Cloud Architecture Tutorial

Constructing Cloud Architecture the Netflix Way

Gluecon May 23rd, 2012

Adrian Cockcroft

@adrianco #netflixcloud

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

Cloud Architecture Tutorial

Rocking Gluecon the Van Halen Way

Gluecon May 23rd, 2012

Adrian Cockcroft

@adrianco #netflixcloud

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

Tutorial Abstract – Set Context

- Dispensing with the usual questions: “Why Netflix, why cloud, why AWS?” as they are old hat now.
- This tutorial explains how developers use the Netflix cloud, and how it is built and operated.
- The real meat of the tutorial comes when we look at how to construct an application with a host of important properties: elastic, dynamic, scalable, agile, fast, cheap, robust, durable, observable, secure. Over the last three years Netflix has figured out cloud based solutions with these properties, deployed them globally at large scale and refined them into a global Java oriented Platform as a Service. The PaaS is based on low cost open source building blocks such as Apache Tomcat, Apache Cassandra, and Memcached. Components of this platform are in the process of being open-sourced by Netflix, so that other companies can get a start on building their own customized PaaS that leverages advanced features of AWS and supports rapid agile development.
- The architecture is described in terms of anti-patterns - things to avoid in the datacenter to cloud transition. A scalable global persistence tier based on Cassandra provides a highly available and durable under-pinning. Lessons learned will cover solutions to common problems, availability and robustness, observability. Attendees should leave the tutorial with a clear understanding of what is different about the Netflix cloud architecture, how it empowers and supports developers, and a set of flexible and scalable open source building blocks that can be used to construct their own cloud platform.

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”

Cloud Tutorial Sections

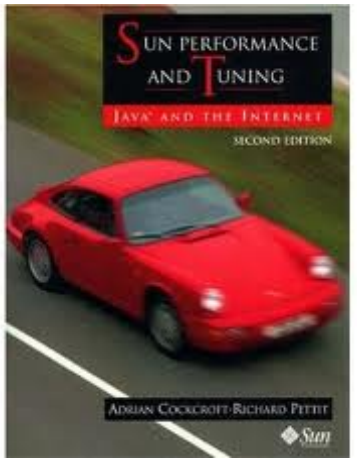
Intro: Who are you, what are your questions?

Part 1 – Writing and Performing
- Developer Viewpoint

Part 2 – Running the Show
- Operator Viewpoint

Part 3 – Making the Instruments
- Builder Viewpoint

Adrian Cockcroft



- Director, Architecture for Cloud Systems, Netflix Inc.
  - Previously Director for Personalization Platform
- Distinguished Availability Engineer, eBay Inc. 2004-7
  - Founding member of eBay Research Labs
- Distinguished Engineer, Sun Microsystems Inc. 1988-2004
  - 2003-4 Chief Architect High Performance Technical Computing
  - 2001 Author: *Capacity Planning for Web Services*
  - 1999 Author: *Resource Management*
  - 1995 & 1998 Author: *Sun Performance and Tuning*
  - 1996 Japanese Edition of *Sun Performance and Tuning*
    - SPARC & Solarisパフォーマンスチューニング (サンソフトプレスシリーズ)
- Heavy Metal Bass Guitarist in "Black Tiger" 1980-1982
  - Influenced by Van Halen, Yesterday & Today, AC/DC
- More
  - Twitter @adrianco – Blog http://perfcap.blogspot.com
  - Presentations at http://www.slideshare.net/adrianco

NETFLIX

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?

NETFLIX

Writing and Performing
Developer Viewpoint

Part 1 of 3

Van Halen

Audience and Fans

Listen to Songs and Albums

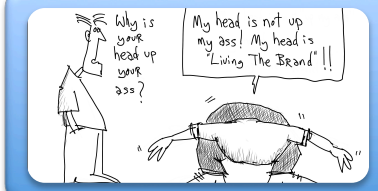
Written and Played by Van Halen

Using Instruments and Studios

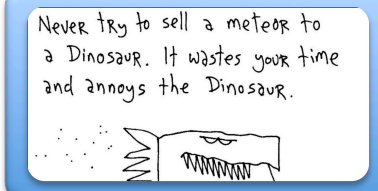
NETFLIX

Developers

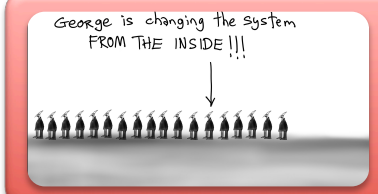
Toons from gapingvoid.com



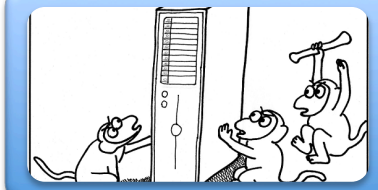
Customers



Use Products



Built by Developers



That run on Infrastructure

NETFLIX

Why Use Cloud?

“Runnin’ with the Devil – Van Halen”

Get stuck with wrong config

Wait

Wait

File tickets

Ask permission

Wait

Wait

Things we don't do

"Unchained – Van Halen"

Wait

Wait

Run out of space/power

Plan capacity in advance

Have meetings with IT

Wait

NETFLIX

What do developers care about?

“Right Now – Van Halen”

Keeping up with Van Halen's Singer

	When?
• David Lee Roth	1973
• Sammy Hagar	1985
• David Lee Roth (again briefly)	1996
• Gary Charone (who?)	1996
• Sammy Hagar (again)	2003
• David Lee Roth (finally…)	2006

Keeping up with Developer Trends

	In production at Netflix
• Big Data/Hadoop	2009
• Cloud	2009
• Application Performance Management	2010
• Integrated DevOps Practices	2010
• Continuous Integration/Delivery	2010
• NoSQL	2010
• Platform as a Service	2010
• Social coding, open development/github	2011

AWS specific feature dependence….

“Why can’t this be love? – Van Halen”

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

NETFLIX

Portable PaaS

- Portable IaaS Base - some AWS compatibility
  - Eucalyptus – AWS licensed compatible subset
  - CloudStack – Citrix Apache project
  - OpenStack – Rackspace, Cloudscaling, HP etc.

- Portable PaaS
  - Cloud Foundry - run it yourself in your DC
  - AppFog and Stackato – Cloud Foundry/Openstack
  - Vendor options: Rightscale, Enstratus, Smartscale

NETFLIX

Functional PaaS

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

- Functional PaaS – based on Netflix features
  - Very large scale, mature, flexible, customizable
  - Asgard console, Monkeys, Big data tools
  - Cassandra/Zookeeper data store automation

Developers choose Functional

Don't let the roadie write the set list!
(yes you do need all those guitars on tour…)

NETFLIX

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"

NETFLIX

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
  - Region – group of Avail Zones – US-East, US-West, EU-Eire, Asia-Singapore, Asia-Japan, SA-Brazil, US-Gov
- 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)

Netflix Deployed on AWS

2009	2009	2010	2010	2010	2011
Content	Logs	Play	WWW	API	CS
Content Management	S3 Terabytes	DRM	Sign-Up	Metadata	International CS lookup
EC2 Encoding	EMR	CDN routing	Search	Device Config	Diagnostics & Actions
S3 Petabytes	Hive & Pig	Bookmarks	Movie Choosing	TV Movie Choosing	Customer Call Log
	Business Intelligence	Logging	Ratings	Social Facebook	CS Analytics
	hadoop	Cassandra	MySQL	Cassandra	Cassandra

CDNs
ISPs
Terabits
Customers

NETFLIX

Datacenter to Cloud Transition Goals

"Go ahead and Jump – Van Halen"

- 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 Anti-Patterns

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

“Me Wise Magic – Van Halen”

Netflix Datacenter vs. Cloud Arch

Anti-Architecture

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

The Central SQL Database

- Datacenter has a central database
  - Everything in one place is convenient until it fails
- Schema changes require downtime
  - Customers, movies, history, configuration

Anti-pattern impacts scalability, availability

NETFLIX

The Distributed Key-Value Store

- Cloud has many key-value data stores
  - More complex to keep track of, do backups etc.
  - Each store is much simpler to administer
  - Joins take place in java code
  - No schema to change, no scheduled downtime

- Minimum Latency for Simple Requests
  - Memcached is dominated by network latency <1ms
  - Cassandra cross zone replication around one millisecond
  - DynamoDB replication and auth overheads around 5ms
  - SimpleDB higher replication and auth overhead >10ms

NETFLIX

The Sticky Session

- Datacenter Sticky Load Balancing
  - Efficient caching for low latency
  - Tricky session handling code

- Encourages concentrated functionality
  - one service that does everything
  - Middle tier load balancer had issues in practice

Anti-pattern impacts productivity, availability

NETFLIX

Shared Session State

- Elastic Load Balancer
  - We don't use the cookie based routing option
  - External "session caching" with memcached
- More flexible fine grain services
  - Any instance can serve any request
  - Works better with auto-scaled instance counts

NETFLIX

Chatty Opaque and Brittle Protocols

- Datacenter service protocols
  - Assumed low latency for many simple requests
- Based on serializing existing java objects
  - Inefficient formats
  - Incompatible when definitions change

Anti-pattern causes productivity, latency and availability issues

NETFLIX

Robust and Flexible Protocols

- Cloud service protocols
  - JSR311/Jersey is used for REST/HTTP service calls
  - Custom client code includes service discovery
  - Support complex data types in a single request
- Apache Avro
  - Evolved from Protocol Buffers and Thrift
  - Includes JSON header defining key/value protocol
  - Avro serialization is **half the size** and several times faster than Java serialization, more work to code

Persisted Protocols

- Persist Avro in Memcached
  - Save space/latency (zigzag encoding, half the size)
  - New keys are ignored
  - Missing keys are handled cleanly
- Avro protocol definitions
  - Less brittle across versions
  - Can be written in JSON or generated from POJOs
  - It's hard, needs better tooling

NETFLIX

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

- New Cloud Code With Strict Layering
  - Compile against interface jar
  - Can use spring runtime binding to enforce
  - Fine grain services as components

- Service interface **is** the service
  - Implementation is completely hidden
  - Can be implemented locally or remotely
  - Implementation can evolve independently

Untangled Service Interfaces

Poundcake – Van Halen

Two layers:

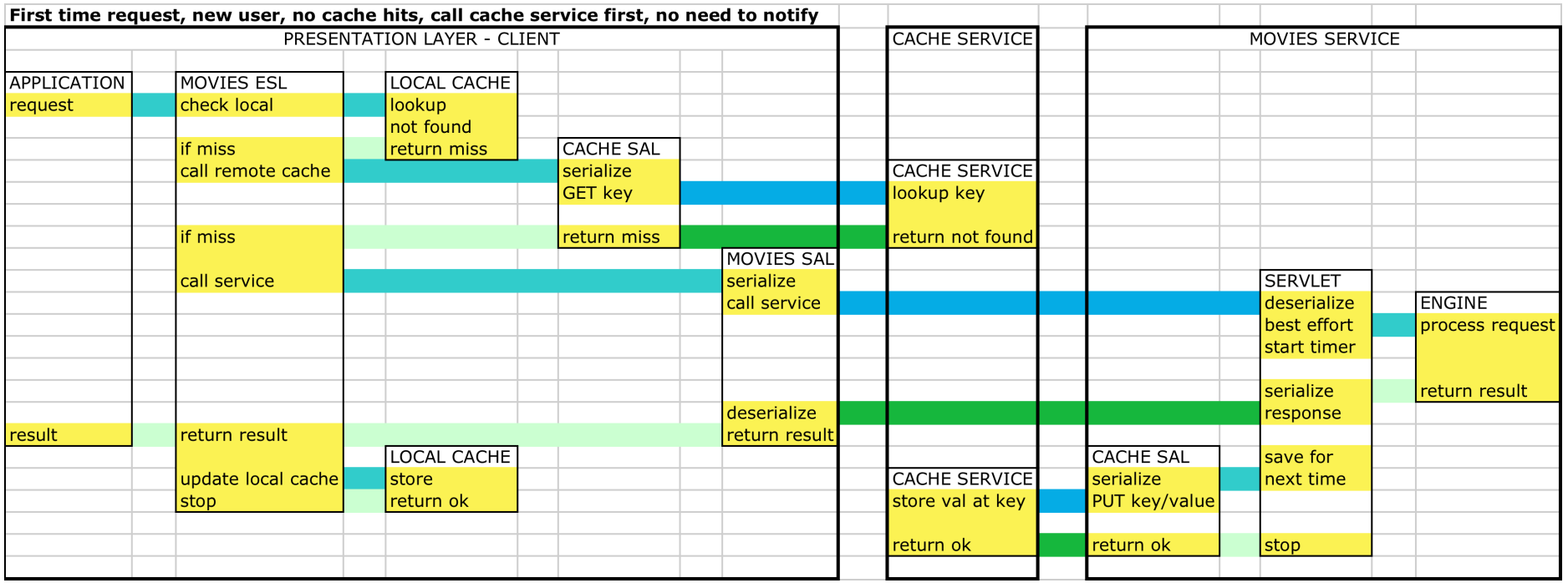
- SAL - Service Access Library
  - Basic serialization and error handling
  - REST or POJO's defined by data provider

