NetflixOSS – A Cloud Native Architecture

LASER Sessions 2&3 – Overview
September 2013
Adrian Cockcroft
@adrianco @NetflixOSS
http://www.linkedin.com/in/adriancockcroft
Presentation vs. Tutorial

• Presentation
  – Short duration, focused subject
  – One presenter to many anonymous audience
  – A few questions at the end

• Tutorial
  – Time to explore in and around the subject
  – Tutor gets to know the audience
  – Discussion, rat-holes, “bring out your dead”
Attendee Introductions

- Who are you, where do you work
- Why are you here today, what do you need
- “Bring out your dead”
  - Do you have a specific problem or question?
  - One sentence elevator pitch
- What instrument do you play?
Content

Why Public Cloud?

Migration Path

Service and API Architectures

Storage Architecture

Operations and Tools

Example Applications
Cloud Native
A new engineering challenge

Construct a highly agile and highly available service from ephemeral and assumed broken components
How to get to Cloud Native

Freedom and Responsibility for Developers

Decentralize and Automate Ops Activities

Integrate DevOps into the Business Organization

Re-Org!
Four Transitions

• Management: Integrated Roles in a Single Organization
  – Business, Development, Operations -> BusDevOps

• Developers: Denormalized Data – NoSQL
  – Decentralized, scalable, available, polyglot

• Responsibility from Ops to Dev: Continuous Delivery
  – Decentralized small daily production updates

• Responsibility from Ops to Dev: Agile Infrastructure - Cloud
  – Hardware in minutes, provisioned directly by developers
Netflix BusDevOps Organization

Chief Product Officer

VP Product Management
- Directors Product
- Developers + DevOps
- UI Data Sources
- AWS

VP UI Engineering
- Directors Development
- Developers + DevOps
- Discovery Data Sources
- AWS

VP Discovery Engineering
- Directors Development
- Developers + DevOps
- Platform Data Sources
- AWS

VP Platform
- Directors Platform
- Developers + DevOps
- Platform Data Sources
- AWS

Code, independently updated continuous delivery

Denormalized, independently updated and scaled data

Cloud, self service updated & scaled infrastructure
Decentralized Deployment
Asgard Developer Portal


This cluster contains two ASGs

No traffic on old version

Live traffic on new version
Ephemeral Instances

- Largest services are autoscaled
- Average lifetime of an instance is 36 hours
Netflix Member Web Site Home Page

Personalization Driven – How Does It Work?
How Netflix Used to Work

Customer Device (PC, PS3, TV...)

Monolithic Web App

Oracle
MySQL

Monolithic Streaming App

Oracle
MySQL

Limelight/Level 3 Akamai CDNs

Content Management
Content Encoding
How Netflix Streaming Works Today

- Customer Device (PC, PS3, TV...)
- Web Site or Discovery API
- User Data
- Personalization
- DRM
- QoS Logging
- CDN Management and Steering
- Content Encoding
- OpenConnect CDN Boxes
- Streaming API

- Consumer Electronics
- AWS Cloud Services
- CDN Edge Locations
- Datacenter

- AWS Cloud Services
- CDN Edge Locations
- Datacenter
The AWS Question

Why does Netflix use AWS when Amazon Prime is a competitor?
Netflix vs. Amazon Prime

• Do retailers competing with Amazon use AWS?
  – Yes, lots of them, Netflix is no different

• Does Prime have a platform advantage?
  – No, because Netflix also gets to run on AWS

• Does Netflix take Amazon Prime seriously?
  – Yes, but so far Prime isn’t impacting our growth
**Nov 2012 Streaming Bandwidth**

