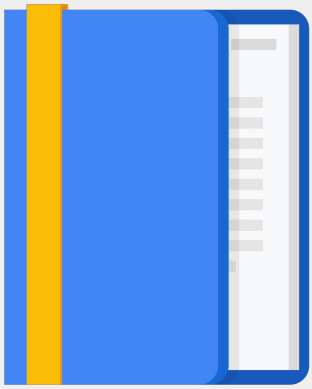
Google Cloud

Architecting Hybrid Infrastructure with Anthos

Managing Traffic Routing with Service Mesh

Agenda



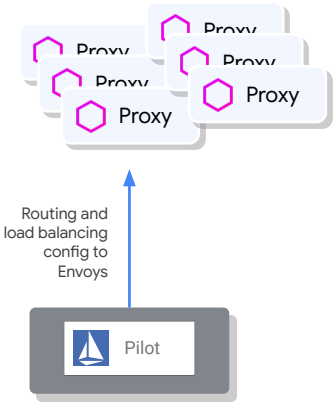
- **Pilot Architecture**
- Traffic Shaping

Google Cloud

Pilot - The Routing of the Mesh

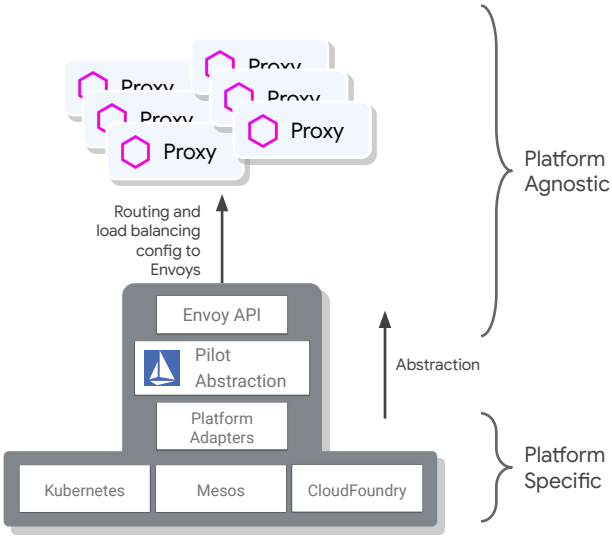
Pilot manages the distributed proxies across the either environments

- Configures Proxies
  - Service Discovery
  - Traffic Management
  - Intelligent Routing
  - Resiliency

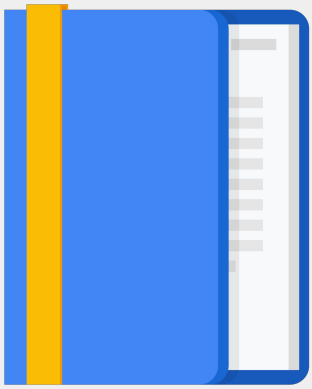


Pilot - The Routing of the Mesh

Pilot collects topological information from the environment it runs on and converts it to Envoy API, called xDS API



Agenda



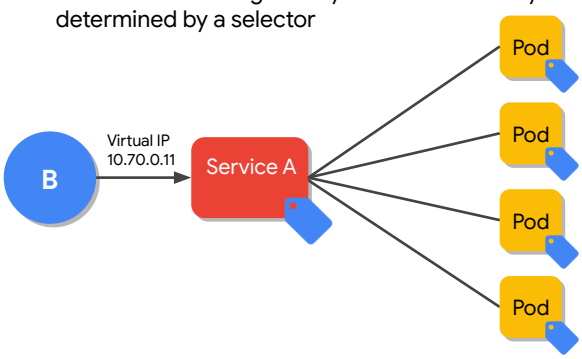
- Pilot Architecture
- **Traffic Shaping**

Google Cloud

Services

Kubernetes

- Abstraction which defines a logical set of Pods and a policy by which to access them
- The set of Pods targeted by a Service is usually determined by a selector

