Evolution of application architecture

... How did we get to service mesh?
Monolith application

Single unit of executable
= Application
= Single process
Application modules

- Alerts
- Handle HTTP requests
- UI
- Data processing

Application
Multiple processes

- Alerts
- UI
- Application
- Handle HTTP requests
- Data processing
Microservices

Language agnostic

Scaled separately

Upgraded separately
A shift in Application Packaging and Runtime
Containerizing an app
Run multiple containers
Orchestrate containers

- Run many containers on multiple hosts
- Scale - manage several instances (replicas) of the same container
- Manage a container based environment
Container orchestration platforms

- Kubernetes
- OKD
  (Openshift)
Kubernetes building blocks (some...)

- Pod - a group of one or more containers, with shared storage/network
- Deployment - manages pod definition and defines replicas of pods
- Service - an abstraction, an access point to a set of Pods
  - Sometimes called a microservice
Microservices - the Kubernetes way

Access point = microservice

Service A

Container

Pod

Service B

Code

Container

Pod

Replica 1

Replica 2

Replica 3

Access point = microservice
High Complexity
Multiple points of failure
Challenges

• How are the requests routed between services?
• How do I detect failures and downtime?
• How to upgrade and test new versions of a service?
• Securing the communication
Service mesh to the rescue
What is a service mesh

- Infrastructure/framework that handles communication between services
- Often implemented as network proxies deployed alongside the microservices
Istio - Ιστίο

Open source service mesh
The dry facts

- Started in May 2017
- Means “sail” in Greek
- Developed in Go
Istio features

- Load balancing (HTTP, gRPC, TCP...)
- Traffic control (routing rules, retries, timeouts, fault injection, mirroring)
- Secure service-to-service communication
- Access controls (authorization)
- Metrics and traces for traffic
Important Terminology

● Workload - anything owning/controlling pods (like a Deployment) or the pods themselves
● Service - a microservice
● Application - *label* “app” on a pod/service
● Version - *label* “version” on a pod/service
Before Istio

POD A

- Routing code
- Circuit breaker code
- Business logic code

Container

POD B

- Routing code2
- Circuit breaker code2
- Business logic code2

Container2
Sidecar Proxy

- A proxy is deployed in a container next to each instance of microservice (inside a pod)
- Container name: `istio-proxy`
- It is **transparent** to application code
- Envoy open source proxy is currently used
How is the sidecar injected?

- Manually
- Automatically injected to pod on creation
  - `kubectl label namespace default istio-injection=enabled`
  - Mutating Admission Webhook is used for sidecar injection
  - Actually… 2 containers are injected: istio-init and istio-proxy
Sidecar Proxy in Istio and Kubernetes

Before Istio, no sidecar

With sidecar
With Istio - sidecar intercepts all traffic

Configuration is transparent to the services and not part of the code
Istio routing in Kubernetes

Communication is “Envoy to Envoy” bypassing the Kubernetes Service
Different routing scenarios

- A/B testing
- Traffic shifting
  - Canary deployment (an example of traffic shifting)
- Mirroring traffic
Weighted Routing with Istio - A/B

Proportion of traffic routed to a version is independent of number of instances of that version.
Weighted Routing - Canary

Proportion of traffic routed to a version is independent of number of instances of that version
Matching Routing with Istio

Service A

Pod

Service B

Pod Version 1

All users

User Alissa

Pod Version 2
Mirroring traffic

Service A

Pod

Service B

Pod Version 1

Pod Version 2

Real traffic

Copy of traffic

Response disregarded
“Anything that can go wrong will go wrong”

(Murphy’s law)
KEEP CALM AND DO CHAOS ENGINEERING
Chaos engineering with Istio

- **Inject delays**
  - Simulate network latency
  - Simulate an overloaded service

- **Define aborts (Inject Errors)**
  - Simulate failure in a service (return a predefined HTTP Error)
  - A good alternative for a manual shutdown or “scale to zero”
Inject delay

Add 7 seconds delay to response

Service A

Pod

Service B

Pod Version 1
Instance 1

Pod Version 2
Instance 2
Inject Error

Service A

Pod

Service B

Pod

Version 1
Instance 1

Version 2
Instance 2

Work as usual for all the users

Return Error 500 for user Alissa

Pod

Version 2
Circuit breaker

- Set a connection pool to limit connections and requests
- **Example:** “Set a connection pool of 100 connections with no more than 10 req/connection to service A”
Outlier detection