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

NETFLIX

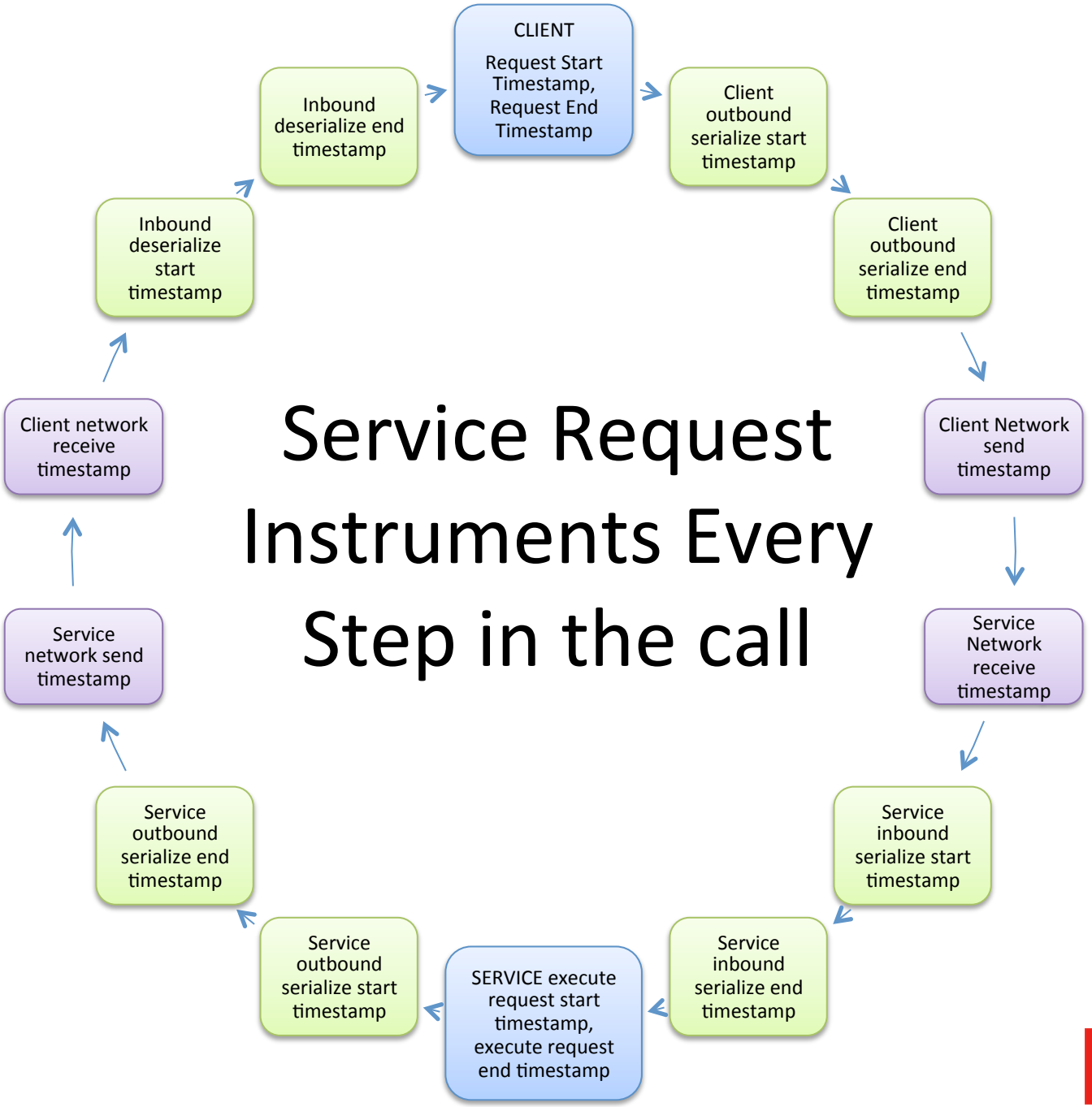
Service Interaction Pattern

Sample Swimlane Diagram



Service Architecture Patterns

- Internal Interfaces Between Services
  - Common patterns as templates
  - Highly instrumented, observable, analytics
  - Service Level Agreements – SLAs
- Library templates for generic features
  - Instrumented Netflix Base Servlet template
  - Instrumented generic client interface template
  - Instrumented S3, SimpleDB, Memcached clients


CLIENT
Request Start Timestamp, Request End Timestamp
Client outbound serialize start timestamp
Client outbound serialize end timestamp
Client Network send timestamp
Service Network receive timestamp
Service inbound serialize start timestamp
Service inbound serialize end timestamp
SERVICE execute request start timestamp, execute request end timestamp
Service outbound serialize start timestamp
Service outbound serialize end timestamp
Service network send timestamp
Client network receive timestamp
Inbound deserialize start timestamp
Inbound deserialize end timestamp
Service Request Instruments Every Step in the call

NETFLIX

Boundary Interfaces

- Isolate teams from external dependencies
  - Fake SAL built by cloud team
  - Real SAL provided by data provider team later
  - ESL built by cloud team using faceted objects
- Fake data sources allow development to start
  - e.g. Fake Identity SAL for a test set of customers
  - Development solidifies dependencies early
  - Helps external team provide the right interface

NETFLIX

One Object That Does Everything

Can't Get This Stuff No More – Van Halen

- Datacenter uses a few big complex objects
  - Good choice for a small team and one instance
  - Problematic for large teams and many instances

- False sharing causes tangled dependencies
  - Movie and Customer objects are foundational
  - Unproductive re-integration work

Anti-pattern impacting productivity and availability

NETFLIX

An Interface For Each Component

- Cloud uses faceted Video and Visitor
  - Basic types hold only the identifier
  - Facets scope the interface you actually need
  - Each component can define its own facets
- No false-sharing and dependency chains
  - Type manager converts between facets as needed
  - video.asA(PresentationVideo) for www
  - video.asA(MerchableVideo) for middle tier

Stan Lanning's Soap Box

Listen to the bearded guru…

- Business Level Object - Level Confusion
  - Don't pass around IDs when you mean to refer to the BLO
- Using Basic Types helps the compiler help you
  - Compile time problems are better than run time problems
- More readable by people
  - But beware that asA operations may be a lot of work
- Multiple-inheritance for Java?
  - Kinda-sorta…

NETFLIX

Model Driven Architecture

- Traditional Datacenter Practices
  - Lots of unique hand-tweaked systems
  - Hard to enforce patterns
  - Some use of Puppet to automate changes

- Model Driven Cloud Architecture
  - Perforce/Ivy/Jenkins based builds for *everything*
  - Every production instance is a pre-baked AMI
  - Every application is managed by an Autoscaler

Every change is a new AMI

NETFLIX

Netflix PaaS Principles

- Maximum Functionality
  - Developer productivity and agility
- Leverage as much of AWS as possible
  - AWS is making huge investments in features/scale
- Interfaces that isolate Apps from AWS
  - Avoid lock-in to specific AWS API details
- Portability is a long term goal
  - Gets easier as other vendors catch up with AWS

Netflix Global PaaS Features

- Supports all AWS Availability Zones *and* Regions
- Supports multiple AWS accounts {test, prod, etc.}
- Cross Region/Acct Data Replication and Archiving
- Internationalized, Localized and GeoIP routing
- Security is fine grain, dynamic AWS keys
- Autoscaling to thousands of instances
- Monitoring for millions of metrics
- Productive for 100s of developers on one product
- 25M+ users USA, Canada, Latin America, UK, Eire

Basic PaaS Entities

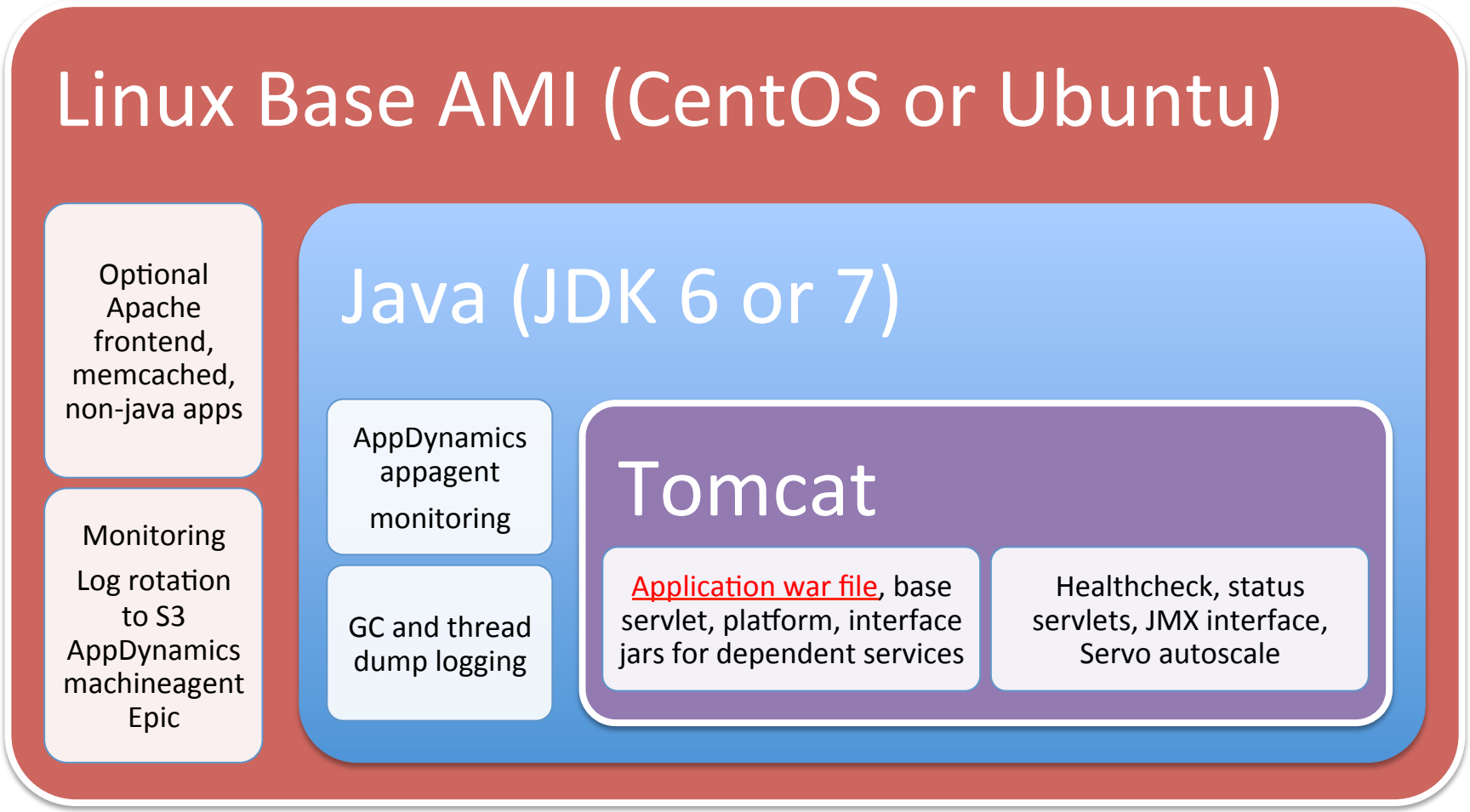
- AWS Based Entities
  - Instances and Machine Images, Elastic IP Addresses
  - Security Groups, Load Balancers, Autoscale Groups
  - Availability Zones and Geographic Regions

