Migrating to Microservices

Adrian Cockcroft @adrianco
Technology Fellow - Battery Ventures
GOTO Berlin - November 2014
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

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

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

By Simon Wardley http://enterpriseitadoption.com/
Cloud Adoption

By Simon Wardley http://enterpriseitadoption.com/
Cloud Adoption

By Simon Wardley http://enterpriseitadoption.com/
Cloud Adoption

@adrianco’s new job at the intersection of cloud and Enterprise IT

By Simon Wardley http://enterpriseitadoption.com/
This is the year that Enterprises finally embraced cloud.
This is the year that Enterprises finally embraced cloud.
This is the year that Enterprises finally embraced cloud.
This is the year that Enterprises finally embraced cloud.
What separates incumbents from disruptors?
“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
Observe

Orient

Decide

Act

Continuous Delivery
Observe

Orient

Decide

Act

Land grab opportunity

Measure Customers

Competitive Move

Customer Pain Point

Continuous Delivery
Observe

Land grab opportunity

Measure Customers

Competitive Move

Customer Pain Point

INNOVATION

Observe

Act

Continuous Delivery

Orient

Decide
Observe
Orient
Decide
Act

Land grab opportunity
Competitive Move
Customer Pain Point

Measure Customers

Continuous Delivery

Analysis
Model Hypotheses

BIG DATA

Continuous Delivery

Decide

Act

INNOVATION
Observe

Orient

Decide

Act

Land grab opportunity

Competitive Move

Customer Pain Point

Measure Customers

BIG DATA

Analysis

Model Hypotheses

CULTURE

Innovation

Continuous Delivery

Incremental Features

Automatic Deploy

Launch AB Test

Share Plans

Plan Response

JFDI
Observe

Orient

Decide

Act

Land grab opportunity

Competitive Move

Customer Pain Point

Measure Customers

Analysis

Model Hypotheses

Launch AB Test

Continuous Delivery

Automatic Deploy

CLOUD

Incremental Features

Act

CULTURE

Share Plans

Continuous Delivery

Plan Response

JFDI

BIG DATA

Analysis

INNOVATION

CULTURE
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

Product Team Using Microservices
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

Platform Team

Prod Mgr  UX  Dev  QA  DBA  Sys Adm  Net Adm  SAN Adm
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

Platform Team
Breaking Down the SILOs

DevOps is a Re-Org

Product Team Using Monolithic Delivery

Product Team Using Monolithic Delivery

Product Team Using Microservices

Product Team Using Microservices

Product Team Using Microservices

Platform Team

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
Immutable microservice deployment is faster, scales with large teams and diverse platform components.
Immutable microservice deployment is faster, scales with large teams and diverse platform components.
Immutable microservice deployment is faster, scales with large teams and diverse platform components.
Immutable microservice deployment is faster, scales with large teams and diverse platform components.
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
  - Finally make default for everyone, keeping old code for a while
Developing at the Speed of Docker

Developers
- Compile/Build
- Seconds

Extend container
- Package dependencies
- Seconds

PaaS deploy Container
- Docker startup
- Seconds

eetc…
Developing at the Speed of Docker

- Developers
  - Compile/Build
  - Seconds

- Extend container
  - Package dependencies
  - Seconds

- PaaS deploy Container
  - Docker startup
  - Seconds

Emerging market for Docker runtime orchestration options
What Happened?

Rate of change increased

Cost and size and risk of change reduced
Disruptor: Continuous Delivery with 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.
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 its 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
High Availability Patterns

- Business logic isolation in stateless micro-services
- Immutable code with instant rollback
- Auto-scaled capacity and deployment updates
- Distributed across availability zones and regions
- De-normalized single function NoSQL data stores
- See over 40 Netflix OSS projects at netflix.github.com
- Get “Technical Indigestion” trying to keep up with techblog.netflix.com
Table 2 - Top 10 Peak Period Applications - North America, Fixed Access

