Giulio Santoli - @gjuljo

Microservices Architectures: Become a Unicorn and create a project like Netflix, Twitter or Hailo

Agenda

1. Here comes the Unicorns
2. Monolith v.s. Microservices
3. Services for the Microservices:
    a. Service Discovery
    b. Load Balancing
    c. Fault Tolerance
    d. Edge Server and Data Aggregation
4. The Death Star strikes back
5. Dockerize it
6. Centralized Logging and Monitoring
7. Running on Cloud Foundry
8. Continous Delivery, DevOps and NoOps
9. What's next?

Agenda

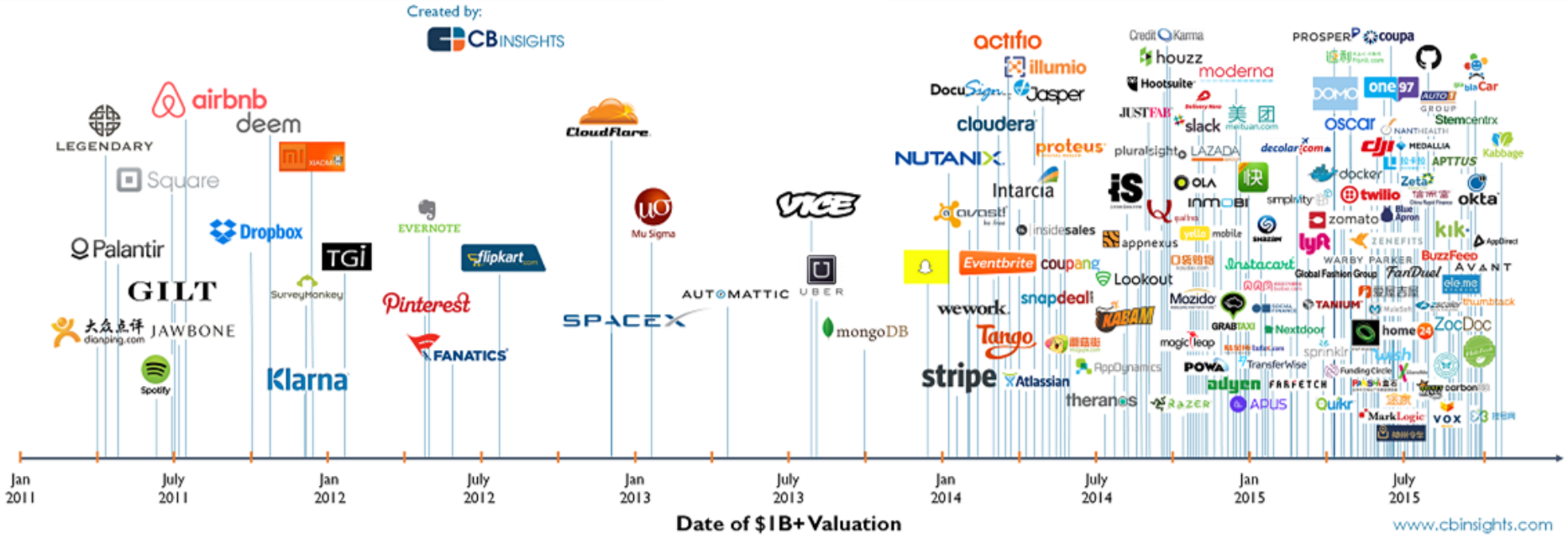
1. Here comes the Unicorns
2. Monolith v.s. Microservices
3. Services for the Microservices:
    a. Service Discovery
    b. Load Balancing
    c. Fault Tolerance
    d. Edge Server and Data Aggregation
4. The Death Star strikes back
5. Dockerize it
6. Centralized Logging and Monitoring
7. Running on Cloud Foundry
8. Continous Delivery, DevOps and NoOps
9. What's next?

Are Unicorns Real?

Aileen Lee, the founder of Cowboy Ventures, in 2013 chose the word **unicorn** to describe the **winner of all winners** among software start-ups, and typically valued at $1 billion or more by investors.

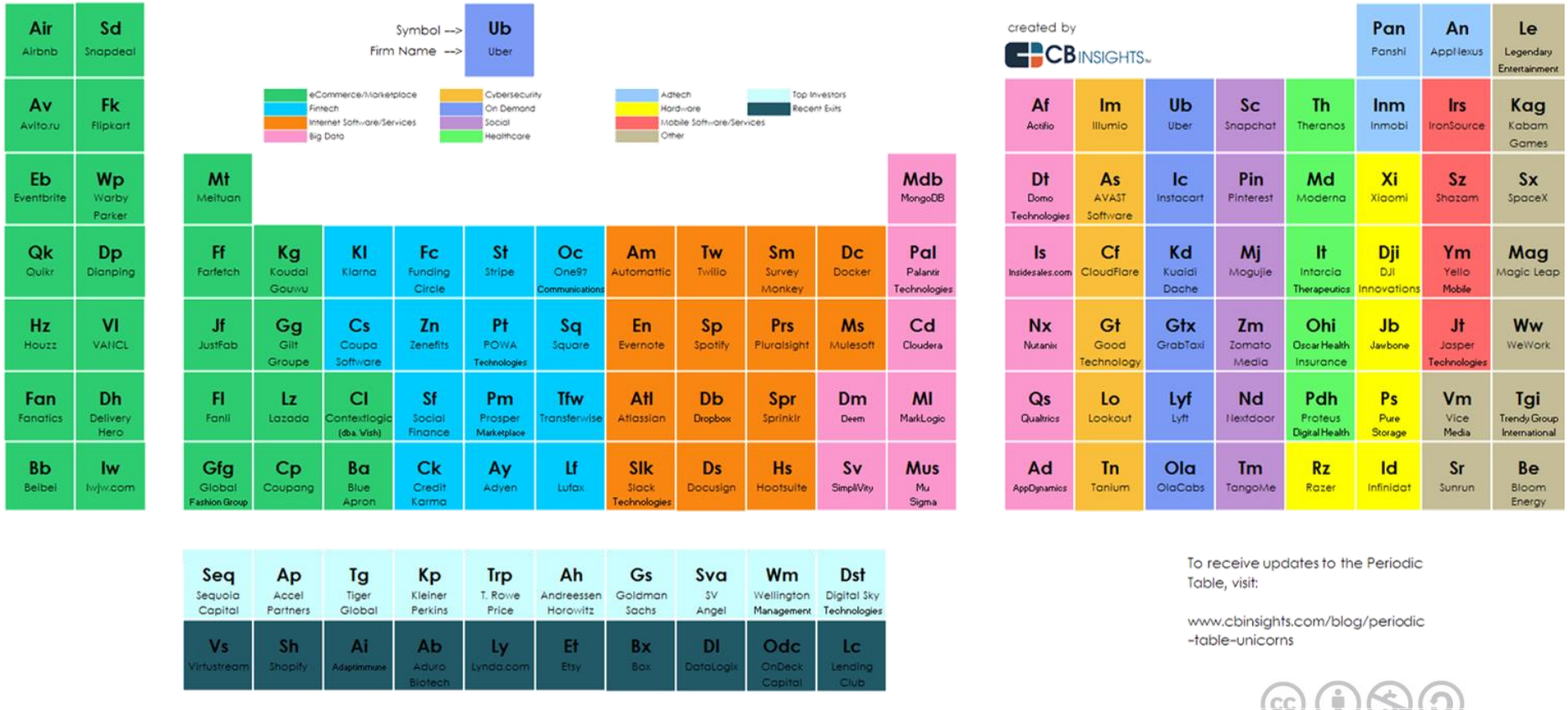
http://www.nytimes.com/2015/08/24/technology/the-unicorn-club-now-admitting-new-members.html
http://bits.blogs.nytimes.com/2015/07/05/unicorns-a-fitting-word-for-its-time-and-place

The Increasingly Crowded Unicorn Club (since 2011)



The Periodic Table of Unicors

A breakdown of companies valued at $1B+ by sector (6/11/2015)



To receive updates to the Periodic Table, visit:

www.cbinsights.com/blog/periodic-table-unicorns

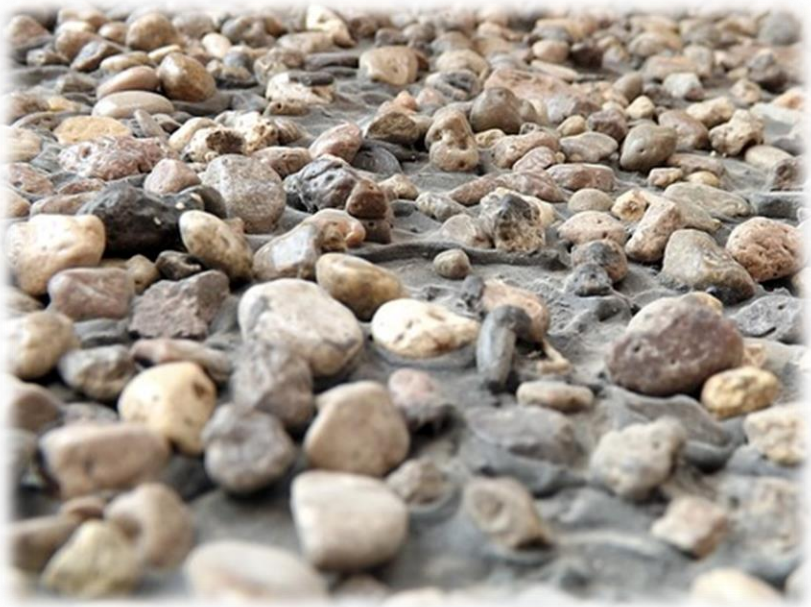
What do these Unicorns have in common?

(among others) An aggresive adoption of the **microservices** architecture approach.

Agenda

1. Here comes the Unicorns
2. **Monolith v.s. Microservices**
3. Services for the Microservices:
    a. Service Discovery
    b. Load Balancing
    c. Fault Tolerance
    d. Edge Server and Data Aggregation
4. The Death Star strikes back
5. Dockerize it
6. Centralized Logging and Monitoring
7. Running on Cloud Foundry
8. Continous Delivery, DevOps and NoOps
9. What's next?

Monolith or Microservices?



lightweight, small footprint, follow Single Responsibility Pattern

"**Microservices** are **tiny apps** talking with **uniform interface** installed as well-behaved **OS services**."

by Eduard Sizovs (www.slideshare.net/eduardsi)

RESTful. decoupled, scalable, discoverable.

self-contained, run with a single one-liner command

What is a Microservice Architecture?

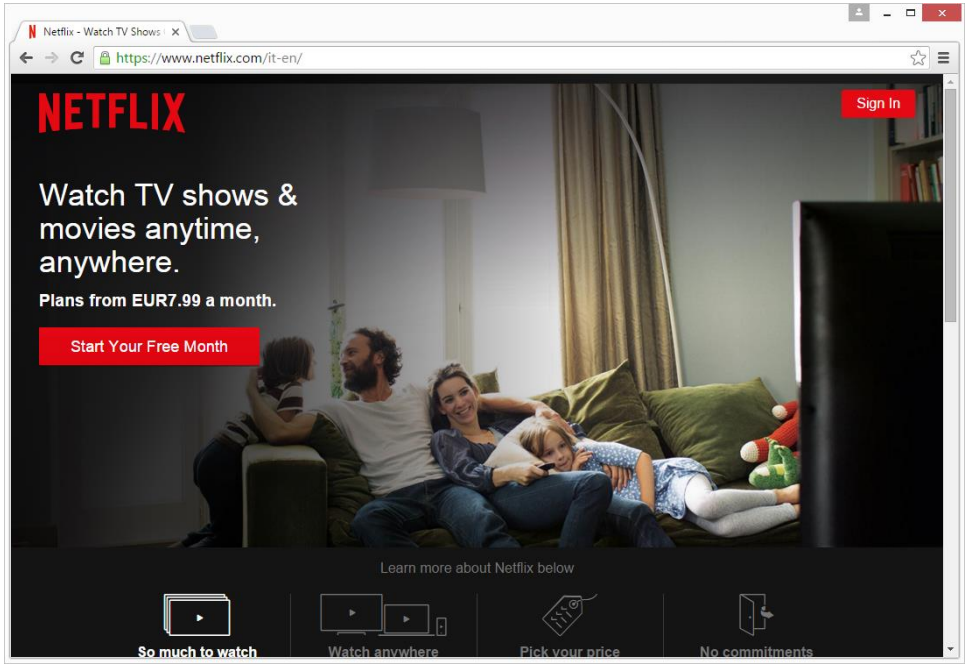
“Loosely coupled service oriented architecture with bounded contexts”

Adrian Cockcroft
@adrianco
(Battery Ventures)

NETFLIX

Netflix is the world's leading Internet television network with over 69 million members in over 60 countries enjoying more than 100 million hours of TV shows and movies per day, including original series, documentaries and feature films.
Members can watch as much as they want, anytime, anywhere, on nearly any Internet-connected screen.

http://ir.netflix.com/



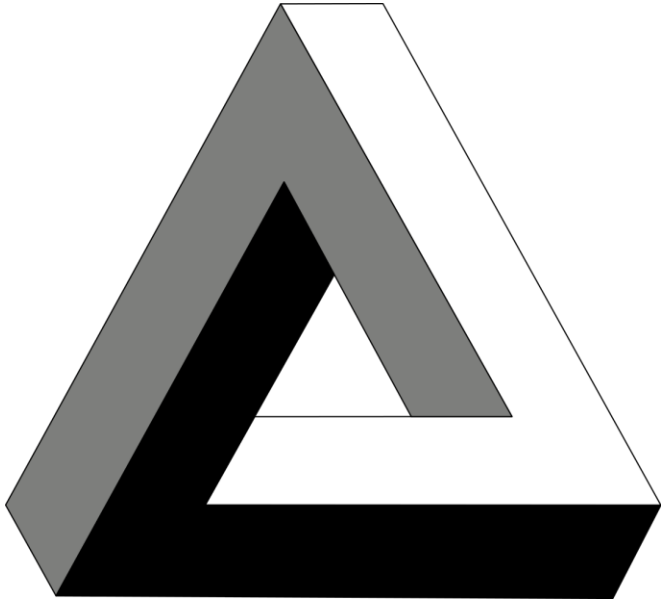
NETFLIX | OSS

Netflix moved from a Monolithic Architecture to Microservices, building a set of infrastructure services, then open-sourced as Netflix OSS.

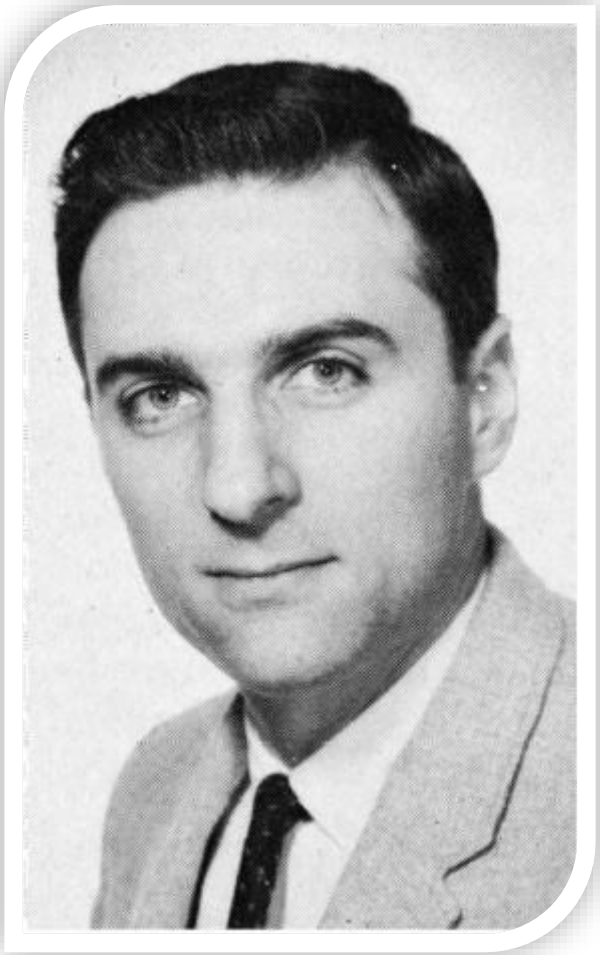
https://netflix.github.io
http://cloud.spring.io/spring-cloud-netflix/spring-cloud-netflix.html

Be aware!
It's not just a matter of

Practices



Architectures Organizations

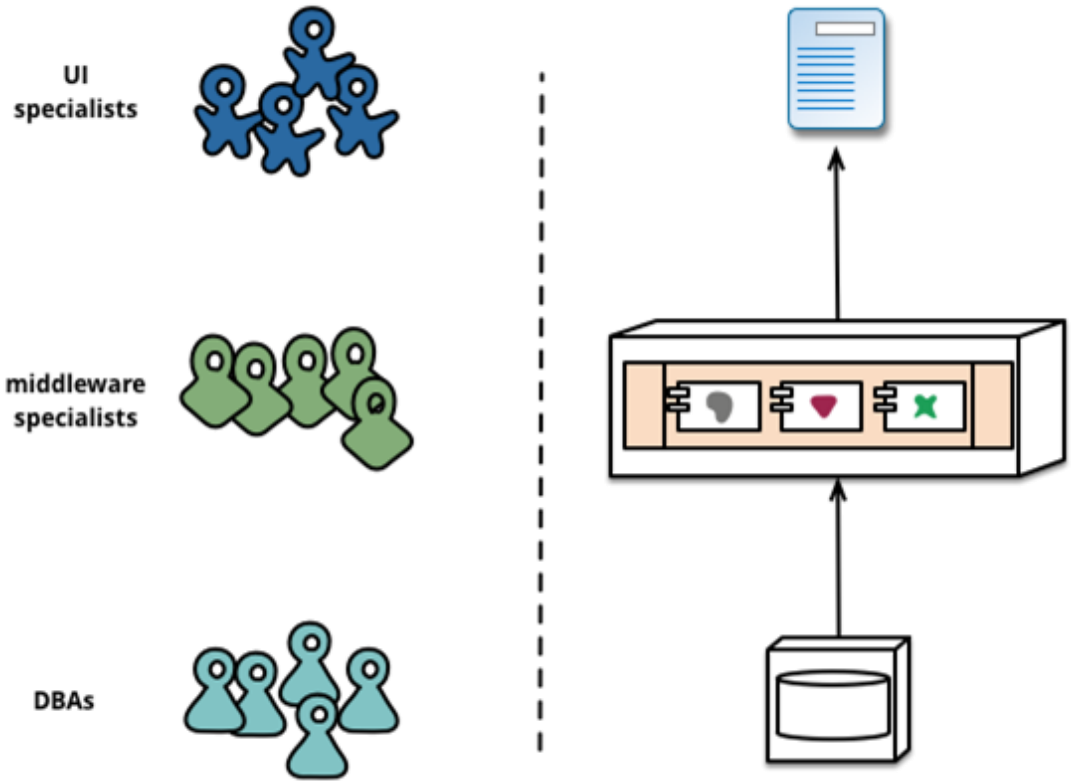


Conway's Law

"Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations"

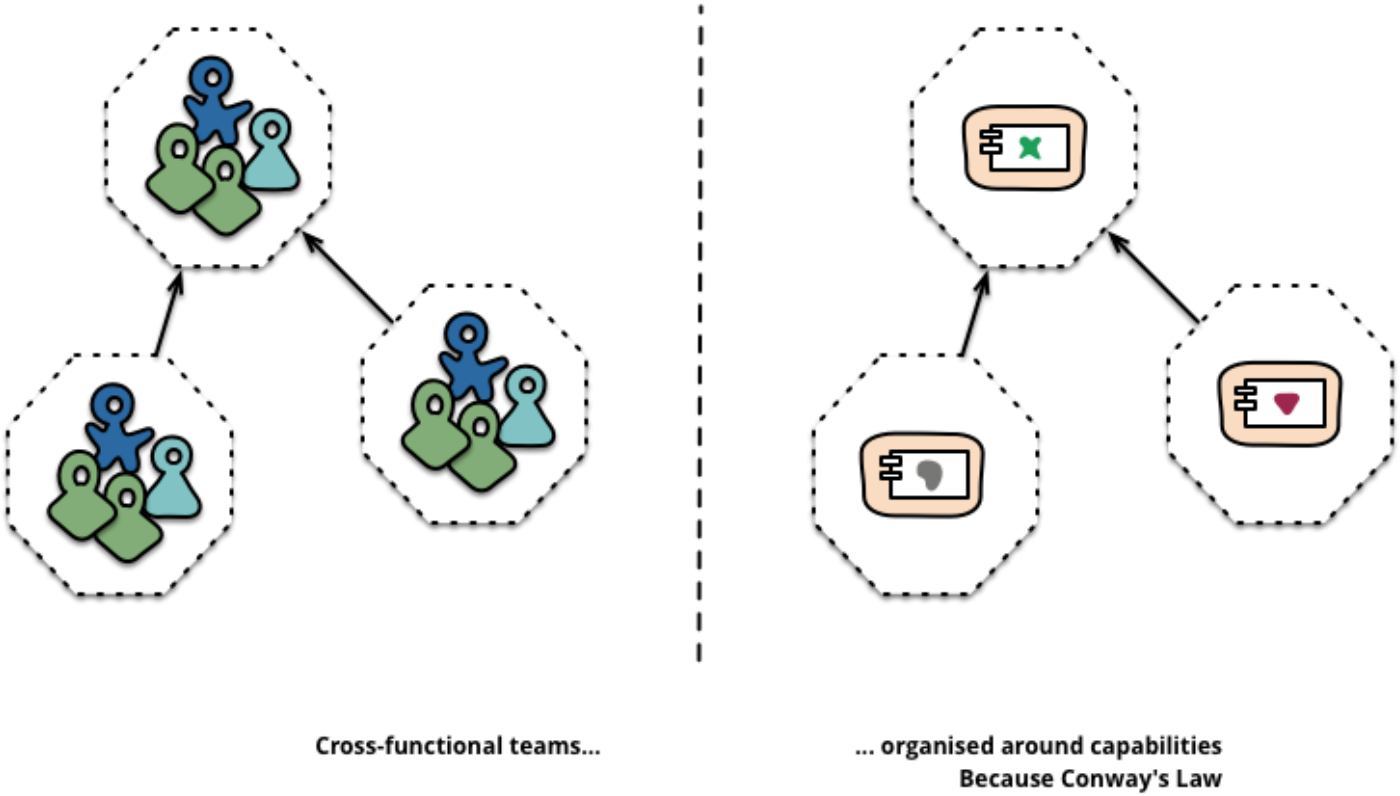
Conway, Melvin E. (1968)

Siloed functional teams lead to Siloed application architectures



http://martinfowler.com/articles/microservices.html

Cross-functional teams lead to capability-oriented architectures



Cross-functional teams...