- Netflix PaaS Entities
  - Applications (registered services)
  - Clusters (versioned Autoscale Groups for an App)
  - Properties (dynamic hierarchical configuration)

Core PaaS Services

- AWS Based Services
  - S3 storage, to 5TB files, parallel multipart writes
  - SQS – Simple Queue Service. Messaging layer.
- Netflix Based Services
  - EVCache – memcached based ephemeral cache
  - Cassandra – distributed persistent data store
- External Services
  - GeoIP Lookup interfaced to a vendor
  - Secure Keystore HSM

Instance Architecture



NETFLIX

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 support

NETFLIX

Continuous Integration / Release

Lightweight process scales as the organization grows

- No centralized two-week sprint/release “train”
- Thousands of builds a day, tens of releases
- Engineers release at their own pace
- Unit of release is a web service, over 200 so far…
- Dependencies handled as exceptions

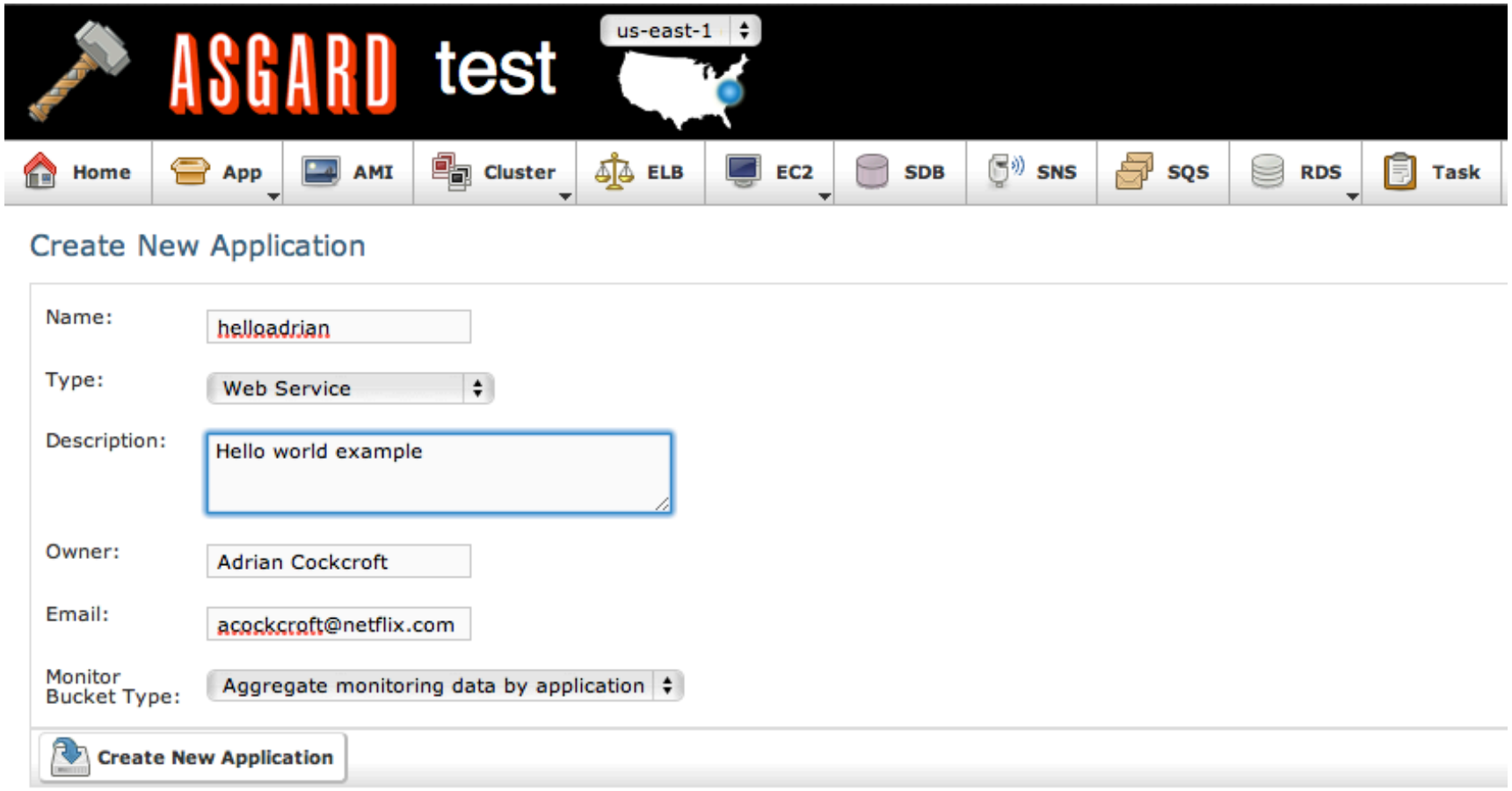
Hello World?

Getting started for a new developer…

- Register the “helloadrian” app name in Asgard
- Get the example helloworld code from perforce
- Edit some properties to update the name etc.
- Check-in the changes
- Clone a Jenkins build job
- Build the code
- Bake the code into an Amazon Machine Image
- Use Asgard to setup an AutoScaleGroup with the AMI
- Check instance healthcheck is “Up” using Asgard
- Hit the URL to get “HTTP 200, Hello” back

Register new application name

naming rules: all lower case with underscore, no spaces or dashes



NETFLIX

Application Details in us-east-1 (Virginia)

Edit Application | Delete Application | Edit Application Security Access

Name:	helloadrian
Type:	Web Service
Description:	Hello world example
Owner:	Adrian Cockcroft
Email:	acockcroft@netflix.com
Monitor Bucket Type:	Aggregate monitoring data by application
Create Time:	2012-05-18 14:54:51 PDT
Update Time:	2012-05-18 14:54:51 PDT

Pattern Matches in us-east-1 (Virginia)

Clusters:	
Auto Scaling:	
Load Balancers:	
Security Groups:	
Launch Configurations:	
Instances:	Running Instance List
Images:	Image List

NETFLIX

Create New Auto Scaling Group

Name

Optional variables let you customize:

- Environment variables
- Request routing
- Selection of fast property values

Name Preview

helloadrian

(Required) Application	Stack	Free-form details	Countries	Dev phase	Hardware devices	Partners	Revision
helloadrian	-Stack-		Ex: latam	Ex: stage	Ex: phones	Ex: sony	Ex: 27
Filter	Filter						
	New Stack						

Auto Scaling

Load Balancers (cannot be added or removed later):

Filter

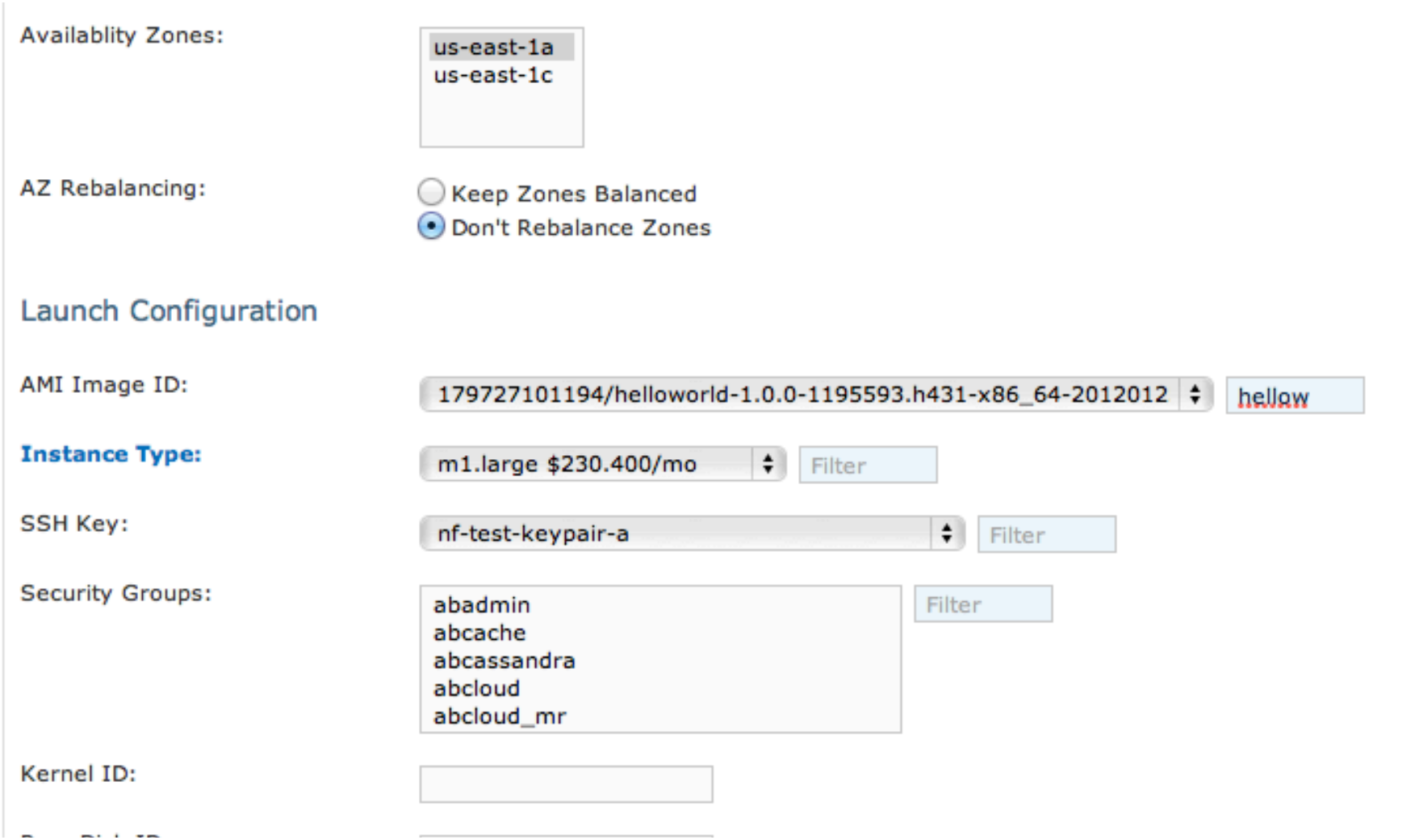
- abadmin
- accountweb-dev-frontend
- accountweb-release-frontend
- accountweb-test-frontend
- api-alttest-be
- api-alttest-latam
- api-bivl-staging-be
- api-ci-dev-backend
- api-ci-test-backend
- api-dev-be

Instance Counts: Min: 1 Desired: 1 Max: 1

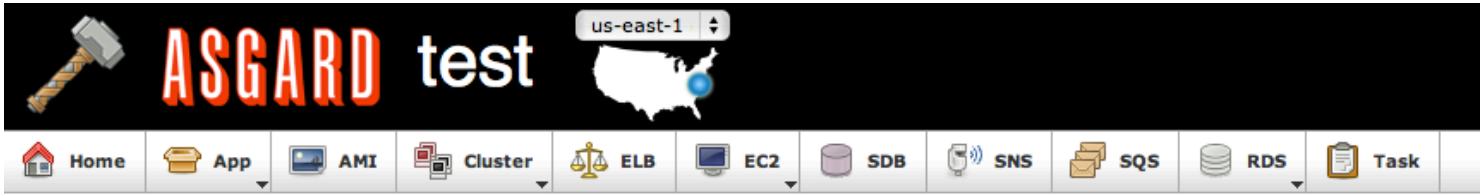
Cooldown: 10 seconds

ASG Health Check Type: EC2 (Replace terminated instances)

NETFLIX


Availablity Zones:
us-east-1a
us-east-1c
AZ Rebalancing:
Keep Zones Balanced
Don't Rebalance Zones
Launch Configuration
AMI Image ID:
179727101194/helloworld-1.0.0-1195593.h431-x86_64-2012012
hellow
Instance Type:
m1.large $230.400/mo
Filter
SSH Key:
nf-test-keypair-a
Filter
Security Groups:
abadmin
abcache
abcassandra
abcloud
abcloud_mr
Filter
Kernel ID:

NETFLIX



Auto Scaling Group Details

Launch Config 'helloadrian-20120518173623' has been created. Auto Scaling Group 'helloadrian' has been created.

