

# What is a Service Mesh?

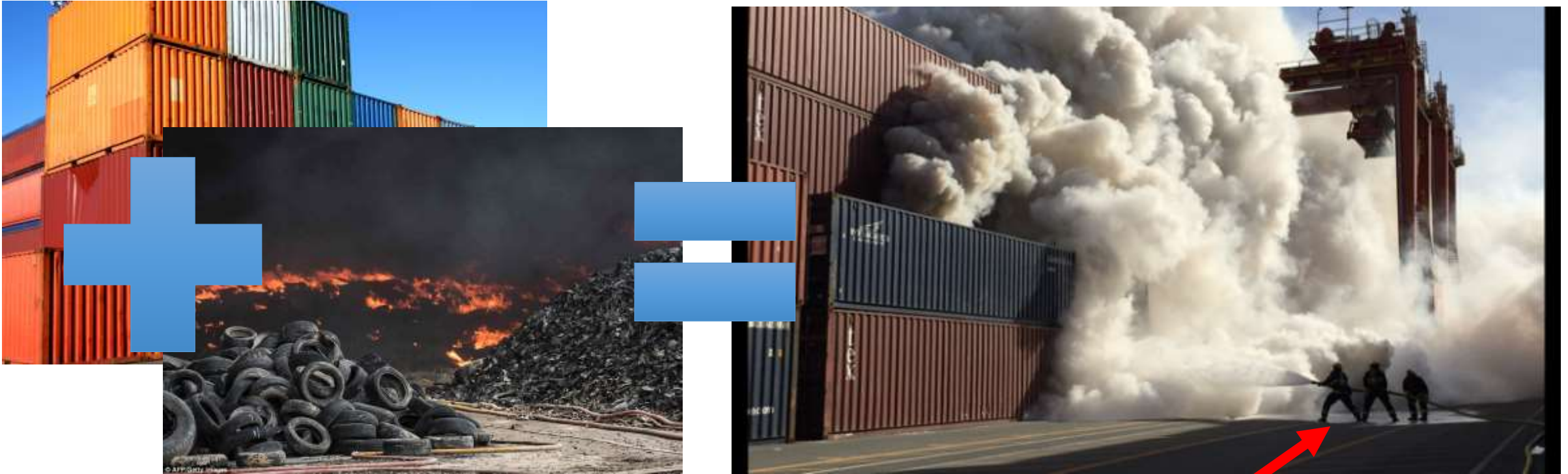
## *And Do I Need One When Developing Cloud Native Systems?*

---

Daniel Bryant

@danielbryantuk

# Cloud Native Apps: Expectations versus reality

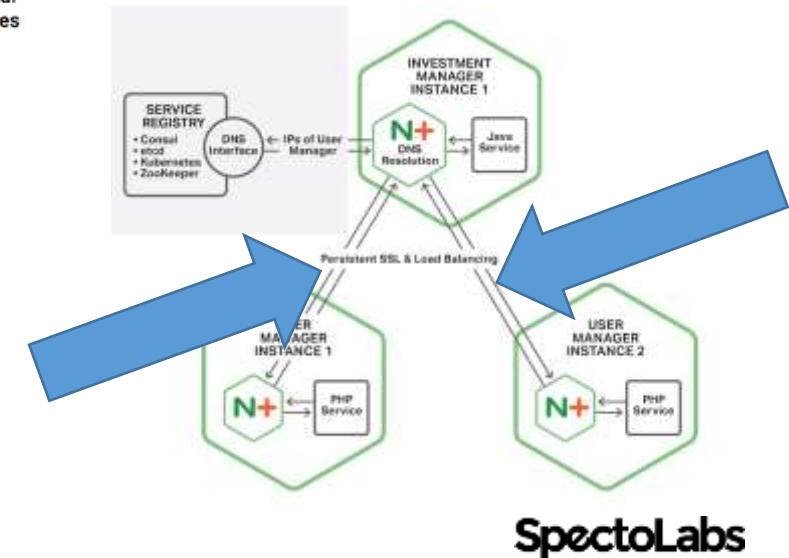
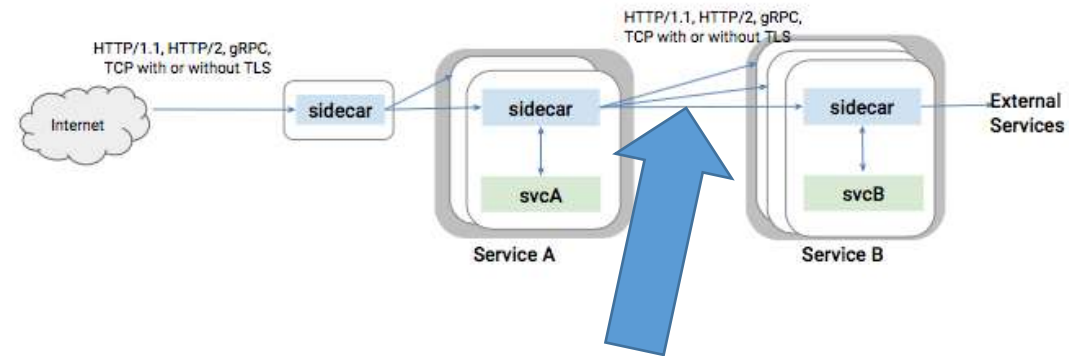
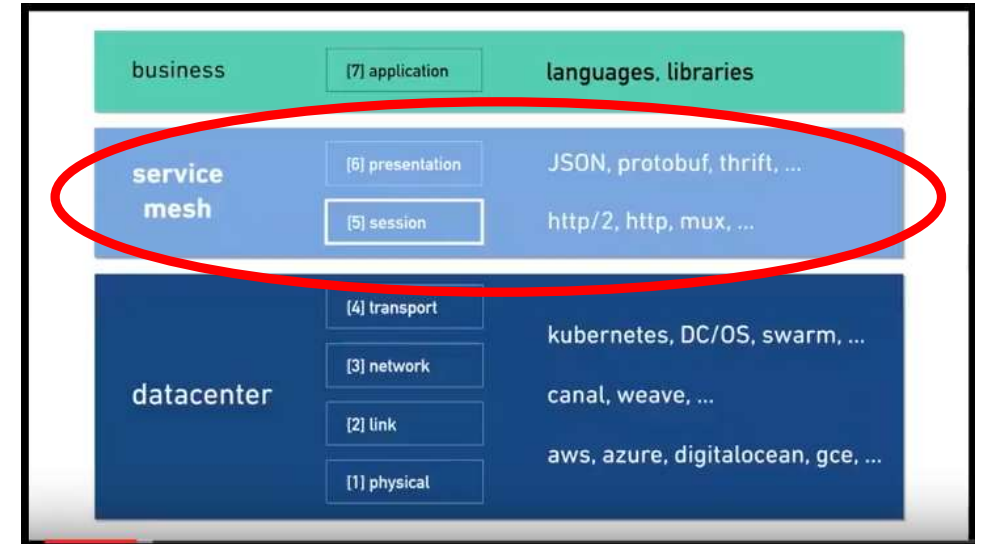
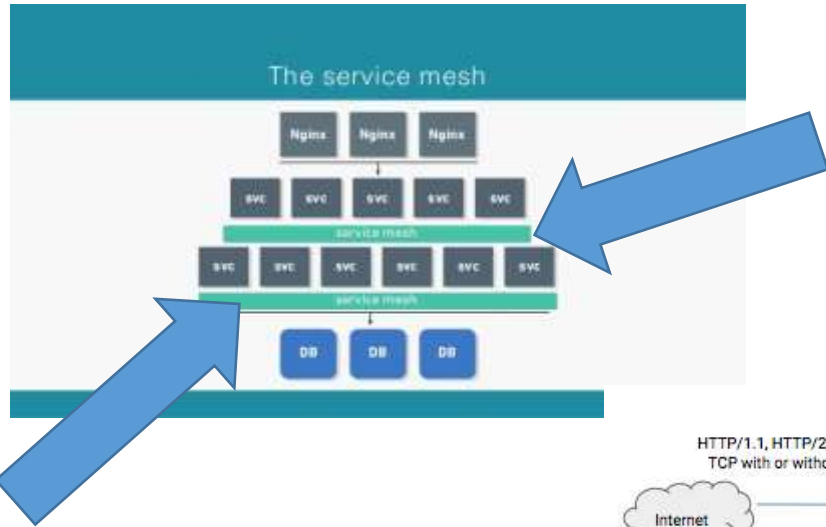


“DevOps”

# tl;dr – Service Meshes

- A service mesh is a dedicated infrastructure layer for making service-to-service communication safe, fast, reliable, and (operator) configurable
- Consists of control plane (“brains”, API, UI) and data plane (service proxies)
  - Some confusion on where the “service mesh” begins and ends
- Essential as we move from deployment of *complicated* monoliths/services to orchestration of *complex* cloud native microservices and functions

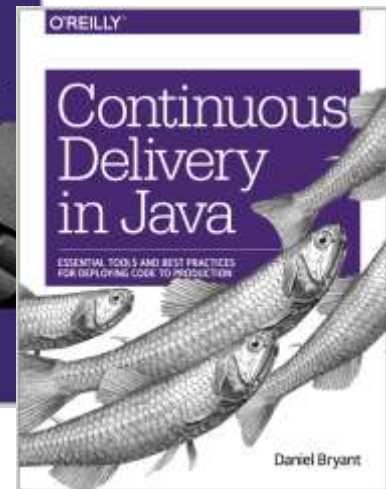
# tl;dr – Service Meshes



# @danielbryantuk



- Independent Technical Consultant, CTO at SpectoLabs
  - Architecture, DevOps, Java, microservices, cloud, containers
  - Continuous Delivery (CI/CD) advocate
  - Leading change through technology and teams



[bit.ly/2jWDSF7](https://bit.ly/2jWDSF7)

# Setting the Scene

**Complex**  
**(Probe, Sense, Respond)**

**2010s**

Microservices, functions, SaaS-all-the-things  
Polyglot languages  
Cloud and containers (Datacenter as a Computer)  
Software-Defined Everything  
Optimise for innovation (and Antifragility)  
Business teams ("FinDev", SRE and Platform Team)

**Complicated**  
**(Sense, Analyse, Respond)**

**2000s**

Monoliths, Coarse-grained SOA, SaaS  
Frontend/backend language  
"Co-lo" or private datacenters  
Configuration management  
Optimise for Recovery (MTTR)  
Generalist teams (Full Stack and "DevOps")

**Simple**  
**(Sense, Categorise, Respond)**

**1990s**

Monoliths  
Single language  
In-house hardware (servers, SAN, networks)  
Manual config and scripting  
Optimise for Stability (MTBF)  
Specialist staff/departments



# What do “cloud native” comms look like?

- Services communicate over a network
- These interactions are non-trivial
- Lot of value in understanding the network
- The application is ultimately responsible



Christian Posta

Chief Architect, cloud application development @ Red Hat, author *Microservices for Java Developers*, open-source enthusiast, coeditor @ *Apache, Cloud, Integration, Ephemeral, Docker, OpenShift, Fabric8*, #blogger

Twitter  
Google+  
LinkedIn  
GitHub  
Stackoverflow

## Application Network Functions With ESBs, API Management, and Now.. Service Mesh?

I've talked quite a bit recently about the evolution of microservices patterns and how *service proxies like Envoy from Lyft* can help push the responsibility of resilience, service discovery, routing, metrics collection, etc down a layer below the application. Otherwise we risk hoping and praying that the various applications will correctly implement these critical functionalities or depend on language-specific libraries to make this happen. Interestingly, this service mesh idea is related to other concepts that our customers in the enterprise space know about, and I've gotten a lot of questions about this relationship. Specifically, how does a service mesh relate to things like ESBs, Message Brokers, and API Management? There definitely is overlap in these concepts, so let's dig in. Feel free to follow along [@christianposta on Twitter](#) for more on this topic!

