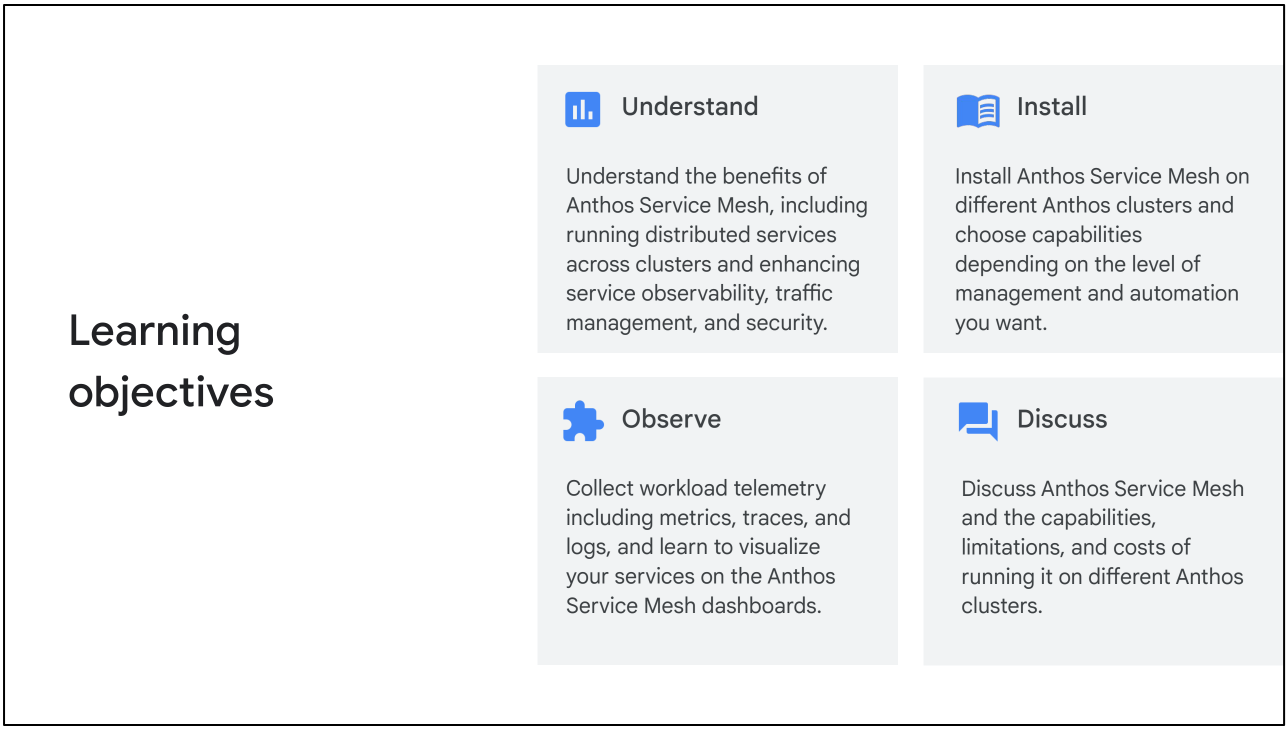
Google Cloud

Introducing Anthos Service Mesh

Welcome to Introducing Anthos Service Mesh.



Learning objectives

Understand

Understand the benefits of Anthos Service Mesh, including running distributed services across clusters and enhancing service observability, traffic management, and security.

Install

Install Anthos Service Mesh on different Anthos clusters and choose capabilities depending on the level of management and automation you want.

Observe

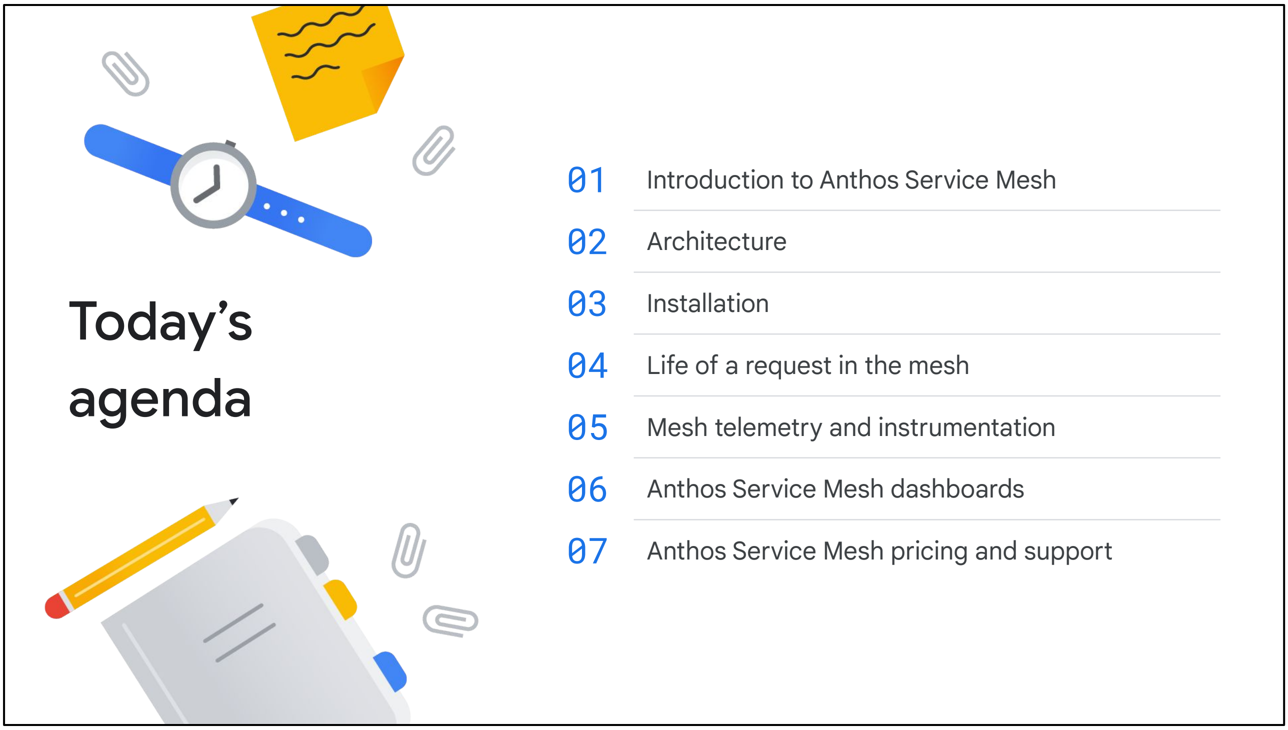
Collect workload telemetry including metrics, traces, and logs, and learn to visualize your services on the Anthos Service Mesh dashboards.

Discuss

Discuss Anthos Service Mesh and the capabilities, limitations, and costs of running it on different Anthos clusters.

In this module, we:

- Understand the benefits of Anthos Service mesh, including running distributed services across clusters and enhancing service observability, traffic management, and security.
- Install Anthos Service Mesh on different Anthos clusters and choose the right capabilities for you depending on the level of management and automation you want.
- Collect workload telemetry including metrics, traces and logs, and learn to visualize your services on the Anthos Service Mesh dashboards.
- Operate an Anthos Service Mesh understanding the capabilities, limitations, and costs of running it on different Anthos clusters.



Today's agenda

01 Introduction to Anthos Service Mesh

02 Architecture

03 Installation

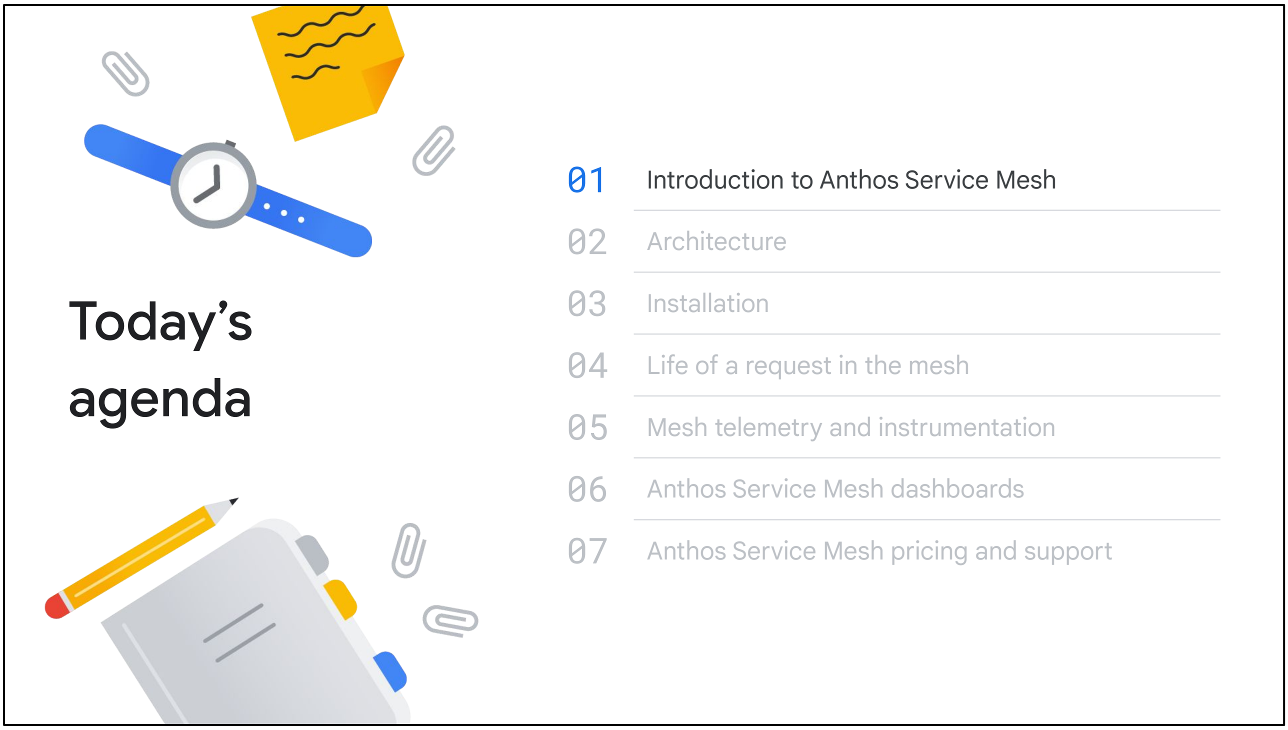
04 Life of a request in the mesh

05 Mesh telemetry and instrumentation

06 Anthos Service Mesh dashboards

07 Anthos Service Mesh pricing and support

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



Today's agenda

01 Introduction to Anthos Service Mesh

02 Architecture

03 Installation

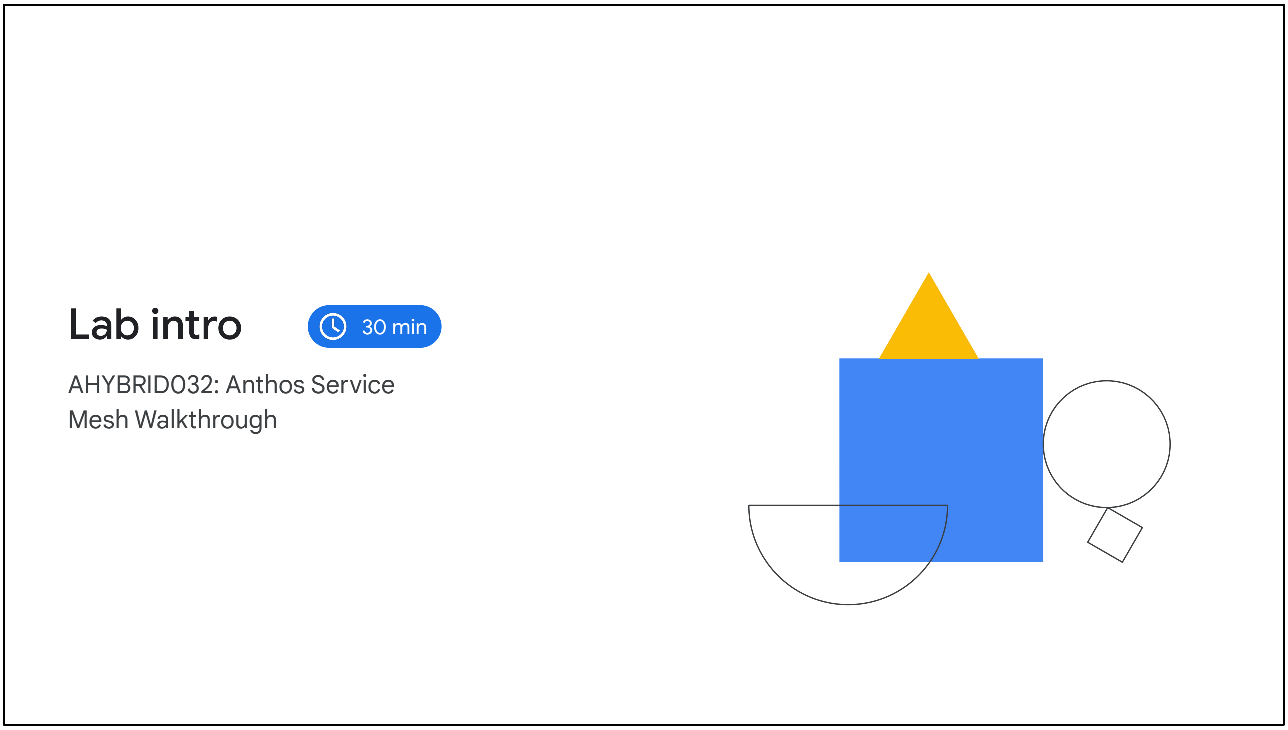
04 Life of a request in the mesh

05 Mesh telemetry and instrumentation

06 Anthos Service Mesh dashboards

07 Anthos Service Mesh pricing and support

We'll start with an Introduction to Anthos Service Mesh, by doing a lab exercise.



Lab intro

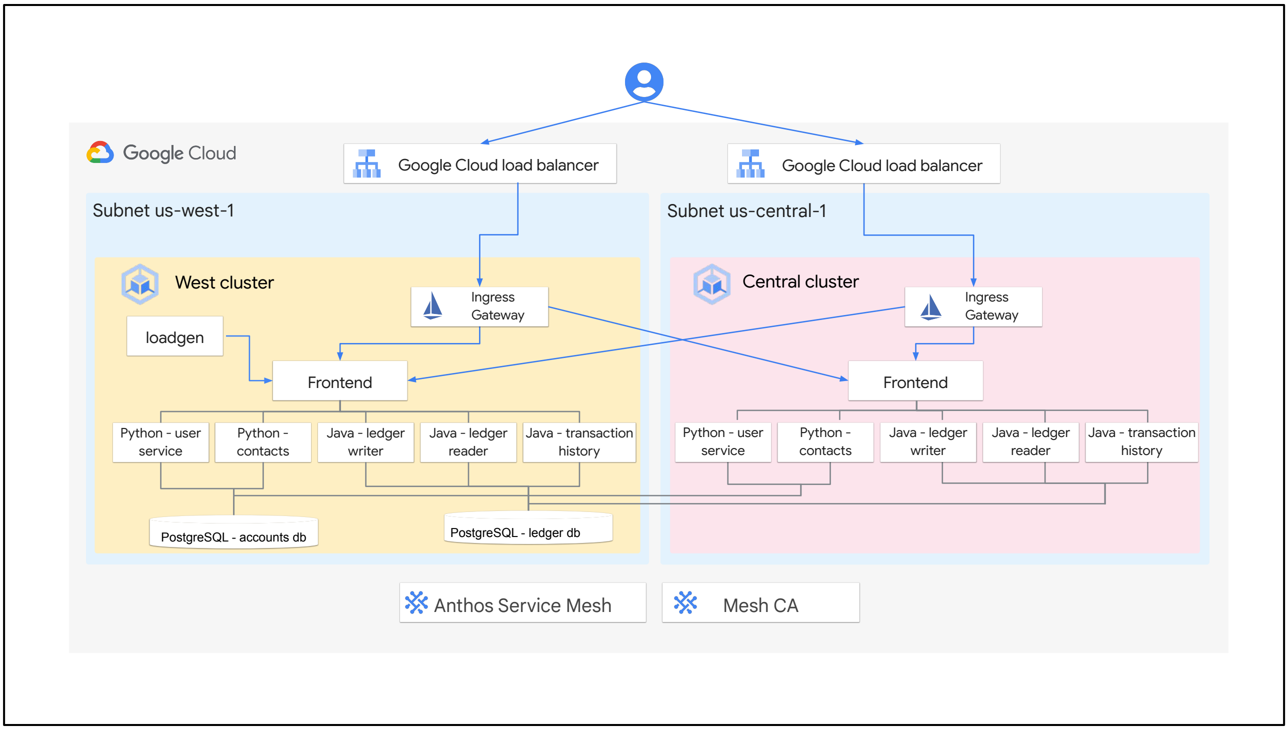
30 min

AHYBRID032: Anthos Service Mesh Walkthrough

Instead of talking through all the features that Anthos Service Mesh offers, let's go directly to the first lab, which provides a walkthrough of its capabilities. After the lab, you can can continue with the module, which covers these features and capabilities in more detail.

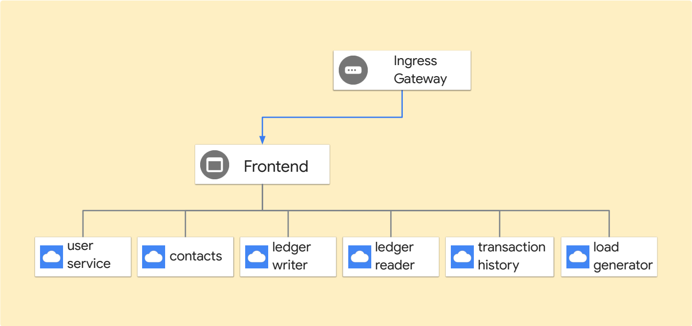
In this lab, you perform the following tasks:

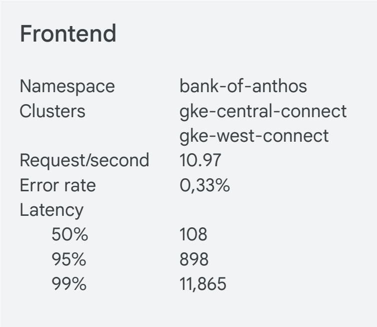
- Explore the app and its workloads deployed in Anthos clusters
- Force cross-cluster traffic routing
- Observe distributed services
- Verify service mesh security



In this lab, we saw a multi-cluster GKE environment running Anthos Service Mesh. Services were deployed into multiple clusters as distributed services.

Lab review: Routing distributed services across clusters



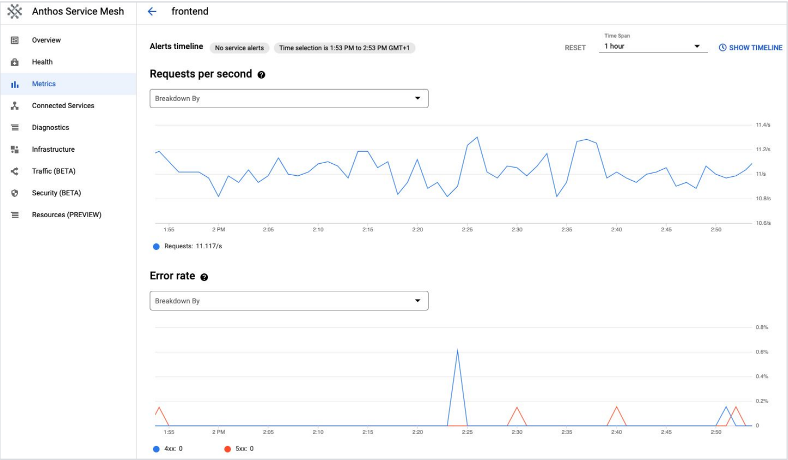


Frontend

Namespace	bank-of-anthos
Clusters	gke-central-connect gke-west-connect
Request/second	10.97
Error rate	0,33%
Latency	
50%	108
95%	898
99%	11,865

Distributed services provide multi-regional availability and remain up even if one or more GKE clusters are down, as long as the healthy clusters are able to serve the load.

Lab review: Service telemetry and observability



Alerts timeline: No service alerts — Time selection is 1:53 PM to 2:53 PM GMT+1

Time Span: 1 hour

Requests per second

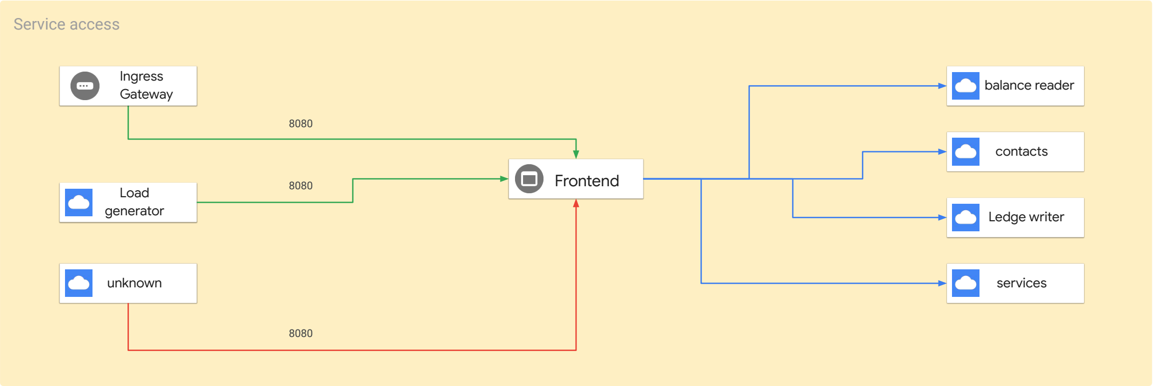
● Requests: 11.117/s

Error rate

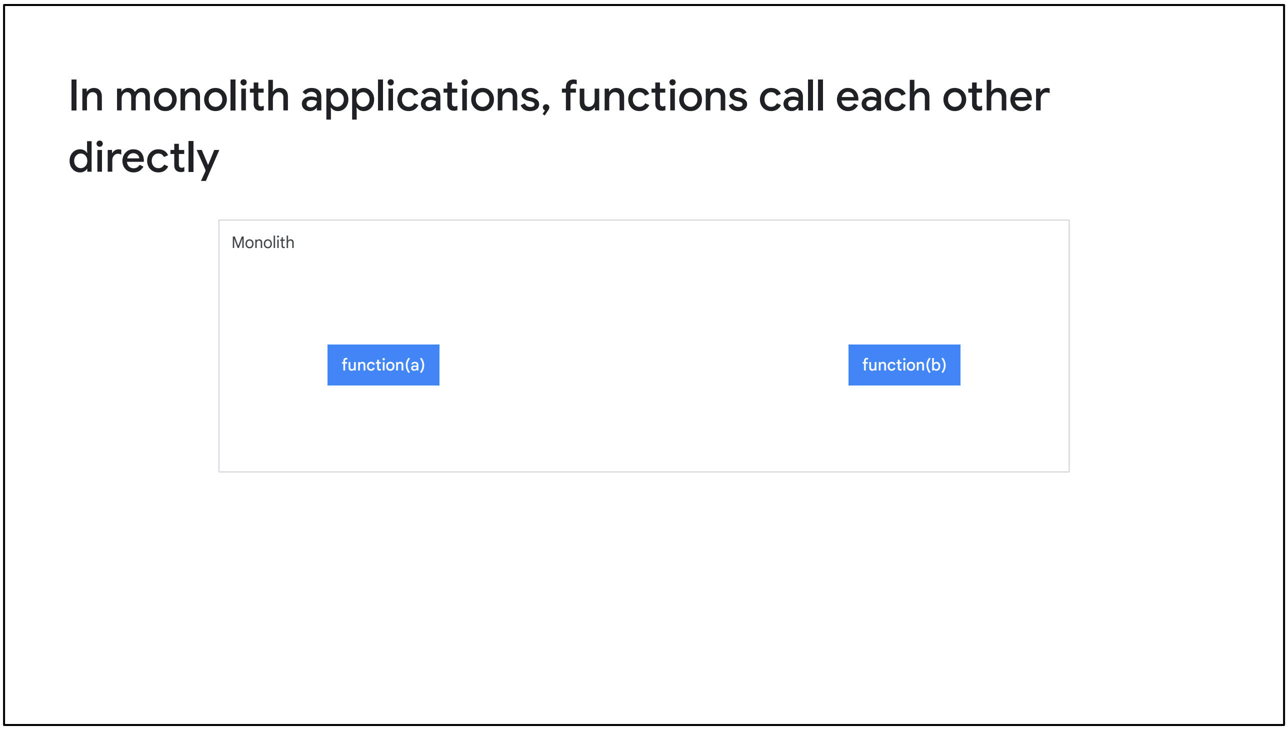
● 4xx: 0 ● 5xx: 0

All service telemetry was collected and aggregated in the Anthos Service Mesh dashboards, and made ready for consumption, so that you can analyze data over time and make sure your services meet your Service Level Objectives, or SLOs.

Lab review: Secure service-to-service communication

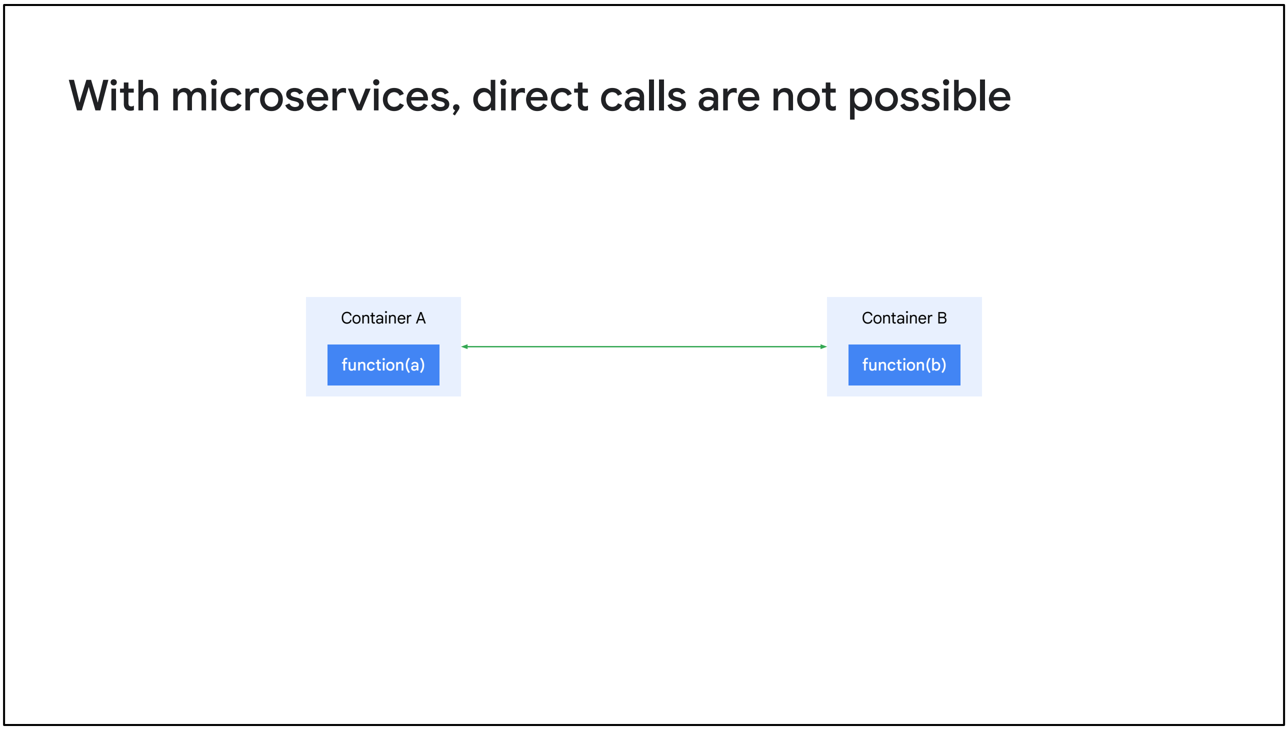


All traffic between services was encrypted over mTLS and you had the possibility to implement and enforce policies to and from every service.



In monolith applications, functions call each other directly

In monolith applications, functions call each other directly.



With microservices, direct calls are not possible

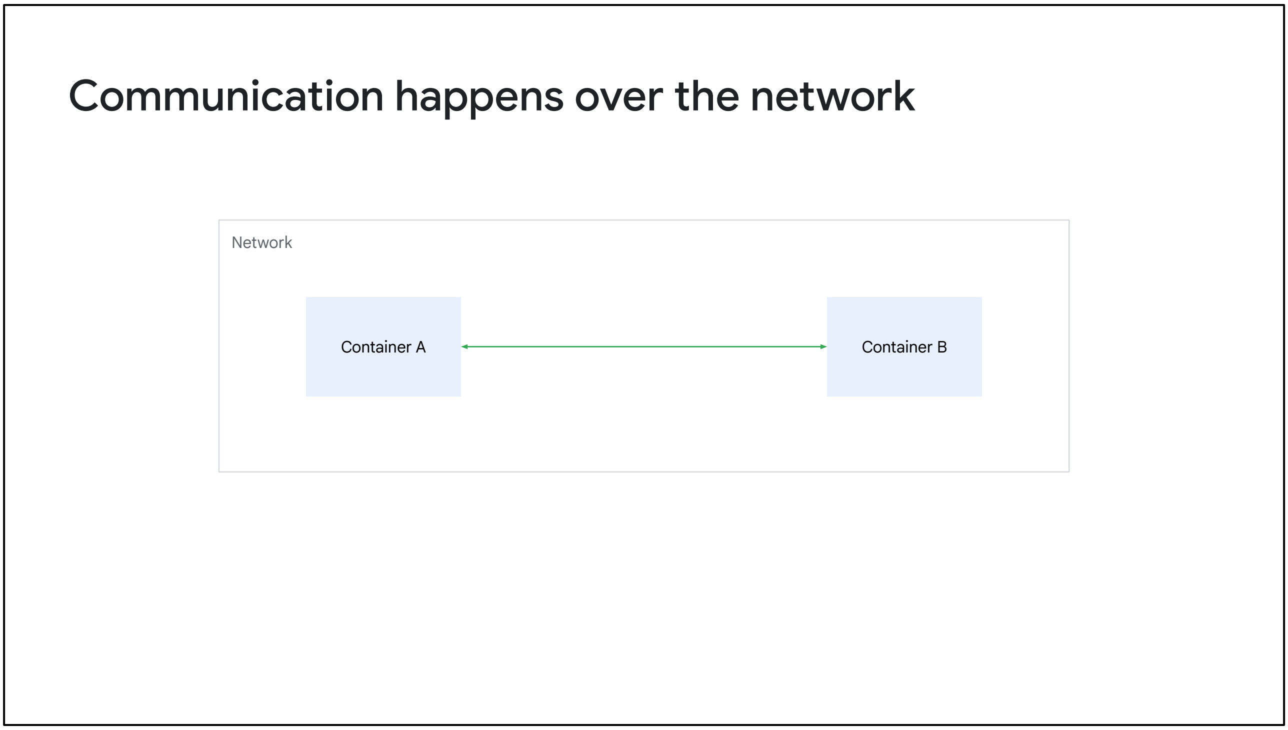
Container A

function(a)

Container B

function(b)

When moving to microservices, the functions may be encapsulated in separate services, and those services might be containerized. So, no direct calls can be made from function a to function b.