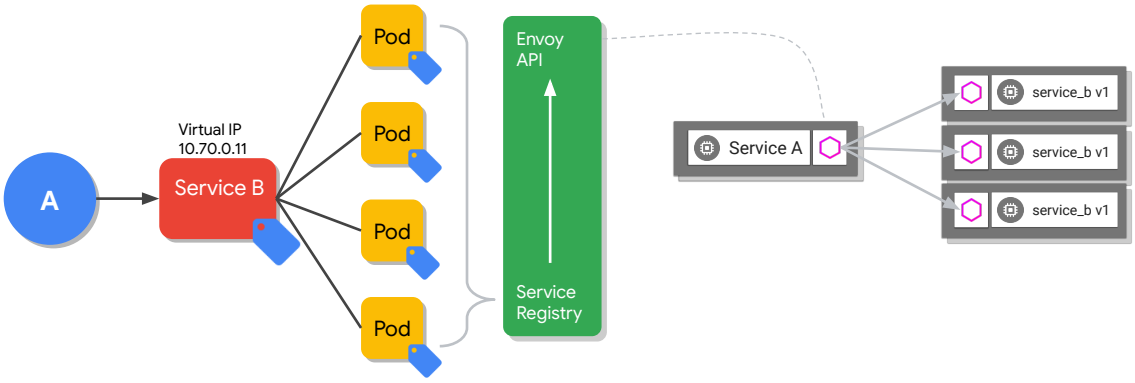


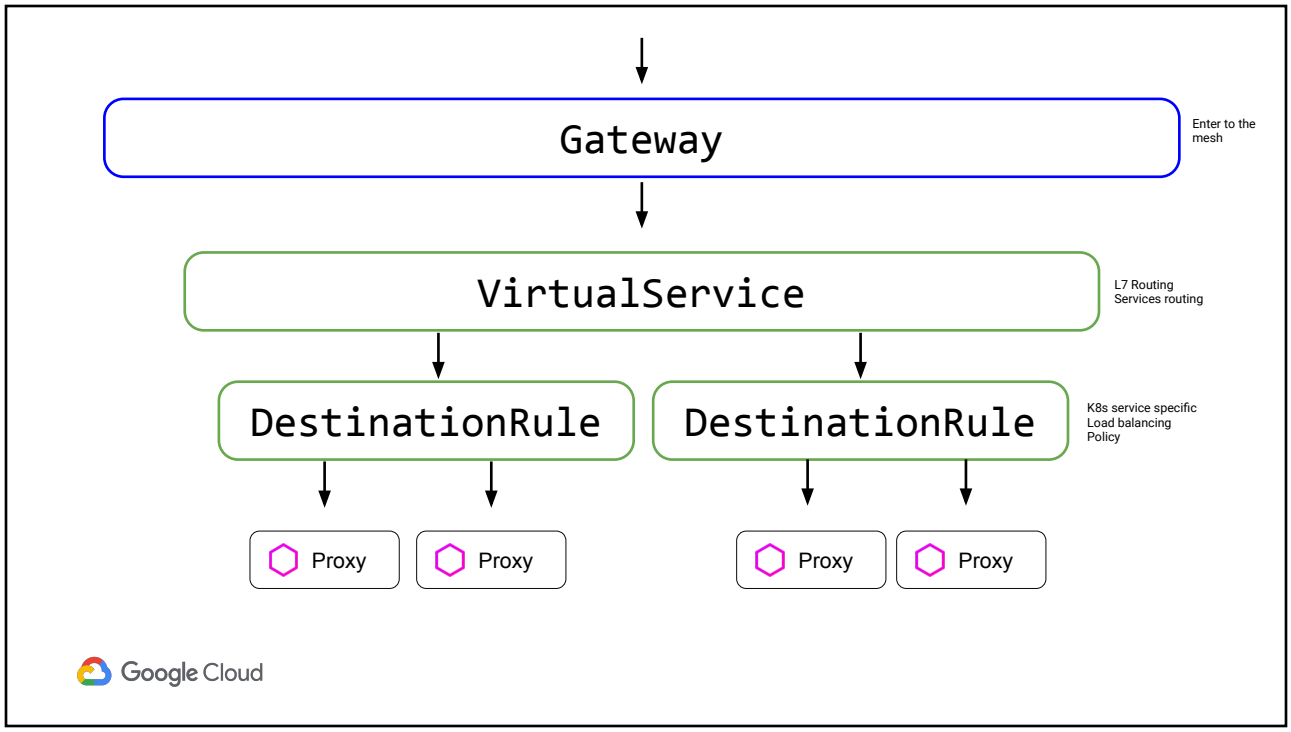
Istio

- Virtual Service
  - rules that control how requests for a service are routed within an Istio service mesh
- Destination Rule
  - configures the set of policies to be applied to a request after VirtualService routing has occurred
- Service Entry
  - commonly used to enable requests to services outside of an Istio service mesh.
- Gateway
  - configures a load balancer for HTTP/TCP traffic, enables ingress for an application

