Globally Distributed Cloud Applications

Adrian Cockcroft
@adrianco

Netflix Inc.
Abstract

• Netflix on Cloud – What, Why and When

• Globally Distributed Availability Model

• Open Source Components

• Build Your Own PaaS
What Netflix Did

• Moved to SaaS
  – Corporate IT – OneLogin, Workday, Box, Evernote...
  – Tools – Pagerduty, AppDynamics, Elastic MapReduce

• Built our own PaaS
  – Customized to make our developers productive
  – When we started, we had little choice

• Moved incremental capacity to IaaS
  – No new datacenter space since 2008 as we grew
  – Moved our streaming apps to the cloud
Why Use Cloud?
Get stuck with wrong config
Wait Wait File tickets
Ask permission Wait Wait
Wait Run out of space/power
Plan capacity in advance
Wait

Things we don’t do

Have meetings with IT Wait
Keeping up with Developer Trends

- Big Data/Hadoop
- AWS Cloud
- Application Performance Management
- Integrated DevOps Practices
- Continuous Integration/Delivery
- NoSQL
- Platform as a Service; Fine grain SOA
- Social coding, open development/github

In production at Netflix

- 2009
- 2009
- 2010
- 2010
- 2010
- 2010
- 2011
AWS specific feature dependence....
Portability vs. Functionality

• Portability – the Operations focus
  – Avoid vendor lock-in
  – Support datacenter based use cases
  – Possible operations cost savings

• Functionality – the Developer focus
  – Less complex test and debug, one mature supplier
  – Faster time to market for your products
  – Possible developer cost savings
Functional PaaS

• IaaS base - all the features of AWS
  – Very large scale, mature, global, evolving rapidly
  – ELB, Autoscale, VPC, SQS, EIP, EMR, etc, etc.
  – Large files (TB) and multipart writes in S3

• Functional PaaS – Netflix added features
  – Very large scale, mature, flexible, customizable
  – Asgard console, Monkeys, Big data tools
  – Cassandra/Zookeeper data store automation
Developers choose Functional over Portable

Don’t let the roadie write the set list!

(yes you do need all those guitars on tour...)

Developers choose Functional over Portable
Freedom and Responsibility

• Developers leverage cloud to get freedom
  – Agility of a single organization, no silos

• But now developers are responsible
  – For compliance, performance, availability etc.

“As far as my rehab is concerned, it is within my ability to change and change for the better - Eddie Van Halen”
The Netflix Streaming Service

Coming soon to Denmark (and Norway, Sweden and Finland)
www.netflix.com in DK
Member Web Site
Content Delivery Service
Distributed storage nodes controlled by Netflix cloud services

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 packing 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 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 connectors 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.
Streaming Device API

Netflix Ready Devices
From: May 2008
To: May 2010

[Image of various streaming devices and APIs]
Major Front End Services

• Non-member Web Site www.netflix.com
  – Marketing driven, sign up flow, SOX/PCI scope

• Member Web Site movies.netflix.com
  – Personalization driven

• CDNs for delivering bulk video/audio
  – Netflix CDN: openconnect.netflix.com

• API for external and device user interfaces
  – Mostly private APIs, limited public API docs at developer.netflix.com

• API for controlling video playback
  – DRM, QoS management, Bookmarks
Component Services
(Simplified view using AppDynamics)
Web Server Dependencies Flow
(zooming in to one business transaction in test account)
One Request Snapshot
(captured because it was unusually slow)
Current Architectural Patterns for Availability

• Isolated Services
  – Resilient Business logic

• Three Balanced Availability Zones
  – Resilient to Infrastructure outage

• Triple Replicated Persistence
  – Durable distributed Storage

• Isolated Regions
  – US and EU don’t take each other down
Isolated Services
Test With Chaos Monkey, Latency Monkey
Three Balanced Availability Zones

Test with Chaos Gorilla

Load Balancers

Zone A: Cassandra and Evcache Replicas

Zone B: Cassandra and Evcache Replicas

Zone C: Cassandra and Evcache Replicas
Triple Replicated Persistence
Cassandra maintenance drops individual replicas

Load Balancers

Zone A
Cassandra and Evcache Replicas

Zone B
Cassandra and Evcache Replicas

Zone C
Cassandra and Evcache Replicas
Isolated Regions
## Failure Modes and Effects

