The Global Netflix Platform

A Large Scale Java oriented PaaS running on AWS

October 24th, 2011

Adrian Cockcroft

@adrianco #netflixcloud

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

Netflix Inc.

With more than 20 million streaming members in the United States, Canada and Latin America, Netflix, Inc. is the world's leading Internet subscription service for enjoying movies and TV shows.

<u>International Expansion</u>

Netflix, Inc., the leading global Internet movie subscription service… announced it will expand to the United Kingdom and Ireland in early 2012.

Source: http://ir.netflix.com

The Global Netflix Platform

Netflix Cloud Migration

Netflix Platform Services and Interfaces

Highly Available and Globally Distributed Data

Scalability and Performance

Why Use Public Cloud?

Get stuck with wrong config

Wait Wait File tickets

Ask permission Wait Wait

Wait

Things We Don't Do

Wait

Run out of space/power

Plan capacity in advance

Have meetings with IT Wait

NETFLIX

Better Business Agility

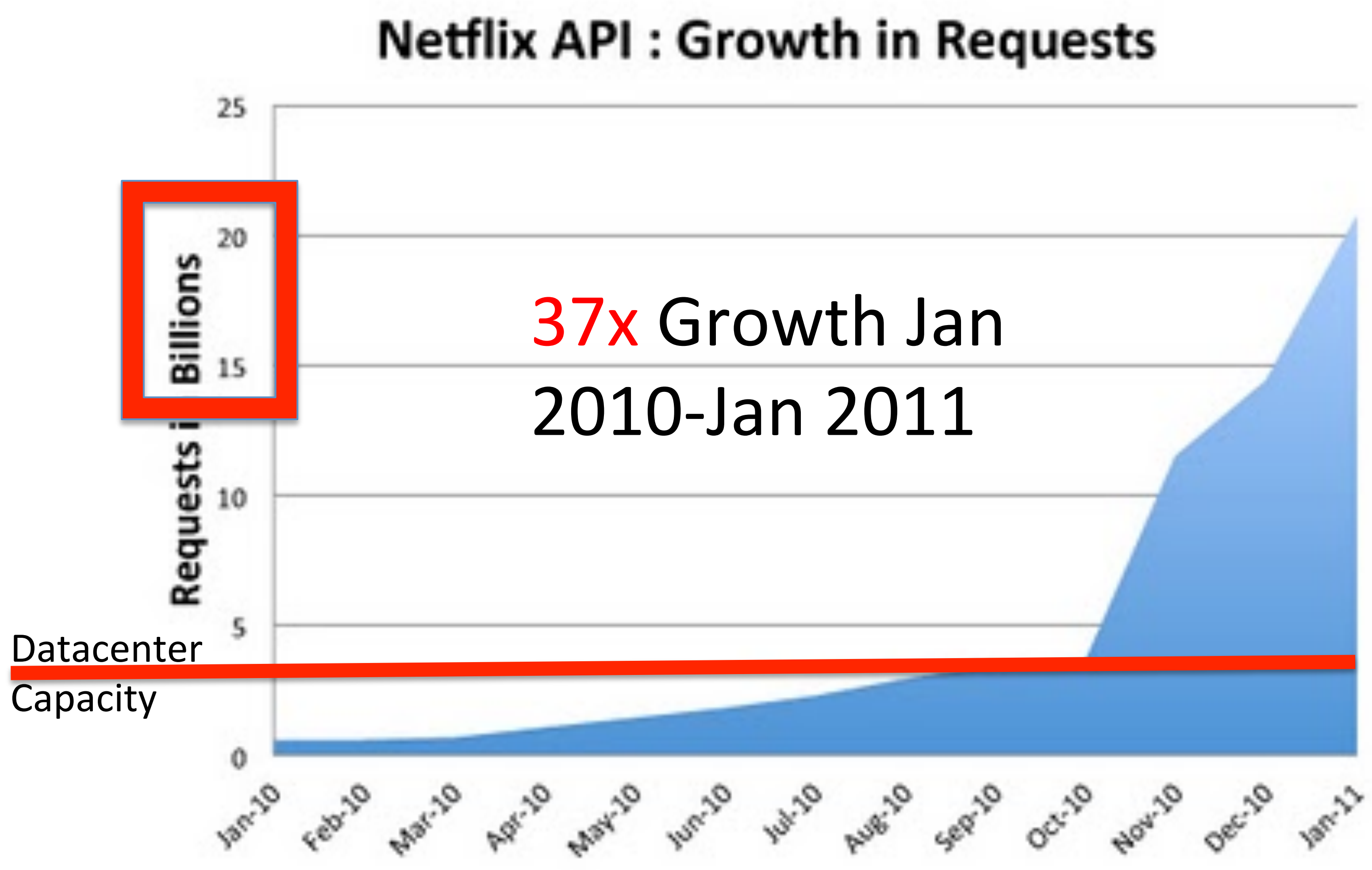
Netflix could not build new datacenters fast enough

Capacity growth is accelerating, unpredictable

Product launch spikes - iPhone, Wii, PS3, XBox

Out-Growing Data Center

http://techblog.netflix.com/2011/02/redesigning-netflix-api.html



Netflix.com is now ~100% Cloud

A few small back end data sources still in progress

All international product is cloud based

USA specific logistics remains in the Datacenter

Working aggressively on billing, PCI compliance on AWS

Netflix Choice was AWS with our own platform and tools

Unique platform requirements and extreme scale, agility and flexibility

Leverage AWS Scale
“the biggest public cloud”

AWS investment in features and automation

Use AWS zones and regions for high availability, scalability and global deployment

But isn’t Amazon a competitor?

Many products that compete with Amazon run on AWS
We are a “poster child” for the AWS Architecture
Netflix is one of the biggest AWS customers
Strategy – turn competitors into partners

Could Netflix use another cloud?

Would be nice, we use three interchangeable CDN Vendors

But no-one else has the scale and features of AWS

You have to be this tall to ride this ride…

Maybe in 2-3 years?

