Microservices Workshop
Why, What and How To Get There

Adrian Cockcroft @adrianco
Technology Fellow - Battery Ventures
July 2015
Agenda

Workshop vs. Presentation
Introductions
Faster Development
Microservice Architectures
Migration and Simulation
Workshop vs. Presentation

Questions at any time
Interactive discussions
Share your experiences
Everyone’s voice should be heard
What does @adrianco do?

- Presentations at Conferences
- Presentations at Companies
- Program Committee for Conferences
- Maintain Relationship with Cloud Vendors
- Technology Due Diligence on Deals
- Technical Advice for Portfolio Companies
- Networking with Interesting People
- Tinkering with Technologies
- Maintain Relationship with Cloud Vendors
Why am I here?

By Simon Wardley http://enterpriseitadoption.com/
**Why am I here?**


2009

- Ignore
- Ignore
- Ignore
- “No”
- “No”
- “No”
- “Yes”
- “Oh No”
- “Oh %*&!”

Adoption vs. Time

- Netflix
- Rest of World
- Enterprise IT
Why am I here?

2009

Adoption

Ignore Ignore Ignore "No" "No" "No" dammit "Oh No" "Oh %*&!"

Rest of World

Enterprise IT

By Simon Wardley http://enterpriseitadoption.com/
Why am I here?

@adrianco’s job at the intersection of cloud and Enterprise IT, looking for disruption and opportunities.

By Simon Wardley http://enterpriseitadoption.com/
Why am I here?

@adrianco’s job at the intersection of cloud and Enterprise IT, looking for disruption and opportunities.

Example: Docker wasn’t on anyone’s roadmap for 2014. It’s on everyone’s roadmap for 2015.
Typical reactions to my Netflix talks…
Typical reactions to my Netflix talks…

“You guys are crazy! Can’t believe it”
– 2009
Typical reactions to my Netflix talks…

“You guys are crazy! Can’t believe it”  
– 2009

“What Netflix is doing won’t work”  
– 2010
Typical reactions to my Netflix talks...

“You guys are crazy! Can’t believe it” – 2009

“What Netflix is doing won’t work” – 2010

It only works for ‘Unicorns’ like Netflix” – 2011
Typical reactions to my Netflix talks...

“You guys are crazy! Can’t believe it” – 2009

“What Netflix is doing won’t work” – 2010

“It only works for ‘Unicorns’ like Netflix” – 2011

“We’d like to do that but can’t” – 2012
Typical reactions to my Netflix talks…

“You guys are crazy! Can’t believe it” – 2009

“What Netflix is doing won’t work” – 2010

It only works for ‘Unicorns’ like Netflix” – 2011

“We’d like to do that but can’t” – 2012

“We’re on our way using Netflix OSS code” – 2013
What I learned from my time at Netflix

• Speed wins in the marketplace
What I learned from my time at Netflix

• *Speed wins in the marketplace*
• *Remove friction from product development*
What I learned from my time at Netflix

• Speed wins in the marketplace
• Remove friction from product development
• High trust, low process, no hand-offs between teams
What I learned from my time at Netflix

• *Speed wins in the marketplace*
• *Remove friction from product development*
• *High trust, low process, no hand-offs between teams*
• *Freedom and responsibility culture*
What I learned from my time at Netflix

• Speed wins in the marketplace
• Remove friction from product development
• High trust, low process, no hand-offs between teams
• Freedom and responsibility culture
• Don’t do your own undifferentiated heavy lifting
What I learned from my time at Netflix

- Speed wins in the marketplace
- Remove friction from product development
- High trust, low process, no hand-offs between teams
- Freedom and responsibility culture
- Don’t do your own undifferentiated heavy lifting
- Use simple patterns automated by tooling
What I learned from my time at Netflix

• *Speed wins in the marketplace*
• *Remove friction from product development*
• *High trust, low process, no hand-offs between teams*
• *Freedom and responsibility culture*
• *Don’t do your own undifferentiated heavy lifting*
• *Use simple patterns automated by tooling*
• *Self service cloud makes impossible things instant*
“You build it, you run it.”