Edit Auto Scaling Group | Delete Auto Scaling Group | Prepare Rolling Push | Manage Cluster of Sequential ASGs

Auto Scaling Group:	helloadrian
Cluster:	helloadrian
Application:	helloadrian
Instance Counts:	Min 1 Desired 1 Max 1
Cooldown:	10 seconds
ASG Health Check Type:	EC2 (Replace terminated instances)
ASG Health Check Grace Period:	600 seconds
Availablility Zones:	us-east-1a
Tags:	–
AZ Rebalancing:	Disabled: 'User suspended at 2012-05-19T00:36:25Z'
New Instance Launching:	Enabled
Instance Terminating:	Enabled
Adding to Load Balancer:	Enabled
Created Time:	2012-05-18 17:36:25 PDT
Load Balancers:	–
Launch Configuration:	helloadrian-20120518173623
Image:	ami-0969bf60 | x86_64 | 179727101194/helloworld-1.0.0-1195593.h431-x86_64-20120125215845-ebs
SSH Key:	nf-test-keypair-a
Security Groups:	[nf-datacenter, nf-infrastructure]
User Data:	#!/bin/bash # udf0 begin

NETFLIX

Instance Type: m1.large

Auto Scaling Activities:

At 2012-05-22T18:40:51Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 1. : Launching a new EC2 instance: i-c72173a1 (0% done) (Status: Successful)

At 2012-05-22T18:40:21Z an instance was taken out of service in response to a system health-check. : Terminating EC2 instance: i-c52f5ca3 (0% done) (Status: Successful)

At 2012-05-18T19:28:27Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 1. : Launching a new EC2 instance: i-c52f5ca3 (0% done) (Status: Successful)

20 activities shown.

Click to show activities in a table:

100 activities
1,000 activities
Unlimited activities

Scaling Policies

Total Policies: 0 | **Create New Scaling Policy**

Instances

Total Instances: 1 | **Run All Health Checks** | **Terminate Instances**

Load Balancing: **Deregister Instances from ELBs** | **Register Instances with group's ELBs**

Discovery: **Deactivate in Discovery** | **Activate in Discovery**

Instance	Zone	State	Launch Time	Package	Ver	CL	Build	ELBs	Discovery	Health
i-c72173a1	us-east-1a	InService	2012-05-22 11:40:52 PDT	hellobmundlapudi	1.0.0	1213877	1		UP	

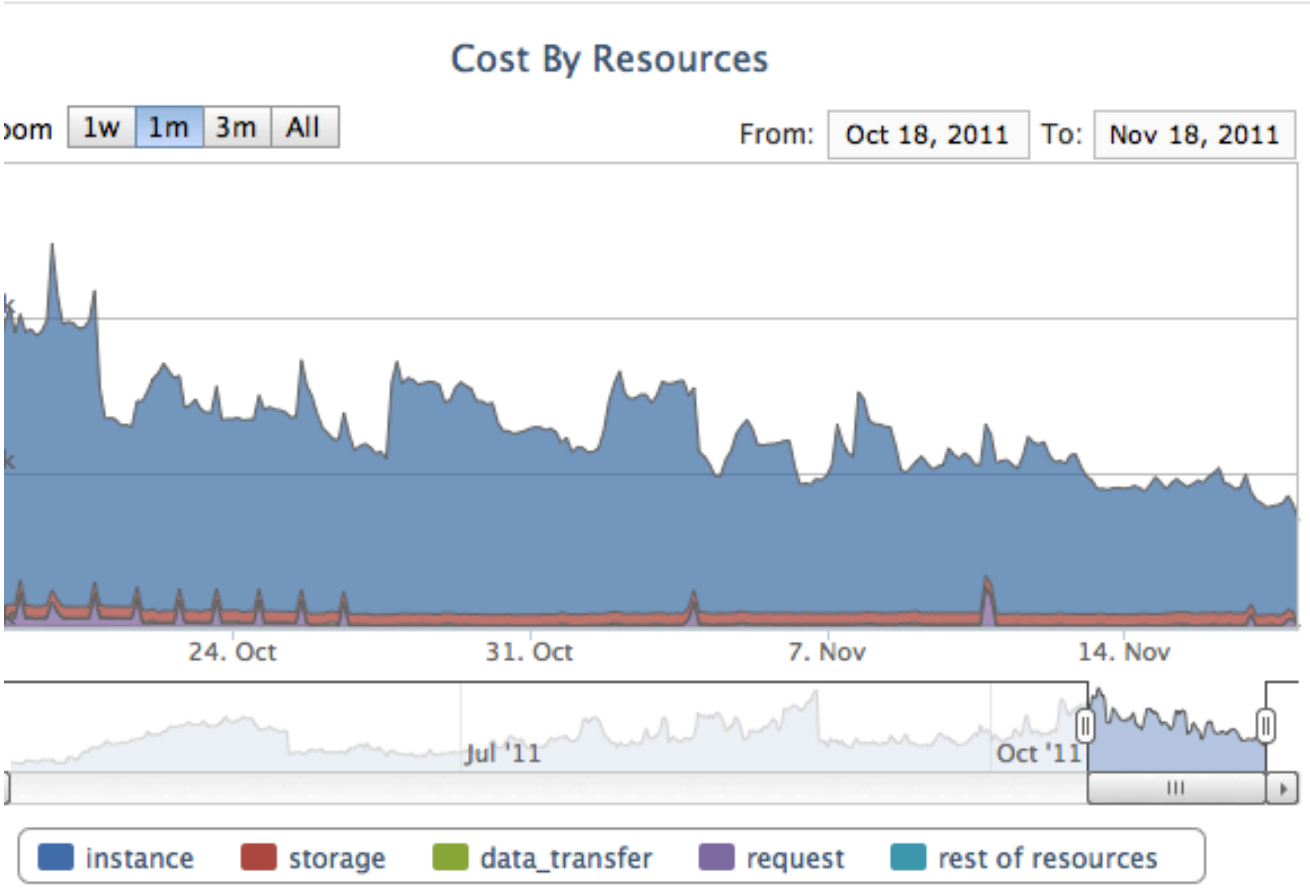
NETFLIX

Portals and Explorers

- Netflix Application Console (Asgard/NAC)
  - Primary AWS provisioning/config interface
- AWS Usage Analyzer
  - Breaks down costs by application and resource
- Cassandra Explorer
  - Browse clusters, keyspaces, column families
- Base Server Explorer
  - Browse service endpoints configuration, perf

AWS Usage

for test, carefully omitting any $ numbers…



NETFLIX

Platform Services

- Discovery – service registry for “Applications”
- Introspection – Entrypoints
- Cryptex – Dynamic security key management
- Geo – Geographic IP lookup
- Configuration Service – Dynamic properties
- Localization – manage and lookup local translations
- Evcache – ephemeral volatile cache
- Cassandra – Cross zone/region distributed data store
- Zookeeper – Distributed Coordination (Curator)
- Various proxies – access to old datacenter stuff

NETFLIX

Introspection - Entrypoints

- REST API for tools, apps, explorers, monkeys…
  - E.g. GET /REST/v1/instance/$INSTANCE_ID
- AWS Resources
  - Autoscaling Groups, EIP Groups, Instances
- Netflix PaaS Resources
  - Discovery Applications, Clusters of ASGs, History
- Full History of all Resources
  - Supports Janitor Monkey cleanup of unused resources

Entrypoints Queries

MongoDB used for low traffic complex queries against complex objects

Description	Range expression
Find all active instances.	all()
Find all instances associated with a group name.	%(cloudmonkey)
Find all instances associated with a discovery group.	/^cloudmonkey$/discovery()
Find all auto scale groups with no instances.	asg(),-has(INSTANCES;asg())
How many instances are not in an auto scale group?	count(all(),-info(eval(INSTANCES;asg())))
What groups include an instance?	*(i-4e108521)
What auto scale groups and elastic load balancers include an instance?	filter(TYPE;asg,elb;*(i-4e108521))
What instance has a given public ip?	filter(PUBLIC_IP;174.129.188.{0..255};all())

NETFLIX

Metrics Framework

- System and Application
  - Collection, Aggregation, Querying and Reporting
  - Non-blocking logging, avoids log4j lock contention
  - Honu-Streaming -> S3 -> EMR -> Hive

- Performance, Robustness, Monitoring, Analysis
  - Tracers, Counters – explicit code instrumentation log
  - SLA – service level response time percentiles
  - Servo annotated JMX extract to Cloudwatch

- Latency Testing and Inspection Infrastructure
  - Latency Monkey injects random delays and errors into service responses
  - Base Server Explorer Inspect client timeouts
  - Global property management to change client timeouts

Interprocess Communication

- Discovery Service registry for “applications”
  - “here I am” call every 30s, drop after 3 missed
  - “where is everyone” call
  - Redundant, distributed, moving to Zookeeper
- NIWS – Netflix Internal Web Service client
  - Software Middle Tier Load Balancer
  - Failure retry moves to next instance
  - Many options for encoding, etc.