- Classify instances as healthy/unhealthy
- Eject unhealthy instances for a defined timeframe which can be increased over time
- **Example:** “Scan all pods every 5 mins, any instance that fails 7 consecutive times with 5XX error code will be ejected for 15 minutes.”
Authorization and Authentication

● Authentication
  ○ End user authentication (JSON Web Token (JWT))
  ○ Service to service authentication (mutual TLS)
    ■ Permissive mode is possible for flexible migration

● Authorization
  ○ Can service <A> send <this request> to service <B>?
  ○ Roles are visible across namespaces
  ○ ServiceRole and ServiceRoleBinding
Security

● Defining a Gateway ingress/egress to enable traffic in/out of mesh

● Citadel monitors service accounts creation and creates a certificate for them
  ○ Certificates only in memory, sent to Envoy via SDS API

● mTLS can be defined on multiple levels
  ○ Client and server exchange certificates, 2 way
  ○ All mesh, specific service, etc.
Configuration objects

• **VirtualService** != Kubernetes service
  - Rules for how requests to a service are routed within service mesh
  - Routing logic, load weighting, chaos injection

• **DestinationRule**
  - Configures policies to be applied to a request after VirtualService routing has occurred
  - Load balancer, circuit breaker

• **MeshPolicy, Gateway, ServiceEntry and more...**
All Istio objects are CRD (CustomResource Definition)

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
    - route:
      - destination:
        host: reviews
        subset: v1
        weight: 50
      - destination:
        host: reviews
        subset: v2
        weight: 25
      - destination:
        host: reviews
        subset: v3
        weight: 25
```
New set of challenges

- How many versions exist for service A?
- Is there any traffic now?
- Is routing configured for service B?
- Is my configuration valid?
- Is security on?
- Is the app healthy?
Kiali - Κιάλι

Open source
Istio service mesh observability
Dry facts

- Started in January 2018
- Means “spyglass” or “monocular” in Greek
- Developed in Go and React
Kiali Features

- Visualize mesh connections and traffic
- Service and application health
- Configure routing via UI
- Validate Istio configurations
- View metrics, traces and logs
- Visualize security configuration
A picture is worth a thousand yamls
Demos based on Bookinfo example
Let’s see Kiali in action

- Mesh visualization
- Fault Injection
- Configuration Validation
- Configure routing rules
- Tracing
- Traffic stats
Bookinfo example

Istio Service Mesh

Requests → Ingress Envoy → Product page → Reviews-v1

Reviews-v2 → Reviews-v3 → Ratings v1

Ratings v2

MongoDB instance

Ruby
Bookinfo on Kiali
Kiali Features
Overview page
Mesh Topology Graph
Hide and Seek
### Details Page

#### Overview

**Labels**
- `app` `details` `service` `details`
- **Type**: ClusterIP
- **IP**: 172.30.23.185
- **Created at**: 5/23/2019, 11:33:02 AM
- **Resource Version**: 44630

**Ports**
- **TCP** http (9080)

**Endpoints**
- 172.17.0.21: details-v1-
- 74c4f8c9bbf-rt68c

**Health**
- **Status**: Healthy
- **Error Rate over last 1m**: 0.00%

#### Workloads

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Labels</th>
<th>Created at</th>
<th>Resource version</th>
</tr>
</thead>
<tbody>
<tr>
<td>details-v1</td>
<td>Deployment</td>
<td><code>app</code> <code>details</code> <code>version</code> <code>v1</code></td>
<td>5/23/2019, 11:33:02 AM</td>
<td>81259</td>
</tr>
</tbody>
</table>
Viewing Logs
Runtime metric dashboards
Weighted Routing
Configuration validations
Tracing (integration with Jaeger)
Visualizing security
Connect with the community

Kiali.io

Istio.io

KialiProject

IstioMesh

github.com/kiali

github.com/istio
Icon credits

- Twitter by Lubos Volkov, the Noun Project
- **Light Bulb** by artworkbean, the Noun Project
- **Magnifying Glass** by Musket from the Noun Project
- **Questions** by Rediffusion from the Noun Project
- **Mug** by Alex Getty from the Noun Project
- **Diamond** by MarkieAnn Packer from the Noun Project
- **Box** by Cornelius Danger from the Noun Project
Thank you!

mikeyteva
Introduction to service mesh with Istio and Kiali

Alissa Bonas

mikeyteva