... organised around capabilities
Because Conway's Law

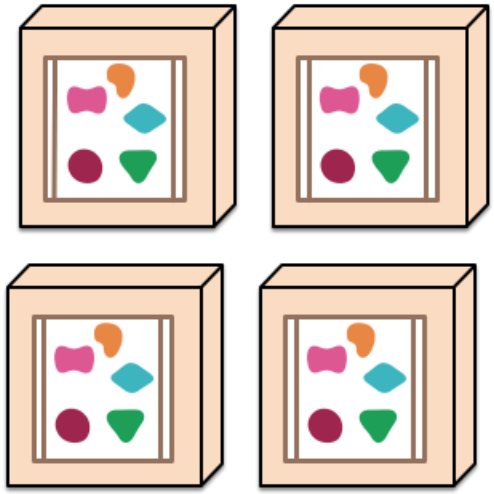
http://martinfowler.com/articles/microservices.html

Microservices v.s. Monolithic

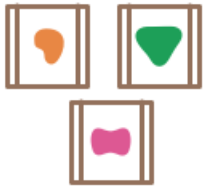
Scalability

A monolithic application puts all its functionality into a single process...

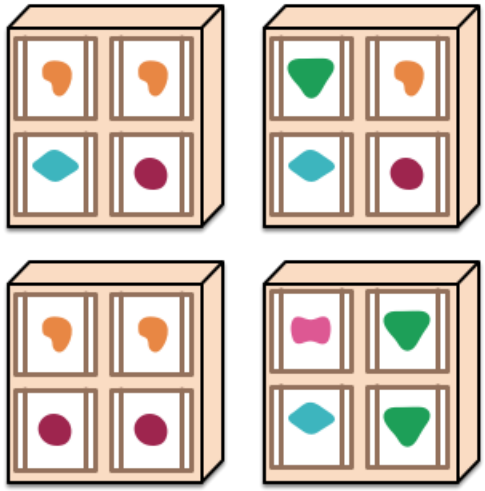
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



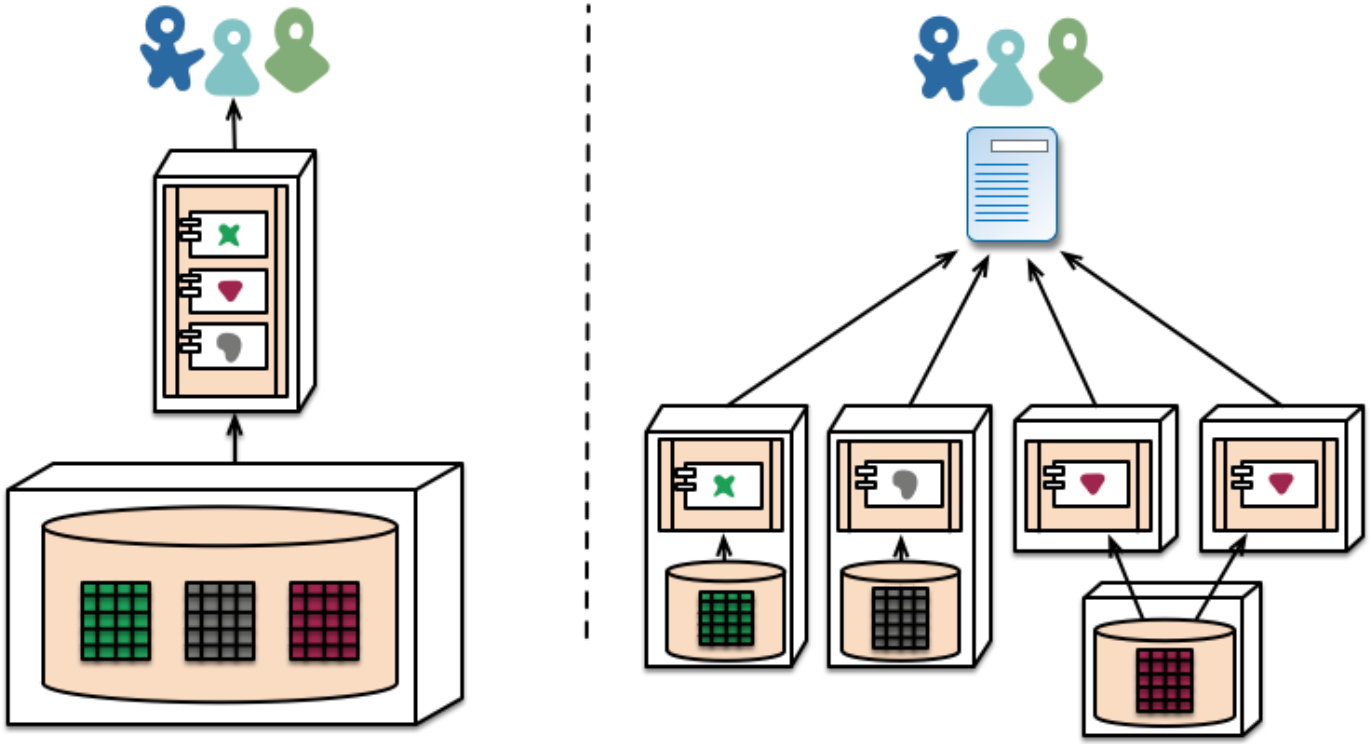
... and scales by distributing these services across servers, replicating as needed.



http://martinfowler.com/articles/microservices.html

Microservices v.s. Monolithic

Database Polyglotism



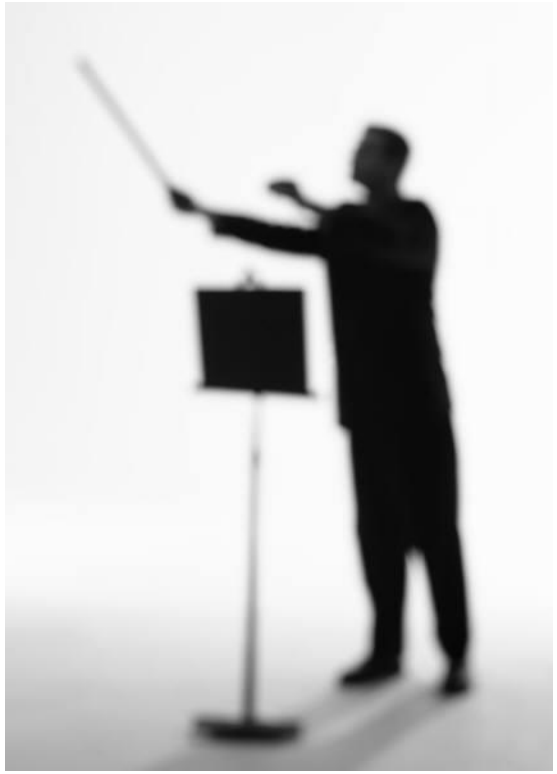
monolith - single database

microservices - application databases

http://martinfowler.com/articles/microservices.html

Microservices v.s. Monolithic

Orchestration v.s. Choreography

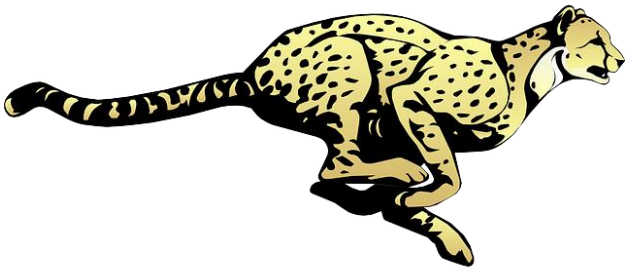


in summary...

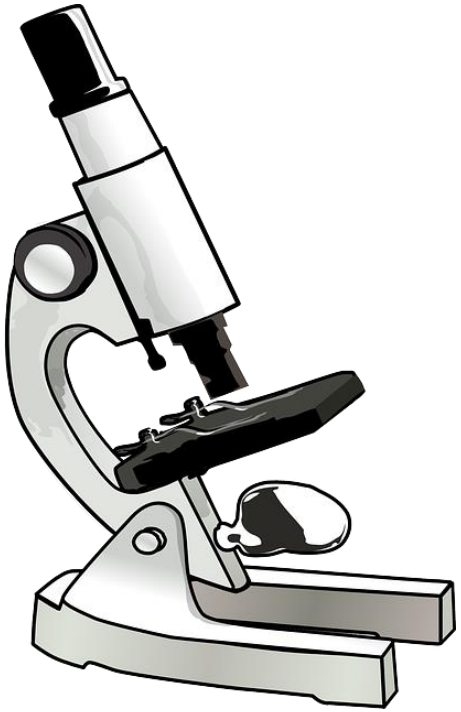
microservices should be:
- **cheap to replace**
- **quick to scale**
- **withstand failure**

James Lewis
(ThoughWorks)

and should allow you to go as “fast as possible”



How Big is Micro?



Rules of Thumb

2-Pizza Team Rule (Jeff Bezos, Amazon)

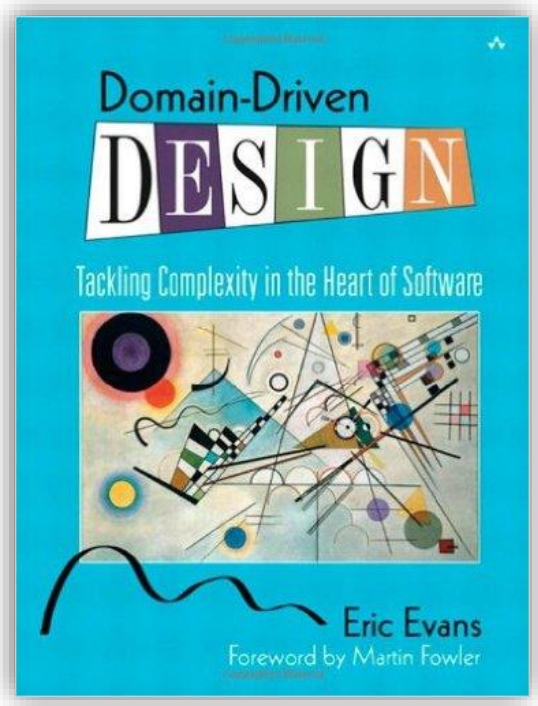
~~No more than 300 lines of code~~

Focus on what Matters

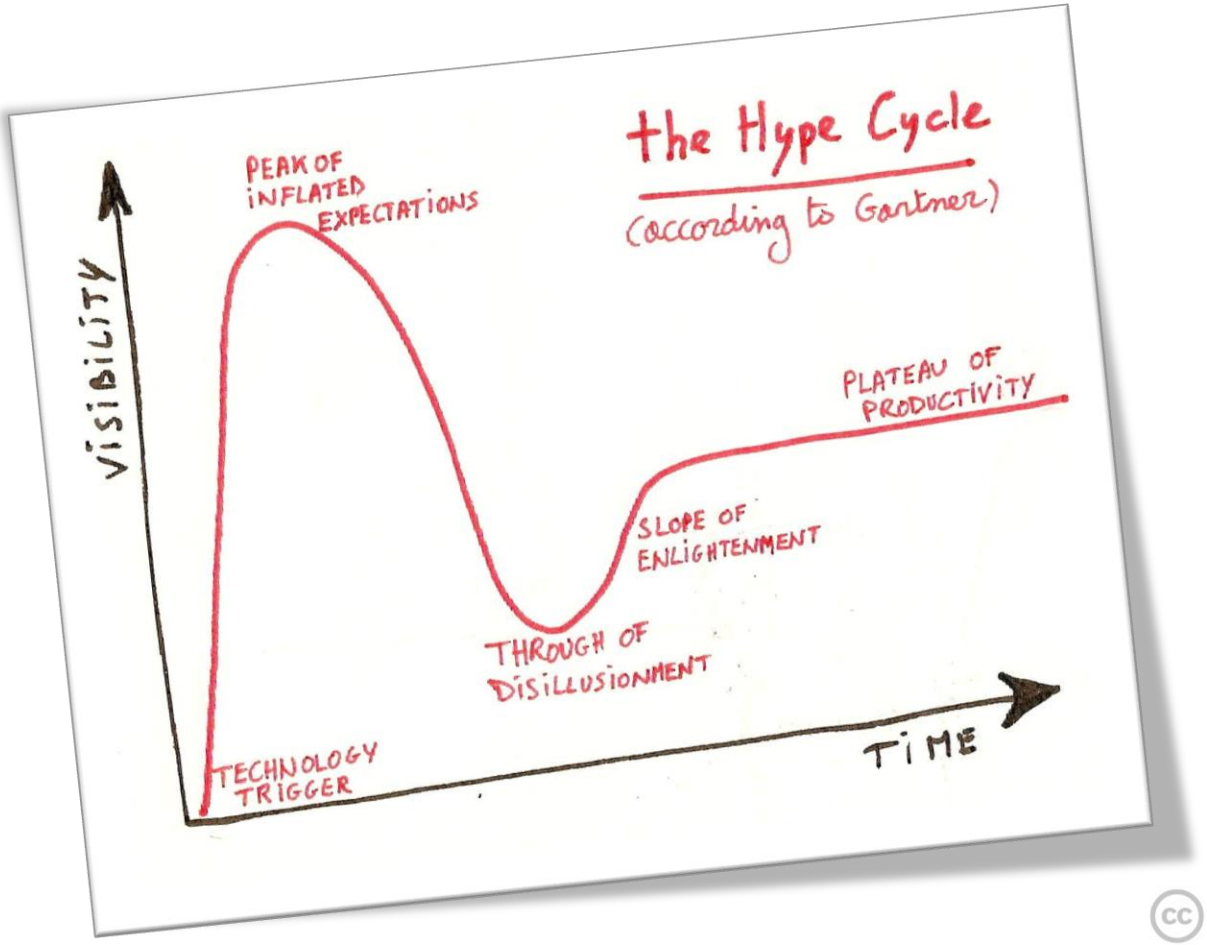
Adopt DDD Bounded Contexts

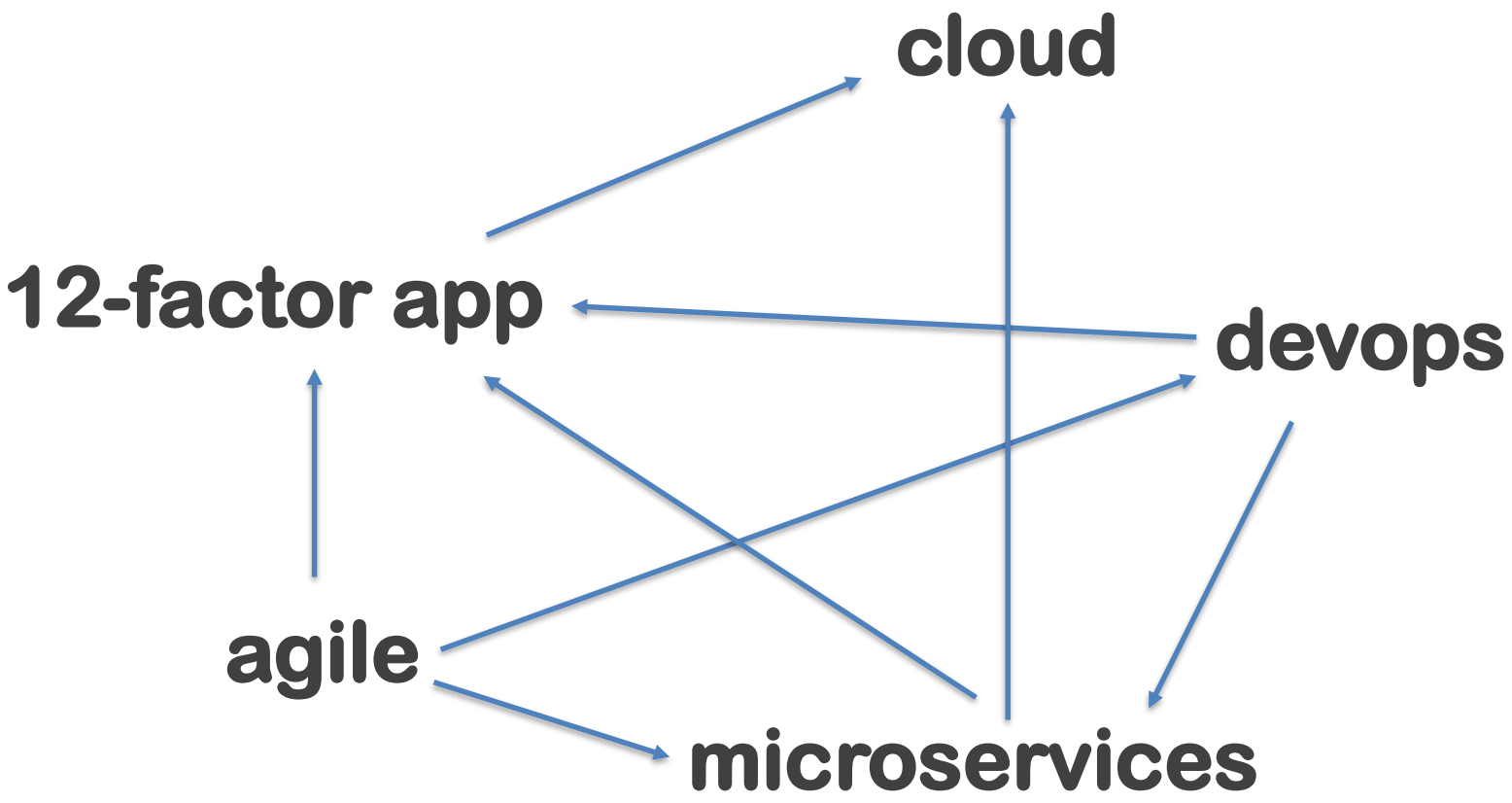
“The heart of software is its ability to solve domain-related problems for its user.”

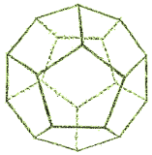
Eric Evans



Is this just another hype?




cloud
12-factor app
devops
agile
microservices



12-Factor Applications (http://12factor.net)

The **twelve-factor app** is a methodology for building **cloud native** applications.

I.	**Codebase**	One codebase tracked in revision control, many deploys
II.	**Dependencies**	Explicitly declare and isolate dependencies
III.	**Configuration**	Store configuration in the environment
IV.	**Backing Services**	Treat backing services as attached resources
V.	**Build, release, run**	Strictly separate build and run stages
VI.	**Processes**	Execute the app as one or more stateless processes
VII.	**Port binding**	Export services via port binding
VIII.	**Concurrency**	Scale out via the process model
IX.	**Disposability**	Maximize robustness with fast startup and graceful shutdown
X.	**Dev/prod parity**	Keep development, staging, &production as similar as possible
XI.	**Logs**	Treat logs as event streams
XII.	**Admin processes**	Run admin/management tasks as one-off processes

Spring Boot & Spring Cloud Netflix

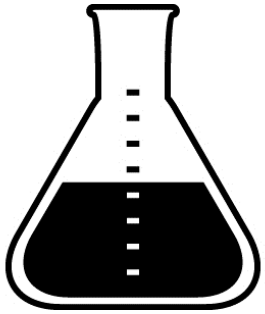
Spring Boot makes it easy to create **stand-alone**, production-grade Spring based Applications that you can "just run".

app.groovy

```
@Controller class ThisWillActuallyRun {
    @RequestMapping("/")
    @ResponseBody
    String home() {
        return "Hello World!"
    }
}
```

```
$> spring run app.groovy
```