Security Key Management

- AKMS
  - Dynamic Key Management interface
  - Update AWS keys at runtime, no restart
  - All keys stored securely, none on disk or in AMI

- Cryptex - Flexible key store
  - Low grade keys processed in client
  - Medium grade keys processed by Cryptex service
  - High grade keys processed by hardware (Ingrian)

AWS Persistence Services

- SimpleDB
  - Got us started, migrated to Cassandra now
  - NFSDB - Instrumented wrapper library
  - Domain and Item sharding (workarounds)

- S3
  - Upgraded/Instrumented JetS3t based interface
  - Supports multipart upload and 5TB files
  - Global S3 endpoint management

Netflix Platform Persistence

- Ephemeral Volatile Cache – evcache
  - Discovery-aware memcached based backend
  - Client abstractions for zone aware replication
  - Option to write to all zones, fast read from local

- Cassandra
  - Highly available and scalable (more later…)
- MongoDB
  - Complex object/query model for small scale use
- MySQL
  - Hard to scale, legacy and small relational models

Priam – Cassandra Automation

Available at http://github.com/netflix

- Netflix Platform Tomcat Code
- Zero touch auto-configuration
- State management for Cassandra JVM
- Token allocation and assignment
- Broken node auto-replacement
- Full and incremental backup to S3
- Restore sequencing from S3
- Grow/Shrink Cassandra “ring”

Astyanax

Available at http://github.com/netflix

- Cassandra java client
- API abstraction on top of Thrift protocol
- "Fixed" Connection Pool abstraction (vs. Hector)
  - Round robin with Failover
  - Retry-able operations not tied to a connection
  - Netflix PaaS Discovery service integration
  - Host reconnect (fixed interval or exponential backoff)
  - Token aware to save a network hop – lower latency
  - Latency aware to avoid compacting/repairing nodes – lower variance
- Batch mutation: set, put, delete, increment
- Simplified use of serializers via method overloading (vs. Hector)
- ConnectionPoolMonitor interface for counters and tracers
- Composite Column Names replacing deprecated SuperColumns

Astyanax Query Example

Paginate through all columns in a row

```
ColumnList<String> columns;
int pageize = 10;
try {
    RowQuery<String, String> query = keyspace
        .prepareQuery(CF_STANDARD1)
        .getKey("A")
        .setIsPaginating()
        .withColumnRange(new RangeBuilder().setMaxSize(pageize).build());

    while (!(columns = query.execute().getResult()).isEmpty()) {
        for (Column<String> c : columns) {
        }
    }
} catch (ConnectionException e) {
}
```

NETFLIX

High Availability

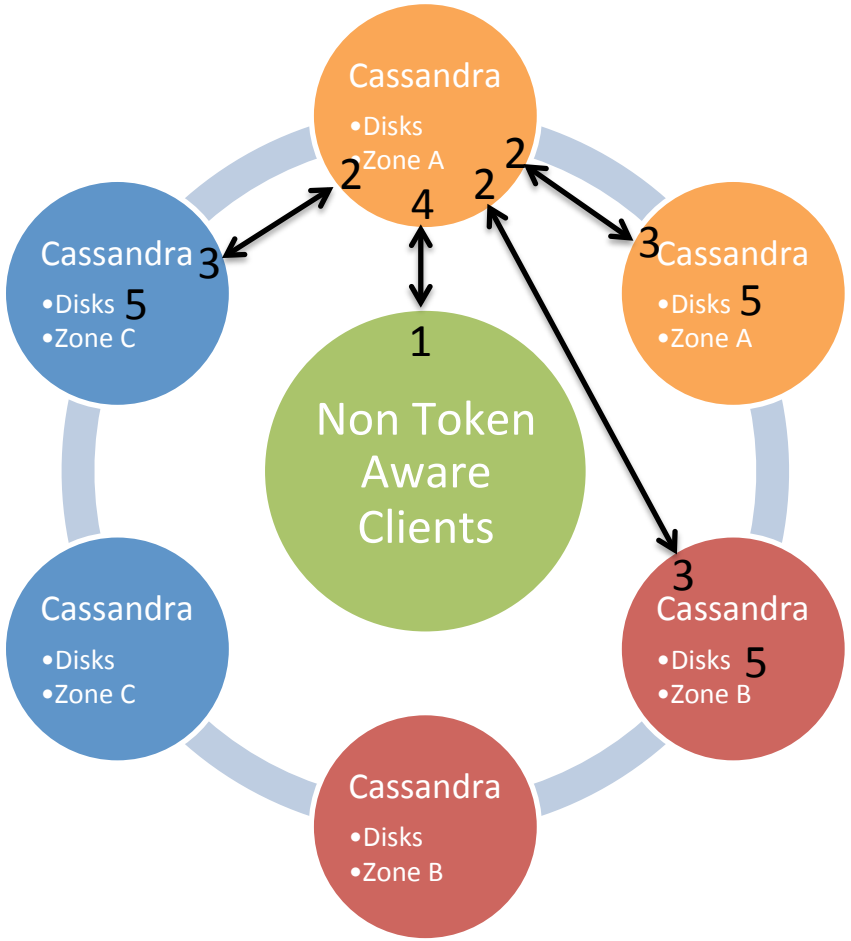
- Cassandra stores 3 local copies, 1 per zone
  - Synchronous access, durable, highly available
  - Read/Write One fastest, least consistent - ~1ms
  - Read/Write Quorum 2 of 3, consistent - ~3ms

- AWS Availability Zones
  - Separate buildings
  - Separate power etc.
  - Fairly close together

“Traditional” Cassandra Write Data Flows

Single Region, Multiple Availability Zone, Not Token Aware

1. Client Writes to any Cassandra Node
2. Coordinator Node replicates to nodes and Zones
3. Nodes return ack to coordinator
4. Coordinator returns ack to client
5. Data written to internal commit log disk (no more than 10 seconds later)



If a node goes offline, hinted handoff completes the write when the node comes back up.

Requests can choose to wait for one node, a quorum, or all nodes to ack the write

SSTable disk writes and compactions occur asynchronously

NETFLIX

Astyanax - Cassandra Write Data Flows

Single Region, Multiple Availability Zone, Token Aware

1. Client Writes to nodes and Zones
2. Nodes return ack to client
3. Data written to internal commit log disks (no more than 10 seconds later)

If a node goes offline, hinted handoff completes the write when the node comes back up.

Requests can choose to wait for one node, a quorum, or all nodes to ack the write

SSTable disk writes and compactions occur asynchronously

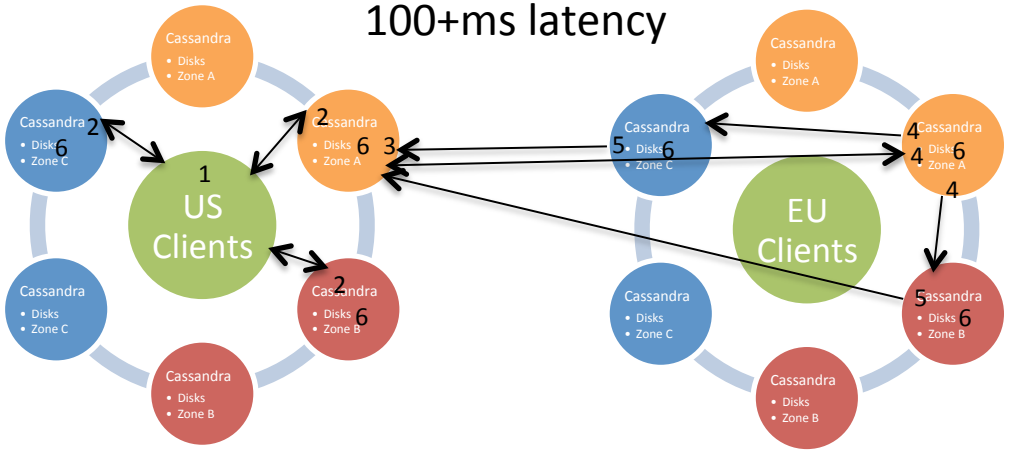
NETFLIX

Data Flows for Multi-Region Writes

Token Aware, Consistency Level = Local Quorum

1. Client writes to local replicas
2. Local write acks returned to Client which continues when 2 of 3 local nodes are committed
3. Local coordinator writes to remote coordinator.
4. When data arrives, remote coordinator node acks and copies to other remote zones
5. Remote nodes ack to local coordinator
6. Data flushed to internal commit log disks (no more than 10 seconds later)

If a node or region goes offline, hinted handoff completes the write when the node comes back up. Nightly global compare and repair jobs ensure everything stays consistent.



NETFLIX

Part 2. Running the Show
Operator Viewpoint

Rules of the Roadie

- Don’t lose stuff
- Make sure it scales
- Figure out when it breaks and what broke
- Yell at the right guy to fix it
- Keep everything organized

NETFLIX

Cassandra Backup

- Full Backup
  - Time based snapshot
  - SSTable compress -> S3
- Incremental
  - SSTable write triggers compressed copy to S3
- Archive
  - Copy cross region

NETFLIX

ETL for Cassandra

- Data is de-normalized over many clusters!
- Too many to restore from backups for ETL
- Solution – read backup files using Hadoop
- Aegisthus
  - http://techblog.netflix.com/2012/02/aegisthus-bulk-data-pipeline-out-of.html
  - High throughput raw SSTable processing
  - Re-normalizes many clusters to a consistent view
  - Extract, Transform, then Load into Teradata

Cassandra Archive

Appropriate level of paranoia needed…

- Archive could be un-readable
  - Restore S3 backups weekly from prod to test, and daily ETL
- Archive could be stolen
  - PGP Encrypt archive
- AWS East Region could have a problem
  - Copy data to AWS West
- Production AWS Account could have an issue
  - Separate Archive account with no-delete S3 ACL
- AWS S3 could have a global problem
  - Create an extra copy on a different cloud vendor….

NETFLIX

Tools and Automation

- Developer and Build Tools
  - Jira, Perforce, Eclipse, Jenkins, Ivy, Artifactory
  - Builds, creates .war file, .rpm, bakes AMI and launches
- Custom Netflix Application Console
  - AWS Features at Enterprise Scale (hide the AWS security keys!)
  - Auto Scaler Group is unit of deployment to production
- Open Source + Support
  - Apache, Tomcat, Cassandra, Hadoop
  - Datastax support for Cassandra, AWS support for Hadoop via EMR
- Monitoring Tools
  - Alert processing gateway into Pagerduty
  - AppDynamics – Developer focus for cloud http://appdynamics.com

NETFLIX

Scalability Testing

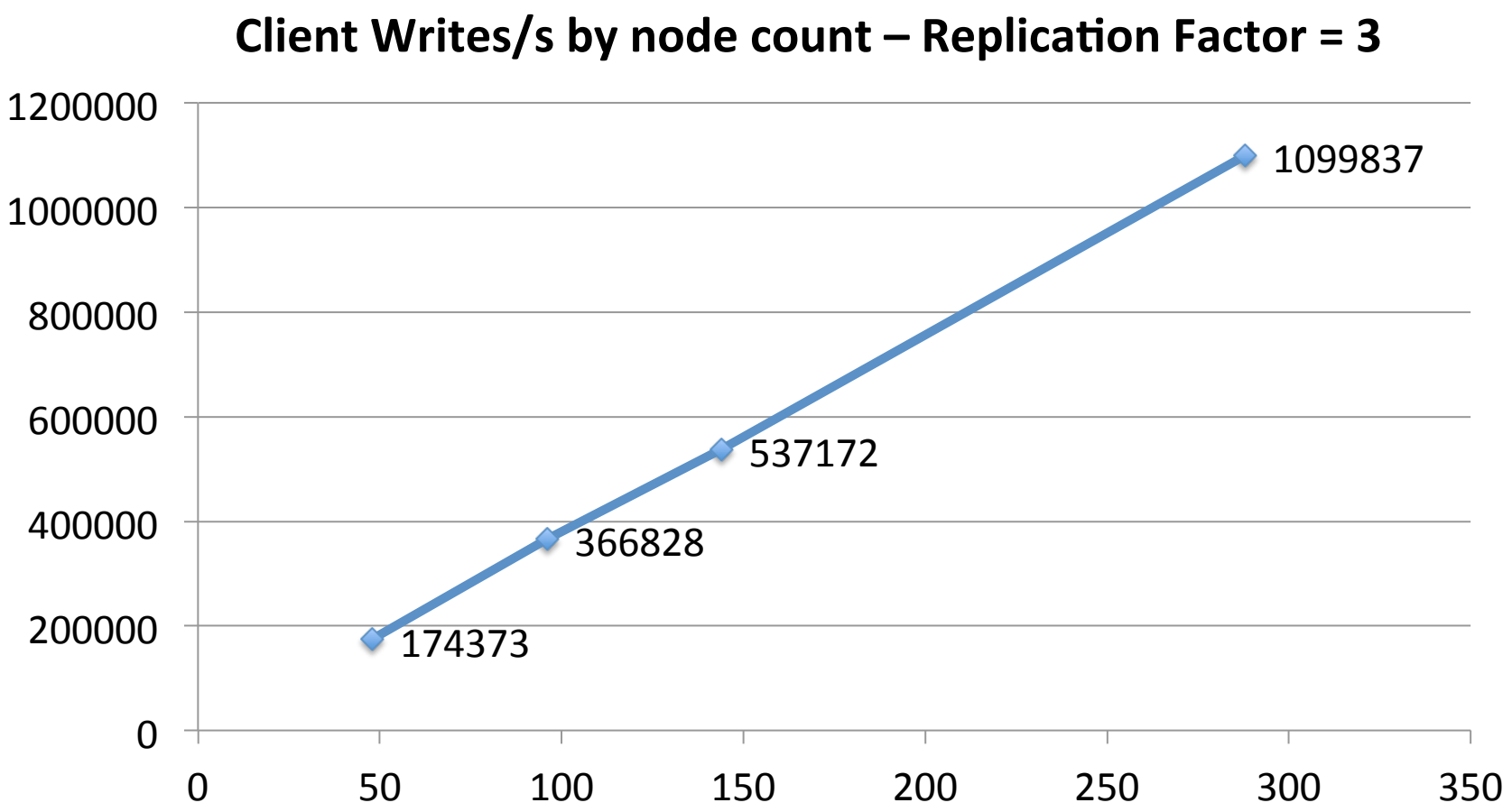
- Cloud Based Testing – frictionless, elastic
  - Create/destroy any sized cluster in minutes
  - Many test scenarios run in parallel
- Test Scenarios
  - Internal app specific tests
  - Simple “stress” tool provided with Cassandra
- Scale test, keep making the cluster bigger
  - Check that tooling and automation works…
  - How many ten column row writes/sec can we do?