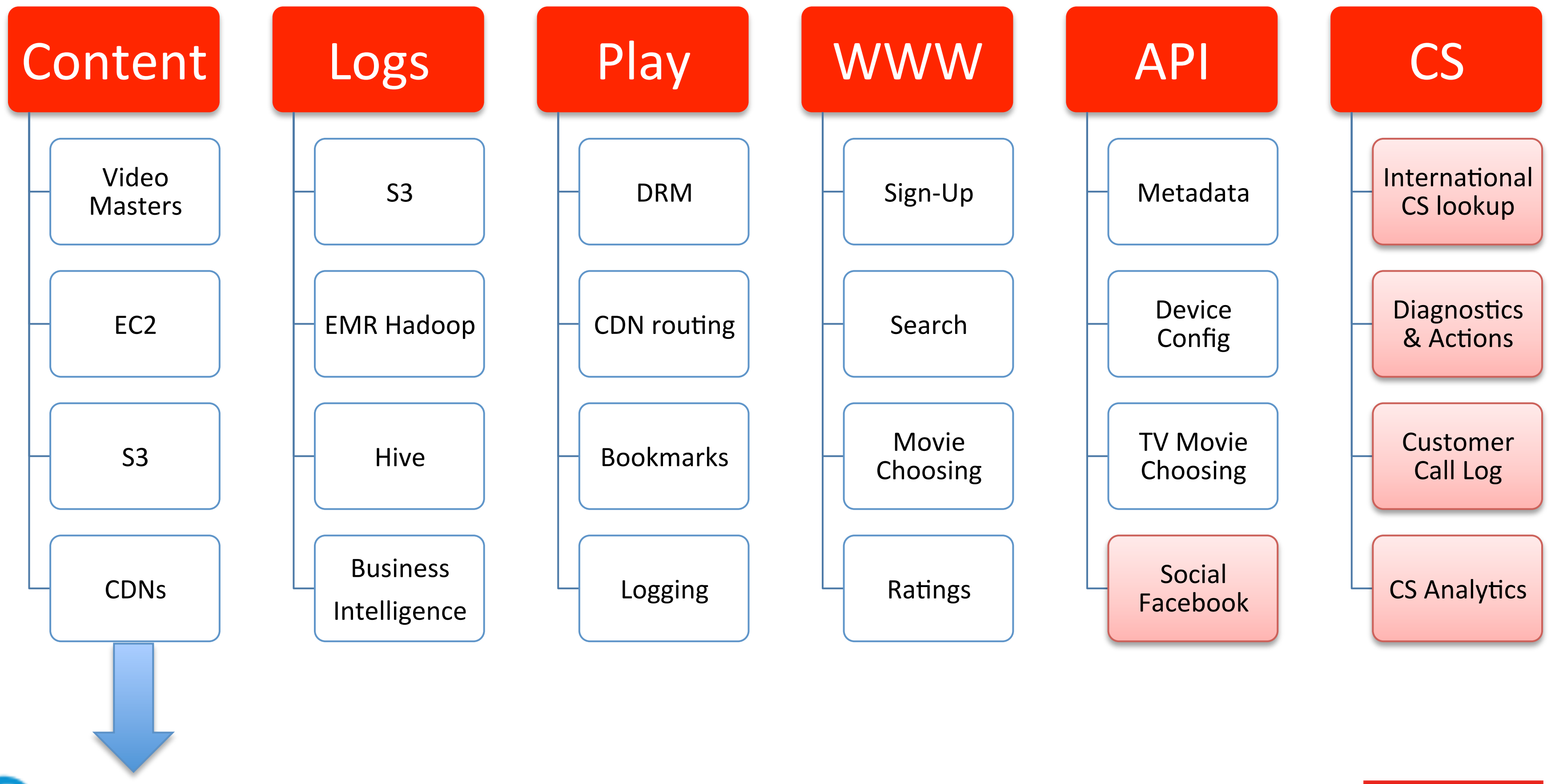
We want to use clouds, we don’t have time to build them

Public cloud for agility and scale

We use electricity too, but don’t want to build our own power station…

AWS because they are big enough to allocate thousands of instances per hour when we need to

Netflix Deployed on AWS



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 Availability Zones – US-East, US-West, EU-Eire, Asia-Singapore, Asia-Japan, 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)
- SDB – Simple Data Base (hosted http based NoSQL data store)
- 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 (extension of enterprise datacenter network into cloud)
- IAM – Identity and Access Management (fine grain role based security keys)

Boot Camp

- One day "Netflix Cloud Training" class
  - Has been run 5 times for 20-45 people each time
- Half day of presentations
- Half day hands-on
  - Create your own hello world app
  - Launch in AWS test account
  - Login to your cloud instances
  - Find monitoring data on your cloud instances
  - Connect to Cassandra and read/write data

Netflix Built a PaaS!

- Netflix Cloud Systems team (50+ rock-stars :)
  - – VP Cloud Systems (Yury Izrailevsky)
  - – Site Reliability Engineering (@jedberg) Hiring++!
  - – Cloud Performance (Denis Sheahan)
  - – Database Engineering - Cassandra+MySQL (@r39132)
  - – Platform Engineering – Astyanax (Eran Landau)
  - – Cloud Tools Engineering – Jenkins (@cquinn)
  - – Cloud Solutions Team – Monkeys (@atseitlin)
  - – Security (Jason Chan)
  - – Architecture (@adrianco)

