KubeCon | CloudNativeCon

North America 2019

Take Envoy Beyond a K8s Service Mesh- to Legacy VMs + More

Steve Sloka, VMware
Steven Wong, VMware

Abstract

This slide hidden during presentation – retained for search classification of published deck

Envoy’s mission is to extract network and communication security code from applications in a way that developers and users can deploy components that just work no matter where they run or what hosts them.

This session will show how to leverage Envoy to achieve interoperation of applications and services, split across Kubernetes and traditional VM or bare metal hosts. We’ll look at how to incrementally bring Kubernetes into an existing application architecture based on existing VM or bare metal applications and services.

Specific examples will demonstrate:
- Using Contour with Envoy as an Ingress and load balancer solution with a richer feature set than some common alternatives
- Sending requests from VM workloads to Kubernetes services
- Direct requests to services running on a VM from Kubernetes
- Dynamical traffic steering - K8s and VM workloads at the same time

Speakers

Steve Sloka

Pittsburgh, PA

Senior Member of Technical Staff, VMware

Maintainer of Contour, Gimbal, and the Elasticsearch Operator. Steve is also a Kubernetes contributor and has been working with it since early 2015.

@stevesloka

Steven Wong

Los Angeles, CA

Open Source Engineer, VMware

Active in Kubernetes community since 2015 – storage, IoT+Edge, running K8s on VMware infrastructure.

GitHub: @cantbewong

Agenda

Overview:

- What are we trying to Solve
- What is Envoy?
- What is Service Mesh?

Taking Envoy beyond Service Mesh:

- Envoy as an Ingress and Load Balancer

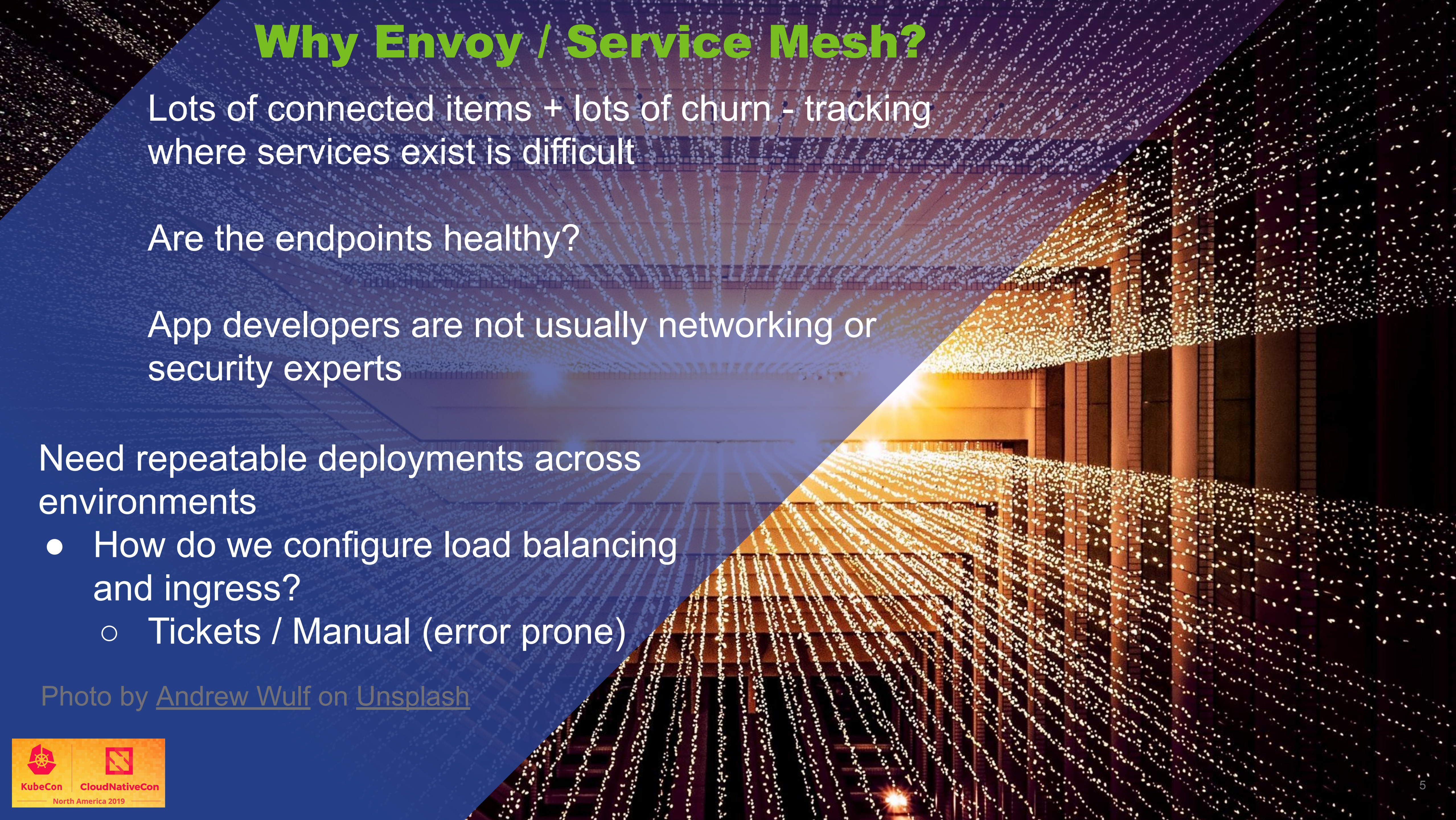
Demo:

- Bridge Legacy VMs and bare metal into a Kubernetes cluster

Resources:

- Where to learn more

Why Envoy / Service Mesh?