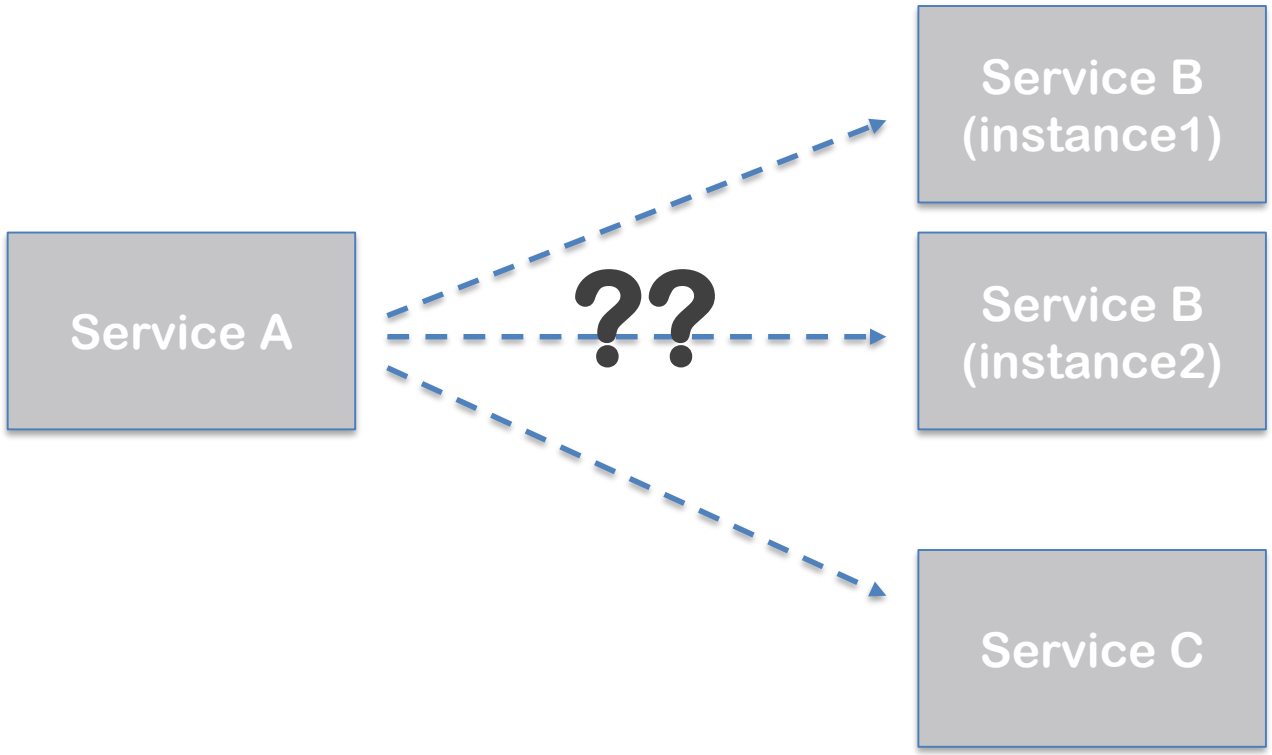
LAB: Scenario 1



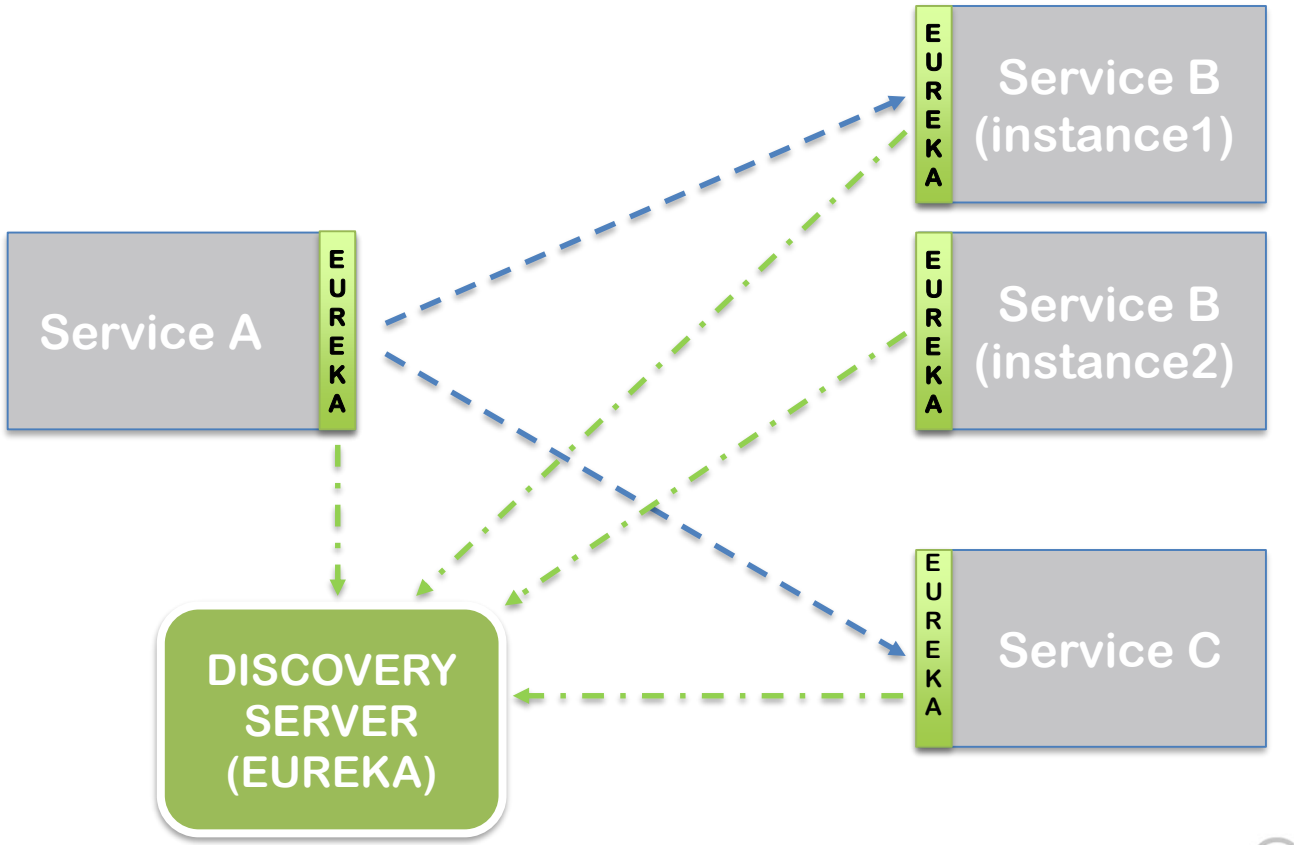
Agenda

1. Here comes the Unicorns
2. Monolith v.s. Microservices
3. Services for the Microservices:
    a. Service Discovery
    b. Load Balancing
    c. Fault Tolerance
    d. Edge Server and Data Aggregation
4. The Death Star strikes back
5. Dockerize it
6. Centralized Logging and Monitoring
7. Running on Cloud Foundry
8. Continous Delivery, DevOps and NoOps
9. What's next?

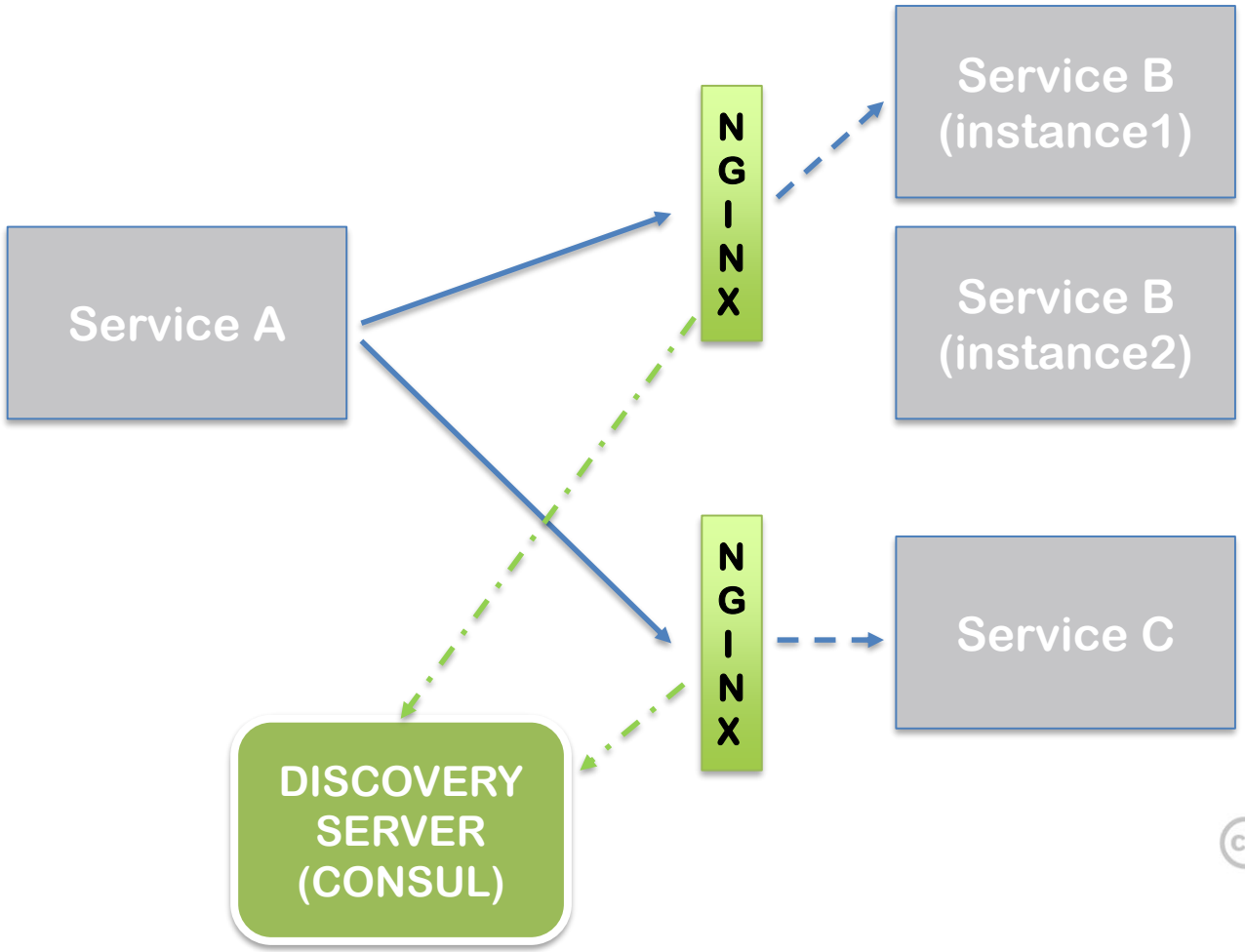
Service Discovery



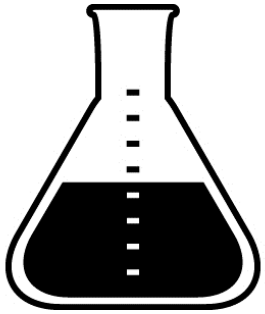
Client-Side Service Discovery (Netflix)



Server-Side Service Discovery



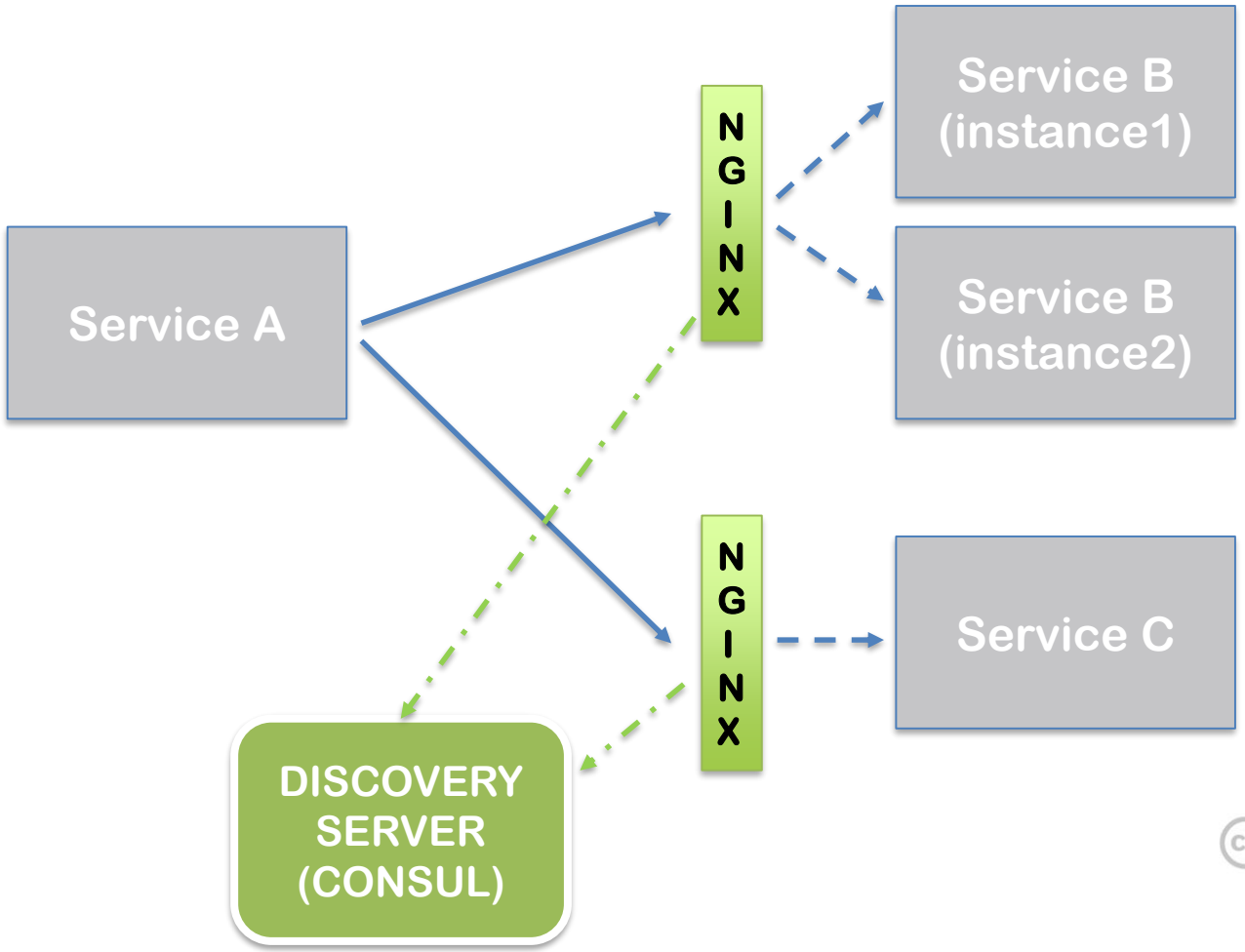
LAB: Scenario 2



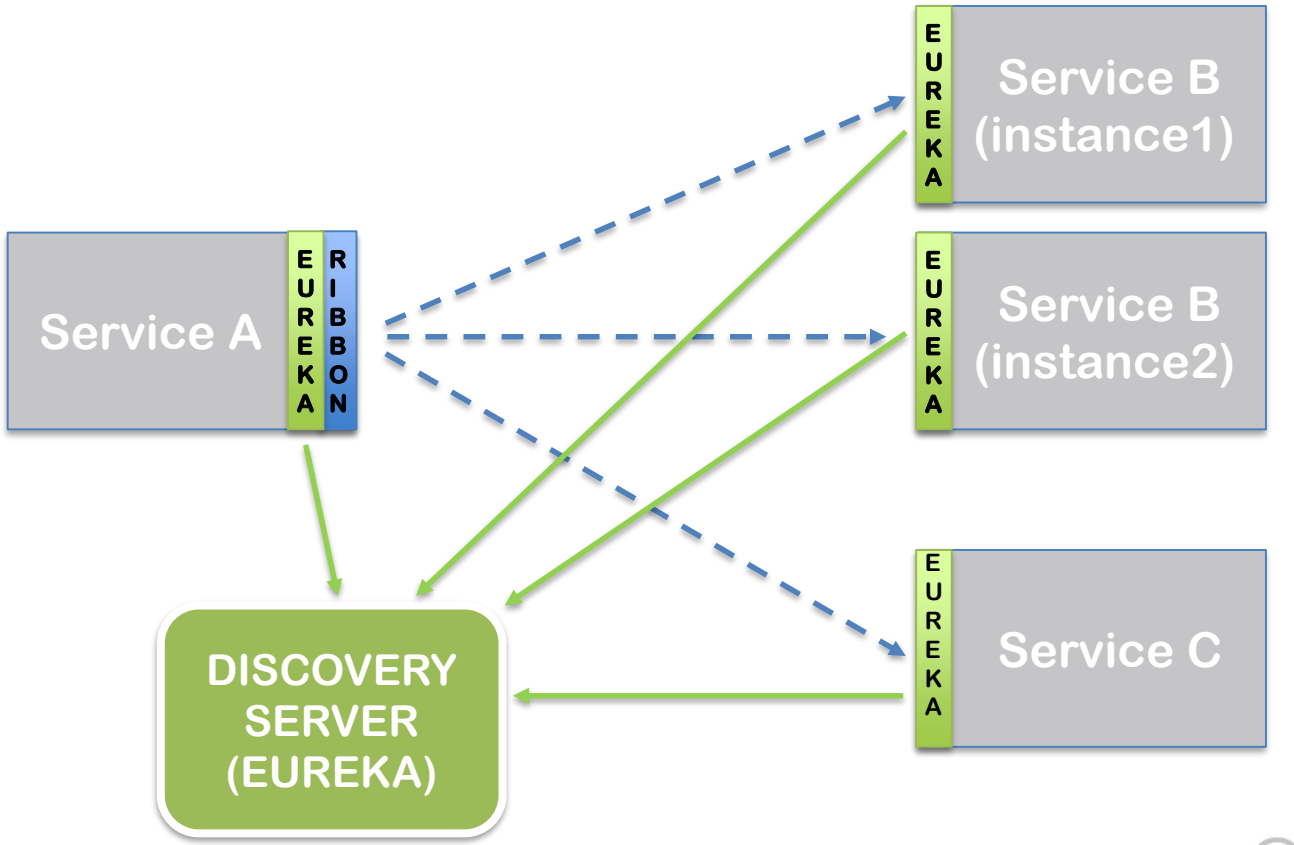
Agenda

1. Here comes the Unicorns
2. Monolith v.s. Microservices
3. Services for the Microservices:
    a. Service Discovery
    b. **Load Balancing**
    c. Fault Tolerance
    d. Edge Server and Data Aggregation
4. The Death Star strikes back
5. Dockerize it
6. Centralized Logging and Monitoring
7. Running on Cloud Foundry
8. Continous Delivery, DevOps and NoOps
9. What's next?

Load Balancing: DNS + Proxy



Client-Side Load Balancing (Neflix)



Discovery and Configuration Technolgies

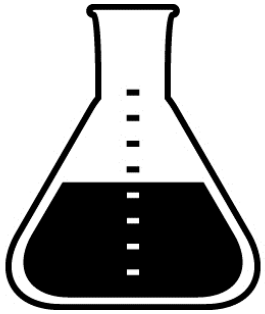
etcd is a distributed, consistent key-value store for shared configuration and service discovery.

Consul is strongly consistent datastore that uses gossip to form dynamic clusters.

ZooKeeper is a centralized service for maintaining configuration information and naming.

https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture
http://technologyconversations.com/2015/09/08/service-discovery-zookeeper-vs-etcd-vs-consul/

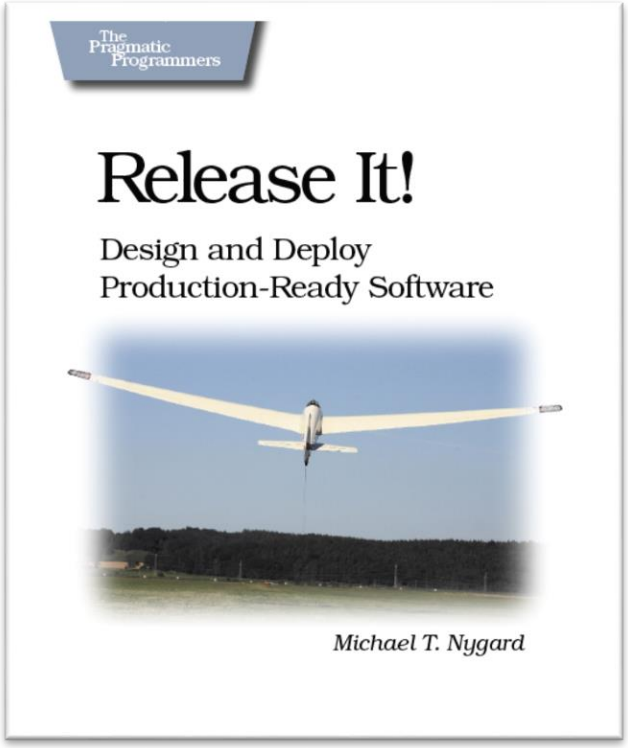
LAB: Scenario 3



Agenda

1. Here comes the Unicorns
2. Monolith v.s. Microservices
3. Services for the Microservices:
    a. Service Discovery
    b. Load Balacing
    c. **Fault Tolerance**
    d. Edge Server and Data Aggregation
4. The Death Star strikes back
5. Dockerize it
6. Centralized Logging and Monitoring
7. Running on Cloud Foundry
8. Continous Delivery, DevOps and NoOps
9. What's next?

The Exception that Grounded an Aireline



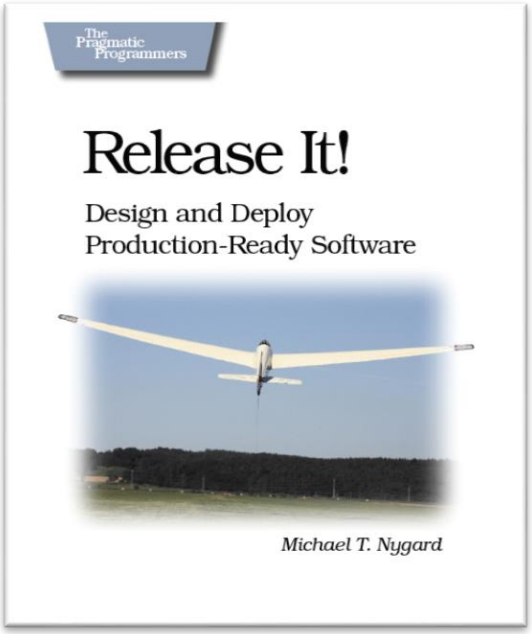
Chapter 2

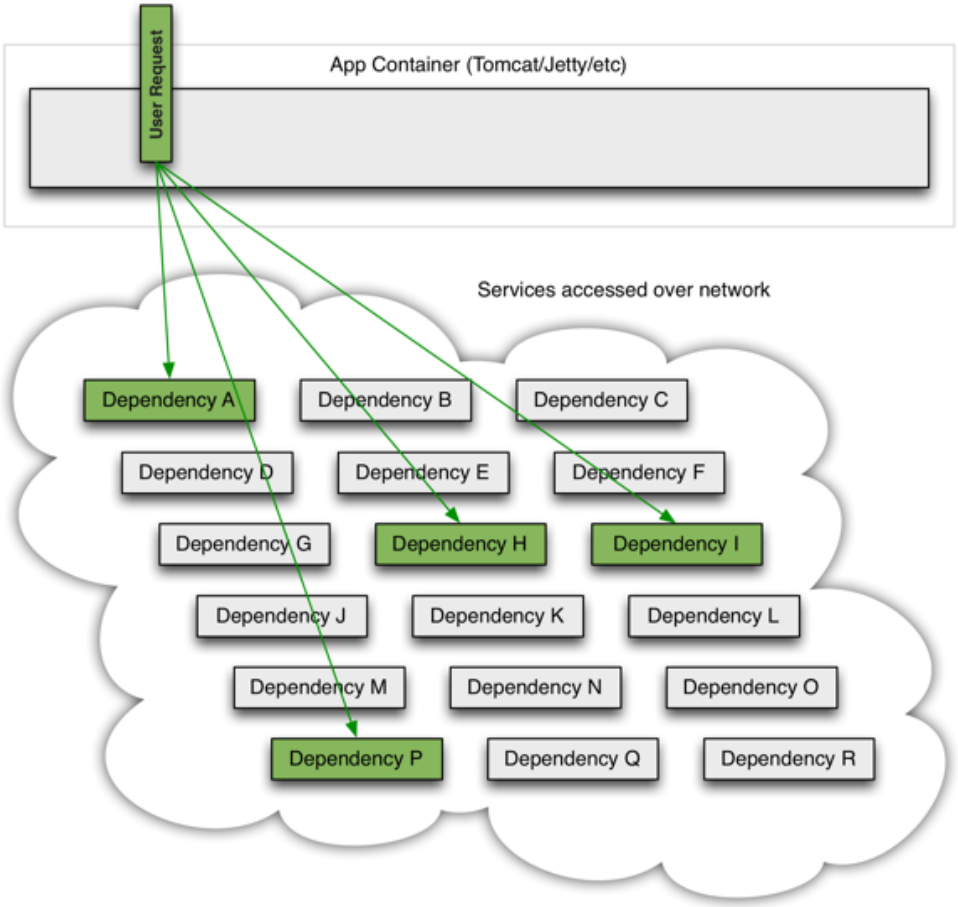
```java
public class FlightSearch implements SessionBean {
    public List lookupByCity(...) throws Exception {
        Connection conn = null;
        Statement  stmt = null;

        try {
            conn = connectionPool.getConnection();
            stmt = conn.createStatement():
            /* do lookup */
        } finally {
            if(stmt!=null) stmt.close();
            if(conn!=null) conn.close();
        }
    }
}
```

“Hope is not a design method.”

