Service Mush
Debugging Istio Deployments

Sandeep Parikh
Google Cloud
Hi, I’m Sandeep

I write code, best practices, and work with technical practitioners (ops, devops, secops) to build and operate cloud native infrastructure.

Find me @crcsmnky on Twitter and Github.
Microservices introduce...challenges
Challenges

- More services to keep track of
- Network congestion
- Service reliability
- Inter and intra service security
- Aggregating metrics and logs
Istio
How does Istio help?

Examine **everything** happening with your services with little to no instrumentation.

Manage the flow of traffic **into, out of, and within** your complex deployments.

Secure **access and communications** between some or all services.
Great!

But....
Istio is not without complexity

Multiple **control plane** components ➔ each generate **lots of logs**

Large **config** model, multiple istio APIs ➔ steep learning curve

Istio policies are highly **customizable** ➔ many paths to a failing state

Istio sits on top of **Kubernetes** ➔ which itself is complicated
And Istio is still growing and evolving

Shipping **version 1.2**

**Documentation** is growing in size / reorganizing

**Tools** ecosystem is growing but small (lots of CLI basics like `curl` and `jq`)
What we’ll cover

Let’s walk through

- Traffic not routing correctly
- Missing telemetry data
- Authentication issues

Goal: share an Istio debugging toolbox with you, through demos.

How to **diagnose** and **fix** Istio configuration problems
Unified traffic management
Centralized policy checks + telemetry
Debugging traffic routing
Recap: how Istio Pilot works

- Observes the service **topology**
- Converts **Istio API resources** into Envoy config
- Pushes **Envoy config** to the sidecar proxies
Recap: Istio traffic API

North-South controls like Gateway and ServiceEntry affect inter-cluster traffic, inbound and outbound, respectively.

East-West controls like VirtualService and DestinationRule affect intra-cluster traffic, inbound and outbound, respectively.
Our demo app

- Weather frontend
- Weather backend v1
- Weather backend v2
<table>
<thead>
<tr>
<th>Location</th>
<th>Current Temp</th>
<th>Weather</th>
<th>Low</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Austin, TX, US</td>
<td>87.89°</td>
<td>Clouds</td>
<td>84.99°</td>
<td>91.0°</td>
</tr>
<tr>
<td>New York, NY, US</td>
<td>53.98°</td>
<td>Clear</td>
<td>51.01°</td>
<td>57.2°</td>
</tr>
<tr>
<td>Seattle, WA, US</td>
<td>63.21°</td>
<td>Clear</td>
<td>60.01°</td>
<td>66.0°</td>
</tr>
<tr>
<td>San Francisco, CA, US</td>
<td>60.17°</td>
<td>Clouds</td>
<td>55.99°</td>
<td>63.0°</td>
</tr>
</tbody>
</table>
Demo: VirtualService woes

Want this.

90%

weather backend v1

10%

weather frontend

weather backend v2

But seeing this.

50%

weather backend v1

50%

weather frontend

weather backend v2
Demo: what’s deployed?
<table>
<thead>
<tr>
<th>Service Name</th>
<th>Port</th>
<th>Protocol</th>
<th>Target</th>
<th>Type</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>istio-policy.istio-system.svc.cluster.local</td>
<td>15804</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>istio-policy.istio-system.svc.cluster.local</td>
<td>15814</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>istio-telemetry.istio-system.svc.cluster.local</td>
<td>9901</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>istio-telemetry.istio-system.svc.cluster.local</td>
<td>15814</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>istio-telemetry.istio-system.svc.cluster.local</td>
<td>15804</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jaeger-collector.istio-system.svc.cluster.local</td>
<td>14267</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jaeger-collector.istio-system.svc.cluster.local</td>
<td>14268</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jaeger-query.istio-system.svc.cluster.local</td>
<td>10666</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kiali.istio-system.svc.cluster.local</td>
<td>20081</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kube-dns.kube-system.svc.cluster.local</td>
<td>53</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kubernetes.default.svc.cluster.local</td>
<td>443</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>metadata.google.internal</td>
<td>88</td>
<td>outbound</td>
<td>ORIGINAL_DST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>metadata.google.internal</td>
<td>443</td>
<td>outbound</td>
<td>ORIGINAL_DST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>metrics-server.kube-system.svc.cluster.local</td>
<td>443</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mgmtCluster</td>
<td>15829</td>
<td>inbound</td>
<td>STATIC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>prometheus.istio-system.svc.cluster.local</td>
<td>9099</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>prometheus_stats</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sleep.default.svc.cluster.local</td>
<td>80</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tracing.istio-system.svc.cluster.local</td>
<td>80</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>weather-backend.default.svc.cluster.local</td>
<td>8000</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>weather-backend.default.svc.cluster.local</td>
<td>5080</td>
<td>outbound</td>
<td>MULTIPLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>weather-backend.default.svc.cluster.local</td>
<td>5080</td>
<td>outbound</td>
<td>SINGLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>weather-frontend.default.svc.cluster.local</td>
<td>5080</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>weather-frontend.default.svc.cluster.local</td>
<td>5080</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xds-grpc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>zipkin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>zipkin.istio-system.svc.cluster.local</td>
<td>9411</td>
<td>outbound</td>
<td>EDS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