Lots of connected items + lots of churn - tracking where services exist is difficult

Are the endpoints healthy?

App developers are not usually networking or security experts

Need repeatable deployments across environments

- How do we configure load balancing and ingress?
  - Tickets / Manual (error prone)

Photo by Andrew Wulf on Unsplash

What is Envoy

Announced by Lyft in 2016

Envoy is an open source edge and service proxy, designed for cloud-native applications.

Goal:

- Move network details+code out of application - make network transparent to app devs and users

Architecture Emphasis:

- API driven; dynamic configuration support
- top notch support for observability and debugging

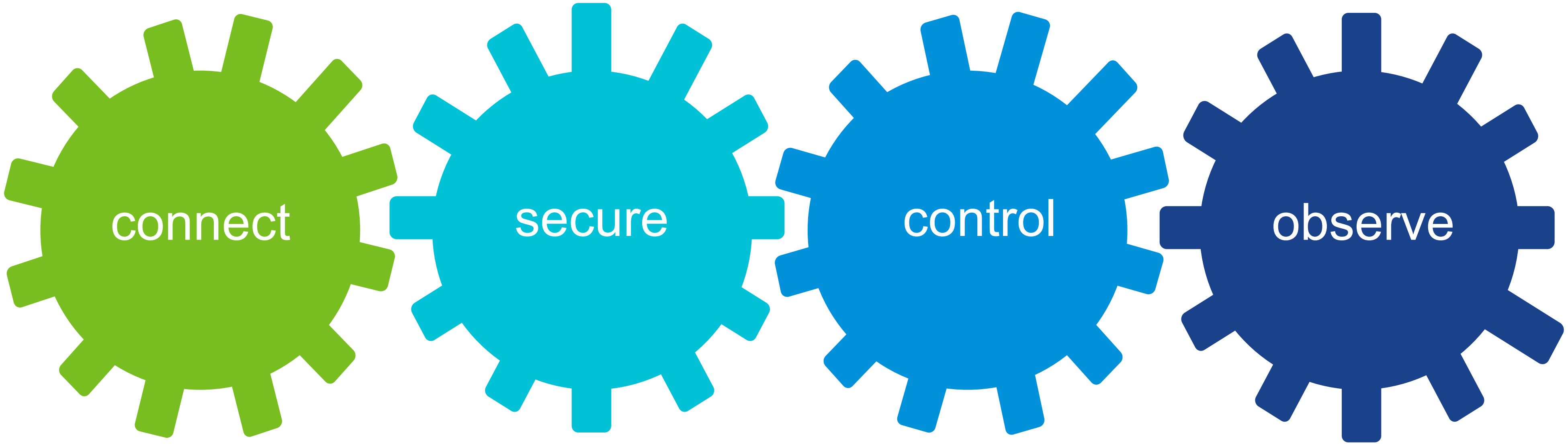
Why service mesh?

solve / manage network + security issues at scale

- Automatic load balancing for HTTP, gRPC, WebSocket, and TCP traffic
- Fine-grained control of traffic behavior with rich routing rules, retries, failovers, and fault injection
- A pluggable policy layer and configuration API supporting access controls, rate limits and quotas
- Automatic metrics, logs, and traces for all traffic within a cluster, including cluster ingress and egress
- Secure service-to-service authentication with strong identity assertions between services in a cluster