<table>
<thead>
<tr>
<th>Rank</th>
<th>Application</th>
<th>Share</th>
<th>Rank</th>
<th>Application</th>
<th>Share</th>
<th>Rank</th>
<th>Application</th>
<th>Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BitTorrent</td>
<td>36.8%</td>
<td>1</td>
<td>Netflix</td>
<td>33.0%</td>
<td>1</td>
<td>Netflix</td>
<td>28.8%</td>
</tr>
<tr>
<td>2</td>
<td>HTTP</td>
<td>9.83%</td>
<td>2</td>
<td>YouTube</td>
<td>14.8%</td>
<td>2</td>
<td>YouTube</td>
<td>13.1%</td>
</tr>
<tr>
<td>3</td>
<td>Skype</td>
<td>4.76%</td>
<td>3</td>
<td>HTTP</td>
<td>12.0%</td>
<td>3</td>
<td>HTTP</td>
<td>11.7%</td>
</tr>
<tr>
<td>4</td>
<td>Netflix</td>
<td>4.51%</td>
<td>4</td>
<td>BitTorrent</td>
<td>5.89%</td>
<td>4</td>
<td>BitTorrent</td>
<td>10.3%</td>
</tr>
<tr>
<td>5</td>
<td>SSL</td>
<td>3.73%</td>
<td>5</td>
<td>iTunes</td>
<td>3.92%</td>
<td>5</td>
<td>iTunes</td>
<td>3.43%</td>
</tr>
<tr>
<td>6</td>
<td>YouTube</td>
<td>2.70%</td>
<td>6</td>
<td>MPEG</td>
<td>2.22%</td>
<td>6</td>
<td>SSL</td>
<td>2.23%</td>
</tr>
<tr>
<td>7</td>
<td>PPStream</td>
<td>1.65%</td>
<td>7</td>
<td>Flash Video</td>
<td>2.21%</td>
<td>7</td>
<td>Flash Video</td>
<td>2.01%</td>
</tr>
<tr>
<td>8</td>
<td>Facebook</td>
<td>1.62%</td>
<td>8</td>
<td>SSL</td>
<td>1.97%</td>
<td>8</td>
<td>Facebook</td>
<td>1.50%</td>
</tr>
<tr>
<td>9</td>
<td>Apple PhotoStream</td>
<td>1.46%</td>
<td>9</td>
<td>Apple PhotoStream</td>
<td>1.75%</td>
<td>9</td>
<td>Facebook</td>
<td>1.48%</td>
</tr>
<tr>
<td>10</td>
<td>Dropbox</td>
<td>1.17%</td>
<td>10</td>
<td>Hyundai</td>
<td>1.17%</td>
<td>10</td>
<td>Facebook</td>
<td>1.17%</td>
</tr>
<tr>
<td></td>
<td><strong>Top 10</strong></td>
<td></td>
<td></td>
<td><strong>Top 10</strong></td>
<td></td>
<td></td>
<td><strong>Top 10</strong></td>
<td></td>
</tr>
</tbody>
</table>