Netflix Global PaaS

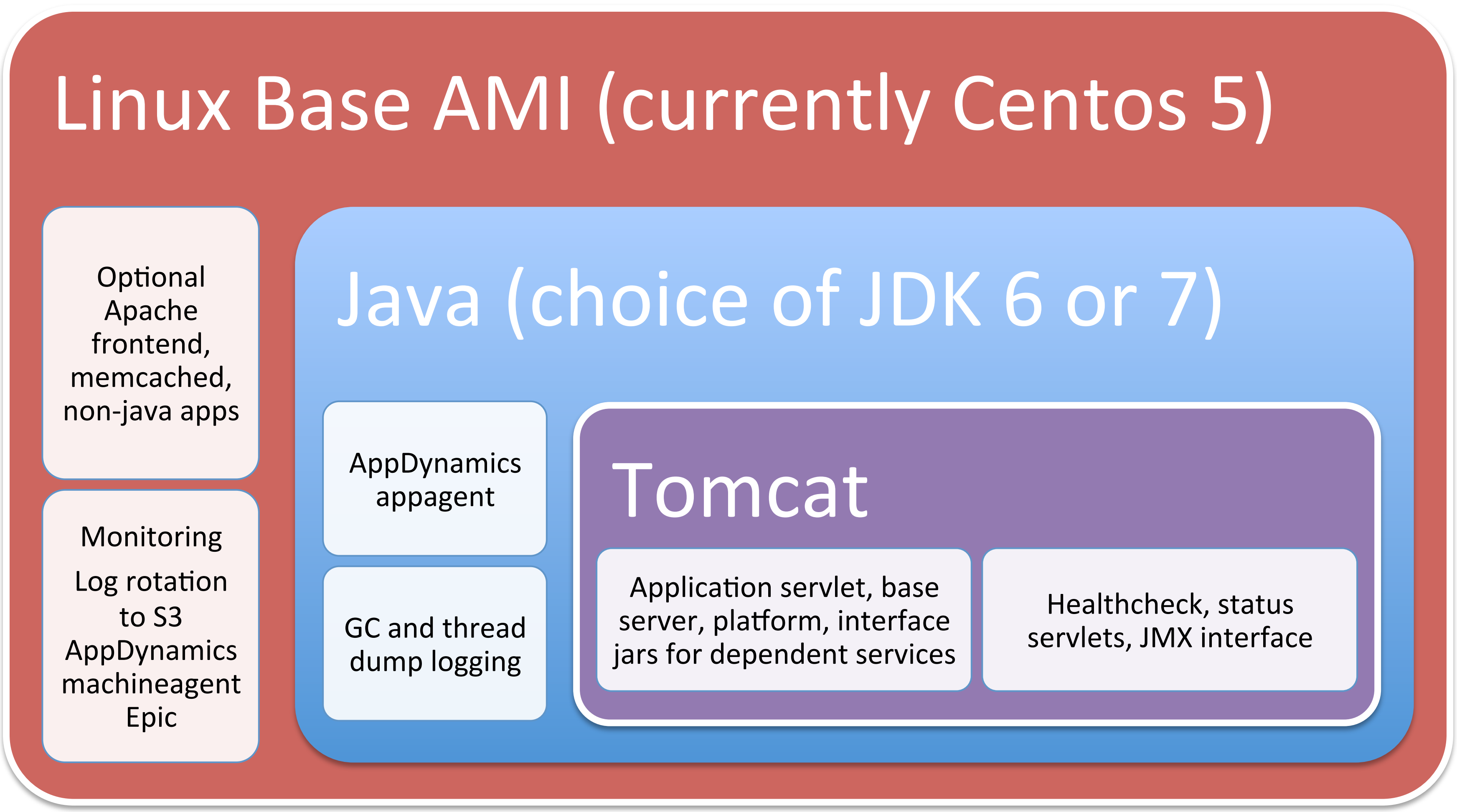
- Architecture Features and Overview
- Portals and Explorers
- Platform Services
- Platform APIs
- Platform Frameworks
- Persistence
- Scalability Benchmark

Global PaaS?

Toys are nice, but this is the real thing…

- 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
- 20M+ users USA, Canada, Latin America (UK, Eire)

