Service Mesh with Enhanced Observability and Traceability

Presenter: Veer Muchandi
Company: Red Hat Inc
Title: Chief Architect - Container Solutions, NA Commercial
Social Handle: @VeerMuchandi
Blogs: https://blog.openshift.com/author/veermuchandi/
Session Goals

- Understanding Service Mesh
- Reasons for Service Mesh on a Kubernetes Cluster
- Kiali and Jaeger
- Demos of enhanced Observability and Traceability
Calls with Distributed Computing
World of Microservices

100s of Microservices running on cluster

Issues to address:
- Load balancing
- Network Faults
- Circuit Breakers
- Service Discovery
- QOS & SLA
and more..

Requires “Plumbing Code”
Current Solutions

BL - Business Logic

PL - Plumbing Logic to solve the infrastructure issues such as Service discovery, load balancing, fault-tolerance, rate limiting, QoS etc.

Examples: Frameworks such as NetflixOSS
Issues with libraries/frameworks embedded in code

Developers have to worry about the plumbing code

- Code intrusive
- Learning curve for such frameworks
- Not language-agnostic; hampers polyglot microservices
- Maintenance overhead
  - thousands of services using version of libraries
  - updates to infra libraries require integration, testing and re-deployment of all services
Sidecar Proxy

- Intercepts all network communication between microservices
- Encapsulates Service Infrastructure code
- Application code (business logic) unaware of Sidecar proxy
- Examples - Linkerd, Envoy

SideCar Proxy
Service Mesh is a dedicated infrastructure layer to handle service-service communications. Typically implemented as an array of lightweight network proxies deployed alongside application code. Interconnected Proxies form a mesh network.
Service Mesh Implementation

Control Plane

Implementation of mesh network involves a “Control Plane”

Proxies managed by centralized Control Plane
Kubernetes/OpenShift Pods

Pods support multiple containers. Sidecars are naturally available.

Pods scale and are typically front-ended by a Kubernetes Service. Service Discovery is built inside K8S using SkyDNS.
Load balancing, Traffic Routing in OpenShift vs K8S

OpenShift provides routing as ingress into the cluster which does load balancing.

Also implements patterns such as blue/green and A/B Testing via Router.

Whereas, Kubernetes depends on an external load balancer.
For Traffic Splitting, you can emulate it by changing the pod proportions behind services.
While Kubernetes goes to an extent and OpenShift a little more in terms of the handling infrastructure needs such as service discovery, load balancing and some request routing, we need more OOTB features -

- Content based routing
- Canary, AB deployments
- Rate Limiting
- End-to-End access control
- Fault tolerance and fault injection
- Routing, Ingress and Egress rules
- Circuit Breakers
- Integration with tools for logging, monitoring, quotas, ACLs and more
Welcome to Istio

Istio implements Service Mesh and adds

- **Traffic Management:** Load balancing, Failure Recovery, Circuit breakers etc
- **Observability:** Monitoring, Metrics, Tracing etc
- **Policy Enforcement:** Routing Rules, Ingress rules, Egress Rules, Canary, Blue/Green, A/B, Rate Limiting etc
- **Service Identity, Security:** verifiable identity to services, Access control, End-to-end authentication etc

Currently built for Kubernetes and other platforms such as Mesos coming up

Integrates with existing solutions for ACL, Logging, Monitoring, Quotas, Auditing etc.
Istio Architecture
BookInfo Control Flow
Demos
### Create Weighted Routing

<table>
<thead>
<tr>
<th>WORKLOAD</th>
<th>TRAFFIC WEIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>discounts-v1</td>
<td><img src="##" alt="Slider" /> - 90 + %</td>
</tr>
<tr>
<td>discounts-v2</td>
<td><img src="##" alt="Slider" /> - 10 + %</td>
</tr>
</tbody>
</table>

- Evenly distribute traffic
Thank you