### Four assumptions

#### 1) Services communicate over a network

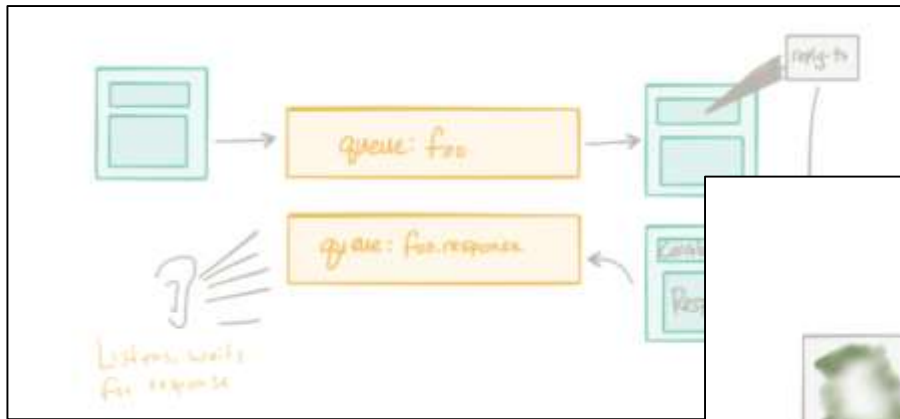
First point to make: We're talking about services communicating and interacting with each other over asynchronous, packet-switched networks. This means they are running in their own processes and in their own "time boundaries" (thus the notion of asynchronicity here) and communicate by sending packets across a network. Unfortunately, there are no guarantees about asynchronous network interaction: we can end up with failed interactions, stalled/latent interactions, etc, and these scenarios are indistinguishable from each other.



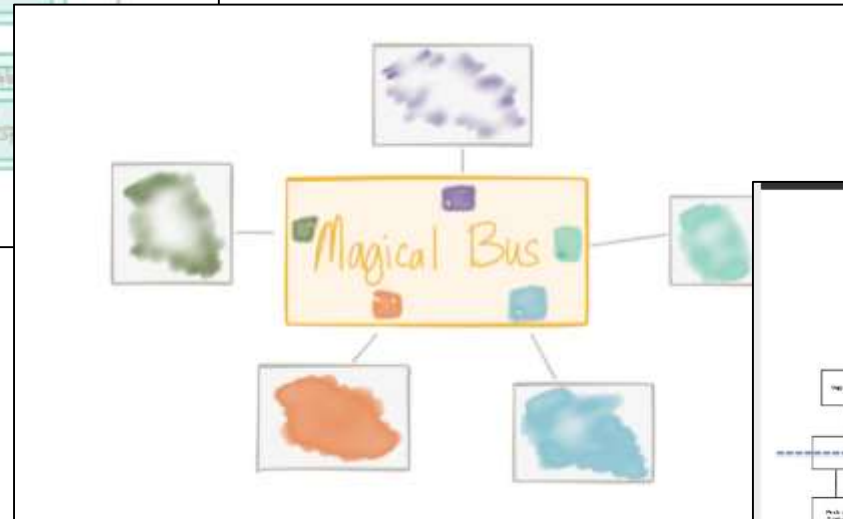
[blog.christianposta.com/microservices/application-network-functions-with-esbs-api-management-and-now-service-mesh/](http://blog.christianposta.com/microservices/application-network-functions-with-esbs-api-management-and-now-service-mesh/)



# But we've been here before...



[blog.christianposta.com/microservices/application-network-functions-with-esbs-api-management-and-now-service-mesh/](http://blog.christianposta.com/microservices/application-network-functions-with-esbs-api-management-and-now-service-mesh/)



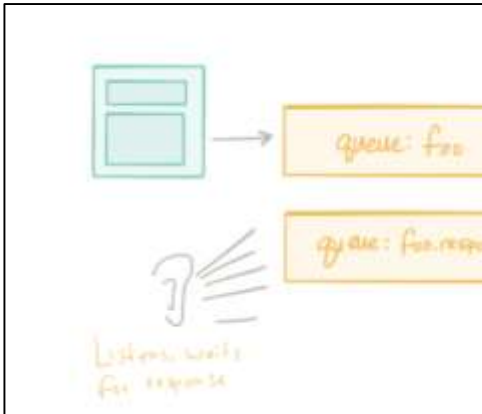
## THE ESB IS DEAD - LONG LIVE THE API GATEWAY!

- WATCH FOR THE API GATEWAY MORPHING INTO AN ENTERPRISE SERVICE BUS
  - **LOOSE COUPLING** IS VITAL
- BUT LET ME BE CLEAR..
  - THE API GATEWAY PATTERN IS **AWESOME**
  - CENTRALISE **CROSS-CUTTING** CONCERNS
  - PREVENT WHEEL-REINVENTION (PLUGINS)
  - CHECK OUT **KONG**, **APIGEE**, **MULESOFT** ETC

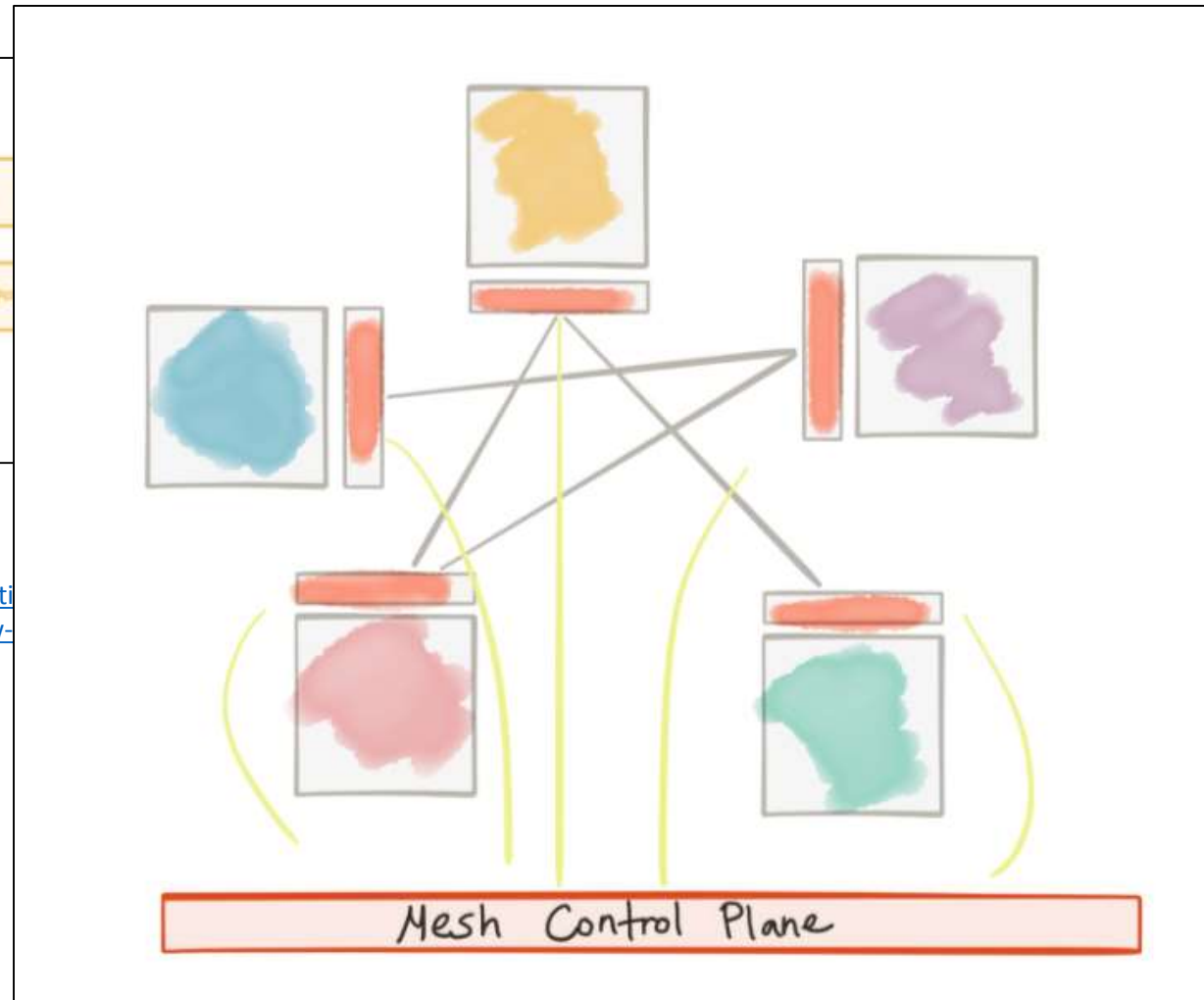
27/04/2017 @danielbryantuk SpectoLabs

[www.slideshare.net/dbryant\\_uk/goto-chicagocraftconf-2017-the-seven-more-deadly-sins-of-microservices](http://www.slideshare.net/dbryant_uk/goto-chicagocraftconf-2017-the-seven-more-deadly-sins-of-microservices)

# But we've been here before...



[blog.christianposta.com/microservices/application-functions-with-esbs-api-management-and-now-](http://blog.christianposta.com/microservices/application-functions-with-esbs-api-management-and-now-)



## AD - LONG LIVE THE API GATEWAY!

- WATCH FOR THE API GATEWAY MORPHING INTO AN ENTERPRISE SERVICE BUS
  - LOOSE COUPLING IS VITAL
- BUT LET ME BE CLEAR...
  - THE API GATEWAY PATTERN IS **AWESOME**
  - CENTRALISE **CROSS-CUTTING** CONCERNS
  - PREVENT WHEEL-REINVENTION (PLUGINS)
  - CHECK OUT **KONG**, **APIGEE**, **MULESOFT** ETC

@danielbryantuk SpectoLabs

[are.net/dbryant](http://are.net/dbryant) [uk/goto-chicagocraftconf-e-deadly-sins-of-microservices](http://uk/goto-chicagocraftconf-e-deadly-sins-of-microservices)

# Let's go unicorn spotting...



- Netflix
  - Karyon + HTTP/JSON or RxNetty RPC + Eureka + Hystrix + ...
- Twitter
  - Finagle + Thrift + ZooKeeper + Zipkin
- Google
  - Stubby (gRPC) + GSLB + GFE + Dapper
- AirBnB
  - HTTP/JSON + SmartStack + ZooKeeper + Charon/Dyno

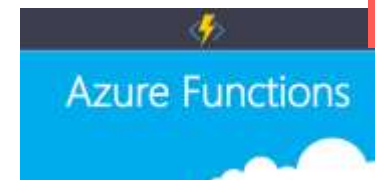
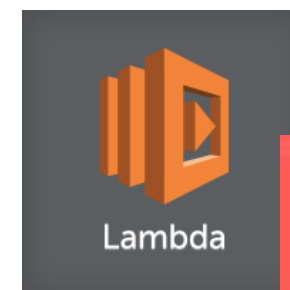
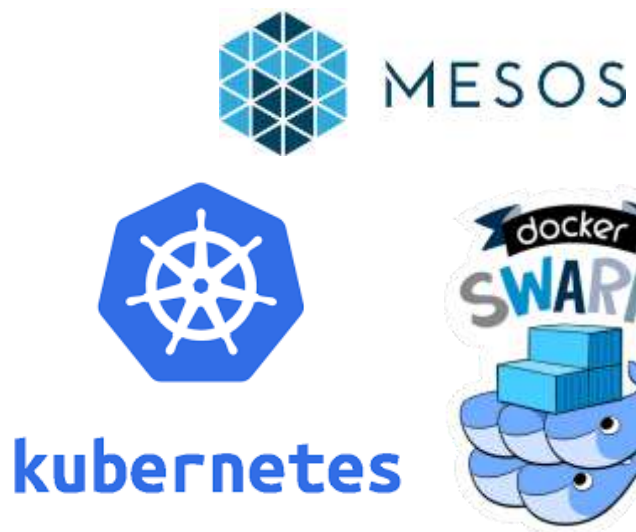
# So, from a technology perspective...

