Kubernetes Networking and Istio

Apurva Bhandari

$whoami

Apurva Bhandari, CKAD
SRE / DevOps
Docker & Kubernetes Enthusiast
Speaker at Meetups
Email: apurvbhandari@gmail.com
LinkedIn: https://www.linkedin.com/in/apurvabhandari-linux
GitHub: https://github.com/apurvabhandari/Kubernetes

Agenda

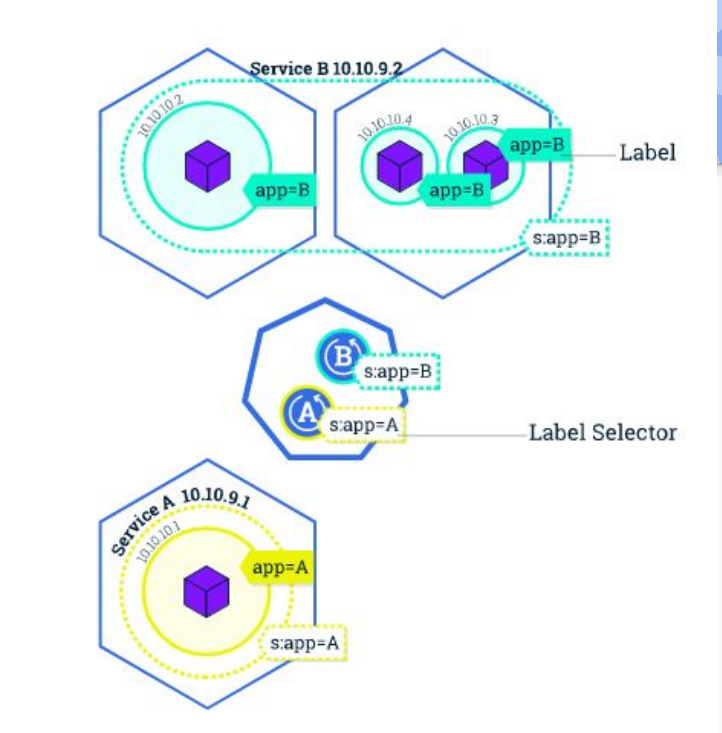
1. Kubernetes Networking Basic
2. Advance routing
3. Introduction to Service Mesh Istio
4. Traffic Routing by Istio
5. Networking with and without Istio

Apurva Bhandari

Services

Types of Services

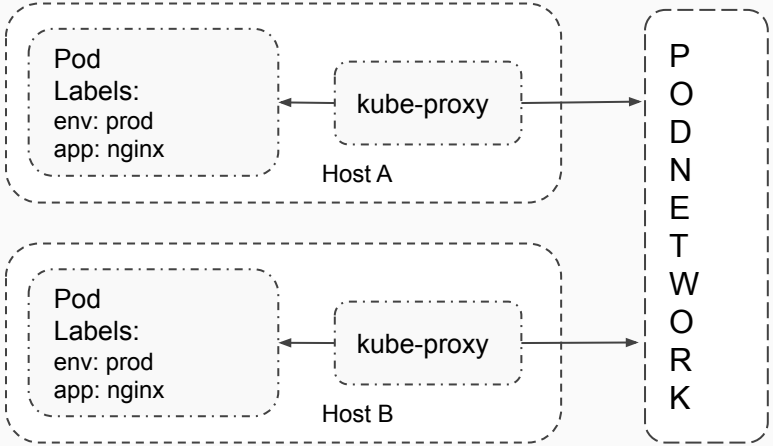
1. ClusterIP (Default)
2. NodePort
3. LoadBalancer
4. ExternalName



Apurva Bhandari

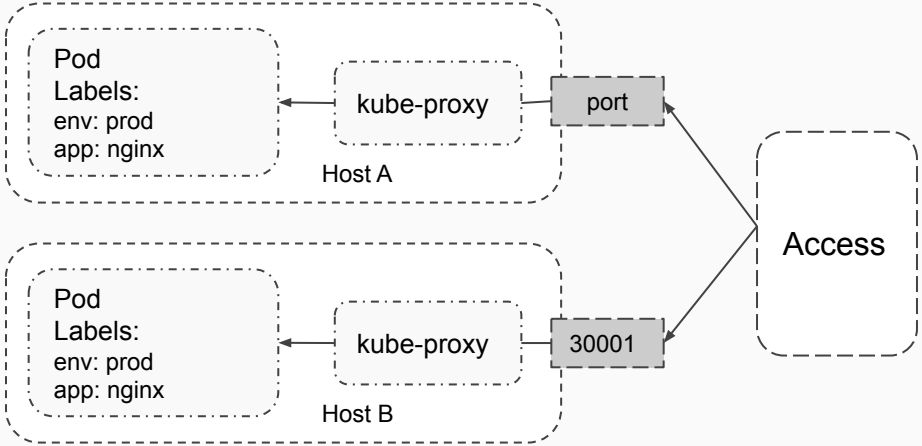
kubernetes

a. ClusterIP



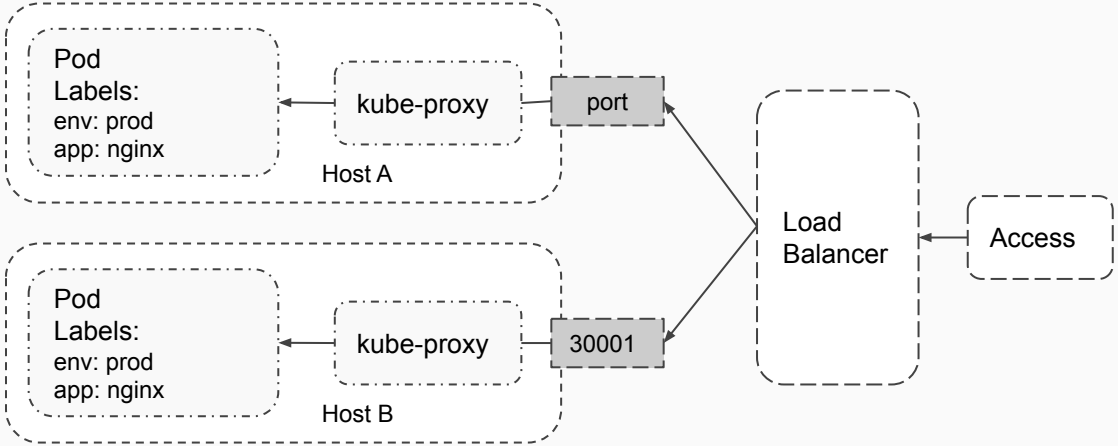

```yaml
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
 selector:
  app: nginx
  env: prod
 ports:
 - protocol: TCP
   port: 80
   targetPort: 80
 type: ClusterIP
```

b. NodePort




```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
spec:
  ports:
   - port: 80
     protocol: TCP
     targetPort: 80
     nodePort: 30001
  selector:
    run: nginx
  type: NodePort
```

