KubeCon

CloudNativeCon

Europe 2018

How to get a service mesh into production without getting fired

William Morgan

Service mesh connoisseur

You're standing on the precipice...

About me

William Morgan

Used to write code

Now writes email at Buoyant

Putting service meshes into prod for almost 2 years!

About this talk

This is a talk about two things:

1. Manipulating human beings
2. Good software engineering practices

At the end of this talk:

You'll be prepared to take a service mesh to prod!

(Or know why that isn't a good idea)

Are you really going to get fired?

If you’re really worried about your job:
talk to me after class.

Four ways to fail

1. You can’t convince your colleagues that it’s a good idea
2. You convince them, but getting it into prod fails
3. You get it into prod, things go wrong, it’s not your fault, but you get blamed anyway
4. Things go wrong, and it IS your fault

Failure Scenario I

You can't convince your peers that it's a good idea.

Well, **is** it a good idea?

Why are you doing this?

What **problem** are you trying to solve?

Good Problem / Bad Problem

Good problem / bad problem

We ne ervice mesh!

Good problem / bad problem

Our services are written in 6 different languages and we don’t have consistent telemetry libraries across them.

GOOD PROBLEM

Good problem / bad problem

We want to h d native.

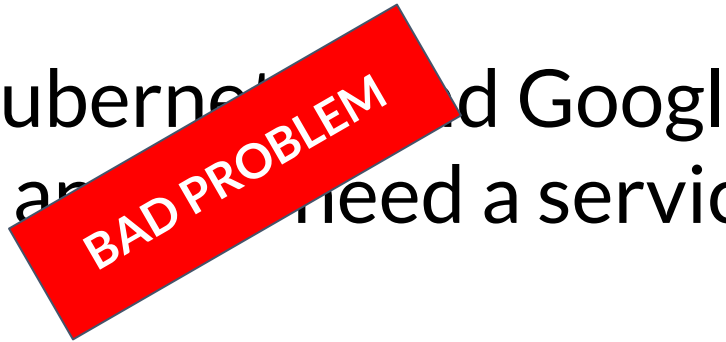
Good problem / bad problem

We have 70 service teams and getting them to add TLS to all of their services would be an impossible organizational task.

GOOD PROBLEM

Fun Game

Google made Kuberne[illegible]d Google makes a service mesh a[illegible]need a service mesh



A service mesh isn't **always** the answer.
Solve the right problem.

William's Guide to Convincing Human Beings

1. Identify who is affected (the *stakeholders*)
2. Determine what the service mesh improves for them (the *incentives*)
3. Understand what they're worried about (the *concerns*)
4. Mitigate concerns, extol incentives, and communicate!

This is called ***getting stakeholder buy-in***.

(Morgan's 4th Law: sufficiently advanced engineering work is indistinguishable from sales.)

Examples

Stakeholder	Incentive	Concern
Platform engineers	• Unified visibility across all services • Failure isolation	• Is it reliable? • Will it introduce complexity?
Developers / service owners	• Remove complex communication logic from your code • Easily run parallel versions of a service	• What do I have to change? • Do I have to learn a new complicated way of doing things?
Security team	• Consistent application of TLS and authz/authn across services • Policy	• Will it make things *less* secure? • What new attack vectors are introduced?
The Management	• Faster pace of development • Fewer outages	• What dependencies are we introducing to our business?

Case study: 4k-person education co.

Problem: Long-running (9 months!) feature freezes. Making progress on product features is difficult.

Incentive: Linkerd can run parallel service versions concurrently, so that dev teams can continue iterating.

Concerns: Reliability. Changing workflows.

Solution: Address concerns via testing & education during Linkerd roll out. Has been in prod for over one year.

Failure Scenario II

You convince your peers, but the rollout to prod fails

Are you trying to boil the ocean?

Have you taken the time to address risks?

Are you clearly communicating the value?

Is it taking too long to demonstrate value?

Case study: Multi-billion$ finance co.

Production deploy: Complex rollout of Linkerd. Multiple configs, multiple envs (including non-K8s), blue/green deploys, NGINX, hardware load balancers, etc.

Production failure: Lots of hard-to-reproduce issues. Hard to understand what's going on. Mitigations, but not great ones.

Almost removed Linkerd from prod (luckily, we fixed thigns before this!)

What went wrong?

Root cause:

- Linkerd bugs (yes, these sometimes happen)
- Hardware LB bugs/misconfiguration

These were actually minor! **Real** problem was the compounding factors:

- Complex environment
- Insufficient communication
- Hard to understand Linkerd's internal state

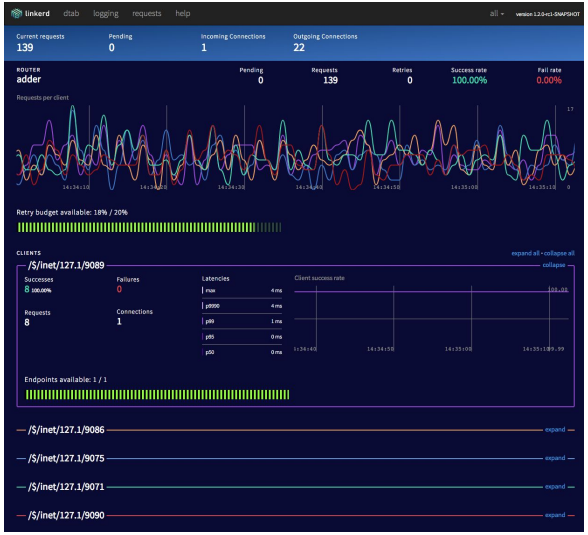
Lessons learned

- **Start small**. One config, one environment. (Product features can help!)
- **Encourage communication**. Sometimes it takes work to make different teams talk to each other.
- **Get good diagnostics.** You *must* be able to clearly reason about what the service mesh is doing when things go wrong. (Product features can help!)