- Deploying cloud native services/functions to a “platform” is essential
  - Abstracts underlying resources and provides runtime foundations
- Need clear collaboration zones for dev/ops/platform
  - Must also cultivate “mechanical sympathy”
- Managing lots of out-of-process communication going “over the wire”
  - We must not treat local and remote calls the same (dev, observability etc)

# So, from a technology perspective...

- **Deploying cloud native services/functions to a “platform” is essential**
- Need clear collaboration zones for dev/ops/platform
- Managing lots of out-of-process communication going “over the wire”

# Service/function platforms



# So, from a technology perspective...

- Deploying services/functions to a “platform” is essential
- **Need clear collaboration zones for dev/ops/platform**
- Managing lots of out-of-process communication going “over the wire”



# Collaboration zones for deployment

**Built-in or BYO**

**cf push**

The application is tested against a set of curated buildpacks

**cf push --buildpack <url>**

The buildpack is referenced by a Git URL

**Example Manifest**

Although you can deploy apps without a manifest, manifests provide consistency and reproducibility. This can be useful when you want your apps to be portable between different clouds.

Manifests are written in YAML. The manifest below illustrates some YAML conventions, as follows:

- The manifest begins with three dashes.
- The `application:` block begins with a heading followed by a colon.
- The application `name` is preceded by a single dash and one space.
- Subsequent lines in the block are indented two spaces to align with `name`.

```
---
application:
- name: my-app
  memory: 512M
  instances: 2
```

A minimal manifest requires only an application `name`. To create a valid minimal manifest, remove the `memory` and `instances` properties from this example.

deployment.yaml

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2 # tells deployment to run 2 pods matching the template
  template: # create pods using pod definition in this template
    metadata:
      # unlike pod-nginx.yaml, the name is not included in the meta data as a
      # generated from the deployment name
    labels:
      app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Let's start by writing a simple SAM template.yaml:

```
YAML
AWSTemplateFormatVersion : '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  VotesTable:
    Type: 'AWS::Serverless::SimpleTable'
  VoteSpesTable:
    Type: 'AWS::Serverless::Function'
  Properties:
    Runtime: python3.6
    Handler: lambda_function.lambda_handler
    Policies: AmazonDynamoDBFullAccess
    Environment:
      Variables:
```



MEET SAM



USE SAM TO BUILD TEMPLATES THAT DEFINE YOUR SERVERLESS APPLICATIONS.



DEPLOY YOUR SAM TEMPLATE WITH AWS CLOUDFORMATION.

# So, from a technology perspective...

- Deploying services/functions to a “platform” is essential
- Need clear collaboration zones for dev/ops/platform
- **Managing lots of out-of-process communication going “over the wire”**

# The Eight Fallacies of Distributed Computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.



OR

**NETFLIX**  
**OSS**



(~ 2013)

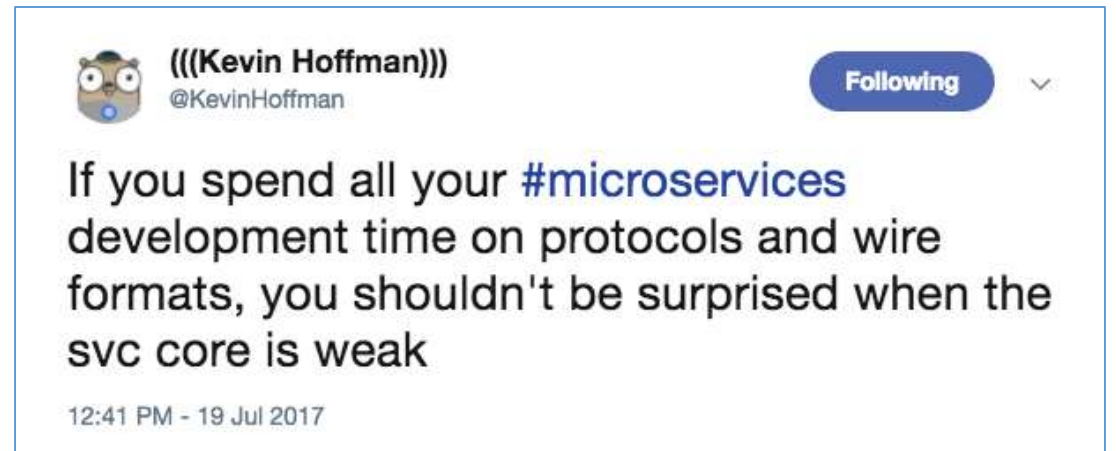
<https://www.somethingsimilar.com/2013/01/14/notes-on-distributed-systems-for-young-bloods/>

# So, from a technology perspective...

- Deploying services/functions to a “platform” is essential
- Need clear collaboration zones for dev/ops/platform
- Managing lots of out-of-process communication going “over the wire”

# But be careful, technology is seductive...

- Service meshes are an emerging and rapidly evolving space
- Only one part of cloud native solution
- For big picture and people aspects:
  - [“Microservices: Org and People Impact”](#)
  - [“Seven Deadly Sins of Microservices”](#)



<https://twitter.com/KevinHoffman/status/887638576409837569>

# Service Mesh Functionality

# Service mesh features

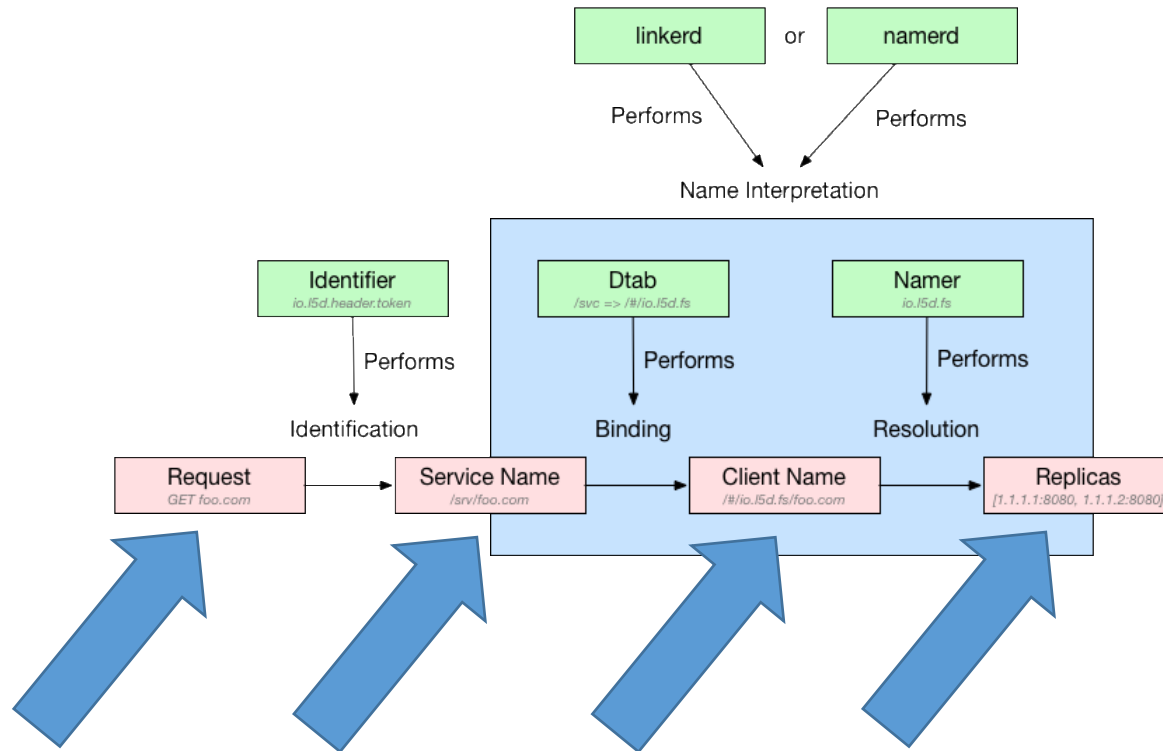
- Normalises naming and adds logical routing
  - user-service -> AWS-us-east-1a/prod/users/v4
- Adds traffic shaping and traffic shifting
  - Load balancing
  - Deploy control
  - Per-request routing (shadowing, fault injection, debug)
- Adds baseline reliability
  - Health checks, timeouts/deadlines, circuit breaking, and retry (budgets)



# Service mesh features

- Increased security
  - Transparent mutual TLS
  - Policies (service Access Control Lists - ACL)
- Observability / monitoring
  - Top-line metrics like request volume, success rates and latencies
  - Distributed tracing
- Sane defaults (to protect the system)
  - With options to tune

# Naming and load balancing



MARCH 16, 2016 STEVE HENDON

This post was co-written with [Ruben Omta](#).

Load balancing is a critical component of any large-scale software deployment. But there are many ways to do load balancing. Which way is best? And how can we evaluate the different options?

In the modern software ecosystem, load balancing plays several roles. First, it is fundamental to the notion of scalability. Software is deployed in multiple, identical replicas. We “scale” software by deploying additional replicas. Traffic must be distributed across replicas, and this distribution is the act of load balancing.

Load balancing provides another necessary feature: that of resilience. Intuitively speaking, a resilient system is one in which the failure of individual components does not cause the system itself to fail. Software, or the hardware on which it runs, will fail. By distributing traffic only to instances which are capable of serving it, load balancing allows us to join multiple fallible components into a single resilient system.

We can extend this model of resilience one step further, to address another unwelcome visitor in distributed systems: latency. Just as the components of a system may fail, so too may they become slow. A good load balancer must protect against latency, just as it protects against failure. Even in the presence of slow replicas, the system as a whole must remain fast.

This third criterion is more subtle than the first two. Algorithmically speaking, addressing scalability and resilience is straightforward: given a set of replicas, distribute traffic across all live replicas, and don’t distribute traffic to replicas that have failed. (We will ignore, for the moment, the not insignificant challenge of assessing the health of a replica.) For latency, the story is less clear: given a set of replicas performing at a variety of speeds, what is the best strategy for distributing load among them?

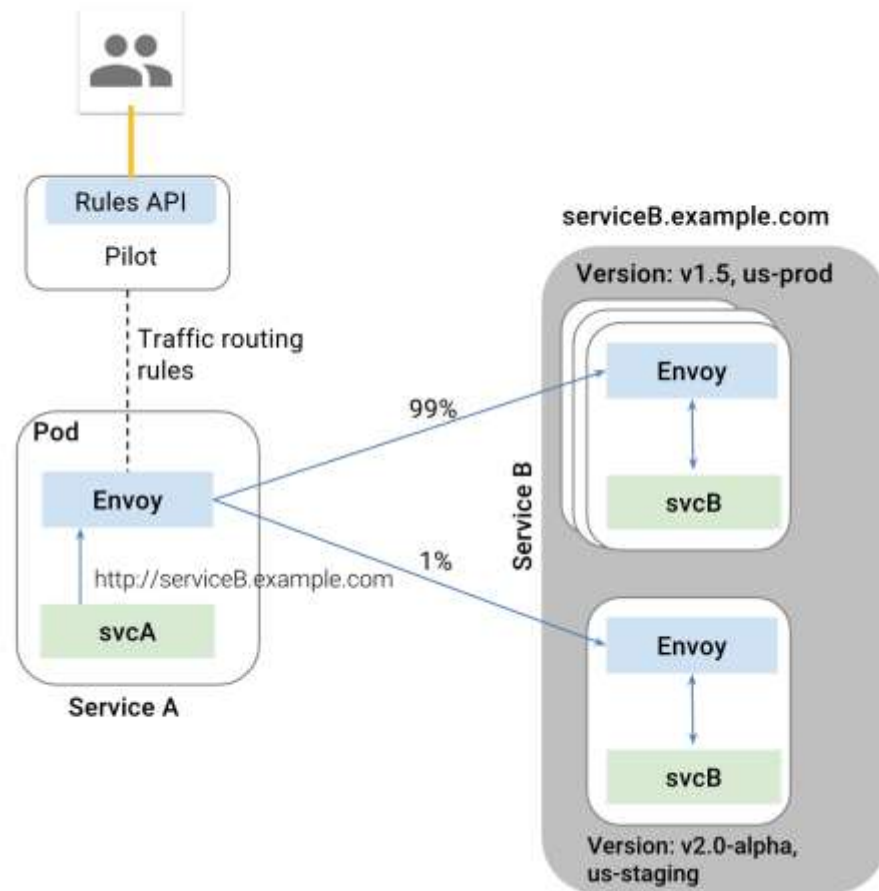
In this article, we run a simple experiment with three algorithmic round robin, least loaded, and peak exponentially-weighted moving average (“peak EWMA”). The three algorithms serve as a test bed for demonstrating the effect that the right—or wrong—choice of load balancing algorithm can have. In particular, we test the effectiveness of these algorithms at handling component latency.

