Netflix and Open Source

April 2013

Adrian Cockcroft

@adrianco #netflixcloud @NetflixOSS

http://www.linkedin.com/in/adriancockcroft
Cloud Native

Netflix OSS – Cloud Native On-Ramp

Netflix Open Source Cloud Cloud Prize
We are Engineers

We solve hard problems
We build amazing and complex things
We fix things when they break
We strive for perfection

Perfect code
Perfect hardware
Perfectly operated
But perfection takes too long...

So we compromise
Time to market vs. Quality
Utopia remains out of reach
Where time to market wins big

Web services
Agile infrastructure - cloud
Continuous deployment
How Soon?

Code features in days instead of months
Hardware in minutes instead of weeks
Incident response in seconds instead of hours
Tipping the Balance

Utopia

Static
Better
Cheaper

Dystopia

Sooner
Dynamic
Broken
Inefficient
A new engineering challenge

Construct a highly agile and highly available service from ephemeral and often broken components
Cloud Native

How does Netflix work?
Netflix Member Web Site Home Page

Personalization Driven – How Does It Work?
How Netflix Streaming Works

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

Open Source Hardware Design + FreeBSD, bird, nginx

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 3TB 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 select 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 multiplexers 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 connects 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.

The firmware components were developed and first released with code of 2012.
### November 2012 Traffic

<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>36.8%</td>
<td>Netflix</td>
<td>33.0%</td>
<td>Netflix</td>
<td>28.8%</td>
</tr>
<tr>
<td>2</td>
<td>HTTP</td>
<td>9.83%</td>
<td>YouTube</td>
<td>14.8%</td>
<td>YouTube</td>
<td>13.1%</td>
</tr>
<tr>
<td>3</td>
<td>Skype</td>
<td>4.76%</td>
<td>HTTP</td>
<td>12.0%</td>
<td>HTTP</td>
<td>11.7%</td>
</tr>
<tr>
<td>4</td>
<td>Netflix</td>
<td>4.51%</td>
<td>BitTorrent</td>
<td>5.89%</td>
<td>BitTorrent</td>
<td>10.3%</td>
</tr>
<tr>
<td>5</td>
<td>SSL</td>
<td>3.73%</td>
<td>iTunes</td>
<td>3.92%</td>
<td>iTunes</td>
<td>3.43%</td>
</tr>
<tr>
<td>6</td>
<td>YouTube</td>
<td>2.70%</td>
<td>MPEG</td>
<td>2.22%</td>
<td>SSL</td>
<td>2.23%</td>
</tr>
<tr>
<td>7</td>
<td>PPStream</td>
<td>1.65%</td>
<td>Flash Video</td>
<td>2.21%</td>
<td>MPEG</td>
<td>2.05%</td>
</tr>
<tr>
<td>8</td>
<td>Facebook</td>
<td>1.62%</td>
<td>SSL</td>
<td>1.97%</td>
<td>Flash Video</td>
<td>2.01%</td>
</tr>
<tr>
<td>9</td>
<td>Apple PhotoStream</td>
<td>1.46%</td>
<td>Amazon Video</td>
<td>1.75%</td>
<td>Facebook</td>
<td>1.50%</td>
</tr>
<tr>
<td>10</td>
<td>Dropbox</td>
<td>1.17%</td>
<td>Facebook</td>
<td>1.48%</td>
<td>RTMP</td>
<td>1.41%</td>
</tr>
<tr>
<td></td>
<td><strong>Top 10</strong></td>
<td><strong>68.24%</strong></td>
<td><strong>Top 10</strong></td>
<td><strong>79.01%</strong></td>
<td><strong>Top 10</strong></td>
<td><strong>76.54%</strong></td>
</tr>
</tbody>
</table>

*Table 3 - Top 10 Peak Period Applications (North America, Fixed Access)*
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

Three Personalization movie group choosers (for US, Canada and Latam)
Cloud Native Architecture

Clients

Autoscaled Micro Services

Autoscaled Micro Services

Distributed Quorum NoSQL Datastores

Zone A  Zone B  Zone C

Things

JVM

Cassandra

Memcached
Non-Native Cloud Architecture

Agile Mobile Mammals

Cloudy Buffer

Datacenter Dinosaurs

iOS/Android

MySQL

Legacy Apps
New Anti-Fragile Patterns

Micro-services
Chaos engines
Highly available systems composed from ephemeral components
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

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

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)

Java (JDK 7)

Cassandra Server

- Local Ephemeral Disk Space – 2TB of SSD or 1.6TB disk holding Commit log and SSTables
- Monitoring AppDynamics machineagent Epic/Atlas
- Monitoring AppDynamics appagent monitoring
- GC and thread dump logging

Tomcat and Priam on JDK Healthcheck, Status
Cloud Native

Master copies of data are cloud resident
Everything is dynamically provisioned
All services are ephemeral
Dynamic Scalability
Asgard

Cloud Deployment Scalability

New Autoscaled AMI – zero to 500 instances from 21:38:52 - 21:46:32, 7m40s
Scaled up and down over a few days, total 2176 instance launches, m2.2xlarge (4 core 34GB)

Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
41.0    104.2    149.0    171.8    215.8    562.0
Ephemeral Instances

- Largest services are autoscaled
- Average lifetime of an instance is 36 hours
Leveraging Public Scale

1,000 Instances  100,000 Instances

Public  Grey Area  Private

Startups  Netflix  Google
How big is Public?