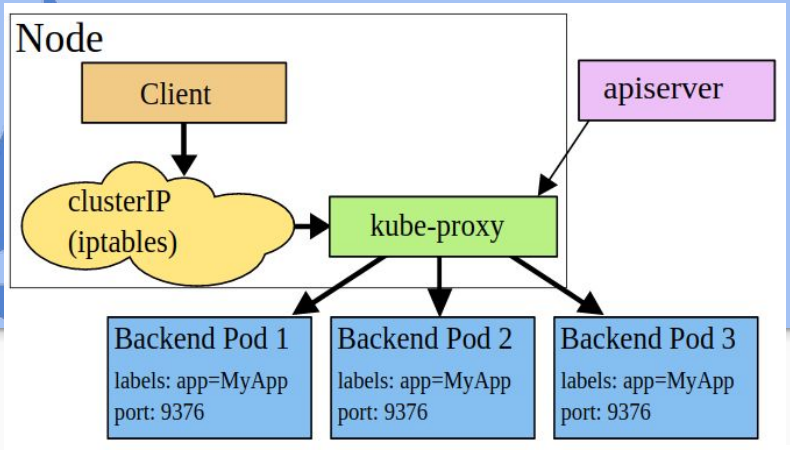
c. LoadBalancer



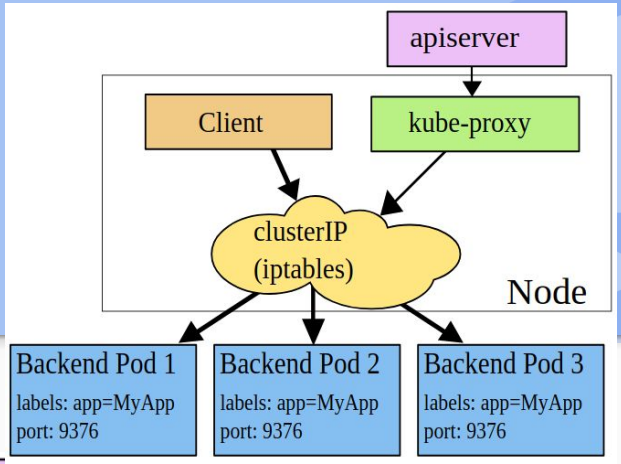

```yaml
apiVersion: v1
kind: Service
metadata:
  name: tomcat
  namespace: default
spec:
  ports:
  - name: healthz
    nodePort: 31768
    port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    run: tomcat
  type: LoadBalancer
```

d. ExternalName

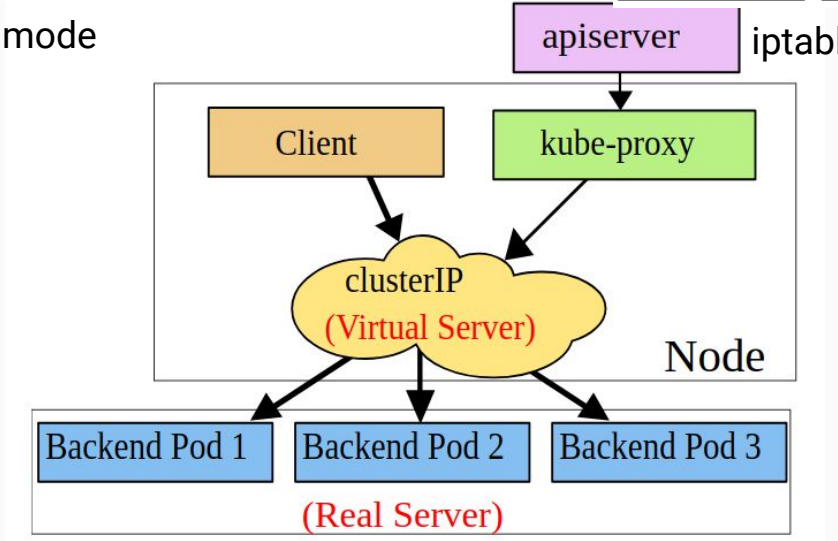
```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
  namespace: prod
spec:
  type: ExternalName
  externalName: my.database.example.com
```