<table>
<thead>
<tr>
<th>Failure Mode</th>
<th>Probability</th>
<th>Mitigation Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Failure</td>
<td>High</td>
<td>Automatic degraded response</td>
</tr>
<tr>
<td>AWS Region Failure</td>
<td>Low</td>
<td>Wait for region to recover</td>
</tr>
<tr>
<td>AWS Zone Failure</td>
<td>Medium</td>
<td>Continue to run on 2 out of 3 zones</td>
</tr>
<tr>
<td>Datacenter Failure</td>
<td>Medium</td>
<td>Migrate more functions to cloud</td>
</tr>
<tr>
<td>Data store failure</td>
<td>Low</td>
<td>Restore from S3 backups</td>
</tr>
<tr>
<td>S3 failure</td>
<td>Low</td>
<td>Restore from remote archive</td>
</tr>
</tbody>
</table>
Netflix Deployed on AWS

2009
- Content
  - Content Management
  - EC2 Encoding
  - S3 Petabytes
  - CDNs
  - ISPs
  - Terabits
  - Customers

2009
- Logs
  - S3 Terabytes
  - EMR
  - Hive & Pig
  - Business Intelligence

2010
- Play
  - DRAM
  - CDN routing
  - Bookmarks
  - Logging

2010
- WWW
  - Sign-Up
  - Search
  - Movie Choosing
  - Ratings

2010
- API
  - Metadata
  - Device Config
  - TV Movie Choosing
  - Social Facebook

2011
- CS
  - International CS lookup
  - Diagnostics & Actions
  - Customer Call Log
  - CS Analytics
Cloud Architecture Patterns

Where do we start?
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
Netflix Datacenter vs. Cloud Arch

**Anti-Architecture**

- Central SQL Database vs. Distributed Key/Value NoSQL
- Sticky In-Memory Session vs. Shared Memcached 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
Availability and Resilience
Chaos Monkey
http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html

- Computers (Datacenter or AWS) randomly die
  - Fact of life, but too infrequent to test resiliency
- Test to make sure systems are resilient
  - Allow any instance to fail without customer impact
- Chaos Monkey hours
  - Monday-Friday 9am-3pm random instance kill
- Application configuration option
  - Apps now have to opt-out from Chaos Monkey
Responsibility and Experience

• Make developers responsible for failures
  – Then they learn and write code that doesn’t fail
• Use Incident Reviews to find gaps to fix
  – Make sure its not about finding “who to blame”
• Keep timeouts short, fail fast
  – Don’t let cascading timeouts stack up
• Make configuration options dynamic
  – You don’t want to push code to tweak an option
Resilient Design – Circuit Breakers

http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html
Build Your Own PaaS
Components

- Continuous build framework turns code into AMIs
- AWS accounts for test, production, etc.
- Cloud access gateway
- Service registry
- Configuration properties service
- Persistence services
- Monitoring, alert forwarding
- Backups, archives
Common Build Framework

Extracted from
“Building and Deploying Netflix in the Cloud”
by @bmoyles and @garethbowles
On slideshare.net/netflix
Build Pipeline

Jenkins
- CBF steps
  - sync
  - check
  - build
  - test
  - publish
  - report
  - compile
  - resolve

Perforce

GitHub

Artifactory

yum

libraries

source
Jenkins Architecture

- Standard slave group
  - Amazon Linux m1.xlarge
- Custom slave group
  - Amazon Linux various

Single Master
- Red Hat Linux
- 2x quad core x86_64
- 26G RAM

Ad-hoc slaves
- misc. O/S & architectures
  - ~40 custom slaves
  - maintained by product teams

Netflix data center and office

us-west-1 VPC

 Netflix data center
Netflix Extensions to Jenkins

Job DSL plugin: allow jobs to be set up with minimal definition, using templates and a Groovy-based DSL (Gradle)

Housekeeping and maintenance processes implemented as Jenkins jobs, system Groovy scripts
The Bakery

• Create base AMIs
  – We have CentOS, Ubuntu and Windows base AMIs
  – All the generic code, apache, tomcat etc.
  – Standard system and application monitoring tools
  – Update ~monthly with patches and new versions

• Add yummy topping and bake
  – Jenkins builds app specific AMI including all code etc.
  – Bakery mounts EBS snapshot, installs and bakes
  – One bakery per region, delivers into paastest
  – Tweak config and publish AMI to paasprod
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% on your baseline load
  – Priority when you request reserved capacity
  – Unused reservations are shared across accounts
Cloud Access Control

developers

Cloud Access ssh Gateway

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

Dal-prod
  - Userid dalprod

Cass-prod
  - Userid cassprod
Now Add Code

Netflix has open sourced a lot of what you need, more is on the way...
Netflix Open Source Strategy