/Users/paurikhs/Work/service-mush/traffic [git::cluster] [19:08]
Missing telemetry data
Recap: how Mixer works

Mixer has an open API and a **pluggable** architecture:

Send **telemetry**, **logs** and **traces** to your system of choice

Challenge: **many ways to fail**
Missing metrics

Standard Istio metrics are showing up in Prometheus (e.g. server request count). Custom workload metrics don’t appear. **Why not?**
In order for Prometheus to gather your custom metrics, you must supply annotations in your deployment’s spec.template.metadata that tell it where to grab them from.

```yaml
annotations:
  prometheus.io/scrape: "true"
  prometheus.io/port: "5000"
  prometheus.io/path: "~/metrics"
```
What about other Mixer issues?

Troubleshooting

- Confirming Mixer report calls
- Identifying Mixer configuration problems
- Examining Mixer logs
- Reviewing handler and metrics configurations
- Check out Missing Metrics on istio.io

Because Mixer components are tightly coupled, you may have to re-apply the configuration.

curl -L https://git.io/getLatestIstio | sh -

for flag in true false; do
  helm template
    --set mixer.enabled=$FLAG
    --namespace istio-system
    install/kubernetes/helm/istio
  > mixer-$FLAG.yaml
done

diff --line-format=\%L
  mixer-true.yaml mixer-false.yaml
  > mixer-config.yaml

kubectl apply -f mixer-config.yaml
Debugging security
Istio mTLS authentication

TLS handshake
secure naming check
connection established
Istio mTLS architecture

Create certs/keys for service accounts
Istio mTLS architecture

Service A

Citadel

New pods get cert / key
User enforces mTLS for Service A
Istio mTLS architecture

Pilot sends mTLS Policy to sidecar proxy
Demo: enforce mTLS for weather-backend

```yaml
apiVersion: ...  
kind: "Policy"  
metadata:  
  name: "weather-backend-mtls"  
spec:  
  targets:  
    - name: weather-backend  
peers:  
  - mtls: {}  
```

```yaml
apiVersion: ...  
kind: DestinationRule  
metadata:  
  name: "dr-weather-backend"  
spec:  
  host: "weather-backend.default"  
  trafficPolicy:  
    tls:  
      mode: ISTIO_MUTUAL
```
Demo: enforce mTLS for weather-backend

Want this.

But seeing this.

weather frontend

weather backend

weather frontend

weather backend

200

503
DestinationRule Conflicts

apiVersion: ...
kind: DestinationRule
metadata:
  name: "weather-backend"
spec:
  host: "weather-backend.default"
  trafficPolicy:
    tls:
      mode: DISABLE
subsets:
- name: single
  labels:
    version: single
- name: multiple
  labels:
    version: multiple

apiVersion: ...
kind: DestinationRule
metadata:
  name: "dr-weather-backend"
spec:
  host: "weather-backend.default"
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
subsets:
- name: single
  labels:
    version: single
- name: multiple
  labels:
    version: multiple

Ok
Conflict
Ok
Whew! We made it...
What we covered

How to determine if a **VirtualService** is working

How to use **istioctl**

How to parse **Envoy logs**

How to diagnose Mixer **rules** and **metrics**

How to diagnose Istio **mTLS** Policies
<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>istioctl</strong></td>
<td>Envoy status, TLS checks</td>
</tr>
<tr>
<td><strong>kubectl exec</strong></td>
<td>Mounted certs, Envoy debug logs</td>
</tr>
<tr>
<td><strong>stern</strong></td>
<td>Readability for k8s logs</td>
</tr>
<tr>
<td><strong>jq</strong></td>
<td>Read and filter JSON output</td>
</tr>
<tr>
<td><strong>sleep</strong></td>
<td>Pod for debugging east-west traffic</td>
</tr>
<tr>
<td><strong>curl</strong></td>
<td>Testing with HTTP requests</td>
</tr>
</tbody>
</table>

And don’t forget the [Istio docs](https://istio.io)!
Thank You

Questions, comments?

@crcsmnky

github.com/crcsmnky/service-mush