Introducing AWS App Mesh

Shubha Rao
Sr. Product Manager
AWS

Tony Pujals
Sr. Developer Advocate
AWS
Introducing AWS App Mesh

Visibility and control for microservices on AWS

App Mesh is a service mesh
Agenda

A lot of whys ...
- Why Microservices
- Why Service Mesh
- Why Service Mesh Control Plane
- Why App Mesh

And some hows ...
- What is it
- Features
- Roadmap
- Representing my application to use App Mesh
- Demo
Similar to

Chef-special boutique restaurant

Franchise
Similar to

**Chef-special boutique restaurant**

- Innovation slow and gradual—relies on chef brand/recognition
- Slow evolution—some succeed, some fail

**Franchise**

- Preset guidelines on menu options and preparation
- Faster rollouts once success is proven
Microservices increase release agility
Monolith
Best practices are emerging and evolving rapidly

Standardize and automate operations by **modeling infrastructure as code**

Improve application performance with **full stack observability**

Create a culture of innovation by **organizing into small DevOps teams**

Update applications and infrastructure quickly by **automating CI/CD**

Ensure trust by **automating security and compliance**
What is needed

- Consistent communications management
- Complete visibility
- Failure isolation and protection
- Fine-grained deployment controls
Implementation options

Option 1

In-process (SDK)

Microservice Container

Option 2

Out-of-process (sidecar proxy)

Microservice Container

Proxy
Option 1: in-process SDK

- **SDK maintenance**
- **Application code changes**
- **Consistency across services**

**Languages**
- Java
- Scala
- Node.js
- Python
- C++
- Django
- .NET
- GO
- ...

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Implementation options

**Option 1**
- In-process (SDK)
- Microservice Container

**Option 2**
- Microservice Container
- Out-of-process (sidecar proxy)
- Proxy
Option 2: side-car proxy

Proxy

- Monitoring
- Routing
- Discovery
- Deployment

Microservice

Application Code
Option 2: side-car proxy

Task or Pod

Application Code

External traffic

Proxy runs as a container
Option 2: side-car proxy

- Decouples install/upgrade
- Configurable—separates business logic from operations
- Minimizes inconsistencies
Why service mesh proxy
vs. libraries or app code

Reduce work required by developers
Follow best practices
Use any language or platform
Simplify visibility, troubleshooting, and deployments

Overall—migrate to microservices safer and faster
App Mesh uses Envoy proxy

- OSS project
- Wide community support, numerous integrations
- Stable and production-proven
- “Graduated Project” in Cloud Native Computing Foundation
- Started at Lyft in 2016
How do we tell every proxy what to do?
Configuring lots of proxies is hard!
Easily deliver configuration and receive data.
Dynamic state changes impact proxy configuration
Why service mesh control plane
vs. static config or self-built control plane

Don’t need to spend dev to build and ops to maintain

Not tied to application deployment system (e.g., container orchestration)

Works across different compute systems

Reliably store and distribute configuration
Additional use cases based on customer input vs. other control plane solutions

Works across Clusters
Container services
Containers & VMs

AWS run for scale and stability

AWS and partner integrations

Extensible architecture from OSS base
Dynamic state changes impact proxy configuration

App Mesh API

AWS App Mesh

Integrations

Deployments

Scaling

Health

Integrations

envoy

App developer

Infra operator

App developer

App developer
App Mesh configures every proxy
App Mesh configures every proxy
Today App Mesh is available as a preview for all customers

Observability and traffic control

- Easily export logs, metrics, and traces (available)
- Client side traffic policies—circuit breaking, retries (coming soon)
- Routes for deployments (available)

AWS container services compatibility

**Today:** Amazon Elastic Container Service (Amazon ECS) & Amazon Elastic Container Service for Kubernetes (Amazon EKS)
**Coming soon:** AWS Fargate
Regions

US West
Oregon (US-West-2)

US East
N. Virginia (US-East-1)
Ohio (US-East-2)

Europe
Ireland (EU-West-1)
Application observability

- Faster troubleshooting due to consistent data across services
- Existing tools or dashboards with a lot more metrics, logs and traces
- Distinguish between service and network issues
Proactive operations helps mitigate issues

Latency

Time (ms)

Outage

Degraded state
Client side traffic management

Traffic shaping*
- Service discovery
- Retries
- Timeouts
- Circuit breaks
- Health checks

Routing controls
- Protocols support
- Header based
- Cookie based
- Path based
- Host based
Fine-grained deployment control

Can do this now...
Fine-grained deployment control
## Preview v/s GA (capabilities)

<table>
<thead>
<tr>
<th>Preview</th>
<th>GA</th>
<th>Post GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>API ready</td>
<td>Console Integrations</td>
<td>TLS Ingress</td>
</tr>
<tr>
<td>For use with sample apps not production</td>
<td>Traffic management</td>
<td></td>
</tr>
<tr>
<td>HTTP path based routing</td>
<td>AWS Cloud Map</td>
<td></td>
</tr>
<tr>
<td>Statd based logs, metrics integrations</td>
<td>Cross account</td>
<td></td>
</tr>
<tr>
<td>Amazon EC2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
AWS integrations

AWS

Cloud Map

Service Discovery

Microservices
How it works
Representing your sample app in App Mesh

Mesh-{myapp}

Virtual Node A
- Listener
- Service Discovery
- Backends

Virtual Node B
- Listener
- Service Discovery
- Backends

Elastic Load Balancing

Microservices

App Mesh
Virtual Node

Listeners
Policies to handle incoming traffic
Ed: port, Health check*, Circuit breaker*, Retries*

Service Discovery
Describes how its callers locate this node
(DNS hostname or AWS Cloud Map* namespace, serviced, and selectors)

Backends
Set of destinations that this node will communicate with (hostnames)
Create route

Microservices

App Mesh

Elastic Load Balancing

Mesh:[myapp]

Virtual Node A

Virtual Node B

Virtual Router

Routes
Virtual router

Virtual Router: B

Service names

Service name

Names that clients will use to connect to the service
Virtual routes

Destination’s virtual router and route

Virtual router: B

Route Name: B1
HTTP routes
Match Prefix: /
Action: Targets
B

Route Name: B2
Other Protocol routes
Match
Action:

Route B
Virtual node destination + weight
Update routes

Virtual router: B

HTTP route
prefix: /

targets:

Route B
Virtual node destination + weight

Route B’
New service or service version
Mesh—[myapp]

Virtual Node A
- Listener
- Service Discovery
- Backend

Virtual router
- Domains
- match:/
- action:
  - B
  - B'

Virtual Node B
- Listener
- Service Discovery
- Backends

Service B
- Virtual Node Node B
- Listener
- Service Discovery
- Backends

Service B'
- Virtual Node Node B'
- Listener
- Service Discovery
- Backends
Mesh—myapp

Envoy Bootstrap:

//"tracing": {...},
//"stats": {...},
//"logging": {...}
Getting started

Product overview
https://aws.amazon.com/app-mesh

Documentation
https://docs.aws.amazon.com/app-mesh/index.html

Examples
https://github.com/awslabs/aws-app-mesh-examples

Issues & Roadmap
https://github.com/awslabs/aws-app-mesh-examples/issues
Thank you!

Shubha Rao
Sr. Product Manager
AWS

Tony Pujals
Sr. Developer Advocate
AWS
Please complete the session survey in the mobile app.