Werner Vogels 2006
2014 was the year that Enterprises finally embraced cloud and DevOps.
2014 was the year that Enterprises finally embraced cloud and DevOps.
2014 was the year that Enterprises finally embraced cloud and DevOps.
2014 was the year that Enterprises finally embraced cloud and DevOps.
Key Goals of the CIO?
Align IT with the business
Develop products faster
Try not to get breached
Insecure applications hidden behind firewalls make you feel safe until the breach happens...

http://peanuts.wikia.com/wiki/Linus'_security_blanket
What needs to change?
Developer responsibilities: Faster, cheaper, safer
Value Chain Mapping

Related tools and training http://www.wardleymaps.com/
Value Chain Mapping

Your unique product - Agile

Related tools and training http://www.wardleymaps.com/
Value Chain Mapping

Your unique product - Agile

Best of breed as a Service - Lean

Related tools and training http://www.wardleymaps.com/
Value Chain Mapping

Your unique product - Agile

Best of breed as a Service - Lean

Undifferentiated utility suppliers - 6sigma

Related tools and training http://www.wardleymaps.com/
“It isn't what we don't know that gives us trouble, it's what we know that ain't so.”

Will Rogers
Assumptions
Optimizations
Assumption: Process prevents problems
Organizations build up slow complex “Scar tissue” processes
"This is the IT swamp draining manual for anyone who is neck deep in alligators.”
Product Development Processes
**Waterfall Product Development**

- **Business Need**
  - Documents
  - Weeks

- **Approval Process**
  - Meetings
  - Weeks

- **Hardware Purchase**
  - Negotiations
  - Weeks

- **Software Development**
  - Specifications
  - Weeks

- **Deployment and Testing**
  - Reports
  - Weeks

- **Customer Feedback**
  - It sucks!
  - Weeks
**Waterfall Product Development**

- **Business Need**
  - Documents
  - Weeks

- **Approval Process**
  - Meetings
  - Weeks

- **Hardware Purchase**
  - Negotiations
  - Weeks

- **Software Development**
  - Specifications
  - Weeks

- **Deployment and Testing**
  - Reports
  - Weeks

- **Customer Feedback**
  - It sucks!
  - Weeks

*Hardware provisioning is undifferentiated heavy lifting – replace it with IaaS*
Hardware provisioning is undifferentiated heavy lifting – replace it with IaaS
Waterfall Product Development

- **Business Need**
  - Documents
  - Weeks

- **Software Development**
  - Specifications
  - Weeks

- **Deployment and Testing**
  - Reports
  - Weeks

- **Customer Feedback**
  - It sucks!
  - Weeks

---

**Hardware provisioning is undifferentiated heavy lifting – replace it with IaaS**
Process Hand-Off Steps for Agile Development on IaaS

1. Product Manager
2. Development Team
3. QA Integration Team
4. Operations Deploy Team
5. BI Analytics Team
IaaS Agile Product Development

Business Need
- Documents
- Weeks

Software Development
- Specifications
- Weeks

Deployment and Testing
- Reports
- Days

Customer Feedback
- It sucks!
- Days
IaaS Agile Product Development

- **Business Need**
  - Documents
  - Weeks

- **Software Development**
  - Specifications
  - Weeks

- **Deployment and Testing**
  - Reports
  - Days

- **Customer Feedback**
  - It sucks!
  - Days

etc...
IaaS Agile Product Development

- Business Need
  - Documents
  - Weeks
- Software Development
  - Specifications
  - Weeks
- Deployment and Testing
  - Reports
  - Days
- Customer Feedback
  - It sucks!
  - Days

etc...
Software provisioning is undifferentiated heavy lifting – replace it with PaaS
IaaS Agile Product Development

- Business Need: Documents, Weeks
- Software Development: Specifications, Weeks
- Deployment and Testing: Reports, Days
- Customer Feedback: It sucks!, Days

Software provisioning is undifferentiated heavy lifting – replace it with PaaS
Software provisioning is undifferentiated heavy lifting – replace it with PaaS
Process for Continuous Delivery of Features on PaaS

Product Manager

Developer

BI Analytics Team
PaaS CD Feature Development

Business Need
- Discussions
- Days

Software Development
- Code
- Days

Customer Feedback
- Fix this Bit!
- Hours

etc…
PaaS CD Feature Development

- Business Need
  - Discussions
  - Days

- Software Development
  - Code
  - Days

- Customer Feedback
  - Fix this Bit!
  - Hours

etc…
PaaS CD Feature Development

Business Need
• Discussions
• Days

Software Development
• Code
• Days

Customer Feedback
• Fix this Bit!
• Hours

etc…
Building your own business apps is undifferentiated heavy lifting – use SaaS
Building your own business apps is undifferentiated heavy lifting – use SaaS
Building your own business apps is undifferentiated heavy lifting – use SaaS
SaaS Based Business Application Development