Istio

Service mesh



connect

- Discovery
- Advanced Routing
- Canaries
- Load Balancing

secure

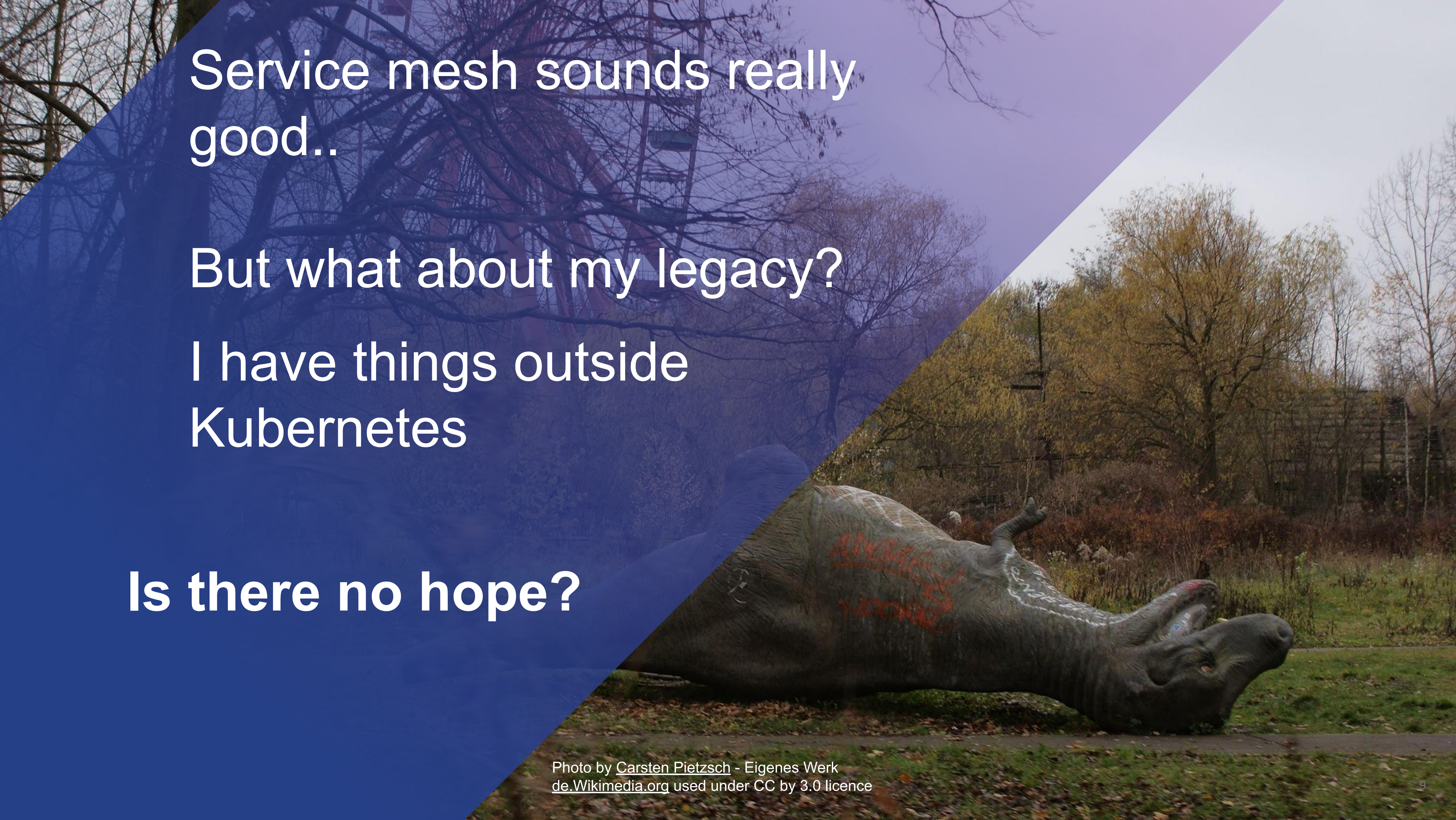
- Authentication
- Encryption
- TLS Certificates
- Org-wide policies

control

- Centrally manage operations
- Heath checks,
- circuit breakers
- Retries
- Rate limiting

observe

- Monitoring
  - Traffic
  - Service dependencies
- Distributed Tracing
- Audit Logs



Service mesh sounds really good..

But what about my legacy?

I have things outside Kubernetes

Is there no hope?

Photo by Carsten Pietzsch - Eigenes Werk
de.Wikimedia.org used under CC by 3.0 licence

Very fast look at Istio mesh expansion

too much to demo in a 35 minute session

but in deck so you can come back later

much of his is not currently documented

Photo toine Garnier on Unsplash

Istio Mesh expansion - join VMs and bare metal to K8s

summary of steps - lots more detail in “hidden slide” in our published deck

pre-conditions in Kubernetes cluster

- Install Istio with global.meshExpansion.enabled = true (not default)
- You will need a load balancer (such as MetalLB)

On VM(s)

- cluster.env file (config settings) key and cert files
- Istio/Envoy sidecar .deb package
- Verify node_agent works
- start istio-auth-node-agent, istio service daemons

If hosting a service on VM

- create a ServiceEntry resource using kubectl
- use isioctl to register (“map”) the VM hosted service

Taking Envoy beyond a Service Mesh

Use Envoy to make Ingress / Load Balancer

Why not service mesh?