**March 2013 Mean Bandwidth +39% 6mo**

<table>
<thead>
<tr>
<th>Rank</th>
<th>Application</th>
<th>Share</th>
<th>Rank</th>
<th>Application</th>
<th>Share</th>
<th>Rank</th>
<th>Application</th>
<th>Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BitTorrent</td>
<td>34.81%</td>
<td>1</td>
<td>Netflix</td>
<td>32.25%</td>
<td>1</td>
<td>Netflix</td>
<td>28.88%</td>
</tr>
<tr>
<td>2</td>
<td>HTTP</td>
<td>7.53%</td>
<td>2</td>
<td>YouTube</td>
<td>17.11%</td>
<td>2</td>
<td>YouTube</td>
<td>15.43%</td>
</tr>
<tr>
<td>3</td>
<td>SSL</td>
<td>5.81%</td>
<td>3</td>
<td>HTTP</td>
<td>11.11%</td>
<td>3</td>
<td>HTTP</td>
<td>10.66%</td>
</tr>
<tr>
<td>4</td>
<td>Netflix</td>
<td>5.38%</td>
<td>4</td>
<td>BitTorrent</td>
<td>5.57%</td>
<td>4</td>
<td>BitTorrent</td>
<td>9.23%</td>
</tr>
<tr>
<td>5</td>
<td>Skype</td>
<td>4.88%</td>
<td>5</td>
<td>MPEG</td>
<td>2.58%</td>
<td>5</td>
<td>SSL</td>
<td>2.39%</td>
</tr>
<tr>
<td>6</td>
<td>YouTube</td>
<td>3.71%</td>
<td>6</td>
<td>Hulu</td>
<td>2.41%</td>
<td>6</td>
<td>MPEG</td>
<td>2.30%</td>
</tr>
<tr>
<td>7</td>
<td>Facebook</td>
<td>1.71%</td>
<td>7</td>
<td>Facebook</td>
<td>1.90%</td>
<td>7</td>
<td>SSL</td>
<td>2.16%</td>
</tr>
<tr>
<td>8</td>
<td>Apple PhotoStream</td>
<td>1.34%</td>
<td>8</td>
<td>SSL</td>
<td>1.89%</td>
<td>8</td>
<td>iTunes</td>
<td>1.71%</td>
</tr>
<tr>
<td>9</td>
<td>Dropbox</td>
<td>1.21%</td>
<td>9</td>
<td>Flash Video</td>
<td>1.72%</td>
<td>9</td>
<td>Facebook</td>
<td>1.53%</td>
</tr>
<tr>
<td>10</td>
<td>Carbonite</td>
<td>0.99%</td>
<td>10</td>
<td>Facebook</td>
<td>1.48%</td>
<td>10</td>
<td>Facebook</td>
<td>1.52%</td>
</tr>
<tr>
<td></td>
<td><strong>Top 10</strong></td>
<td></td>
<td></td>
<td><strong>Top 10</strong></td>
<td></td>
<td></td>
<td><strong>Top 10</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 3 - Top 10 Peak Period Applications (North America, Fixed Access)
The Google Cloud Question

Why doesn’t Netflix use Google Cloud as well as AWS?
Google Cloud – Wait and See

Pro’s
• Cloud Native
• Huge scale for internal apps
• Exposing internal services
• Nice clean API model
• Starting a price war
• Fast for what it does
• Rapid start & minute billing

Con’s
• In beta until recently
• Few big customers yet
• Missing many key features
• Different arch model
• Missing billing options
• No SSD or huge instances
• Zone maintenance windows

But: Anyone interested is welcome to port NetflixOSS components to Google Cloud
Cloud Wars: Price and Performance

What Changed: Everyone using AWS or GCS gets the price cuts and performance improvements, as they happen. No need to switch vendor.

AWS vs. GCS War
- Faster newer systems
- Instance cost reduction
- Storage cost reduction

Private Cloud $$
- Maintenance renewal
- Aging system failures
- Labor cost increase
- Power cost increase

No Change: Locked in for three years.
The DIY Question

Why doesn’t Netflix build and run its own cloud?
Fitting Into Public Scale

1,000 Instances  100,000 Instances

Public    Grey Area    Private

Startups    Netflix    Facebook
How big is Public?

AWS Maximum Possible Instance Count 4.2 Million – May 2013
Growth >10x in Three Years, >2x Per Annum - http://bit.ly/awsiprange

AWS upper bound estimate based on the number of public IP Addresses
Every provisioned instance gets a public IP by default (some VPC don’t)
The Alternative Supplier Question

What if there is no clear leader for a feature, or AWS doesn’t have what we need?
Things We Don’t Use AWS For

SaaS Applications – Pagerduty, Appdynamics
Content Delivery Service
DNS Service
Content Delivery Service

Open Source Hardware Design + FreeBSD, bird, nginx
see openconnect.netflix.com

Open Connect Appliance Hardware

Objectives
When designing the Open Connect Appliance Hardware, we focused on these fundamental design goals:

- Very high storage density without sacrificing space and power efficiency. Our target was fitting 100 terabytes into a 4u chassis that is less than 2’ deep.
- High throughput: 10 Gbps throughput via an optical network connection.
- Very low field maintenance: the appliance must tolerate a variety of hardware failures including hard drives, network optics, and power supply units.
- Simple racking and installation. Front mounted power and network ports are the only things to connect at install time.