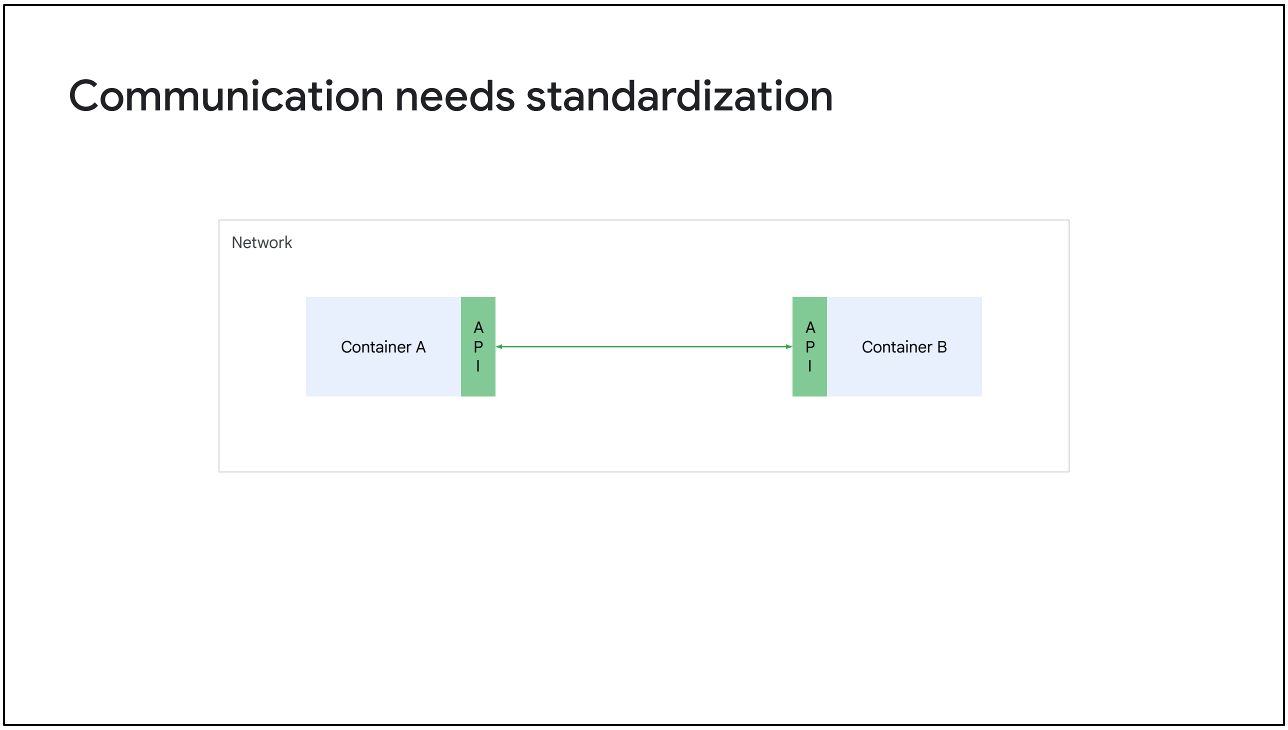
Communication happens over the network

Network

Container A

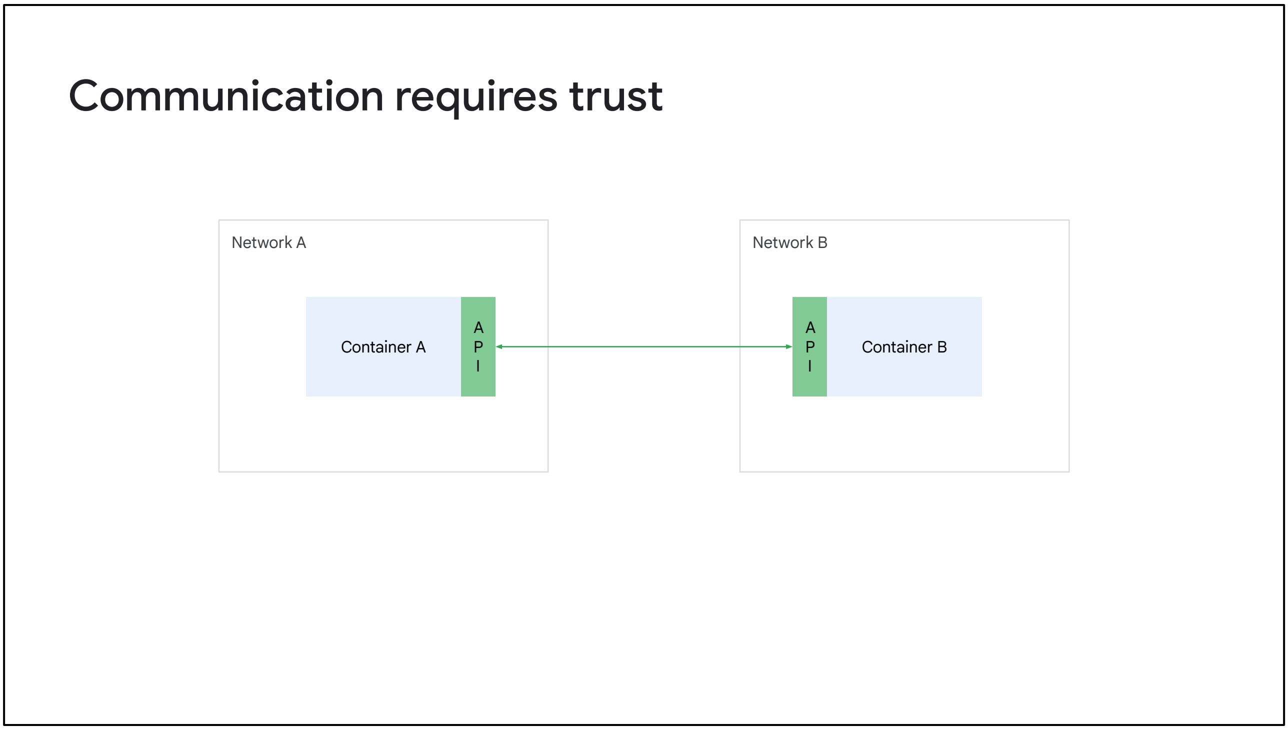
Container B

The services are no longer necessarily inside the same computer, yet they still must communicate with each other. This requires networking.



Communication needs standardization

We want to expose the two services via a contract. We abstract away how we handle the communication and standardize the way that the services communicate.



Communication requires trust

Network A

Container A

A
P
I

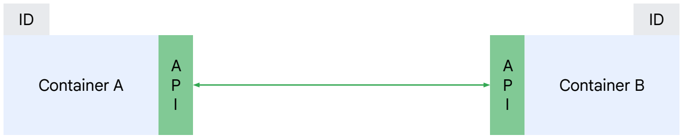
Network B

A
P
I

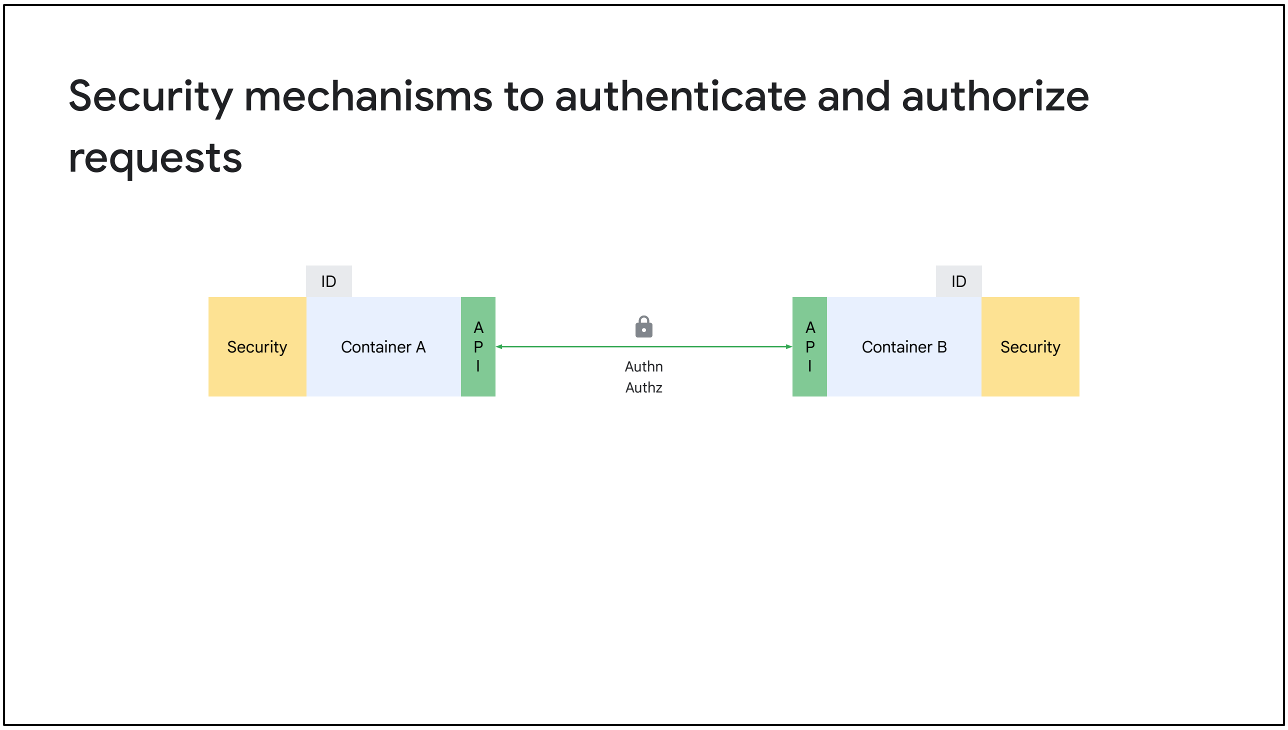
Container B

You don’t want to assume that just because the two containers are on the same network, that Container B should trust Container A. Rather, you want Container A to go through authentication and authorization whenever it calls the API exposed by Container B.

Trust is established between known identities

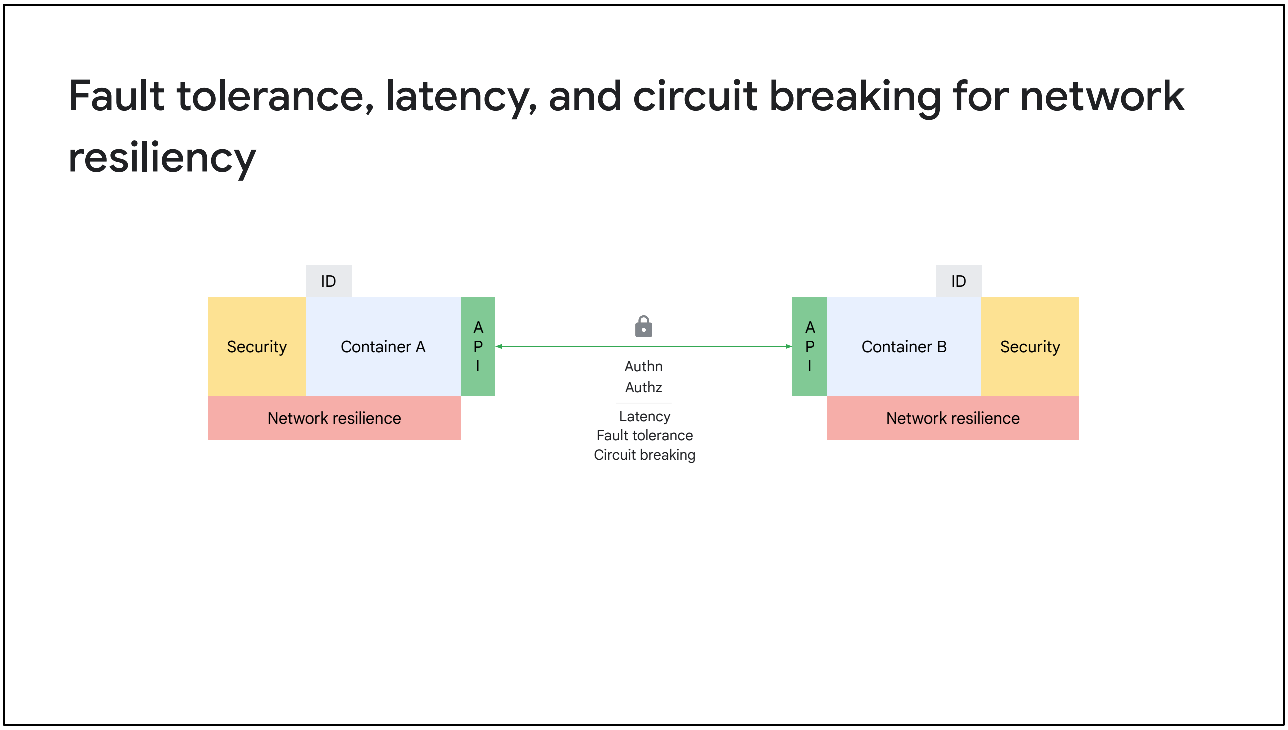


Therefore, each container needs to be provided an identity.



Security mechanisms to authenticate and authorize requests

And there needs to be some security scaffolding that can request/accept the identity of a calling service and handle authorization.



Fault tolerance, latency, and circuit breaking for network resiliency

ID

Security

Container A

API

Network resilience

Authn

Authz

Latency

Fault tolerance

Circuit breaking

ID

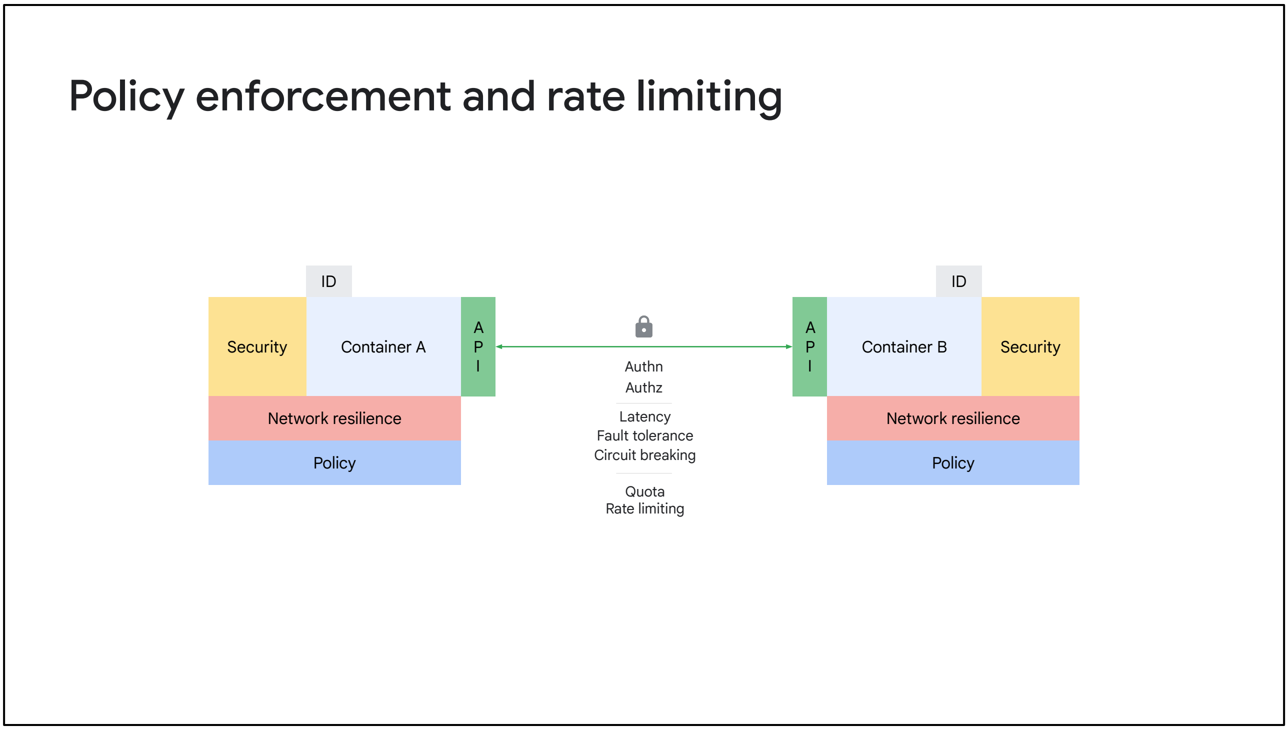
API

Container B

Security

Network resilience

In addition, developers and architects must think about fault tolerance, latency, circuit breakers, etc. This requires additional scaffolding.



Policy enforcement and rate limiting

ID
Security
Container A
API
Network resilience
Policy

Authn
Authz

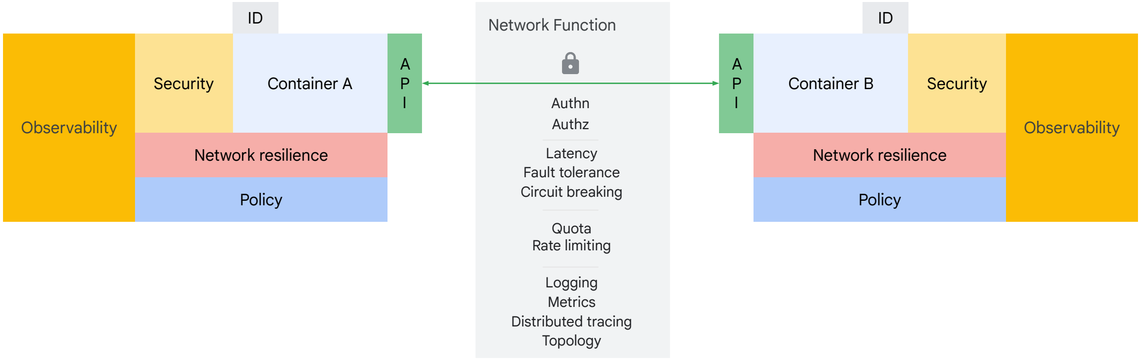
Latency
Fault tolerance
Circuit breaking

Quota
Rate limiting

ID
API
Container B
Security
Network resilience
Policy

The application may also need to enforce quotas and rate limiting and other API access policies. Thus, more supporting functionality needs to be in place.

Observability with logs, metrics, tracing, and topology


ID

Observability

Security

Container A

API

Network resilience

Policy

Network Function

Authn
Authz

Latency
Fault tolerance
Circuit breaking

Quota
Rate limiting

Logging
Metrics
Distributed tracing
Topology

ID

API

Container B

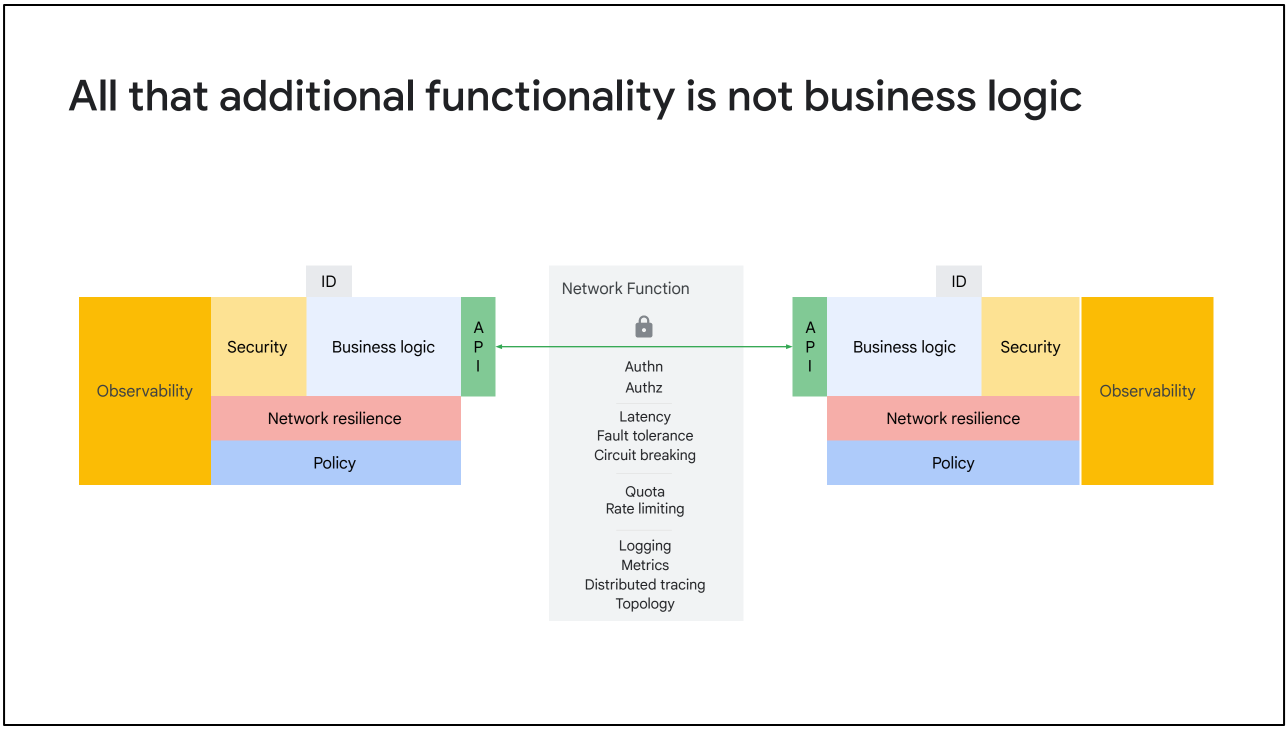
Security

Observability

Network resilience

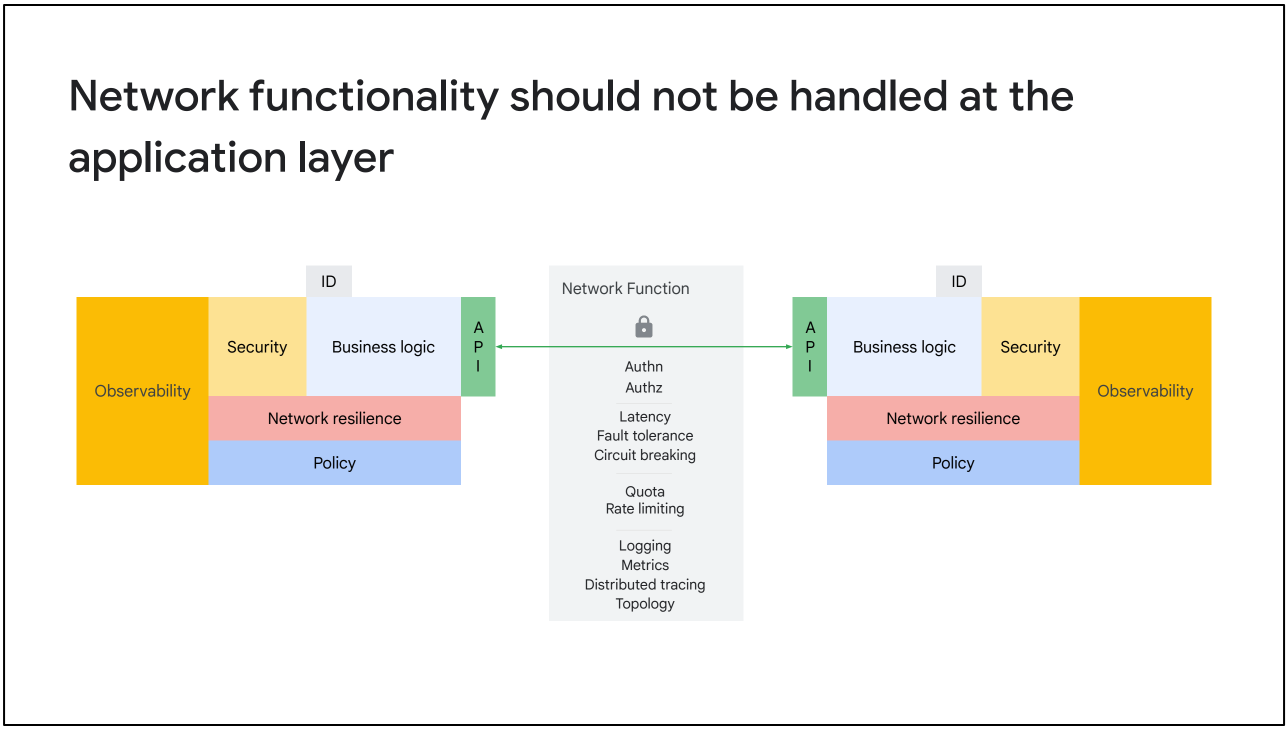
Policy

Lastly, if you want observability data like logs, metrics, tracing, and topology, you need yet more supporting functionality.



All that additional functionality is not business logic

So, there is a lot of technical functionality that’s required for successful operations of your application. This functionality is replicated for every instance of your service, but it really has nothing to do with the business logic of the services themselves.



Network functionality should not be handled at the application layer

ID

Observability

Security

Business logic

API

Network resilience

Policy

Network Function

Authn
Authz

Latency
Fault tolerance
Circuit breaking

Quota
Rate limiting

Logging
Metrics
Distributed tracing
Topology

ID

API

Business logic

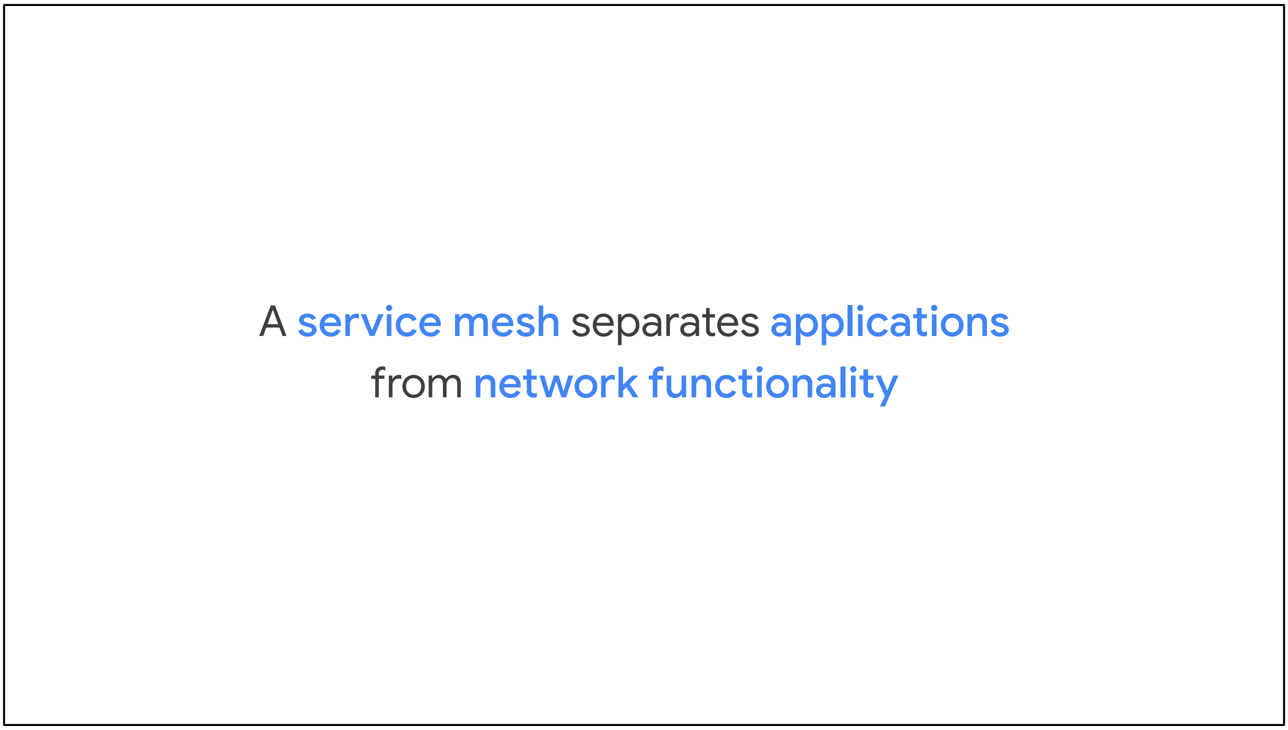
Security

Observability

Network resilience

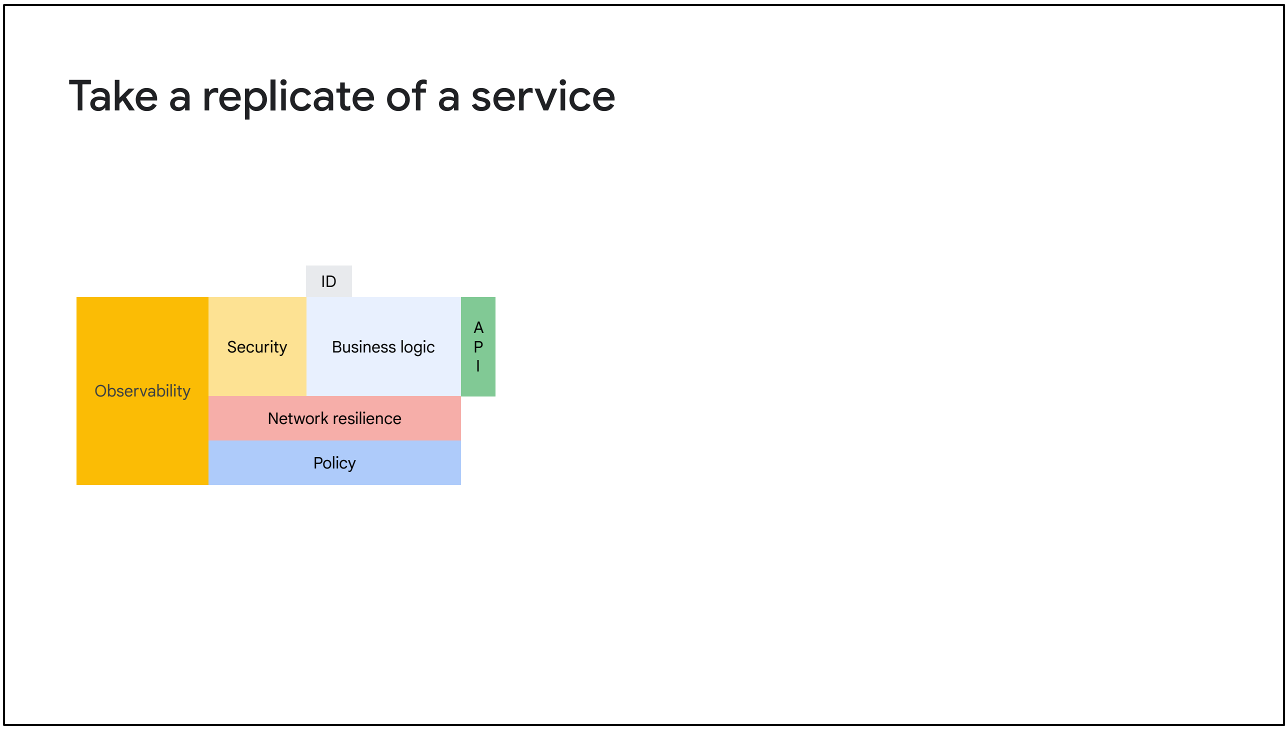
Policy

This is really networking functionality, and perhaps can be handled at the network level rather than the application level.



A service mesh separates applications from network functionality

A service mesh separates applications from network functionality.



Take a replicate of a service

ID

Observability

Security

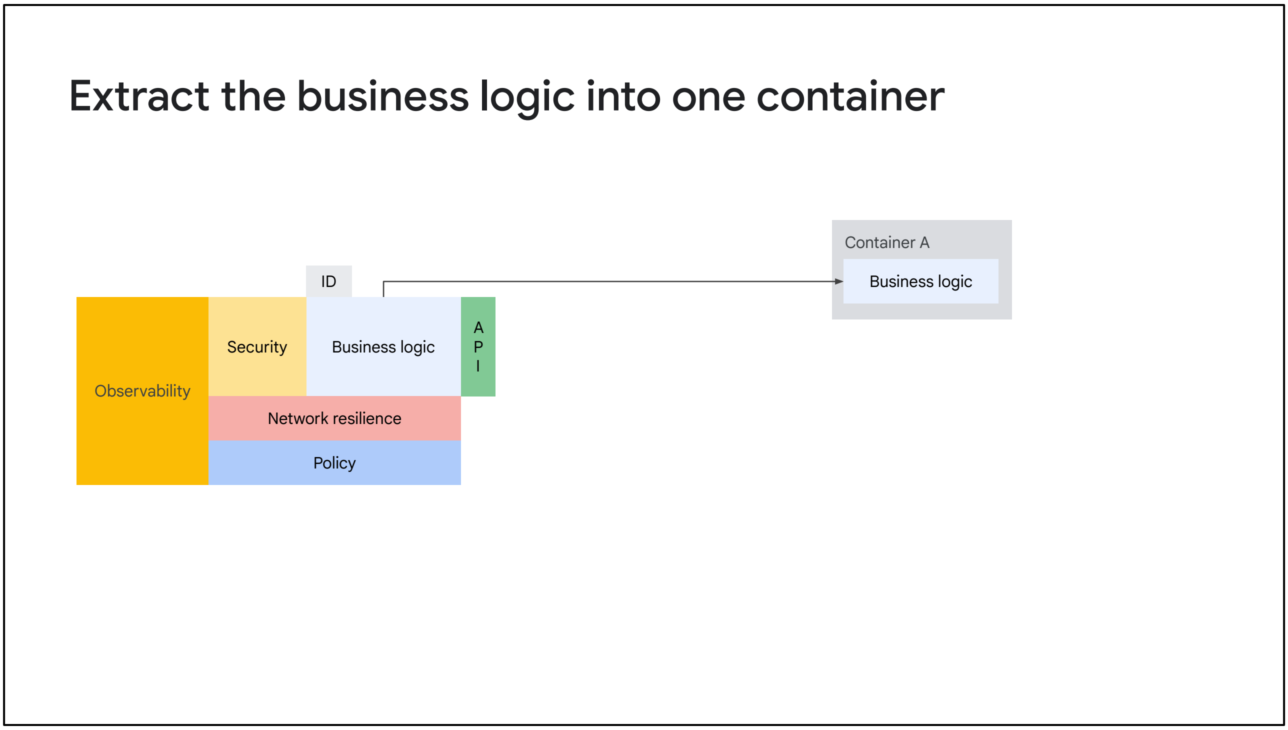
Business logic

A
P
I

Network resilience

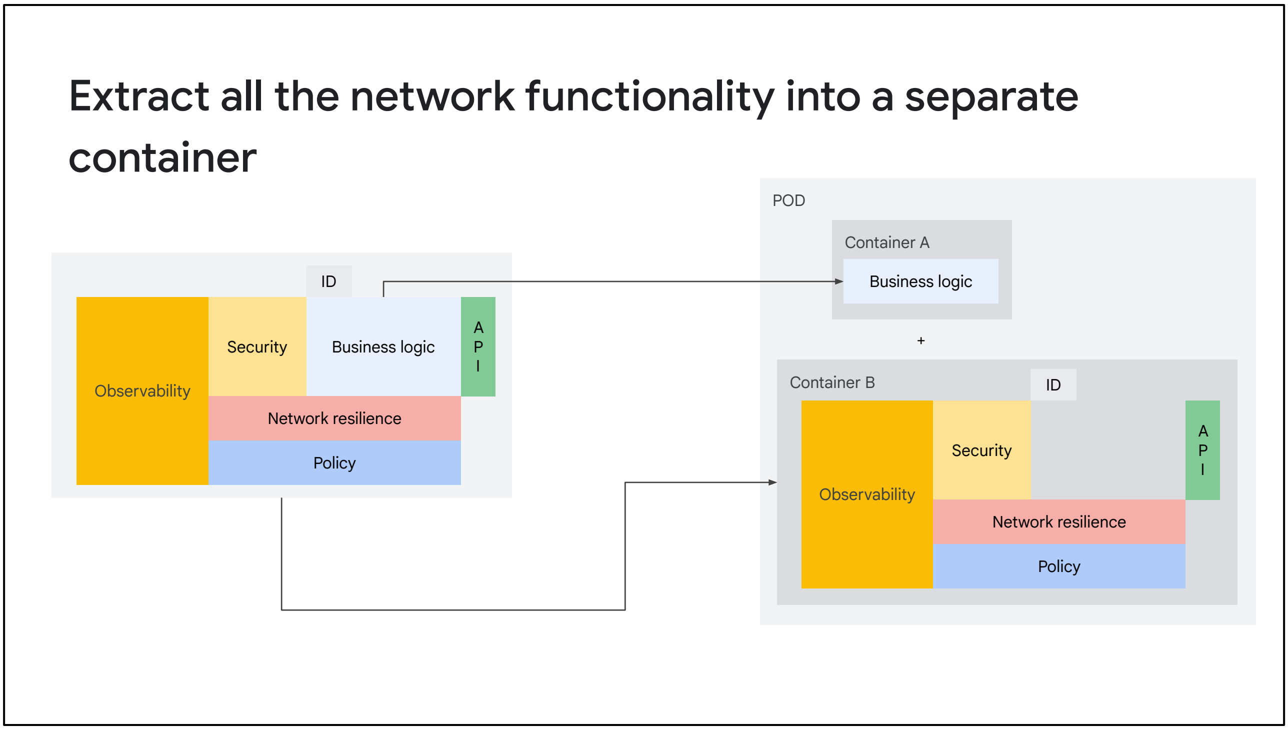
Policy

One approach is to take a replica of a service.