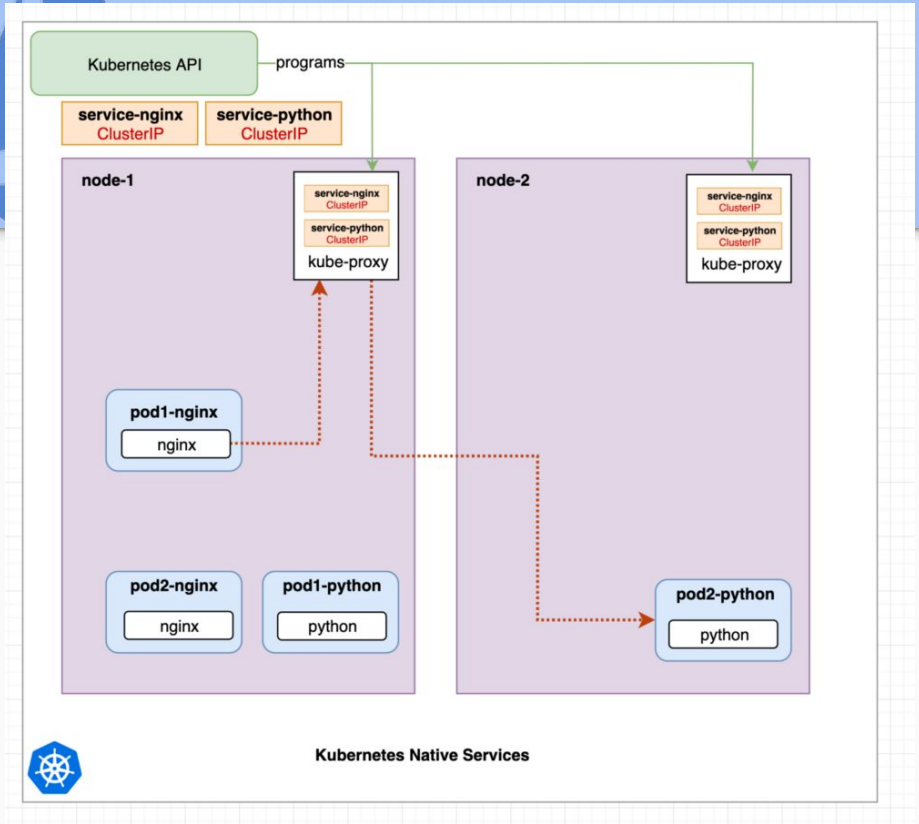
User space proxy mode

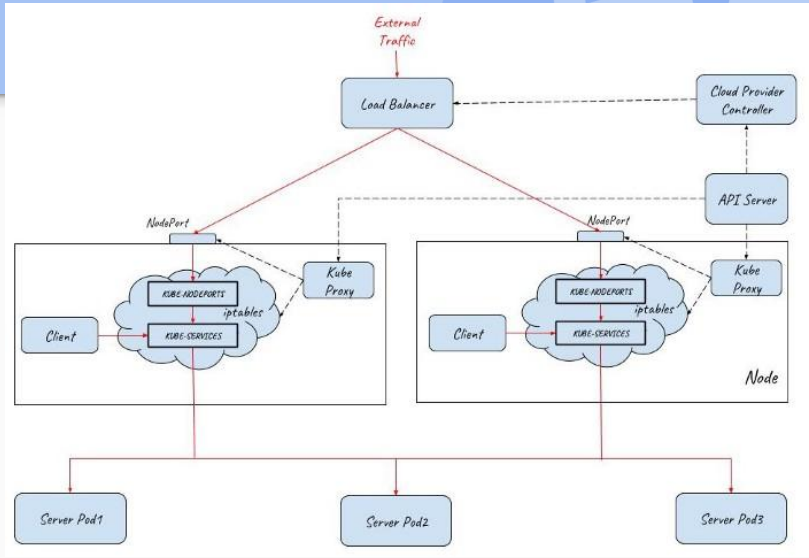


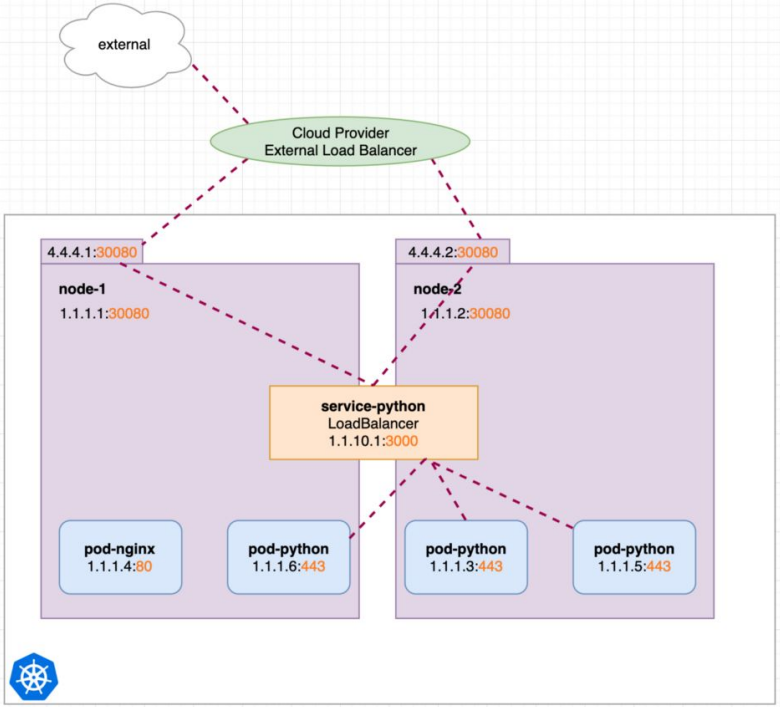
iptables proxy mode

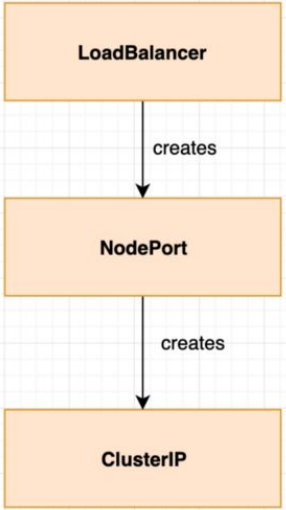


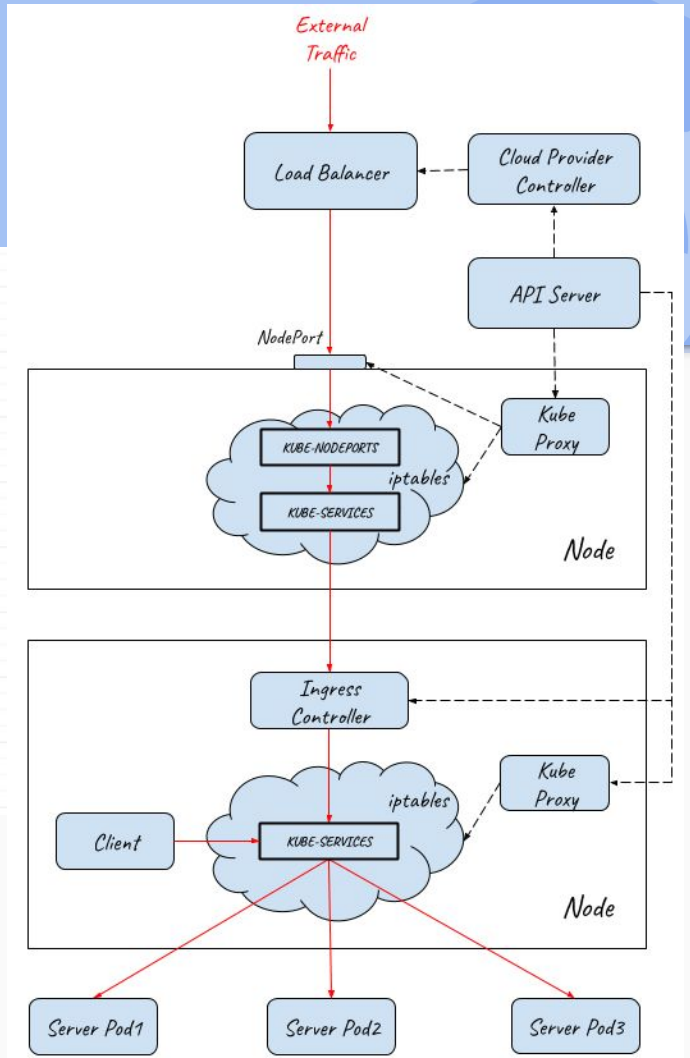
IPVS proxy mode


Kubernetes API
programs
service-nginx
ClusterIP
service-python
ClusterIP
node-1
service-nginx
ClusterIP
service-python
ClusterIP
kube-proxy
pod1-nginx
nginx
pod2-nginx
nginx
pod1-python
python
node-2
service-nginx
ClusterIP
service-python
ClusterIP
kube-proxy
pod2-python
python
Kubernetes Native Services


External Traffic
Load Balancer
Cloud Provider Controller
API Server
NodePort
NodePort
Kube Proxy
KUBE-NODEPORTS
iptables
KUBE-SERVICES
Client
Kube Proxy
KUBE-NODEPORTS
iptables
KUBE-SERVICES
Client
Node
Server Pod1
Server Pod2
Server Pod3


external
Cloud Provider
External Load Balancer
4.4.4.1:30080
4.4.4.2:30080
node-1
1.1.1.1:30080
node-2
1.1.1.2:30080
service-python
LoadBalancer
1.1.10.1:3000
pod-nginx
1.1.1.4:80
pod-python
1.1.1.6:443
pod-python
1.1.1.3:443
pod-python
1.1.1.5:443


LoadBalancer
creates
NodePort
creates
ClusterIP


External
Traffic
Load Balancer
Cloud Provider
Controller
API Server
NodePort
Kube
Proxy
KUBE-NODEPORTS
iptables
KUBE-SERVICES
Node
Ingress
Controller
Kube
Proxy
iptables
Client
KUBE-SERVICES
Node
Server Pod1
Server Pod2