<table>
<thead>
<tr>
<th>Rank</th>
<th>Application</th>
<th>Share</th>
<th>Application</th>
<th>Share</th>
<th>Application</th>
<th>Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BitTorrent</td>
<td>24.53%</td>
<td>Netflix</td>
<td>34.21%</td>
<td>Netflix</td>
<td>31.09%</td>
</tr>
<tr>
<td>2</td>
<td>HTTP</td>
<td>14.27%</td>
<td>YouTube</td>
<td>13.19%</td>
<td>YouTube</td>
<td>12.28%</td>
</tr>
<tr>
<td>3</td>
<td>SSL</td>
<td>6.54%</td>
<td>HTTP</td>
<td>11.65%</td>
<td>HTTP</td>
<td>11.84%</td>
</tr>
<tr>
<td>4</td>
<td>Netflix</td>
<td>6.44%</td>
<td>iTunes</td>
<td>3.64%</td>
<td>BitTorrent</td>
<td>5.96%</td>
</tr>
<tr>
<td>5</td>
<td>YouTube</td>
<td>5.52%</td>
<td>SSL</td>
<td>3.42%</td>
<td>SSL</td>
<td>3.80%</td>
</tr>
<tr>
<td>6</td>
<td>Skype</td>
<td>2.23%</td>
<td>BitTorrent</td>
<td>3.40%</td>
<td>iTunes</td>
<td>3.33%</td>
</tr>
<tr>
<td>7</td>
<td>Facebook</td>
<td>2.17%</td>
<td>MPEG</td>
<td>2.85%</td>
<td>MPEG</td>
<td>2.62%</td>
</tr>
<tr>
<td>8</td>
<td>FaceTime</td>
<td>1.50%</td>
<td>Facebook</td>
<td>1.99%</td>
<td>Facebook</td>
<td>1.83%</td>
</tr>
<tr>
<td>9</td>
<td>Dropbox</td>
<td>1.20%</td>
<td>Amazon Video</td>
<td>1.90%</td>
<td>Amazon Video</td>
<td>1.82%</td>
</tr>
<tr>
<td>10</td>
<td>iTunes</td>
<td>1.15%</td>
<td>Hulu</td>
<td>1.74%</td>
<td>Hulu</td>
<td>1.58%</td>
</tr>
</tbody>
</table>
US Bandwidth April 2014

<table>
<thead>
<tr>
<th>Rank</th>
<th>Application</th>
<th>Share</th>
<th>Application</th>
<th>Share</th>
<th>Application</th>
<th>Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BitTorrent</td>
<td>24.53%</td>
<td>Netflix</td>
<td>34.21%</td>
<td>Netflix</td>
<td>31.09%</td>
</tr>
<tr>
<td>2</td>
<td>HTTP</td>
<td>14.27%</td>
<td>YouTube</td>
<td>13.19%</td>
<td>YouTube</td>
<td>12.28%</td>
</tr>
<tr>
<td>3</td>
<td>SSL</td>
<td>6.54%</td>
<td>HTTP</td>
<td>11.65%</td>
<td>HTTP</td>
<td>11.84%</td>
</tr>
<tr>
<td>4</td>
<td>Netflix</td>
<td>6.44%</td>
<td>iTunes</td>
<td>3.64%</td>
<td>BitTorrent</td>
<td>5.96%</td>
</tr>
<tr>
<td>5</td>
<td>YouTube</td>
<td>5.52%</td>
<td>SSL</td>
<td>3.42%</td>
<td>SSL</td>
<td>3.80%</td>
</tr>
<tr>
<td>6</td>
<td>Skype</td>
<td>2.23%</td>
<td>BitTorrent</td>
<td>3.40%</td>
<td>iTunes</td>
<td>3.33%</td>
</tr>
<tr>
<td>7</td>
<td>Facebook</td>
<td>2.17%</td>
<td>MPEG</td>
<td>2.85%</td>
<td>MPEG</td>
<td>2.62%</td>
</tr>
<tr>
<td>8</td>
<td>FaceTime</td>
<td>1.50%</td>
<td>Facebook</td>
<td>1.99%</td>
<td>Facebook</td>
<td>1.83%</td>
</tr>
<tr>
<td>9</td>
<td>Dropbox</td>
<td>1.20%</td>
<td>Amazon Video</td>
<td>1.90%</td>
<td>Amazon Video</td>
<td>1.82%</td>
</tr>
<tr>
<td>10</td>
<td>iTunes</td>
<td>1.15%</td>
<td>Hulu</td>
<td>1.74%</td>
<td>Hulu</td>
<td>1.58%</td>
</tr>
</tbody>
</table>

Table 2 - Top 10 Peak Period Applications - North America, Fixed Access
### Table 2 - Top 10 Peak Period Applications - North America, Fixed Access