Business Need
• GUI Builder
• Hours

Customer Feedback
• Fix this bit!
• Seconds
SaaS Based Business Application Development

Business Need
• GUI Builder
• Hours

Customer Feedback
• Fix this bit!
• Seconds

and thousands more...
Observe

Orient

Decide

Act

Continuous Delivery
Observe

Orient

Decide

Act

Continuous Delivery

Land grab opportunity

Competitive Move

Measure Customers

Customer Pain Point
INNOVATION

Observation:
- Land grab opportunity
- Measure Customers

Orientation:
- Competitive Move
- Customer Pain Point

Decision:
- Analysis
- Model Hypotheses

Action:
- Continuous Delivery
INNOVATION

- Observe
  - Land grab opportunity
  - Competitive Move
  - Customer Pain Point

- Decide

- Act
  - Measure Customers
  - Continuous Delivery
  - Analysis
  - Model Hypotheses

- Orient

BIG DATA

Continuous Delivery

Customers

Continuous
Delivery

Hypotheses

Model

INNOVATION
INNOVATION

Observe

- Land grab opportunity
- Competitive Move
- Customer Pain Point

Orient

- Measure Customers

Decide

- Continuous Delivery

Act

- Analysis
- Model Hypotheses

Share Plans

Plan Response

JFDI

BIG DATA

Measurement

Customers

Continuous Delivery
INNOVATION

- Land grab opportunity
- Competitive Move
- Customer Pain Point

CULTURE

- Launch AB Test
- Automatic Deploy
- Incremental Features
- Share Plans

BIG DATA

- Analysis
- Model Hypotheses
- Plan Response
- JFDI

Observe

Orient

Decide

Act

Continuous Delivery
INNOVATION

- Land grab opportunity
- Competitive Move
- Customer Pain Point

CLOUD

- Launch AB Test
- Automatic Deploy
- Incremental Features

CONTINUOUS DELIVERY

- Observe
- Act

- Measure Customers
- Continuous Delivery

CULTURE

- Orient
- Decide

- Share Plans
- BIG DATA
- Analysis
- Model Hypotheses
- Plan Response
- JFDI

INNOVATION

CULTURE

CLOUD

CONTINUOUS DELIVERY
Breaking Down the SILOs
Breaking Down the SILOs

Prod Mgr  UX  Dev  QA  DBA  Sys Adm  Net Adm  SAN Adm
Breaking Down the SILOs

Product Team Using Monolithic Delivery

Prod Mgr  UX  Dev  QA  DBA  Sys Adm  Net Adm  SAN Adm
Breaking Down the SILOs

Product Team Using Monolithic Delivery

Product Team Using Microservices

Prod Mgr  UX  Dev  QA  DBA

Product Team Using Monolithic Delivery

Sys Adm  Net Adm  SAN Adm

Product Team Using Microservices

Product Team Using Microservices

Product Team Using Microservices
Breaking Down the SILOs

Product Team Using Monolithic Delivery

Product Team Using Microservices

Product Team Using Microservices

Product Team Using Microservices

Product Team Using Microservices

Product Team Using Microservices

Platform Team
Breaking Down the SILOs

Product Team Using Monolithic Delivery

Product Team Using Monolithic Delivery

Product Team Using Microservices

Product Team Using Microservices

Product Team Using Microservices

Product Team Using Microservices

Product Team Using Microservices

Platform Team
Breaking Down the SILOs

DevOps is a Re-Org!
Monolithic service updates

Works well with a small number of developers and a single language like php, java or ruby
Monolithic service updates

Works well with a small number of developers and a single language like php, java or ruby
Monolithic service updates

Works well with a small number of developers and a single language like php, java or ruby
Use monolithic apps for small teams, simple systems and when you must, to optimize for efficiency and latency.
Immutable microservice deployment scales, is faster with large teams and diverse platform components
Immutable microservice deployment scales, is faster with large teams and diverse platform components
Immutable microservice deployment scales, is faster with large teams and diverse platform components.
Immutable microservice deployment scales, is faster with large teams and diverse platform components.
Standardized container deployment saves time and effort

https://hub.docker.com
Standardized container deployment saves time and effort

https://hub.docker.com
Run What You Wrote
Run What You Wrote

Diagram showing developers and microservices.
Run What You Wrote
Run What You Wrote
Run What You Wrote