Michael T. Nygard

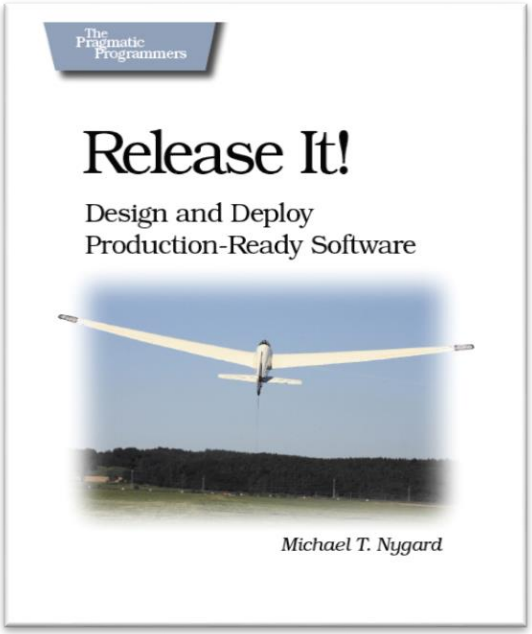


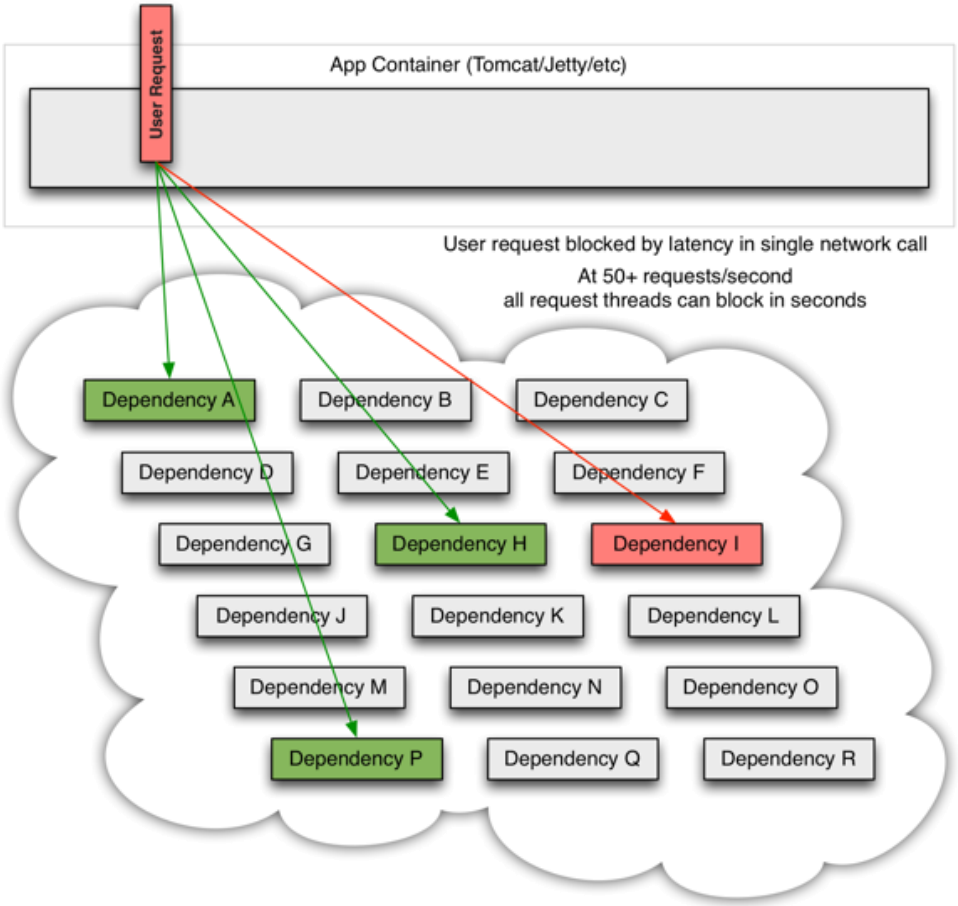


Especially in a microservices architecture, an application needs to access one or more services over the network.

“Every integration point will eventually fail in some way”

Michael T. Nygard

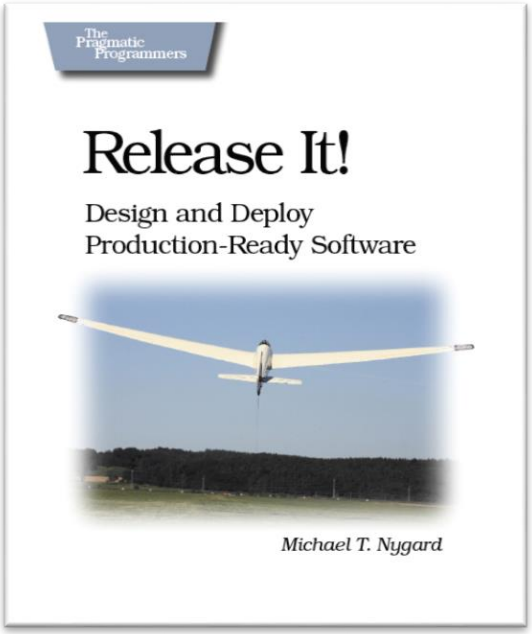


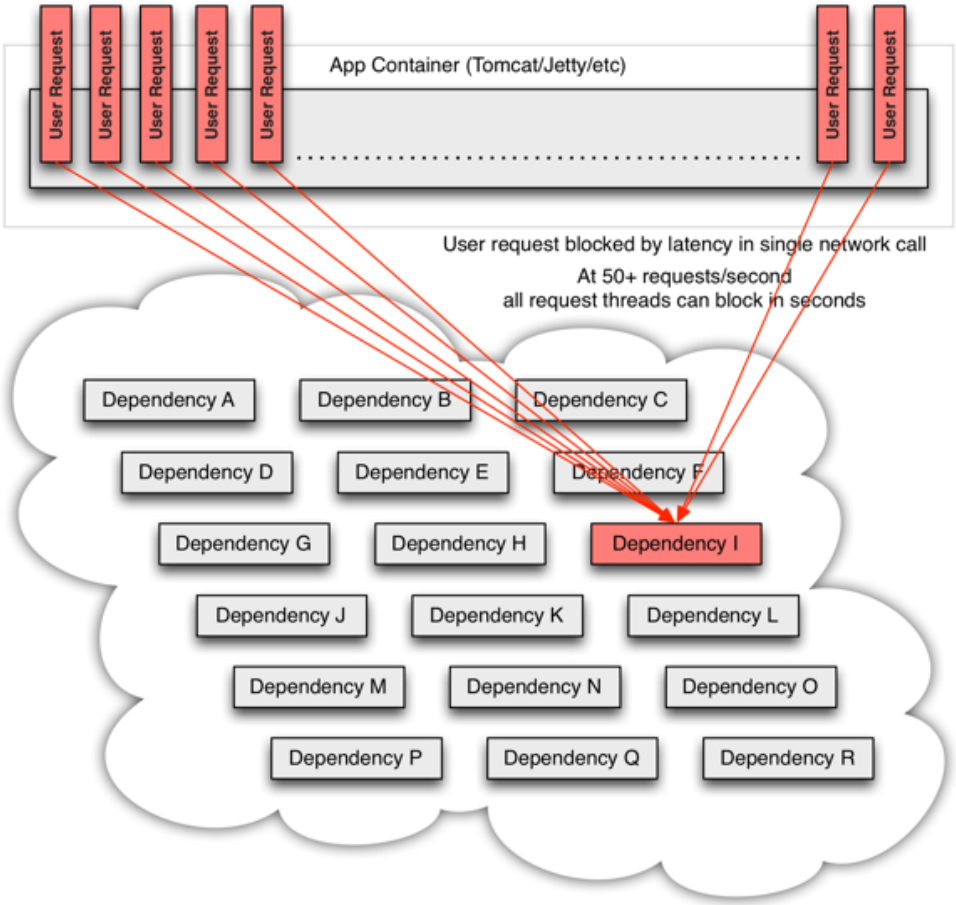


Each dependency is isolated from one other, restricted in the resources it can saturate when latency occurs, and covered in fallback logic that decides what response to make when any type of failure occurs in the dependency

“Generating a slow response is worse than refusing a connection or returning an error”

Michael T. Nygard

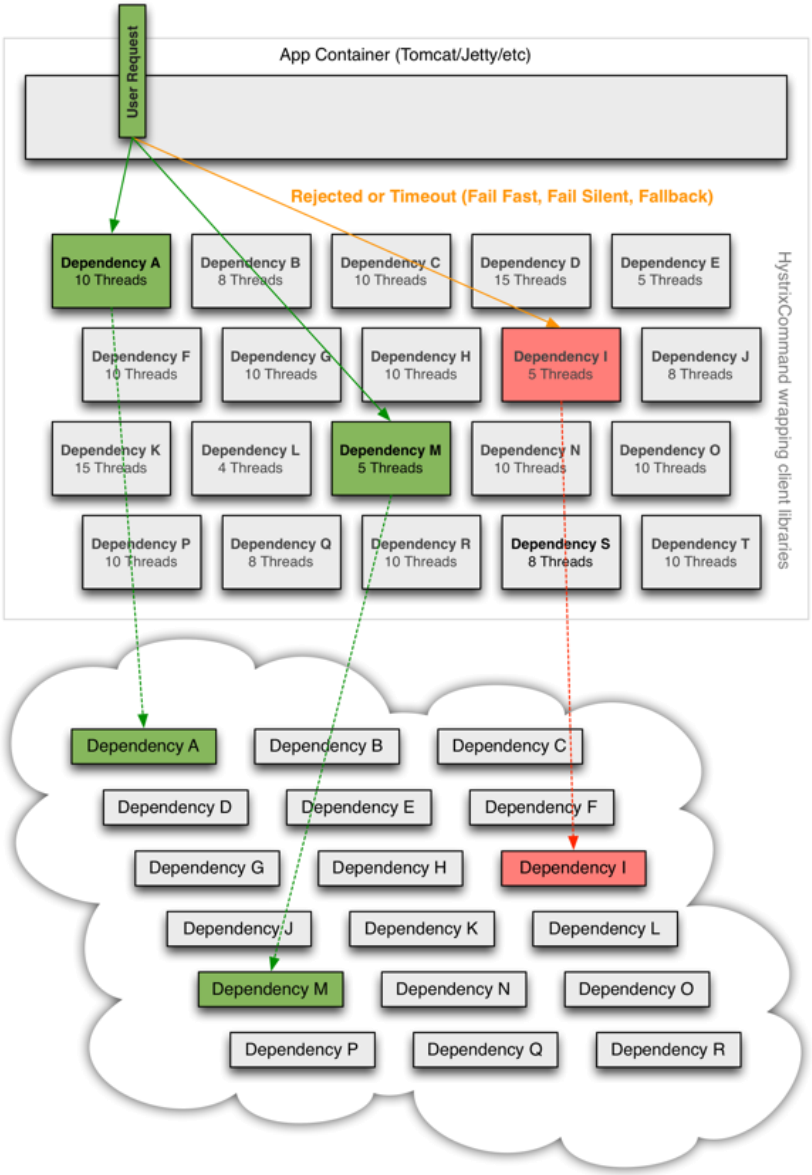




Each dependency is isolated from one other, restricted in the resources it can saturate when latency occurs, and covered in fallback logic that decides what response to make when any type of failure occurs in the dependency

Engineer the whole system for resilience

https://github.com/Netflix/Hystrix/wiki



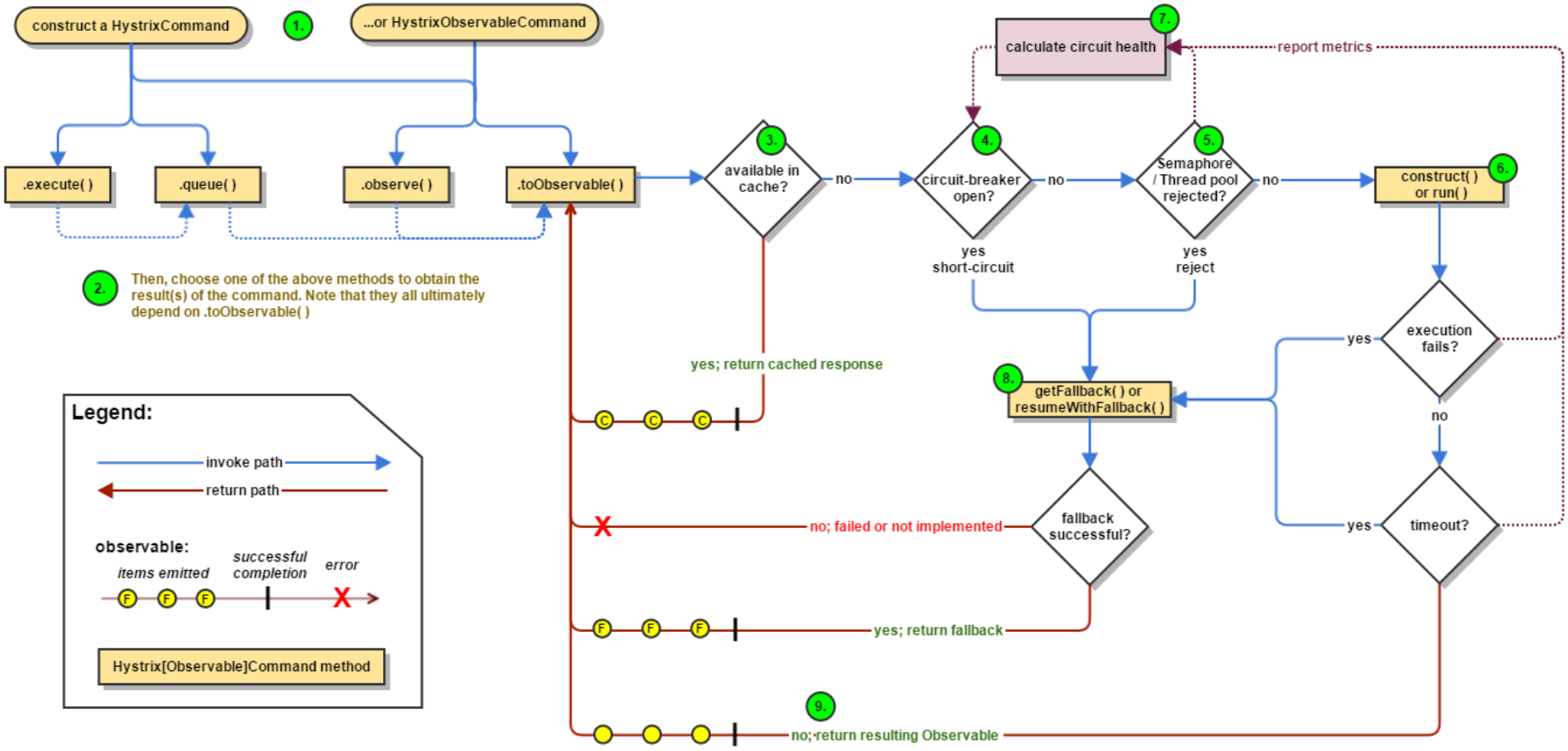
Hystrix Helps

Each dependency is isolated from one other, restricted in the resources it can saturate when latency occurs, and covered in fallback logic that decides what response to make when any type of failure occurs in the dependency.

Circuit Breaker Statechart

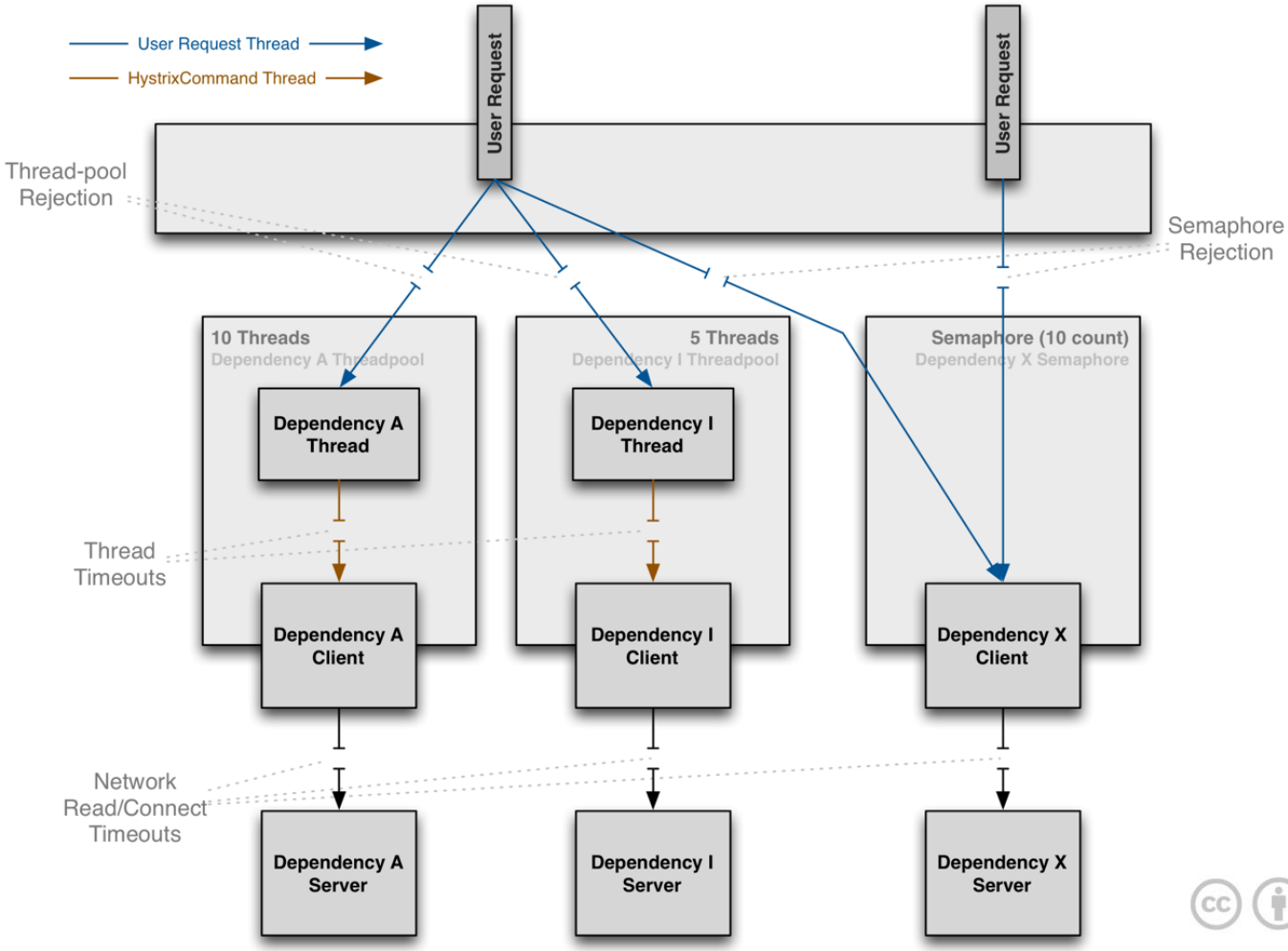
http://django-notes.blogspot.it/2012/02/release-it-design-and-deploy-production.html