- Complex
- Overhead in cluster
- Difficult to debug

CONTOUR

Envoy as Ingress and Load Balancer

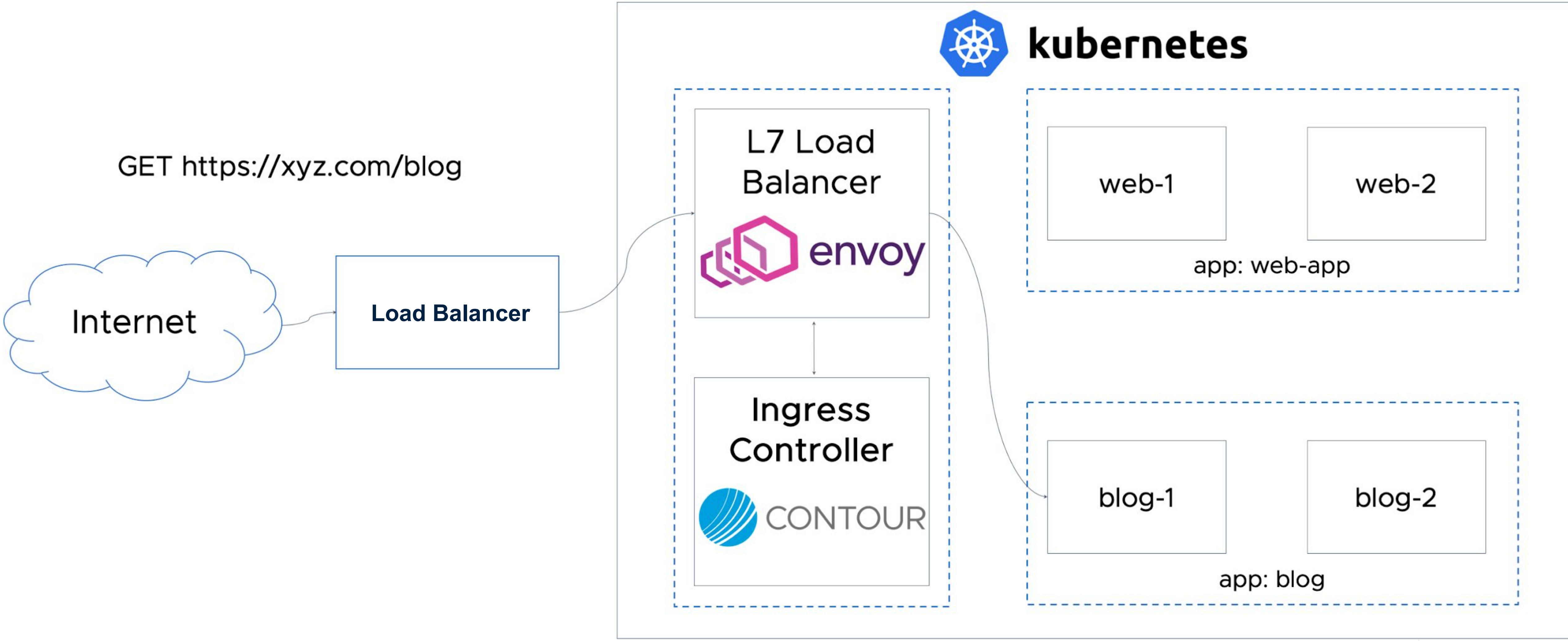
Contour Highlights

Deploy and manage Envoy as a load balancer

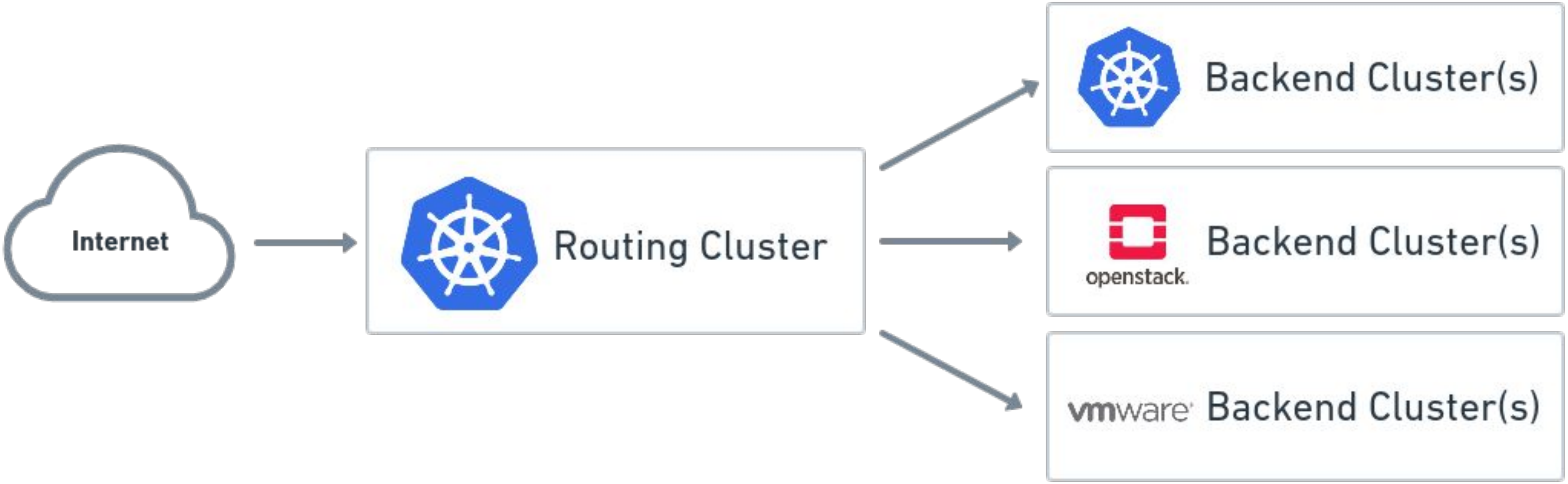
Kubernetes Ingress Controller that leverages Envoy as the data plane:

- **Dynamically updated** load balancing configuration without dropped connections
- Safely supports ingress in **multi-team Kubernetes clusters**
- Enables **delegation of routing configuration** for a path/header or domain to another Namespace
- Flexibly defines service weighting, load balancing strategies, and more **without annotations**

Contour Overview

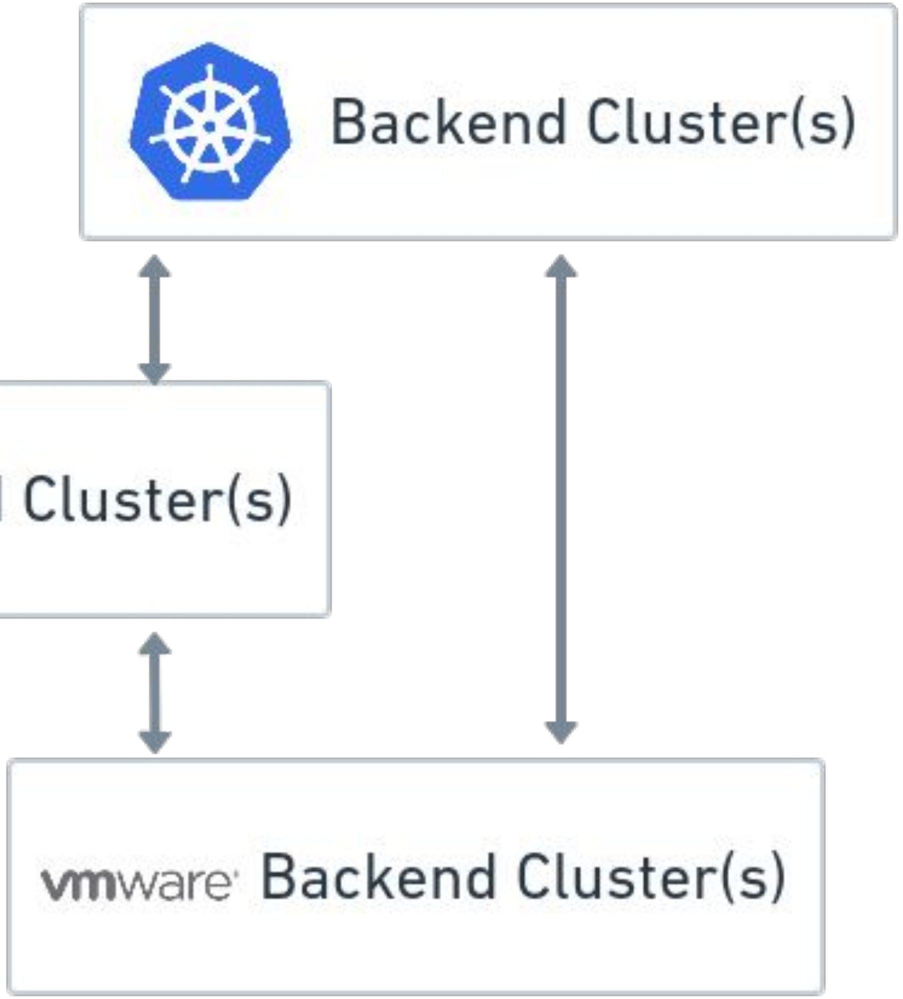


Routing Overview



What we want!