<DrEvil>ONE MILLION</DrEvil>

NETFLIX

Scale-Up Linearity

http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html



NETFLIX

Availability and Resilience

Chaos Monkey

- 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-Thursday 9am-3pm random instance kill
- Application configuration option
  - Apps now have to opt-out from Chaos Monkey

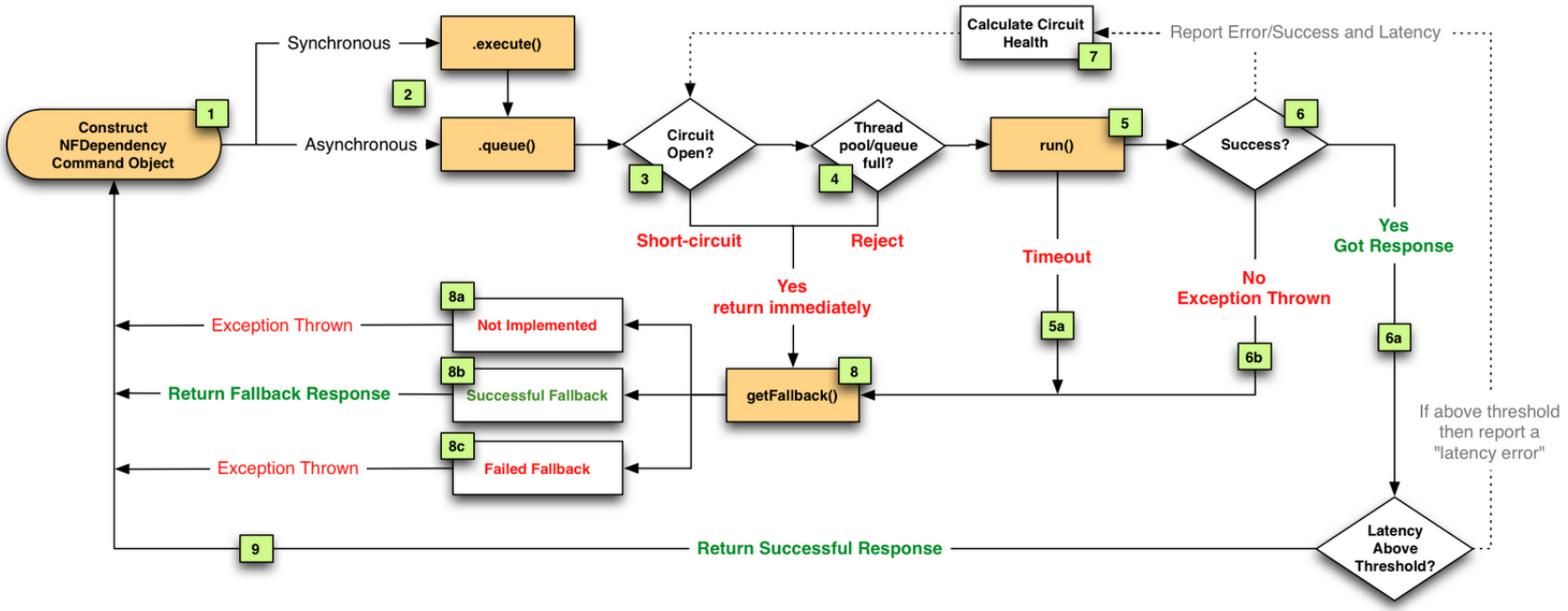
NETFLIX

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



NETFLIX

PaaS Operational Model

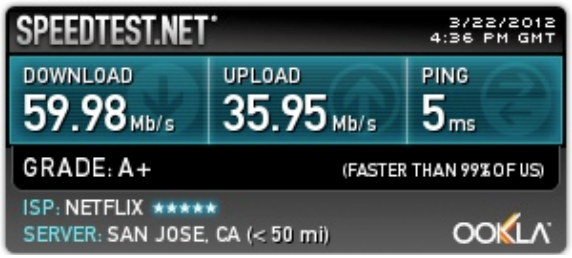
- Developers
  - Provision and run their own code in production
  - Take turns to be on call if it breaks (pagerduty)
  - Configure autoscalers to handle capacity needs

- DevOps and PaaS (aka NoOps)
  - DevOps is used to build and run the PaaS
  - PaaS constrains Dev to use automation instead
  - PaaS puts more responsibility on Dev, with tools

What's Left for Corp IT?

- Corporate Security and Network Management
  - Billing and remnants of streaming service back-ends in DC
- Running Netflix' DVD Business
  - Tens of Oracle instances
  - Hundreds of MySQL instances
  - Thousands of VMWare VMs
  - Zabbix, Cacti, Splunk, Puppet
- Employee Productivity
  - Building networks and WiFi
  - SaaS OneLogin SSO Portal
  - Evernote Premium, Safari Online Bookshelf, Dropbox for Teams
  - Google Enterprise Apps, Workday HCM/Expense, Box.com
  - Many more SaaS migrations coming…

Corp WiFi Performance



NETFLIX

Implications for IT Operations

- Cloud is run by developer organization
  - Product group's "IT department" is the AWS API and PaaS
  - CorpIT handles billing and some security functions

Cloud capacity is 10x bigger than Datacenter

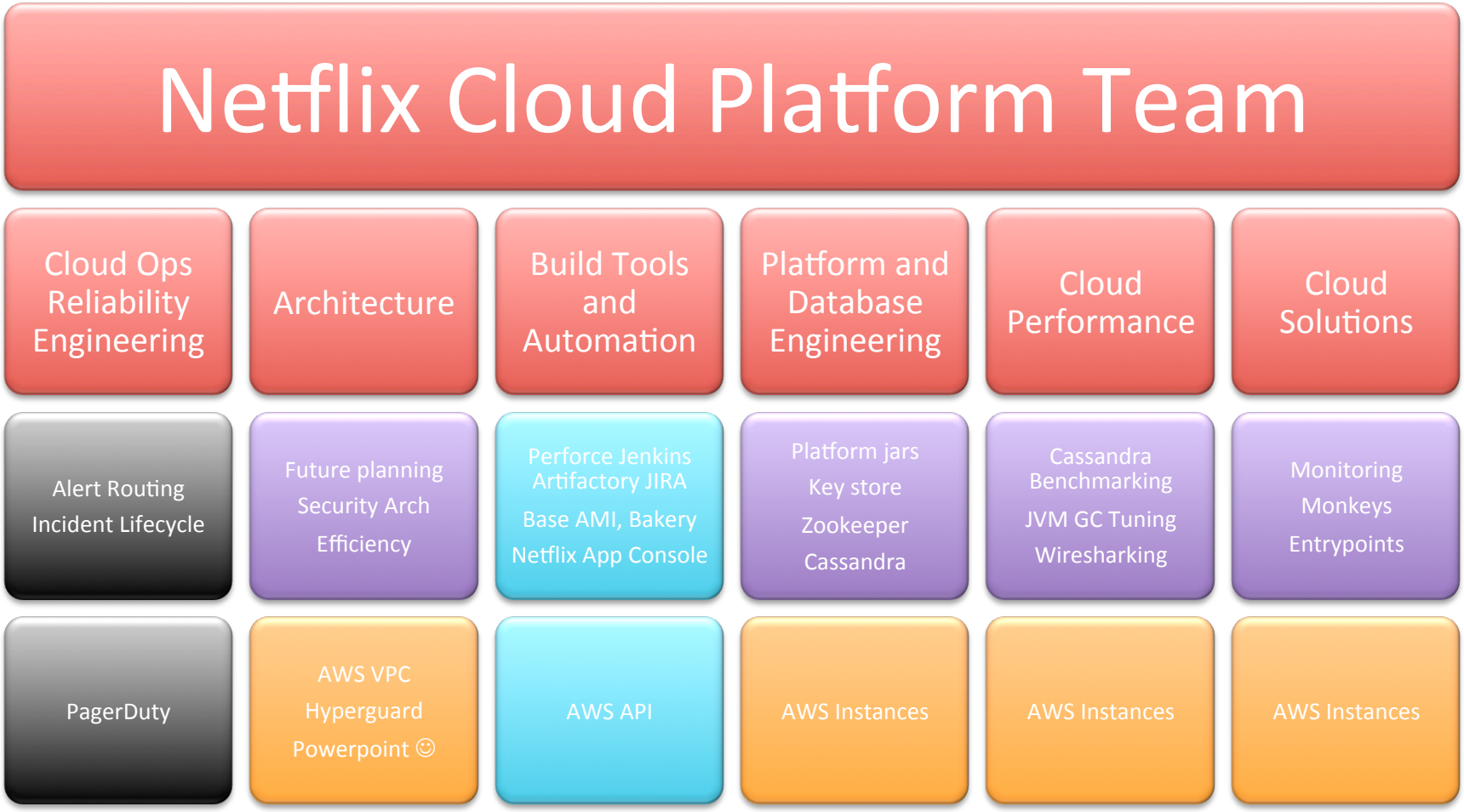
  - Datacenter oriented IT didn't scale up as we grew
  - We moved a few people out of IT to do DevOps for our PaaS

- Traditional IT Roles and Silos are going away
  - We don't have SA, DBA, Storage, Network admins for cloud
  - Developers deploy and "run what they wrote" in production