Server Pod3

Introduction to Service Mesh - Istio

Secure, monitor and manage services

Intelligent routing

Control traffic between services with **dynamic route configuration**, conduct **A/B tests**, release **canaries**, and **gradually upgrade versions** using red/black deployments.

Resilience

Increase reliability by shielding applications from flaky networks and cascading failures in adverse conditions. **Timeouts**, **retries**, **health checks** and **circuit breakers** -- all applied regardless of language, across the fleet.

Security & policy

Transparently inject **mutual TLS** on each call, securing and encrypting traffic. Apply **organizational policy** to the interaction between services, ensure **access policies** are enforced and resources are fairly distributed among consumers.

Telemetry

Understand the **dependencies between services**, the nature and **flow of traffic** between them, and quickly identify issues with **distributed tracing**

Istio Value Proposition

Securing service communications

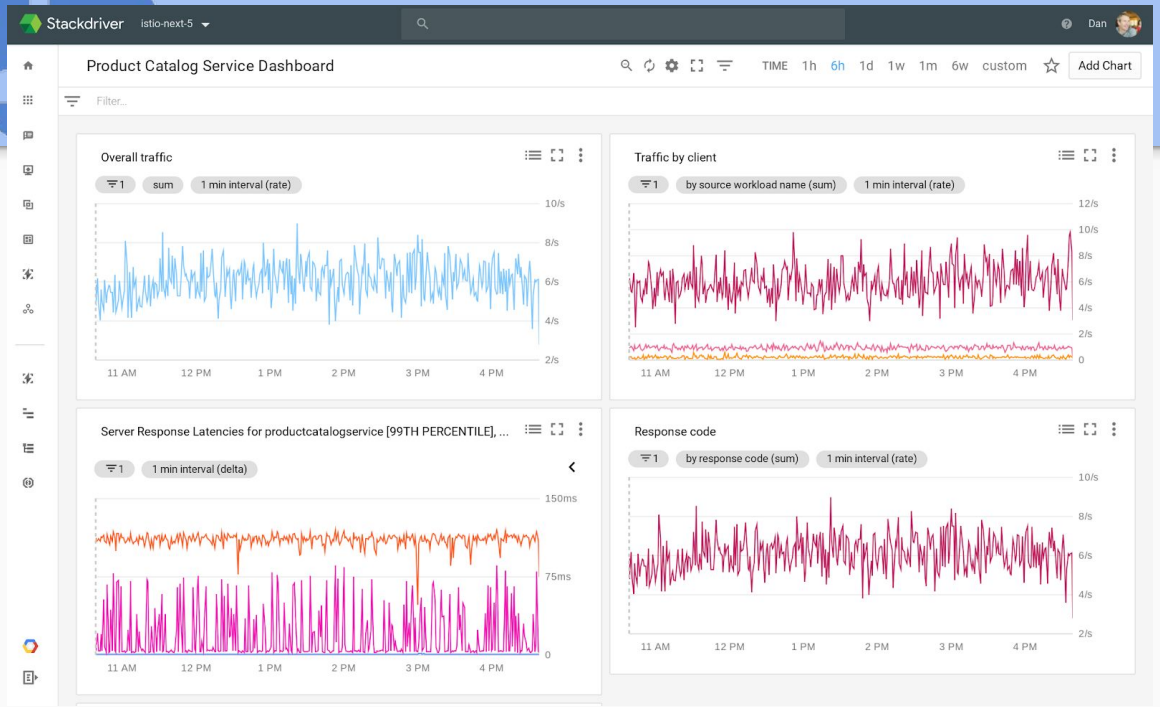
Strongly **authenticate services** (not hosts) across heterogeneous deployment environments. Limit access of sensitive data to authorized services **without relying on L3** controls. Understand security posture of production environment through **service dependency graphs.**