Loosely speaking, the three algorithms behave as follows:

- **Round robin:** distribute requests to each replica in turn.
- **Least loaded:** maintain a count of outstanding requests to each replica, and distribute traffic to replicas with the smallest number of outstanding requests.
- **Peak EWMA:** maintain a moving average of each replica’s round-trip time, weighted by the number of outstanding requests, and distribute traffic to replicas where that cost function is smallest.

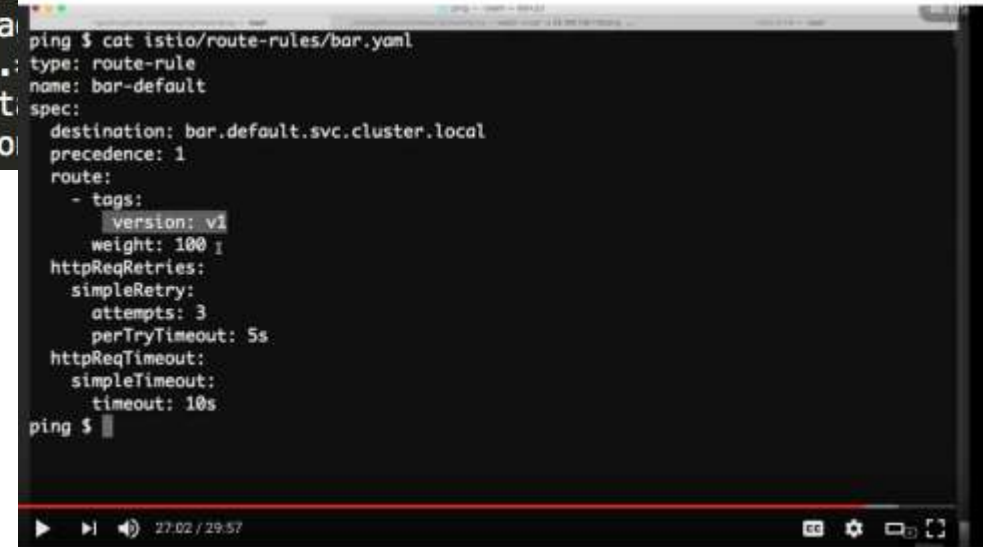
<https://buoyant.io/2016/03/16/beyond-round-robin-load-balancing-for-latency/>

# Traffic control



```
1 destination: serviceB.example.cluster.local match:  
2 source: serviceA.example.cluster.local route:  
3 - tags:  
4   version: v1.0  
5   env: uk-prod  
6   weight: 90  
7 - tags:  
8   version: v2.0-RC  
9   env: uk-staging  
10 weight: 10
```

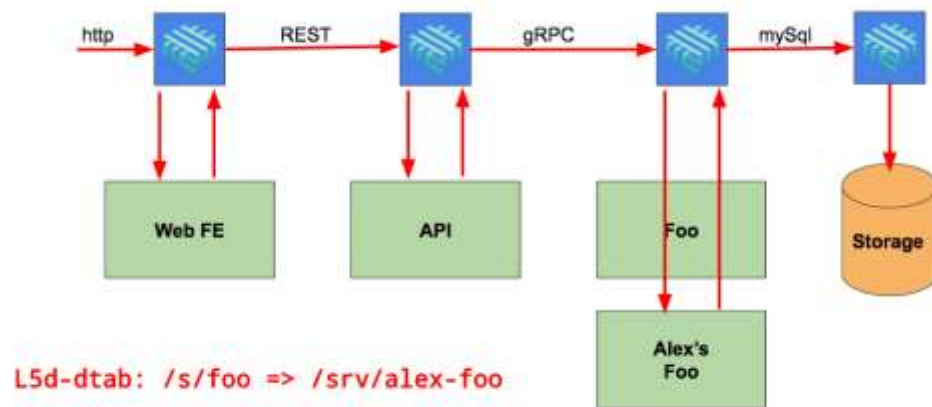
```
1 destination: serviceB.example.cluster.local match:  
2 httpHeaders:  
3   user-agent:   
4   regex: ^(.  
5   route: - t  
6   version
```



<https://istio.io/docs/concepts/traffic-management/request-routing.html>

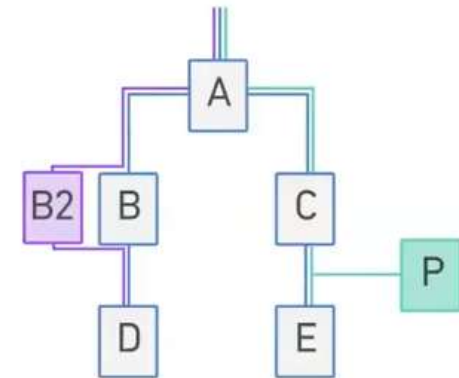
[https://www.youtube.com/watch?v=s4gasWn\\_mFc](https://www.youtube.com/watch?v=s4gasWn_mFc)

# Per-request routing: shadow, fault inject, debug



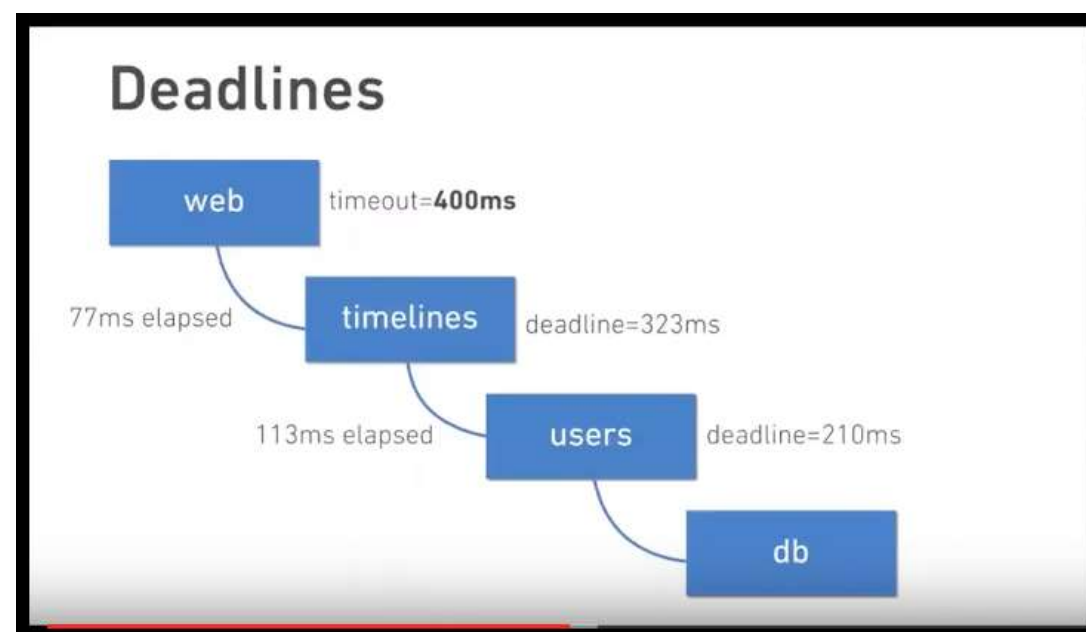
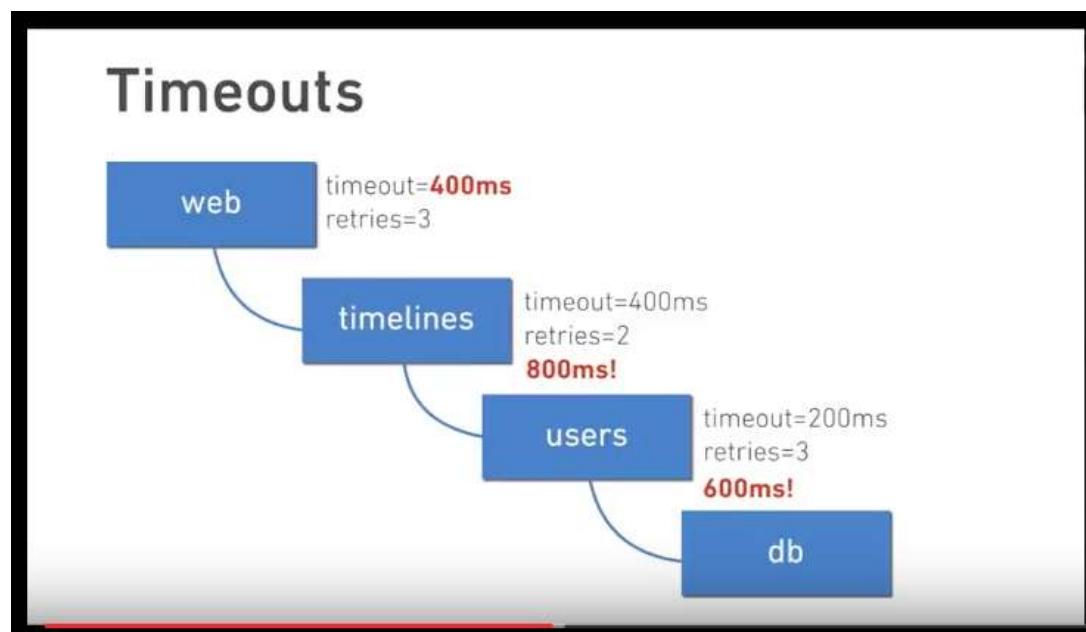
## Per-request routing: staging

GET / HTTP/1.1  
Host: mysite.com  
l5d-dtab: /svc/B => /svc/B2



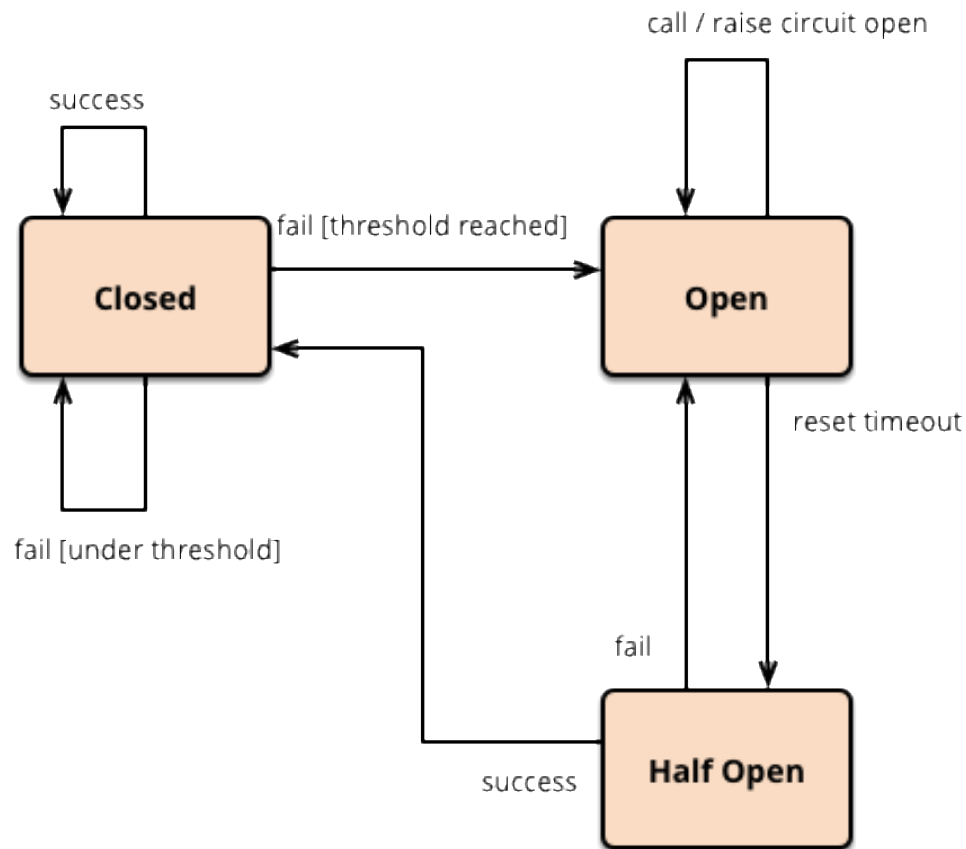
<https://buoyant.io/2017/01/06/a-service-mesh-for-kubernetes-part-vi-staging-microservices-without-the-tears/>