Extract the business logic into one container

Extract the business logic into one container.



Extract all the network functionality into a separate container

ID

Observability

Security

Business logic

A
P
I

Network resilience

Policy

POD

Container A

Business logic

+

Container B

ID

Observability

Security

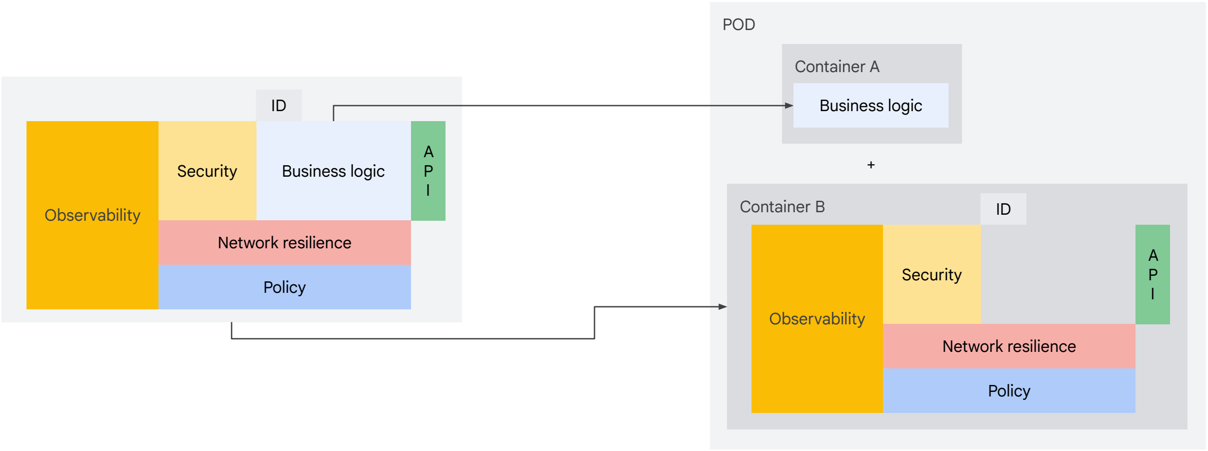
A
P
I

Network resilience

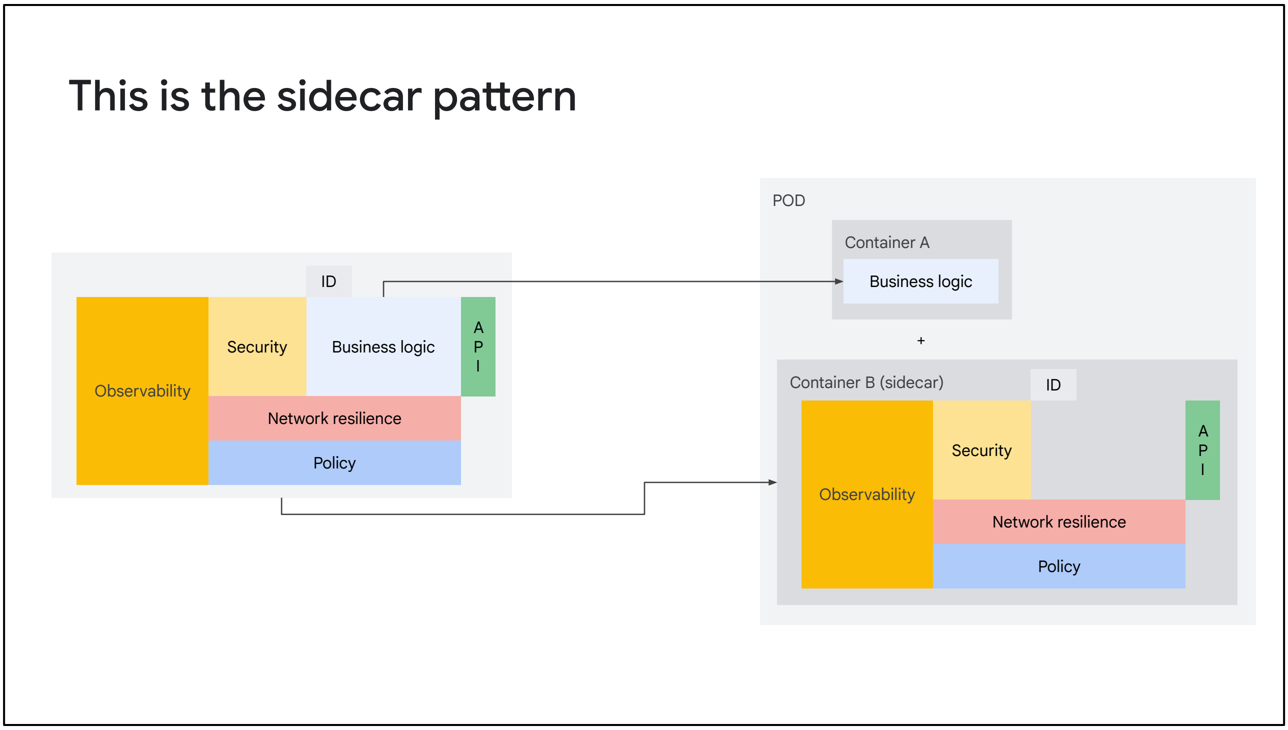
Policy

And then extract all the network functionality into a separate container.

In Kubernetes, you can place two containers on a pod

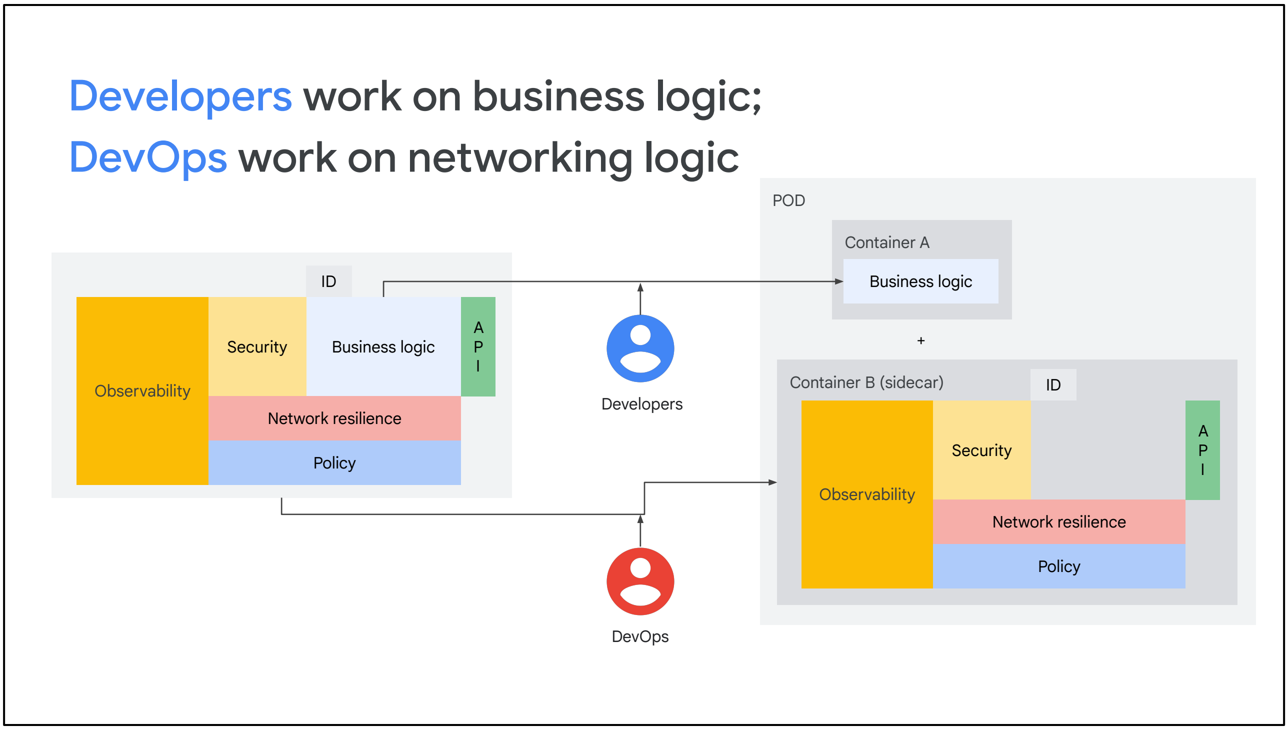


In Kubernetes, you would place the two containers together into a pod, which represents one instance of your service.



This is the sidecar pattern

This pattern of marrying a business logic container with a utility container that handles additional technical functionality is called the sidecar pattern.



Developers work on business logic; DevOps work on networking logic

ID

Observability

Security

Business logic

A P I

Network resilience

Policy

Developers

DevOps

POD

Container A

Business logic

+

Container B (sidecar)

ID

Observability

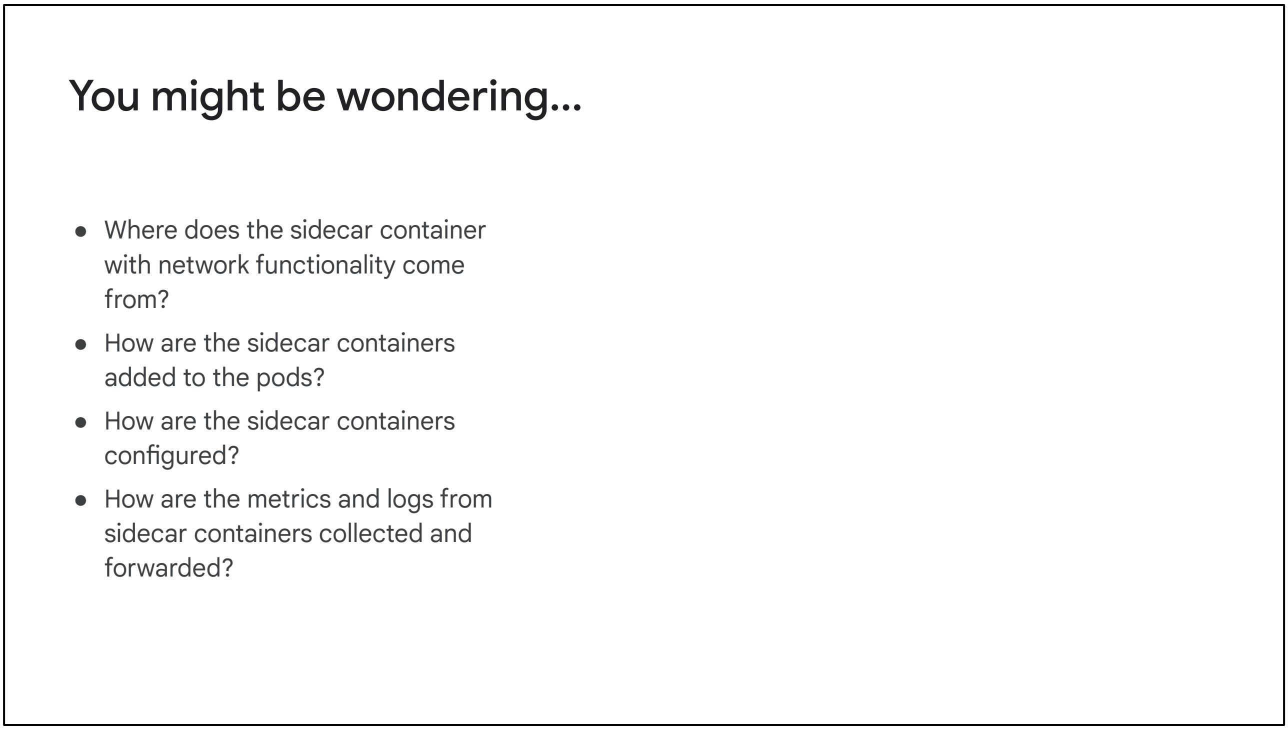
Security

A P I

Network resilience

Policy

This allows developers to focus on implementation of business logic, without having to invest in all the surrounding technology. Meanwhile, DevOps/SRE teams can focus on building all the technical scaffolding that adds valuable functionality around the business logic.

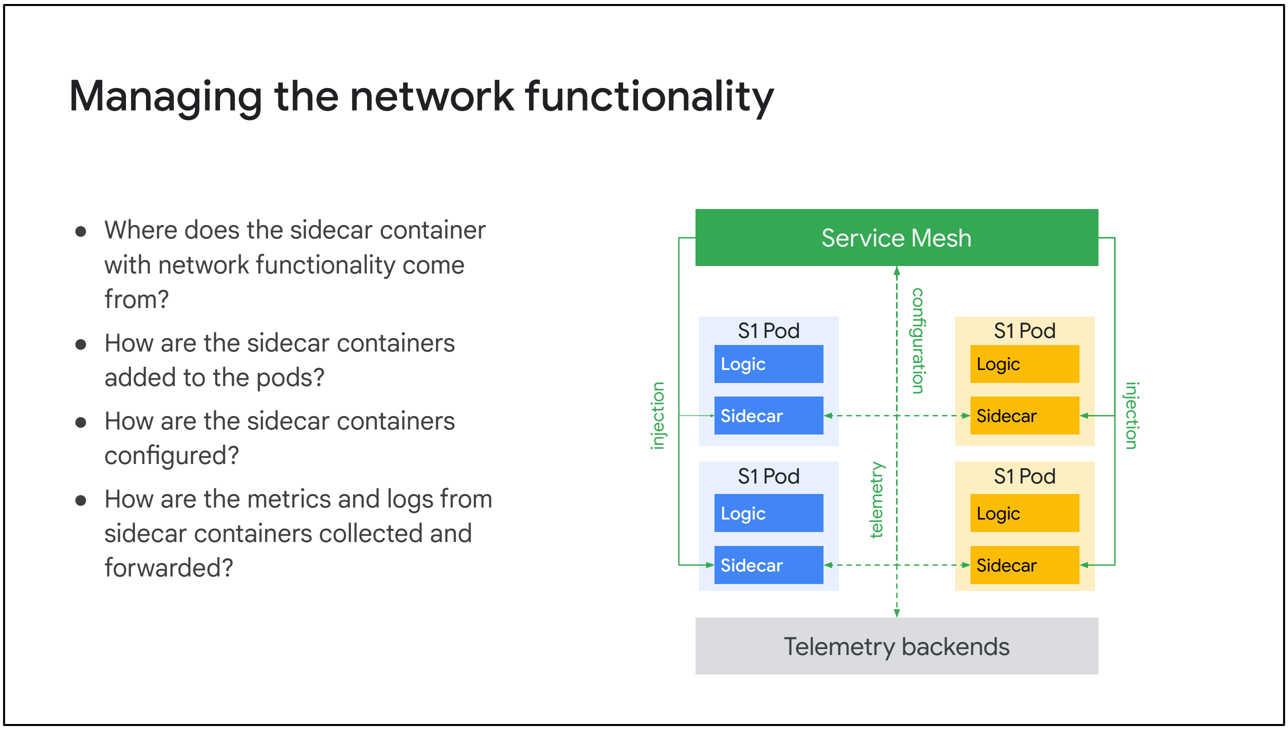


You might be wondering...

- Where does the sidecar container with network functionality come from?
- How are the sidecar containers added to the pods?
- How are the sidecar containers configured?
- How are the metrics and logs from sidecar containers collected and forwarded?

You might be wondering…

- Where does the sidecar container with network functionality come from?
- How are the sidecar containers added to the pods?
- How are the sidecar containers configured?
- How are the metrics and logs from sidecar containers collected and forwarded?

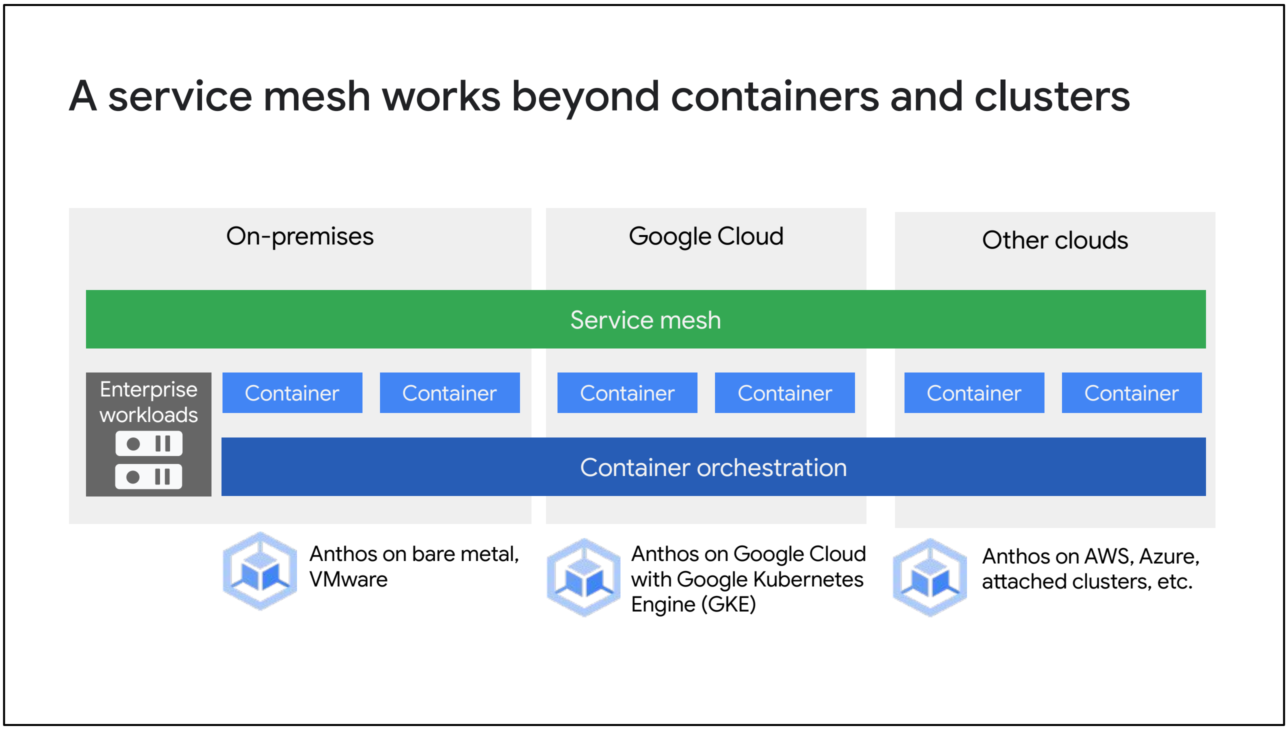


Managing the network functionality

- Where does the sidecar container with network functionality come from?
- How are the sidecar containers added to the pods?
- How are the sidecar containers configured?
- How are the metrics and logs from sidecar containers collected and forwarded?

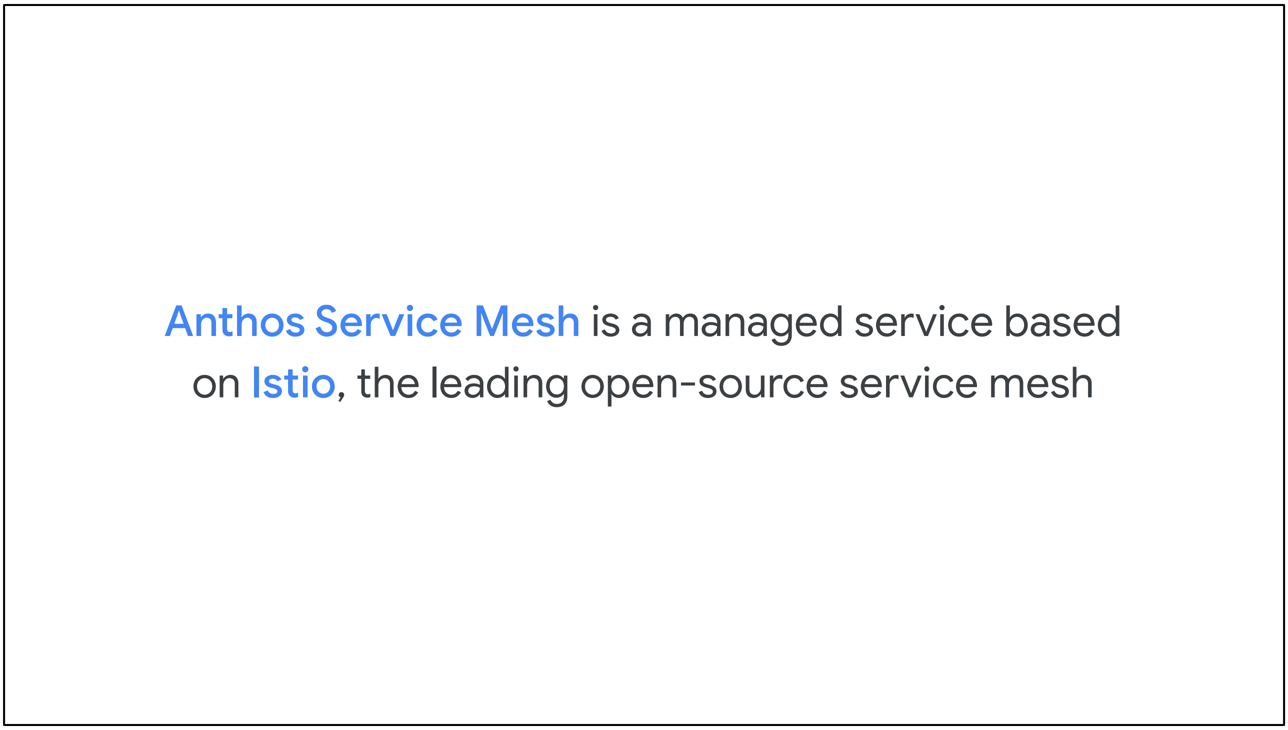
The service mesh provides, manages, and works with the sidecar containers to make everything work across all your services.

- First, a DevOps operator configures the service mesh with security and routing policies.
- Second, on pod creation, the service mesh injects the sidecar container and configures it with the network configuration.
- Third, communication is handled by the injected container and metrics, logs, and traces get exported from the sidecars to enhance observability.



A service mesh works beyond containers and clusters

While service mesh started with containers and microservices, its benefits can be applied to traditional applications. Service mesh addresses workloads that stretch across clusters and environments with isolation and security. It extends beyond Kubernetes container clusters to bare metal servers and virtual machines and links them together in an elegant, logical, and secure manner. That way, you can modernize existing brownfield applications in place, without the need to go through the expensive process of rewriting them.

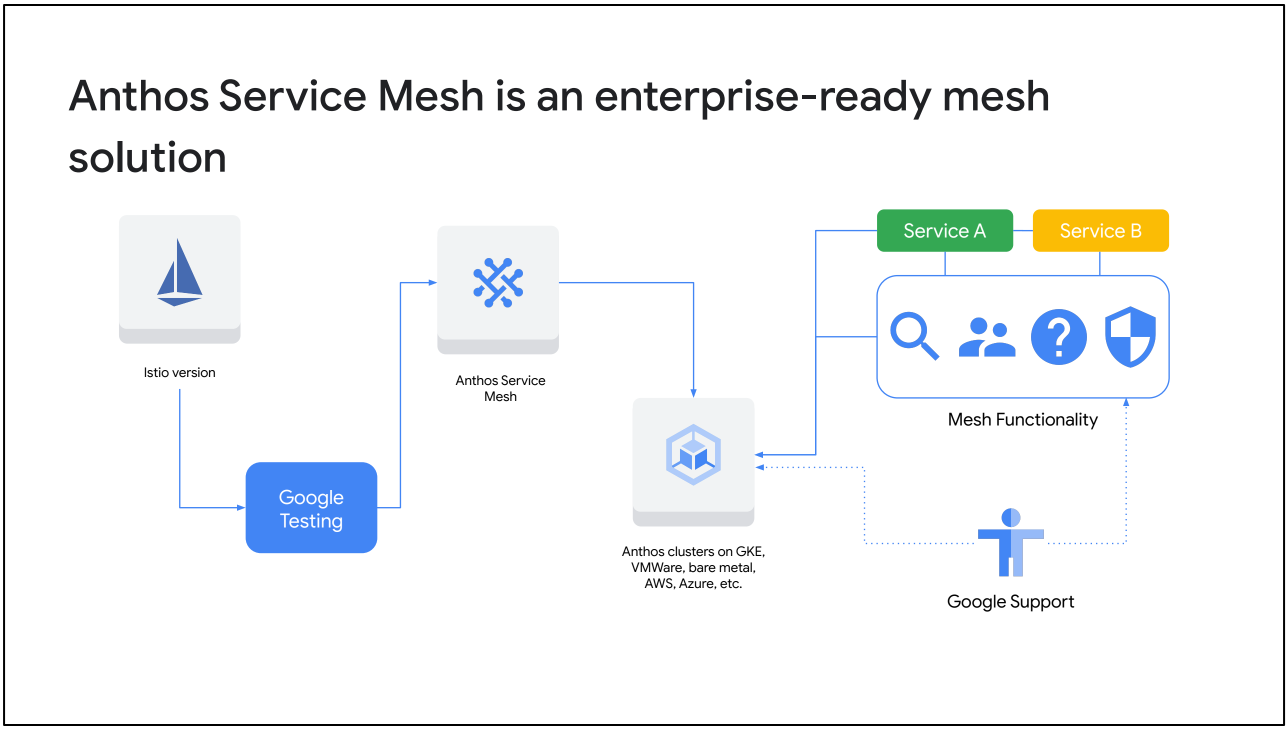


Anthos Service Mesh is a managed service based on Istio, the leading open-source service mesh

In the previous video, you learned what a service mesh can do. Let’s now discuss the specifics of Anthos Service Mesh.

Anthos Service Mesh is a managed service based on Istio, the leading open-source service mesh.

Istio draws on a lot of inspiration from Google-internal systems, but is built by multiple parties, including Google, IBM, and Lyft, the ride-sharing company.



Anthos Service Mesh is an enterprise-ready mesh solution

Istio version

Google Testing

Anthos Service Mesh

Anthos clusters on GKE, VMWare, bare metal, AWS, Azure, etc.

Service A

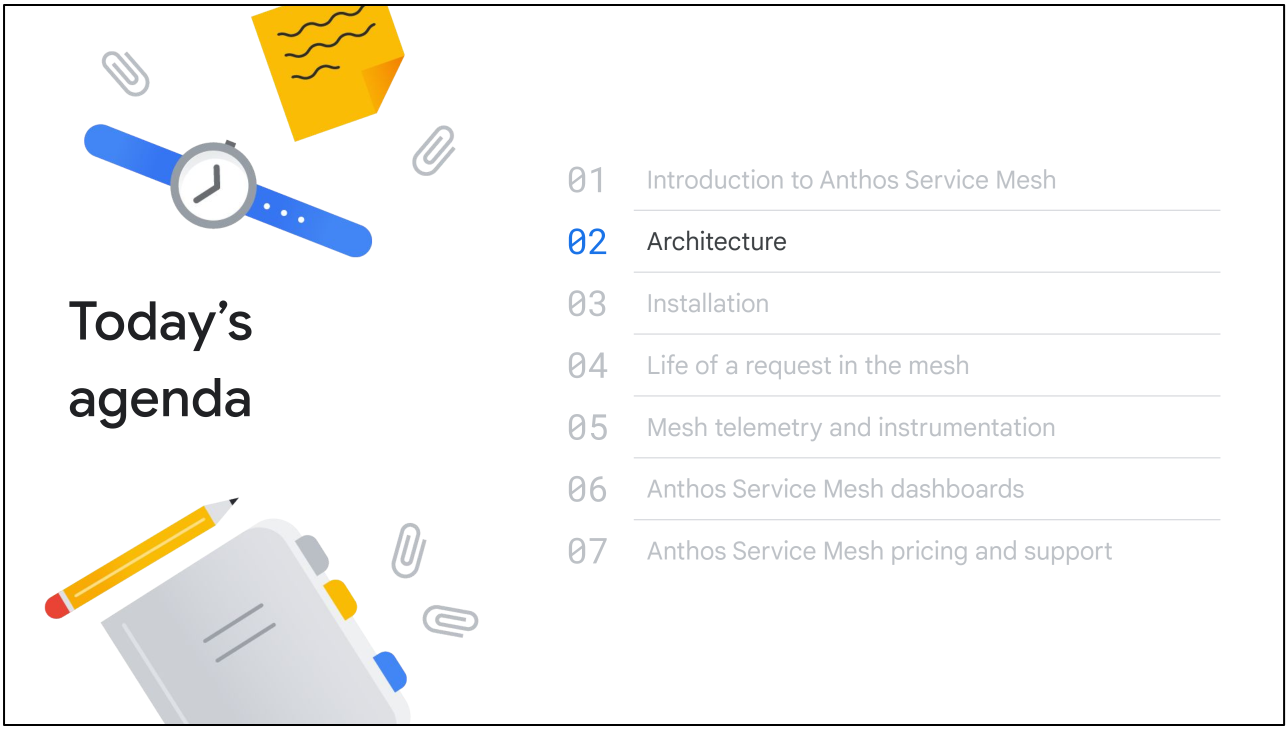
Service B

Mesh Functionality

Google Support

Google takes new Istio versions, tests them, adds in some Google Cloud-specific components that improve management, and makes them available for installation on all your Anthos clusters.

- Google Cloud provides support and SLAs.
- When installed, Anthos Service Mesh provides all the standard Istio goodness with additional features such as the user interface.
- Anthos Service Mesh can be purchased as a standalone product or as part of the Anthos subscription.



Today's agenda

01 Introduction to Anthos Service Mesh

02 Architecture

03 Installation

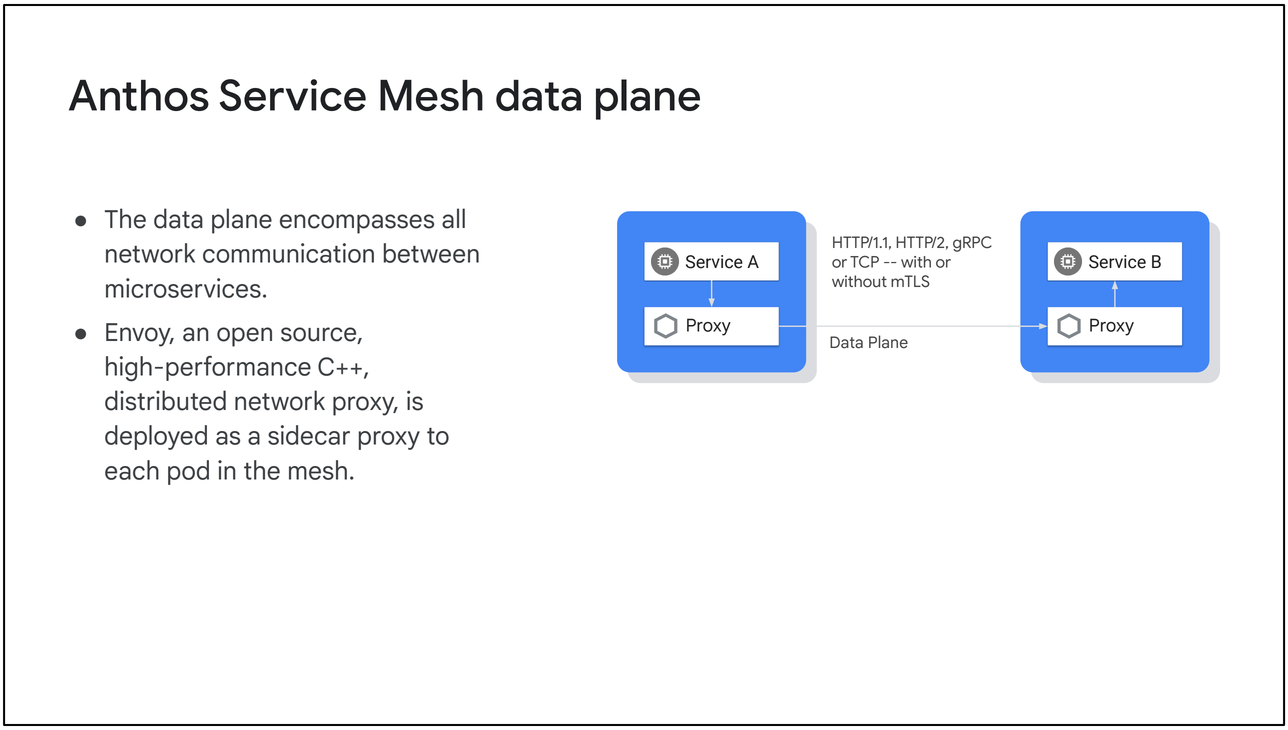
04 Life of a request in the mesh

05 Mesh telemetry and instrumentation

06 Anthos Service Mesh dashboards

07 Anthos Service Mesh pricing and support

Let's talk about the architecture of Anthos Service Mesh.