Uniform service-level **observability**

Monitor the "**golden signals**" (traffic, error rates and latency) for all services, and **collect logs** on all calls. Use distributed tracing for in-depth **performance analysis**. Service dependency graphs make it easy to debug and to understand latency and hotspots.

Traffic management and operational agility

Send inter-cluster and inter-environment without manually provisioning ingress, egress, edge layers or hardware LBs. Change **service behavior** and **traffic flow** without redeploying or change of code. Control which services can talk to whom via **policy and routing rules**.



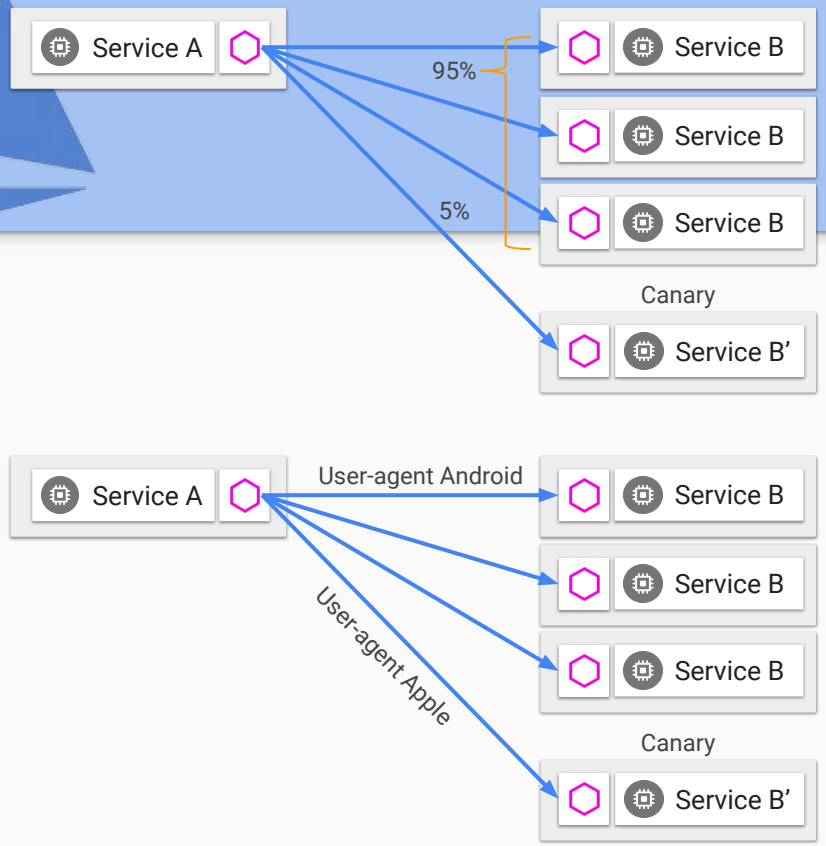
Uniform observability

Collect the **golden signals** for every service and logs for every call.

Understand services and their **dependencies**.

Set, monitor and **enforce SLOs** on services

Bird's eye view of service behavior for issue triage, **reduce time to detect, triage**



Operational agility

Scale by directing traffic to multiple versions

Roll out new versions without worrying about ops challenges

Apply access control, rate limiting policies to **protect services** from bad behavior

Policy driven security

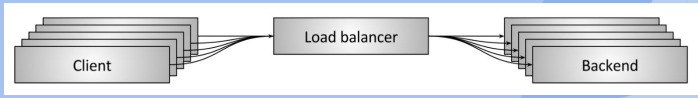
Defence in depth - security does not stop at the edge.

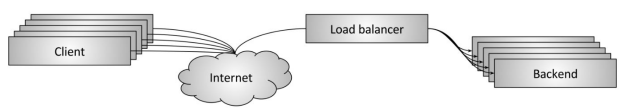
Enable mTLS for authentication and encryption.

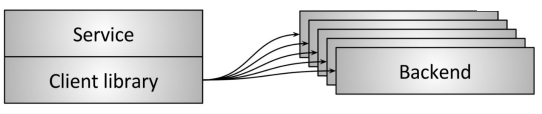
Authorize access based on service identity or any channel attribute.

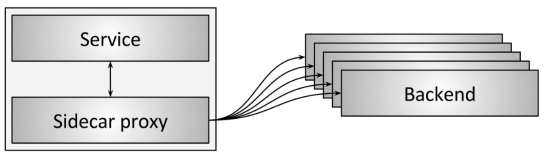
Configure finer grained RPC-level access control for REST and gRPC.

Networking proxy types







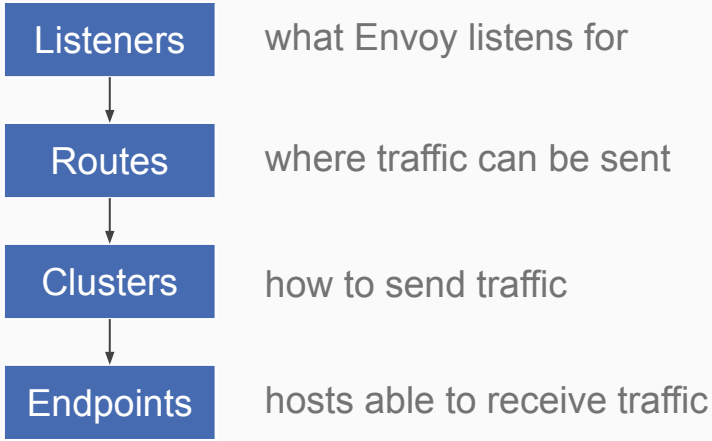


- Middle Proxy
- Edge Proxy
- Embedded Client Library
- Client Side / Sidecar Proxy

- Client Side Load Balancing, no SPoF
- Traditional Load Balancer is Layer 4
- Lightweight sidecars to manage traffic between services
- Scaling capabilities + polyglot aspect
- Sidecars can do much more than just load balancing!