Open Connect Appliances are servers based on commodity PC components (similar to the model used by all large scale content delivery networks). We were influenced by the excellent write-ups from the Backblaze team, and use a custom chassis due to a lack of ready made options for a compact unit.

To achieve over 100 TB of storage, spinning hard drives provide the highest affordable density, in particular 36 TB SATA units. The hard drives are not hot swappable, as we wish to avoid the operational burden of field service. For lower power utilization and simpler sourcing we selected commodity units from two vendors and use software to manage failure modes and avoid field replacement. Dead drives reduce the total storage available for the system, but don’t take it offline. We also add 1 TB of flash storage (2 solid state drives) for system files, logs and popular content. To augment the motherboard attached controller, we use two 16 port LSI SAS controller cards that connect directly to the SATA drives. This avoids I/O bottlenecks of SATA multipliers or SAS expanders, and also reduces system complexity.

From a compute point of view, the system has modest requirements moving bits from the storage to network packets on the interface. To reduce the power usage and hence also cooling requirement (which in turn reduces vibration from case fans) we use a single low power 4 core Intel Sandy Bridge CPU on a small form factor Supermicro mATX board with the full 32 GB of RAM installed.

We use redundant, hot swappable power supply units that have interchangeable AC and DC options for maximum installation flexibility. Zippy reversed the fan rotation of the units to allow mounting at the front of the case, and thus allow network and power planes to be positioned here.

The network card has two 10 Gbps modules, which can power a variety of SR and LR optic modules, for installation flexibility and scalable interconnection.

For additional details, please download and read the full documentation from openconnect.netflix.com.
DNS Service

AWS Route53 is missing too many features (for now)
Multiple vendor strategy Dyn, Ultra, Route53
Abstracted (broken) DNS APIs with Denominator
What Changed?

Get out of the way of innovation
Best of breed, by the hour
Choices based on scale
Availability Questions

Is it running yet?
How many places is it running in?
How far apart are those places?
The STRANGE WORLD of the FUTURE

STRANDED without video!
No way to fill their empty hours!
They were victims of...

THE CLOUD OF BROKEN STREAMS

CREATED WITH PULP-O-MIZER COVER MAKER
Netflix Outages

• Running very fast with scissors
  – Mostly self inflicted – bugs, mistakes from pace of change
  – Some caused by AWS bugs and mistakes

• Incident Life-cycle Management by Platform Team
  – No runbooks, no operational changes by the SREs
  – Tools to identify what broke and call the right developer

• Next step is multi-region active/active
  – Investigating and building in stages during 2013
  – Could have prevented some of our 2012 outages
Real Web Server Dependencies Flow

(Netflix Home page business transaction as seen by AppDynamics)

Each icon is three to a few hundred instances across three AWS zones

Start Here

Cassandra

memcached

Web service

S3 bucket

Personalization movie group choosers
(for US, Canada and Latam)
Three Balanced Availability Zones

Test with Chaos Gorilla

- Zone A: Cassandra and Evcache Replicas
- Zone B: Cassandra and Evcache Replicas
- Zone C: Cassandra and Evcache Replicas
Isolated Regions

US-East Load Balancers

Cassandra Replicas

Zone A

Zone B

Zone C

EU-West Load Balancers

Cassandra Replicas

Zone A

Zone B

Zone C
Highly Available NoSQL Storage

A highly scalable, available and durable deployment pattern based on Apache Cassandra
Single Function Micro-Service Pattern

One keyspace, replaces a single table or materialized view

Many Different Single-Function REST Clients

Stateless Data Access REST Service
Astyanax Cassandra Client

Single function Cassandra Cluster Managed by Priam
Between 6 and 144 nodes

Over 50 Cassandra clusters
Over 1000 nodes
Over 30TB backup
Over 1M writes/s/cluster