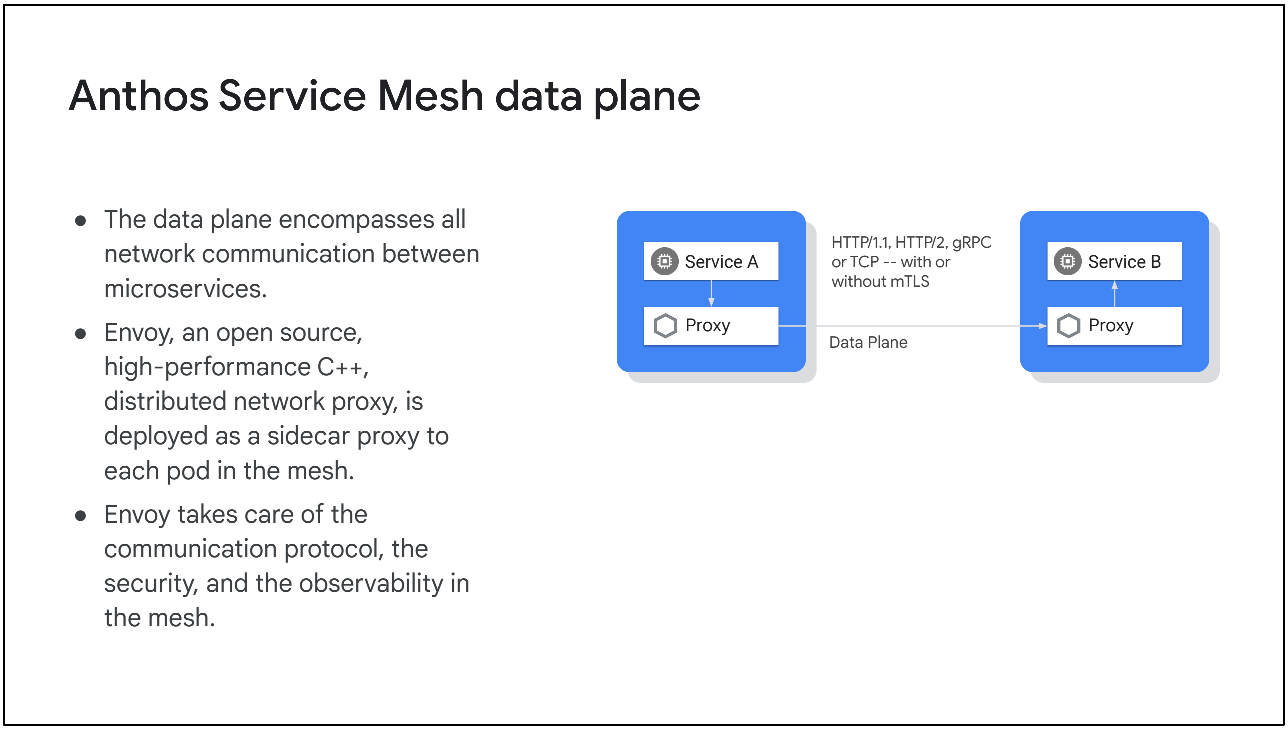
Anthos Service Mesh data plane

- The data plane encompasses all network communication between microservices.
- Envoy, an open source, high-performance C++, distributed network proxy, is deployed as a sidecar proxy to each pod in the mesh.

Anthos Service Mesh is composed of the data plane and the control plane.

The data plane encompasses all network communication between microservices.

It does so by leveraging Envoy, an open-source, high-performance C++, distributed network proxy, that is deployed as a sidecar proxy to each Kubernetes pod in the mesh.



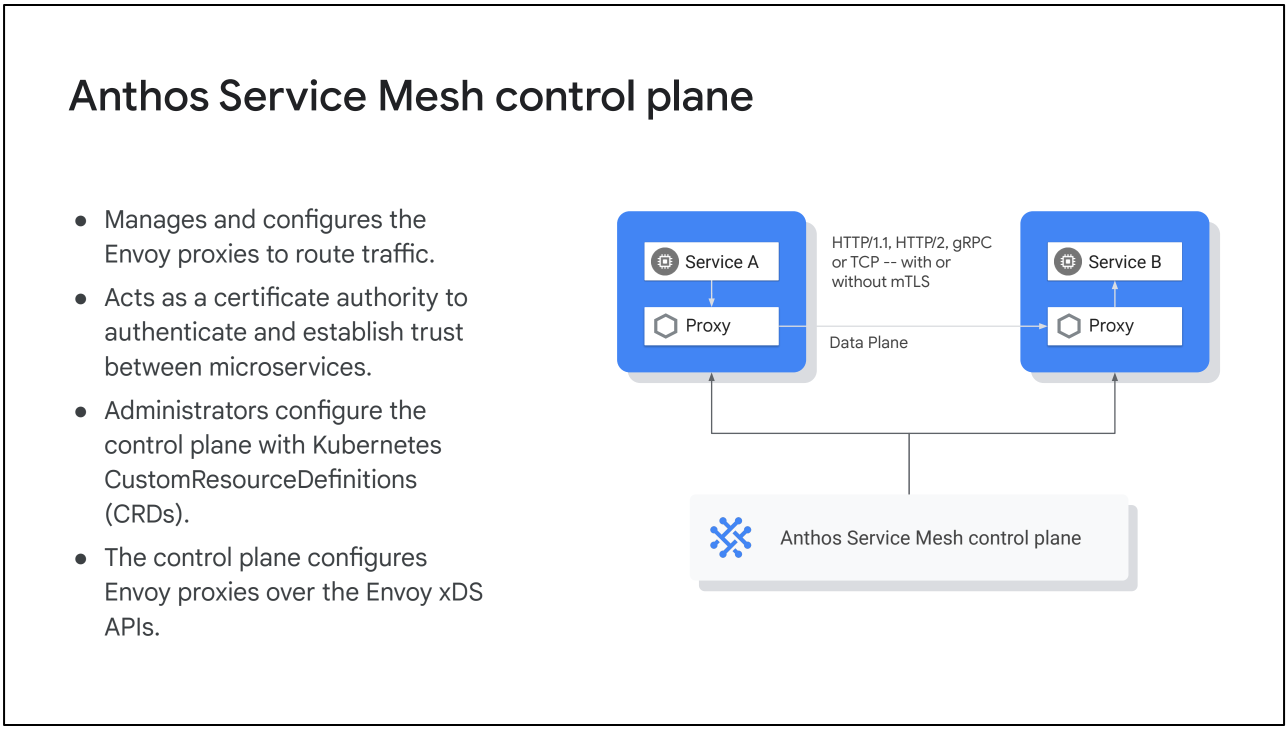
Anthos Service Mesh data plane

- The data plane encompasses all network communication between microservices.
- Envoy, an open source, high-performance C++, distributed network proxy, is deployed as a sidecar proxy to each pod in the mesh.
- Envoy takes care of the communication protocol, the security, and the observability in the mesh.

Envoy takes care of the communication protocol, security, and observability in the mesh.

You can use:

- Protocols such as HTTP 1 and 2, gRPC or TCP.
- Encryption over TLS, mutual TLS, or none at all, having the communication go over plain text.
- And telemetry reporting on the mesh, capturing information such as requests latencies and package sizes.



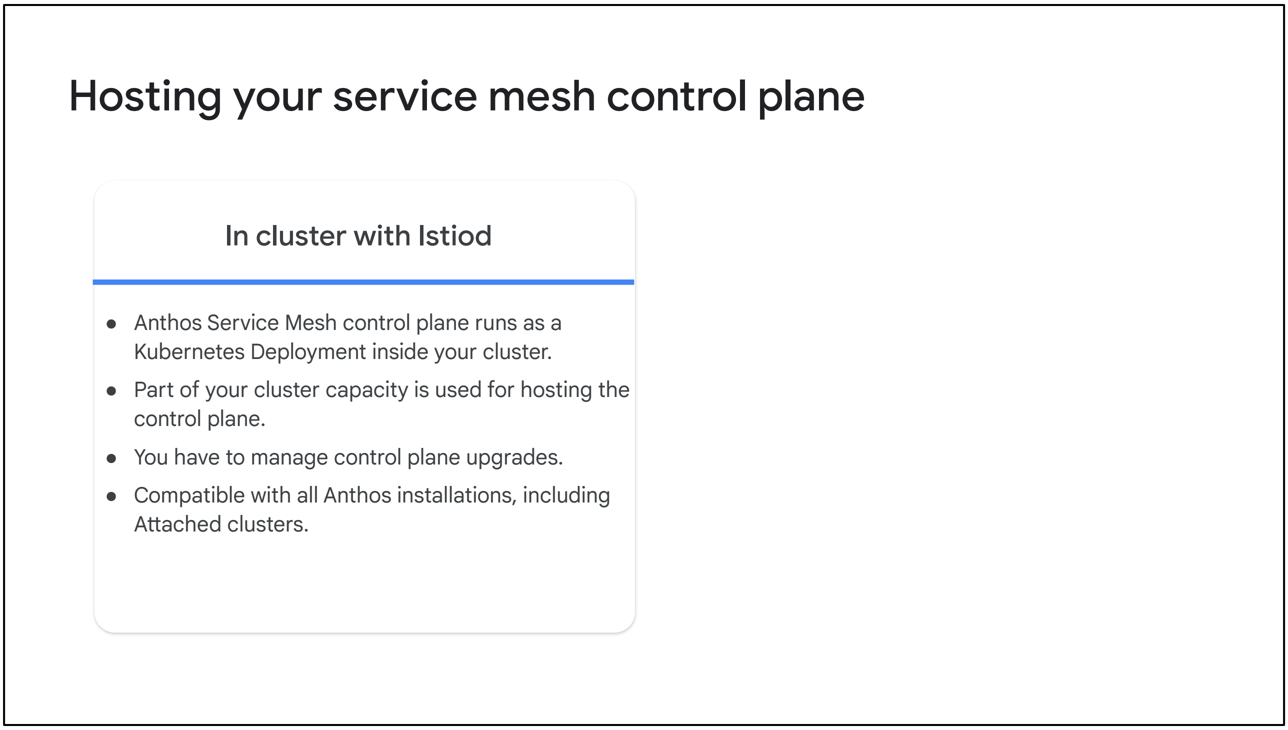
Anthos Service Mesh control plane

- Manages and configures the Envoy proxies to route traffic.
- Acts as a certificate authority to authenticate and establish trust between microservices.
- Administrators configure the control plane with Kubernetes CustomResourceDefinitions (CRDs).
- The control plane configures Envoy proxies over the Envoy xDS APIs.

Service A
Proxy
HTTP/1.1, HTTP/2, gRPC or TCP -- with or without mTLS
Data Plane
Service B
Proxy
Anthos Service Mesh control plane

The control plane manages and configures the Envoy proxies to route traffic and acts as a certificate authority to authenticate and establish trust between microservices.

To configure the Envoy proxies, administrators use Kubernetes CustomResourceDefinitions implemented by the Istio project. Once they are applied, the control plane then synchronizes changes with the Envoy proxies over the Envoy xDS APIs.



Hosting your service mesh control plane

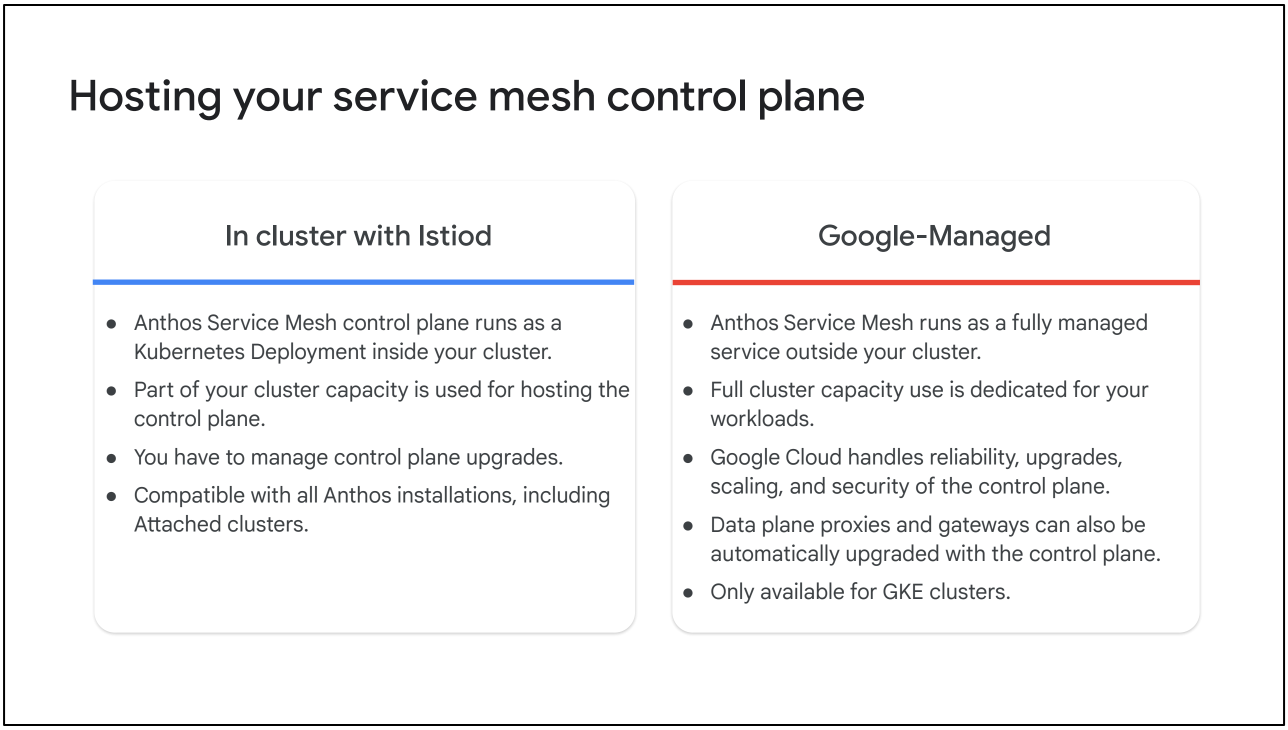
In cluster with Istiod

- Anthos Service Mesh control plane runs as a Kubernetes Deployment inside your cluster.
- Part of your cluster capacity is used for hosting the control plane.
- You have to manage control plane upgrades.
- Compatible with all Anthos installations, including Attached clusters.

You have two options for hosting your service mesh control plane.

You can use the in cluster installation with Istiod. In this case:

- Anthos Service Mesh runs as a Kubernetes Deployment inside your cluster.
- Part of your cluster capacity is used for hosting the control plane.
- You have to manage control plane upgrades.
- It’s compatible with all Anthos installations including attached clusters.



Hosting your service mesh control plane

In cluster with Istiod

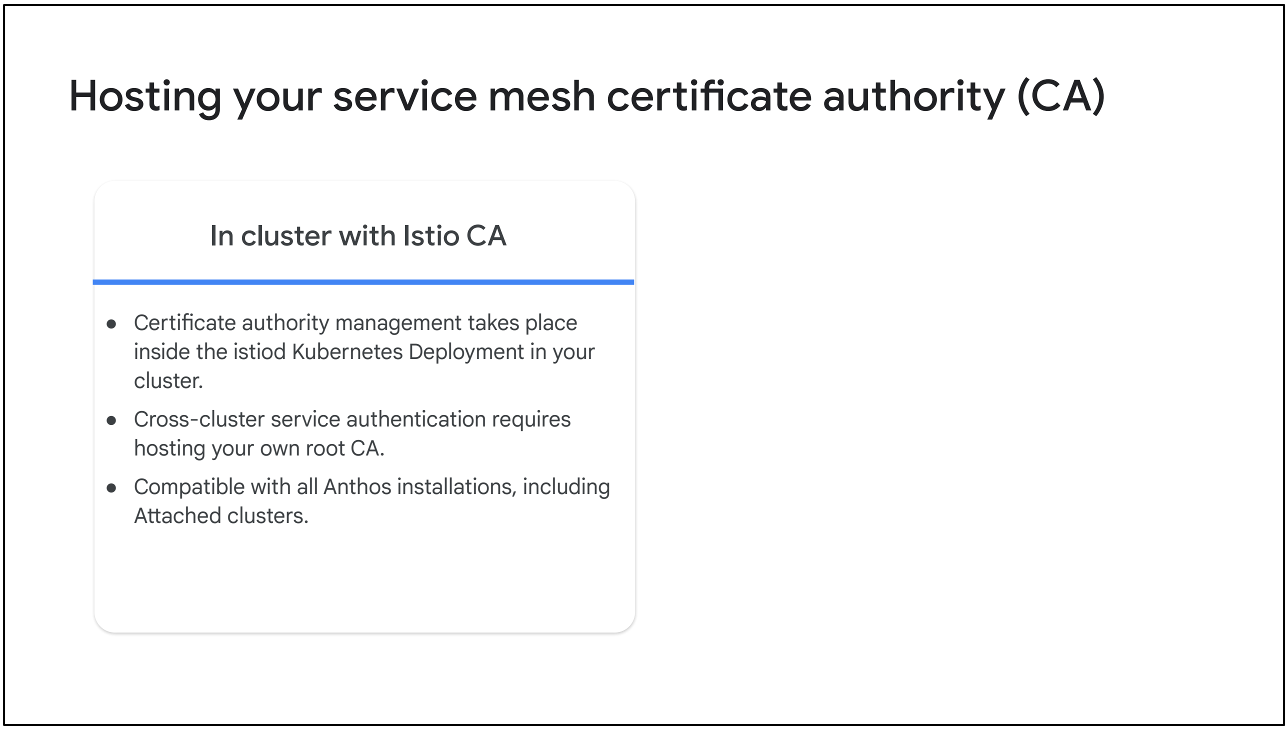
- Anthos Service Mesh control plane runs as a Kubernetes Deployment inside your cluster.
- Part of your cluster capacity is used for hosting the control plane.
- You have to manage control plane upgrades.
- Compatible with all Anthos installations, including Attached clusters.

Google-Managed

- Anthos Service Mesh runs as a fully managed service outside your cluster.
- Full cluster capacity use is dedicated for your workloads.
- Google Cloud handles reliability, upgrades, scaling, and security of the control plane.
- Data plane proxies and gateways can also be automatically upgraded with the control plane.
- Only available for GKE clusters.

Or you can use the Google Managed control plane. In this case:

- Anthos Service Mesh runs as a managed service outside of your cluster.
- Full cluster capacity utilization is dedicated to running your workloads.
- Google Cloud handles reliability, upgrades, scaling, and security of the control plane.
- Optionally, you can enable a Google-managed data plane, so that proxies and gateways are also automatically upgraded with the control plane. If you do so, you can subscribe your mesh to the Rapid, Regular, or Stable channel, depending on the cadence of updates that you want. Rapid uses the latest version of Anthos Service Mesh with the newest features. Regular offers a balance of feature availability and release stability, and is what we recommend for most users. And Stable prioritizes stability over new functionality, so you get a version that has been tested and validated on the Rapid and Regular channels.
- Anthos Service Mesh with a managed control plane is only available for GKE clusters.



Hosting your service mesh certificate authority (CA)

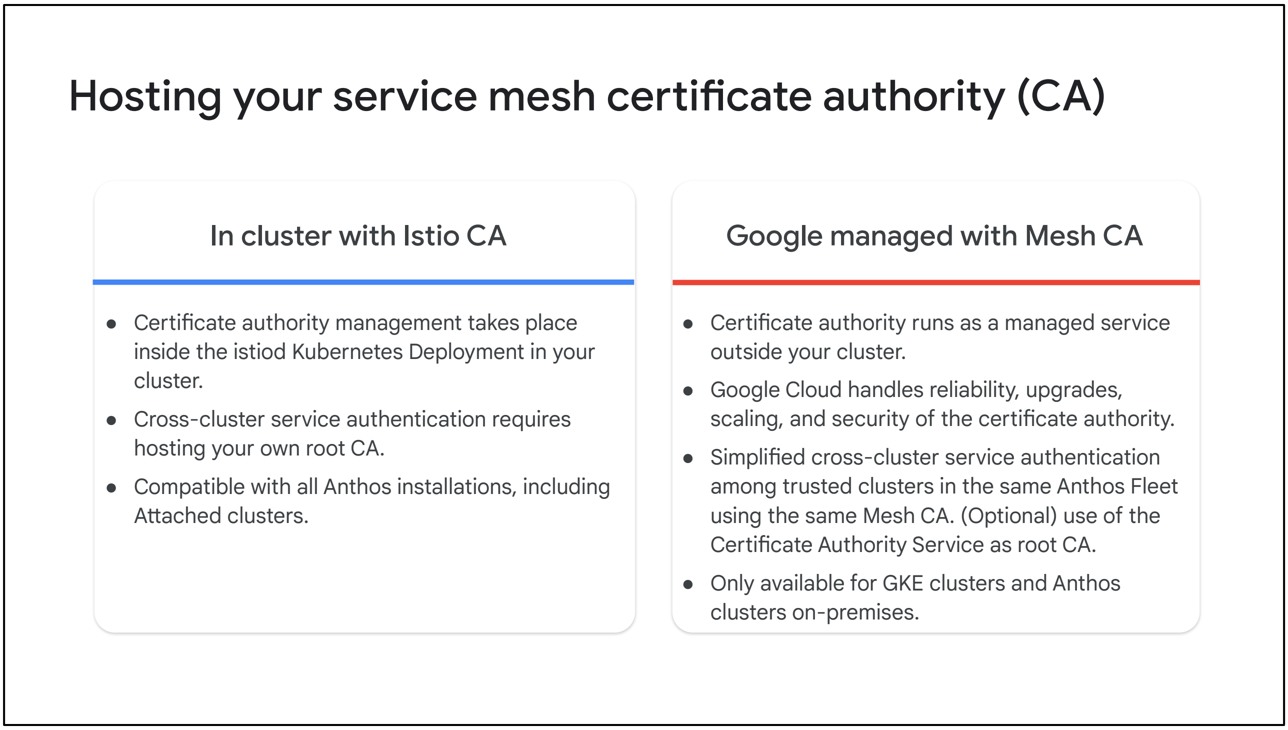
In cluster with Istio CA

- Certificate authority management takes place inside the istiod Kubernetes Deployment in your cluster.
- Cross-cluster service authentication requires hosting your own root CA.
- Compatible with all Anthos installations, including Attached clusters.

You also have two options for hosting your service mesh certificate authority, or CA. Remember that the certificate authority generates certificates to allow secure mTLS communication in the data plane.

You can use the in cluster installation with Istio CA. In this case:

- Certificate authority management takes place inside the istiod Kubernetes Deployment in your cluster.
- Cross-cluster service authentication requires hosting your own root CA.
- And Istio CA is compatible with all Anthos installations, including attached clusters.



Hosting your service mesh certificate authority (CA)

In cluster with Istio CA

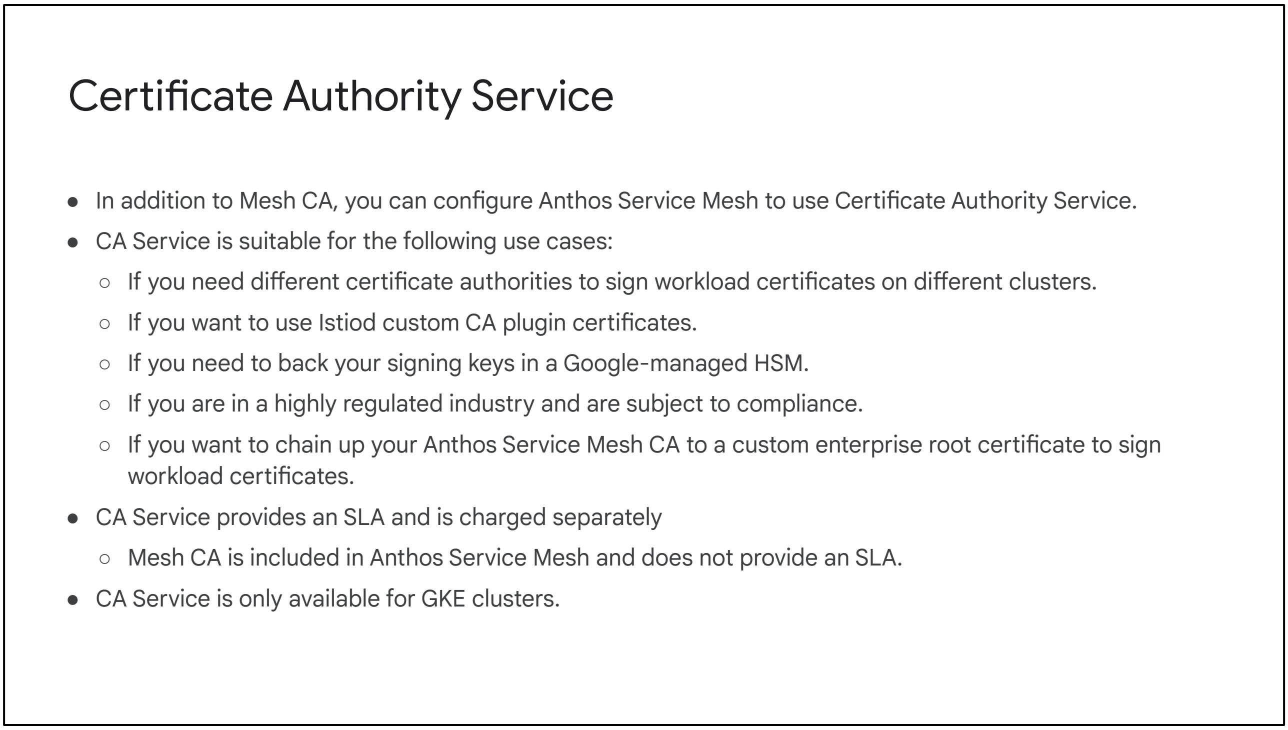
- Certificate authority management takes place inside the istiod Kubernetes Deployment in your cluster.
- Cross-cluster service authentication requires hosting your own root CA.
- Compatible with all Anthos installations, including Attached clusters.

Google managed with Mesh CA

- Certificate authority runs as a managed service outside your cluster.
- Google Cloud handles reliability, upgrades, scaling, and security of the certificate authority.
- Simplified cross-cluster service authentication among trusted clusters in the same Anthos Fleet using the same Mesh CA. (Optional) use of the Certificate Authority Service as root CA.
- Only available for GKE clusters and Anthos clusters on-premises.

Alternatively, you can use the Google managed Mesh CA. In this case:

- The certificate authority runs as a managed service outside of your cluster.
- Google Cloud handles reliability, upgrades, scaling, and security of the certificate authority.
- When running services across multiple clusters that are part of the same Anthos Fleet, you can use a single Mesh CA to simplify secure service-to-service communication. In addition, you can use Certificate Authority Service as a root CA.
- Mesh CA is only available for GKE clusters and Anthos clusters on-premises.



Certificate Authority Service

- In addition to Mesh CA, you can configure Anthos Service Mesh to use Certificate Authority Service.
- CA Service is suitable for the following use cases:
  - If you need different certificate authorities to sign workload certificates on different clusters.
  - If you want to use Istiod custom CA plugin certificates.
  - If you need to back your signing keys in a Google-managed HSM.
  - If you are in a highly regulated industry and are subject to compliance.
  - If you want to chain up your Anthos Service Mesh CA to a custom enterprise root certificate to sign workload certificates.
- CA Service provides an SLA and is charged separately
  - Mesh CA is included in Anthos Service Mesh and does not provide an SLA.
- CA Service is only available for GKE clusters.

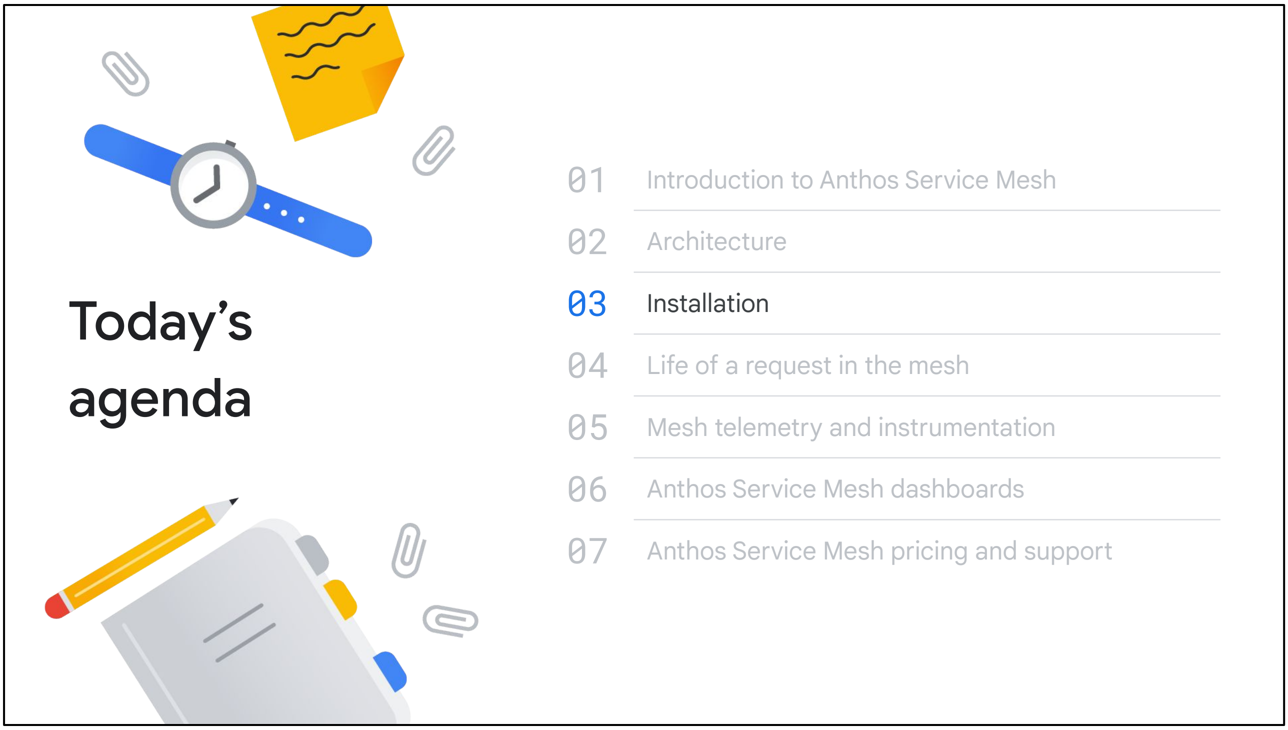
In addition to Mesh CA, you can configure Anthos Service Mesh to use Certificate Authority Service, or CA Service.

CA Service is suitable for the following use cases:

- If you need different certificate authorities to sign workload certificates on different clusters.
- If you want to use Istiod custom CA plugin certificates.
- If you need to back your signing keys in a Google-managed HSM (or Hardware Security Module.)
- If you are in a highly regulated industry and are subject to compliance.
- If you want to chain up your Anthos Service Mesh CA to a custom enterprise root certificate to sign workload certificates.

CA Service provides an SLA and is charged separately.

While Mesh CA does not provide an SLA, the service is included in the Anthos Service Mesh base price.



