Making a Mesh: A Primer on Istio Running on KuBo

Ramiro Salas, Product Lead, Networking R&D @ Pivotal
It’s all about microservices…
Microservices’ expectations for the network

- Be available
- Be fast
- Be secure
- Have low latency
- Be transparent
- Never fail

*(but we all know better...)*
Network’s expectations for microservices

- Implement your own Service Discovery
- Implement your own app-level flow control
- Implement your own internal security
- Implement your own protection in case of network partitioning
Protocols Start to Emerge

But there is still need for:
- Encryption everywhere
- HTTP(s)/1.1
- Raw TCP
- Raw UDP
- Multi-ports

Unfortunately, this is not enough!
Microservices have more complex needs

- Service health checks
- Extensive metrics
- Efficient patterns like Circuit Breakers
- Higher-level flow controls and QoS
- Advanced Policy controls

And IP networks can’t really provide all that by themselves
30 years ago, we *would* have created entirely new network stacks to satisfy these needs
But we can’t. IP won. So we **augment** the network…
So instead of a direct communication model
We insert sidecars alongside every μS
(your new data plane)
And add a Control Plane of course…
And that is essentially a **Service Mesh**
The Market Today:

• **Data Planes:**
  Linkerd, Envoy, HAPerxy, Traefik, more…

• **Control Planes:**
  Istio, Synapse, Nelson

They *can* work together
Pivotal’s Interest is Primarily on Istio

- Platform agnostic
  - Already being progressively added to Cloud Foundry
- Backed by Google, IBM and Lyft
- Very lightweight choice of Sidecar (Envoy)
- Dynamic, vibrant community
- Emphasis on pluggability and extensibility
- Enabled for distributed tracing
- Intelligent traffic management
- Distributed security
Lyft’s Envoy

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

**Features**
- 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
North-South, and East-West

Outbound features:
- Service authentication
- Load balancing
- Retry and circuit breaker
- Fine-grained routing
- Telemetry

Inbound features:
- Service authentication
- Authorization
- Rate limits
- Load shedding
- Telemetry

HTTP/1.1, HTTP/2, gRPC, TCP with or without TLS
Control Plane Components

Control Plane API

- Pilot
- Mixer
- Istio-Auth

Pod

- Envoy
- svcA
  - Service A
- Envoy
- svcB
  - Service B

- Discovery & Config data to Envoys
- Policy checks, telemetry
- TLS certs to Envoy

Control flow during request processing
Mixer

- Mixer
  - Policy evaluation and telemetry
  - Precondition checking
  - Quotas & Rate Limiting
  - Primary point of extensibility
  - Enabler for platform mobility
  - Operator-focused configuration model
Pilot

- Pilot
  - Lifecycle of Envoy instances
  - API for Service Discovery
  - Load Balancing Pools
  - Routing Tables
Istio-Auth

- Strong service-to-service and end-user authentication using mutual TLS
- Built-in identity and credential management
- RBAC and authorization hooks in roadmap
Nice, but what about KuBo?
KuBo

• Invented to *delight* a person who creates Kubernetes clusters
• Vanilla K8s, and very up-to-date
• All controlled and managed by BOSH
• Strong isolation and multitenancy is achieved by hard separation

• Istio, like K8s, doesn’t do strong multitenancy very well
• So *Istio on K8s + KuBo == goodness*
Setting it up

1. Create a K8s cluster:
   
   ```
   ./bin/deploy_k8s ~/kubo-env/kubo my-cluster
   ```

2. Access your K8s environment:
   
   ```
   ./bin/set_kubeconfig ~/kubo-env/kubo my-cluster
   ```

3. Get your Istio release:
   
   ```
   curl -L https://git.io/getLatestIstio | sh -
   ```

4. Setup mTLS for all sidecars and automatic injection:
   
   ```
   kubectl apply -f install/kubernetes/istio-auth.yaml
   kubectl apply -f \
   install/kubernetes/istio-initializer.yaml
   ```
Verify Istio is installed and running

```bash
$ kubectl get svc -n istio-system
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>istio-egress</td>
<td>10.83.247.89</td>
<td>&lt;none&gt;</td>
<td>80/TCP</td>
<td>5h</td>
</tr>
<tr>
<td>istio-ingress</td>
<td>10.83.245.171</td>
<td>35.184.245.62</td>
<td>80:32730/TCP,443:30574/TCP</td>
<td>5h</td>
</tr>
<tr>
<td>istio-pilot</td>
<td>10.83.251.173</td>
<td>&lt;none&gt;</td>
<td>8080/TCP,8081/TCP</td>
<td>5h</td>
</tr>
<tr>
<td>istio-mixer</td>
<td>10.83.244.253</td>
<td>&lt;none&gt;</td>
<td>9091/TCP,9094/TCP,42422/TCP</td>
<td>5h</td>
</tr>
</tbody>
</table>
Deploy your app

$ kubectl create -f <your-app-spec>.yaml

OR

$ kubectl create -f <(istioctl kube-inject -f <your-app-spec>.yaml)

Only needed if you didn’t install the Istio initializer
The Future is Great
KuBo, create me a K8s cluster with Istio pre-installed in my local vSphere environment

now, use ODSB and create me a BOSH-managed Postgres cluster fronted by an Envoy proxy for my own use, and make it part of my mesh.

And while you are at it, extend the mesh to Org Foo and Spaces Bar and Baz in my development CF foundation in my Google Cloud, using my credentials

Now create me a new K8s cluster in my AWS environment and extend The Istio mesh there.
The Need to Adapt

• SDN vendors will have to up their game
  • The network fabric will always have a crucial role but once the intelligence moves to L7, their role will mostly be to protect the mesh
• Orchestrators will also have to up their game, since essentially most of them will look the same as Istio becomes more intelligent.
• Envoy is great, but there may be a need for more specialized proxies
  • SCTP for NFV in Telecom
  • Native MQTT or CoAP for IoT
A Sense of Realism

• Istio is a very young project, still untested at a very large scale
• Best practices for HA in all conditions still need to be created
• No federation capabilities at the moment
• Integration and cooperation with lower network layers still dependent on orchestrator specifics and SDN plugins
• No opinions about end-to-end policies provides flexibility, but can create complex designs down the road
• Vendors will need to find balance between open source and proprietary extensions
• Usage of istiocli still very explicit and deliberate.
Thanks

• To Eric Johnson and Mandar Jog from Google for their contributions to this talk.

• Watch:
  • Adding Sidecars to Cloud Foundry - Gabriel Rosenhouse & Angela Chin, Pivotal
  • Project Office Hours: Routing - Shannon Coen & Gabriel Rosenhouse
  • Getting a Handle on Your Microservices: Istio and the Open Source Broker API - Christopher M Luciano & Morgan Bauer, IBM