99.95% customer success rate
Run What You Wrote

99.95% customer success rate

- Site Reliability
- Monitoring Tools
- Availability Metrics
- Microservice

Manager

Developer

Manager

Developer
Run What You Wrote

99.95% customer success rate

Availability Metrics

Site Reliability

Monitoring Tools

VP Engineering

Manager

Developer

Micro service

Micro service

Micro service

Micro service

Micro service

Micro service

Micro service
Non-Destructive Production Updates

- “Immutable Code” Service Pattern
  - Existing services are unchanged, old code remains in service
  - New code deploys as a new service group
  - No impact to production until traffic routing changes
- A|B Tests, Feature Flags and Version Routing control traffic
  - First users in the test cell are the developer and test engineers
  - A cohort of users is added looking for measurable improvement
**Deliver four features every four weeks**

<table>
<thead>
<tr>
<th>Work In Progress</th>
<th>Opportunity for bugs: 100% (baseline)</th>
<th>Time to debug each: 100% (baseline)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Deliver four features every four weeks

Bugs! Which feature broke?
Need more time to test!
Extend release to six weeks?

Work In Progress = 4
Opportunity for bugs: 100% (baseline)
Time to debug each: 100% (baseline)
Deliver four features every four weeks

Bugs! Which feature broke?
Need more time to test!
Extend release to six weeks?

Work In Progress = 4
Opportunity for bugs: 100% (baseline)
Time to debug each: 100% (baseline)

But: risk of bugs in delivery increases with interactions!
Deliver four features every four weeks

Bugs! Which feature broke?
Need more time to test!
Extend release to six weeks?

Work In Progress = 4
Opportunity for bugs: 100% (baseline)
Time to debug each: 100% (baseline)

But: risk of bugs in delivery increases with interactions!
Deliver six features every six weeks
Deliver six features every six weeks

Work In Progress = 6
Individual bugs: 150%
Interactions: 150%?
Deliver six features every six weeks

More features
What broke?
More interactions
Even more bugs!!

Work In Progress = 6
Individual bugs: 150%
Interactions: 150%?
Deliver six features every six weeks

More features
What broke?
More interactions
Even more bugs!!

Work In Progress = 6
Individual bugs: 150%
Interactions: 150%?
Deliver six features every six weeks

More features
What broke?
More interactions
Even more bugs!!

Work In Progress = 6
Individual bugs: 150%
Interactions: 150%?

Risk of bugs in delivery increased to 225% of original!
Deliver two features every two weeks

- Fewer interactions
- Fewer bugs
- Better flow
- Less Work In Progress

Work In Progress = 2
Opportunity for bugs: 50%
Time to debug each: 50%

Complexity of delivery decreased by 75% from original
Change One Thing at a Time!
What Happened?

Rate of change increased

Cost and size and risk of change reduced
Low Cost of Change Using Docker

Developers
• Compile/Build
• Seconds

Extend container
• Package dependencies
• Seconds

PaaS deploy Container
• Docker startup
• Seconds
Low Cost of Change Using Docker

- Developers
  - Compile/Build
  - Seconds
- Extend container
  - Package dependencies
  - Seconds
- PaaS deploy Container
  - Docker startup
  - Seconds

Fast tooling supports continuous delivery of many tiny changes
Disruptor: Continuous Delivery with Containerized Microservices
It’s what you know that isn’t so
It’s what you know that isn’t so

- Make your assumptions explicit
It’s what you know that isn’t so

- Make your assumptions explicit
- Extrapolate trends to the limit
It’s what you know that isn’t so

• Make your assumptions explicit
• Extrapolate trends to the limit
• Listen to non-customers
It’s what you know that isn’t so

- Make your assumptions explicit
- Extrapolate trends to the limit
- Listen to non-customers
- Follow developer adoption, not IT spend
It’s what you know that isn’t so

• *Make your assumptions explicit*

• *Extrapolate trends to the limit*

• *Listen to non-customers*

• *Follow developer adoption, not IT spend*

• *Map evolution of products to services to utilities*
It’s what you know that isn’t so

- Make your assumptions explicit
- Extrapolate trends to the limit
- Listen to non-customers
- Follow developer adoption, not IT spend
- Map evolution of products to services to utilities
- Re-organize your teams for speed of execution
Microservices
A Microservice Definition

Loosely coupled service oriented architecture with bounded contexts
A Microservice Definition

Loosely coupled service oriented architecture with bounded contexts

If every service has to be updated at the same time it’s not loosely coupled
A Microservice Definition

Loosely coupled service oriented architecture with bounded contexts

If every service has to be updated at the same time it’s not loosely coupled.