Today's agenda

01 Introduction to Anthos Service Mesh

02 Architecture

03 Installation

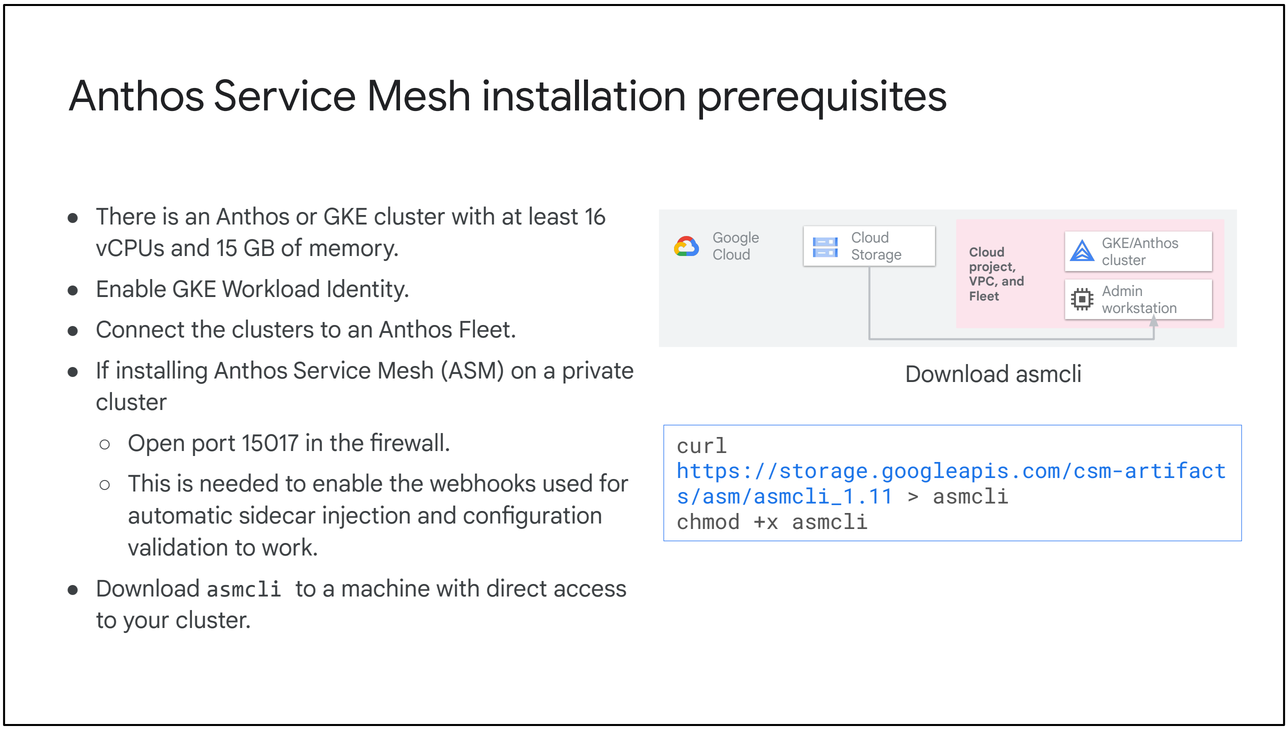
04 Life of a request in the mesh

05 Mesh telemetry and instrumentation

06 Anthos Service Mesh dashboards

07 Anthos Service Mesh pricing and support

Now that we have seen the main Anthos Service Mesh architectural components, let's see how to install them on an Anthos cluster.



Anthos Service Mesh installation prerequisites

- There is an Anthos or GKE cluster with at least 16 vCPUs and 15 GB of memory.
- Enable GKE Workload Identity.
- Connect the clusters to an Anthos Fleet.
- If installing Anthos Service Mesh (ASM) on a private cluster
  - Open port 15017 in the firewall.
  - This is needed to enable the webhooks used for automatic sidecar injection and configuration validation to work.
- Download `asmcli` to a machine with direct access to your cluster.

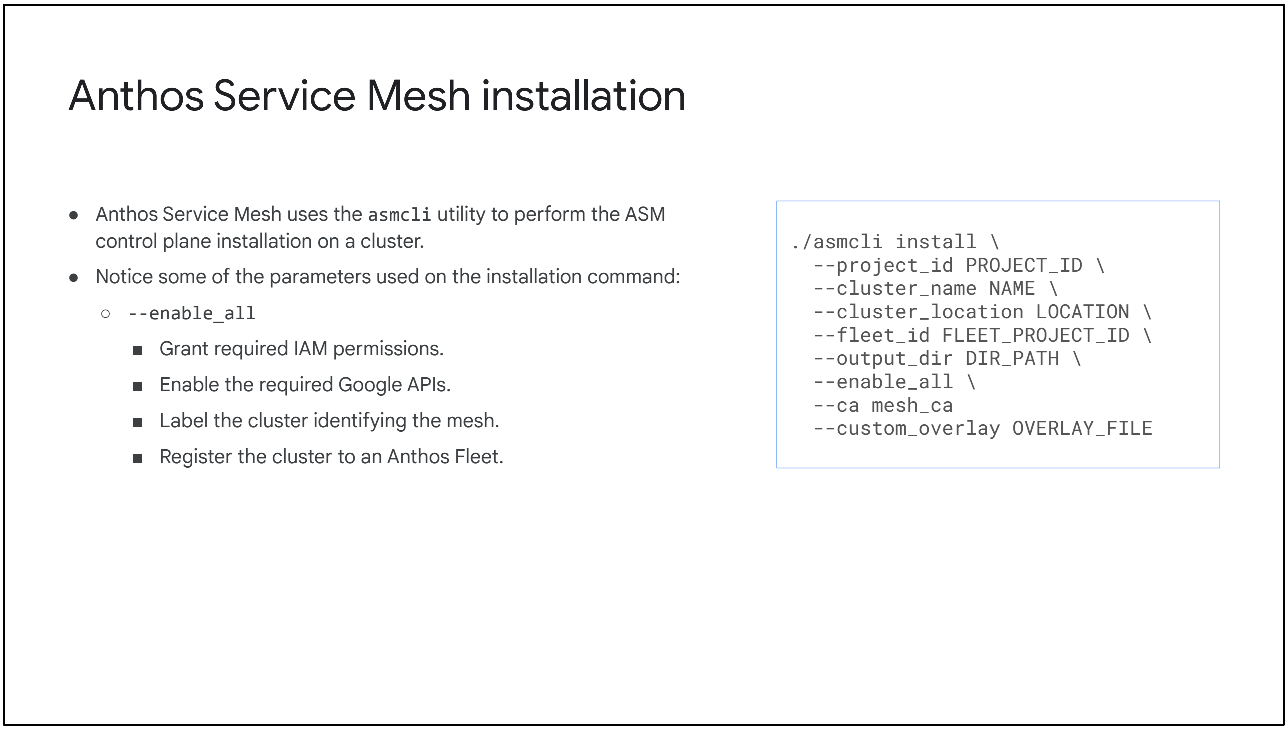
Download asmcli

```
curl
https://storage.googleapis.com/csm-artifacts/asm/asmcli_1.11 > asmcli
chmod +x asmcli
```

Before installing Anthos Service Mesh, you must fulfill certain requirements. You must have created a GKE or an Anthos cluster with at least 16 vCPUs and 15 GB of memory. Also, your clusters must have GKE Workload Identity enabled and be registered to an Anthos Fleet. When deploying Anthos clusters on VMware, bare metal, or AWS, the enablement of GKE workload Identity and the registration in an Anthos Fleet takes place automatically.

Additionally, if you want to install Anthos Service Mesh on a private cluster, you must open port 15017 in the firewall to get the webhooks used for automatic sidecar injection and configuration validation to work.

Once those prerequisites are fulfilled, download the asmcli utility from Cloud Storage into a machine with direct access to your cluster in order to perform the installation.



Anthos Service Mesh installation

- Anthos Service Mesh uses the `asmcli` utility to perform the ASM control plane installation on a cluster.
- Notice some of the parameters used on the installation command:
    - `--enable_all`
        - Grant required IAM permissions.
        - Enable the required Google APIs.
        - Label the cluster identifying the mesh.
        - Register the cluster to an Anthos Fleet.

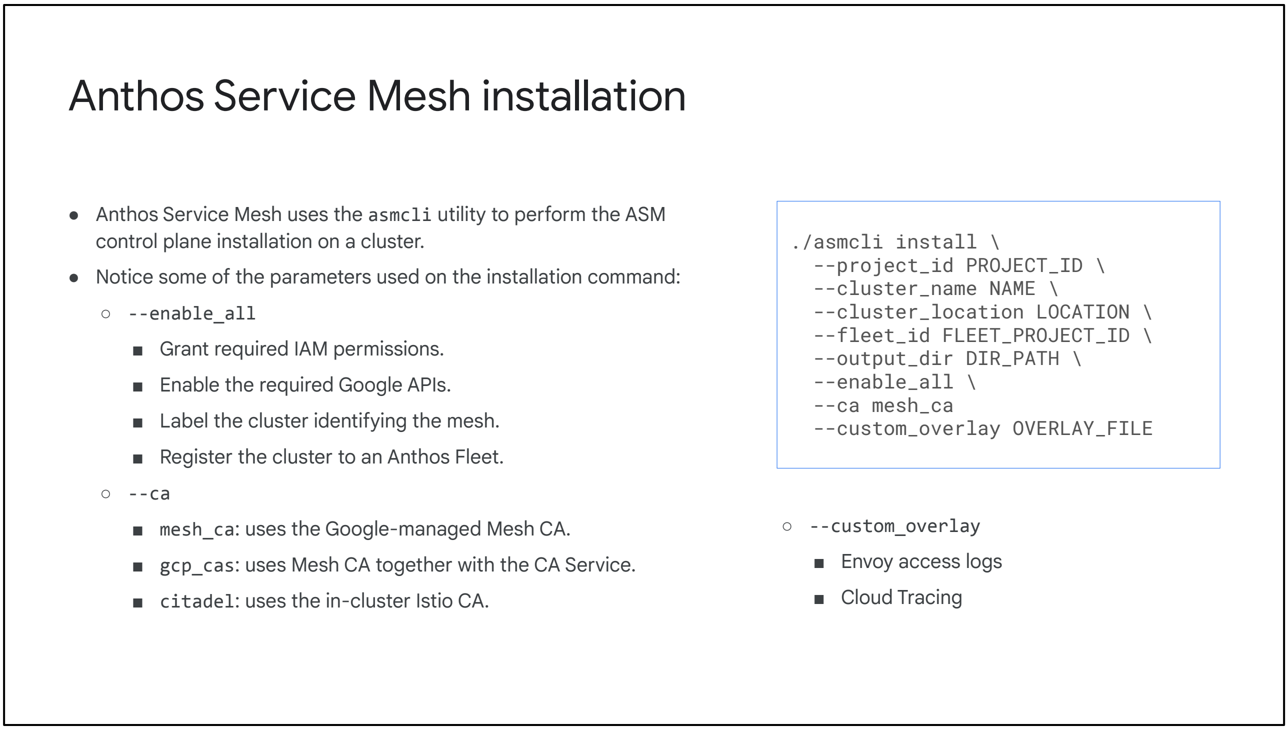
```
./asmcli install \
  --project_id PROJECT_ID \
  --cluster_name NAME \
  --cluster_location LOCATION \
  --fleet_id FLEET_PROJECT_ID \
  --output_dir DIR_PATH \
  --enable_all \
  --ca mesh_ca
  --custom_overlay OVERLAY_FILE
```

Anthos Service Mesh uses the asmcli utility to perform the ASM control plane installation on a cluster.

Let’s take a look at some of the flags available with this command.

Enable all, performs all steps required before installing Anthos Service Mesh. Those include:

- Granting required IAM permissions.
- Enabling the required Google APIs.
- Labeling the cluster identifying the mesh.
- And registering the cluster to an Anthos Fleet.



Anthos Service Mesh installation

- Anthos Service Mesh uses the `asmcli` utility to perform the ASM control plane installation on a cluster.
- Notice some of the parameters used on the installation command:
  - `--enable_all`
    - Grant required IAM permissions.
    - Enable the required Google APIs.
    - Label the cluster identifying the mesh.
    - Register the cluster to an Anthos Fleet.
  - `--ca`
    - `mesh_ca`: uses the Google-managed Mesh CA.
    - `gcp_cas`: uses Mesh CA together with the CA Service.
    - `citadel`: uses the in-cluster Istio CA.

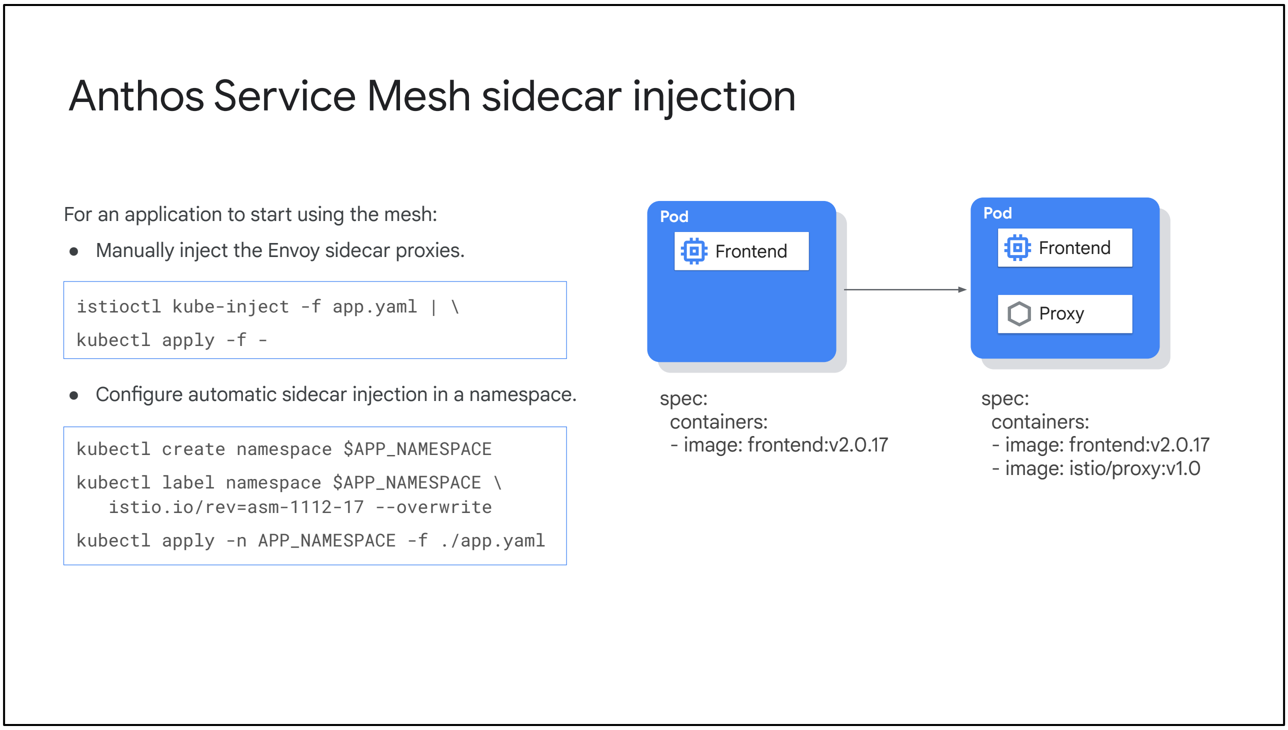
```
./asmcli install \
  --project_id PROJECT_ID \
  --cluster_name NAME \
  --cluster_location LOCATION \
  --fleet_id FLEET_PROJECT_ID \
  --output_dir DIR_PATH \
  --enable_all \
  --ca mesh_ca
  --custom_overlay OVERLAY_FILE
```

- `--custom_overlay`
  - Envoy access logs
  - Cloud Tracing

The CA flag, enables you to choose your certificate authority of choice. The options include:

- mesh_ca, which uses the Google-managed Mesh CA.
- gcp_cas, which uses Mesh CA together with the CA Service.
- And citadel, which uses the in cluster Istio CA.

Finally, you can add additional features by including a custom_overlay. For instance, you can configure the mesh to collect Envoy access logs in Cloud Logging and traces in Cloud Tracing.



Anthos Service Mesh sidecar injection

For an application to start using the mesh:

- Manually inject the Envoy sidecar proxies.

```
istioctl kube-inject -f app.yaml | \
kubectl apply -f -
```

- Configure automatic sidecar injection in a namespace.

```
kubectl create namespace $APP_NAMESPACE
kubectl label namespace $APP_NAMESPACE \
   istio.io/rev=asm-1112-17 --overwrite
kubectl apply -n APP_NAMESPACE -f ./app.yaml
```

```
spec:
 containers:
 - image: frontend:v2.0.17
```

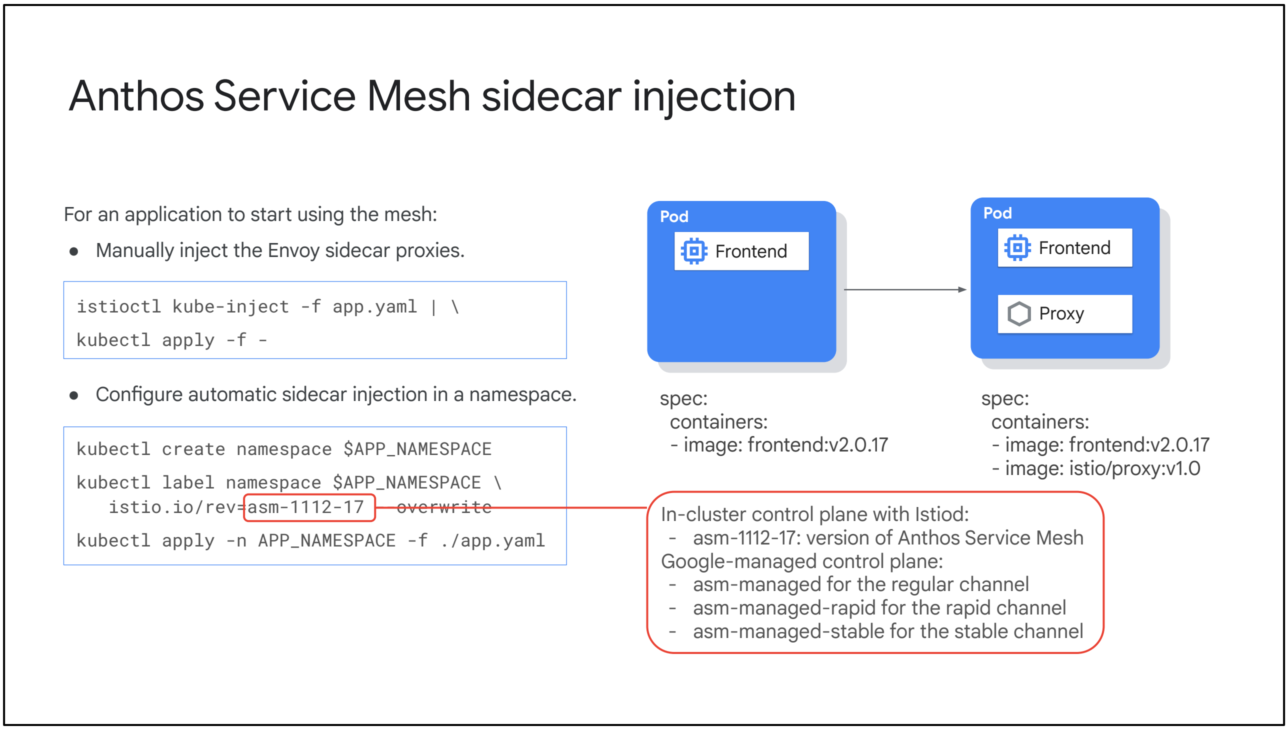
```
spec:
 containers:
 - image: frontend:v2.0.17
 - image: istio/proxy:v1.0
```

At this point, we have the mesh created but applications are not using it yet.

For an application to start using the mesh, we need to either:

- Manually inject the Envoy sidecar proxies with the “istioctl kube-inject” command. This command injects the sidecar proxy configuration in your application YAML and it applies the new configuration.
- Or configure automatic sidecar injection in a namespace by labeling the namespace with the same asm revision as the control plane. Sidecars are automatically added to your pods using a mutating webhook admission controller.

As soon as the sidecar proxies are running, they will intercept traffic, add mTLS when possible, and collect telemetry data which will be accessible from the Anthos Service Mesh console. No configuration is required.



Anthos Service Mesh sidecar injection

For an application to start using the mesh:

- Manually inject the Envoy sidecar proxies.

```
istioctl kube-inject -f app.yaml | \
kubectl apply -f -
```

- Configure automatic sidecar injection in a namespace.

```
kubectl create namespace $APP_NAMESPACE
kubectl label namespace $APP_NAMESPACE \
   istio.io/rev=asm-1112-17 --overwrite
kubectl apply -n APP_NAMESPACE -f ./app.yaml
```

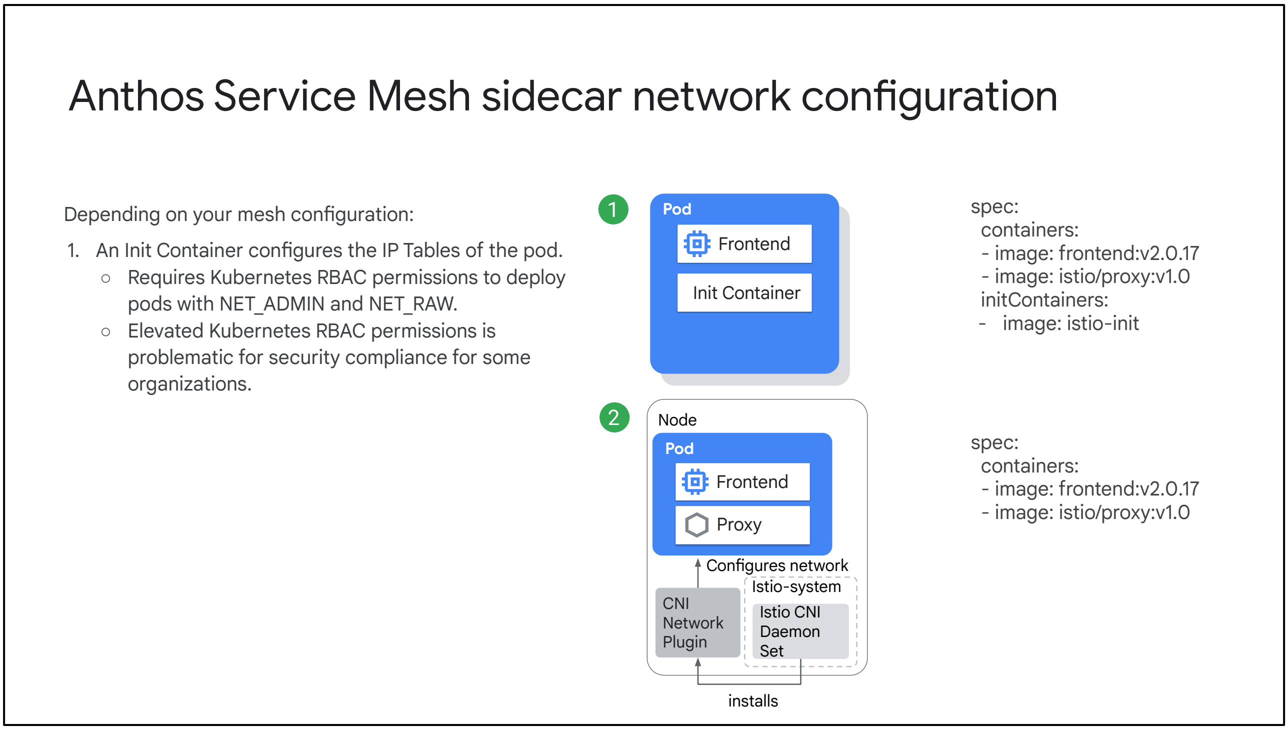
```
spec:
  containers:
  - image: frontend:v2.0.17
```

```
spec:
  containers:
  - image: frontend:v2.0.17
  - image: istio/proxy:v1.0
```

In-cluster control plane with Istiod:
- asm-1112-17: version of Anthos Service Mesh

Google-managed control plane:
- asm-managed for the regular channel
- asm-managed-rapid for the rapid channel
- asm-managed-stable for the stable channel

Notice that here we are using an in-cluster Istiod with ASM 11.12. If we were using the Google-managed control plane with managed upgrades, we could use the revision to specify the cadence at which Anthos Service Mesh gets upgraded. Use the revision asm-managed for the regular channel, asm-managed-rapid for the rapid channel, and asm-managed-stable for the stable channel. In open-source Istio, you would use a different label called “istio-injection” and set it to “enabled”.



Anthos Service Mesh sidecar network configuration

Depending on your mesh configuration:

1. An Init Container configures the IP Tables of the pod.
   - Requires Kubernetes RBAC permissions to deploy pods with NET_ADMIN and NET_RAW.
   - Elevated Kubernetes RBAC permissions is problematic for security compliance for some organizations.

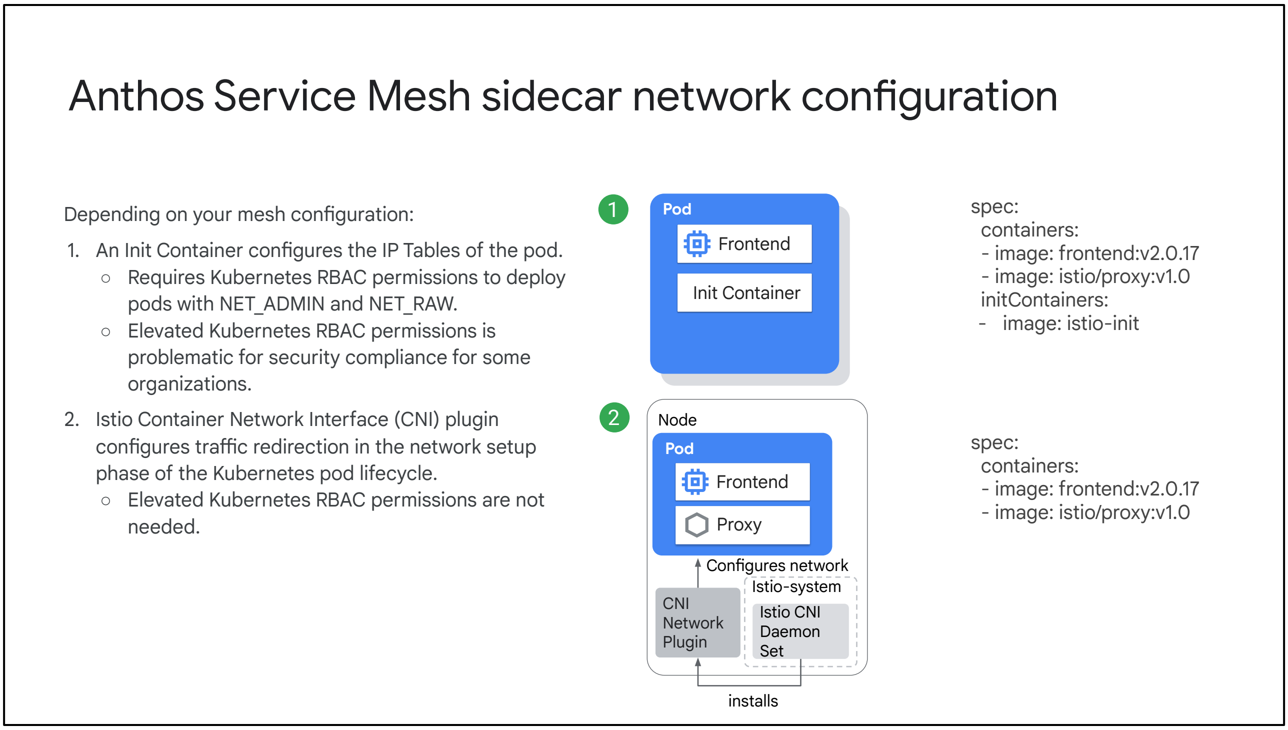
```
spec:
  containers:
  - image: frontend:v2.0.17
  - image: istio/proxy:v1.0
  initContainers:
  -   image: istio-init
```

```
spec:
  containers:
  - image: frontend:v2.0.17
  - image: istio/proxy:v1.0
```

You might still be wondering, how do the sidecars configure the pod's networking to intercept all requests?

There are two options depending on your mesh configuration:

- By default, an Init Container configures the IP Tables of the pod.
  - In this configuration, the user or service-account deploying pods to the mesh must have Kubernetes RBAC permissions to deploy pods with NET_ADMIN and NET_RAW.
  - Elevated Kubernetes RBAC permissions is problematic for some organizations' security compliance.



Anthos Service Mesh sidecar network configuration

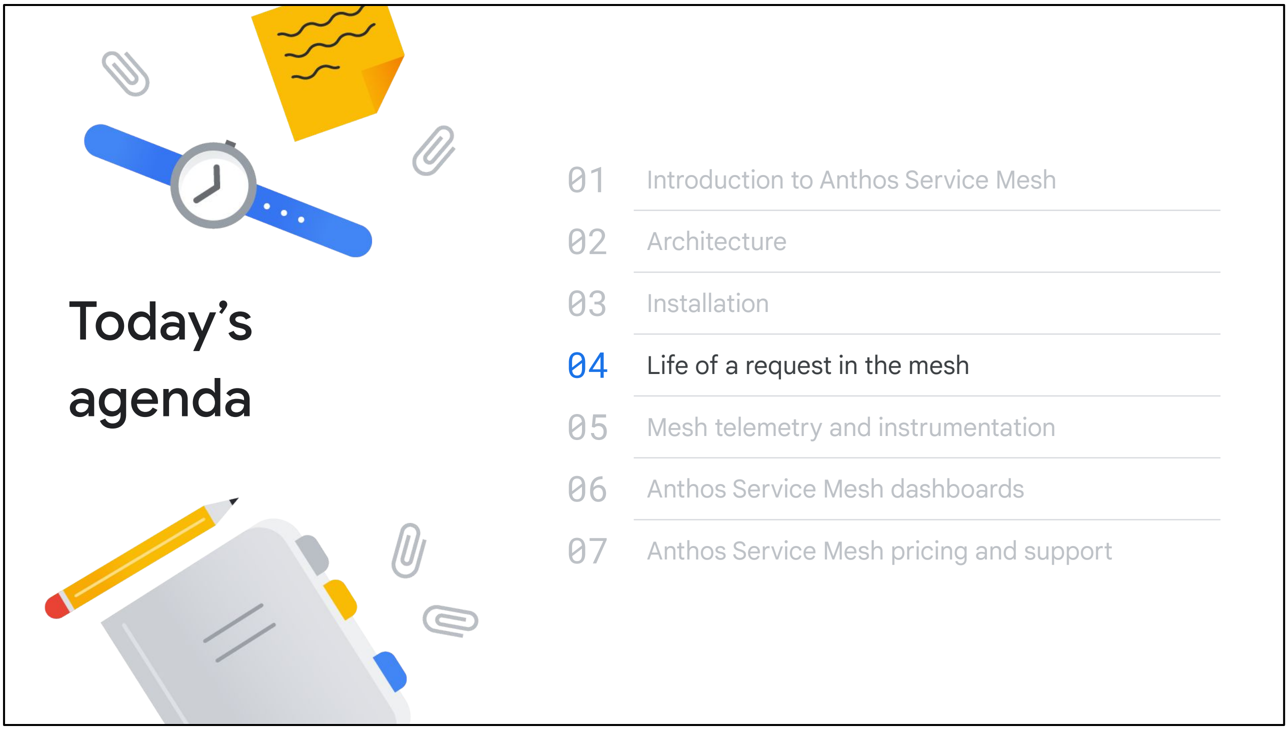
Depending on your mesh configuration:

1. An Init Container configures the IP Tables of the pod.
   - Requires Kubernetes RBAC permissions to deploy pods with NET_ADMIN and NET_RAW.
   - Elevated Kubernetes RBAC permissions is problematic for security compliance for some organizations.
2. Istio Container Network Interface (CNI) plugin configures traffic redirection in the network setup phase of the Kubernetes pod lifecycle.
   - Elevated Kubernetes RBAC permissions are not needed.

```
spec:
 containers:
 - image: frontend:v2.0.17
 - image: istio/proxy:v1.0
 initContainers:
 -   image: istio-init
```

```
spec:
 containers:
 - image: frontend:v2.0.17
 - image: istio/proxy:v1.0
```

- Alternatively, you can enable the Istio CNI plugin, which configures traffic redirection in the Kubernetes pod lifecycle's network setup phase.
  - This option does not require elevated Kubernetes RBAC permissions.



Today's agenda

01 Introduction to Anthos Service Mesh

02 Architecture

03 Installation

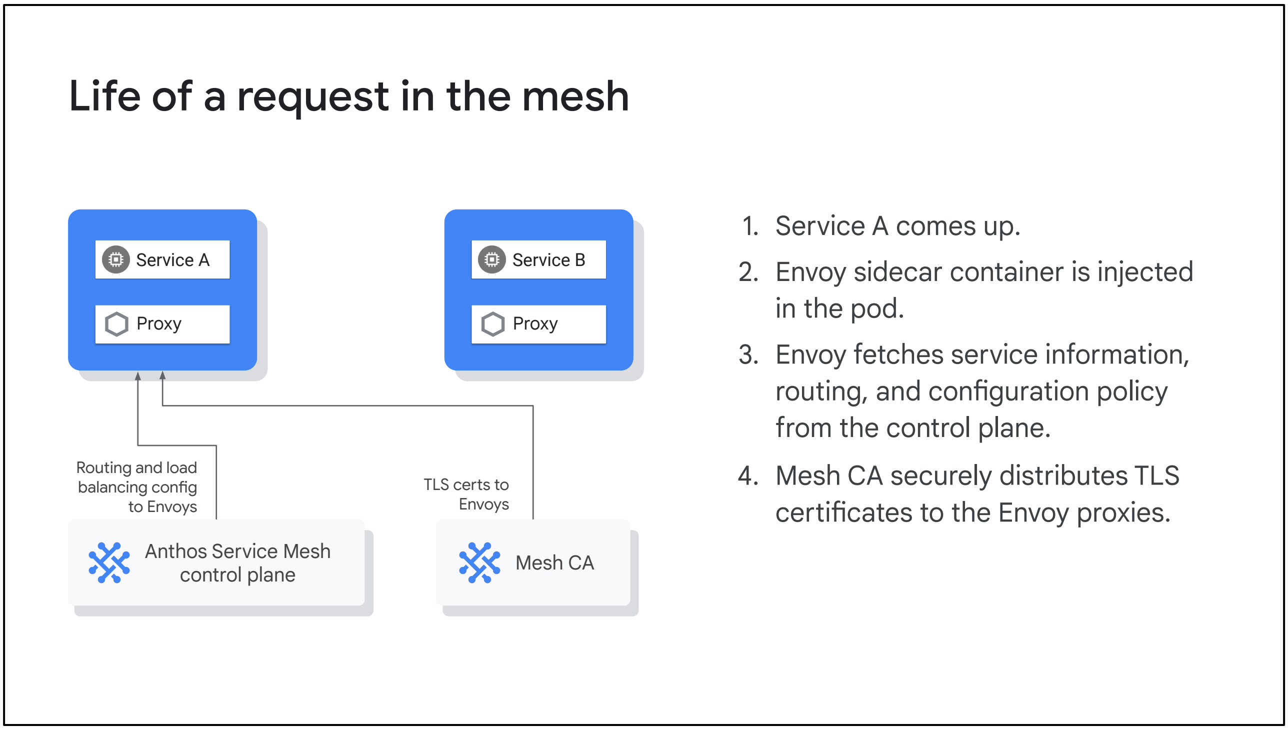
04 Life of a request in the mesh

05 Mesh telemetry and instrumentation

06 Anthos Service Mesh dashboards

07 Anthos Service Mesh pricing and support

Let's take a look a the life of a request inside a service mesh.