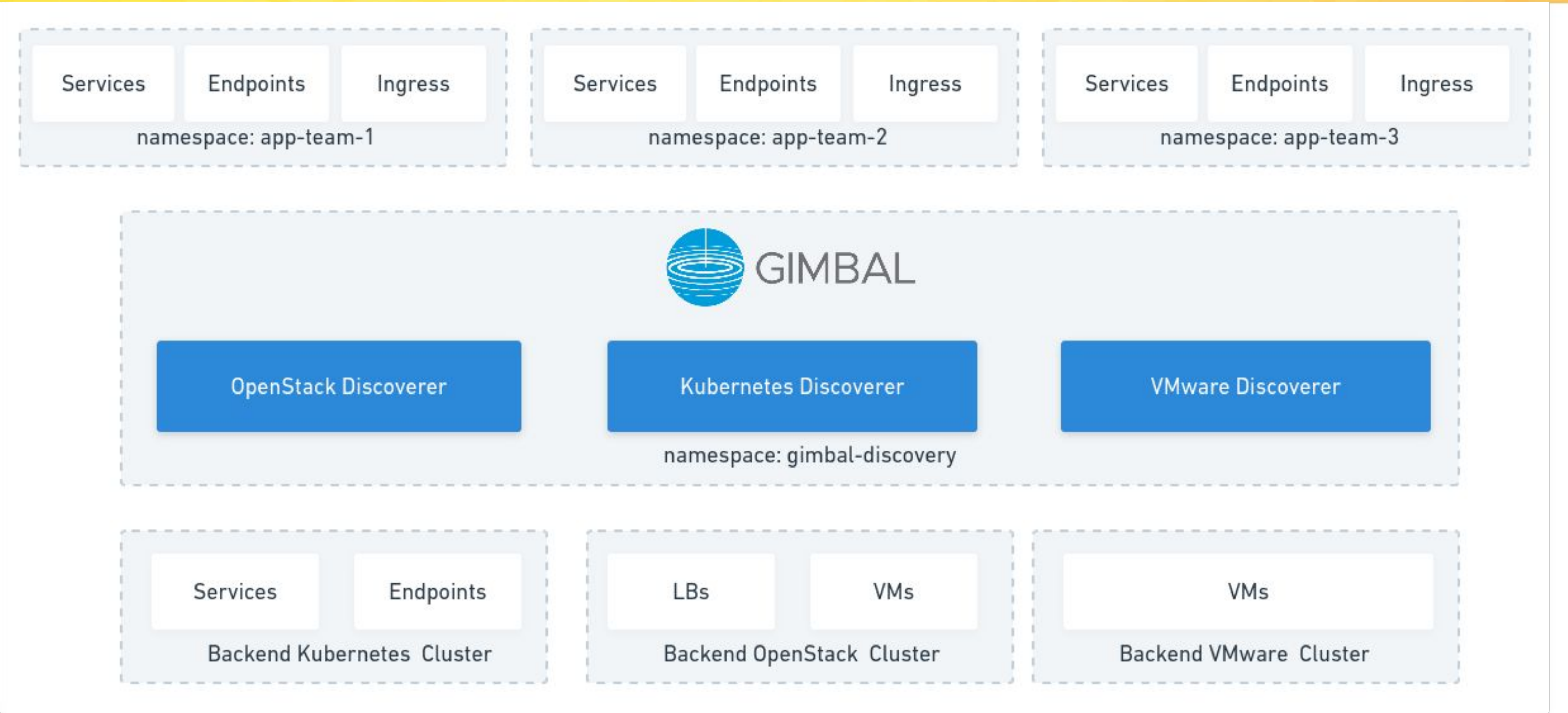
Something “meshy”