Optional Datacenter Update Flow

Each icon represents a horizontally scaled service of three to hundreds of instances deployed over three availability zones.
Stateless Micro-Service Architecture

Linux Base AMI (CentOS or Ubuntu)

- Optional Apache frontend, memcached, non-java apps
- Monitoring
  - Log rotation to S3
  - AppDynamics machineagent Epic/Atlas

Java (JDK 6 or 7)

- AppDynamics appagent monitoring
- GC and thread dump logging

Tomcat

- Application war file, base servlet, platform, client interface jars, Astyanax
- Healthcheck, status servlets, JMX interface, Servo autoscale
Cassandra Instance Architecture

Linux Base AMI (CentOS or Ubuntu)

Tomcat and Priam on JDK
Healthcheck, Status

Java (JDK 7)

AppDynamics appagent monitoring
GC and thread dump logging

Cassandra Server

Local Ephemeral Disk Space – 2TB of SSD or 1.6TB disk holding Commit log and SSTables
Cassandra at Scale

Benchmarking to Retire Risk
Scalability from 48 to 288 nodes on AWS


Client Writes/s by node count – Replication Factor = 3

Used 288 of m1.xlarge
4 CPU, 15 GB RAM, 8 ECU
Cassandra 0.86
Benchmark config only existed for about 1hr

ONE MILLION WRITES/SEC in the CLOUD? That’s Fast.
APACHE CASSANDRA™ Fast.
2013 - Cross Region Use Cases

• Geographic Isolation
  – US to Europe replication of subscriber data
  – Read intensive, low update rate
  – Production use since late 2011

• Redundancy for regional failover
  – US East to US West replication of everything
  – Includes write intensive data, high update rate
  – Testing now
Benchmarking Global Cassandra

Write intensive test of cross region replication capacity
16 x hi1.4xlarge SSD nodes per zone = 96 total
192 TB of SSD in six locations up and running Cassandra in 20 min

Test Load

1 Million reads
After 500ms
CL.ONE with no
Data loss

US-West-2 Region - Oregon

Validation Load

1 Million writes
CL.ONE (wait for
one replica to ack)

US-East-1 Region - Virginia

Test Load

Inter-Zone Traffic

Inter-Region Traffic
Up to 9Gbits/s, 83ms

18TB backups from S3
Managing Multi-Region Availability

AWS Route53

DynECT DNS

Regional Load Balancers

Zone A
Cassandra Replicas

Zone B
Cassandra Replicas

Zone C
Cassandra Replicas

Regional Load Balancers

Zone A
Cassandra Replicas

Zone B
Cassandra Replicas

Zone C
Cassandra Replicas

Denominator – manage traffic via multiple DNS providers with Java code
2013 Timeline - Concept Jan, Code Feb, OSS March, Production use May
Incidents – Impact and Mitigation

- **PR** Incidents
  - XXX Incidents
    - No Impact – fast retry or automated failover
    - YYY Incidents
      - YYY Incidents mitigated by better data tagging
  - YYYY Incidents
    - Mitigated by better data tagging
  - YY Incidents
    - Mitigated by better tools and practices
  - Y Incidents
    - Mitigated by Active Active, game day practicing
- **CS** Incidents
  - XX Incidents
    - YXX Incidents
    - Mitigated by Active
    - GGXX Incidents
      - Mitigated by better tools and practices
- **Metrics impact – Feature disable**
  - XXX Incidents
    - Mitigated by feature disable
- **High Customer Service Calls**
  - XX Incidents
    - Mitigated by better tools and practices
- **Affects AB Test Results**
  - X Incidents
    - Mitigated by better tools and practices
- **Public Relations Media Impact**
  - PR Incidents
    - Mitigated by active active, game day practicing
Cloud Native Big Data

Size the cluster to the data
Size the cluster to the questions
Never wait for space or answers
Netflix Dataoven