Life of a request in the mesh

Service A
Proxy
Service B
Proxy
Routing and load balancing config to Envoys
TLS certs to Envoys
Anthos Service Mesh control plane
Mesh CA

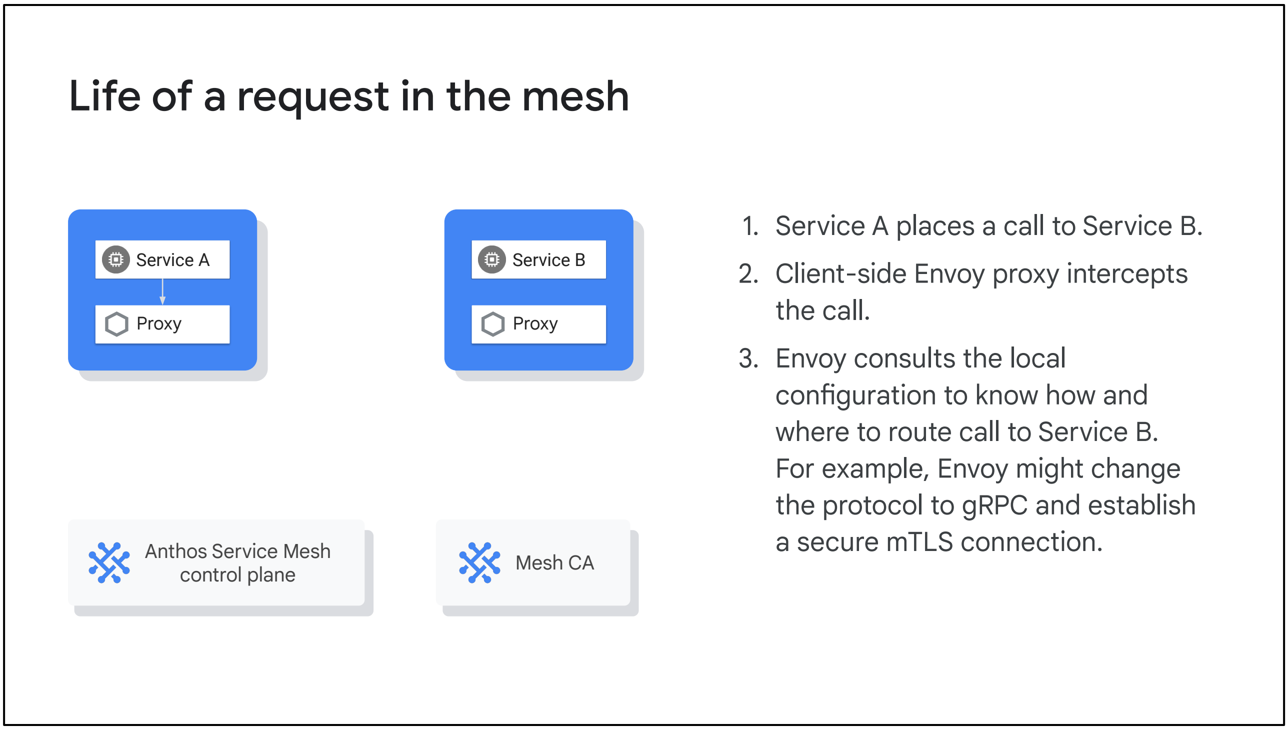
1. Service A comes up.
2. Envoy sidecar container is injected in the pod.
3. Envoy fetches service information, routing, and configuration policy from the control plane.
4. Mesh CA securely distributes TLS certificates to the Envoy proxies.

When a pod is created, your workload container comes up, which in this case is called “Service A”.

Then the Envoy sidecar container is injected in the pod and IP Tables are configured so that all requests are redirected to Envoy.

Envoy fetches service information, routing, and configuration policy from the control plane so that it knows where to send requests.

Finally, the service mesh certificate authority, in this case Mesh CA, securely distributes TLS certificates to the Envoy proxies.



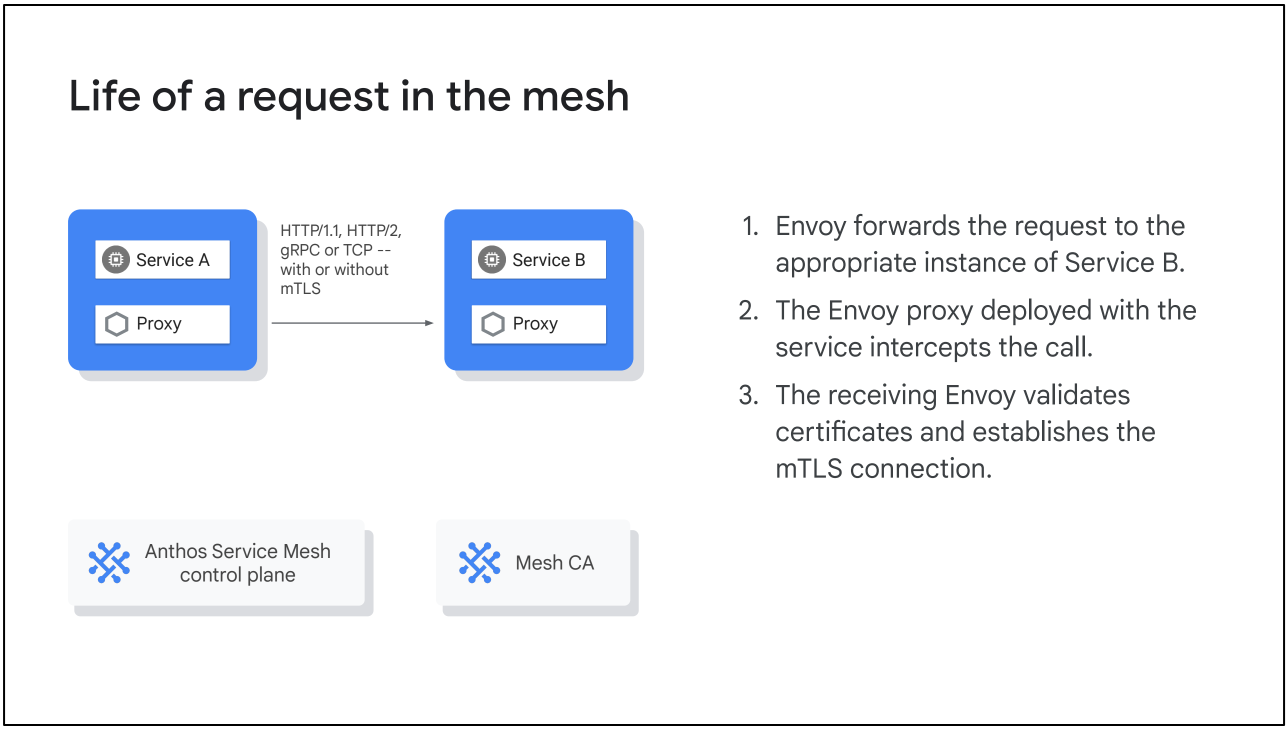
Life of a request in the mesh

1. Service A places a call to Service B.
2. Client-side Envoy proxy intercepts the call.
3. Envoy consults the local configuration to know how and where to route call to Service B. For example, Envoy might change the protocol to gRPC and establish a secure mTLS connection.

When Service A places a call to service B, the client-side Envoy proxy intercepts the call.

Envoy consults the local configuration to know how and where to route call to service B.

For instance, Envoy might change the protocol to gRPC and establish a secure mTLS connection.



Life of a request in the mesh

Service A

Proxy

HTTP/1.1, HTTP/2, gRPC or TCP -- with or without mTLS

Service B

Proxy

Anthos Service Mesh control plane

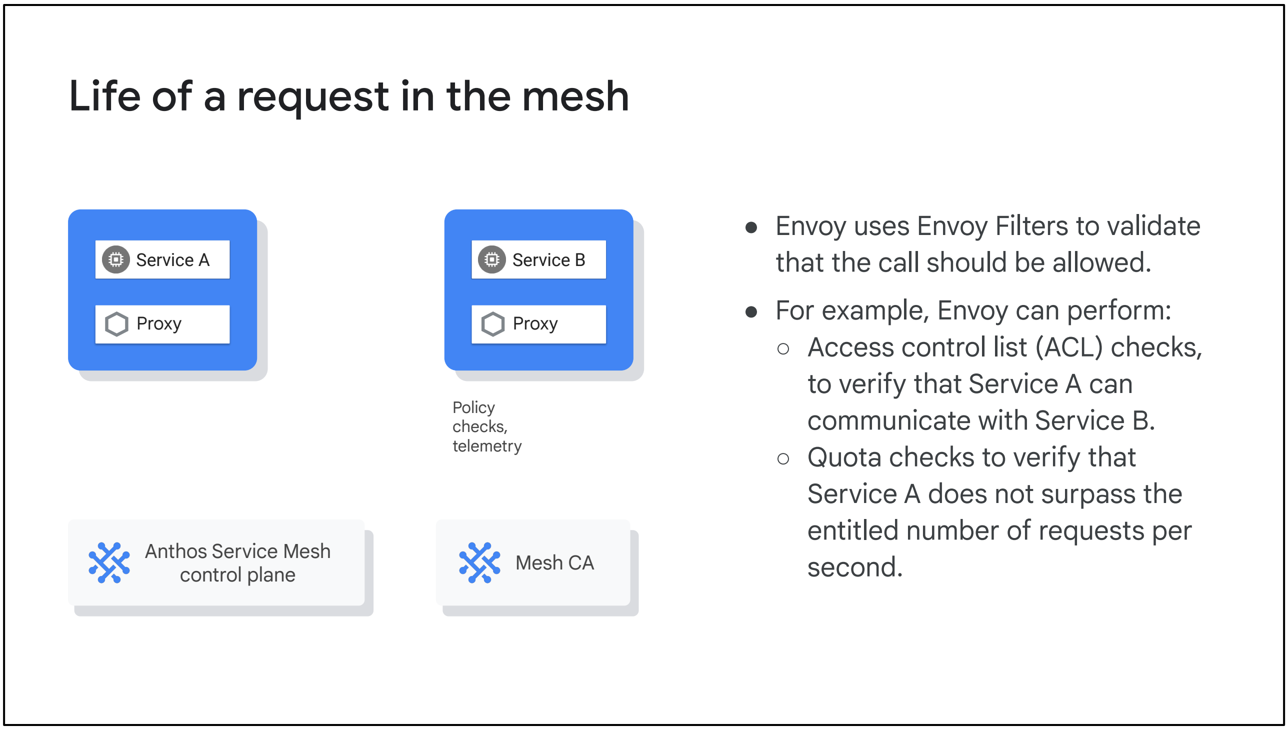
Mesh CA

1. Envoy forwards the request to the appropriate instance of Service B.
2. The Envoy proxy deployed with the service intercepts the call.
3. The receiving Envoy validates certificates and establishes the mTLS connection.

Then Envoy forwards the request to the appropriate instance of service B.

There, the Envoy proxy deployed with the service intercepts the call.

The receiving Envoy validates certificates and establishes the mTLS connection.



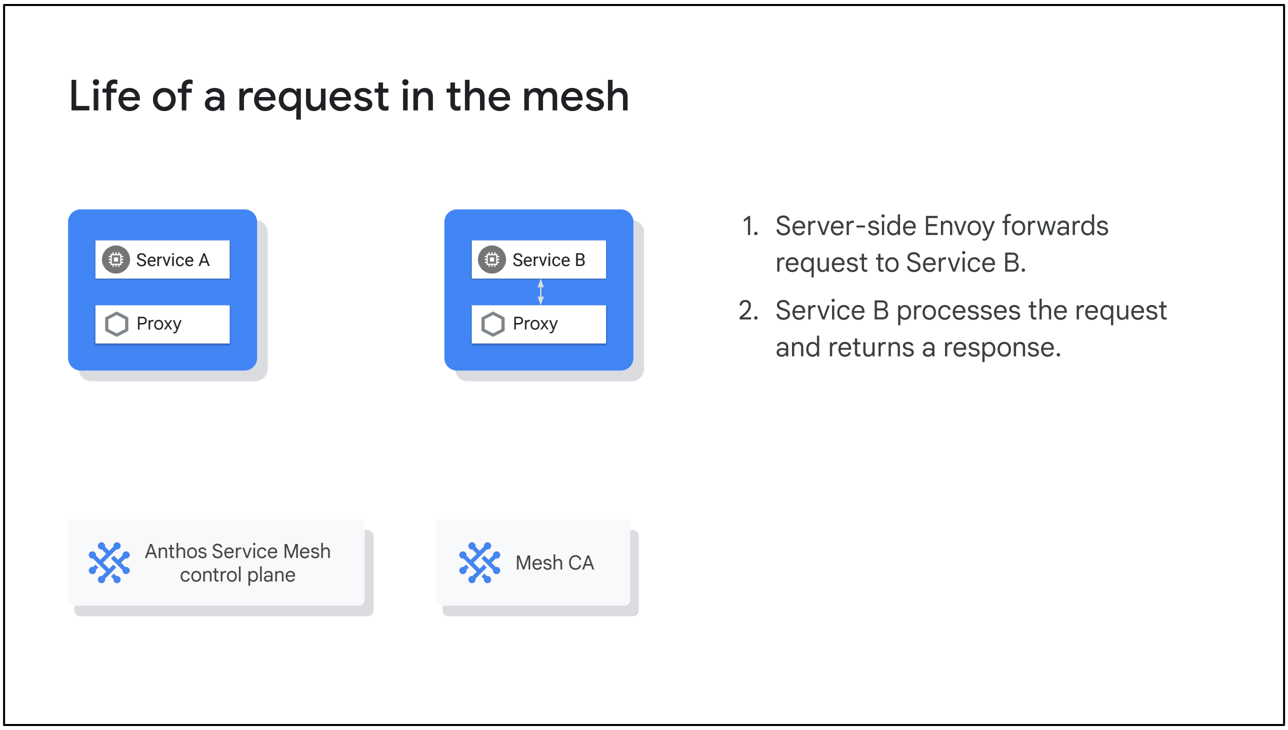
Life of a request in the mesh

- Envoy uses Envoy Filters to validate that the call should be allowed.
- For example, Envoy can perform:
  - Access control list (ACL) checks, to verify that Service A can communicate with Service B.
  - Quota checks to verify that Service A does not surpass the entitled number of requests per second.

The receiving Envoy checks the request using the Envoy Filters to validate that call should be allowed.

For instance, Envoy can perform:

- Access control list, or ACL, checks to verify service A can communicate with service B.
- Or quota checks to verify that service A does not surpass the entitled number of requests per second.



Life of a request in the mesh

Service A

Proxy

Service B

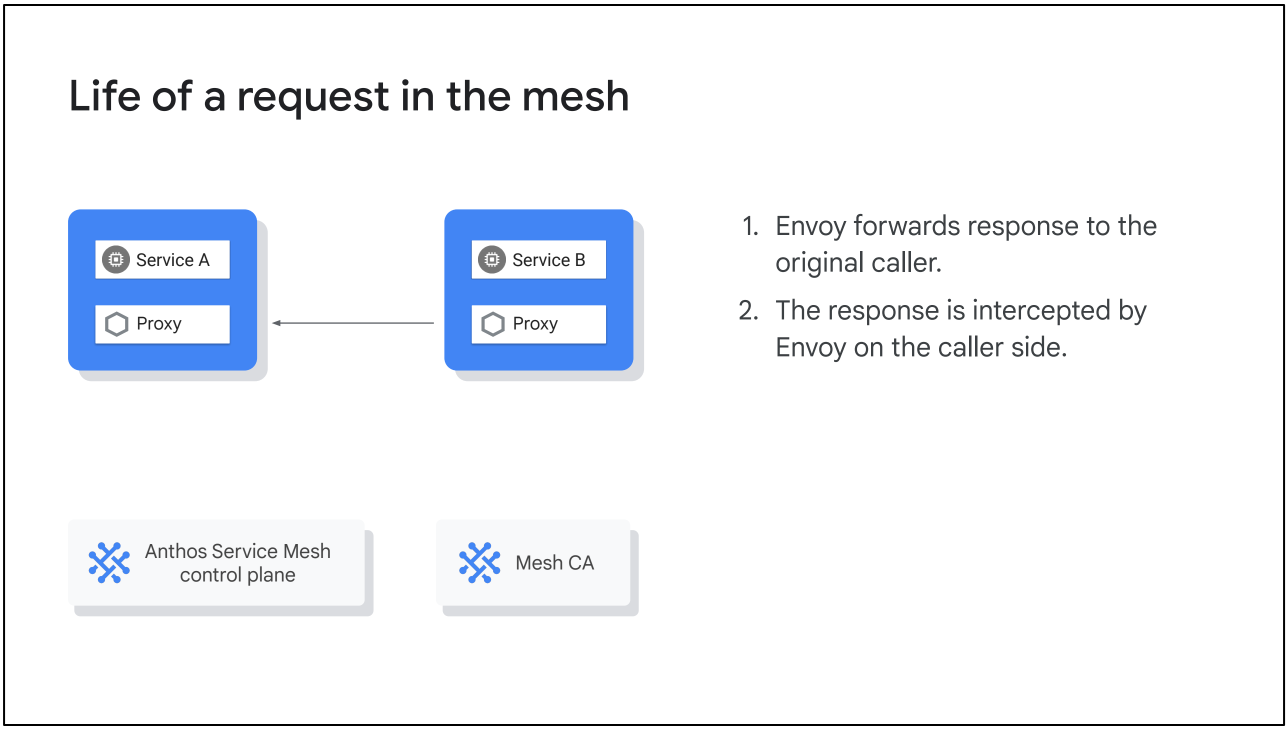
Proxy

Anthos Service Mesh control plane

Mesh CA

1. Server-side Envoy forwards request to Service B.
2. Service B processes the request and returns a response.

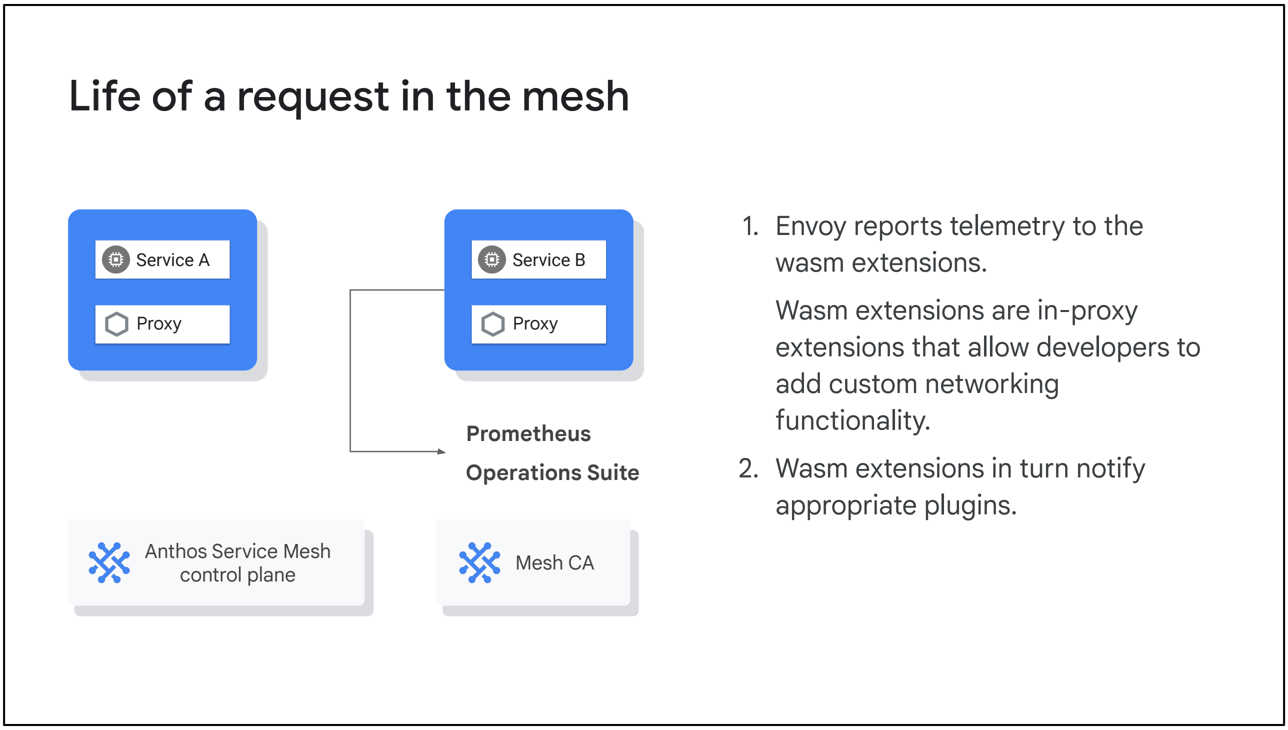
Once those validations take place, the server-side Envoy forwards request to service B, which processes the request and returns a response.



Life of a request in the mesh

1. Envoy forwards response to the original caller.
2. The response is intercepted by Envoy on the caller side.

Envoy forwards response to the original caller, where the response is then intercepted by Envoy on the caller side.



Life of a request in the mesh

Service A

Proxy

Service B

Proxy

Prometheus

Operations Suite

Anthos Service Mesh control plane

Mesh CA

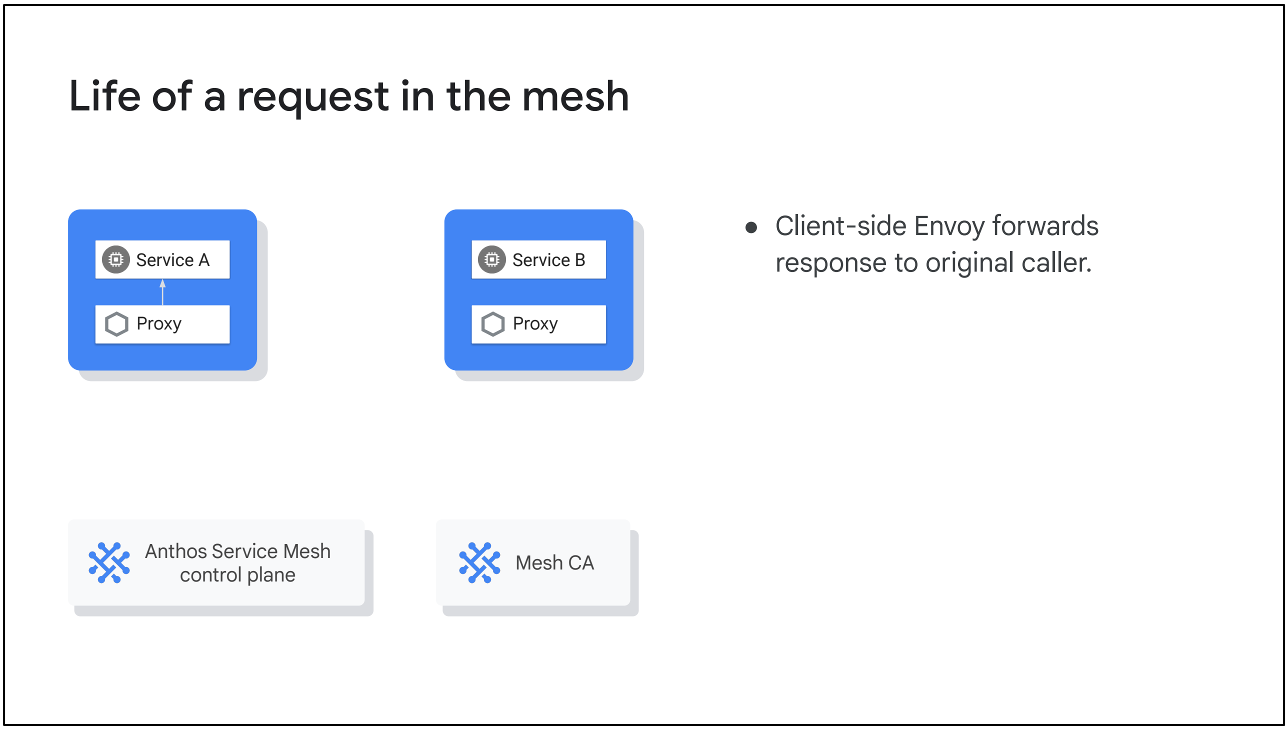
1. Envoy reports telemetry to the wasm extensions.

   Wasm extensions are in-proxy extensions that allow developers to add custom networking functionality.
2. Wasm extensions in turn notify appropriate plugins.

The server-side Envoy is done with the request and it reports telemetry to the wasm extensions.

Wasm extensions are in-proxy extensions that allow developers to add custom networking functionality.

Wasm extensions in turn notify appropriate plugins.

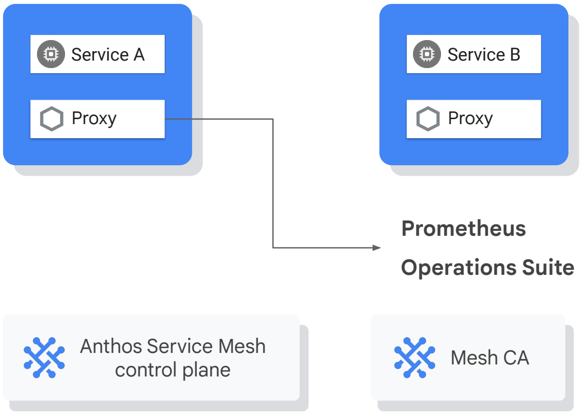


Life of a request in the mesh

- Client-side Envoy forwards response to original caller.

Client-side Envoy forwards response to original caller.

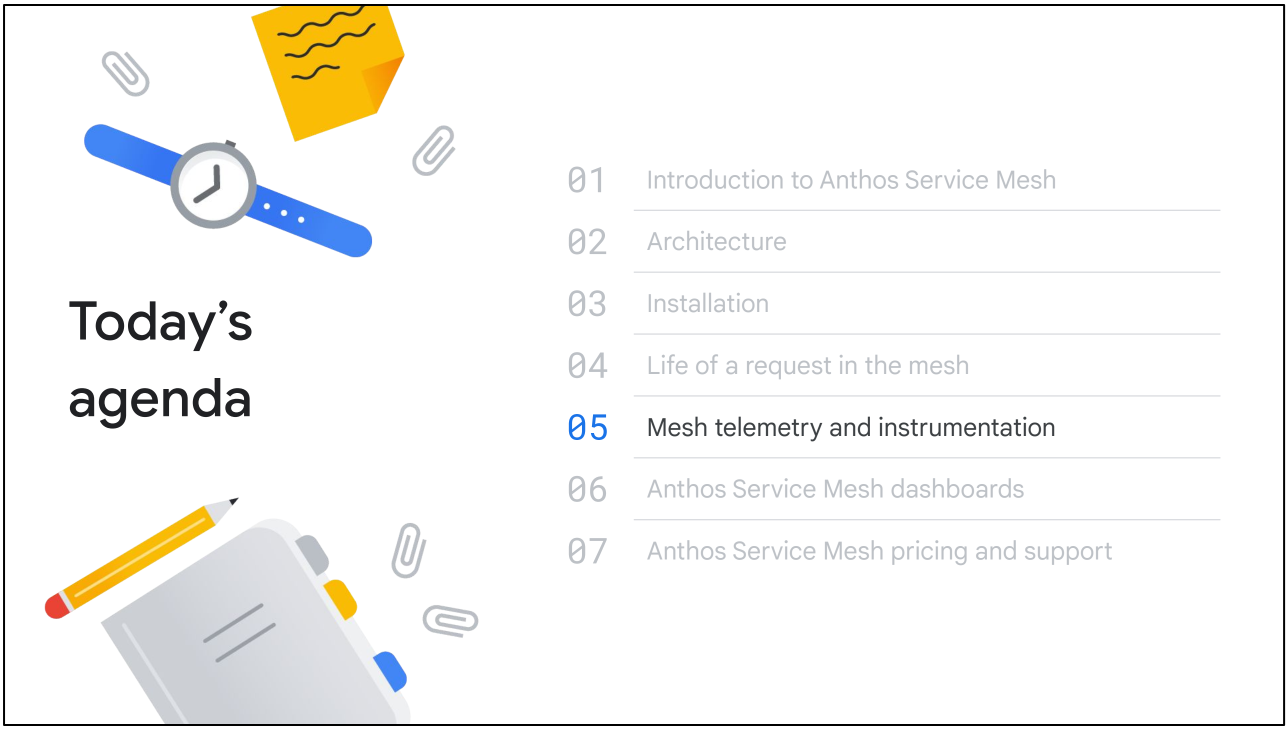
Life of a request in the mesh



1. Client-side Envoy reports telemetry to Envoy extensions, including client-perceived latency.
2. The Envoy extensions in turn notify the appropriate plugins.

Once the client-side Envoy has processed the request, it reports telemetry to Envoy extensions, including client-perceived latency.

The Envoy extensions in turn notify the appropriate plugins.



Today's agenda

01 Introduction to Anthos Service Mesh

02 Architecture

03 Installation

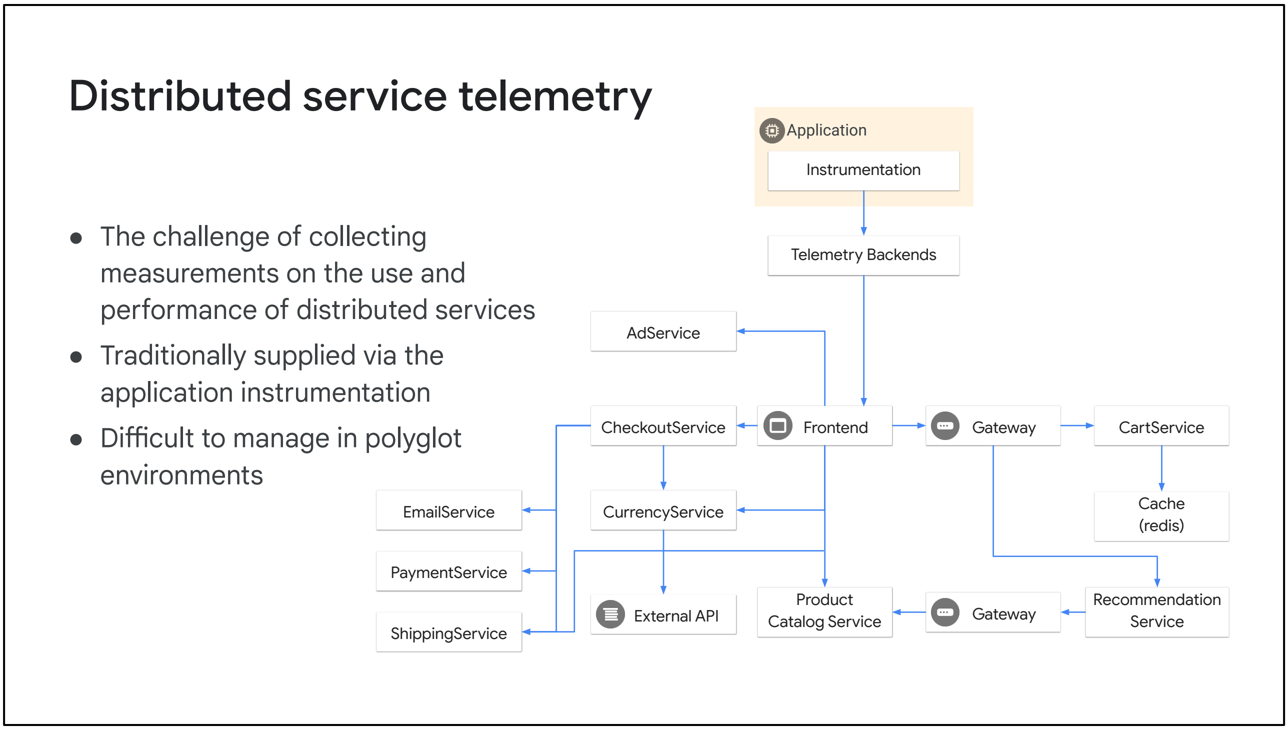
04 Life of a request in the mesh

05 Mesh telemetry and instrumentation

06 Anthos Service Mesh dashboards

07 Anthos Service Mesh pricing and support

Let's take a closer look at mesh telemetry and instrumentation.