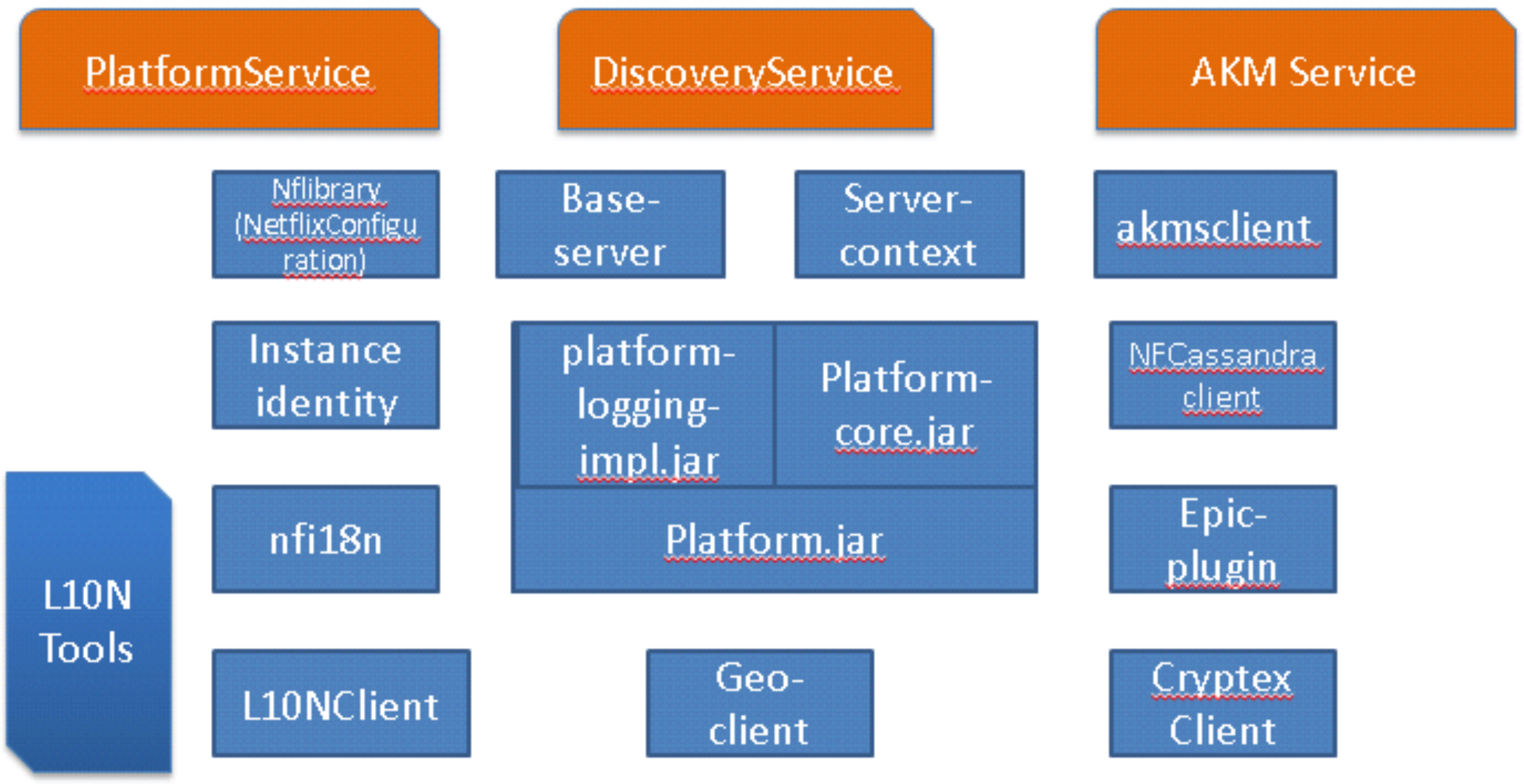
Instance Architecture


Linux Base AMI (currently Centos 5)
Optional Apache frontend, memcached, non-java apps
Monitoring
Log rotation to S3
AppDynamics machineagent
Epic
Java (choice of JDK 6 or 7)
AppDynamics appagent
GC and thread dump logging
Tomcat
Application servlet, base server, platform, interface jars for dependent services
Healthcheck, status servlets, JMX interface

Security Architecture

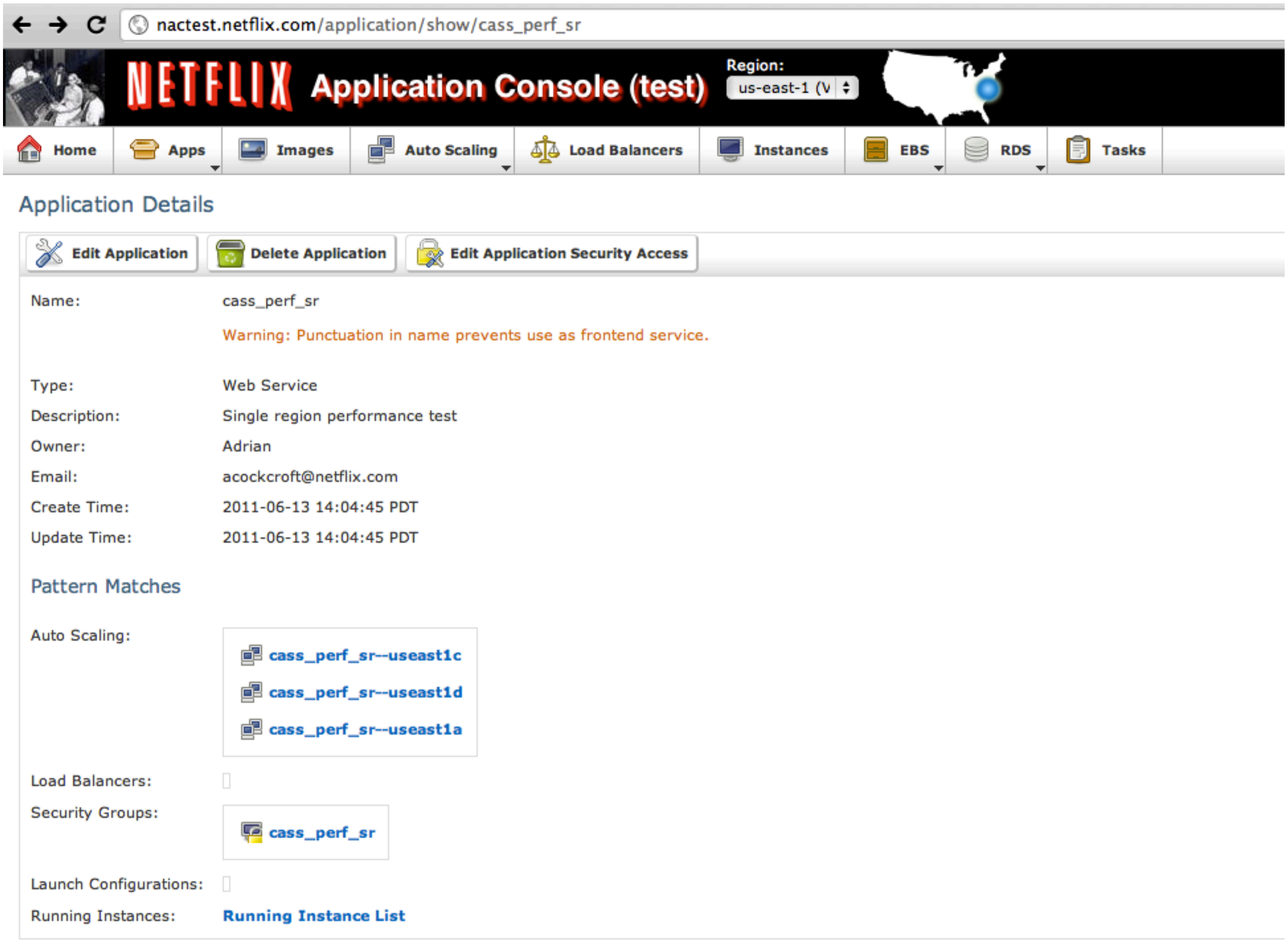
- Instance Level Security baked into base AMI
  - Login via ssh only allowed via portal
  - Each app type runs as its own userid app{test|prod}
- AWS Security, Identity and Access Management
  - Each app has its own security group (firewall ports)
  - Fine grain user roles and resource ACLs
- Key Management
  - AWS Keys dynamically provisioned, easy updates
  - High grade app key management support