If you have to know too much about surrounding services you don’t have a bounded context. See the Domain Driven Design book by Eric Evans.
Coupling Concerns

- Conway’s Law - organizational coupling
- Centralized Database Schemas
- Enterprise Service Bus - centralized message queues
- Inflexible Protocol Versioning

http://en.wikipedia.org/wiki/Conway's_law
Speeding Up The Platform

Datacenter Snowflakes
- Deploy in months
- Live for years
Speeding Up The Platform

Datacenter Snowflakes
- Deploy in months
- Live for years

Virtualized and Cloud
- Deploy in minutes
- Live for weeks
Speeding Up The Platform

Datacenter Snowflakes
• Deploy in months
• Live for years

Virtualized and Cloud
• Deploy in minutes
• Live for weeks

Container Deployments
• Deploy in seconds
• Live for minutes/hours
Speeding Up The Platform

Datacenter Snowflakes
- Deploy in months
- Live for years

Virtualized and Cloud
- Deploy in minutes
- Live for weeks

Container Deployments
- Deploy in seconds
- Live for minutes/hours

Lambda Deployments
- Deploy in milliseconds
- Live for seconds
Speeding Up The Platform

Speed enables and encourages microservice architectures
Separate Concerns with Microservices

- Invert Conway’s Law – teams own service groups and backend stores
- One “verb” per single function micro-service, size doesn’t matter
- One developer independently produces a micro-service
- Each micro-service is it’s own build, avoids trunk conflicts
- Deploy in a container: Tomcat, AMI or Docker, whatever...
- Stateless business logic. Cattle, not pets.
- Stateful cached data access layer using replicated ephemeral instances

http://en.wikipedia.org/wiki/Conway's_law
Inspiration
State of the Art in Web Scale Microservice Architectures

AWS Re:Invent : Asgard to Zuul https://www.youtube.com/watch?v=p7ysHhs5hl0
Resiliency at Massive Scale https://www.youtube.com/watch?v=ZfYJHtVL1_w
Microservice Architecture https://www.youtube.com/watch?v=CriDUYtfrjs

http://www.infoq.com/presentations/scale-gilt

http://www.slideshare.net/mcculloughsean/itier-breaking-up-the-monolith-philly-ete

http://www.infoq.com/presentations/Twitter-Timeline-Scalability
http://www.infoq.com/presentations/twitter-soa
http://www.infoq.com/presentations/Zipkin

Microservice Concerns

- Tooling
- Configuration
- Discovery
- Routing
- Observability

Datastores

Operational: Orchestration and Deployment Infrastructure

Development: Languages and Container
Microservices

- Asgard, Aminator, Tooling
- Edda, Archaius, Configuration
- Eureka, Prana, Discovery
- Denominator, Zuul, Netty, Ribbon 2.0, Routing
- Hystrix, Pytheus, SALP, Observability

Ephemeral datastores using Dynomite, Memcached, Astyanax, Staash, Priam, Cassandra

Manual Orchestration with Asgard and deployment on AWS or Eucalyptus

Java, Groovy, Scala, Clojure, Python with AMI and Docker Containers
Microservices

Asgard
Aminator
Tooling

Edda
Archaius
Configuration

Eureka
Prana
Discovery

Denominator
Zuul, Netty
Ribbon 2.0
Routing

Hystrix
Pytheus
SALP
Observability

Ephemeral datastores using Dynomite, Memcached, Astyanax, Staash, Priam, Cassandra

Manual Orchestration with Asgard and deployment on AWS or Eucalyptus

Java, Groovy, Scala, Clojure, Python with AMI and Docker Containers

Focus on global distribution, high scale and availability
Cloud Native Storage

Diagram:
- Business Logic
  - Database Master
    - Fabric
      - Storage Arrays
    - Database Slave
      - Fabric
        - Storage Arrays
Cloud Native Storage

- Business Logic
  - Database Master
    - Fabric
      - Storage Arrays
  - Database Slave
    - Fabric
      - Storage Arrays

- Business Logic
  - Cassandra Zone A nodes
  - Cassandra Zone B nodes
  - Cassandra Zone C nodes