NETFLIX

Netflix PaaS Organization

Developer Org Reporting into Product Development, not ITops



NETFLIX

Part 3. Making the Instruments Builder Viewpoint

NETFLIX

Components

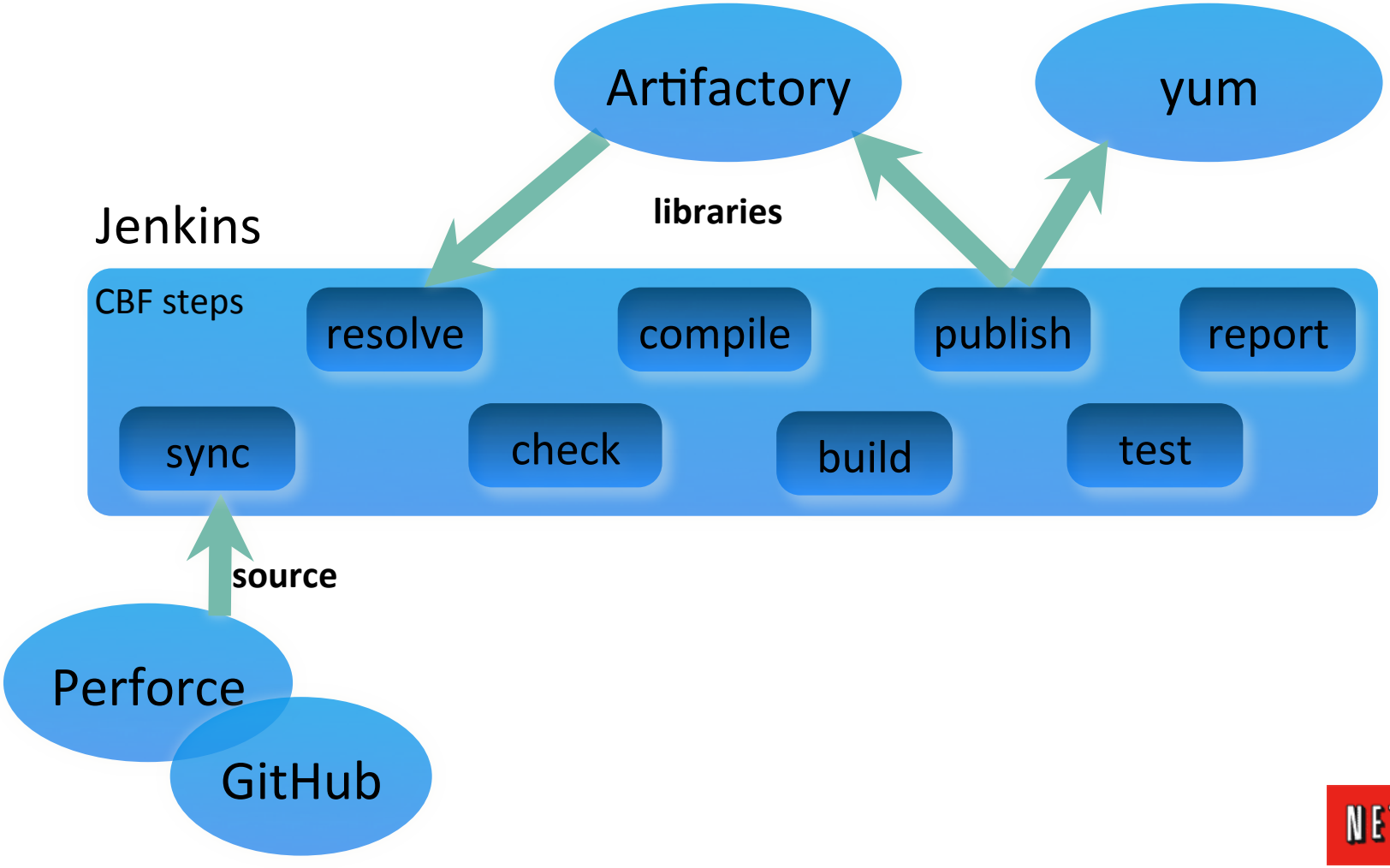
- Continuous build framework turns code into AMIs
- AWS accounts for test, production, etc.
- Cloud access gateway
- Service registry
- Configuration properties service
- Persistence services
- Monitoring, alert forwarding
- Backups, archives

NETFLIX

Common Build Framework

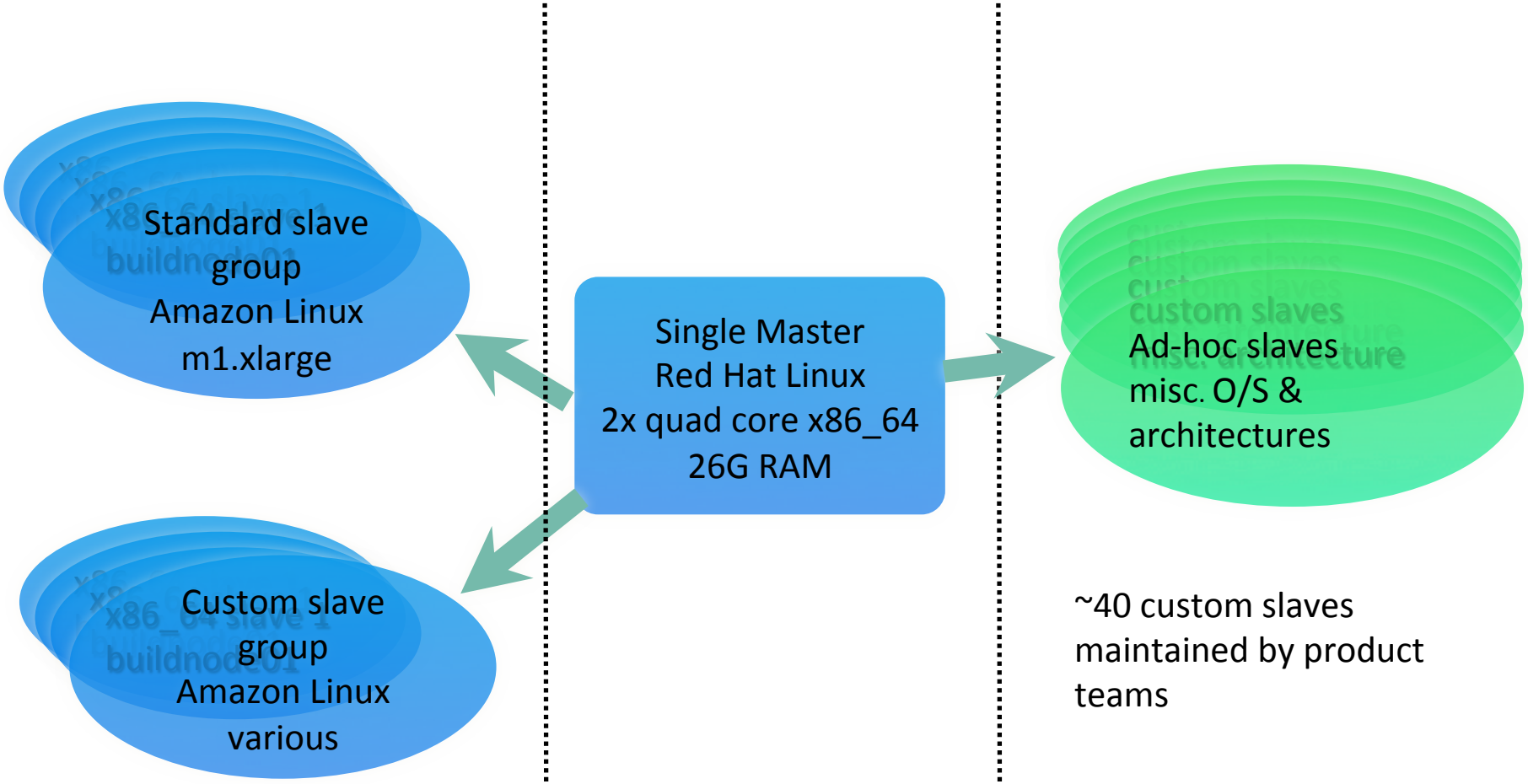
Extracted from
"Building and Deploying Netflix in the Cloud"
by @bmoyles and @garethbowles
On slideshare.net/netflix

Build Pipeline



NETFLIX

Jenkins Architecture



Other Uses of Jenkins

Maintence of test and prod Cassandra clusters

Automated integration tests for bake and deploy

Production bake and deployment

Housekeeping of the build / deploy infrastructure

Netflix Extensions to Jenkins

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

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

The DynaSlave Plugin
What We Have

Exposes a new endpoint in Jenkins that EC2 instances in VPC use for registration

Allows a slave to name itself, label itself, tell Jenkins how many executors it can support

EC2 == Ephemeral. Disconnected nodes that are gone for > 30 mins are reaped

Sizing handled by EC2 ASGs, tweaks passed through via user data (labels, names, etc)

The DynaSlave Plugin
What's Next

Enhanced security/registration of nodes

Dynamic resource management

- have Jenkins respond to build demand

Slave groups

- Allows us to create specialized pools of build nodes

Refresh mechanism for slave tools

- JDKs, Ant versions, etc.

Give it back to the community

- watch techblog.netflix.com!

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
  - Build 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

NETFLIX

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

NETFLIX

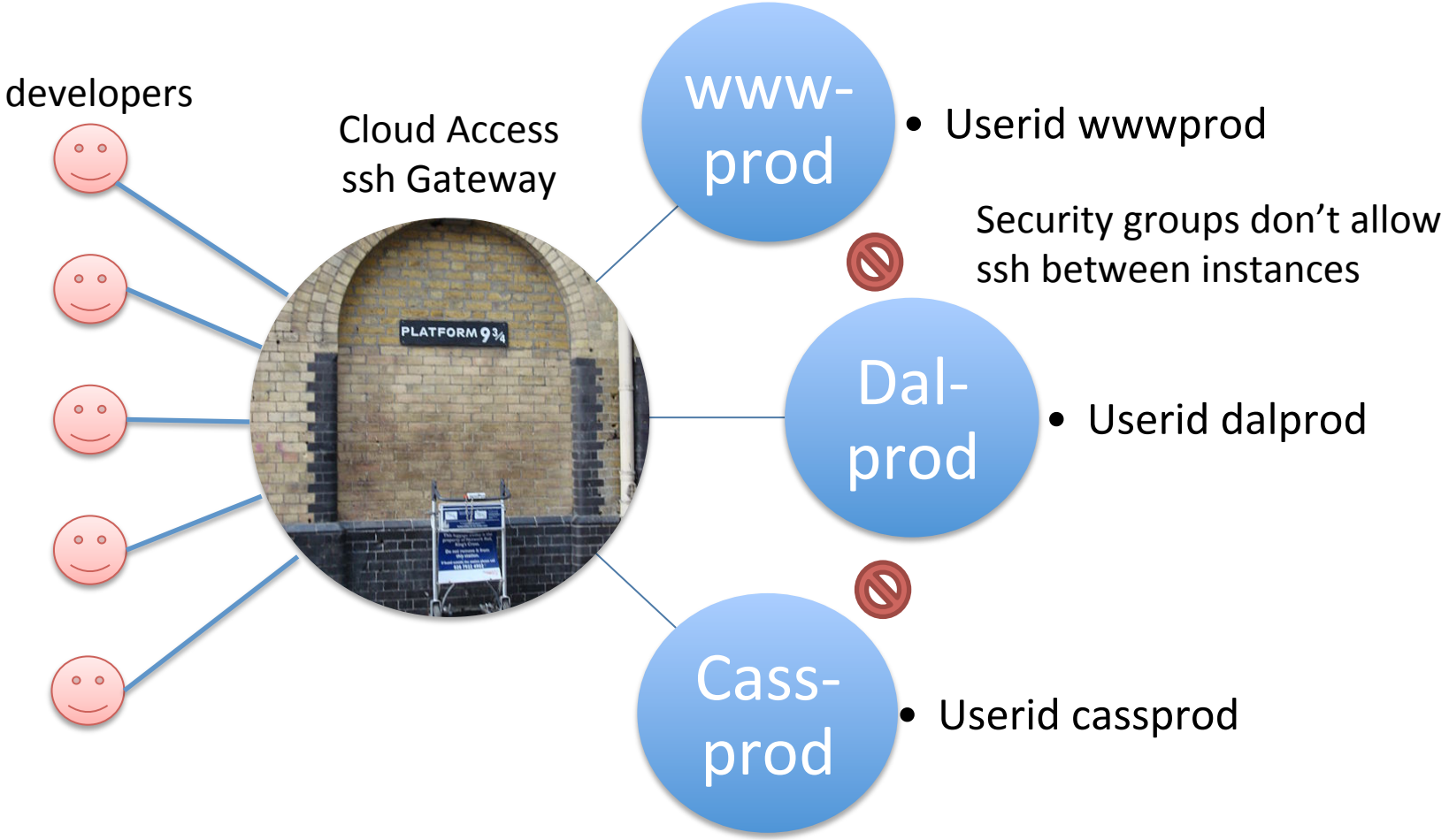
Cloud Access Gateway

- Datacenter or office based
  - A separate VM for each AWS account
  - Two per account for high availability
  - Mount NFS shared home directories for developers
  - Instances trust the gateway via a security group
- Manage how developers login to cloud
  - Access control via ldap group membership
  - Audit logs of every login to the cloud
  - Similar to awsfabrictasks ssh wrapper