From cloud Services
~100 Billion Events/day

From C*
Terabytes of Dimension data

Data Pipelines

S3

Data Warehouse
Over 2 Petabytes

RDS

Metadata

Gateways

Hadoop Clusters – AWS EMR

1300 nodes
800 nodes
Multiple 150 nodes Nightly

Tools

Java

Python

More?
Cloud Native Development Patterns

Master copies of data are cloud resident
Dynamically provisioned micro-services
Services are distributed and ephemeral
Datacenter to Cloud Transition Goals

• Faster
  – **Lower latency** than the equivalent datacenter web pages and API calls
  – Measured as mean and 99th percentile
  – For both first hit (e.g. home page) and in-session hits for the same user

• Scalable
  – **Avoid needing any more datacenter capacity** as subscriber count increases
  – No central vertically scaled databases
  – Leverage AWS elastic capacity effectively

• Available
  – Substantially **higher robustness and availability** than datacenter services
  – Leverage multiple AWS availability zones
  – No scheduled down time, no central database schema to change

• Productive
  – Optimize **agility** of a large development team with automation and tools
  – Leave behind complex tangled datacenter code base (~8 year old architecture)
  – Enforce clean layered interfaces and re-usable components
Datacenter Anti-Patterns

What do we currently do in the datacenter that prevents us from meeting our goals?
Rewrite from Scratch

Not everything is cloud specific
Pay down technical debt
Robust patterns
Netflix Datacenter vs. Cloud Arch

**Anti-Architecture**

- Central SQL Database vs. Distributed Key/Value NoSQL
- Sticky In-Memory Session vs. Shared Memcachced Session
- Chatty Protocols vs. Latency Tolerant Protocols
- Tangled Service Interfaces vs. Layered Service Interfaces
- Instrumented Code vs. Instrumented Service Patterns
- Fat Complex Objects vs. Lightweight Serializable Objects
- Components as Jar Files vs. Components as Services
Tangled Service Interfaces

• Datacenter implementation is exposed
  – Oracle SQL queries mixed into business logic

• Tangled code
  – Deep dependencies, false sharing

• Data providers with sideways dependencies
  – Everything depends on everything else

Anti-pattern affects productivity, availability
Untangled Service Interfaces

Two layers:

• SAL - Service Access Library
  – Basic serialization and error handling
  – REST or POJO’s defined by data provider

• ESL - Extended Service Library
  – Caching, conveniences, can combine several SALs
  – Exposes faceted type system (described later)
  – Interface defined by data consumer in many cases
Service Interaction Pattern
Sample Swimlane Diagram

First time request, new user, no cache hits, call cache service first, no need to notify

APPLICATION

request

MOVIES ESL

check local

LOCAL CACHE

lookup

not found

return miss

CACHE SAL

serialize

get key

CACHE SERVICE

lookup key

if miss

call remote cache

return miss

CACHE SERVICE

return not found

call service

MOVIES SAL

serialize
call service

SERVLET

deserialize

best effort

start timer

ENGINE

process request

result

return result

LOCAL CACHE

deserialize

response

return result

update local cache

store

return ok

CACHE SERVICE

store val at key

return ok

CACHE SAL

serialize

PUT key/value

return ok

stop
NetflixOSS Details

• Platform entities and services

• AWS Accounts and access management

• Upcoming and recent NetflixOSS components

• In-depth on NetflixOSS components
Basic Platform Entities

• AWS Based Entities
  – Instances and Machine Images, Elastic IP Addresses
  – Security Groups, Load Balancers, Autoscale Groups
  – Availability Zones and Geographic Regions

• NetflixOS Specific Entities
  – Applications (registered services)
  – Clusters (versioned Autoscale Groups for an App)
  – Properties (dynamic hierarchical configuration)
Core Platform Services

• AWS Based Services
  – S3 storage, to 5TB files, parallel multipart writes

• Netflix Based Services
  – EVCache – memcached based ephemeral cache
  – Cassandra – distributed persistent data store
Cloud Security