# Timeouts / deadlines



[William Morgan Introduction to Linkerd: https://www.youtube.com/watch?v=0xYSy6OmjUM](https://www.youtube.com/watch?v=0xYSy6OmjUM)

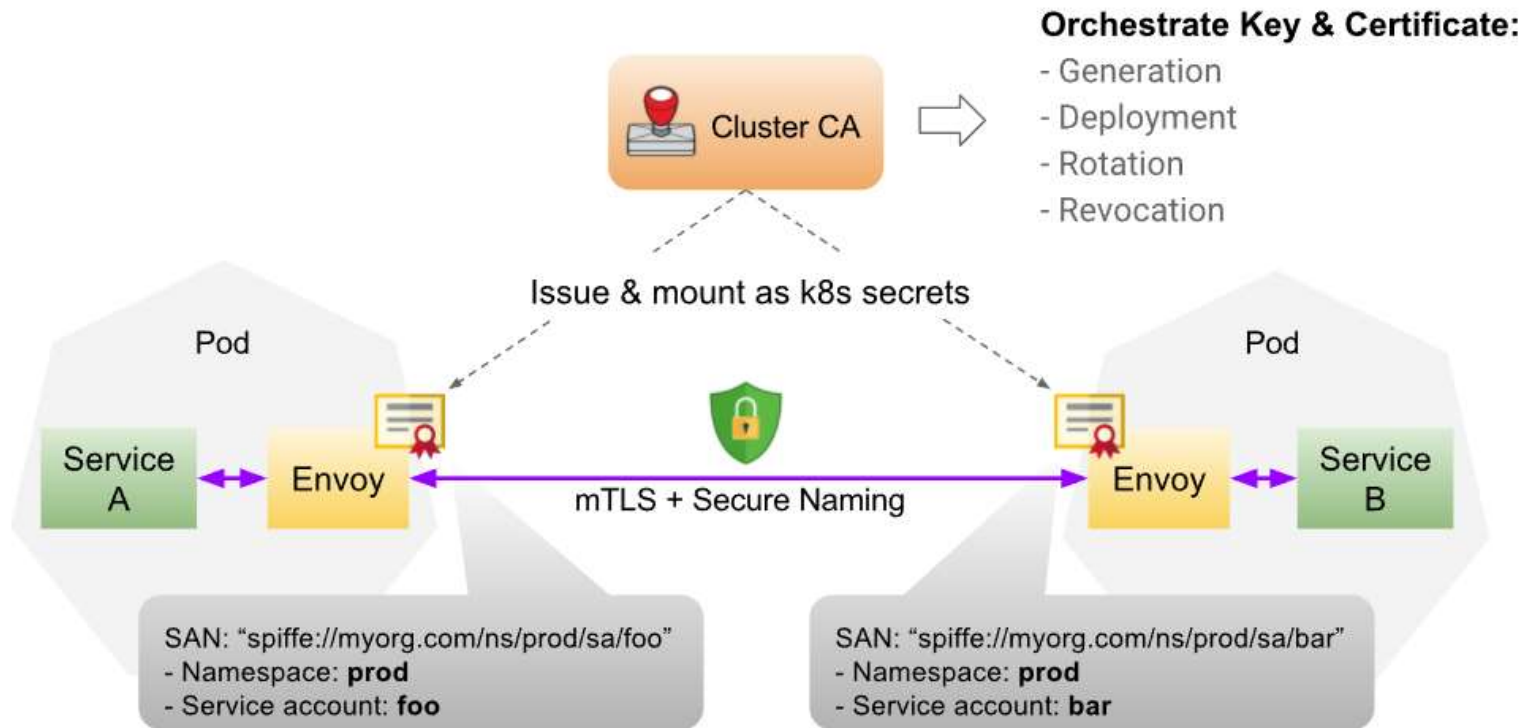
# Circuit breaking (out of process, not Hystrix)



```
1 destination: serviceB.example.cluster.local policy:
2 - tags:
3   version: v1
4   circuitBreaker:
5     simpleCb:
6       maxConnections: 100
7       httpMaxRequests: 1000
8       httpMaxRequestsPerConnection: 10
9       httpConsecutiveErrors: 7
10      sleepWindow: 15m
11      httpDetectionInterval: 5m
```

```
1 - protocol: http
2   client:
3     failureAccrual:
4       kind: io.l5d.successRate
5       successRate: 0.9
6       requests: 20
7     backoff:
8       kind: constant
9       ms: 10000
```

# Mutual TLS (transparent protocol upgrade)



<https://istio.io/blog/istio-auth-for-microservices.html>



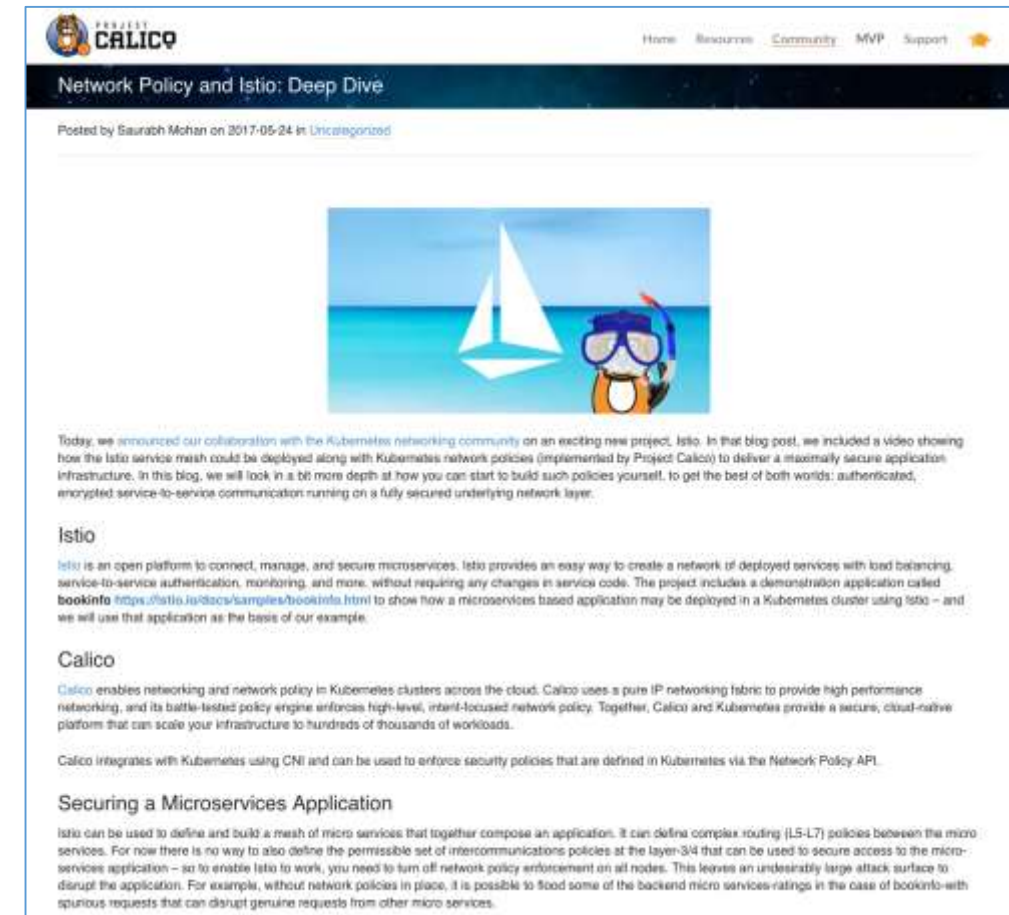
# Communication policies

```
aspects:
- kind: lists          # the aspect's kind
  adapter: myListChecker # the adapter to use to implement this aspect
  params:
    blacklist: true
    checkExpression: source.ip
```

The `kind` field distinguishes the behavior of the aspect being defined. The supported kinds of aspects are shown in the following table.

Kind	Description
quotas	Enforce quotas and rate limits.
metrics	Produce metrics.
lists	Enforce simple whitelist- or blacklist-based access control.
access-logs	Produces fixed-format access logs for every request.
application-logs	Produces flexible application logs for every request.
attributes	Produces supplementary attributes for every request.
denials	Systematically produces a predictable error code.

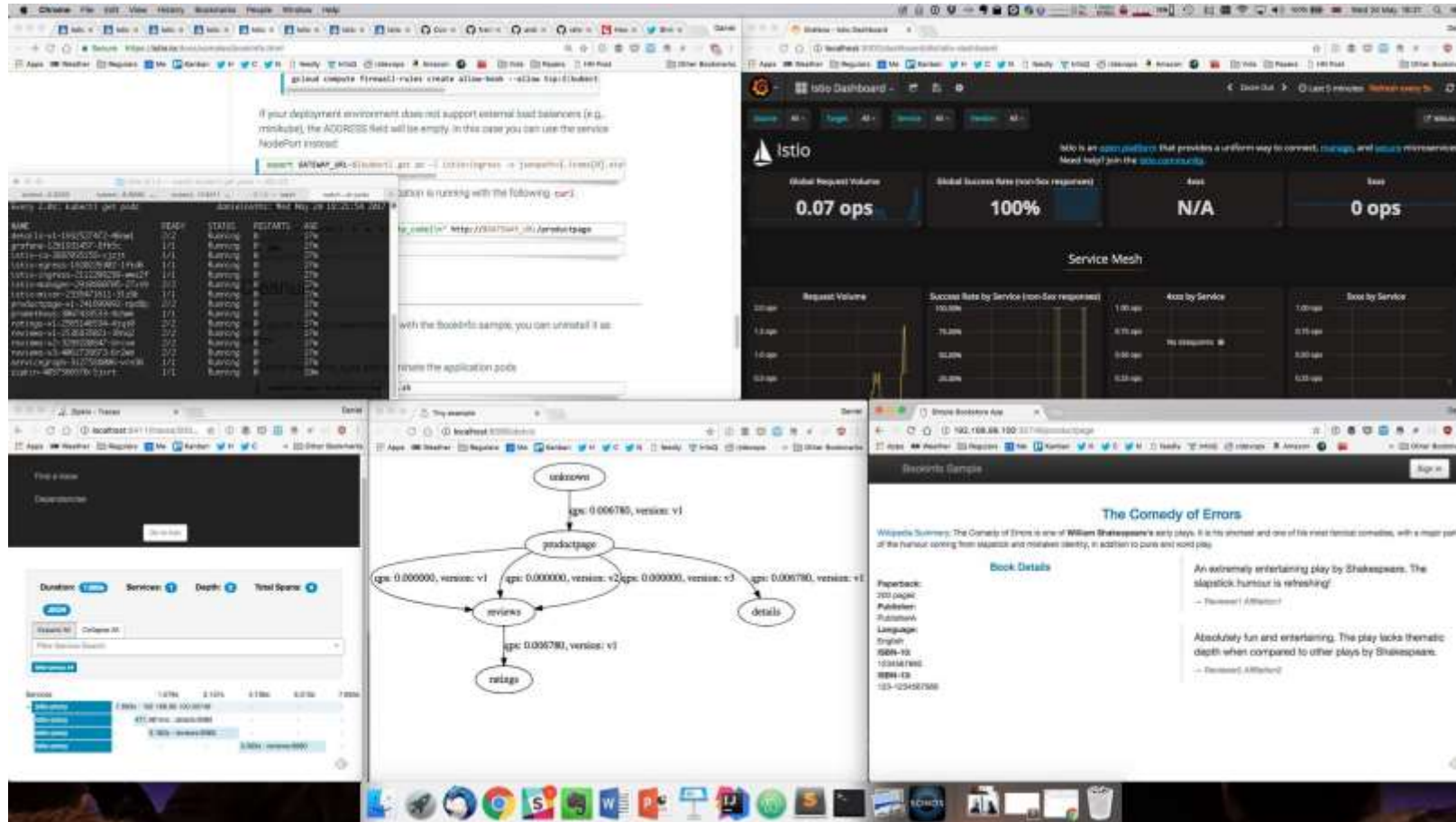
<https://istio.io/docs/concepts/policy-and-control/mixer-config.html#aspects>



The screenshot shows the Project Calico website with a blog post titled "Network Policy and Istio: Deep Dive" by Saurabh Mohan. The post discusses the collaboration between Calico and Istio to provide a secure, cloud-native platform for microservices. It includes an illustration of a sailboat and a diver. The text describes how Istio can be used to define and build a mesh of microservices, and how Calico can be used to enforce security policies. The post also mentions that Calico integrates with Kubernetes using CNI and can be used to enforce security policies defined in Kubernetes via the Network Policy API.