- Cloud Object Store Backups
Cloud Native Storage

- Database Master
- Database Slave
- Fabric
- Storage Arrays

- Cassandra Zone A nodes
- Cassandra Zone B nodes
- Cassandra Zone C nodes

SSDs inside arrays disrupt incumbent suppliers

Cloud Object Store Backups
Cloud Native Storage

SSDs inside arrays disrupt incumbent suppliers

SSDs inside ephemeral instances disrupt an entire industry
Cloud Native Storage

Netflix OSS Uses Priam to create Cassandra clusters in minutes
- Edda - the “black box flight recorder” for configuration state
- Chaos Monkey - enforcing stateless business logic
- Chaos Gorilla - enforcing zone isolation/replication
- Chaos Kong - enforcing region isolation/replication
- Security Monkey - watching for insecure configuration settings
- See over 40 NetflixOSS projects at netflix.github.com
- Get “Technical Indigestion” trying to keep up with techblog.netflix.com
These companies are using and contributing to Netflix OSS Components

Email netflixoss@netflix.com to have your logo here.
Twitter Microservices

- Tooling
- Decider Configuration
- Finagle Zookeeper Discovery
- Finagle Netty Routing
- Zipkin Observability

Custom Cassandra-like datastore: Manhattan
Orchestration using Aurora deployment in datacenters using Mesos
Scala with JVM Container
Twitter Microservices

Tooling

Decider
Configuration

Finagle
Zookeeper
Discovery

Finagle
Netty
Routing

Zipkin
Observability

Custom Cassandra-like datastore: Manhattan

Orchestration using Aurora deployment in datacenters using Mesos

Scala with JVM Container

Focus on efficient datacenter deployment at scale
Gilt Microservices

- Ion Cannon
- SBT
- Rake
- Tooling
- Decider
- Configuration
- Finagle
- Zookeeper
- Discovery
- Akka
- Finagle
- Netty
- Routing
- Zipkin
- Observability

- Datastores per Microservice using MongoDB, Postgres, Voldemort
- Deployment on AWS
- Scala and Ruby with Docker Containers
Gilt Microservices

- Ion Cannon
- SBT
- Rake
- Tooling
- Decider
- Configuration
- Finagle
- Zookeeper
- Discovery
- Akka
- Finagle
- Netty
- Routing
- Zipkin
- Observability

Datastores per Microservice using MongoDB, Postgres, Voldemort

Deployment on AWS

Scala and Ruby with Docker Containers

Focus on fast development with Scala and Docker
Hailo Microservices

Hubot
Janky
Jenkins
Tooling

Configuration

go-platform
Discovery

go-platform
RabbitMQ
Routing

Observability

Deployment on AWS

Deployment on AWS

Go using AMI Container and Docker
Hailo Microservices

Hubot
Janky
Jenkins
Tooling

Configuration

go-platform
Discovery

RabbitMQ
Routing

Observability

Deployment on AWS

Deployment on AWS

Go using AMI Container and Docker

Also watching: https://github.com/peterbourgon/gokit
Microservices

Tooling
Gconfig
Configuration
Grout
Discovery
Grout
Routing
Observability

Deployment Infrastructure

Javascript with Node.js container
With AWS Lambda compute resources are charged by the 100ms, not the hour

First 1M node.js executions/month are free
Next Generation Applications

Docker? PaaS?
Tooling

Configuration

Eureka? Consul?
Discovery


Zipkin? Metrics? Hystrix?
Observability

Datastores: Distributed Ephemeral, Orchestrated or DBaaS

Operational: Many orchestration choices across public and private clouds

Development: Components assembled from Docker Hub as a composable “app store”
Next Generation Applications

Docker? PaaS?
Tooling

Configuration

Eureka? Consul?
Discovery


Zipkin? Metrics? Hystrix?
Observability

Datastores: Distributed Ephemeral, Orchestrated or DBaaS

Operational: Many orchestration choices across public and private clouds

Development: Components assembled from Docker Hub as a composable “app store”

Fill in the gaps, rapidly evolving ecosystem choices
Cloud Native Monitoring and Microservices
Cloud Native Microservices

- **High rate of change**
  - Code pushes can cause floods of new instances and metrics
  - Short baseline for alert threshold analysis – everything looks unusual

- **Ephemeral Configurations**
  - Short lifetimes make it hard to aggregate historical views
  - Hand tweaked monitoring tools take too much work to keep running

- **Microservices with complex calling patterns**
  - End-to-end request flow measurements are very important
  - Request flow visualizations get overwhelmed
Microservice Based Architectures

AS OF LAST WEEK WE HAVE MORE THAN 450 SERVICES

See http://www.slideshare.net/LappleApple/gilt-from-monolith-ruby-app-to-micro-service-scala-service-architecture
Continuous Delivery and DevOps

- Changes are smaller but more frequent
- Individual changes are more likely to be broken
- Changes are normally deployed by developers
- Feature flags are used to enable new code
- Instant detection and rollback matters much more
Whoops! I didn’t mean that! Reverting...