Core Platform Frameworks and APIs



Portals and Explorers

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

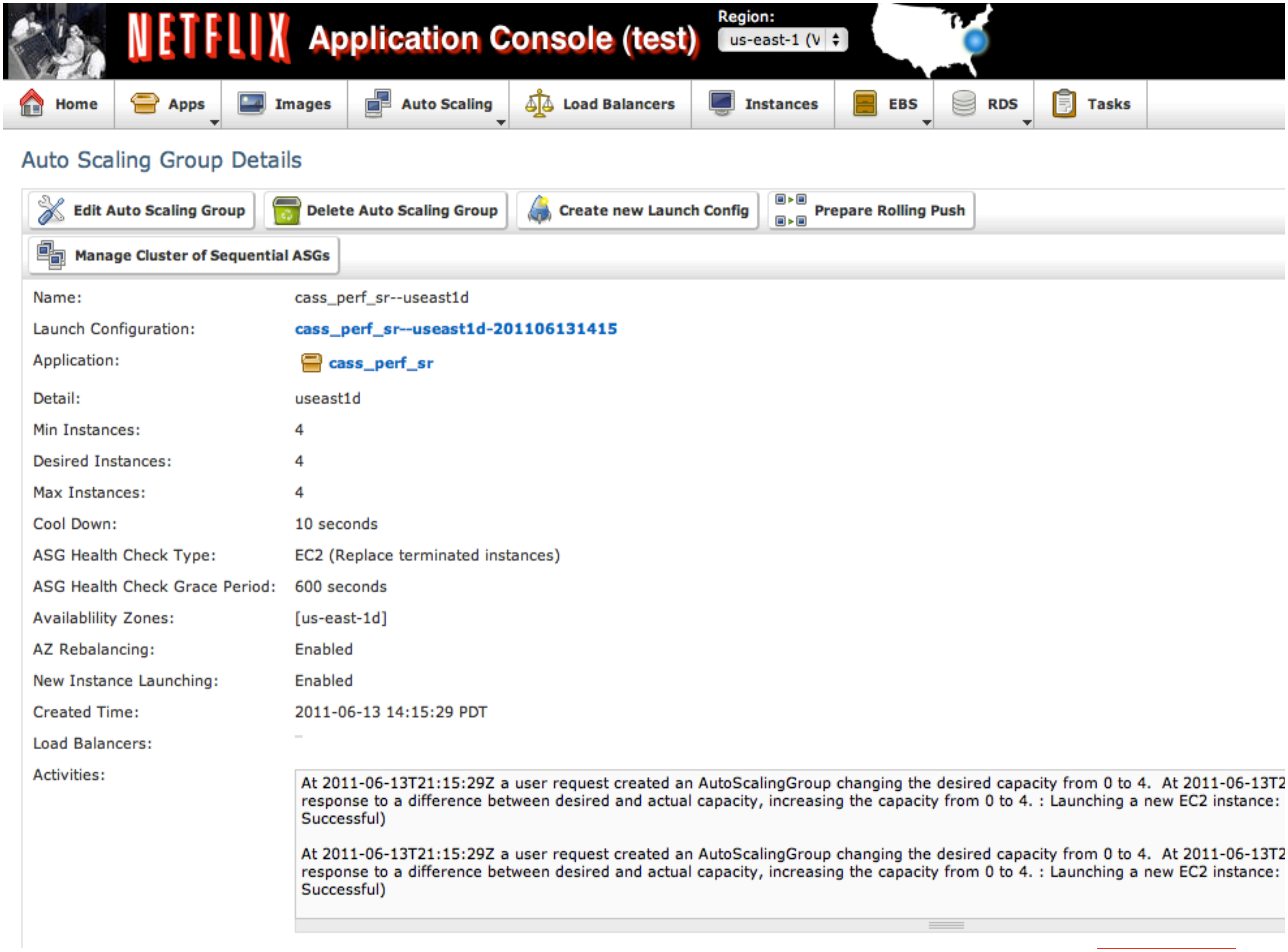

nactest.netflix.com/application/show/cass_perf_sr
NETFLIX Application Console (test)
Region: us-east-1 (V
Home
Apps
Images
Auto Scaling
Load Balancers
Instances
EBS
RDS
Tasks
Edit Application
Delete Application
Edit Application Security Access

Application Details

Name:	cass_perf_sr
	Warning: Punctuation in name prevents use as frontend service.
Type:	Web Service
Description:	Single region performance test
Owner:	Adrian
Email:	acockcroft@netflix.com
Create Time:	2011-06-13 14:04:45 PDT
Update Time:	2011-06-13 14:04:45 PDT

Pattern Matches

Auto Scaling:	cass_perf_sr--useast1c cass_perf_sr--useast1d cass_perf_sr--useast1a
Load Balancers:	
Security Groups:	cass_perf_sr
Launch Configurations:	
Running Instances:	Running Instance List