Fine grain security rather than perimeter
Leveraging AWS Scale to resist DDOS attacks
Automated attack surface monitoring and testing

http://www.slideshare.net/jason_chan/resilience-and-security-scale-lessons-learned
Security Architecture

• Instance Level Security baked into base AMI
  – Login: ssh only allowed via portal (not between instances)
  – Each app type runs as its own userid app{test|prod}

• AWS Security, Identity and Access Management
  – Each app has its own security group (firewall ports)
  – Fine grain user roles and resource ACLs

• Key Management
  – AWS Keys dynamically provisioned, easy updates
  – High grade app specific key management using HSM
AWS Accounts
Accounts Isolate Concerns

- **paastest** – for development and testing
  - Fully functional deployment of all services
  - Developer tagged “stacks” for separation

- **paasprod** – for production
  - Autoscale groups only, isolated instances are terminated
  - Alert routing, backups enabled by default

- **paasaudit** – for sensitive services
  - To support SOX, PCI, etc.
  - Extra access controls, auditing

- **paasarchive** – for disaster recovery
  - Long term archive of backups
  - Different region, perhaps different vendor
Reservations and Billing

- Consolidated Billing
  - Combine all accounts into one bill
  - Pooled capacity for bigger volume discounts
    http://docs.amazonwebservices.com/AWSConsolidatedBilling/1.0/AWSConsolidatedBillingGuide.html

- Reservations
  - Save up to 71%, priority when you request reserved capacity
  - Unused reservations are shared across accounts

- Cost Aware Cloud Architectures – with Jinesh Varia of AWS
  http://www.slideshare.net/AmazonWebServices/building-costaware-architectures-jinesh-varia-aws-and-adrian-cockroft-netflix
Cloud Access Control

developers

Cloud Access audit log ssh/sudo Gateway

WWW-prod
- Userid wwwprod
- Security groups don’t allow ssh between instances

Dal-prod
- Userid dalprod

Cass-prod
- Userid cassprod
Our perspiration...
A Cloud Native Open Source Platform
See netflix.github.com
Example Application – RSS Reader

Zuul Traffic Processing and Routing

Zuul

Eureka Service

Registrs Instances of Middle Tier

Configuration Properties

Discovers Instances of Middle Tier

Hystrx

Archaius

Eureka Client

RSS Edge Service

Blitz4j

Karion

RSS Publishers

RSS Middle Tier Service

Eureka Client

Archaius

Blitz4j

Karyon

Gather Metrics

Ribbon

Servo

Astyanax

Cassandra

Clients

Logs

Manages Server

Manages communication between Edge and Middle Tier

Netflix Open Source Components
Zuul Architecture

http://techblog.netflix.com/2013/06/announcing-zuul-edge-service-in-cloud.html
Ice – AWS Usage Tracking

http://techblog.netflix.com/2013/06/announcing-ice-cloud-spend-and-usage.html

AWS Usage Dashboard

[Image of AWS Usage Dashboard]

SHOW ALL

<table>
<thead>
<tr>
<th>Operation</th>
<th>Total</th>
<th>Max</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregated</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OnDemandInstances</td>
<td>$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReservedInstances</td>
<td>$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BorrowedInstances</td>
<td>$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UnusedInstances</td>
<td>$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UpfrontAmortized</td>
<td>$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Graph showing cost per hour]
NetflixOSS Continuous Build and Deployment

- Github NetflixOSS Source
- Maven Central
- AWS Base AMI
- Cloudbees Jenkins Aminator Bakery
- Dynaslake AWS Build Slaves
- AWS Baked AMIs
- Odin Orchestration API
- Asgard (+ Frigga) Console
- AWS Account
AWS Account

Multiple AWS Regions

3 AWS Zones

- Application Clusters
  - Autoscale Groups
  - Instances
- Priam
  - Cassandra
  - Persistent Storage
- Evcache
  - Memcached
  - Ephemeral Storage