Not cool if it takes 5 minutes to see it failed and 5 more to see a fix
No-one notices if it only takes 5 seconds to detect and 5 to see a fix
NetflixOSS Hystrix/Turbine Circuit Breaker

Netflix OSS Hystrix/Turbine Circuit Breaker

Low Latency SaaS Based Monitors

www.vividcortex.com signalfx.com wavefront.com sysdig.com
Metric to display latency needs to be less than human attention span (~10s)
Challenges for Microservice Platforms
Managing Scale
It's much more challenging than just a large number of machines

A Possible Hierarchy
- Continents
- Regions
- Zones
- Services
- Versions
- Containers
- Instances

How Many?
- 3 to 5
- 2-4 per Continent
- 1-5 per Region
- 100’s per Zone
- Many per Service
- 1000’s per Version
- 10,000’s
Flow
Some tools can show the request flow across a few services.
But interesting architectures have a lot of microservices! Flow visualization is a big challenge.

See http://www.slideshare.net/LappleApple/gilt-from-monolith-ruby-app-to-micro-service-scala-service-architecture
Failures
ELB Load Balancer

Zuul API Proxy

Karyon Business Logic

Staash Data Access Layer

Priam Cassandra Datastore

Simple Netflix OSS style microservices architecture on three AWS Availability Zones
Simple Netflix OSS style microservices architecture on three AWS Availability Zones.
Simple Netflix OSS style microservices architecture on three AWS Availability Zones.

What should you do?
What should monitors show?

Zone partition/failure

ELB Load Balancer

Zuul API Proxy

Karyon Business Logic

Staash Data Access Layer

Priam Cassandra Datastore
Simple Netflix OSS style microservices architecture on three AWS Availability Zones

- ELB Load Balancer
- Zuul API Proxy
- Karyon Business Logic
- Staash Data Access Layer
- Priam Cassandra Datastore

Zone partition/failure
What should you do?
What should monitors show?

By design, everything works with 2 of 3 zones running. This is not an outage, inform but don’t touch anything!
Halt deployments perhaps?
Simple Netflix OSS style microservices architecture on three AWS Availability Zones

Zone partition/failure
What should you do?
What should monitors show?

By design, everything works with 2 of 3 zones running.
This is not an outage, inform but don’t touch anything!
Halt deployments perhaps?

Challenge: understand and communicate common microservice failure patterns.
Testing
Testing monitoring tools at scale gets expensive quickly...
Simulation
Simulated Microservices

Model and visualize microservices
Simulate interesting architectures
Generate large scale configurations
Eventually stress test real tools

See [github.com/adriano/spigo](https://github.com/adriano/spigo)
Simulate Protocol Interactions in Go
Visualize with D3
Definition of an architecture

Header includes chaos monkey victim

New tier name

Tier package

Node count

Region count: 1

List of tier dependencies
Why Build Spigo/Simianviz?

Generate test microservice configurations at scale
Stress monitoring tools display capabilities

Eventually (i.e. not implemented yet)
Dynamically vary configuration: autoscale, code push
Chaos monkey for microservice, zone, region failures
D3 websocket dynamic browser interface
Simulated Game Day Testing and Training...
Conference Driven Development...

OSCON Microservices Workshop
Day 2 - Explore and extend Spigo and Simianviz
What’s Next?
Developer Concerns
Agile, Lean & Rugged
(Faster, Cheaper and Safer)

See GOTO London Sept 2015
Forward Thinking
Forward Thinking
Q&A

Adrian Cockcroft @adrianco
http://slideshare.com/adriancockcroft
Technology Fellow - Battery Ventures
July 2015

See www.battery.com for a list of portfolio investments
Visit [http://www.battery.com/our-companies/](http://www.battery.com/our-companies/) for a full list of all portfolio companies in which all Battery Funds have invested.