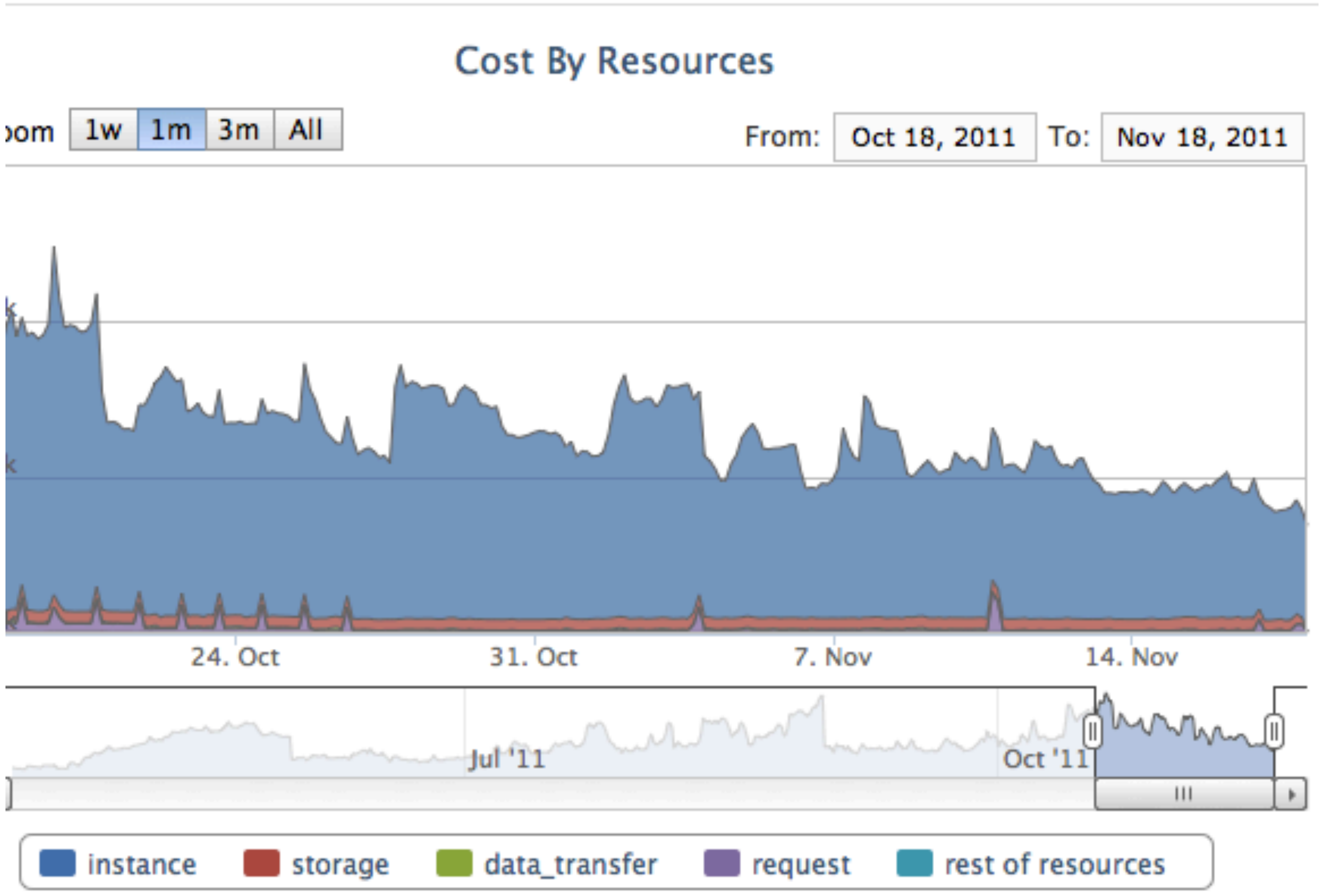
Auto Scaling Group Details

Name:	cass_perf_sr--useast1d
Launch Configuration:	cass_perf_sr--useast1d-201106131415
Application:	cass_perf_sr
Detail:	useast1d
Min Instances:	4
Desired Instances:	4
Max Instances:	4
Cool Down:	10 seconds
ASG Health Check Type:	EC2 (Replace terminated instances)
ASG Health Check Grace Period:	600 seconds
Availablility Zones:	[us-east-1d]
AZ Rebalancing:	Enabled
New Instance Launching:	Enabled
Created Time:	2011-06-13 14:15:29 PDT
Load Balancers:	–
Activities:	

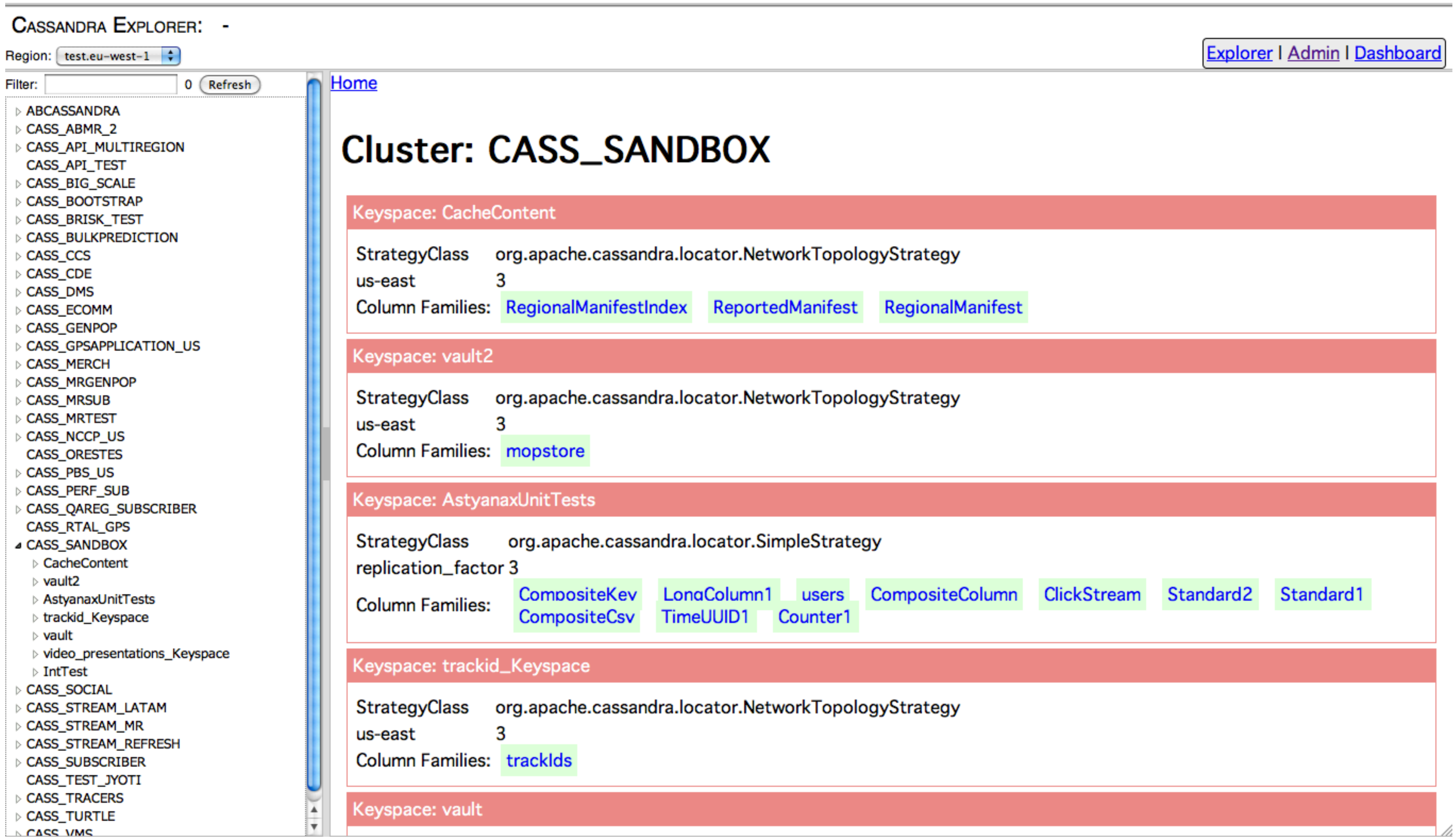
```
At 2011-06-13T21:15:29Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 4.  At 2011-06-13T2
response to a difference between desired and actual capacity, increasing the capacity from 0 to 4. : Launching a new EC2 instance:
Successful)

At 2011-06-13T21:15:29Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 4.  At 2011-06-13T2
response to a difference between desired and actual capacity, increasing the capacity from 0 to 4. : Launching a new EC2 instance:
Successful)
```