The magic of the sidecar!

- Deployed with every workload
- Proxies all traffic into and out of a service
- Directs traffic (including routing rules)
- Enforces policy
- Reports telemetry
- All with no embedded client library



Envoy

- A C++ based L4/L7 proxy
- Lightweight, low memory footprint
- Battle-tested @ Lyft
  - 100+ services
  - 10,000+ VMs
  - 2M req/s

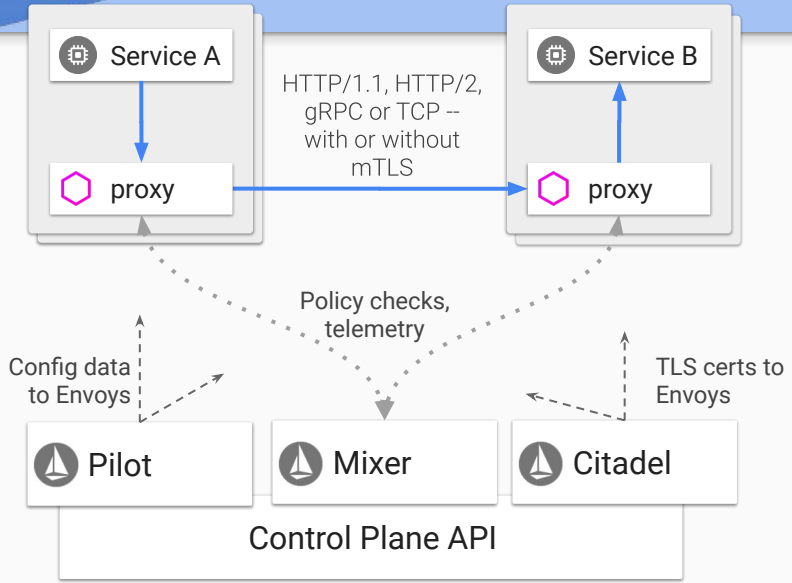
Goodies:

- HTTP/2 & gRPC
- Zone-aware load balancing w/ failover
- Health checks, circuit breakers, timeouts, retry budgets
- No hot reloads - API driven config updates

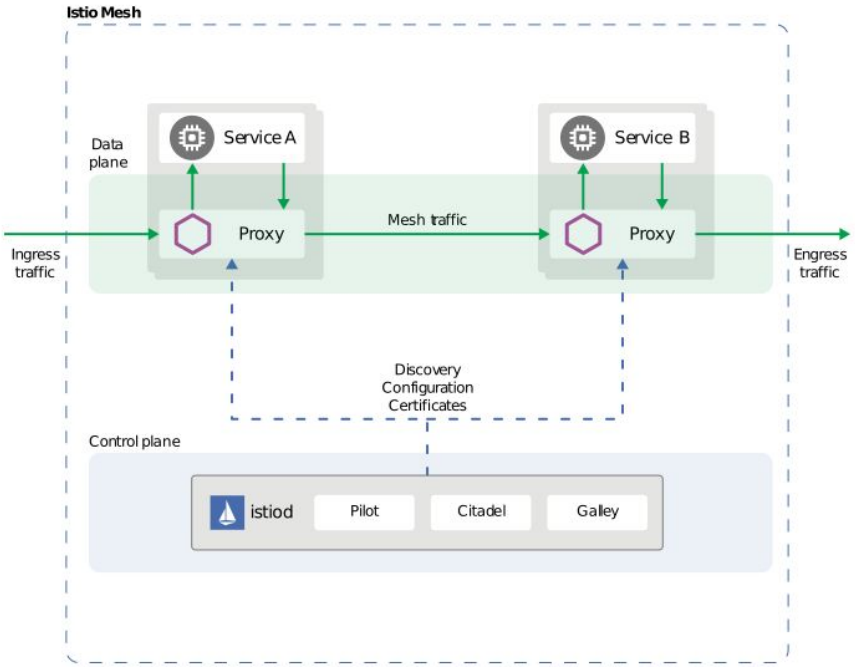
Istio's contributions:

- Transparent proxying w/ SO_ORIGINAL_DST
- Traffic routing and splitting
- Request tracing using Zipkin
- Fault injection