- Eureka Registry
- Exhibitor
- Zookeeper
- Edda History
- Simian Army
- Zuul Traffic Mgr

- Asgard Console
- Archaius
  - Config Service
- Cross region Priam C*
- Pytheas
  - Dashboards
- Atlas
  - Monitoring
- Genie, Lipstick
  - Hadoop Services
- Ice
  - AWS Usage
  - Cost Monitoring

More...
Netflix OSS Instance Libraries

**Initialization**
- Baked AMI – Tomcat, Apache, your code
- Governator – Guice based dependency injection
- Archaius – dynamic configuration properties client
- Eureka - service registration client

**Service Requests**
- Karyon - Base Server for inbound requests
- RxJava – Reactive pattern
- Hystrix/Turbine – dependencies and real-time status
- Ribbon and Feign - REST Clients for outbound calls

**Data Access**
- Astyanax – Cassandra client and pattern library
- Evcache – Zone aware Memcached client
- Curator – Zookeeper patterns
- Denominator – DNS routing abstraction

**Logging**
- Blitz4j – non-blocking logging
- Servo – metrics export for autoscaling
- Atlas – high volume instrumentation
Netflix OSS Testing and Automation

**Test Tools**
- CassJmeter – Load testing for Cassandra
- Circus Monkey – Test account reservation rebalancing

**Maintenance**
- Janitor Monkey – Cleans up unused resources
- Efficiency Monkey
- Doctor Monkey
- Howler Monkey – Complains about AWS limits

**Availability**
- Chaos Monkey – Kills Instances
- Chaos Gorilla – Kills Availability Zones
- Chaos Kong – Kills Regions
- Latency Monkey – Latency and error injection

**Security**
- Conformity Monkey – architectural pattern warnings
- Security Monkey – security group and S3 bucket permissions
Your perspiration – deadline Sept 15th
Boosting the @NetflixOSS Ecosystem
See netflix.github.com
In 2012 Netflix Engineering won this..
We’d like to give out prizes too

But what for?
Contributions to NetflixOSS!
Shared under Apache license
Located on github
Judges choice award
Best example application mash-up
Best usability enhancement
Best portability enhancement
Best new monkey
Best new feature
Best datastore integration
Best contribution to code quality
Best contribution to operational tools
Best contribution to performance
How long do you have?
Entries open March 13th
Entries close September 15th
Six months...
Who can win?

Almost anyone, anywhere...
Except current or former Netflix or AWS employees
Who decides who wins?

Nominating Committee
Panel of Judges
What are Judges Looking For?

Eligible, Apache 2.0 licensed

Original and useful contribution to NetflixOSS

Code that successfully builds and passes a test suite

A large number of watchers, stars and forks on github

NetflixOSS project pull requests

Good code quality and structure

Documentation on how to build and run it

Evidence that code is in use by other projects, or is running in production
What do you win?

One winner in each of the 10 categories
Ticket and expenses to attend AWS Re:Invent 2013 in Las Vegas
A Trophy
$10,000 cash and $5,000 in AWS Credits
How do you enter?

Get a (free) github account
Fork github.com/netflix/cloud-prize
Send us your email address
Describe and build your entry

Twitter #cloudprize
Vendor Driven Portability
Interest in using Netflix OSS for Enterprise Private Clouds

“It’s done when it runs Asgard”
Functionally complete
Demonstrated March
Released June in V3.3

Offering $10K prize for integration work

Vendor and end user interest
Openstack “Heat” getting there
Paypal C3 Console based on Asgard
Takeaways

Cloud Native Manages Scale and Complexity at Speed

NetflixOSS makes it easier for everyone to become Cloud Native

http://netflix.github.com
http://techblog.netflix.com
http://slideshare.net/Netflix
http://www.linkedin.com/in/adriancockcroft

@adrianco #netflixcloud @NetflixOSS