Contour + Gimbal

Route to Legacy VMs, bare metal, and Kubernetes

Contour Overview

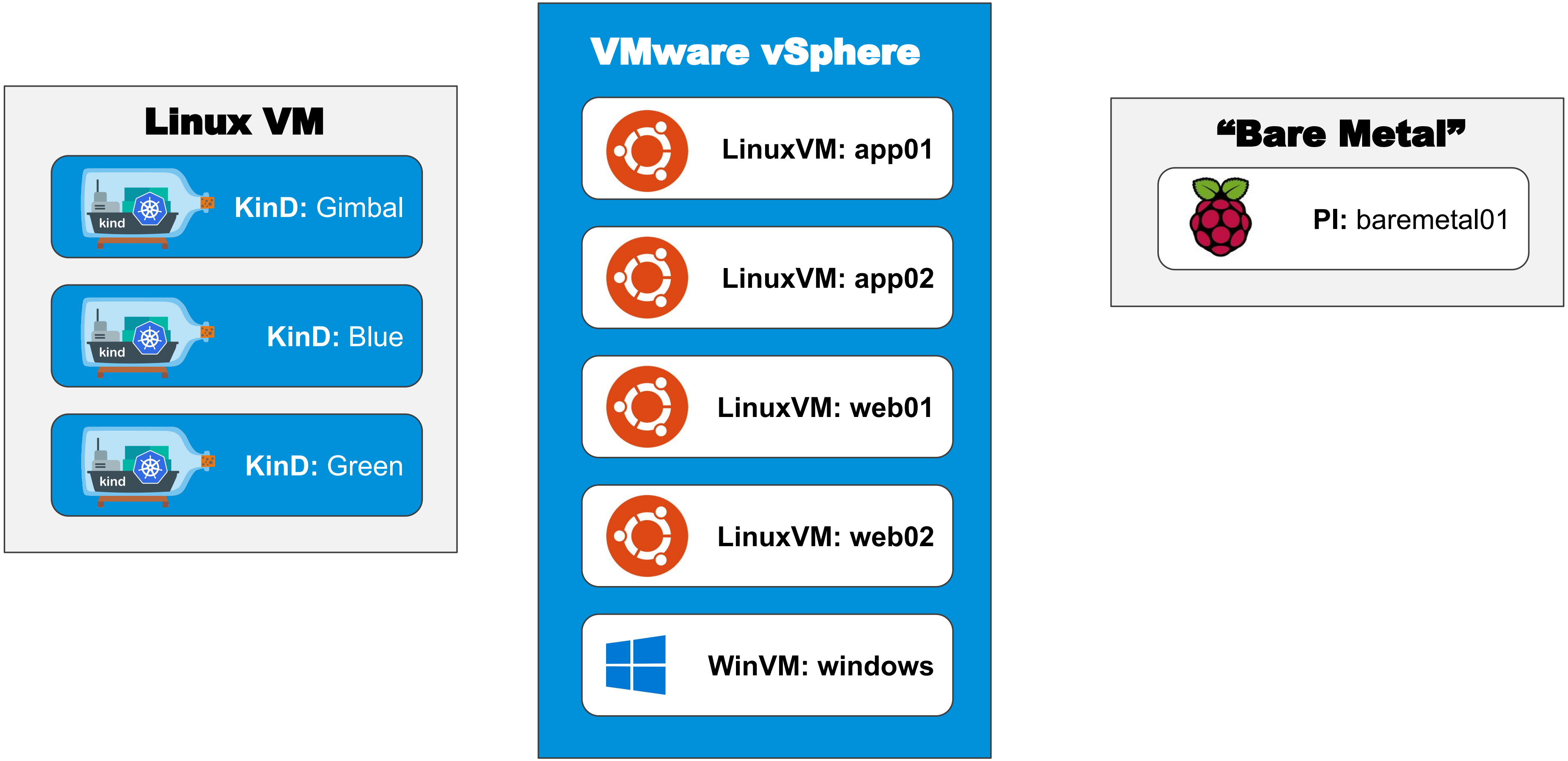

Services
Endpoints
Ingress
namespace: app-team-1
Services
Endpoints
Ingress
namespace: app-team-2
Services
Endpoints
Ingress
namespace: app-team-3
GIMBAL
OpenStack Discoverer
Kubernetes Discoverer
VMware Discoverer
namespace: gimbal-discovery
Services
Endpoints
Backend Kubernetes Cluster
LBs
VMs
Backend OpenStack Cluster
VMs
Backend VMware Cluster

TLDR

Contour is NOT a service mesh!

DEMO TIME!
... WHAT COULD POSSIBLY GO WRONG
38
makeameme.org

Demo environment



Linux VM

- **KinD:** Gimbal
- **KinD:** Blue
- **KinD:** Green

VMware vSphere

- **LinuxVM: app01**
- **LinuxVM: app02**
- **LinuxVM: web01**
- **LinuxVM: web02**
- **WinVM: windows**

“Bare Metal”

- **PI:** baremetal01

Resources

Where to learn more

How to get involved with the Contour Project

Learn more…..

Regular Work Group Meeting:

Community Meeting Tuesday 3pm PT, every 3 weeks

Documentation:

- projectcontour.io/

Slack:

- #contour on Kubernetes Slack

Let’s chat!

Meet the Maintainers @ 12:30pm today!

CONTOUR

Thank You

@stevesloka

@cantbewong

Good News!

Your legacy VMs and bare metal machines can join a service mesh.

Consume services Hosted on Kubernetes, AND expose services to consumers on Kubernetes

too much to demo complete steps today

but in deck so you can come back later

much of his is not currently well documented

Join VMs to an Istio service mesh (Mesh Expansion)

On your Istio Control Plane

During install

```
--set global.meshExpansion.enabled=true
```

Define the namespace the VM joins

```
export SERVICE_NAMESPACE="default"
```

Determine and store the IP address of the Istio ingress gateway since the mesh expansion machines access Citadel and Pilot through this IP address. Used in a later step.

```
export GWIP=$(kubectl get -n istio-system service istio-ingressgateway -o
jsonpath='{.status.loadBalancer.ingress[0].ip}')

echo $GWIP
```

Mesh Expansion

Compose a few files to be deployed to expansion VMs

Define Envoy intercept ranges in cluster.env file

Generate a **cluster.env** file. The script in the current documentation is GCE specific and will fail in a non-GCE deployment. This file contains the Kubernetes cluster IP address ranges to intercept and redirect via Envoy. It also contains the ports of used by services hosts on

```
ISTIO_SERVICE_CIDR=10.96.0.0/12
ISTIO_SYSTEM_NAMESPACE=istio-system
ISTIO_CP_AUTH=MUTUAL_TLS
ISTIO_INBOUND_PORTS=3306,8080
```

Mesh Expansion

Compose cert and key files to be deployed to expansion VMs

Extract the initial keys the service account needs to use on the VMs (to files to be copied to VMs).

```
kubectl -n $SERVICE_NAMESPACE get secret istio.default  \
    -o jsonpath='{.data.root-cert\.pem}' |base64 --decode > root-cert.pem

kubectl -n $SERVICE_NAMESPACE get secret istio.default  \
    -o jsonpath='{.data.key\.pem}' |base64 --decode > key.pem

kubectl -n $SERVICE_NAMESPACE get secret istio.default  \
      -o jsonpath='{.data.cert-chain\.pem}' |base64 --decode > cert-chain.pem
```

On each VM joining K8s mesh

Summary of steps

Fetch and install Istio sidecar package

Install certificate files

Install cluster CIDR definition

Check connectivity

On each VM joining K8s mesh

Details if you “try this at home”

From Kubernetes control plane host:

```
scp cluster.env root-cert.pem cert-chain.pem key.pem my-account@1my-vm:/tmp
```