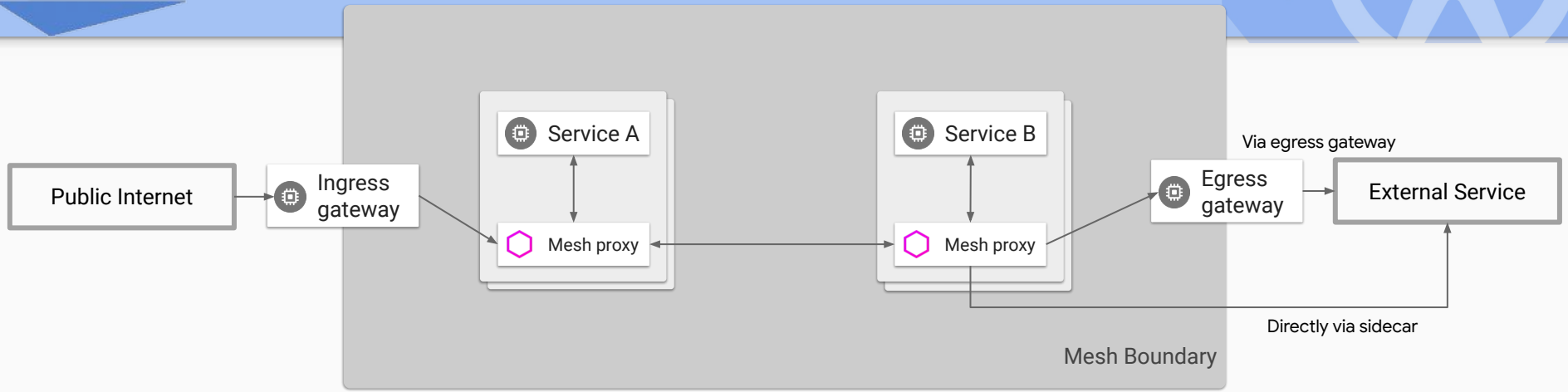
Architectural components

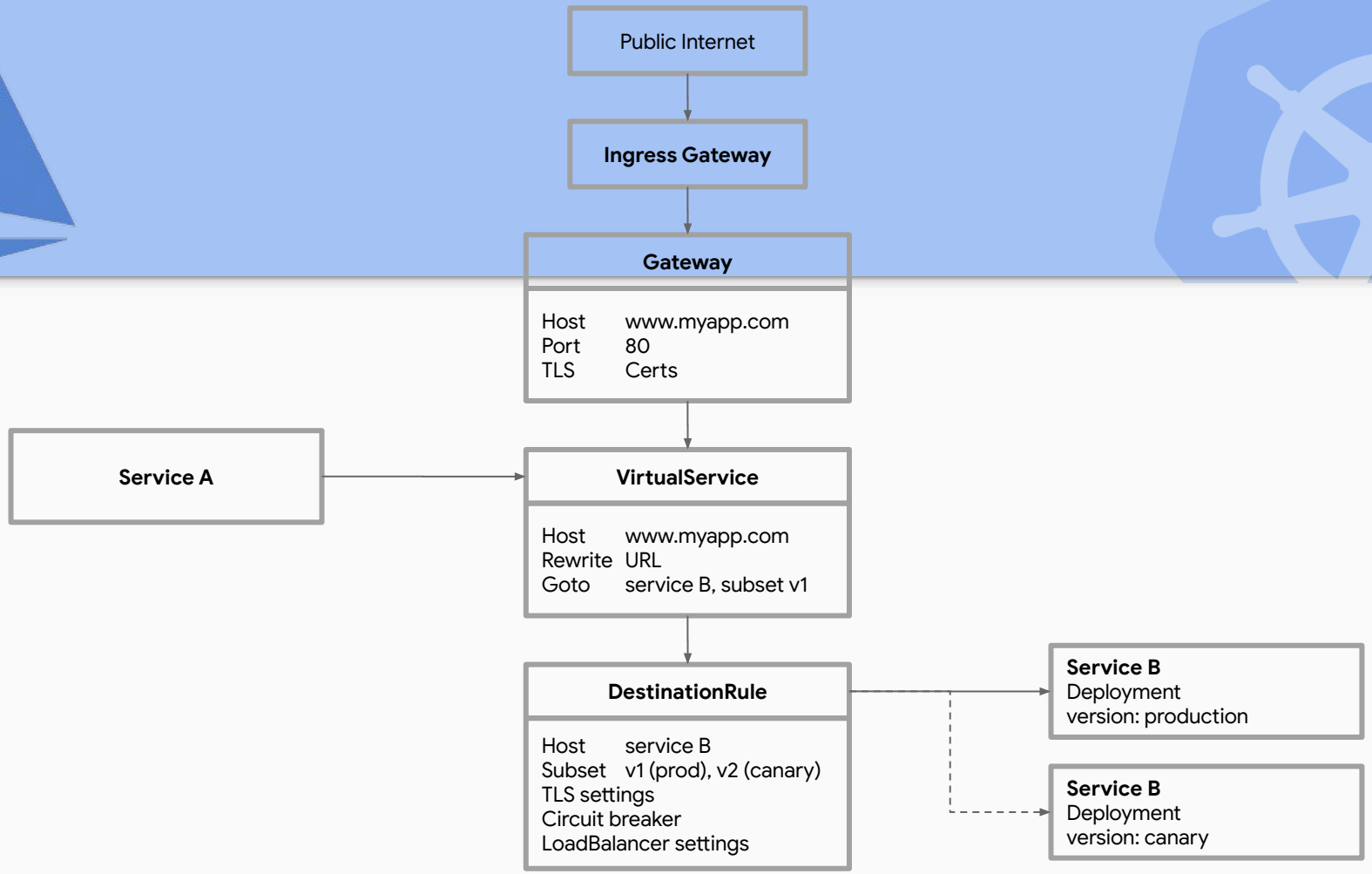


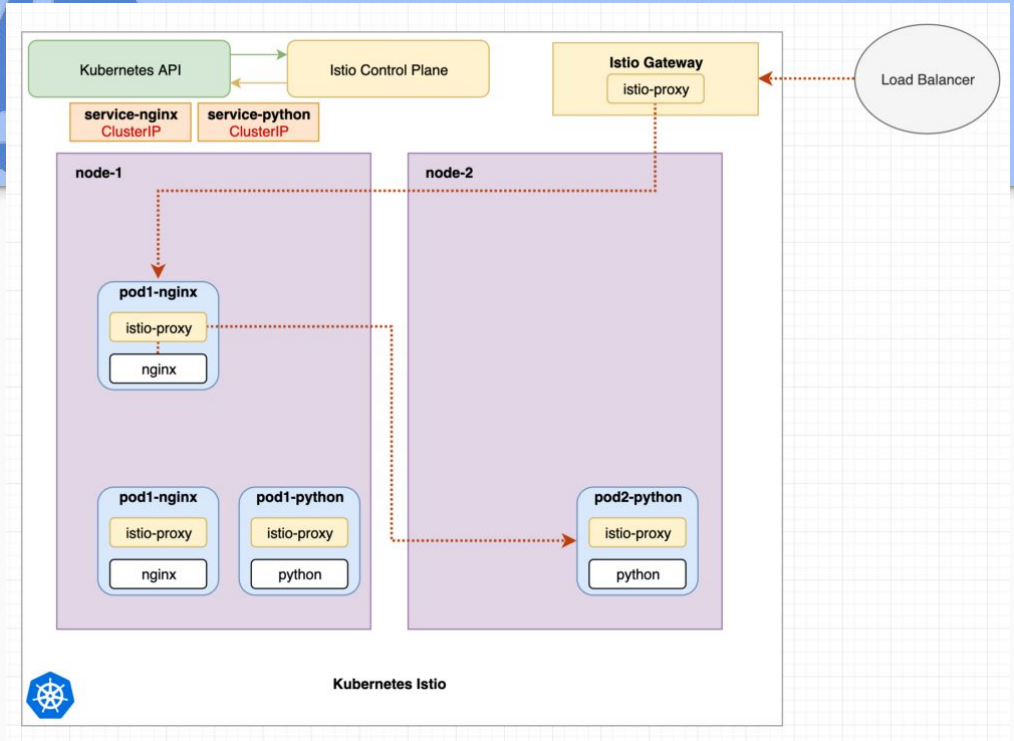
- **Envoy:** Network proxy to intercept communication and apply policies.
- **Pilot**: Control plane to configure and push service communication policies
- **Mixer:** Policy enforcement with a flexible plugin model for providers for a policy.
- **Istio Auth:** Service-to-service auth[n,z] using mutual TLS, with built-in identity and credential management.

