Observability & Debugging Microservices with Linkerd and Conduit

Franziska von der Goltz
Software Engineer at Buoyant
@franziskagoltz

BUOYANT

About me

Software Engineer at Buoyant

Twitter: @franziskagoltz

I work on service meshes!

Debugging Microservices is Hard

— Application logic is split across service boundaries

@franziskagoltz

Debugging Microservices is Hard

– Application logic is split across service boundaries

– Services are owned by different teams

@franziskagoltz

Debugging Microservices is Hard

- Application logic is split across service boundaries
- Services are owned by different teams
- Services are comprised of many running instances

Debugging Microservices is Hard

- Application logic is split across service boundaries
- Services are owned by different teams
- Services are comprised of many running instances
- Instances are potentially being rescheduled any moment

Debugging Microservices is Hard

— Application logic is split across service boundaries

— Services are owned by different teams

— Services are comprised of many running instances

— Instances are potentially being rescheduled any moment

As a result, tools that used to work for monoliths break down, and we see a new set of tools.

@franziskagoltz

Service Mesh

What is a Service Mesh?

— Dedicated infrastructure layer for adding reliability, security, visibility to a cloud native application.

What is a Service Mesh?

— Dedicated infrastructure layer for adding reliability, security, visibility to a cloud native application.

— How? By managing all runtime service communication.

@franziskagoltz

What is a Service Mesh?

— Dedicated infrastructure layer for adding reliability, security, visibility to a cloud native application.

— How? By managing all runtime service communication.

— Provides: circuit breaking, requesting routing, security policy, etc.

What is a Service Mesh?

— Dedicated infrastructure layer for adding reliability, security, visibility to a cloud native application.

— How? By managing all runtime service communication.

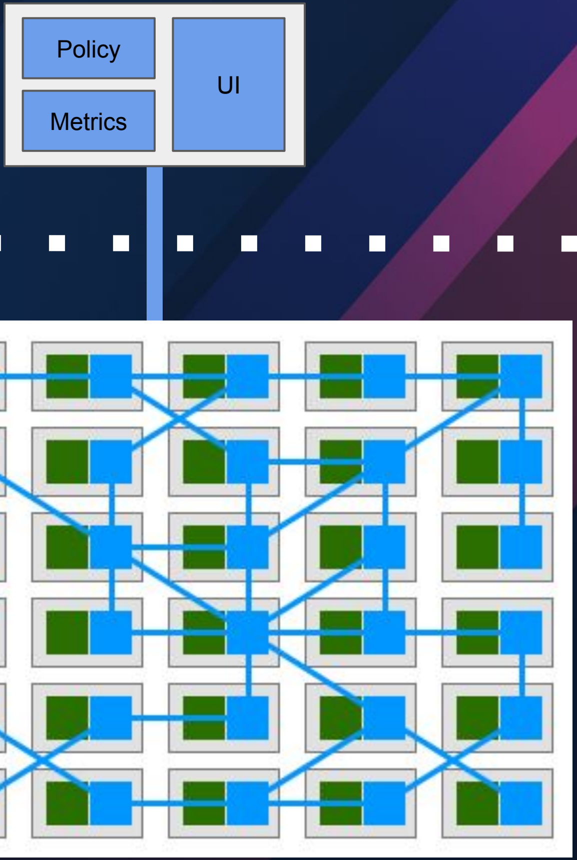
— Provides: circuit breaking, requesting routing, security policy, etc.

Today’s focus: **debugging**.

@franziskagoltz

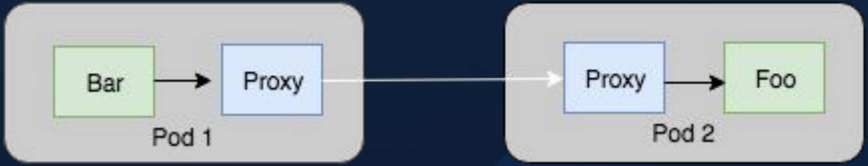
Control plane

Data plane



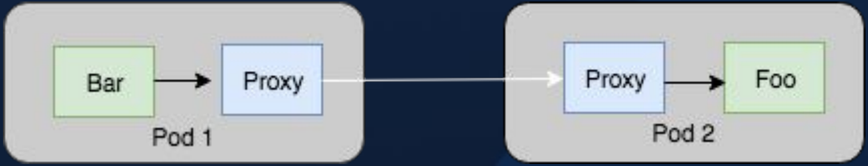
@franziskagoltz

Example service meshed request



@franziskagoltz

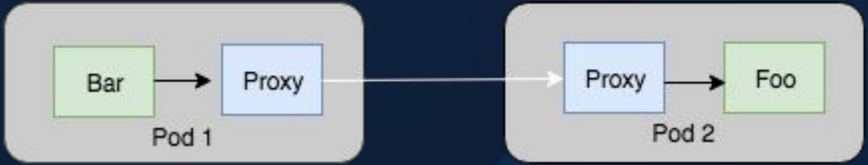
Example service meshed request



The proxies knows:

- How long the request took *(latency)*

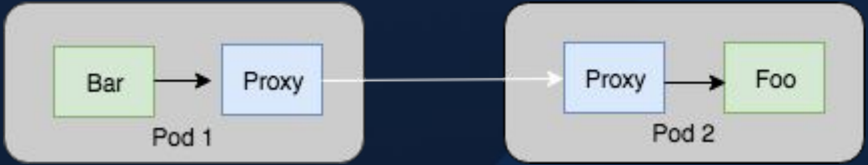
Example service meshed request



The proxies knows:

- How long the request took *(latency)*
- Whether it succeeded or failed *(success rate)*

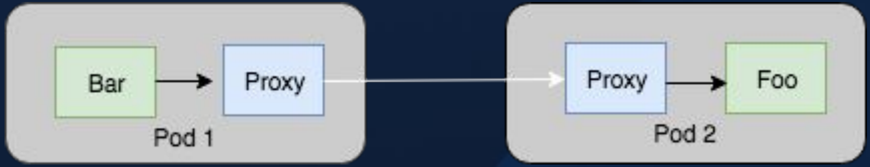
Example service meshed request



The proxies knows:

- How long the request took *(latency)*
- Whether it succeeded or failed *(success rate)*
- How many of these requests happened recently *(volume)*

Example service meshed request

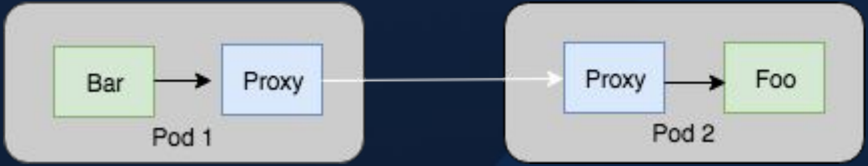


The proxies knows:

- How long the request took *(latency)*
- Whether it succeeded or failed *(success rate)*
- How many of these requests happened recently *(volume)*
- Who sent the request, and who received it *(identity)*

@franziskagoltz

Example service meshed request



The proxies knows:

- How long the request took *(latency)*
- Whether it succeeded or failed *(success rate)*
- How many of these requests happened recently *(volume)*
- Who sent the request, and who received it *(identity)*
- Lots more! (was it retried, which instance it went to, etc.)

How does a service mesh help with debugging?

Aggregating this data allows the mesh to paint a detailed picture of your application!

@franziskagoltz

How does a service mesh help with debugging?

Aggregating this data allows the mesh to paint a detailed picture of your application!

Top-line metrics:

- Success rate
- Latency distribution
- Request volume

How does a service mesh help with debugging?

Aggregating this data allows the mesh to paint a detailed picture of your application!

Top-line metrics:

- Success rate
- Latency distribution
- Request volume

Tied to identity, including caller/callee relationships!

Service mesh debugging

This is a fundamentally different class of diagnostics information.

@franziskagoltz

Service mesh debugging

This is a fundamentally different class of diagnostics information.

— What’s the success rate of Foo?

Service mesh debugging

This is a fundamentally different class of diagnostics information.

— What's the success rate of Foo?

— Which services call Foo?

@franziskagoltz

Service mesh debugging

This is a fundamentally different class of diagnostics information.

- What’s the success rate of Foo?
- Which services call Foo?
- What’s the success rate of Foo when called by Bar?

Service mesh debugging

This is a fundamentally different class of diagnostics information.

- What’s the success rate of Foo?
- Which services call Foo?
- What’s the success rate of Foo when called by Bar?
- What’s the request volume, latency distribution, ...

@franziskagoltz

Service mesh debugging

This is a fundamentally different class of diagnostics information.

- What's the success rate of Foo?
- Which services call Foo?
- What's the success rate of Foo when called by Bar?
- What's the request volume, latency distribution, ...

All *without* having to modify our applications.

CONDUIT

@franziskagoltz

Conduit

- Open source service mesh (Apache v2)
- Built from the ground up for Kubernetes
- Ultralight, ultrafast
- Data plane: tiny Rust proxies, ~2mb RSS, <1ms p99 latency
- Control plane: Go
- Supports: HTTP/2, HTTP, gRPC, and TCP
- 0.4.1 (alpha) released last week!

Conduit

- Open source service mesh (Apache v2)
- Built from the ground up for Kubernetes
- Ultralight, ultrafast
- Data plane: tiny Rust proxies, ~2mb RSS, <1ms p99 latency
- Control plane: Go
- Supports: HTTP/2, HTTP, gRPC, and TCP
- 0.4.1 (alpha) released last week!

Goals:
zero config reliability, security, and visibility for Kubernetes apps.

Demo Time

@franziskagoltz

Follow along at home!

```
$ curl https://run.conduit.io/install | sh
```

@franziskagoltz

Live debugging of a Kubernetes App

1. Install an app.

Live debugging of a Kubernetes App

1. Install an app.
2. Oh no, it's failing! (Some of the time.)

Live debugging of a Kubernetes App

1. Install an app.
2. Oh no, it’s failing! (Some of the time.)
3. Try to use Kubernetes dashboard. No luck.

Live debugging of a Kubernetes App

1. Install an app.
2. Oh no, it's failing! (Some of the time.)
3. Try to use Kubernetes dashboard. No luck.
4. Install Conduit control plane

Live debugging of a Kubernetes App

1. Install an app.
2. Oh no, it’s failing! (Some of the time.)
3. Try to use Kubernetes dashboard. No luck.
4. Install Conduit control plane
5. LIVE INJECT Conduit data plane into the app, without downtime.