AWS Usage

for test, carefully omitting any $ numbers…



Cassandra Explorer



CASSANDRA EXPLORER: -

Region: test.eu-west-1

Explorer | Admin | Dashboard

Filter: 0 Refresh

- ABCASSANDRA
- CASS_ABMR_2
- CASS_API_MULTIREGION
- CASS_API_TEST
- CASS_BIG_SCALE
- CASS_BOOTSTRAP
- CASS_BRISK_TEST
- CASS_BULKPREDICTION
- CASS_CCS
- CASS_CDE
- CASS_DMS
- CASS_ECOMM
- CASS_GENPOP
- CASS_GPSAPPLICATION_US
- CASS_MERCH
- CASS_MRGENPOP
- CASS_MRSUB
- CASS_MRTEST
- CASS_NCCP_US
- CASS_ORESTES
- CASS_PBS_US
- CASS_PERF_SUB
- CASS_QAREG_SUBSCRIBER
- CASS_RTAL_GPS
- CASS_SANDBOX
  - CacheContent
  - vault2
  - AstyanaxUnitTests
  - trackid_Keyspace
  - vault
  - video_presentations_Keyspace
  - IntTest
- CASS_SOCIAL
- CASS_STREAM_LATAM
- CASS_STREAM_MR
- CASS_STREAM_REFRESH
- CASS_SUBSCRIBER
- CASS_TEST_JYOTI
- CASS_TRACERS
- CASS_TURTLE
- CASS_VMS

Home

Cluster: CASS_SANDBOX

Keyspace: CacheContent

StrategyClass	org.apache.cassandra.locator.NetworkTopologyStrategy
us-east	3
Column Families:	RegionalManifestIndex ReportedManifest RegionalManifest

Keyspace: vault2

StrategyClass	org.apache.cassandra.locator.NetworkTopologyStrategy
us-east	3
Column Families:	mopstore

Keyspace: AstyanaxUnitTests

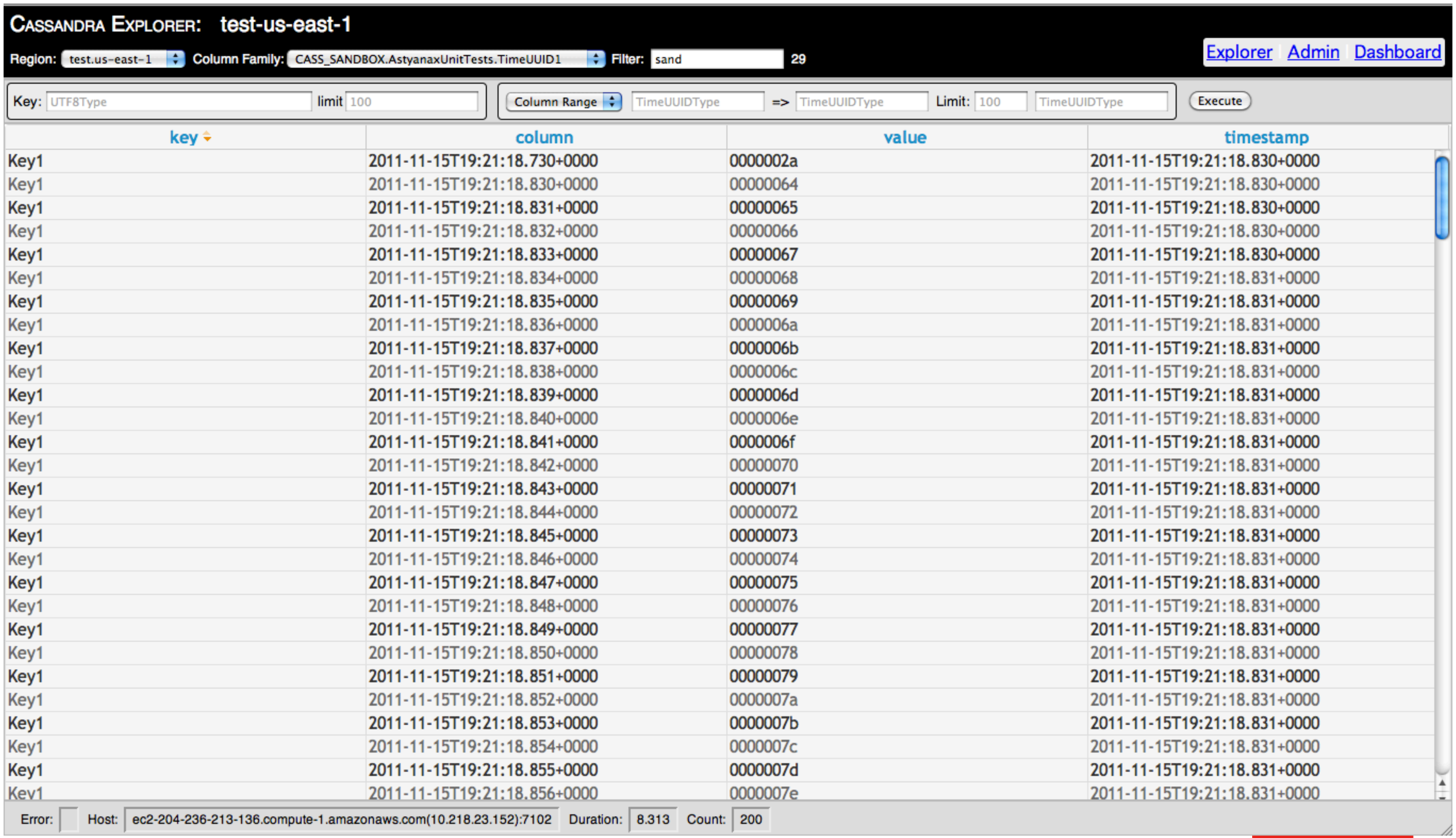
StrategyClass	org.apache.cassandra.locator.SimpleStrategy
replication_factor	3
Column Families:	CompositeKey LongColumn1 users CompositeColumn ClickStream Standard2 Standard1 CompositeCsv TimeUUID1 Counter1