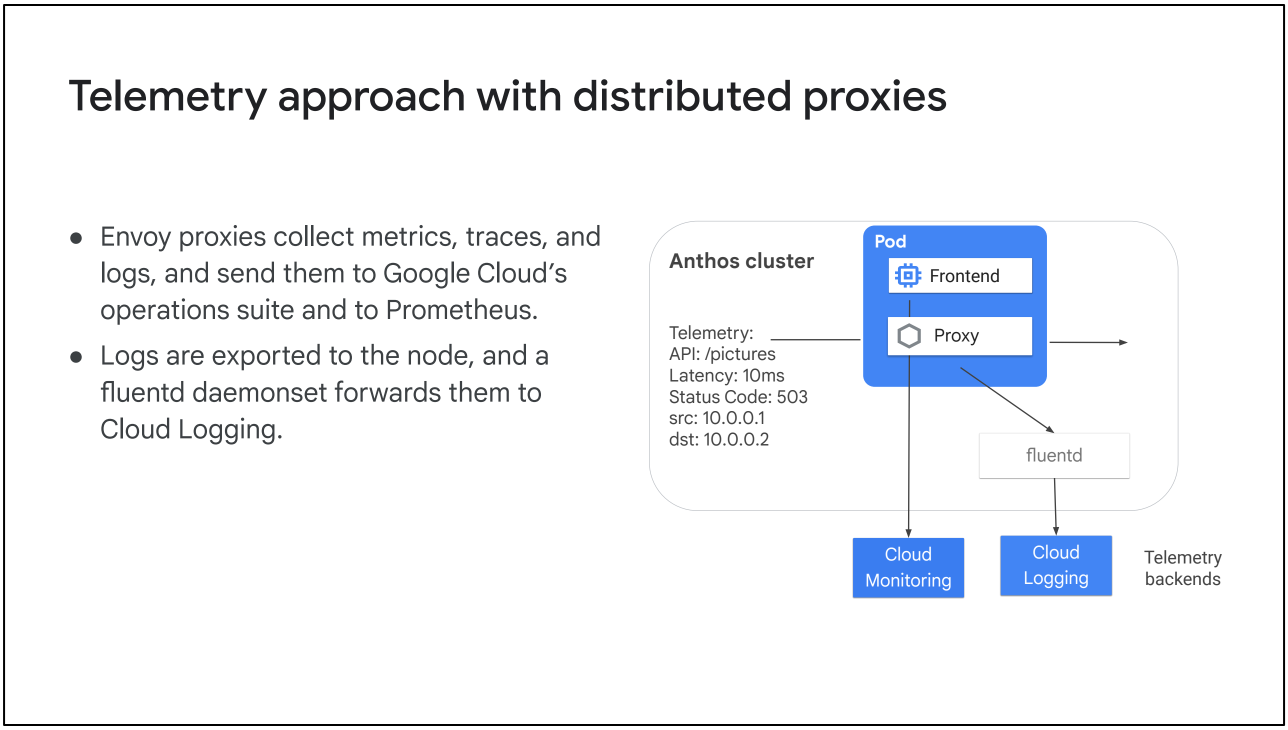


Distributed service telemetry

- The challenge of collecting measurements on the use and performance of distributed services
- Traditionally supplied via the application instrumentation
- Difficult to manage in polyglot environments

Traditionally, programmers have implemented instrumentation in the form of code instructions that monitor specific components in a system in order to monitor or measure the level of a product's performance, to diagnose errors, and to write trace information.

As containers are becoming the compute unit of choice, environments becomes more polyglot as more and more diverse programming languages are introduced, making this effort very hard to maintain and track; it hinders the philosophy of separation of duties between operators and developers.



Telemetry approach with distributed proxies

- Envoy proxies collect metrics, traces, and logs, and send them to Google Cloud’s operations suite and to Prometheus.
- Logs are exported to the node, and a fluentd daemonset forwards them to Cloud Logging.

Anthos cluster

Pod

Frontend

Proxy

Telemetry:
API: /pictures
Latency: 10ms
Status Code: 503
src: 10.0.0.1
dst: 10.0.0.2

fluentd

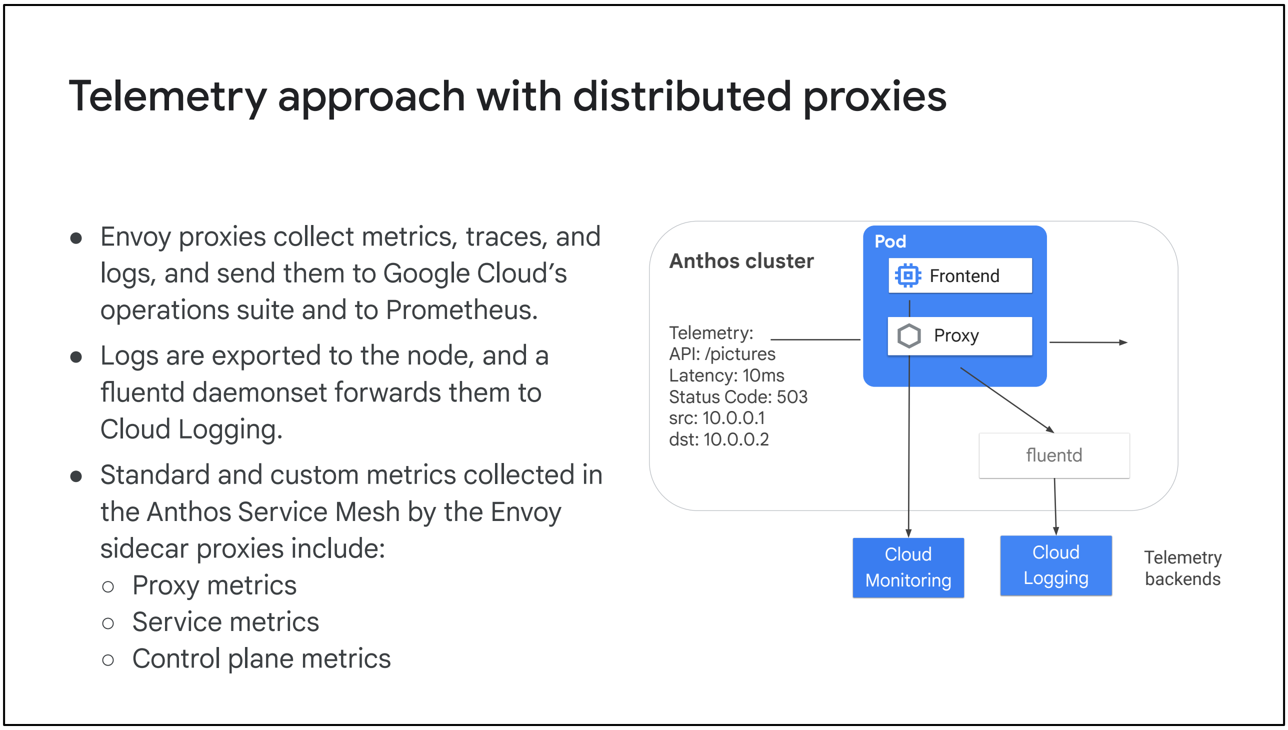
Cloud Monitoring

Cloud Logging

Telemetry backends

To obtain telemetry data, Anthos Service Mesh relies on the Envoy sidecar proxies that you inject in the workload pods. The proxies intercept all inbound and outbound HTTP traffic to the workloads and report the data to Anthos Service Mesh. They report data by calling the Cloud Monitoring, Cloud Logging, and Cloud Tracing APIs, as well as the Prometheus service in the cluster. With Anthos Service Mesh, service developers don't have to instrument their code to collect telemetry data.

Logs are exported to the node. A fluentd daemonset that comes with the Anthos cluster forwards the logs to Cloud Logging.

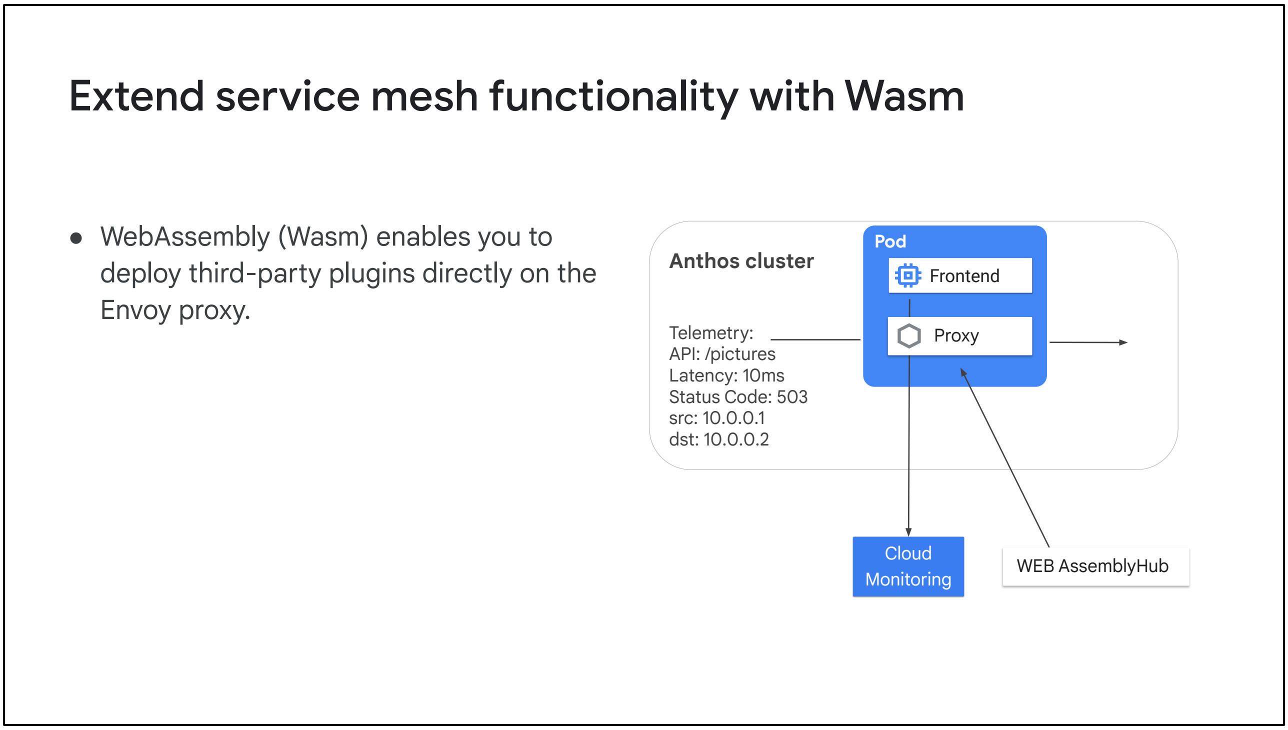


Telemetry approach with distributed proxies

- Envoy proxies collect metrics, traces, and logs, and send them to Google Cloud's operations suite and to Prometheus.
- Logs are exported to the node, and a fluentd daemonset forwards them to Cloud Logging.
- Standard and custom metrics collected in the Anthos Service Mesh by the Envoy sidecar proxies include:
  - Proxy metrics
  - Service metrics
  - Control plane metrics

Standard and custom metrics collected in the Anthos Service Mesh by the Envoy sidecar proxies include proxy, service, and control plane metrics.

You can configure custom metrics with Envoy Filters, an Istio CRD that gets applied directly to the Envoy Proxy. However, Envoy Filters are not supported in ASM 1.11 and, therefore, won't be covered in this course.



Extend service mesh functionality with Wasm

- WebAssembly (Wasm) enables you to deploy third-party plugins directly on the Envoy proxy.

Anthos cluster

Pod

Frontend

Proxy

Telemetry:
API: /pictures
Latency: 10ms
Status Code: 503
src: 10.0.0.1
dst: 10.0.0.2

Cloud Monitoring

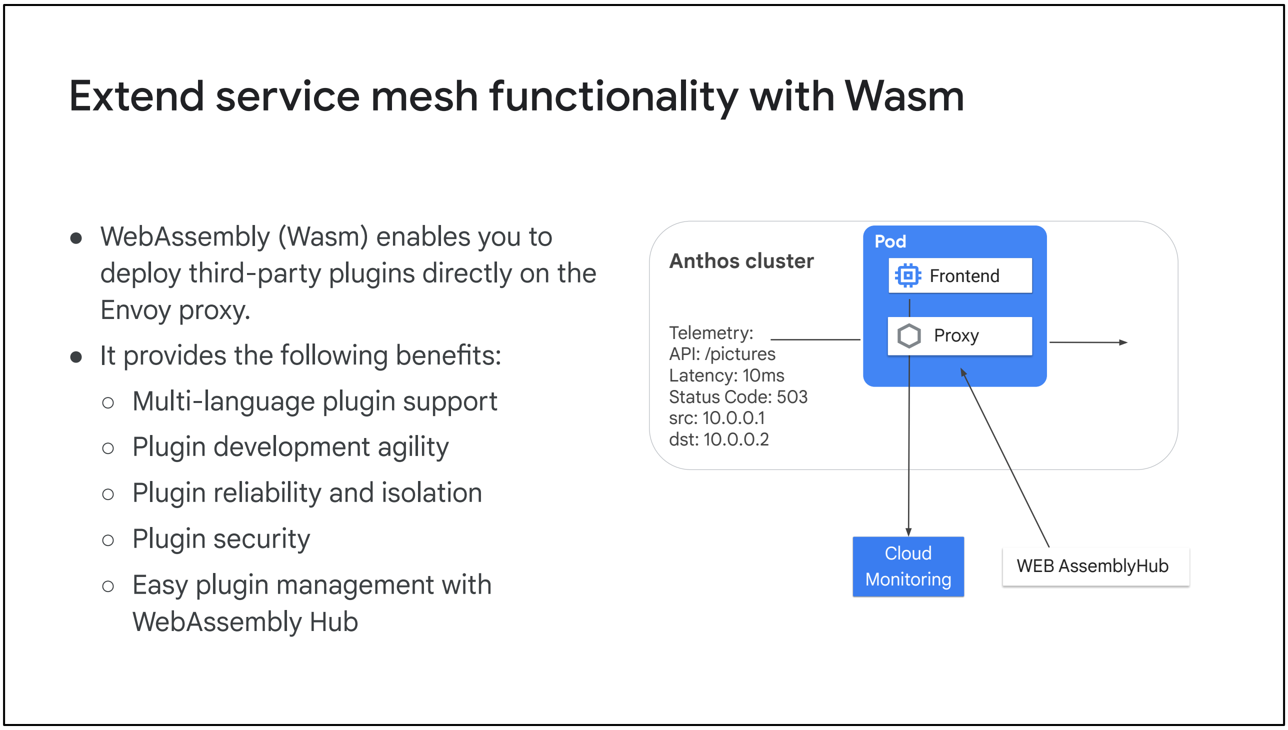
WEB AssemblyHub

You might be wondering, how does Envoy know how to communicate with Google Cloud's operations suite APIs?

Anthos Service Mesh installs a WebAssembly plugin in the Envoy proxy that contains the functionality to call the Cloud Monitoring and Cloud Tracing APIs.

WebAssembly (or wasm for short) is a portable bytecode format for executing code written in multiple languages at near-native speed built into Google's high performance V8 engine. After receiving a W3C recommendation in Dec 2019 to run natively in all major browsers, it is the fourth standard language (following HTML, CSS, and JavaScript).

Wasm enables you to deploy third-party telemetry filters that can redirect requests to your observability and policy backends.



Extend service mesh functionality with Wasm

- WebAssembly (Wasm) enables you to deploy third-party plugins directly on the Envoy proxy.
- It provides the following benefits:
  - Multi-language plugin support
  - Plugin development agility
  - Plugin reliability and isolation
  - Plugin security
  - Easy plugin management with WebAssembly Hub

Anthos cluster

Pod

Frontend

Proxy

Telemetry:
API: /pictures
Latency: 10ms
Status Code: 503
src: 10.0.0.1
dst: 10.0.0.2

Cloud Monitoring

WEB AssemblyHub

Benefits of Wasm include:

- Multi-language plugin support, such as C++, Rust, and AssemblyScript with more to come. Also, over 30 programming languages can be compiled to WebAssembly, allowing developers from all backgrounds to write Envoy extensions in their language of choice.

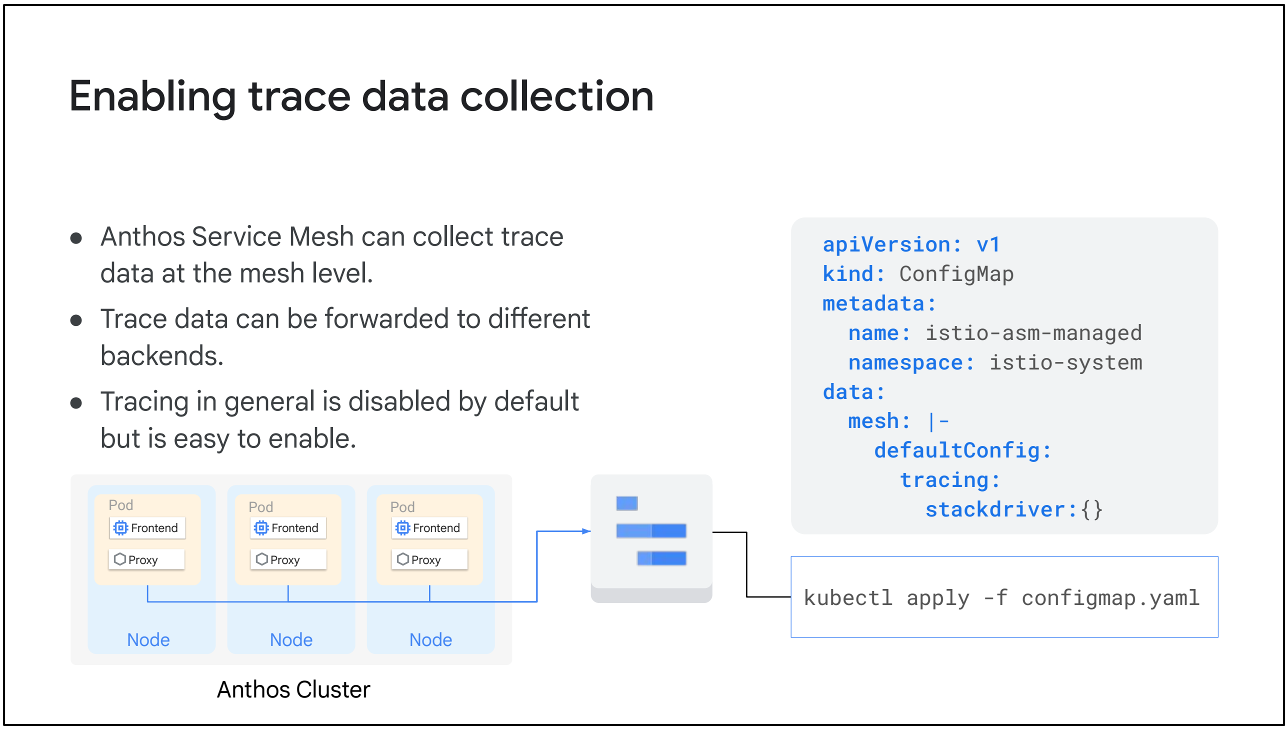
- Plugin development agility. Extensions can be delivered and reloaded at runtime using the Istio control plane. This enables a fast develop → test → release cycle for extensions without requiring Envoy rollouts.

- Reliability and isolation. Extensions are deployed inside a sandbox with resource constraints, which means they can now crash, or leak memory, without bringing the whole Envoy process down. CPU and memory usage can also be constrained.

- Security. The sandbox has a clearly defined API for communicating with Envoy, so extensions only have access to, and can modify, a limited number of properties of a connection or request. Furthermore, because Envoy mediates this interaction, it can hide or sanitize sensitive information from the extension (for example, “Authorization” and “Cookie” HTTP headers, or the client’s IP address).

- Easily create and deploy modules to your service mesh, and publish and share them to WebAssembly Hub as a repository. (This is similar to how Helm charts are deployed to the stable.)

Like Docker, the WebAssembly Hub stores and distributes Wasm extensions as OCI images. This makes pushing, pulling, and running extensions as easy as Docker containers. Wasm extension images are versioned and cryptographically secure, making it safe to run extensions locally the same way you would in production. This allows you to build and push, as well as trust the source when they pull down and deploy images.



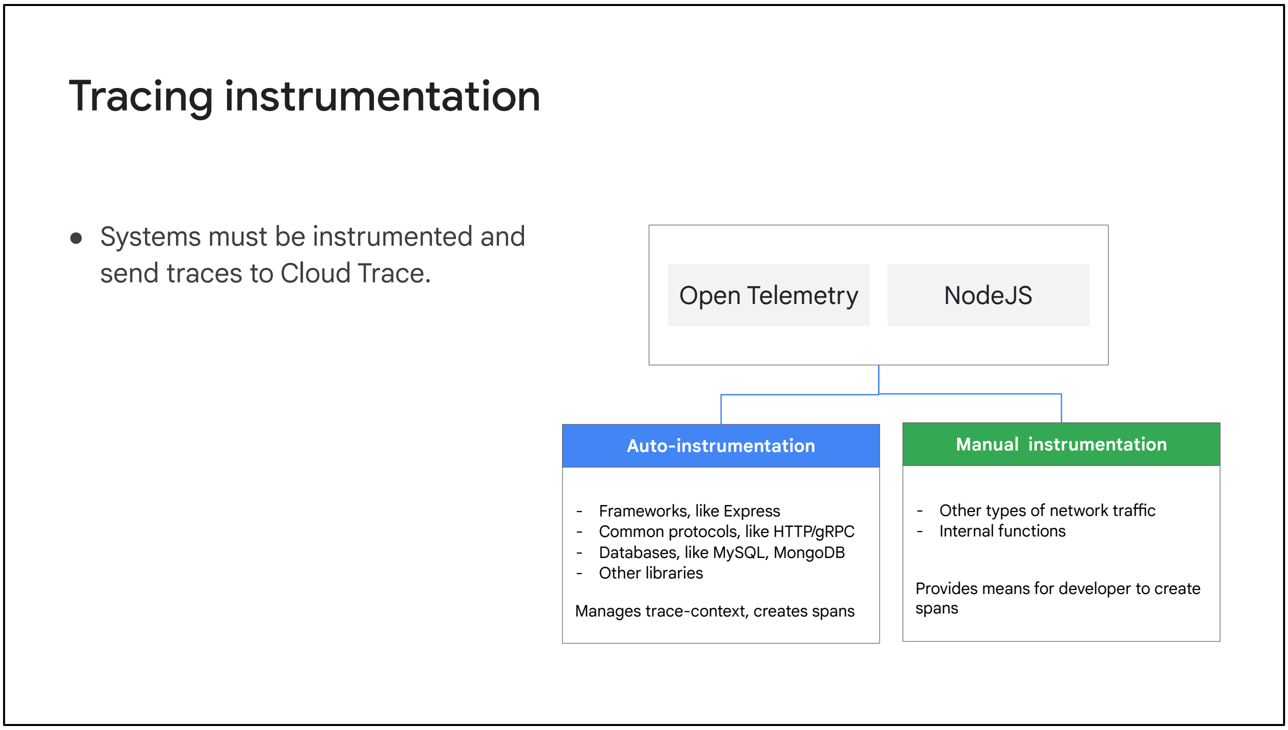
Enabling trace data collection

- Anthos Service Mesh can collect trace data at the mesh level.
- Trace data can be forwarded to different backends.
- Tracing in general is disabled by default but is easy to enable.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: istio-asm-managed
  namespace: istio-system
data:
  mesh: |-
    defaultConfig:
      tracing:
        stackdriver:{}
```