<table>
<thead>
<tr>
<th>Rank</th>
<th>Application</th>
<th>Share</th>
<th>Application</th>
<th>Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BitTorrent</td>
<td>24.53%</td>
<td>Netflix</td>
<td>34.21%</td>
</tr>
<tr>
<td>2</td>
<td>HTTP</td>
<td>14.27%</td>
<td>YouTube</td>
<td>13.19%</td>
</tr>
<tr>
<td>3</td>
<td>SSL</td>
<td>6.54%</td>
<td>HTTP</td>
<td>11.65%</td>
</tr>
<tr>
<td>4</td>
<td>Netflix</td>
<td>6.44%</td>
<td>iTunes</td>
<td>3.64%</td>
</tr>
<tr>
<td>5</td>
<td>YouTube</td>
<td>5.92%</td>
<td>SSL</td>
<td>3.42%</td>
</tr>
<tr>
<td>6</td>
<td>Skype</td>
<td>2.23%</td>
<td>BitTorrent</td>
<td>3.40%</td>
</tr>
<tr>
<td>7</td>
<td>Facebook</td>
<td>2.17%</td>
<td>MPEG</td>
<td>2.85%</td>
</tr>
<tr>
<td>8</td>
<td>FaceTime</td>
<td>1.50%</td>
<td>Facebook</td>
<td>1.99%</td>
</tr>
<tr>
<td>9</td>
<td>Dropbox</td>
<td>1.20%</td>
<td>Amazon Video</td>
<td>1.90%</td>
</tr>
<tr>
<td>10</td>
<td>iTunes</td>
<td>1.15%</td>
<td>Hulu</td>
<td>1.74%</td>
</tr>
</tbody>
</table>

**Upstream**

**Downstream**

**Aggregate**

*ELB: Amazon Web Services* 

*NGiNX: OpenConnect*
Microservices Development

- **Client libraries**
  
  *Even if you start with a raw protocol, a client side driver is the end-state*
  
  *Best strategy is to own your own client libraries from the start*

- **Multithreading and Non-blocking Calls**
  
  *Reactive model RxJava uses Observable to hide concurrency cleanly*
  
  *Netty can be used to get non-blocking I/O speedup over Tomcat container*

- **Circuit Breakers – See Fluxcapacitor.com for code**
  
  *NetflixOSS Hystrix, Turbine, Latency Monkey, Ribbon/Karyon*
  
  *Also look at Finagle/Zipkin from Twitter*
Microservice Datastores

- **Book: Refactoring Databases**
  - SchemaSpy to examine schema structure
  - Denormalization into one datasource per table or materialized view

- **Polyglot Persistence**
  - Use a mixture of database technologies, behind REST data access layers
  - See Netflix OSS Storage Tier as a Service HTTP ([staash.com](https://staash.com)) for MySQL and C*

- **CAP – Consistent or Available when Partitioned**
  - Look at Jepsen torture tests for common systems ([aphyr.com/tags/jepsen](https://aphyr.com/tags/jepsen))
  - There is no such thing as a consistent distributed system, get over it...
Cloud Native Monitoring and Microservices
Cloud Native

- **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
“Death Star” Architecture Diagrams

As visualized by Appdynamics, Boundary.com and Twitter internal tools
“Death Star” Architecture Diagrams

Netflix

Gilt Groupe (12 of 450)

Twitter

As visualized by Appdynamics, Boundary.com and Twitter internal tools
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

NetflixOSS Hystrix/Turbine Circuit Breaker

Low Latency SaaS Based Monitors

www.vividcortex.com and www.boundary.com

1-second data collection and real-time streaming processing on all components of the application stack.
Metric to display latency needs to be less than human attention span (~10s)
Prototyping Ideas

Model and visualize microservices

See [github.com/adrianco/spigo](https://github.com/adrianco/spigo)

Simulate Protocol Interactions in Go

See [github.com/adrianco/d3grow](https://github.com/adrianco/d3grow)

Dynamic visualization concept
Separation of Concerns

Bounded Contexts
Forward Thinking
Forward Thinking
Forward Thinking

LEAN ENTERPRISE
Adopting Continuous Delivery, DevOps, and Lean Startup at Scale

Jez Humble, Joanne Molesky, & Barry O'Reilly

O'Reilly
Forward Thinking

http://eugenedvorkin.com/seven-micro-services-architecture-advantages/
Any Questions?

- Battery Ventures [http://www.battery.com](http://www.battery.com)
- Adrian’s Blog [http://perfcap.blogspot.com](http://perfcap.blogspot.com)
- Slideshare [http://slideshare.com/adriancockcroft](http://slideshare.com/adriancockcroft)

- Monitorama Opening Keynote Portland OR - May 7th, 2014 - Video available
- GOTO Chicago Opening Keynote May 20th, 2014 - Video available
- Qcon New York – Speed and Scale - June 11th, 2014 - Video available
- Structure - Cloud Trends - San Francisco - June 19th, 2014 - Video available
- GOTO Copenhagen/Aarhus – Denmark – Sept 25th, 2014
- DevOps Enterprise Summit - San Francisco - Oct 21-23rd, 2014 #DOES14 - Videos available
- GOTO Berlin - Germany - Nov 6th, 2014
- AWS Re:Invent - Cloud Native Cost Optimization - Las Vegas - November 14th, 2014
- Dockercon Europe - Amsterdam - December 4th, 2014

Disclosure: some of the companies mentioned are Battery Ventures Portfolio Companies
See [www.battery.com](http://www.battery.com) for a list of portfolio investments