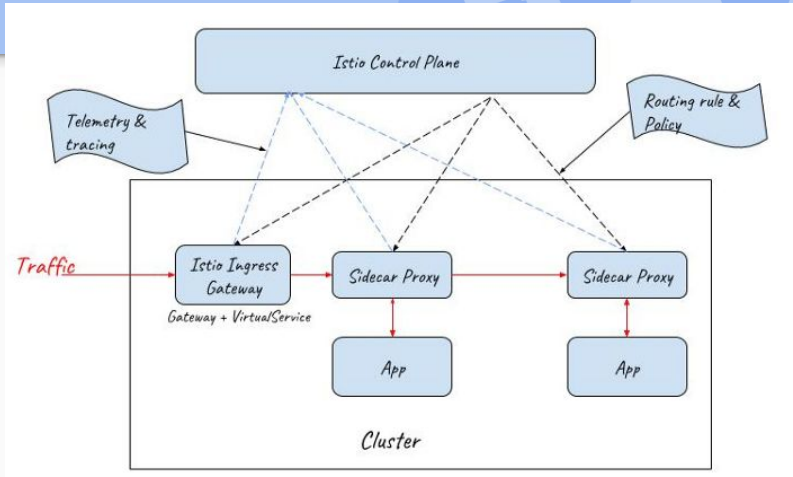


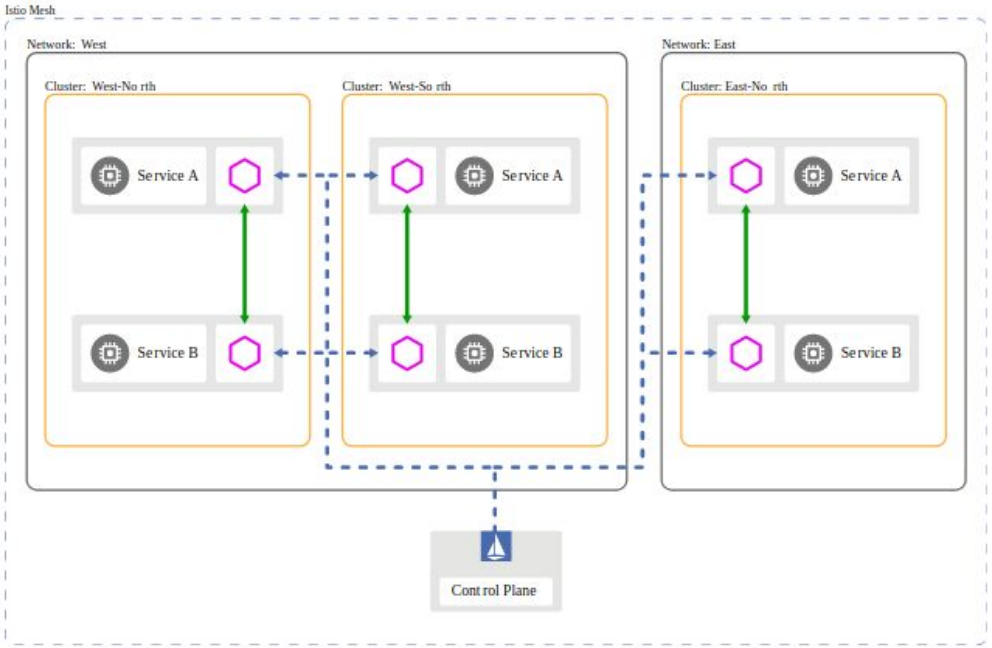
Istio Architecture


Public Internet
Ingress gateway
Service A
Mesh proxy
Service B
Mesh proxy
Egress gateway
Via egress gateway
External Service
Directly via sidecar
Mesh Boundary


Public Internet
Ingress Gateway
Gateway
Host www.myapp.com
Port 80
TLS Certs
Service A
VirtualService
Host www.myapp.com
Rewrite URL
Goto service B, subset v1
DestinationRule
Host service B
Subset v1 (prod), v2 (canary)
TLS settings
Circuit breaker
LoadBalancer settings
Service B
Deployment
version: production
Service B
Deployment
version: canary


Kubernetes API
Istio Control Plane
Istio Gateway
istio-proxy
Load Balancer
service-nginx
ClusterIP
service-python
ClusterIP
node-1
node-2
pod1-nginx
istio-proxy
nginx
pod1-nginx
istio-proxy
nginx
pod1-python
istio-proxy
python
pod2-python
istio-proxy
python
Kubernetes Istio


Istio Control Plane
Telemetry & tracing
Routing rule & Policy
Traffic
Istio Ingress Gateway
Gateway + VirtualService
Sidecar Proxy
Sidecar Proxy
App
App
Cluster



A service mesh with multiple clusters

Traffic Management

- Application rollout (in percentage distribution)
- Traffic steering (content based)
- Resiliency
- Efficiency

Thank You