Hystrix Circuit Breaker Flow Chart

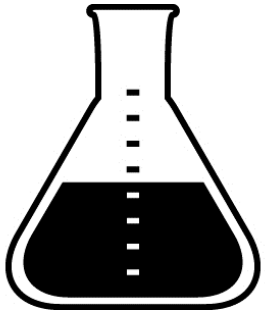


https://github.com/Netflix/Hystrix/wiki/How-it-Works

Hystrix Thread Pools and Semaphores



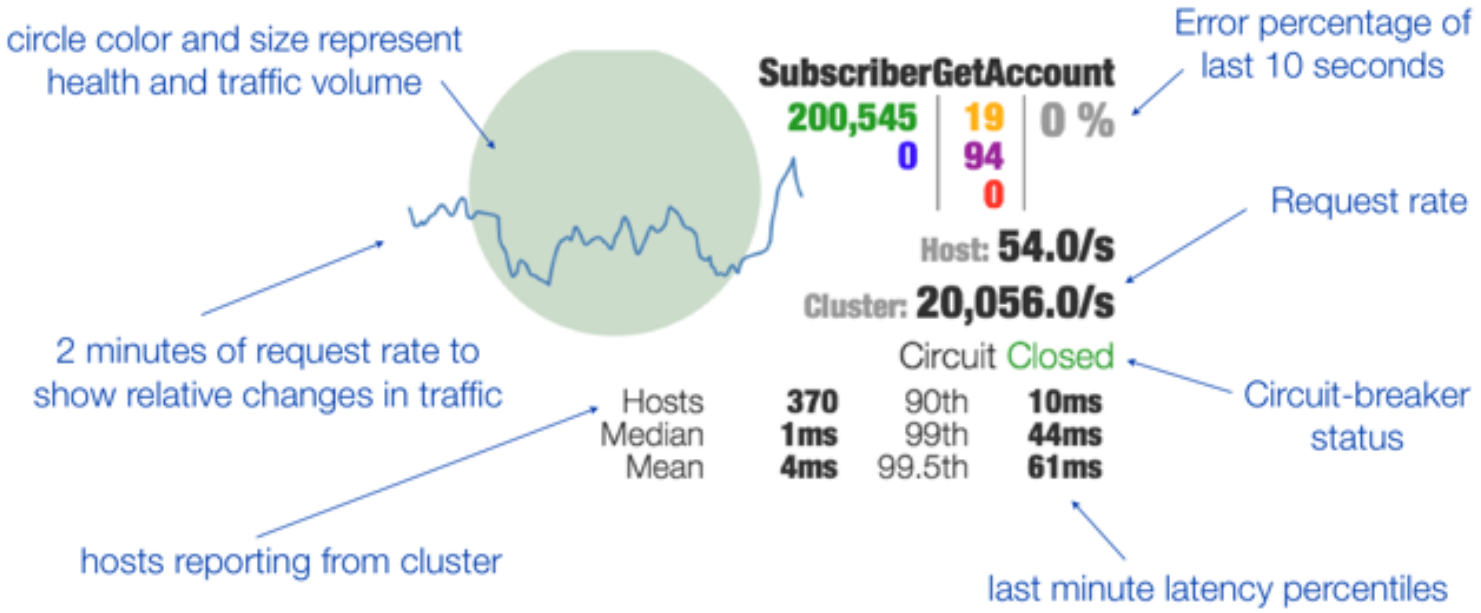
LAB: Scenario 4



Agenda

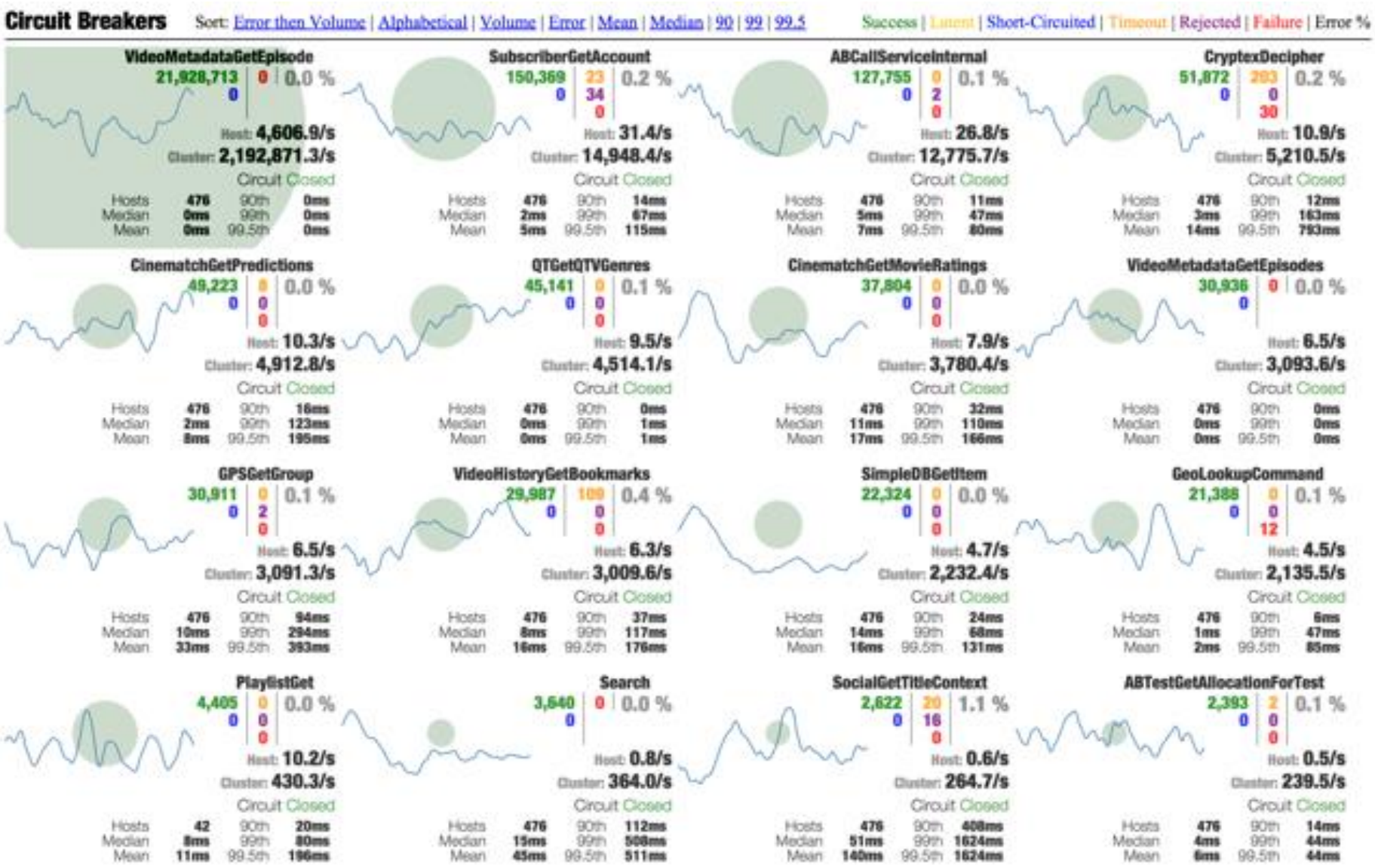
1. Here comes the Unicorns
2. Monolith v.s. Microservices
3. Services for the Microservices:
    a. Service Discovery
    b. Load Balacing
    c. Fault Tolerance
    d. **Edge Server and Data Aggregation**
4. The Death Star strikes back
5. Dockerize it
6. Centralized Logging and Monitoring
7. Running on Cloud Foundry
8. Continous Delivery, DevOps and NoOps
9. What's next?

Hystrix Dashboard

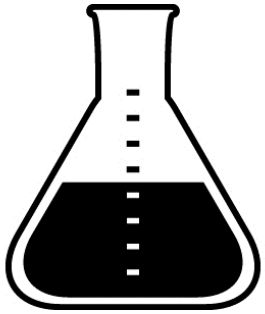


Rolling 10 second counters
with 1 second granularity

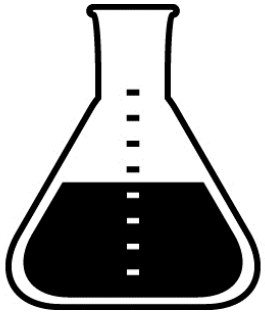
Successes	200,545	19	Thread timeouts
Short-circuited (rejected)	0	94	Thread-pool Rejections
		0	Failures/Exceptions


Circuit Breakers
Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5
Success | Latent | Short-Circuited | Timeout | Rejected | Failure | Error %
VideoMetadataGetEpisode
21,928,713 0 0.0 %
0
Host: 4,606.9/s
Cluster: 2,192,871.3/s
Circuit Closed
Hosts 476 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms
SubscriberGetAccount
150,369 23 0.2 %
0 34
0
Host: 31.4/s
Cluster: 14,948.4/s
Circuit Closed
Hosts 476 90th 14ms
Median 2ms 99th 67ms
Mean 5ms 99.5th 115ms
ABCallServiceInternal
127,755 0 0.1 %
0 2
0
Host: 26.8/s
Cluster: 12,775.7/s
Circuit Closed
Hosts 476 90th 11ms
Median 5ms 99th 47ms
Mean 7ms 99.5th 80ms
CryptexDecipher
51,872 203 0.2 %
0 0
30
Host: 10.9/s
Cluster: 5,210.5/s
Circuit Closed
Hosts 476 90th 12ms
Median 3ms 99th 163ms
Mean 14ms 99.5th 793ms
CinematchGetPredictions
49,223 0 0.0 %
0 0
0
Host: 10.3/s
Cluster: 4,912.8/s
Circuit Closed
Hosts 476 90th 16ms
Median 2ms 99th 123ms
Mean 8ms 99.5th 195ms
QTGetQTVGenres
45,141 0 0.1 %
0 0
0
Host: 9.5/s
Cluster: 4,514.1/s
Circuit Closed
Hosts 476 90th 0ms
Median 0ms 99th 1ms
Mean 0ms 99.5th 1ms
CinematchGetMovieRatings
37,804 0 0.0 %
0 0
0
Host: 7.9/s
Cluster: 3,780.4/s
Circuit Closed
Hosts 476 90th 32ms
Median 11ms 99th 110ms
Mean 17ms 99.5th 166ms
VideoMetadataGetEpisodes
30,936 0 0.0 %
0
Host: 6.5/s
Cluster: 3,093.6/s
Circuit Closed
Hosts 476 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms
GPSGetGroup
30,911 0 0.1 %
0 2
0
Host: 6.5/s
Cluster: 3,091.3/s
Circuit Closed
Hosts 476 90th 94ms
Median 10ms 99th 294ms
Mean 33ms 99.5th 393ms
VideoHistoryGetBookmarks
29,987 109 0.4 %
0 0
0
Host: 6.3/s
Cluster: 3,009.6/s
Circuit Closed
Hosts 476 90th 37ms
Median 8ms 99th 117ms
Mean 16ms 99.5th 176ms
SimpleDBGetItem
22,324 0 0.0 %
0 0
0
Host: 4.7/s
Cluster: 2,232.4/s
Circuit Closed
Hosts 476 90th 24ms
Median 14ms 99th 68ms
Mean 16ms 99.5th 131ms
GeoLookupCommand
21,388 0 0.1 %
0 0
12
Host: 4.5/s
Cluster: 2,135.5/s
Circuit Closed
Hosts 476 90th 6ms
Median 1ms 99th 47ms
Mean 2ms 99.5th 85ms
PlaylistGet
4,405 0 0.0 %
0 0
0
Host: 10.2/s
Cluster: 430.3/s
Circuit Closed
Hosts 42 90th 20ms
Median 8ms 99th 80ms
Mean 11ms 99.5th 196ms
Search
3,640 0 0.0 %
0
Host: 0.8/s
Cluster: 364.0/s
Circuit Closed
Hosts 476 90th 112ms
Median 15ms 99th 508ms
Mean 45ms 99.5th 511ms
SocialGetTitleContext
2,622 20 1.1 %
0 16
0
Host: 0.6/s
Cluster: 264.7/s
Circuit Closed
Hosts 476 90th 408ms
Median 51ms 99th 1624ms
Mean 140ms 99.5th 1624ms
ABTestGetAllocationForTest
2,393 2 0.1 %
0 0
0
Host: 0.5/s
Cluster: 239.5/s
Circuit Closed
Hosts 476 90th 14ms
Median 4ms 99th 44ms
Mean 6ms 99.5th 44ms

LAB: Scenario 5



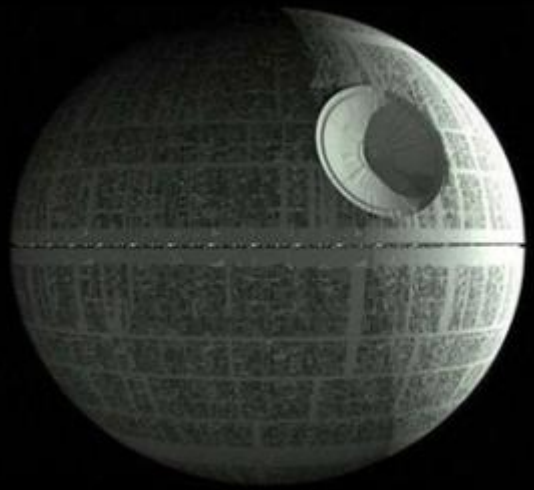
LAB: Scenario 6



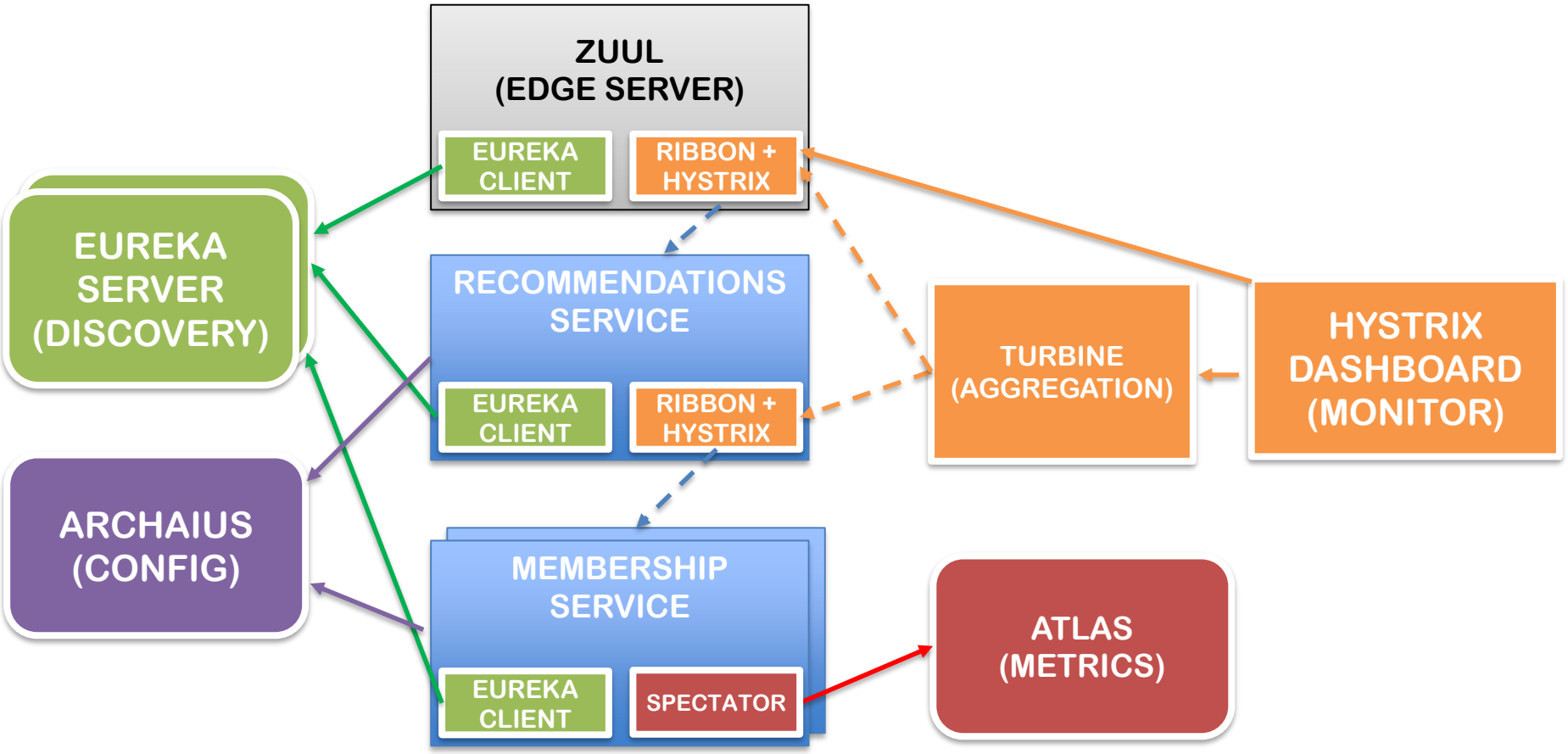
Agenda

1. Here comes the Unicorns
2. Monolith v.s. Microservices
3. Services for the Microservices:
    a. Service Discovery
    b. Load Balacing
    c. Fault Tolerance
    d. Edge Server and Data Aggregation
4. **The Death Star strikes back**
5. Dockerize it
6. Centralized Logging and Monitoring
7. Running on Cloud Foundry
8. Continous Delivery, DevOps and NoOps
9. What's next?

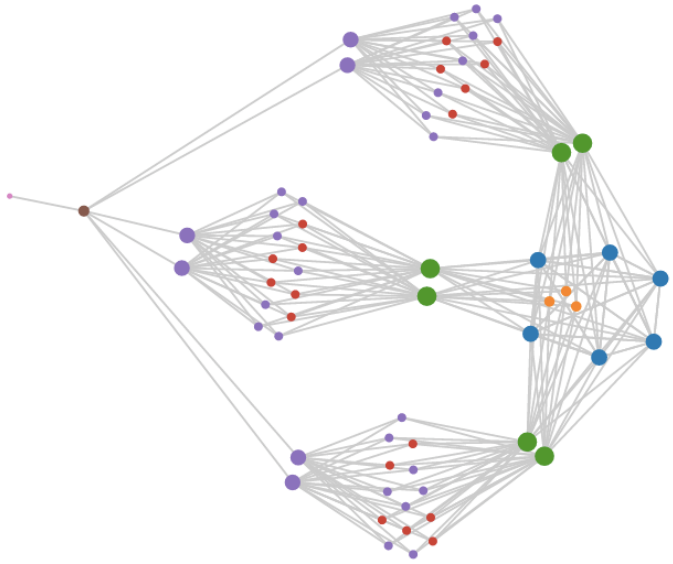
Monolith or Moon-olith?



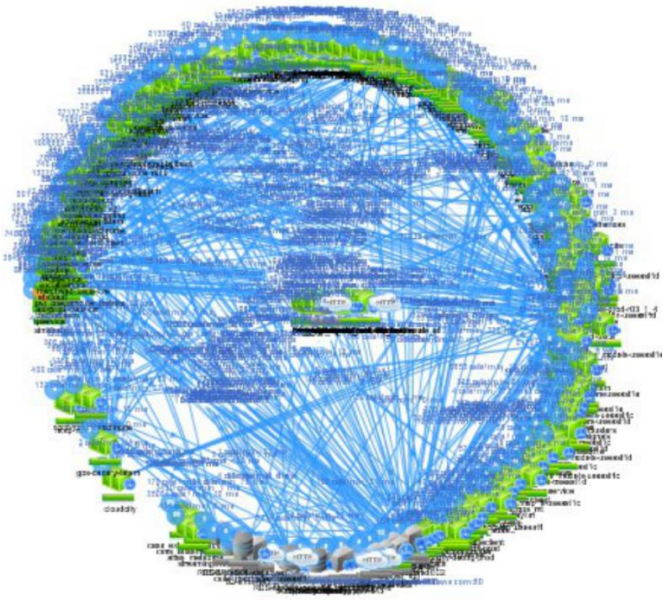
Main Netflix Components



Netflix Architecture
by Adrian Cockcroft



Simplified Architecture

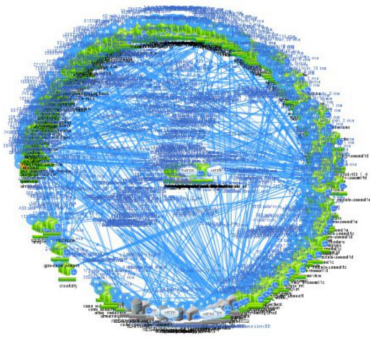


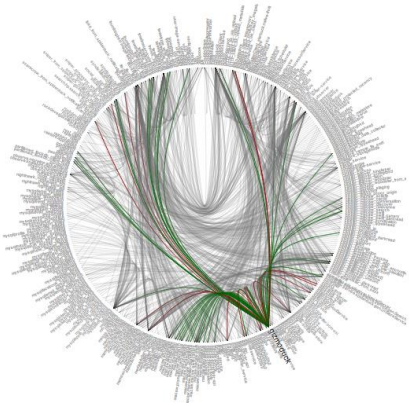
Actual Architecture

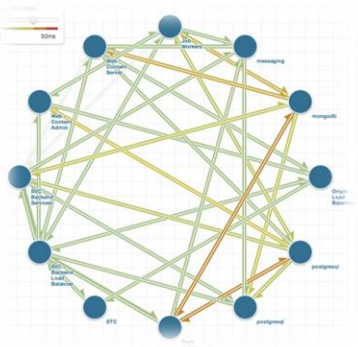
https://github.com/adrianco/spigo

Technology Comparison

	Configuraiton	Discovery	Routing	Observability
Netflix	Archaius	Eureka	Zuul, Ribbon	Hystrix
Twitter	Decider	Finagle, Zookeper	Finagle, Netty	Zipkin
Gilt	Decider	Finagle, Zookeper	Finagle, Akka	Zipkin



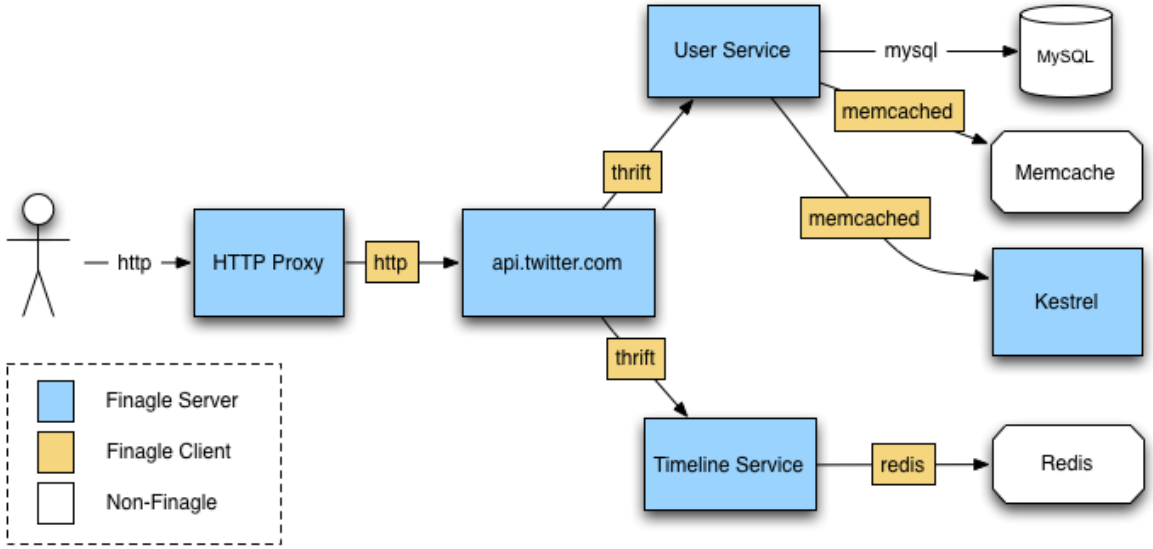




http://www.slideshare.net/adriancockcroft/monitorama-please-no-more

Finagle @Twitter

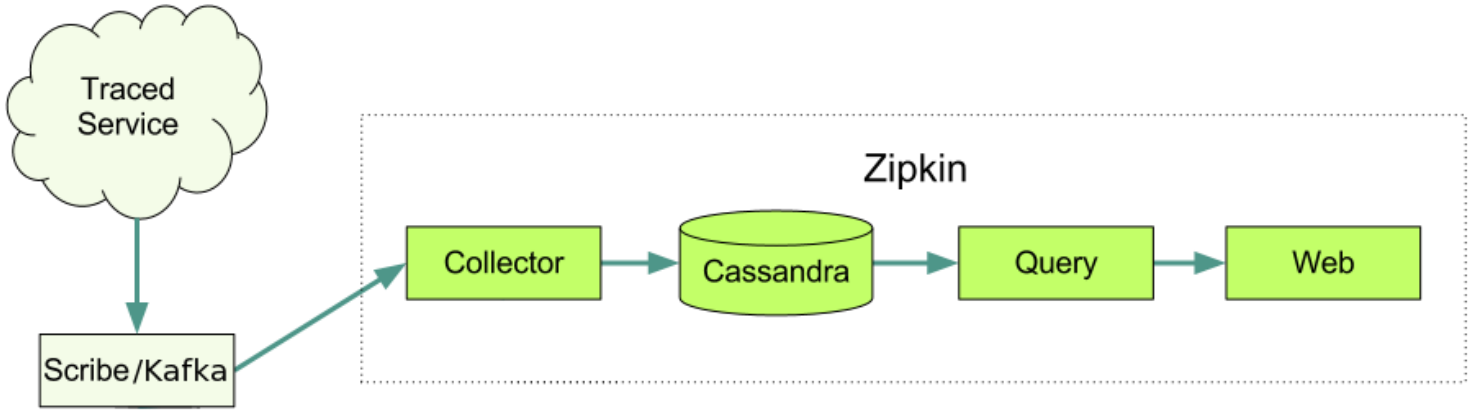
Finagle is an extensible RPC system for the JVM, used to construct high-concurrency servers.
Finagle implements uniform client and server APIs for several protocols, and is designed for high performance and concurrency.
Finagle is written in Scala, but provides both Scala and Java idiomatic APIs.