• Release PaaS Components git-by-git
  – Source at github.com/netflix – we build from it...
  – Intros and techniques at techblog.netflix.com
  – Blog post or new code every few weeks

• Motivations
  – Give back to Apache licensed OSS community
  – Motivate, retain, hire top engineers
  – “Peer pressure” code cleanup, external contributions
http://netflix.github.com
Instance creation

- Image baked
- ASG / Instance started
- Instance Running

- Bakery & Build tools
- Base AMI
- Application Code
- Asgard
- Odin
- Autoscaling scripts
Application Launch

Application initializing

Registering, configuration

Governator (Guice)
Async logging
Servo
Eureka
Archaius
Entrypoints

Netflix
RunLme
Managing service
Resiliency aids

Priam
Exhibitor
Explorers
NIWS LB

Chaos Monkey
Latency Monkey
Janitor Monkey
Cass JMeter

Calling other services
Dependency Command
REST client

Curator
Astyanax

Latency Monkey
Janitor Monkey
Cass JMeter

Cassandra

Runtime

NETFLIX
Asgard

Asgard

• Runs in a VM in our datacenter
  – So it can deploy to an empty account
  – Groovy/Grails/JVM based
  – Supports all AWS regions on a global basis

• Hides the AWS credentials
  – Use AWS IAM to issue restricted keys for Asgard
  – Each Asgard instance manages one account
  – One install each for paastest, paasprod, paasaudit
Discovery and Coordination

• Eureka – Service Directory
  – Map an instance to a service in a private namespace
  – Load balance over clusters of instances
  – Foundation service, first to deploy

• Highly available distributed coordination
  – Netflix Curator for common Zookeeper usage patterns
  – Netflix Exhibitor manages Zookeeper reliably

http://techblog.netflix.com/2012/09/eureka.html
http://techblog.netflix.com/2012/04/introducing-exhibitor-supervisor-system.html
Archaius Dynamic Properties Service
http://techblog.netflix.com/2012/06/announcing-archaius-dynamic-properties.html

• Dynamic hierarchical & propagates in seconds
  – Client timeouts, feature set enables
  – Region specific service endpoints
  – etc. etc.

• Used to configure everything
  – So everything depends on it...
  – Pluggable backend storage interface
  – Files, SimpleDB, Cassandra
Roadmap for 2012

• More resiliency and improved availability
• More automation, orchestration
• “Hardening” the platform, code clean-up
• Lower latency for web services and devices
• IPv6 – now running in prod, rollout in process
• More open sourced components
• See you at AWS Re:Invent in November...
Takeaway

Netflix has built and deployed a scalable global Platform as a Service.

Key components of the Netflix PaaS are being released as Open Source projects so you can build your own custom PaaS.

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

http://www.linkedin.com/in/adriancokcroft
http://www.linkedin.com/in/ruslanmeshenberg

@adriancokcroft @ruslanmeshenberg #netflixcloud
Amazon Cloud Terminology Reference

See http://aws.amazon.com/ This is not a full list of Amazon Web Service features

- AWS – Amazon Web Services (common name for Amazon cloud)
- AMI – Amazon Machine Image (archived boot disk, Linux, Windows etc. plus application code)
- EC2 – Elastic Compute Cloud
  - Range of virtual machine types m1, m2, c1, cc, cg. Varying memory, CPU and disk configurations.
  - Instance – a running computer system. Ephemeral, when it is de-allocated nothing is kept.
  - Reserved Instances – pre-paid to reduce cost for long term usage
  - Availability Zone – datacenter with own power and cooling hosting cloud instances
- ASG – Auto Scaling Group (instances booting from the same AMI)
- S3 – Simple Storage Service (http access)
- EBS – Elastic Block Storage (network disk filesystem can be mounted on an instance)
- RDS – Relational Database Service (managed MySQL master and slaves)
- DynamoDB/SDB – Simple Data Base (hosted http based NoSQL datastore, DynamoDB replaces SDB)
- SQS – Simple Queue Service (http based message queue)
- SNS – Simple Notification Service (http and email based topics and messages)
- EMR – Elastic Map Reduce (automatically managed Hadoop cluster)
- ELB – Elastic Load Balancer
- EIP – Elastic IP (stable IP address mapping assigned to instance or ELB)
- VPC – Virtual Private Cloud (single tenant, more flexible network and security constructs)
- DirectConnect – secure pipe from AWS VPC to external datacenter
- IAM – Identity and Access Management (fine grain role based security keys)