http://readthedocs.org/docs/awsfabrictasks/en/latest/

NETFLIX

Cloud Access Control



NETFLIX

Now Add Code

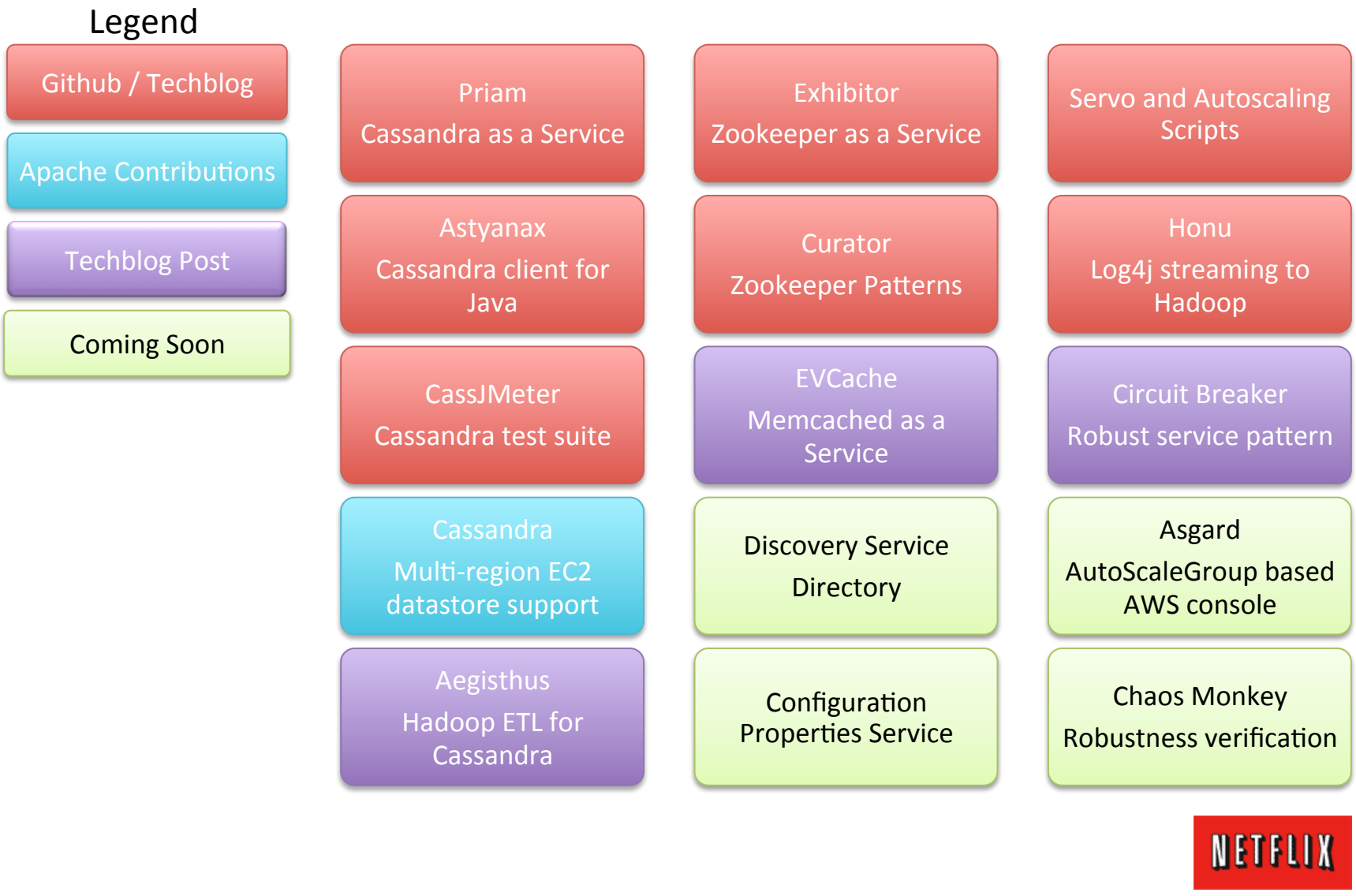
Netflix has open sourced a lot of what you need, more is on the way…

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

Open Source Projects and Posts



Asgard

Not quite out yet…

- 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

NETFLIX

“Discovery” - Service Directory

- Map an instance to a service type
  - Load balance over clusters of instances
  - Private namespace, so DNS isn’t useful
  - Foundation service, first to deploy
- Highly available distributed coordination
  - Deploy one Apache Zookeeper instance per zone
  - Netflix Curator includes simple discovery service
  - Netflix Exhibitor manages Zookeeper reliably

Configuration Properties Service

- Dynamic hierarchical & propagates in seconds
  - Client timeouts, feature set enables
  - Region specific service endpoints
  - Cassandra token assignments etc. etc.
- Used to configure everything
  - So everything depends on it…
  - Coming soon to github
  - Pluggable backend storage interface

Persistence services

- Use SimpleDB as a bootstrap
  - Good use case for DynamoDB or SimpleDB
- Netflix Priam
  - Cassandra automation

Monitoring, alert forwarding

- Multiple monitoring systems
  - Internally developed data collection runs on AWS
  - AppDynamics APM product runs as external SaaS
  - When one breaks the other is usually OK…
- Alerts routed to the developer of that app
  - Alert gateway combines alerts from all sources
  - Deduplication, source quenching, routing
  - Warnings sent via email, critical via pagerduty

NETFLIX

Backups, archives

- Cassandra Backup via Priam to S3 bucket
  - Create versioned S3 bucket with TTL option
  - Setup service to encrypt and copy to archive
- Archive Account with Read/Write ACL to prod
  - Setup in a different AWS region from production
  - Create versioned S3 bucket with TTL option

Chaos Monkey

- Install it on day 1 in test and production
- Prevents people from doing local persistence
- Kill anything not protected by an ASG
- Supports whitelist for temporary do-not-kill

- Open source soon, code cleanup in progress…

NETFLIX

You take it from here…

- Keep watching github for more goodies
- Add your own code
- Let us know what you find useful
- Bugs, patches and additions all welcome
- See you at AWS Re:Invent?

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 support
- More open sourced components

NETFLIX

Wrap Up

Answer your remaining questions…

What was missing that you wanted to cover?

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/adriancockcroft
@adrianco #netflixcloud

End of Part 3 of 3


Gluecon Rocks

You want an Encore?

If there is enough time… (there wasn't)

Something for the hard core complex adaptive systems people to digest.

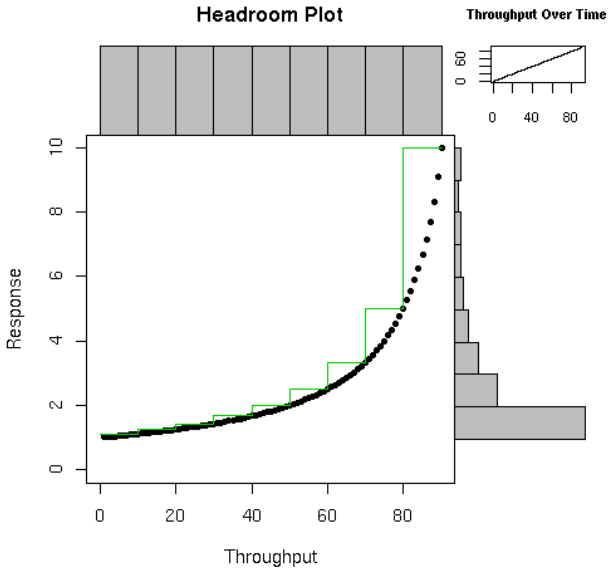
A Discussion of Workloads and How They Behave

Workload Characteristics

- A quick tour through a taxonomy of workload types
- Start with the easy ones and work up
- Why personalized workloads are different and hard
- Some examples and coping strategies

NETFLIX

Simple Random Arrivals



- Random arrival of transactions with fixed mean service time
  - Little's Law: QueueLength = Throughput * Response
  - Utilization Law: Utilization = Throughput * ServiceTime
- Complex models are often reduced to this model
  - By averaging over longer time periods since the formulas only work if you have stable averages
  - By wishful thinking (i.e. how to fool yourself)

Mixed random arrivals of transactions with stable mean service times

- Think of the grocery store checkout analogy
  - Trolleys full of shopping vs. baskets full of shopping
  - Baskets are quick to service, but get stuck behind carts
  - Relative mixture of transaction types starts to matter
- Many transactional systems handle a mixture
  - Databases, web services
- Consider separating fast and slow transactions
  - So that we have a "10 items or less" line just for baskets
  - Separate pools of servers for different services
  - The old rule - don't mix OLTP with DSS queries in databases
- Performance is often thread-limited
  - Thread limit and slow transactions constrains maximum throughput
- Model mix using analytical solvers (e.g. PDQ perfdynamics.com)

Load dependent servers – varying mean service times

- Mean service time may increase at high throughput
  - Due to non-scalable algorithms, lock contention
  - System runs out of memory and starts paging or frequent GC
- Mean service time may also decrease at high throughput
  - Elevator seek and write cancellation optimizations in storage
  - Load shedding and simplified fallback modes
- Systems have "tipping points" if the service time increases
  - Hysteresis means they don't come back when load drops
  - This is why you have to kill catatonic systems
  - Best designs shed load to be stable at the limit – circuit breaker pattern
  - Practical option is to try to avoid tipping points by reducing variance
- Model using discrete event simulation tools
  - Behaviour is non-linear and hard to model

Self-similar / fractal workloads

- Bursty rather than random arrival rates
- Self-similar
  - Looks “random” at close up, stays “random” as you zoom out
  - Work arrives in bursts, transactions aren’t independent
  - Bursts cluster together in super-bursts, etc.
- Network packet streams tend to be fractal
- Common in practice, too hard to model
  - Probably the most common reason why your model is wrong!

State Dependent Service Workloads

- Personalized services that store user state/history
  - Transactions for new users are quick
  - Transactions for users with lots of state/history are slower
  - As user base builds state and ages you get into trouble…
- Social Networks, Recommendation Services
  - Facebook, Flickr, Netflix, Twitter etc.
- "Abandon hope all ye who enter here"
  - Not tractable to model, repeatable tests are tricky
  - Long fat tail response time distribution and timeouts
- Try to transform workloads to more tractable forms

Example - Twitter Workload

- @adrianco tweets – copy to 4300 or so other users
- @zoecello tweets many times a day – to over 1M users
- @barackobama tweets every few days – to over 12M users
- It's the same transaction, but the service time varies by several orders of magnitude
- The best (most active and connected = most valuable) users trigger a "denial of service attack" on the systems when they tweet
- Cascading effect as many others re-tweet

Example - Netflix Movie Choosing

- "Pick 24 genres/subgenres etc. of 75 movies each for me"
  - used by TV based devices like Xbox360, PS/3, iPhone app
- New user
  - No history of what they have rented (DVD) or streamed
  - No star ratings for movies, possibly some genre ratings
  - Basic demographic info
  - Fast to calculate, easy to find many good choices to return
- User with several years tenure
  - Thousands of movies rented or streamed, "seen it already"
  - Hundreds to thousands of star ratings, lots of genre ratings
  - Requests may time out and return fewer or worse choices

Workload Modelling Survival Methods

- Simplify the workload algorithms
  - move from hard or impossible to simpler models
  - decouple, cache and pre-compute to get constant service times
- Stand further away
  - averaging is your friend – gets rid of complex fluctuations
- Minimalist Models
  - most models are far too complex – the classic beginners error…
  - the art of modelling is to only model what really matters
- Don't model details you don't use
  - model peak hour of the week, not day to day fluctuations
  - e.g. "Will the web site survive next Sunday night?"