<https://www.projectcalico.org/network-policy-and-istio-deep-dive/>

# Visibility



# Service Mesh Implementations



NGINX





## Overview

Nelson is a fully automated deployment orchestration tool intended to work with Hashicorp Nomad or Apache Mesos. Nelson is responsible for application fully automated security policy management.

## Philosophy

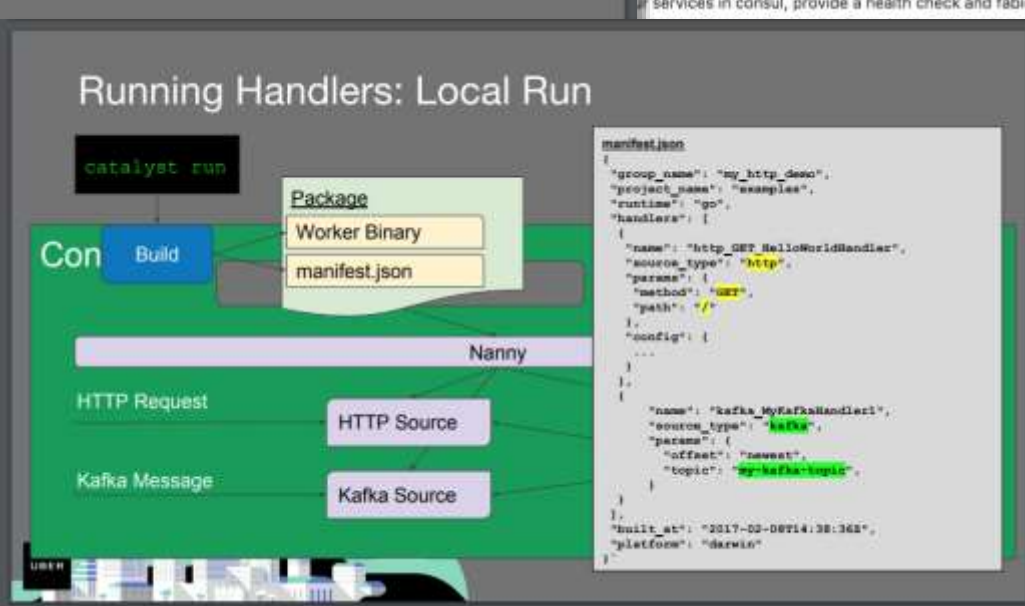
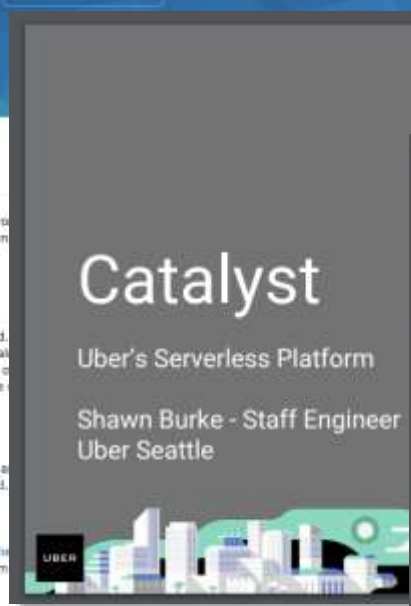
- Uniformity is highly desirable / advantageous.
- As many aspects to deployments as possible should be automated.
- Automated lifecycle management is key to providing a holistic, scalable, and secure deployment process.
- Manual promotion to production is high-cost since the "last-mile" of deployment is often the most difficult.
- Nelson supports manual deployment for those not yet comfortable with automation.

## Features

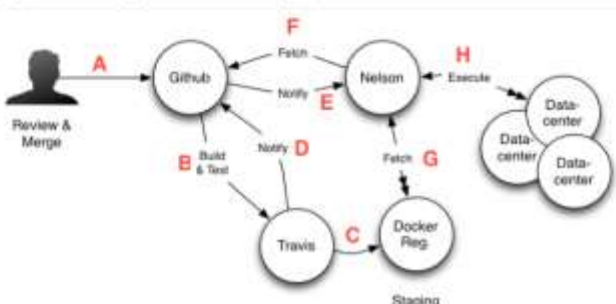
- Fully integrated with GitHub or GitHub Enterprise.
- Developer-driven, automated build & release workflow revisioned and automated.
- Support for multiple cluster managers, including Hashicorp Nomad.
- Compatible with any Docker container.
- Can deploy applications to any number of datacenters.
- State of the art runtime routing via Envoy.
- Integrated support for alert definition and propagation via Prometheus.
- Utilizes secure introduction for safe distribution of credentials from staging.

## Workflow

Nelson splits the build->deploy->release phases into two major parts, each with distinctly different properties. First, the building of the containers and publishing to a staging registry. Second, the deployment, validation, and migration of traffic. The following diagram illustrates a high-level view of the first major part of the workflow.



```
manifest.json
{
  "group_name": "my_http_demo",
  "project_name": "examples",
  "runtime": "go",
  "handlers": [
    {
      "name": "http_GET_HelloWorldHandler",
      "source_type": "http",
      "params": {
        "method": "GET",
        "path": "/"
      },
      "config": {
        ...
      }
    },
    {
      "name": "kafka_MyKafkaHandler",
      "source_type": "kafka",
      "params": {
        "offset": "newest",
        "topic": "my-kafka-topic"
      },
      "built_at": "2017-02-08T14:38:36Z",
      "platform": "darwin"
    }
  ]
}
```



<https://s3-us-west-2.amazonaws.com/emit-website/2017-slides/Shawn-Catalyst-Emit+Conference.pdf>

1. Install from source, binary, Docker or Homebrew.

```
# go 1.8 or higher is required
go get github.com/fabiolb/fabio                                (>= go1.8)

brew install fabio                                              (OSX/macOS stable)
brew install --devel fabio                                     (OSX/macOS devel)

docker pull fabiolb/fabio                                       (Docker)

https://github.com/fabiolb/fabio/releases                       (pre-built binaries)
```

<https://verizon.github.io/nelson/>


<https://github.com/fabiolb/fabio>



# Putting it all together: Istio

- “Istio” is an open platform
  - Connect, manage, secure services
- Proxies are the data plane / mesh
- Proxies are (in theory) swappable
  - But in reality there are different feature sets, security, performance

### Our sidecar of choice - Envoy



- A C++ based L4/L7 proxy
- Low memory footprint
- Battle-tested @ Lyft
  - 100+ services
  - 10,000+ VMs
  - 2M req/s


Goodies:

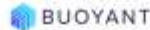
- ◆ HTTP/2 & gRPC
- ◆ Zone-aware load balancing w/ failover
- ◆ Health checks, circuit breakers, timeouts, retry budgets
- ◆ No hot reloads - API driven config updates

Istio's contributions:

- ◆ Transparent proxying w/ SO\_ORIGINAL\_DST
- ◆ Traffic routing and splitting
- ◆ Request tracing using Zipkin

Plus an awesome team willing to work with the community!





BUOYANT

About Ring Resource Careers Contact Us

## LINKERD AND ISTIO: LIKE PEANUT BUTTER AND JELLY

© 2017 BUOYANT MANN BROWN

Today (in addition to some other exciting news) we're happy to announce the release of Linkerd 1.1, which features integration with the Istio project! This integration is currently in a beta state, but is usable today—see below for how to try it out. In the upcoming months we'll continue to invest in making this integration ready for production. (Side note: interested in working on Linkerd and Istio? We're hiring!)

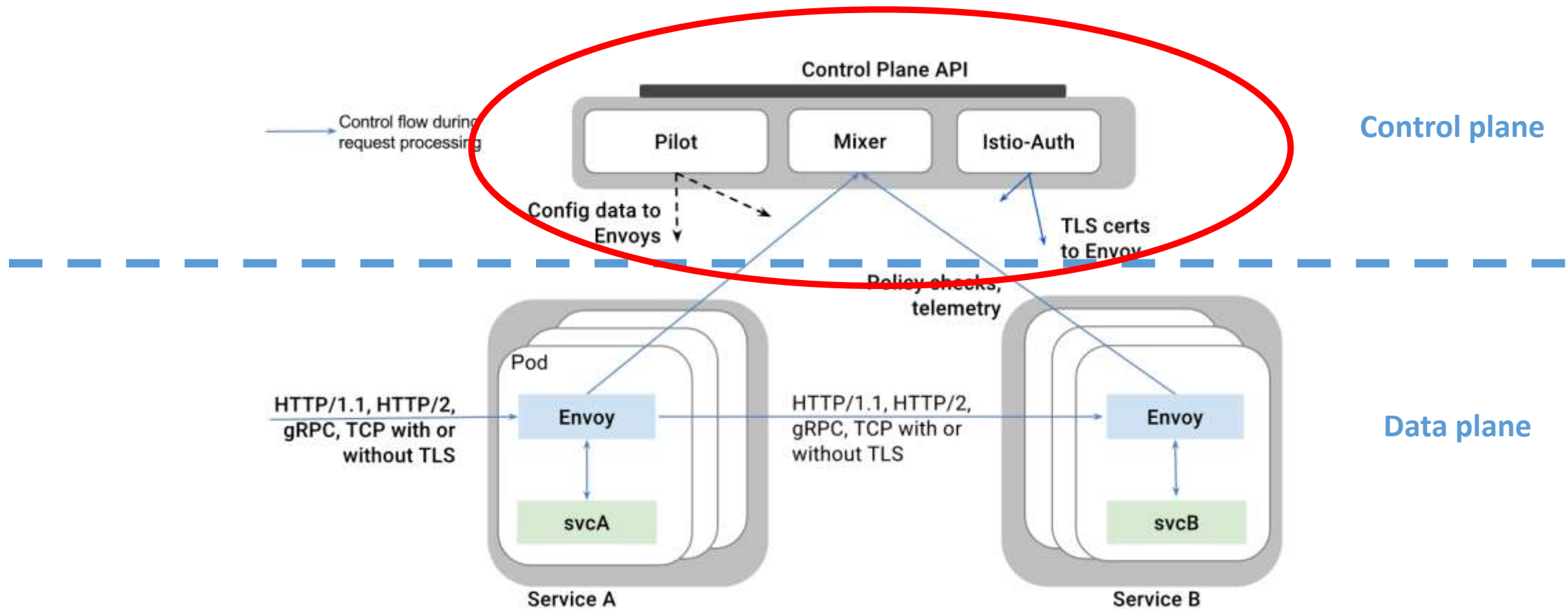
In this blog post, I'll share a little bit about this feature and what it means for Linkerd and Istio users.