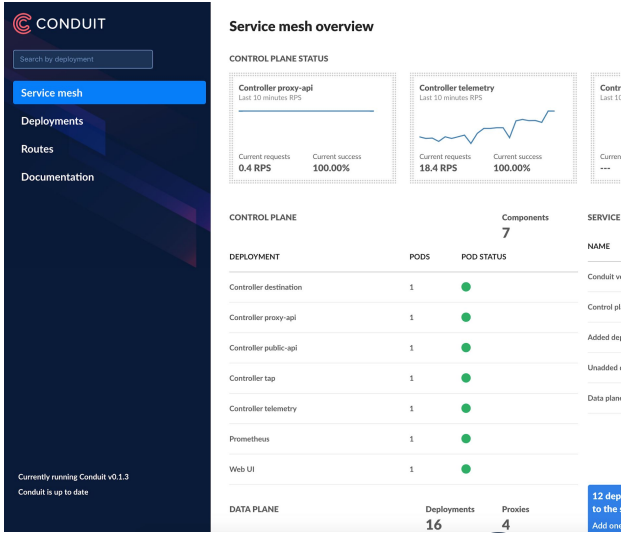
The importance of diagnostics

Prod bugs are often hard-to-reproduce, situational bugs.

Diagnostics are critical for these situations.




```
{
  "/$/inet/localhost/4100": {
    "state": "bound",
    "addresses": [
      "localhost:4100"
    ]
  },
  "/#/io.l5d.k8s/emojivoto/grpc/voting-svc": {
    "state": "bound",
    "addresses": [
      "172.17.0.8:8080",
      "172.17.0.7:8080",
      "172.17.0.5:8080"
    ]
  }
}
```



Examples of good diagnostics

- Headers (*l5d-error*) annotating failed requests
- Failure metrics (*connection-level errors* vs *app-level errors*)
- Access to internal state (*client-state.json*, *k8s-namer-state.json*)
- Human-facing “admin” dashboard
- Error-tracking commands (`conduit wtf`)

Failure Scenario III

“Our thing is breaking! It must be the service mesh!”

The service mesh is new, strange, and sits in between everything.

It will take time before people stop blaming the service mesh.

Case Study Bingo

Things that Linkerd has been blamed for:

- App servers failing (*running out of file descriptors*)
- Huge network latency (*MTU mismatch for network segments*)
- Service failure (*bugs in code, poor failure logging*)
- Connection timeouts (*HW LBs ran out of IPs & reusing ports*)
- Apps being unreachable (*firewall misconfiguration*)

Passing Scenario 3

You can’t avoid misplaced blame. But you can be prepared.

1. Understand what the service mesh does (*and **doesn’t** do*)
2. Use a service mesh that is debuggable & introspectable
3. Education! Documentation! Lunch ‘n learns! Teach teams how to understand and diagnose issues themselves.

Failure Scenario IV

Things go wrong, and it IS the service mesh

Morgan's 3rd law: To take down prod, you must first be in prod.

(Corollary: If you want to be really safe, never put anything into prod.)

This ***will*** happen and you need to plan for it.

Case study: Monzo

Anatomy of a Production Kubernetes Outage - Oliver Beattie, Head of Engineering, Monzo Bank

Linkerd took down a bank! (*sorta*)

- Kubernetes had a bug
- Linkerd had a bug exposed by Monzo's mitigation for K8s's bug
- Public postmortem: great example of learning from failure

Preparing for real failures

1. Remove the culture of blame
2. Learn from failures and outages
3. Quantify trade-offs: the cost per 9 of uptime
4. Understand service mesh failure modes, and have a game plan!

How to get a service mesh into prod

1. Make sure you're solving real problems
2. Find stakeholders and make them happy
3. Roll out incrementally
4. Get results fast and communicate them
5. Accept that things will go wrong and have a plan
6. Profit???

Q&A

THANK YOU

Morgan's 1st Law: Try out Conduit [conduit.io] and give it a GitHub star!
(Corollary: Hands-on demos in our booth & check out the other Linkerd talks)

Linkerd @ Kubecon EU

KubeCon | CloudNativeCon Europe 2018

Wed 11:55	From Unreliable RPC to Resilience with Linkerd	Edward Wilde, **Form3**
Wed 17:10	Anatomy of a Prod Kubernetes Outage	Oliver Beattie, **Monzo**
Thu 11:55	How a Service Mesh Helped Us Build Production Cloud-Native Services	Israel Sotomayor, **Moltin**
Thu 15:50	Hands-on workshop w/Linkerd and Conduit (Linkerd Intro session)	Your friends at Buoyant
Fri 14:45	Lightning talks! (Linkerd Deep Dive)	**BigCommerce, SoundCloud, BrandWatch, AtTest**

@wm