Service Discovery and Load Balancing

Rough grain traffic shaping via
Kubernetes services




Gateway
Enter to the mesh
VirtualService
L7 Routing
Services routing
DestinationRule
DestinationRule
K8s service specific
Load balancing
Policy
Proxy
Proxy
Proxy
Proxy
Google Cloud

VirtualService

Virtual services let you finely configure traffic behavior

Host represents the destination host(s) to which traffic is being sent

- Service
- URL

```yaml
apiVersion:
networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: service_b
spec:
  hosts:
    - service_b
  http:
  - route:
    - destination:
        host: service_b
        subset: v1
      weight: 95
    - destination:
        host: service_b
        subset: v2
      weight: 5
```

VirtualService

VirtualService in combination with a Gateway routes traffic to the appropriate services from outside the Mesh

The `match` attribute lists conditions to be satisfied for the rule to be activated

```yaml
apiVersion:
networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

```yaml
apiVersion:
networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bookinfo
spec:
  hosts:
  - "*"
  gateways:
  - bookinfo-gateway
  http:
  - match:
    - uri:
        exact: /productpage
    route:
    - destination:
        host: productpage
        port:
          number: 9080
```

L7 Traffic Splitting

Traffic can be routed based on

- HTTP header
- URIs
- Ports
- sourceLables

Supports conditional syntax

- Exact
- Prefix
- Regex

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings-route
spec:
  hosts:
  - ratings
  http:
  - match:
    - headers:
        user-agent:
      regex: ^(.*?;)?(iPhone)(;.*)?$
    route:
    - destination:
        host: ratings-iPhone
```

Destination Rule

DestinationRule defines policies that apply to traffic intended for a service after routing has occurred

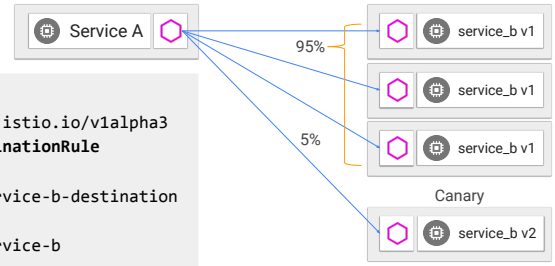
- Load Balancing
- Session affinity
- Connection pool size
- Circuit Breaker

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: bookinfo-ratings-port
spec:
  host: ratings.prod.svc.cluster.local
  trafficPolicy: # Apply to all ports
    portLevelSettings:
    - port:
        number: 80
      loadBalancer:
        simple: LEAST_CONN
    - port:
        number: 9080
      loadBalancer:
        simple: ROUND_ROBIN
```

Traffic Splitting

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name:service-b
spec:
  hosts:
  - service-b
  Http:
  - route:
    - destination:
        host: service-b
        subset: v2
      weight: 5
    - destination:
        host: service-b
        subset: v1
      weight: 95
```

```
apiVersion:
networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: service-b-destination
spec:
  host: service-b
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```



Fault Injection

Inject faults to test the resiliency of your application without instrumentation

Delay

- Delay requests before forwarding, emulating various failures such as network issues, overloaded upstream service, etc.

Abort

- Abort http request attempts and return error codes back to downstream service, giving the impression that the upstream service is faulty

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: service_b
spec:
  hosts:
    - service_b
  http:
  - route:
    - destination:
        host: service_b
        subset: v1
      weight: 95
    - destination:
        host: service_b
        subset: v2
      weight: 5
    fault:
      delay:
        percentage:
          value: 0.1
        fixedDelay: 5s
      abort:
        percentage:
          value: 0.1
        httpStatus: 400
```

Request Timeout

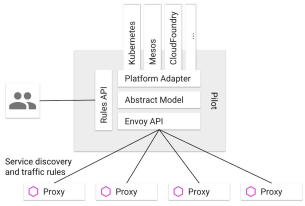
- Default timeout is 15s

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: service_b
spec:
  hosts:
    - service_b
  http:
  - route:
    - destination:
        host: service_b
        subset: v1
      weight: 95
    - destination:
        host: service_b
        subset: v2
      weight: 5
    timeout: 5s
```

Lab

Managing Traffic Routing with Istio and Envoy

60 min



Pilot Architecture

Objectives

- Understand ingress configuration using an Istio Gateway
- Generate traffic, and use Kiali to view routing to multiple versions
- Apply virtual services to route by default to only one version
- Route to a specific version of a service based on user identity
- Shift traffic gradually from one version of a microservice to another