```
kubectl apply -f configmap.yaml
```

While logging and monitoring are enabled by default, you have to enable tracing yourself. To enable tracing, configure an overlay file for In-cluster ASM or apply a Kubernetes ConfigMap with tracing set to stackdriver as a backend for Managed ASM.



Tracing instrumentation

- Systems must be instrumented and send traces to Cloud Trace.

Open Telemetry

NodeJS

Auto-instrumentation

- Frameworks, like Express
- Common protocols, like HTTP/gRPC
- Databases, like MySQL, MongoDB
- Other libraries

Manages trace-context, creates spans

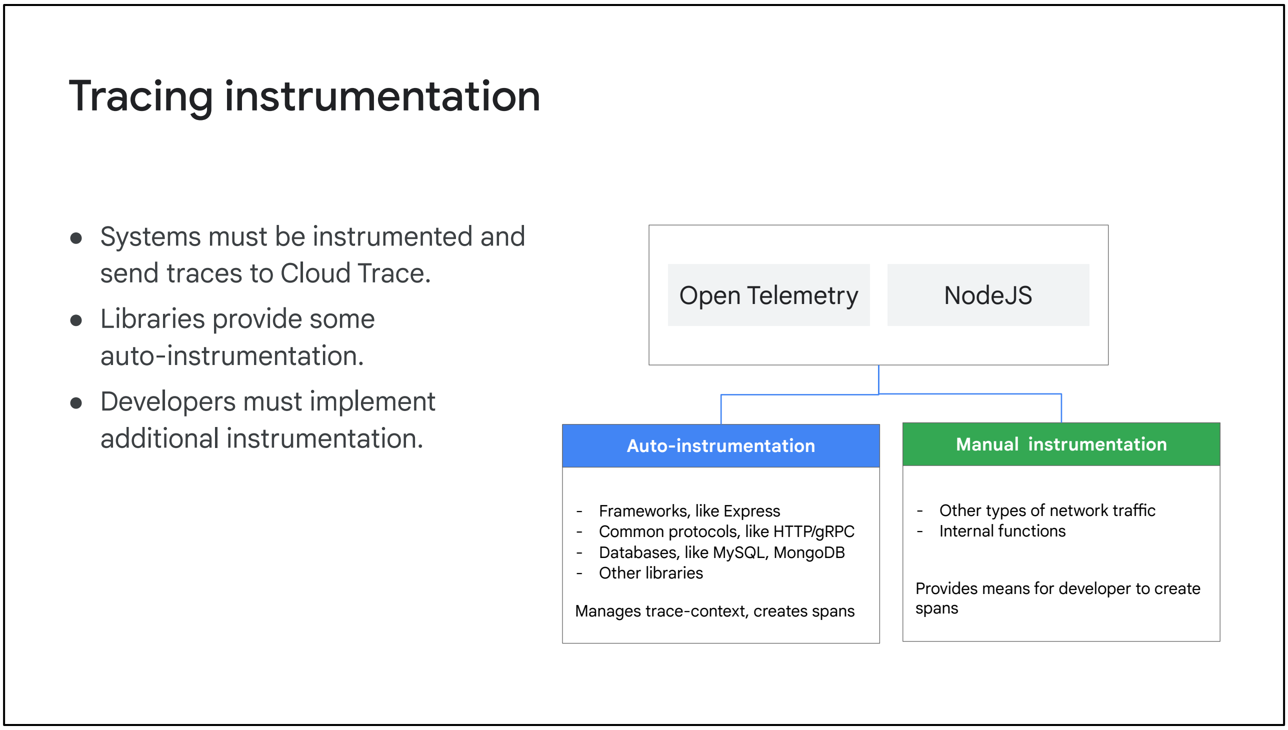
Manual instrumentation

- Other types of network traffic
- Internal functions

Provides means for developer to create spans

When using a Service Mesh, some individual traces are collected automatically, so that you can see the service topography or understand the latency per service.

However, if you want to understand an end-to-end trace that takes places across multiple microservices, you need to instrument your application.



Tracing instrumentation

- Systems must be instrumented and send traces to Cloud Trace.
- Libraries provide some auto-instrumentation.
- Developers must implement additional instrumentation.

Open Telemetry

NodeJS

Auto-instrumentation

- Frameworks, like Express
- Common protocols, like HTTP/gRPC
- Databases, like MySQL, MongoDB
- Other libraries

Manages trace-context, creates spans

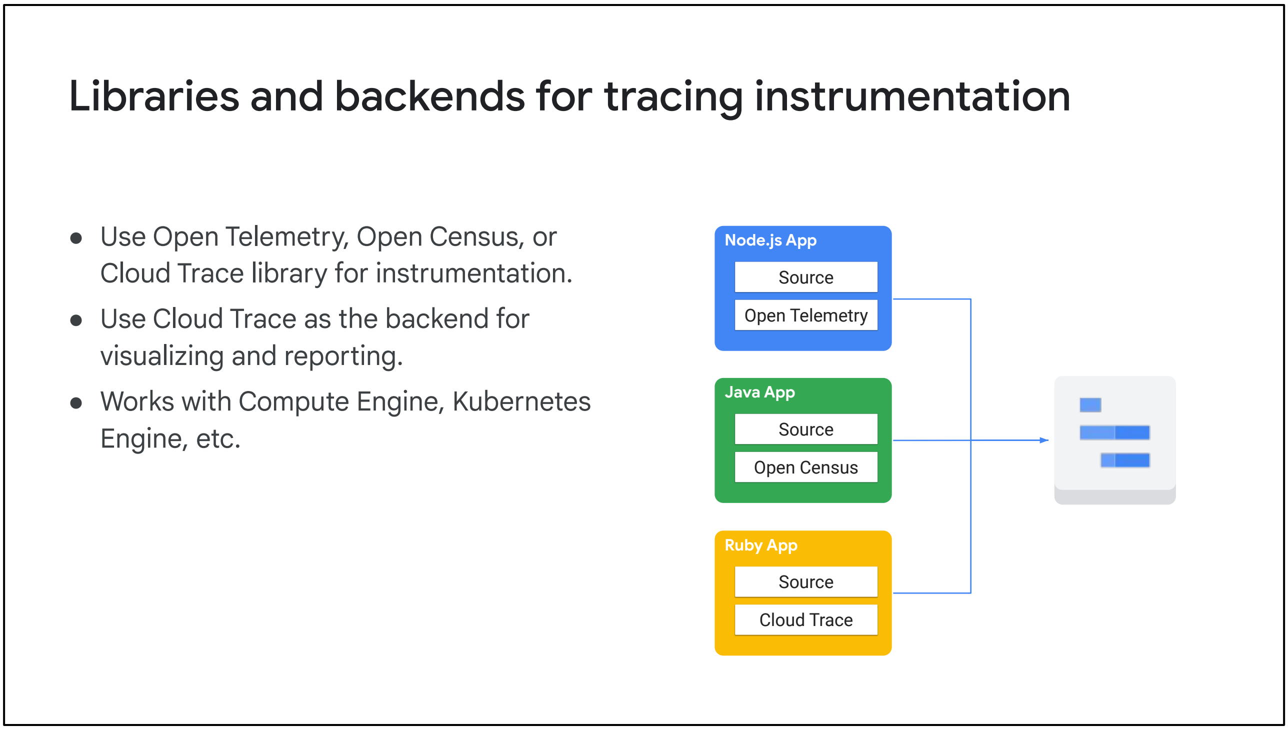
Manual instrumentation

- Other types of network traffic
- Internal functions

Provides means for developer to create spans

In the instrumentation process, you use libraries in your code to add unique headers to your requests, so that you can provide a trace of the request.

There are some frameworks, protocols, and databases that provide automatic instrumentation. However, most of the time, you will need to provide additional instrumentation.



Libraries and backends for tracing instrumentation

- Use Open Telemetry, Open Census, or Cloud Trace library for instrumentation.
- Use Cloud Trace as the backend for visualizing and reporting.
- Works with Compute Engine, Kubernetes Engine, etc.

Node.js App
Source
Open Telemetry

Java App
Source
Open Census

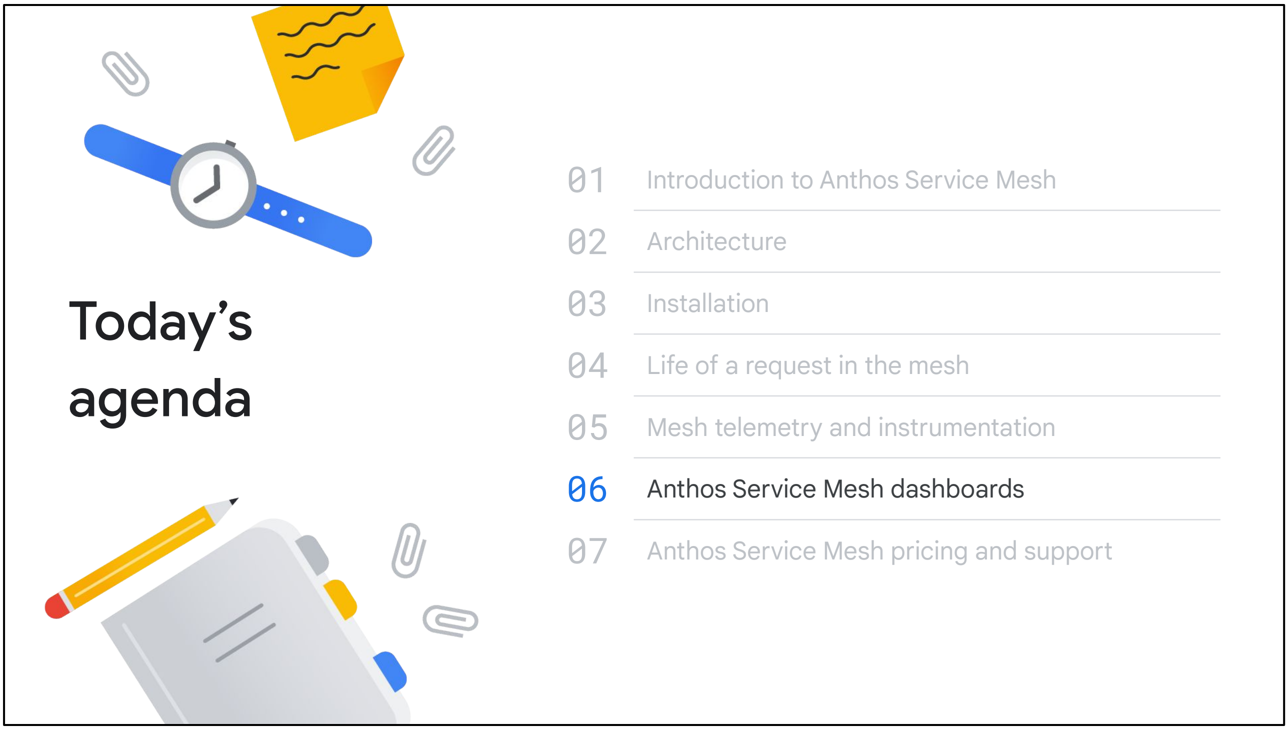
Ruby App
Source
Cloud Trace

To instrument your applications, there are a variety of options. Tracing implementation is evolving rapidly, so the details may change. Across all languages, Google is transitioning to make use of OpenTelemetry libraries.

However, at the moment, for library support by language, use:

- OpenTelemetry for Node.js and Go.
- OpenCensus for Python, Java, Go, and PHP.
- And Cloud Trace for Ruby, Node.js, and C# (both ASP.NET core and regular ASP.NET).

Once traces are collected by your library of choice, you can visualize requests in the Google Cloud Console using the Cloud Trace dashboards.



Today's agenda

01 Introduction to Anthos Service Mesh

02 Architecture

03 Installation

04 Life of a request in the mesh

05 Mesh telemetry and instrumentation

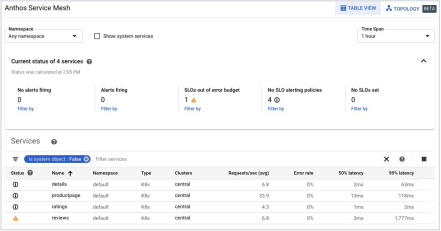
06 Anthos Service Mesh dashboards

07 Anthos Service Mesh pricing and support

Once telemetry has been collected, let's see the visualizations that the Anthos Service Mesh dashboards offer.

Anthos Service Mesh delivers service dashboards

- Displays golden signals for each service.
- Displays service health in relation to SLOs.
- Allows drill-down by service.
- Identifies how traffic flows between services.



Anthos Service Mesh

Namespace: Any namespace

☐ Show system services

Time Span: 1 hour

Current status of 4 services

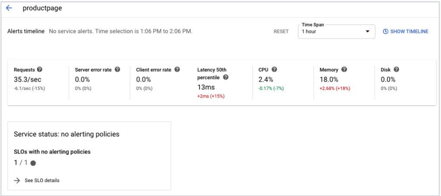
Status was calculated at 2:05 PM

No alerts firing	Alerts firing	SLOs out of error budget	No SLO alerting policies	No SLOs set
0	0	1 ⚠	4 ⓘ	0
Filter by	Filter by	Filter by	Filter by	Filter by

Services

Is system object : False ⊗ Filter services

Status	Name ↑	Namespace	Type	Clusters	Requests/sec (avg)	Error rate	50% latency	99% latency
ⓘ	details	default	K8s	central	6.8	0%	2ms	63ms
ⓘ	productpage	default	K8s	central	33.9	0%	13ms	118ms
ⓘ	ratings	default	K8s	central	4.3	0%	1ms	2ms
⚠	reviews	default	K8s	central	6.8	0%	3ms	1,777ms



← productpage

Alerts timeline No service alerts. Time selection is 1:06 PM to 2:06 PM.

Time Span: 1 hour

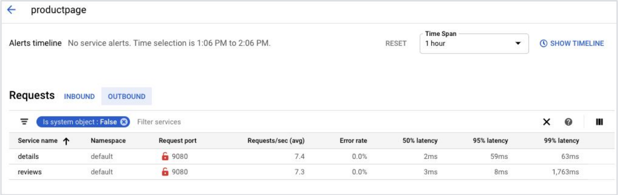
Requests	Server error rate	Client error rate	Latency 50th percentile	CPU	Memory	Disk
35.3/sec	0.0%	0.0%	13ms	2.4%	18.0%	0.0%
-6.1/sec (-15%)	0% (0%)	0% (0%)	+2ms (+15%)	-0.17% (-7%)	+2.68% (+18%)	0% (0%)

Service status: no alerting policies

SLOs with no alerting policies

1 / 1 ●

→ See SLO details



← productpage

Alerts timeline No service alerts. Time selection is 1:06 PM to 2:06 PM.

Time Span: 1 hour

Requests INBOUND OUTBOUND

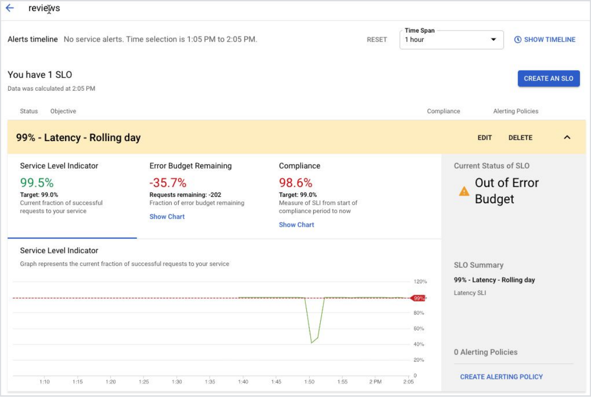
Is system object : False ⊗ Filter services

Service name ↑	Namespace	Request port	Requests/sec (avg)	Error rate	50% latency	95% latency	99% latency
details	default	9080	7.4	0.0%	2ms	59ms	63ms
reviews	default	9080	7.3	0.0%	3ms	8ms	1,763ms

Anthos Service Mesh delivers service dashboards so that you can:

- Get an overview of all services in your mesh, providing you critical, service-level metrics on three of the four golden signals of monitoring: latency, traffic, error, and saturation.
- Define, review, and set alerts against service level objectives, or SLOs, which summarize your service’s user-visible performance.
- View metric charts for individual services and deeply analyze them with filtering and breakdowns, including by response code, protocol, destination pod, traffic source, and more.
- Get detailed information about the endpoints for each service, and see how traffic is flowing between services - as well as what performance looks like for each communication edge.

Anthos Service Mesh enables SLO reporting/alerting



reviews

Alerts timeline No service alerts. Time selection is 1:05 PM to 2:05 PM.

Time Span: 1 hour

You have 1 SLO

Data was calculated at 2:05 PM

Status Objective Compliance Alerting Policies

99% - Latency - Rolling day

Service Level Indicator

99.5%

Target: 99.0%

Current fraction of successful requests to your service

Error Budget Remaining

-35.7%

Requests remaining: -202

Fraction of error budget remaining

Compliance

98.6%

Target: 99.0%

Measure of SLI from start of compliance period to now

Current Status of SLO

Out of Error Budget

Service Level Indicator

Graph represents the current fraction of successful requests to your service

SLO Summary

99% - Latency - Rolling day

Latency SLI

0 Alerting Policies

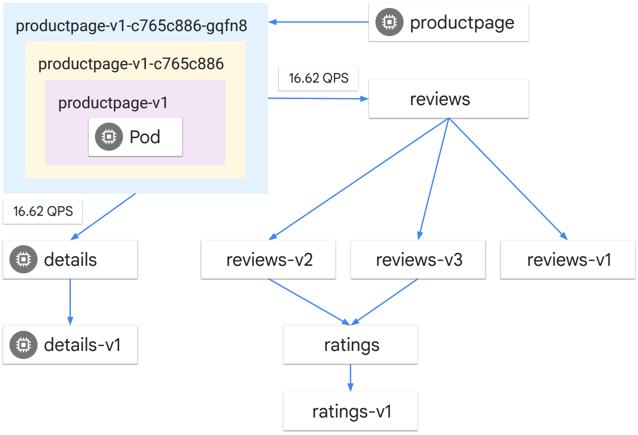
- Allows operators to define SLOs.
- Shows SLO performance, compliance, and error budget for every service.
- Allows alerting based on SLO performance.

Decide on the key metrics that you want to report on, for instance latency and availability of your service. These metrics are your Service Level Indicators, or SLIs.

Set up thresholds for performance and compliance on your SLIs and define an error budget for them. These targets are called Service Level Objectives, or SLOs.

Anthos Service Mesh allows you to monitor your SLIs and set up SLOs, so that service operators have a unified location to understand the health of their services and can set up alerts to notify them when an SLO runs out of budget.

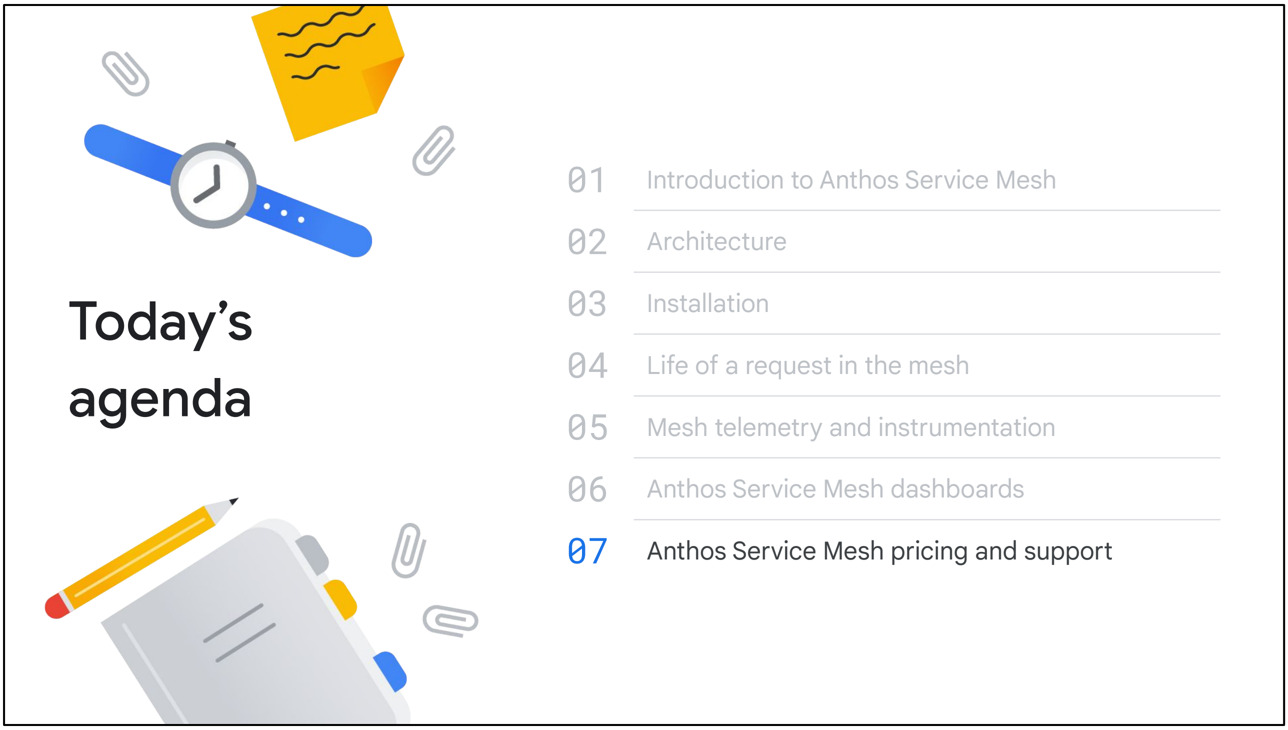
Anthos Service Mesh visualizes your mesh topology



- Creates a chart to represent relationships and traffic flow between services.
- Allows drill-down to see the workloads and pods behind services.
- Displays QPS rates between services.

Anthos Service Mesh dashboards also visualize your mesh topology. A service chart is automatically generated to represent relationships and traffic flow between services.

You can also drill down to see the workloads and pods behind services and see the queries per second, or QPS, rates between services.



Today's agenda

01 Introduction to Anthos Service Mesh

02 Architecture

03 Installation

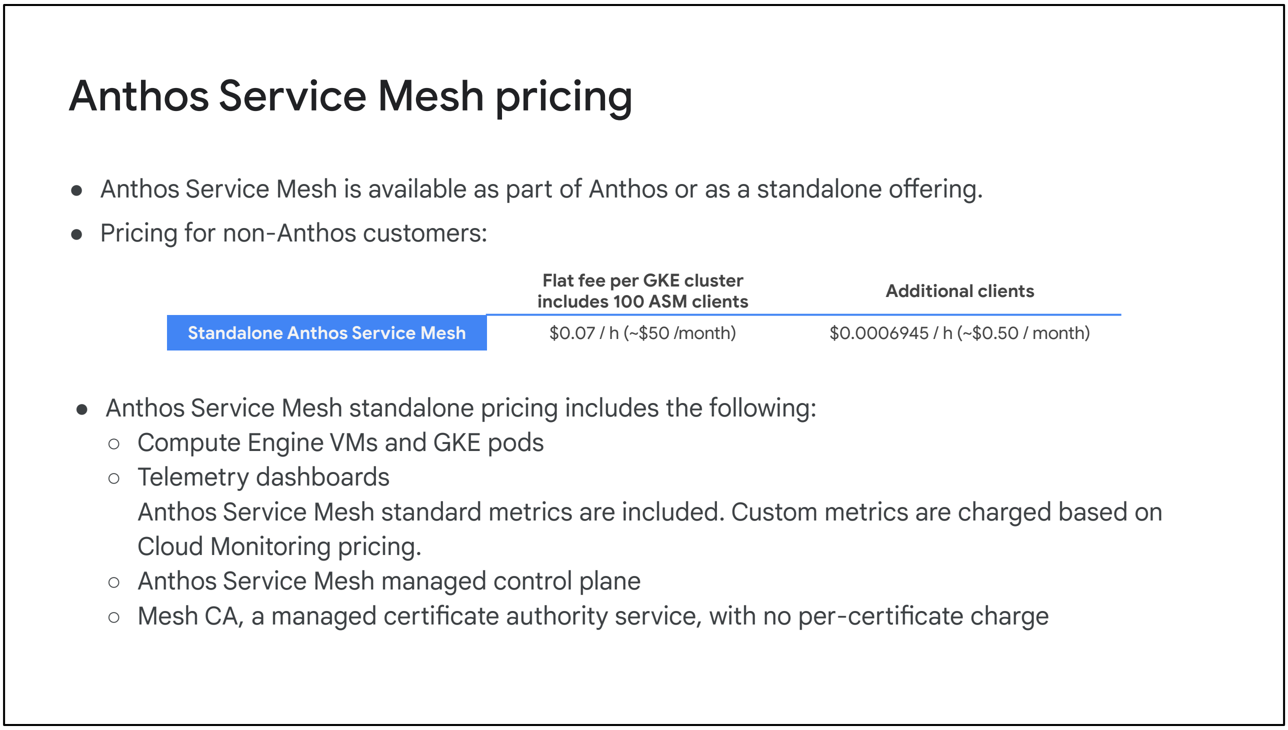
04 Life of a request in the mesh

05 Mesh telemetry and instrumentation

06 Anthos Service Mesh dashboards

07 Anthos Service Mesh pricing and support

Finally, let's review Anthos Service Mesh pricing and support model.



Anthos Service Mesh pricing

- Anthos Service Mesh is available as part of Anthos or as a standalone offering.
- Pricing for non-Anthos customers:

	Flat fee per GKE cluster includes 100 ASM clients	Additional clients
Standalone Anthos Service Mesh	$0.07 / h (~$50 /month)	$0.0006945 / h (~$0.50 / month)

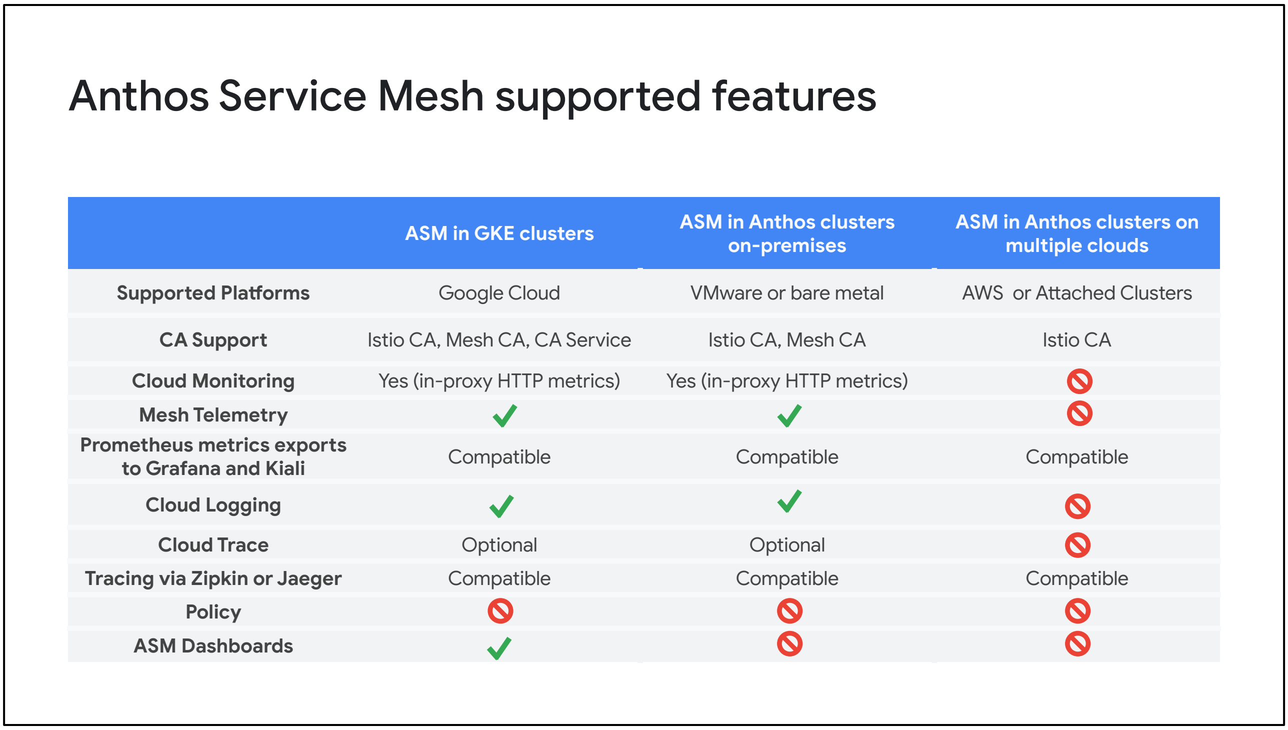
- Anthos Service Mesh standalone pricing includes the following:
  - Compute Engine VMs and GKE pods
  - Telemetry dashboards
    Anthos Service Mesh standard metrics are included. Custom metrics are charged based on Cloud Monitoring pricing.
  - Anthos Service Mesh managed control plane
  - Mesh CA, a managed certificate authority service, with no per-certificate charge

Anthos Service Mesh (or ASM) is available as part of Anthos or as a standalone offering on Google Cloud. Google APIs enabled on the project determine how you are billed. To use Anthos Service Mesh as a standalone service, don't enable the Anthos API on your project. If you want to use Anthos Service Mesh on-premises or on other clouds, you must subscribe to Anthos. If you are subscribed to Anthos, there are no additional costs for ASM.

Pricing for Anthos Service Mesh as a standalone service is based on the number of clusters and the number of Anthos Service Mesh clients. There is a GKE flat fee of 7 cents per hour, which is around 50 dollars per month. The flat fee includes 100 ASM clients, which is equivalent to 72,000 client hours per cluster in a month. Every additional client costs around 50 cents per client per month.

Anthos Service Mesh standalone pricing includes:

- Compute Engine VMs and GKE pods.
- Telemetry dashboards. Anthos Service Mesh standard metrics are included in the fees noted above. Custom metrics are charged based on Cloud Monitoring pricing.
- Anthos Service Mesh managed control plane.
- Mesh CA, a managed certificate authority service, with no per-certificate charge.



Anthos Service Mesh supported features

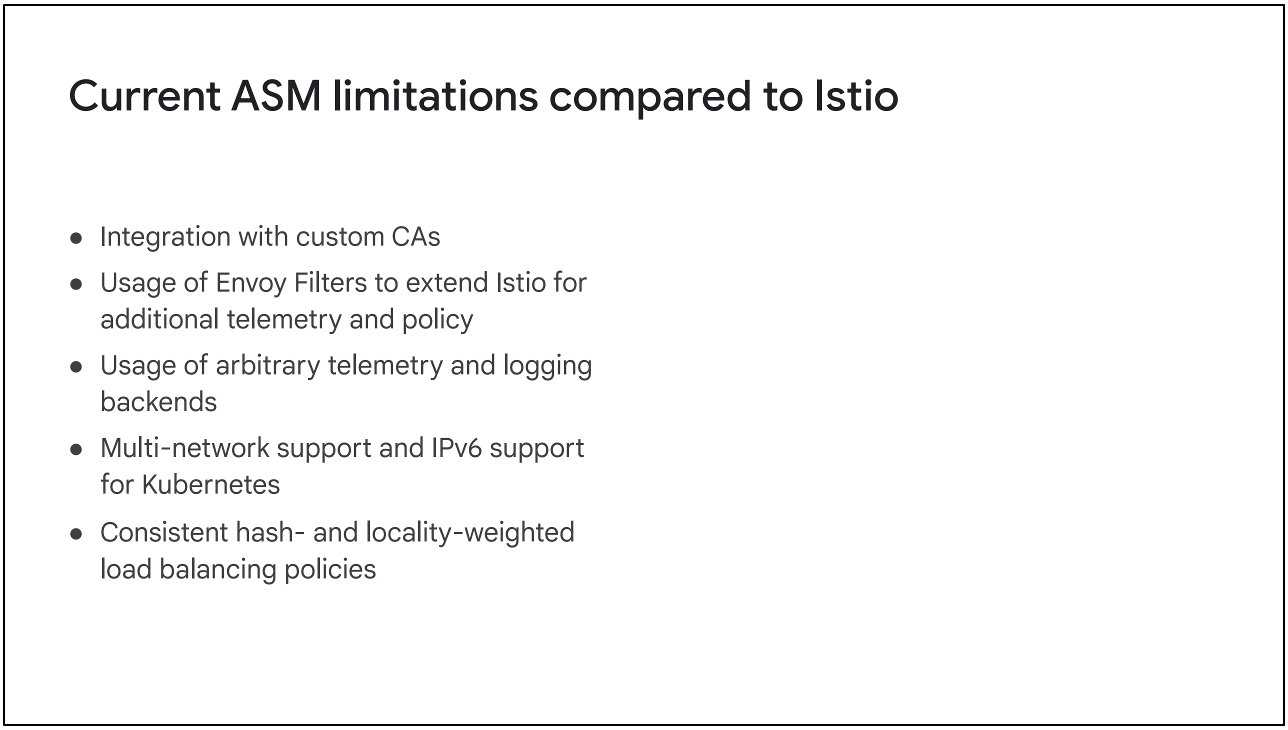
	ASM in GKE clusters	ASM in Anthos clusters on-premises	ASM in Anthos clusters on multiple clouds
Supported Platforms	Google Cloud	VMware or bare metal	AWS or Attached Clusters
CA Support	Istio CA, Mesh CA, CA Service	Istio CA, Mesh CA	Istio CA
Cloud Monitoring	Yes (in-proxy HTTP metrics)	Yes (in-proxy HTTP metrics)	🚫
Mesh Telemetry	✓	✓	🚫
Prometheus metrics exports to Grafana and Kiali	Compatible	Compatible	Compatible
Cloud Logging	✓	✓	🚫
Cloud Trace	Optional	Optional	🚫
Tracing via Zipkin or Jaeger	Compatible	Compatible	Compatible
Policy	🚫	🚫	🚫
ASM Dashboards	✓	🚫	🚫

Not all Anthos Service Mesh features are available in all installations of Anthos, so take a moment to review the supported configurations.

GKE clusters are the first to adopt new ASM features. Therefore, GKE offers the most comprehensive range of options available, including ASM dashboards or the support for CA Service, the SLA-backed root Certificate Authority.

Installations of Anthos clusters on-premises such as VMware and bare metal are the second in line for ASM feature support. All types of metrics, including in-proxy HTTP metrics and a Google's managed Mesh CA, are supported here. Traditionally, operators of clusters on-premises have used open-source dashboarding options such as Grafana, Prometheus, Zipkin, and Kiali to visualize mesh. These can be easily installed to bypass current limitations with regards to ASM dashboards.

Finally, Anthos Service Mesh offers basic support for Anthos clusters running on other clouds, such as AWS or Azure, where most of the managed features can be replaced in the short term with open-source solutions or cloud-specific services.



Current ASM limitations compared to Istio

- Integration with custom CAs
- Usage of Envoy Filters to extend Istio for additional telemetry and policy
- Usage of arbitrary telemetry and logging backends
- Multi-network support and IPv6 support for Kubernetes
- Consistent hash- and locality-weighted load balancing policies

Additionally, there are a couple of limitations on Anthos Service Mesh with respect to Istio as of version 1.11. Those include:

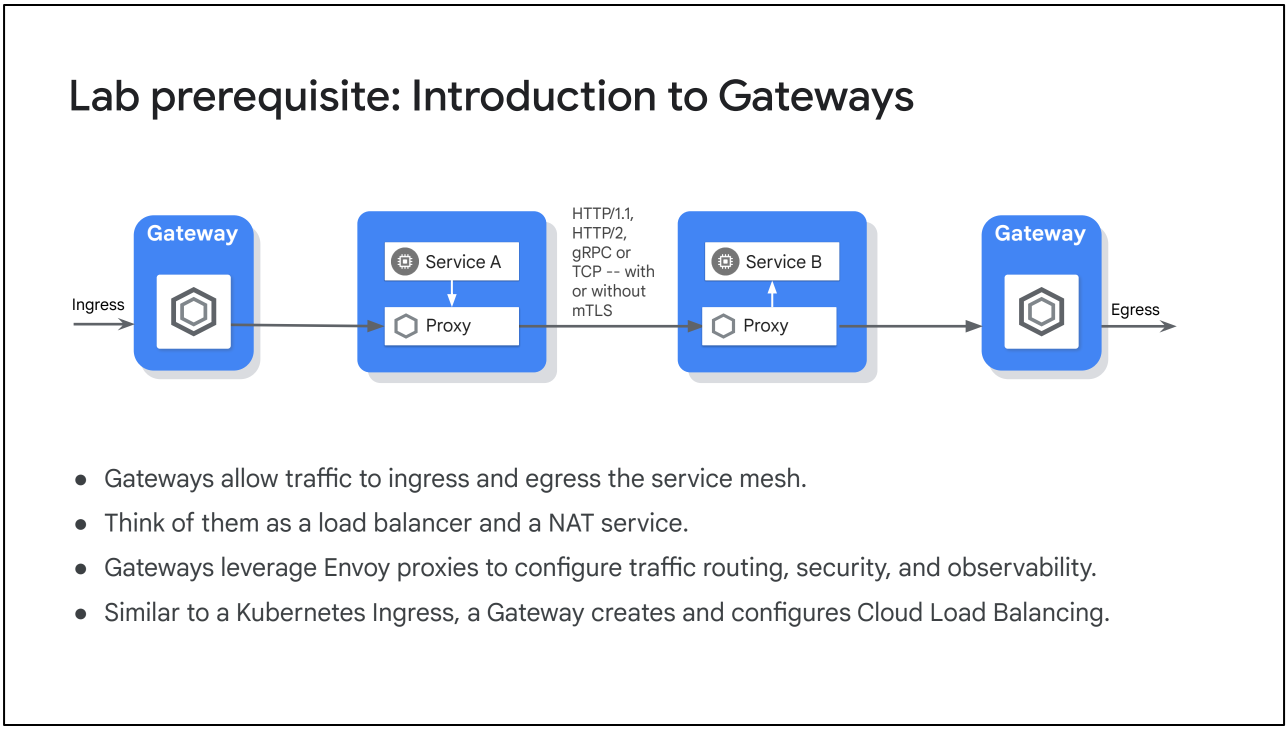
- Service mesh integration with custom CAs. With Istio, you can set up your own custom root CA which might manage identities both inside and outside of your cluster.

- Usage of Envoy Filters to extend the service mesh for additional telemetry and policy. With Istio, you can extend the default functionality to provide additional checks at the networking layer.

- Usage of arbitrary telemetry and logging backends. Anthos Service Mesh provides an out-of-the-box integration with Google Operation Suite, but it's not possible to configure additional backends.

- Multi-network support and IPv6 support for Kubernetes is also only available in Istio.

- Finally, Consistency Hash and locality-weighted lb policies are settings that you can configure on sidecar proxies running open-source istio but are not available on Anthos Service Mesh.

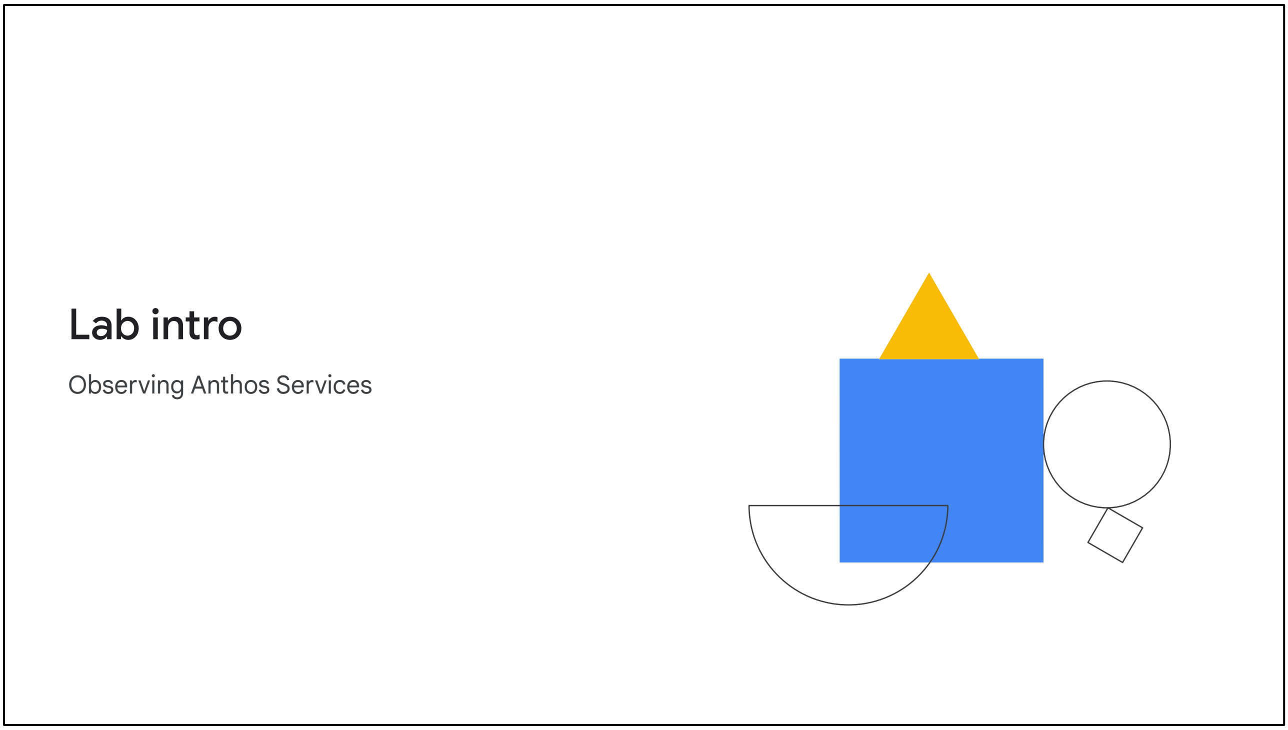


Lab prerequisite: Introduction to Gateways

- Gateways allow traffic to ingress and egress the service mesh.
- Think of them as a load balancer and a NAT service.
- Gateways leverage Envoy proxies to configure traffic routing, security, and observability.
- Similar to a Kubernetes Ingress, a Gateway creates and configures Cloud Load Balancing.

Gateways will be covered in depth later in the course, but we wanted to share a few notes because you encounter them in the next lab.

- Gateways allow traffic to ingress and egress the service mesh.
- You can think of them as a load balancer and a NAT service.
- The same way as other pods in the mesh, Gateways leverage Envoy proxies to configure traffic routing, security, and observability.
- Similar to a Kubernetes Ingress, a Gateway creates and configures Google Cloud Load Balancing, and you can set up ports, protocols, and certificates.



Lab intro

Observing Anthos Services

In this lab, you learn to install Anthos Service Mesh (ASM) on Google Kubernetes Engine.

You will perform the following tasks:

- Install Anthos Service Mesh, with tracing enabled and configured to use Cloud Trace as the backend.
- Deploy Bookinfo, an Istio-enabled multi-service application.
- Enable external access using an Istio Ingress Gateway.
- Use the Bookinfo application.
- Evaluate service performance using Cloud Trace features within Google Cloud.
- Create and monitor service-level objectives (SLOs).
- Leverage the Anthos Service Mesh Dashboard to understand service performance.