### MANAGING MICROSERVICE COMMUNICATION

Istio is "an open platform to connect, manage, and secure microservices". Linkerd is "an open source service mesh for cloud-native applications". Both projects seek to improve the runtime behavior of a microservice application, especially the communication between individual microservices.

Managing this runtime behavior is important because, while we have tools like Docker and Kubernetes to manage deployment and execution of service code, that's not enough to make applications resilient and manageable. The way that microservices interact with each other at runtime—how traffic load flows through the system—needs to be monitored, managed, and controlled. As William writes in "What is a service mesh? And why do I need one?",

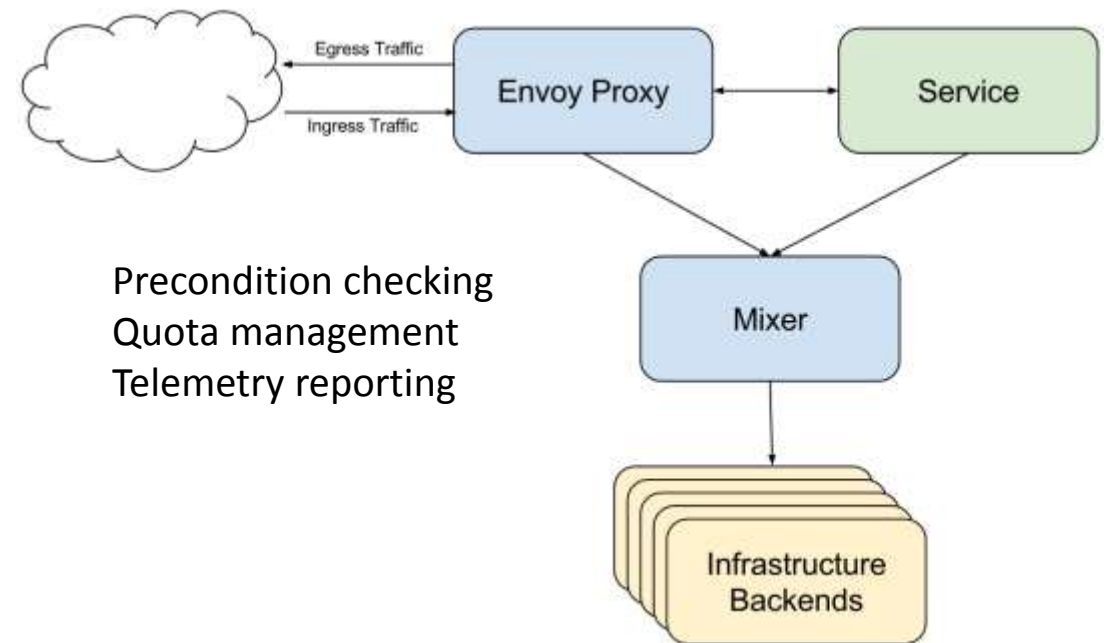
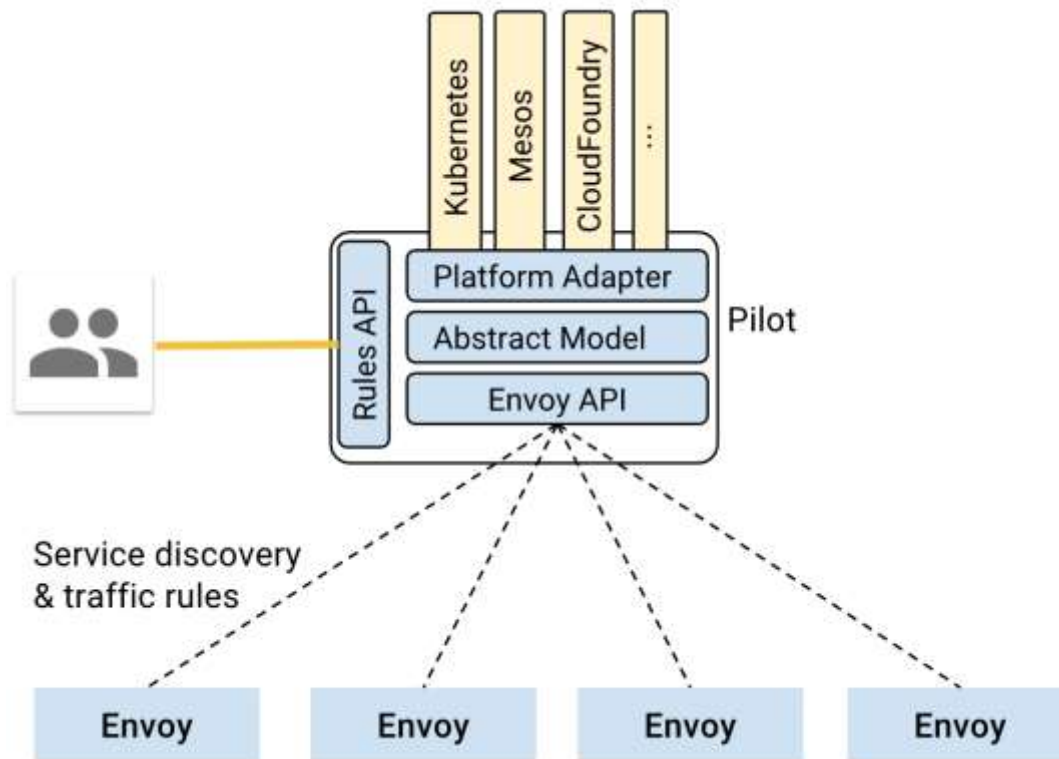
# Control Plane / Data Plane (Istio example)



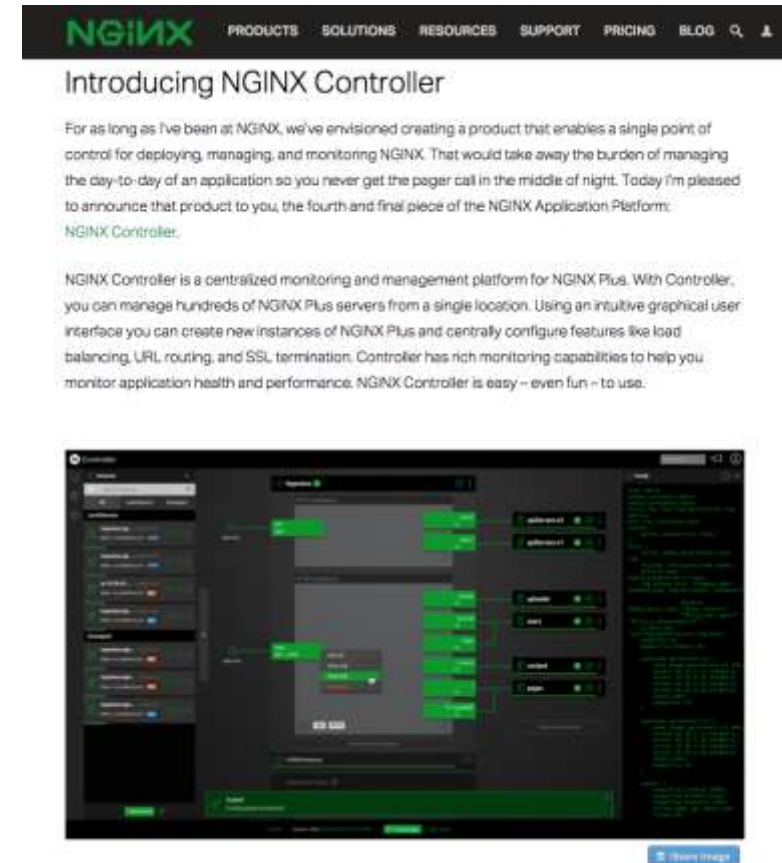
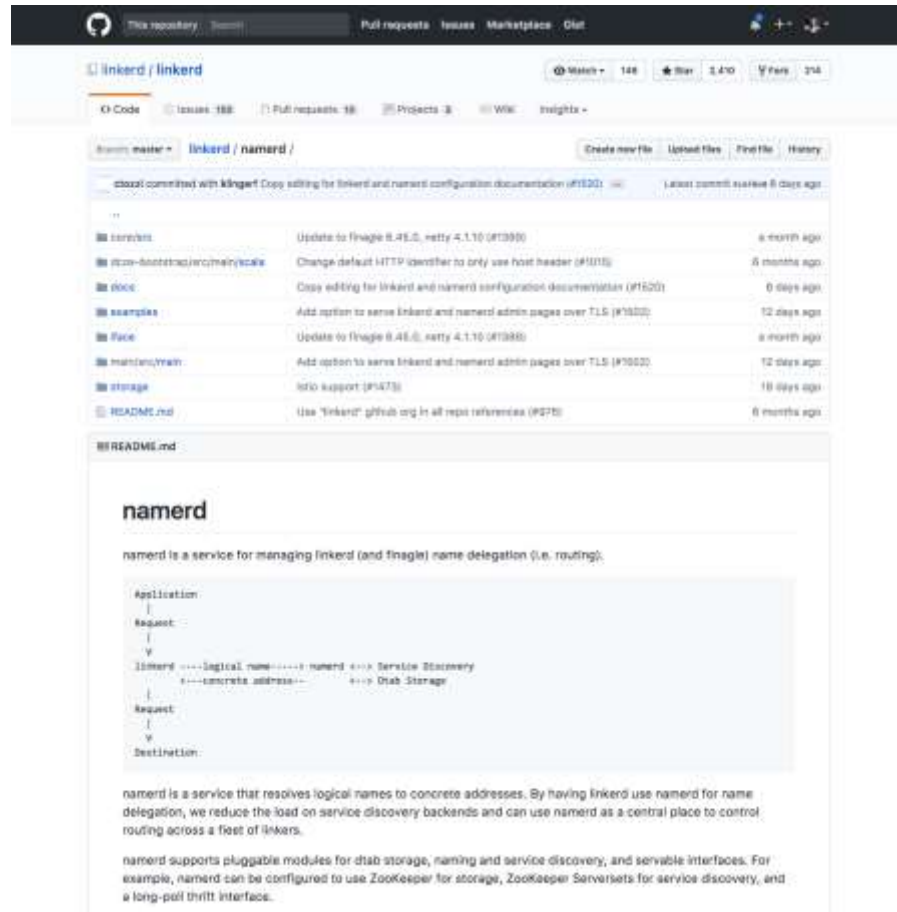
<https://istio.io/docs/concepts/what-is-istio/overview.html>



# Istio control plane: Pilot and Mixer



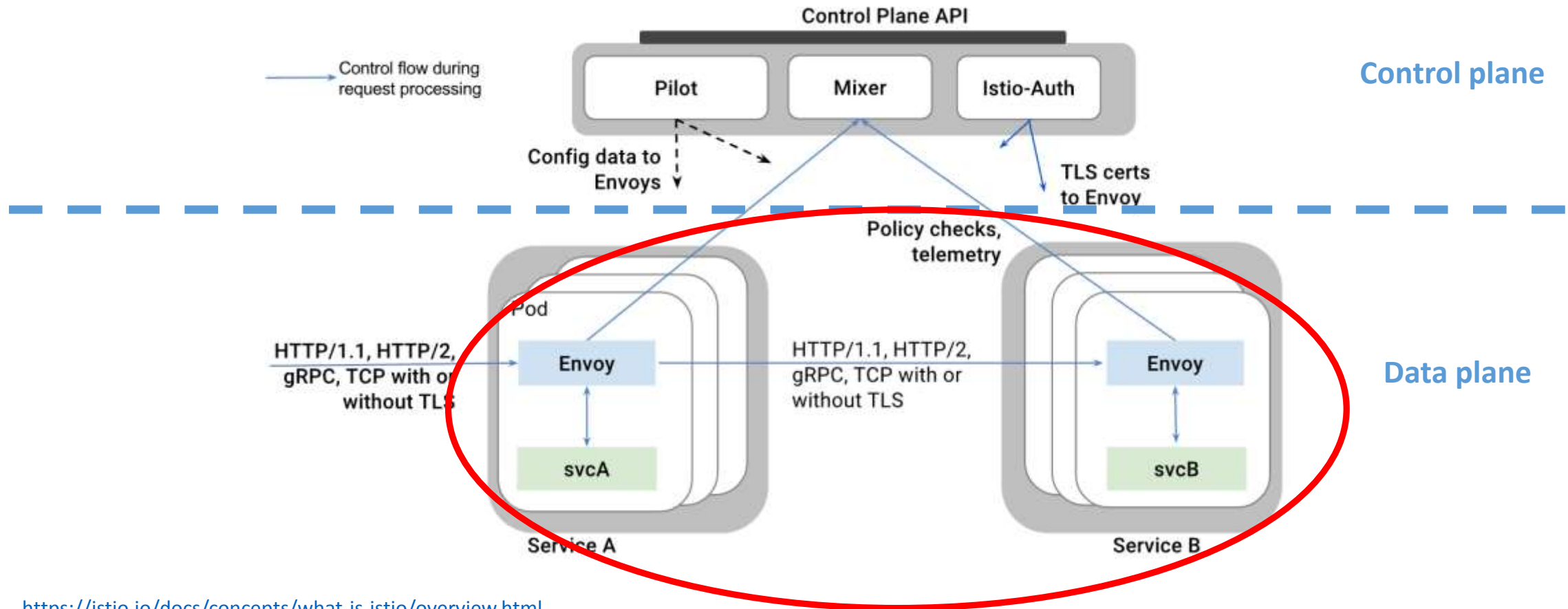
# Linkerd and NGINX control plane



NGINX Controller makes it easier to monitor and manage large NGINX Plus deployments.