https://blog.twitter.com/2011/finagle-a-protocol-agnostic-rpc-system

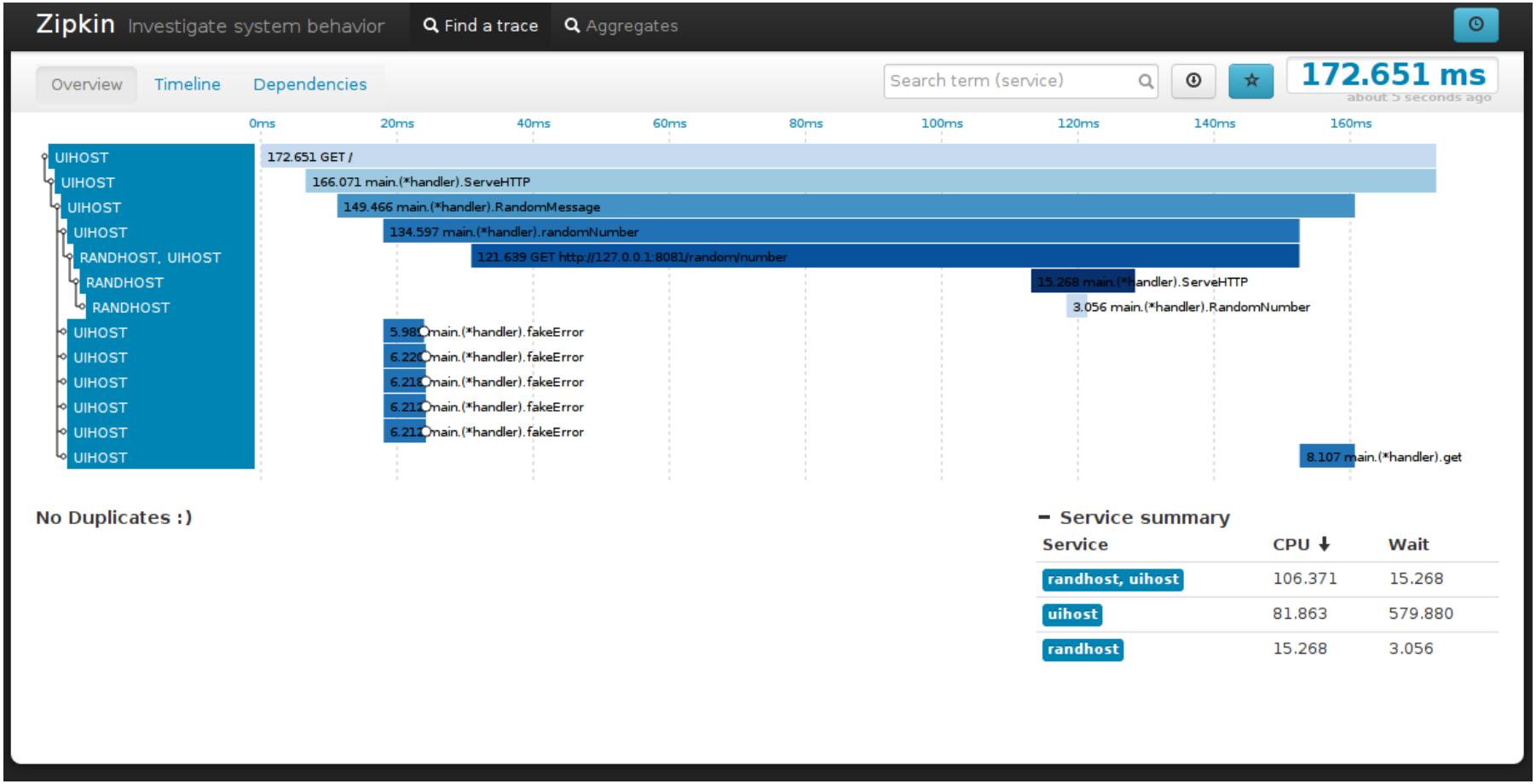
Zipkin @Twitter

Zipkin is a distributed tracing system. It helps gather timing data needed to troubleshoot latency problems in microservice architectures.



https://twitter.github.io/zipkin/Architecture.html
https://github.com/openzipkin/zipkin

Distributed Tracing with Zipkin

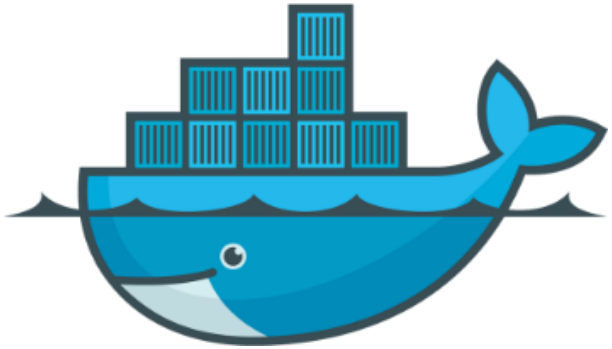


Agenda

1. Here comes the Unicorns
2. Monolith v.s. Microservices
3. Services for the Microservices:
    a. Service Discovery
    b. Load Balacing
    c. Fault Tolerance
    d. Edge Server and Data Aggregation
4. The Death Star strikes back
5. **Dockerize it**
6. Centralized Logging and Monitoring
7. Running on Cloud Foundry
8. Continous Delivery, DevOps and NoOps
9. What's next?

Introducing Docker

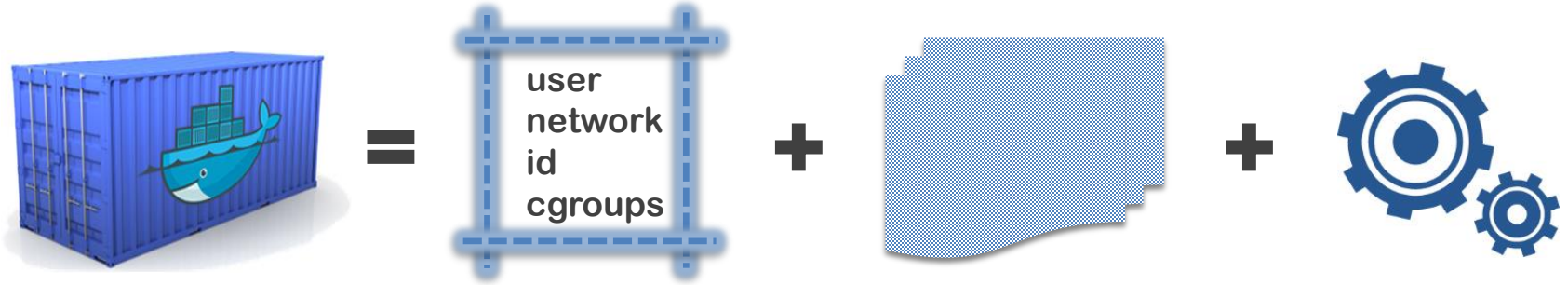
“Docker is an open platform for developers and system admins to build, ship, and run distributed applications.”



How does it work?

“Docker runs as an isolated process in userspace on the host operating system, sharing the kernel with other containers. Thus, it enjoys the resource isolation and allocation benefits of VMs but is much more portable and efficient. ”

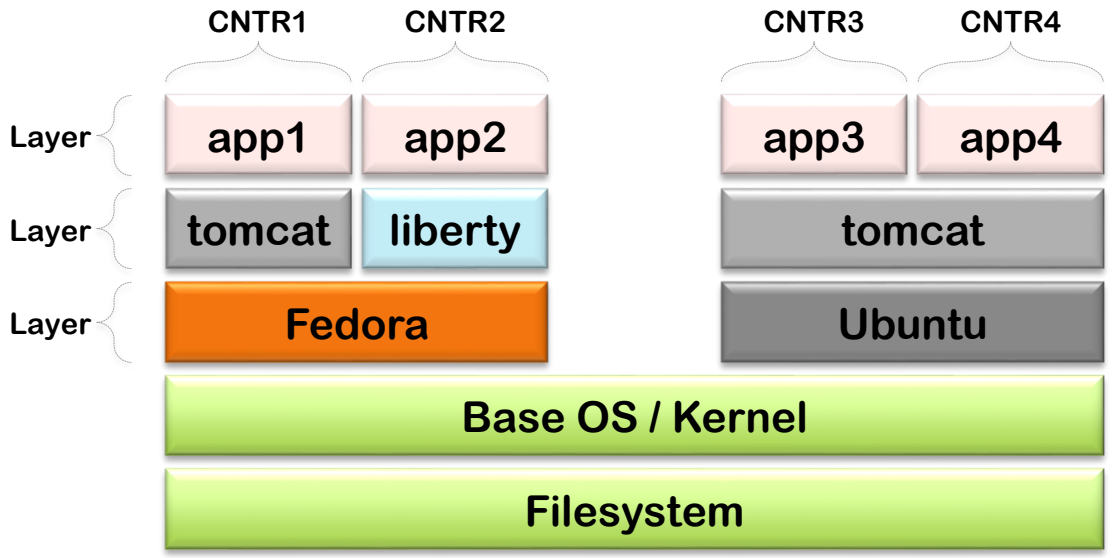
container = **isolation** + **content** + **processes**



"make me a isolated context" *"put this in it"* *"then run this"*

How is this different from Virtual Machines?

Docker is so lightweight because of the reusable layers in Docker images.

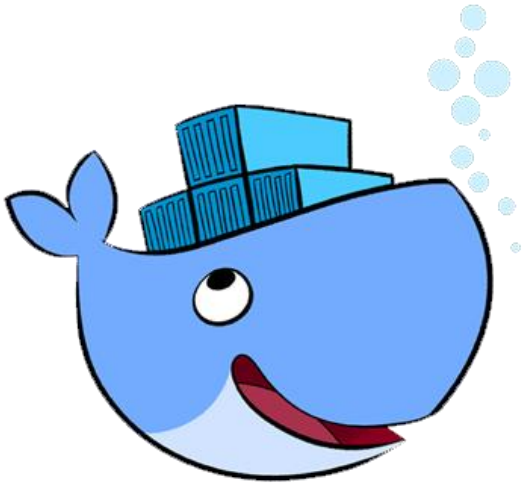


Layers allow for reuse, more containers per host and faster start/re-start time.

Read-only layers allow portability and efficiency.

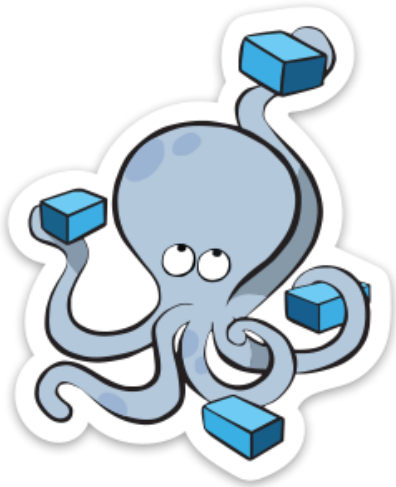
New files and edits are visible to current and above layers.

Docker uses **a copy-on-write (union) filesystem.**



At the core of the Docker platform is **Docker Engine**, a lightweight runtime that builds and runs your containers.

Docker Machine enables any host, whether a laptop, server, VM or remote cloud instance, to run Docker apps.

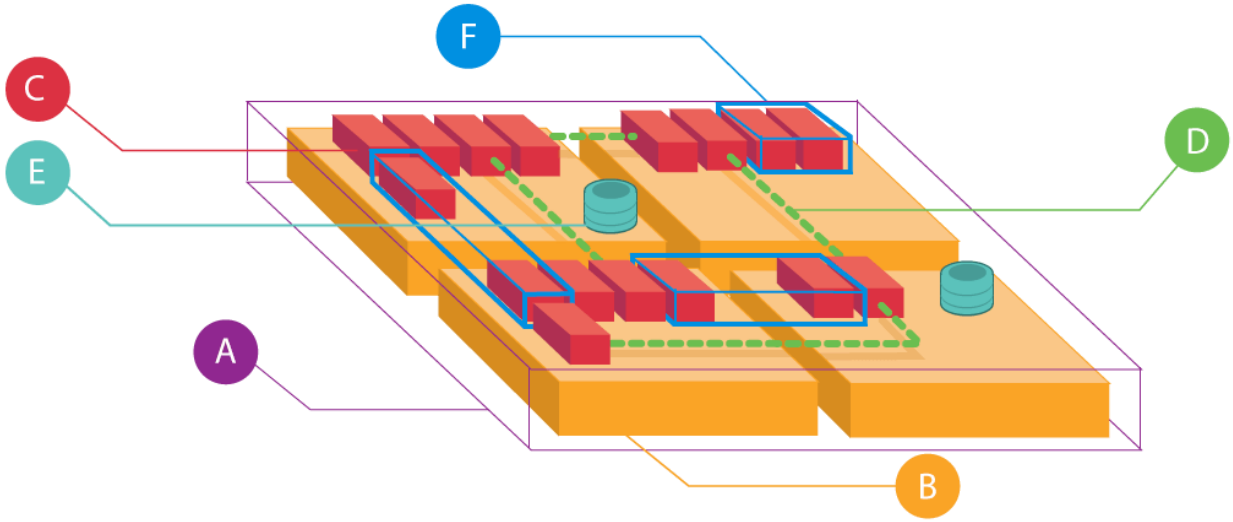


Docker Compose is a tool for defining and running multi-container applications with by using a single declarative file (i.e. docker-compose.yml).

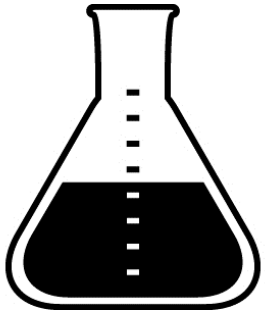
Managing Docker Clusters

Tutum is designed to ease the management of using Docker. Tutum offers a single platform to help with automating deployment, scaling, linking containers that run **on different cloud providers**, achieving redundancy and monitoring the deployment.

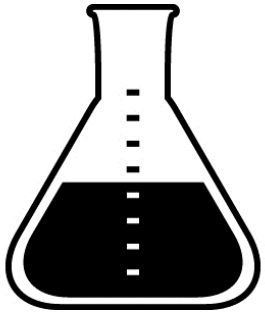
A = Node Clusters
B = Nodes
C = Containers
D = Links
E = Volumes
F = Services



LAB: Scenario 7



LAB: Scenario 8

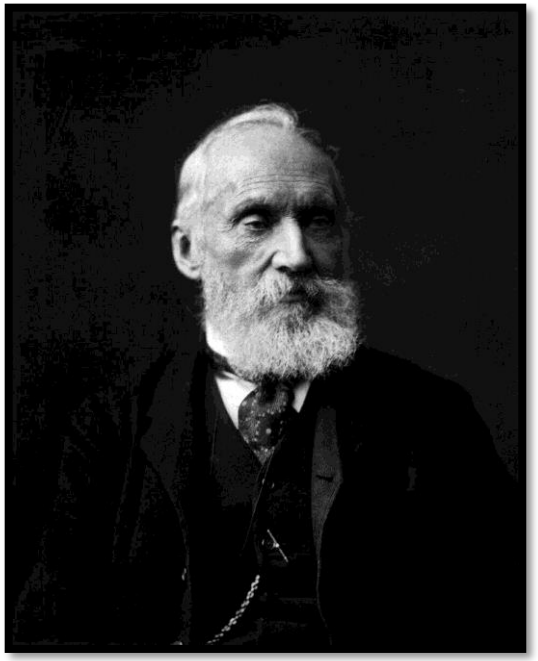


Agenda

1. Here comes the Unicorns
2. Monolith v.s. Microservices
3. Services for the Microservices:
    a. Service Discovery
    b. Load Balacing
    c. Fault Tolerance
    d. Edge Server and Data Aggregation
4. The Death Star strikes back
5. Dockerize it
6. **Centralized Logging and Monitoring**
7. Running on Cloud Foundry
8. Continous Delivery, DevOps and NoOps
9. What's next?

“If you can’t measure it, you can’t improve it.”

Lord Kelvin



Centralized logging with the **ELK Stack**

logstash

Logstash is a flexible, open source data collection, enrichment, and transportation pipeline.

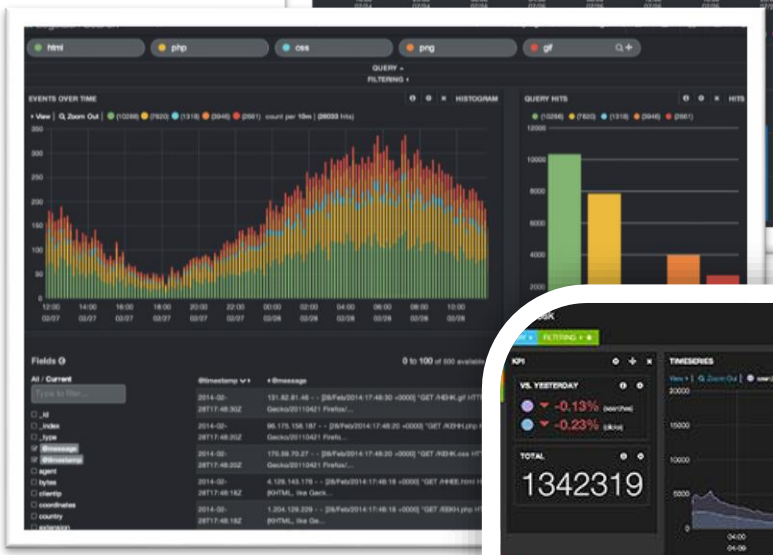
elasticsearch.

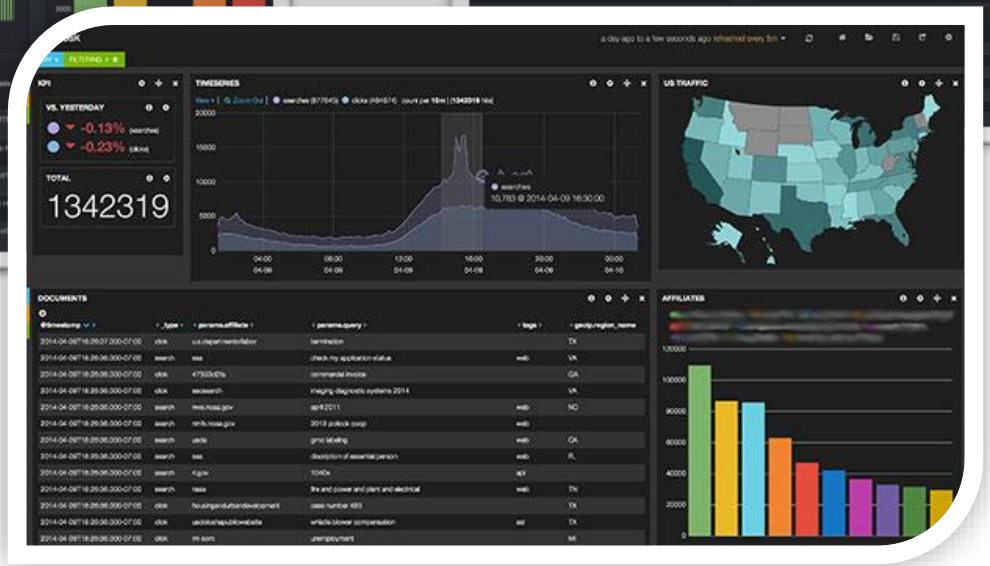
Elasticsearch is a distributed, open source search and analytics engine.