AWS Maximum Possible Instance Count 3.7 Million
Growth >10x in Three Years, >2x Per Annum

AWS upper bound estimate based on the number of public IP Addresses
Every provisioned instance gets a public IP by default
Availability

Is it running yet?
How many places is it running in?
How far apart are those places?
Antifragile API Patterns
Functional Reactive with Circuit Breakers and Bulkheads
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
Outages

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

• Next step is multi-region
  – Investigating and building in stages during 2013
  – Could have prevented some of our 2012 outages
Managing Multi-Region Availability

A portable way to manage multiple DNS providers from Java
Configuration State Management

Datacenter CMDB’s woeful Cloud native is the solution
Dependably complete
Edda – Configuration History


Edda

Eureka
Services metadata

AWS
Instances, ASGs, etc.

AppDynamics
Request flow

Monkeys
Edda Query Examples

Find any instances that have ever had a specific public IP address
$ curl "http://edda/api/v2/view/instances;publicIpAddress=1.2.3.4;_since=0"
["i-0123456789","i-012345678a","i-012345678b"]

Show the most recent change to a security group
$ curl "http://edda/api/v2/aws/securityGroups/sg-0123456789;_diff;_all;_limit=2"
--- /api/v2/aws.securityGroups/sg-0123456789;_pp;_at=1351040779810
+++ /api/v2/aws.securityGroups/sg-0123456789;_pp;_at=135104093504
@@ -1,33 +1,33 @@
{
   ...
   "ipRanges" : [
       "10.10.1.1/32",
       "10.10.1.2/32",
       +   "10.10.1.3/32",
       -   "10.10.1.4/32"
A Cloud Native Open Source Platform
Inspiration

- *Release It!* by Michael T. Nygard
- *Thinking in Systems: A Primer* by Donella H. Meadows
- *Antifragile: Things That Gain from Disorder* by Nassim Nicholas Taleb
- *Drift into Failure* by Sidney Dekker
- *Everything Is Obvious: *Once You Know the Answer* by Duncan J. Watts
Three Questions

Why is Netflix doing this?

How does it all fit together?

What is coming next?
Beware of Geeks Bearing Gifts: Strategies for an Increasingly Open Economy

Simon Wardley - Researcher at the Leading Edge Forum

- **Thinkers**
  - "It isn't impacting our industry yet"

- **Players**
  - "Open is a way of competing with others"

- **Chancers**
  - "It won't impact our industry"

- **Believers**
  - "Open by default"

Uses open to compete
How did Netflix get ahead?

Netflix Business + Developer Org
- Doing it right now
- SaaS Applications
- PaaS for agility
- Public IaaS for AWS features
- Big data in the cloud
- Integrating many APIs
- FOSS from github
- Renting hardware for 1hr
- Coding in Java/Groovy/Scala

Traditional IT Operations
- Taking their time
- Pilot private cloud projects
- Beta quality installations
- Small scale
- Integrating several vendors
- Paying big $ for software
- Paying big $ for consulting
- Buying hardware for 3yrs
- Hacking at scripts
Netflix ended up several years ahead of the industry, but it’s not a sustainable position.
Making it easy to follow

Exploring the wild west each time vs. laying down a shared route

[Image showing covered wagons, a poster, and a train track with people in period clothing.]
Establish our solutions as Best Practices / Standards

Hire, Retain and Engage Top Engineers

Build up Netflix Technology Brand

Benefit from a shared ecosystem

Goals
How does it all fit together?
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

- Asgard Console
- Archaius Config Service
- Cross region Priam C*
- Explorers Dashboards
- Atlas Monitoring
- Genie Hadoop Services
- Eureka Registry
- Exhibitor ZK
- Edda History
- Simian Army

3 AWS Zones

- Application Clusters Autoscale Groups Instances
- Priam Cassandra Persistent Storage
- Evcache Memcached Ephemeral Storage
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 - REST Client 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 expiring certs

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

Security
- Security Monkey
- Conformity Monkey
Example Application – RSS Reader
What’s Coming Next?

Better portability
Higher availability
Easier to deploy
Contributions from end users
Contributions from vendors

More Use Cases

More Features
Vendor Driven Portability
Interest in using NetflixOSS for Enterprise Private Clouds

“It’s done when it runs Asgard”
Functionally complete
Demonstrated March
Release 3.3 in 2Q13

Some vendor interest
Many missing features
Bait and switch AWS API strategy

Some vendor interest
Needs AWS compatible Autoscaler
AWS 2009
Baseline features needed to support Netflix OSS

Eucalyptus 3.3
Netflix Cloud Prize

Boosting the @NetflixOSS Ecosystem
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
Aino Corry
Program Chair for Qcon/GOTO

Simon Wardley
Strategist

Martin Fowler
Chief Scientist Thoughtworks

Werner Vogels
CTO Amazon

Joe Weinman
SVP Telx, Author “Cloudonomics”

Yury Izrailevsky
VP Cloud Netflix
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
Ten Prize Categories

- $10K cash
- $5K AWS
- AWS Re:Invent Tickets
- Trophy

Netflix Engineering

- Nominations
- Categories

Entrants

- Conforms to Rules
- Working Code
- Community Traction

Judges

- Winners

Register on Github by March 13

Contribute your licensed contributions on Github

Close entries by September 15

Award Ceremony Dinner at AWS Re:Invent in November

AWS Re:Invent Tickets for the Award Ceremony Dinner
Functionality and scale now, portability coming

Moving from parts to a platform in 2013

Netflix is fostering an ecosystem

Rapid Evolution - Low MTBIAMSH
(Mean Time Between Idea And Making Stuff Happen)
Takeaway

Netflix is making it easy for everyone to adopt Cloud Native patterns.

Open Source is not just the default, it’s a strategic weapon.

http://netflix.github.com
http://techblog.netflix.com
http://slideshare.net/Netflix

http://www.linkedin.com/in/adriancockcroft

@adrianco #netflixcloud @NetflixOSS