[www.infoq.com/news/2017/09/nginx-platform-service-mesh](http://www.infoq.com/news/2017/09/nginx-platform-service-mesh)

# Control Plane / Data Plane (Istio example)



<https://istio.io/docs/concepts/what-is-istio/overview.html>

# Service Mesh data plane (proxy) comparison

Name	Linkerd	Envoy	Nginx Plus
Website	<a href="https://linkerd.io/">https://linkerd.io/</a>	<a href="https://lyft.github.io/envoy/">https://lyft.github.io/envoy/</a>	<a href="https://www.nginx.com/products/">https://www.nginx.com/products/</a>
Licence	AL2.0	AL2.0	Commercial
Custodians	Buoyant (Twitter pedigree)	Lyft (Matt Klein)	Nginx Inc
Build language	Scala/Rust	C++11	C
Deployment platforms	Any	Any (Istio is K8s only)	Any
Control plane	APIs, namerd, Istio [exp]	APIs, Istio (Google/IBM), Nelson (Verizon)	API, config files, service discovery
Protocols supported	HTTP/1.1, HTTP/2 [exp], gRPC, TCP	HTTP/2, gRPC, TCP	HTTP/1.1, TCP, UDP, <i>gRPC (Q4 2017)</i>
Service discovery integration	File, Consul, ZK, etcd, K8s, Marathon	Static, Strict/logical DNS, SDS (REST API)	File, Consul, ZK, etcd, K8s, Marathon (nixy)
Container Deployment	Per Host/Sidecar	PerHost/Sidecar (Istio sidecar only)	Sidecar
Monitoring	Exports stat (linkerd-viz)	Exports stats (e.g. Prometheus/Grafana)	Exports stats, and bespoke dashboard
Request Tracing	Zipkin (service must pass http headers)	Zipkin (service must pass http headers)	Possible, but bespoke
Commercial support	Yes (Buoyant)	No (Not yet)	Yes (Nginx Inc)
Production Deploys	PayPal, Expedia, AOL, Monzo	Lyft, Verizon, JDI	Buzzfeed, WIX, ARM

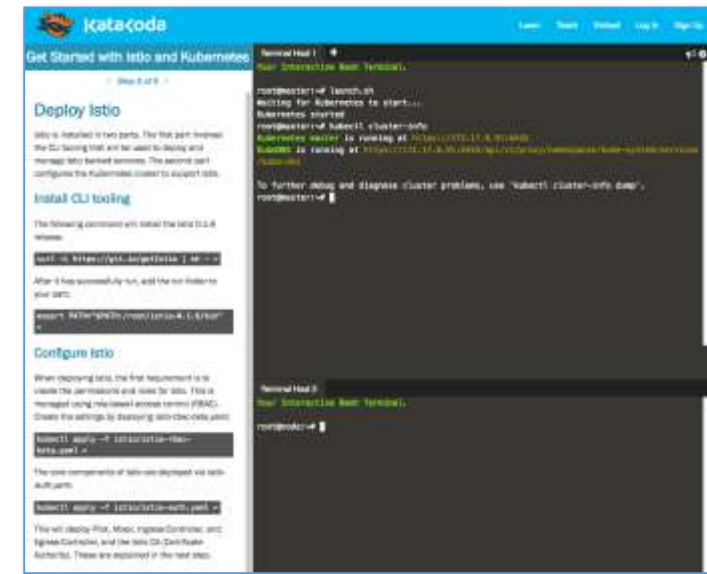
# Getting started

- Articles:

- Linkerd + Kubernetes:
  - <https://buoyant.io/2016/10/04/a-service-mesh-for-kubernetes-part-i-top-line-service-metrics/>
- Installing Istio:
  - <https://istio.io/docs/tasks/installing-istio.html>
- Tim Perrett: Envoy with Nomad and Consul
  - <http://timperrett.com/2017/05/13/nomad-with-envoy-and-consul>
- NGINX Fabric Model:
  - <https://www.nginx.com/blog/microservices-reference-architecture-nginx-fabric-model/>

- Videos:

- William Morgan - Linkerd:
  - <https://www.youtube.com/watch?v=0xYSy6OmjUM>
- Christian Posta – Envoy/Istio:
  - <http://blog.christianposta.com/microservices/00-microservices-patterns-with-envoy-proxy-series/>
- Matt Klein – Envoy:
  - <https://www.youtube.com/watch?v=RVZX4CwKhGE>
- Kelsey Hightower - Istio:
  - [https://www.youtube.com/watch?v=s4qasWn\\_mFc](https://www.youtube.com/watch?v=s4qasWn_mFc)



<https://www.katacode.com/courses/istio/deploy-istio-on-kubernetes>

# Common Questions

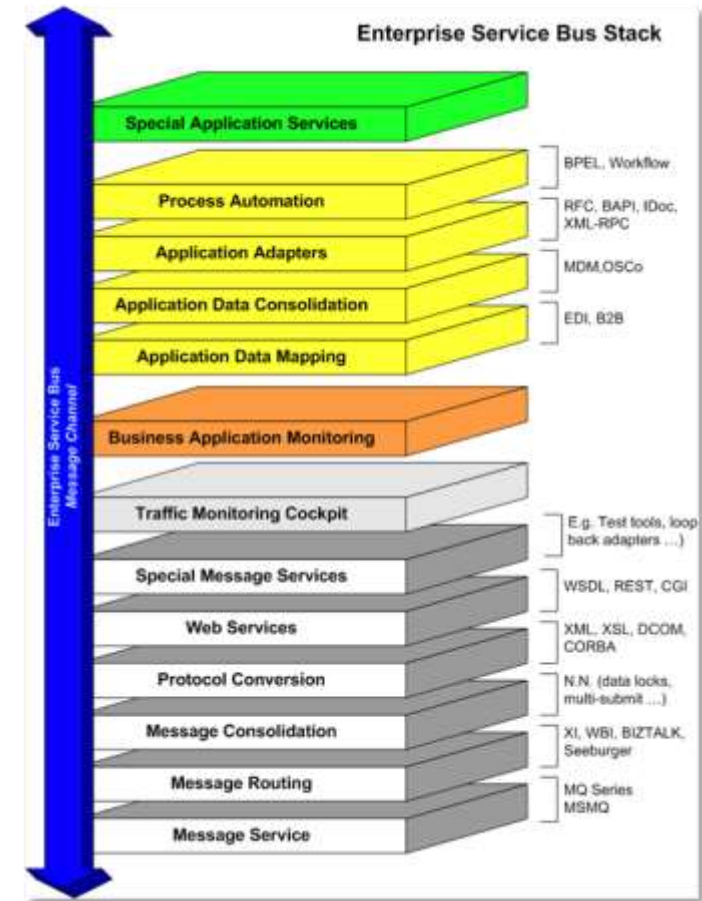
“But what about...”

# How do service meshes relate to (Edge/API) gateways?

- Gateways primarily sit on the edge of your network
  - Perform ingress cross-cutting concerns (authn/z, rate limiting, logging etc)
- My experience
  - NGINX
  - Cloud implementations
  - Traefik and Datawire's Ambassador (based on Envoy)
- Some are vying to act as the communication backbone too
  - Kong API
  - Mulesoft
  - NGINX

# Isn't this just ESB 2.0 or “web scale” ESB

- No
  - At least not yet...
- ESB development was vendor-driven
- Overly centralised/coupled/conflated
  - Process choreography
  - Document transformation
  - Tight integration with vendor products



[https://en.wikipedia.org/wiki/Enterprise\\_service\\_bus#/media/File:ESB\\_Component\\_Hive.png](https://en.wikipedia.org/wiki/Enterprise_service_bus#/media/File:ESB_Component_Hive.png)



# Isn't this just adding more network hops?

- Maybe... It depends on your network config
- ...but good (infrastructure) architecture is all about
  - Choosing the right abstraction
  - Making trade-offs
  - Separation of concerns
- Make an educated choice with your platform, and make it explicitly

# Shouldn't this be part of the “platform”?

- Yep...
- And it probably will be in the near future
  - But expect much innovation (and change) over the next 6-12 months
- Assess if it will be beneficial for your organisation to leverage this now

# Who owns the Service Mesh? Dev, SREs, Ops?

- Yes...
- As mentioned earlier
  - We work with a sociotechnical system when delivering value/software
  - Everything is context dependent (on your organisation)
  - But deployment descriptor and service mesh config can provide good dev/ops collaboration zones as part of the “platform”
- Make a decision, communicate it, and regularly retrospect

# So, Service Mesh all-the-things... right?

- No...
  - It's all about context and trade-offs
- Service meshes are great for point-to-point RPC
- Messaging is useful to decouple services in space and time
  - Async work queues, pub/sub, topics e.g. RabbitMQ
  - Distributed txn logs and stream processing e.g. Kafka

# Look for Problems, Not Solutions

# Use cases for Service Meshes

- Real-time (operator) configuration and observability
- The evolution from complicated to complex systems
- Monolith-to-service migration
  - All components can use the same communication fabric
  - Multi-platform / hybrid cloud etc
  - Routing (shadow traffic, A/B, canarying etc)

# Wrapping Up

# In conclusion...

- Deploying cloud native services/functions to a “platform” is essential
  - Service meshes are responsible for platform comms e.g. routing, traffic shifting
- Need clear collaboration zones for dev/ops/platform
  - Service meshes can provide collaboration zone for run-time config of comms
- Managing lots of out-of-process communication going “over the wire”
  - Service meshes can provide observability, reliability and fault tolerance



# Massive thanks to everyone who has helped!

- William Morgan @ Buoyant
- Owen Garrett @ NGINX
- Christian Posta @ Red Hat
- Matt Klein @ Lyft
- Shriram Rajagopalan (Istio-users)
- Louis Ryan (Istio-users)
- Varun Talwar @ Google
- Many more from the community

# Thanks for listening...

[bit.ly/2jWDSF7](http://bit.ly/2jWDSF7)

Twitter: @danielbryantuk

Email: [daniel.bryant@specto.io](mailto:daniel.bryant@specto.io)

Writing: [www.infoq.com/profile/Daniel-Bryant](http://www.infoq.com/profile/Daniel-Bryant)

Talks: [www.youtube.com/playlist?list=PLoVYf\\_0qOYNeBmrpjuBOOAqJnQb3QAEtM](https://www.youtube.com/playlist?list=PLoVYf_0qOYNeBmrpjuBOOAqJnQb3QAEtM)



Available Q2 2018!