Kibana

Kibana is an open source data visualization platform to interact with your data.

kibana

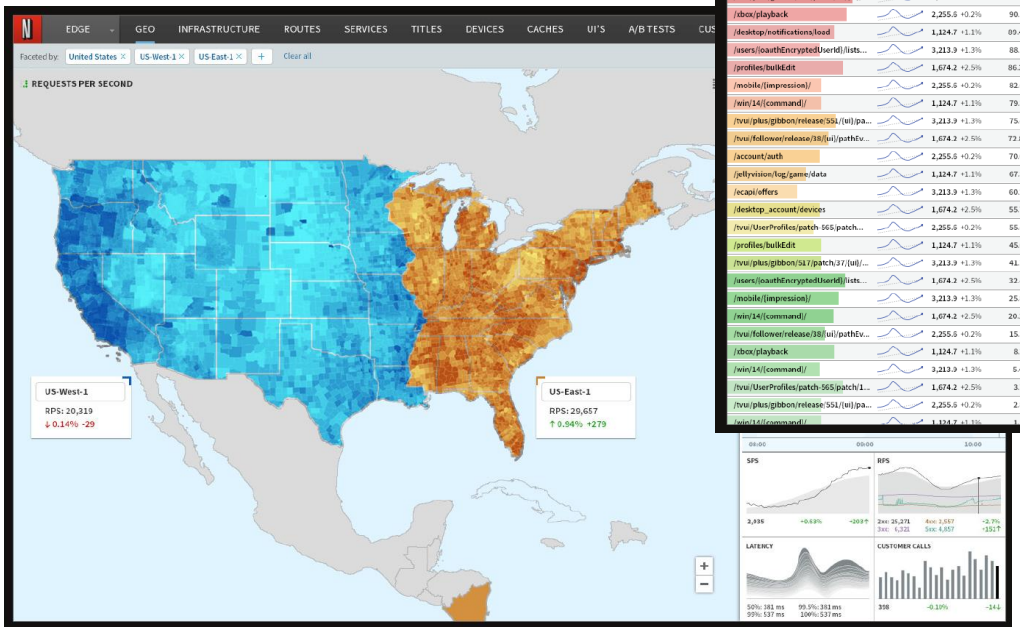


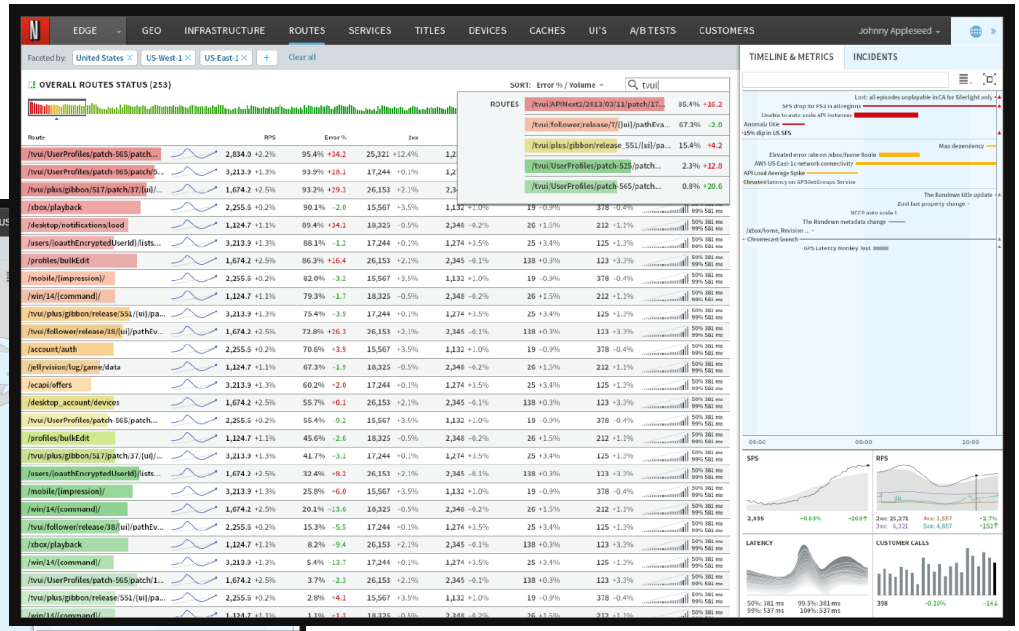


SOME MORE EXAMPLES

A Picture is Worth a Thousand Words

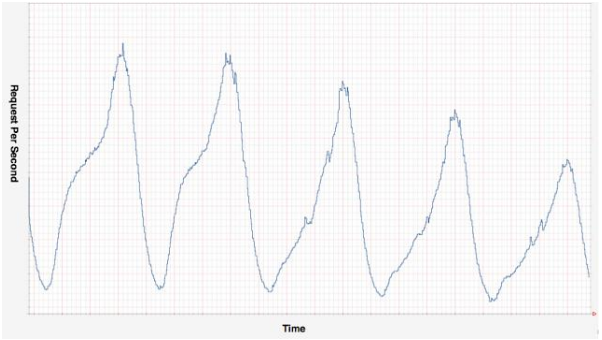
Good visualization helps to communicate and deliver insights effectively.

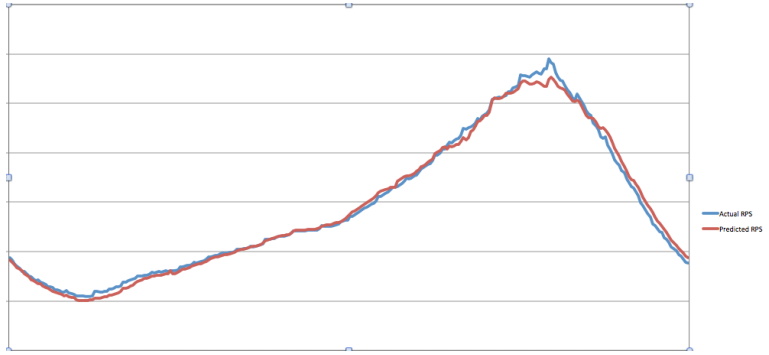


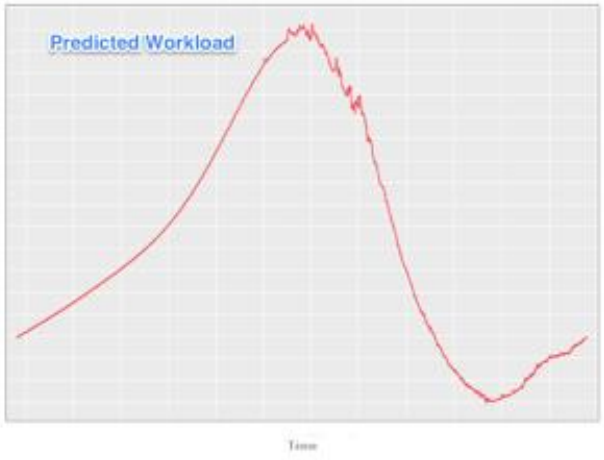


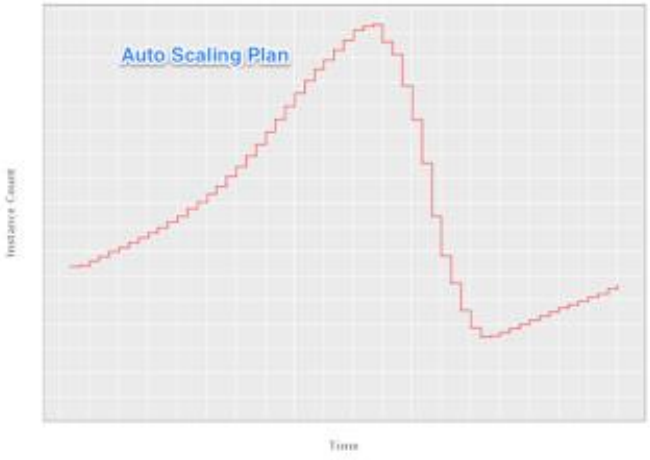
http://techblog.netflix.com/2014/01/improving-netflixs-operational.html

Predictive Auto-Scaling at Netflix



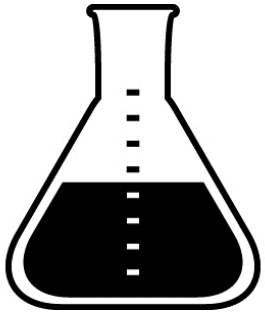






http://techblog.netflix.com/2013/11/scryer-netflixs-predictive-auto-scaling.html

LAB: Scenario 9

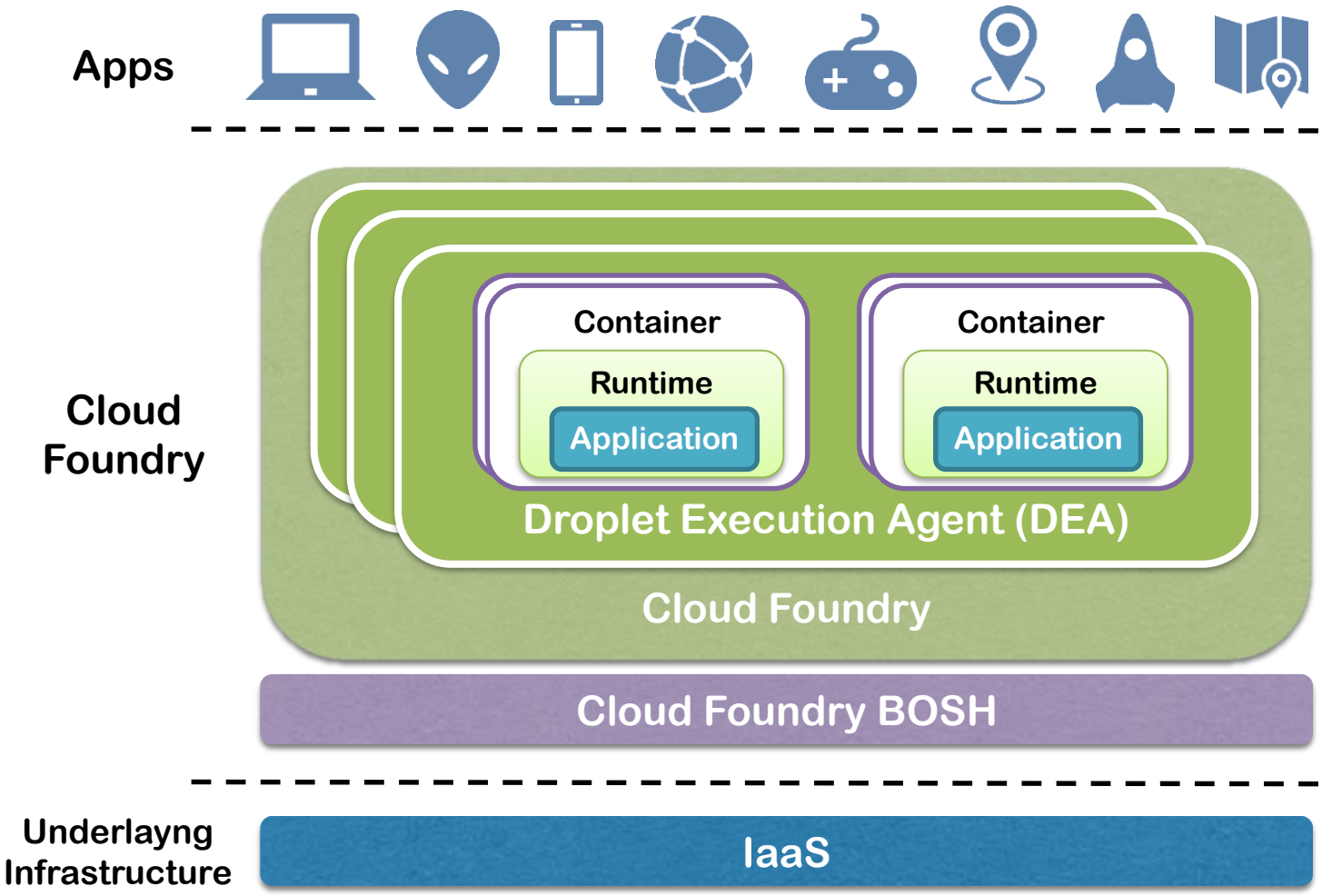


Agenda

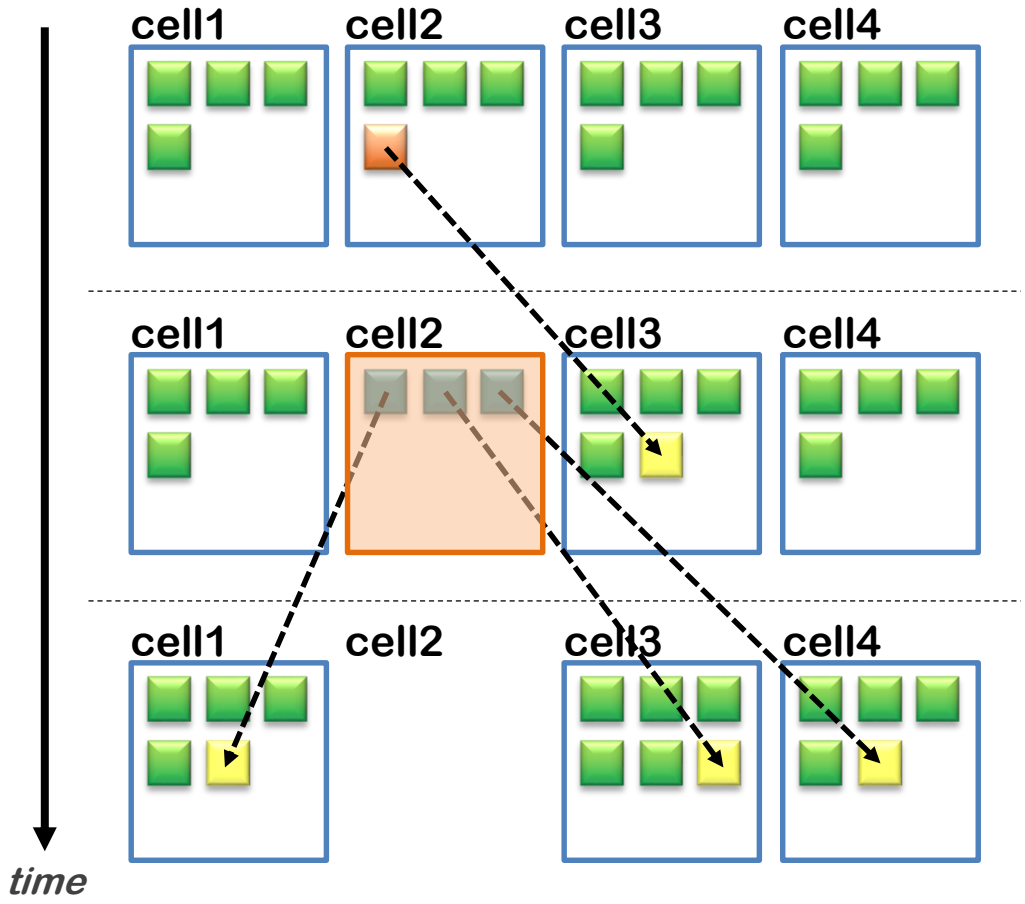
1. Here comes the Unicorns
2. Monolith v.s. Microservices
3. Services for the Microservices:
    a. Service Discovery
    b. Load Balacing
    c. Fault Tolerance
    d. Edge Server and Data Aggregation
4. The Death Star strikes back
5. Dockerize it
6. Centralized Logging and Monitoring
7. **Running on Cloud Foundry**
8. Continous Delivery, DevOps and NoOps
9. What's next?

Introducing Cloud Foundry

Cloud Foundry is an open source project **platform-as-a-service**, making it faster and easier to **build**, **test**, **deploy**, and **scale** applications


Apps
Cloud Foundry
Container
Runtime
Application
Container
Runtime
Application
Droplet Execution Agent (DEA)
Cloud Foundry
Cloud Foundry BOSH
Underlayng Infrastructure
IaaS

Container orchestration in Cloud Foundry

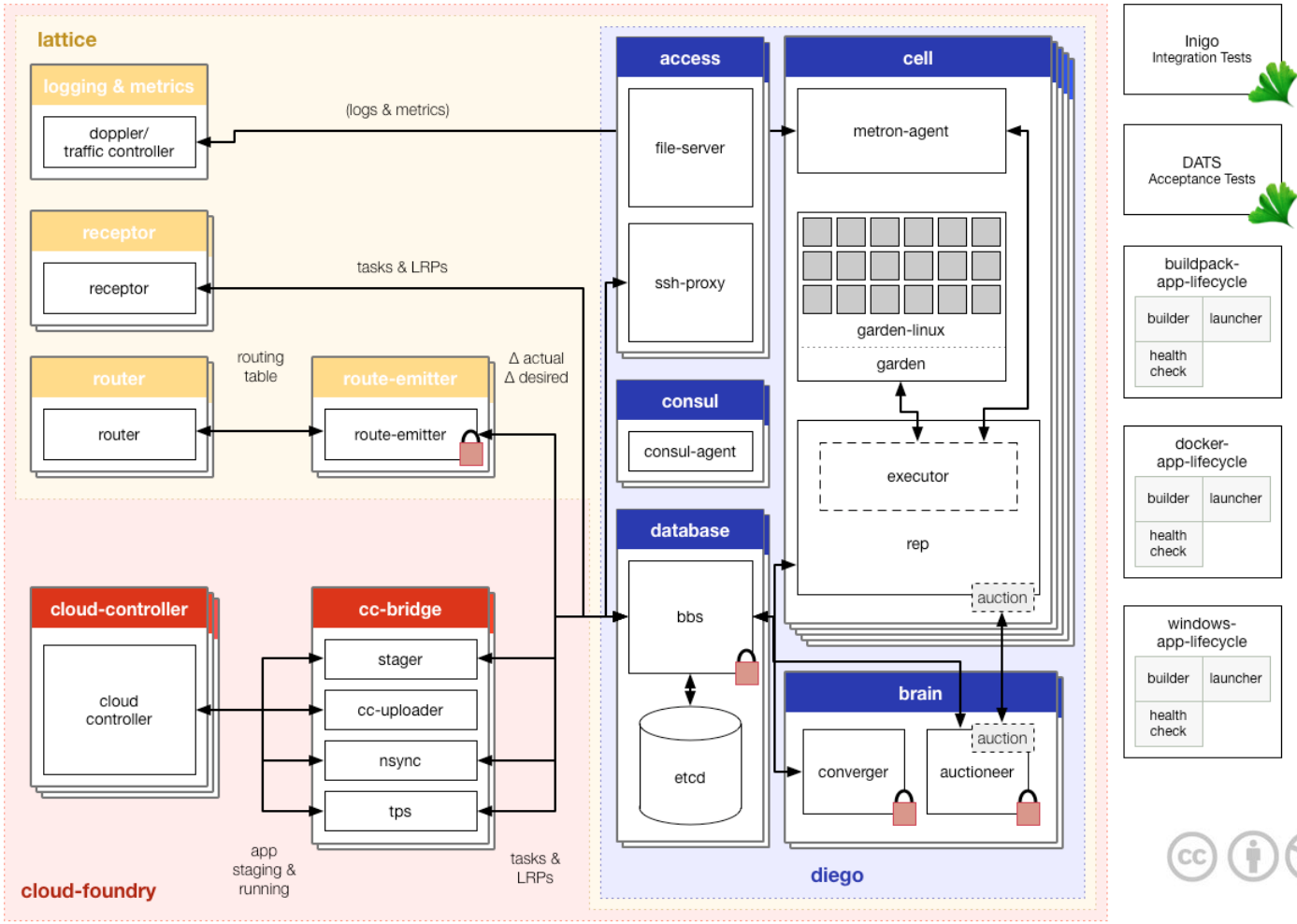


Cloud Foundry distributes the containers among the DEA cells

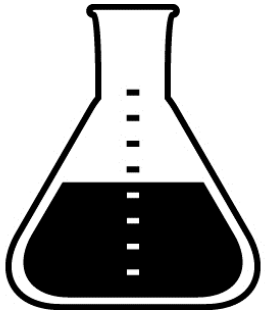
When a container crashes, it's automatically restarted

When a cell crashes, it's workload is distributed among the other cells

Lattice: a minimal Cloud Foundry



LAB: Scenario 10

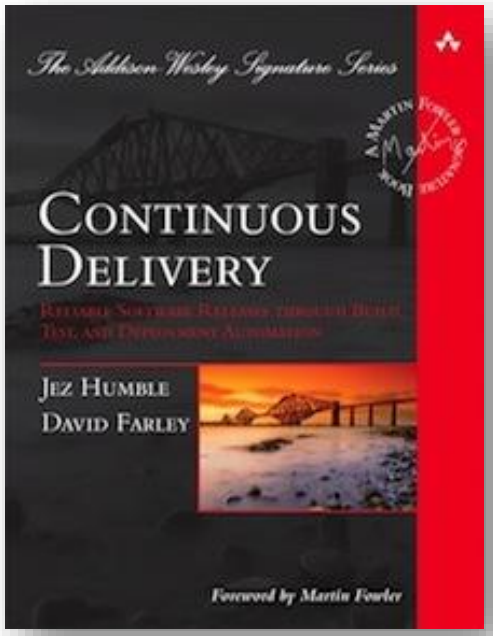


Agenda

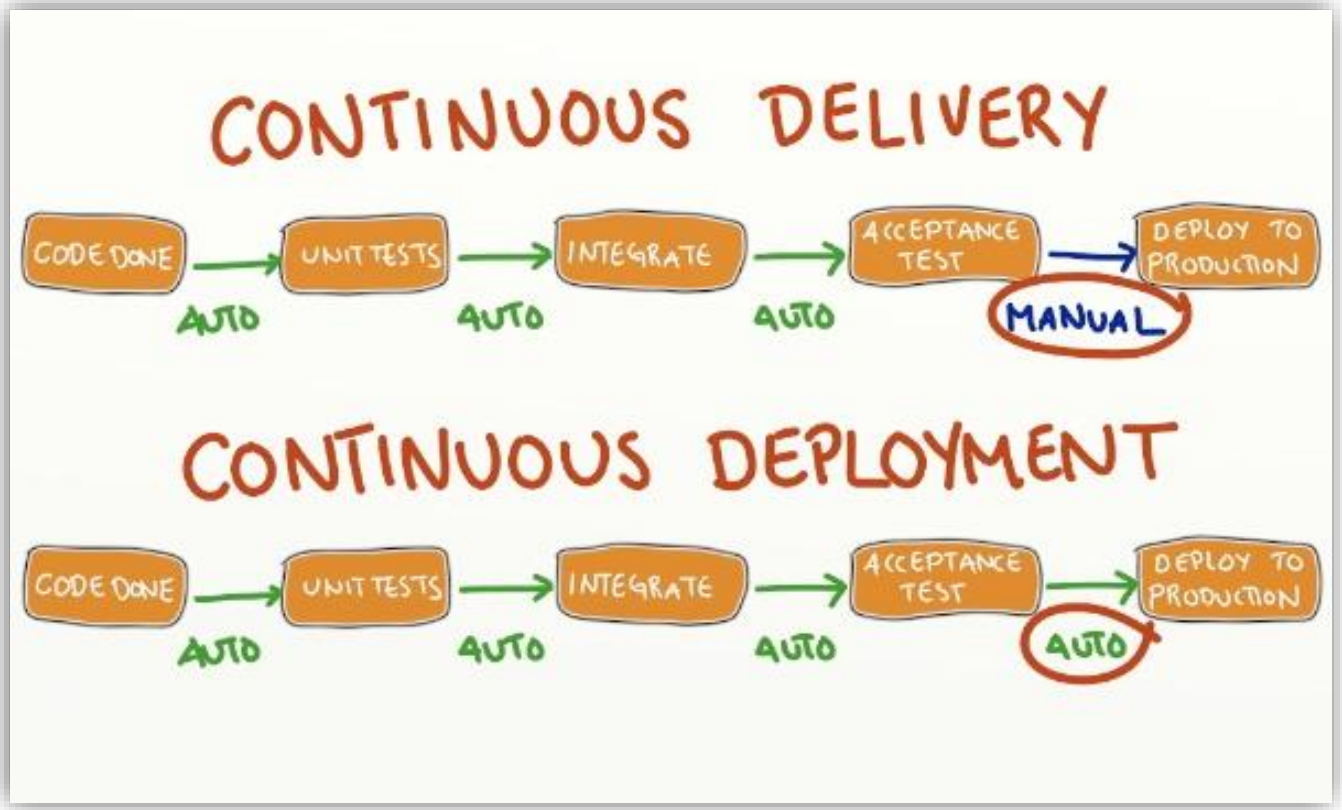
1. Here comes the Unicorns
2. Monolith v.s. Microservices
3. Services for the Microservices:
    a. Service Discovery
    b. Load Balacing
    c. Fault Tolerance
    d. Edge Server and Data Aggregation
4. The Death Star strikes back
5. Dockerize it
6. Centralized Logging and Monitoring
7. Running on Cloud Foundry
8. **Continous Delivery, DevOps and NoOps**
9. What's next?