From mesh expansion machine:

```
cd /tmp
curl -L https://storage.googleapis.com/istio-release/releases/1.3.5/deb/istio-sidecar.deb -o
istio-sidecar.deb
sudo dpkg -i istio-sidecar.deb
sudo mkdir -p /etc/certs
sudo cp {root-cert.pem,cert-chain.pem,key.pem} /etc/certs
sudo cp cluster.env /var/lib/istio/envoy
```

Transfer ownership of the files in /etc/certs/ and /var/lib/istio/envoy/ to the Istio proxy.

```
sudo chown -R istio-proxy /etc/certs /var/lib/istio/envoy
```

Legacy VM

Join an Istio Service Mesh

On your Istio Control Plane:

```
 --set global.meshExpansion.enabled=true
```

On your VM

Istio addresses must be in DNS

Mesh expansion VMs need to connect to citadel and pilot

```
echo “<GW_IP> istio-citadel istio-pilot istio-pilot.istio-system" | sudo tee
-a /etc/hosts
```

Verify the node agent works (note underscore):

```
sudo node_agent
```

```
“CSR is approved successfully. Will renew cert in 1079h59m59.84568493s”
```

Start Istio on VM using systemctl.

```
sudo systemctl start istio-auth-node-agent
sudo systemctl start istio
```

Verify Istio (envoy proxy) is running

```
sudo systemctl status istio
```

```
istio.service - istio-sidecar: The Istio sidecar
   Loaded: loaded (/lib/systemd/system/istio.service; disabled; vendor preset: e
   Active: active (running) since Wed 2019-07-17 22:38:05 UTC;
```

Run a service on mesh expansion machine

Certs and Envoy sidecar binary

On the VM, open a secondary command session (will be tied up running web server) and use python to start an HTTP web server

```
python -m SimpleHTTPServer 8080
```

What is a ServiceEntry?

enable service discovery for services on an expansion machine

You can add VM services to the mesh using a service entry.

Service entries let you manually add additional services to Pilot’s abstract model of the mesh.

Once VM services are part of the mesh’s abstract model, other services can find and direct traffic to them.

Each service entry configuration contains the IP addresses, ports, and appropriate labels of all VMs exposing a particular service

Define a ServiceEntry

enable service discovery for services on an expansion machine

```
kubectl -n ${SERVICE_NAMESPACE} apply -f - <<EOF
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: vmhttp
spec:
  hosts:
  - vmhttp.${SERVICE_NAMESPACE}.svc.cluster.local
  ports:
  - number: 8080
    name: http
    protocol: HTTP
  resolution: STATIC
  endpoints:
  - address: 192.168.99.129
    ports:
      http: 8080
    labels:
      app: vmhttp
      version: "v1“
EOF
```

Map the service

Using the Istio control plane, from the kubectl machine, not the VM

The workloads in a Kubernetes cluster need a mapping to resolve the domain names of VM services. To integrate the mapping, use istioctl to register and create a Kubernetes selector-less service:

```
istioctl  register -n ${SERVICE_NAMESPACE} vmhttp 192.168.99.129 8080
```

```
2019-07-17T22:39:40.551948Z info No pre existing exact matching ports list found, created new subset
{[{192.168.99.129  <nil> nil}] [] [{http 8080 }]}
```

```
2019-07-17T22:39:40.557364Z info Successfully updated vmhttp, now with 1 endpoints
```

Access the VM hosted service from Kubernetes

Using the Istio control plane, from the kubectl machine, not the VM

The Istio sample directory has a spec for a helper pod with curl installed that sleeps until needed. Deploy the pod in the Kubernetes cluster, and wait until it is ready:

```
kubectl apply -f samples/sleep/sleep.yaml

kubectl get pod

sleep-5fb55468cb-tkml7            2/2 Running 0 12s
```

We will use the pod to send a curl request from the sleep pod to the VM’s HTTP service - demonstrating that the VM hosted service is exposed to potential consumers in Kubernetes via service mesh :

```
kubectl exec -it sleep-5fb55468cb-tkml7 -c sleep -- curl
vmhttp.${SERVICE_NAMESPACE}.svc.cluster.local:8080
```

Sending requests from VM worldloads to Kubernetes

Using the Istio control plane, from a kubectl host, not the VM

On a machine managing Kubernetes cluster, get the virtual IP address (clusterIP) for the service:

```
kubectl get svc productpage -o jsonpath='{.spec.clusterIP}'
```

```
10.104.202.166
```

On the mesh expansion machine (VM), add the service name and virtual IP address (clusterIP) for the service to its etc/hosts file. You can then connect to the cluster service from the VM, as in the example below:

```
echo "10.104.202.166 productpage.default.svc.cluster.local" | sudo tee -a /etc/hosts

curl -v productpage.default.svc.cluster.local:9080
```

```
< HTTP/1.1 200 OK
< content-type: text/html; charset=utf-8
< content-length: 1836
< server: envoy
... html content ...
```

The “server: envoy” in the header indicates that envoy intercepted the traffic

Limitations

Istio mesh expansion has opportunities for enhancement

Envoy package is a .deb

.rpm has been done but not a release artifact at this time

no Windows support at this time