Live debugging of a Kubernetes App

1. Install an app.
2. Oh no, it’s failing! (Some of the time.)
3. Try to use Kubernetes dashboard. No luck.
4. Install Conduit control plane
5. LIVE INJECT Conduit data plane into the app, without downtime.
6. Use Conduit to trace the source of the error across the app.

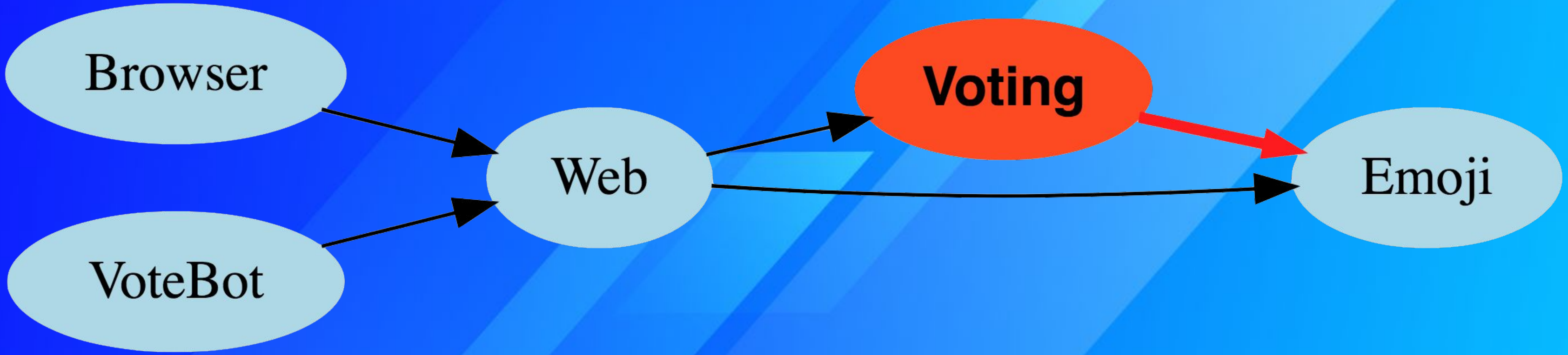
Live debugging of a Kubernetes App

1. Install an app.
2. Oh no, it’s failing! (Some of the time.)
3. Try to use Kubernetes dashboard. No luck.
4. Install Conduit control plane
5. LIVE INJECT Conduit data plane into the app, without downtime.
6. Use Conduit to trace the source of the error across the app.
7. Use Conduit to identify the failing endpoint.

Live debugging of a Kubernetes App

1. Install an app.
2. Oh no, it's failing! (Some of the time.)
3. Try to use Kubernetes dashboard. No luck.
4. Install Conduit control plane
5. LIVE INJECT Conduit data plane into the app, without downtime.
6. Use Conduit to trace the source of the error across the app.
7. Use Conduit to identify the failing endpoint.
8. Profit.

Demo App



@franziskagoltz

Follow along (if you wish)

```
#GitHub repo: http://bit.ly/kubecondemo

$ curl https://run.conduit.io/install | sh
$ conduit install | kubectl apply -f -
$ curl http://bit.ly/emojivoto  |
     conduit inject - | kubectl apply -f -
```

@franziskagoltz

Live Demo Time

@franziskagoltz

Demo Review

— Installed Emojivoto App on K8S

— Saw errors, but saw nothing on the K8s dashboard

— Installed Conduit

— Used stat and tap commands to identify a failing call deep within the application's topology

— Yelled at the correct teams to fix their code!

All without modifying, or even taking down, the app.

@franziskagoltz

In conclusion

— Debugging microservices is fundamentally different from debugging monolithic apps

In conclusion

— Debugging microservices is fundamentally different from debugging monolithic apps

— The service mesh is uniquely positioned to provide debugging information at the right layer of abstraction

In conclusion

— Debugging microservices is fundamentally different from debugging monolithic apps

— The service mesh is uniquely positioned to provide debugging information at the right layer of abstraction

— It’s easy! And you don’t have to modify your application!

In conclusion

— Debugging microservices is fundamentally different from debugging monolithic apps

— The service mesh is uniquely positioned to provide debugging information at the right layer of abstraction

— It’s easy! And you don’t have to modify your application!

— Try it out today: conduit.io

In conclusion

— Debugging microservices is fundamentally different from debugging monolithic apps

— The service mesh is uniquely positioned to provide debugging information at the right layer of abstraction

— It’s easy! And you don’t have to modify your application!

— Try it out today: conduit.io

— 100% open source, Apache v2, works with Kubernetes 1.8+

Any Questions?

Franziska von der Goltz

Software Engineer

@franziskagoltz

franziska@buoyant.io