“In software, when something is painful, the way to reduce the pain is to do it more frequently, not less.”

David Farley



Continuous Delivery vs Continuous Deployment



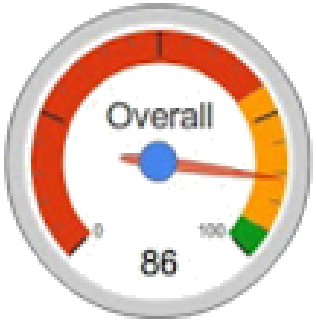
http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment

Deployment Techniques

minimize downtime

minimize risks

increase feedback

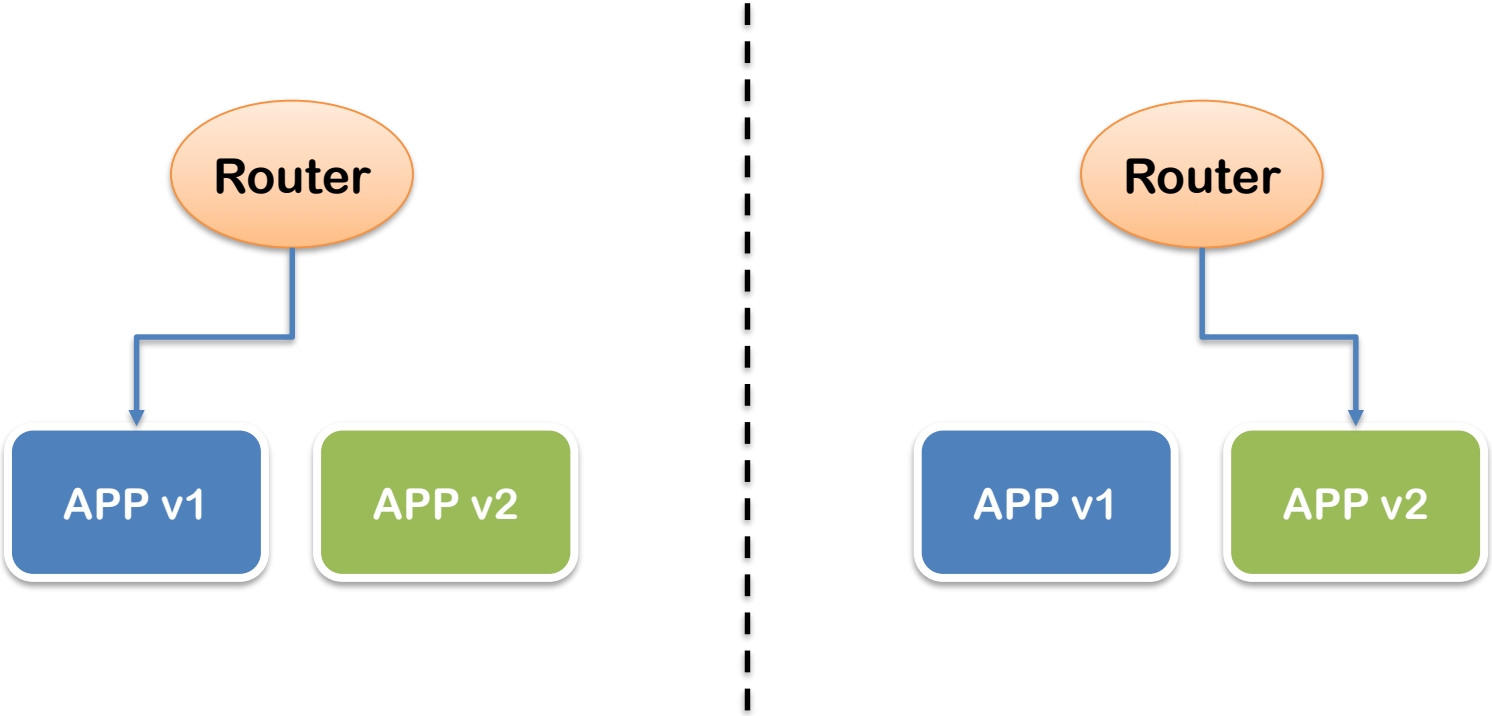


Canary Score

AMI Name: ami-4219582b Date: Tue Aug 13 23:31:27 UTC 2013 -1 Hours

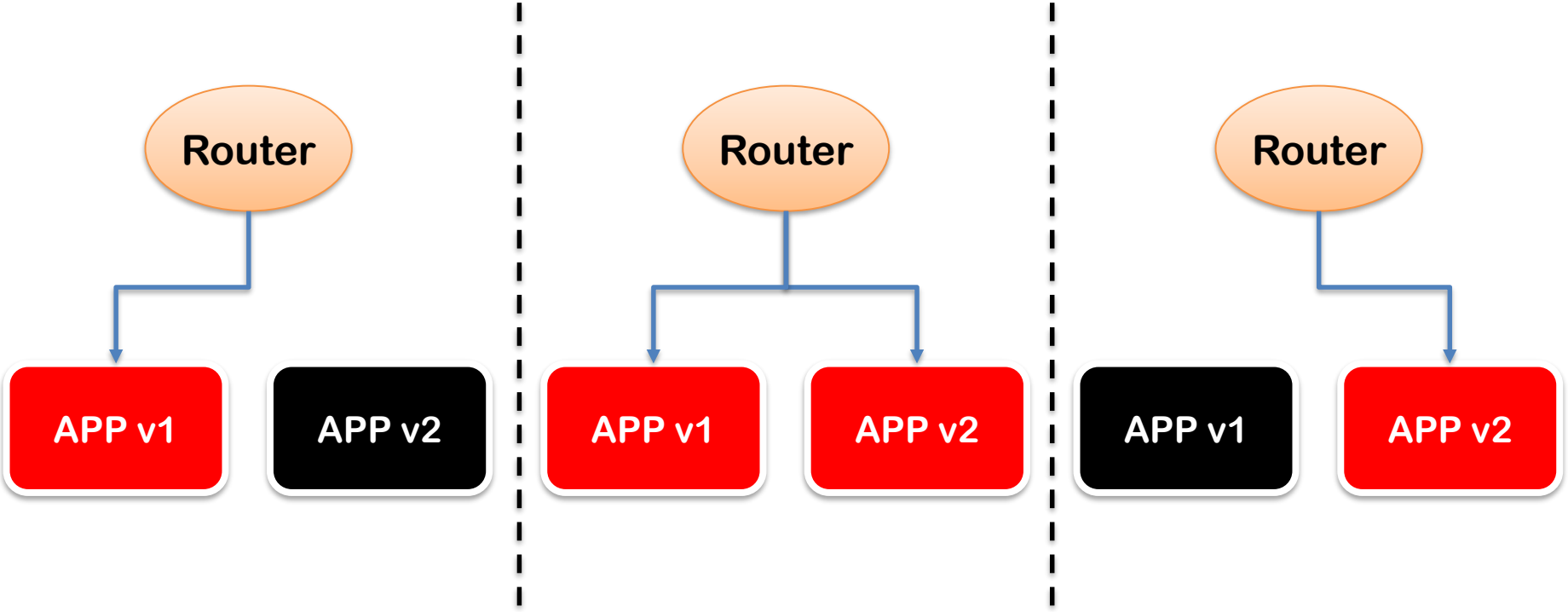
http://techblog.netflix.com/2013/08/deploying-netflix-api.html

Blue-Green Deployment



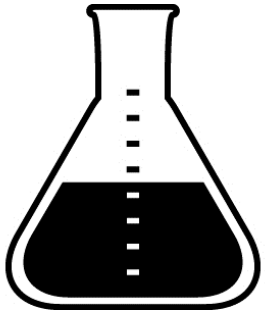
http://martinfowler.com/bliki/BlueGreenDeployment.html

Red-Black Deployment & Canary Deployment



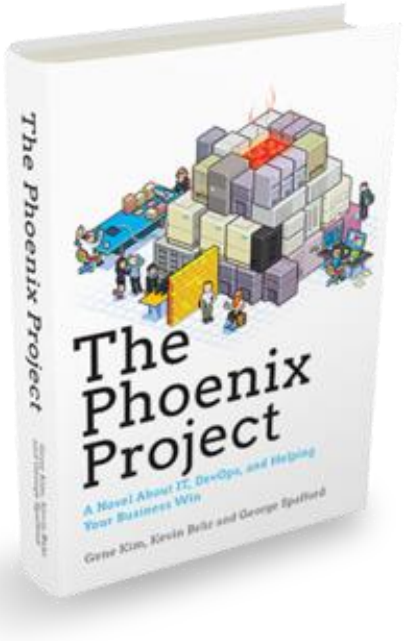
http://techblog.netflix.com/2013/08/deploying-netflix-api.html
http://martinfowler.com/bliki/CanaryRelease.html

LAB: Scenario 11

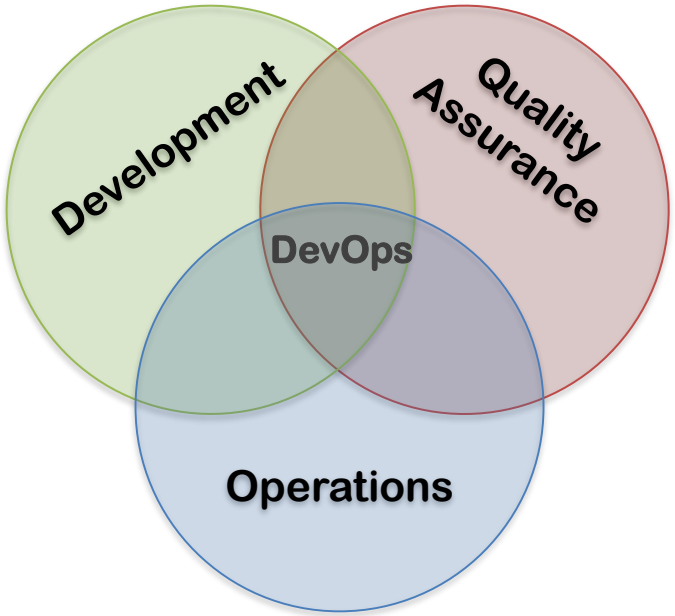


“Until code is in production, no value is actually being generated”

Gene Kim



from DevOps to NoOps

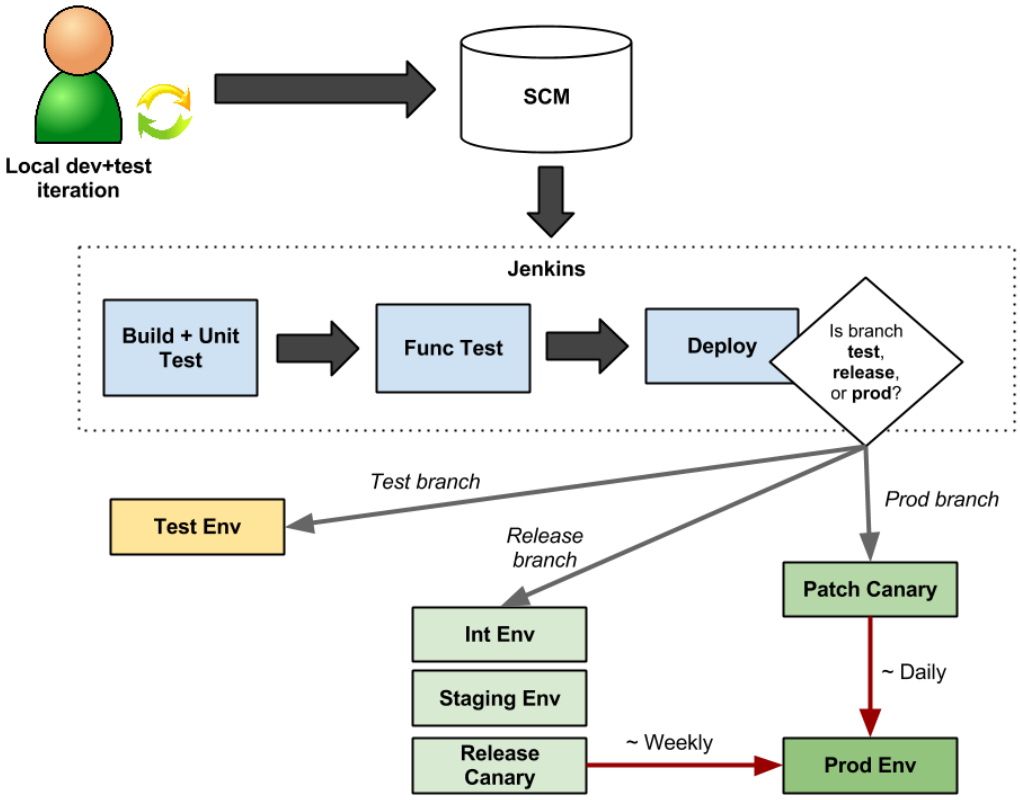


http://blogs.forrester.com/mike_gualtieri/11-02-07-i_dont_want_devops_i_want_noops

“You Build It, You Run It”

Werner Vogels (Amazon)

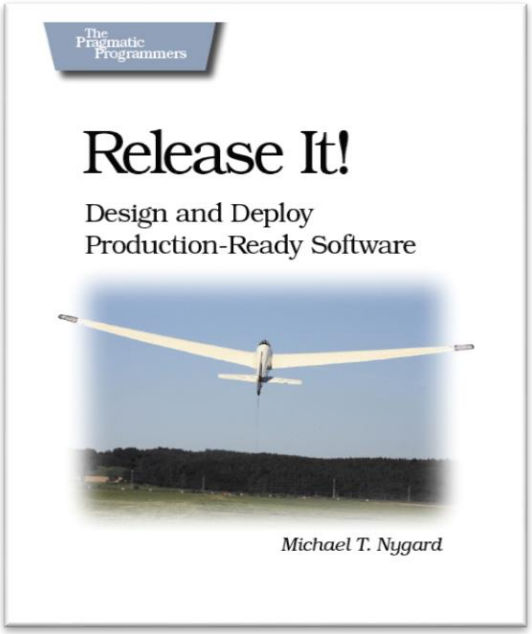
Continous Delivery at Neflix



http://techblog.netflix.com/2013/08/deploying-netflix-api.html

“Run longevity tests. It’s the only way to catch longevity bugs.”

Michael T. Nygard



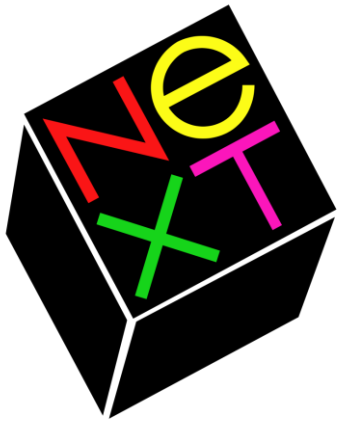
Failures happen, and they inevitably happen when least desired.

Simian Army consists of services (Monkeys) in the cloud for **generating various kinds of failures**, detecting abnormal conditions, and testing our ability to survive them. The goal is to keep our cloud safe, secure, and highly available

http://techblog.netflix.com/2011/07/netflix-simian-army.html

Agenda

1. Here comes the Unicorns
2. Monolith v.s. Microservices
3. Services for the Microservices:
    a. Service Discovery
    b. Load Balacing
    c. Fault Tolerance
    d. Edge Server and Data Aggregation
4. The Death Star strikes back
5. Dockerize it
6. Centralized Logging and Monitoring
7. Running on Cloud Foundry
8. Continous Delivery, DevOps and NoOps
9. **What's next?**

What's  ?

Very good tutorials

Blog Series - Building Microservices

http://callistaenterprise.se/blogg/teknik/2015/05/20/blog-series-building-microservices

Microservice Registration and Discovery with Spring Cloud and Netflix's Eureka

https://spring.io/blog/2015/01/20/microservice-registration-and-discovery-with-spring-cloud-and-netflix-s-eureka

Using Microservices To Build Cloud Native Applications

http://ryanjbaxter.com/2015/07/15/using-microservices-to-build-cloud-native-applications-part-1

Log Management for Spring Boot Applications with Logstash, Elasticsearch and Kibana

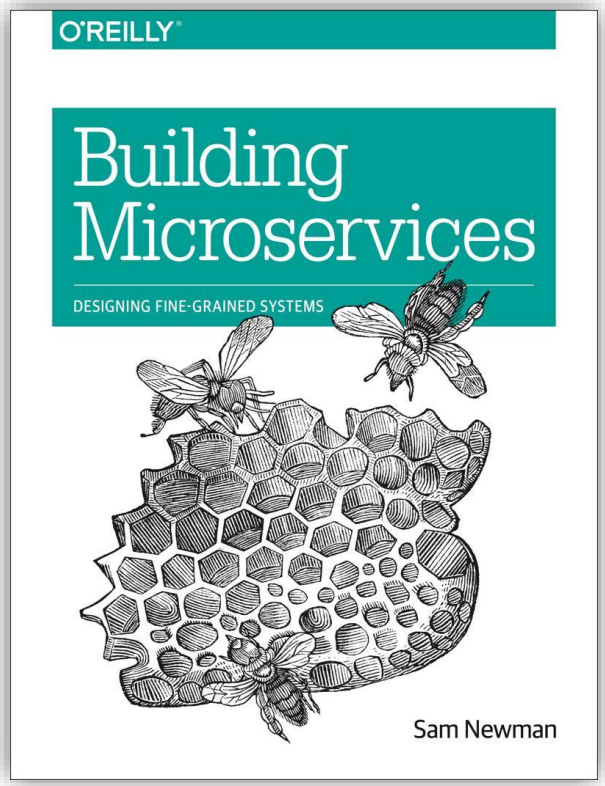
https://blog.codecentric.de/en/2014/10/log-management-spring-boot-applications-logstash-elastichsearch-kibana

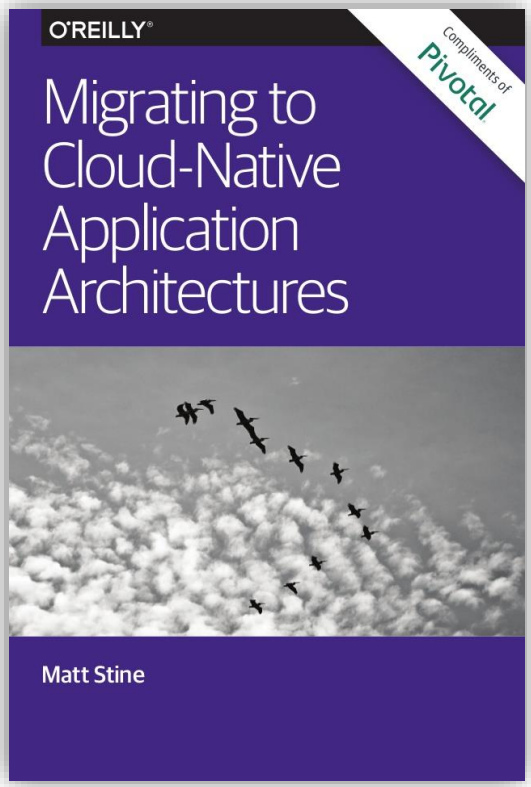
Videos

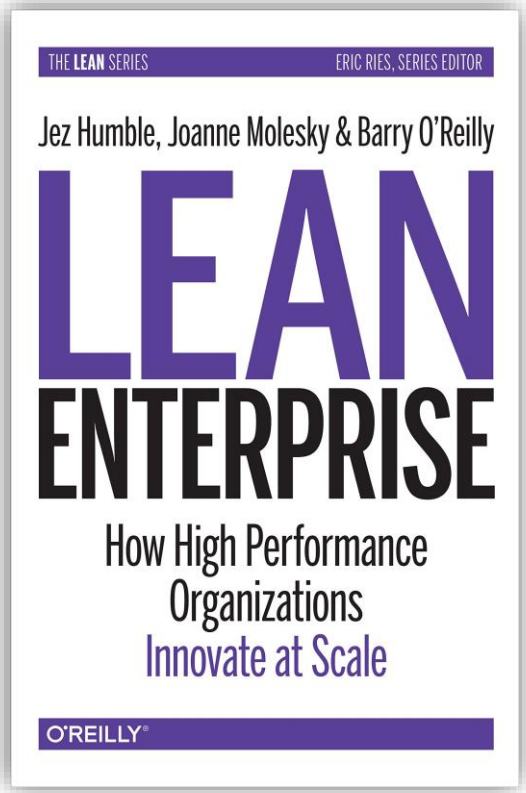
Monitoring and Simulating Microservices: Adrian Cockcroft (Battery Ventures)
https://www.youtube.com/watch?v=2YqiSPjIBqQ

Spring Cloud at Netflix: Jon Schneider and Taylor Wicksell (Netflix)
https://www.youtube.com/watch?v=6wvVANQ6EJ8

More Books


O'REILLY
Building Microservices
DESIGNING FINE-GRAINED SYSTEMS
Sam Newman


O'REILLY
Compliments of Pivotal
Migrating to Cloud-Native Application Architectures
Matt Stine


THE LEAN SERIES
ERIC RIES, SERIES EDITOR
Jez Humble, Joanne Molesky & Barry O'Reilly
LEAN ENTERPRISE
How High Performance Organizations Innovate at Scale
O'REILLY

“Beware the distributed monolith.”

James Lewis
(ThoughWorks)