Keyspace: trackid_Keyspace

StrategyClass	org.apache.cassandra.locator.NetworkTopologyStrategy
us-east	3
Column Families:	trackIds

Keyspace: vault

Cassandra Explorer



CASSANDRA EXPLORER: test-us-east-1

Region: test.us-east-1 Column Family: CASS_SANDBOX.AstyanaxUnitTests.TimeUUID1 Filter: sand 29

Key: UTF8Type limit 100 Column Range TimeUUIDType => TimeUUIDType Limit: 100 TimeUUIDType

key	column	value	timestamp
Key1	2011-11-15T19:21:18.730+0000	0000002a	2011-11-15T19:21:18.830+0000
Key1	2011-11-15T19:21:18.830+0000	00000064	2011-11-15T19:21:18.830+0000
Key1	2011-11-15T19:21:18.831+0000	00000065	2011-11-15T19:21:18.830+0000
Key1	2011-11-15T19:21:18.832+0000	00000066	2011-11-15T19:21:18.830+0000
Key1	2011-11-15T19:21:18.833+0000	00000067	2011-11-15T19:21:18.830+0000
Key1	2011-11-15T19:21:18.834+0000	00000068	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.835+0000	00000069	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.836+0000	0000006a	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.837+0000	0000006b	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.838+0000	0000006c	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.839+0000	0000006d	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.840+0000	0000006e	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.841+0000	0000006f	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.842+0000	00000070	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.843+0000	00000071	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.844+0000	00000072	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.845+0000	00000073	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.846+0000	00000074	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.847+0000	00000075	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.848+0000	00000076	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.849+0000	00000077	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.850+0000	00000078	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.851+0000	00000079	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.852+0000	0000007a	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.853+0000	0000007b	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.854+0000	0000007c	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.855+0000	0000007d	2011-11-15T19:21:18.831+0000
Key1	2011-11-15T19:21:18.856+0000	0000007e	2011-11-15T19:21:18.831+0000

Error: Host: ec2-204-236-213-136.compute-1.amazonaws.com(10.218.23.152):7102 Duration: 8.313 Count: 200

Platform Services

- Discovery – service registry for “applications”
- Introspection – Entrypoints
- Cryptex – Dynamic security key management
- Geo – Geographic IP lookup
- Platformservice – Dynamic property configuration
- Localization – manage and lookup local translations
- Evcache – eccentric volatile (mem)cached
- Cassandra – Persistence
- Zookeeper - Coordination
- Various proxies – access to old datacenter stuff

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

Entrypoints Queries

MongoDB is good 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())

Metrics Framework

- System and Application
  - Collection, Aggregation, Querying and Reporting
  - Non-blocking logging, avoids log4j lock contention
  - Chukwa -> S3 -> EMR -> Hive
- Performance, Robustness, Monitoring, Analysis
  - Tracers, Counters – explicit code instrumentation log
  - Real Time Tracers/Counters
  - SLA – service level response time percentiles
  - Epic (@MonitoredResources) annotated JMX extract
- Latency Monkey Infrastructure
  - Inject random delays into service responses

Configuration Management

- NetflixConfiguration
  - Validation Framework
  - Sitewide Properties Explorer
- PlatformService
- Mapping Service
- ZooKeeper (Curator)
- InstanceIdentity

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, migrating to Cassandra now
  - NFSDB - Instrumented wrapper library
  - Domain and Item sharding (workarounds)
- S3
  - Upgraded/Instrumented JetS3t based interface
  - Supports multipart upload and large files
  - Global S3 endpoint management

Netflix Platform Persistence

- Eccentric 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

Aside: Adrian's Rant on CAP Theorem

- Instances and Networks *will fail*
- Network failure = Partition "P" is a given
- Distributed Systems: two choices – CP or AP
- "Vendor claims CA" **!!Bullshit detector!!**
  - Usually they mean available when instances fail
- Master-Slave = Consistent when Partitioned
  - You **can't write** unless you can see the master
- Quorum = Available when Partitioned
  - Writes proceed, conflicts will be patched up later

NETFLIX

Why Cassandra?

- We value Availability over Consistency – AP
  - Cassandra is a Java distributed systems toolkit
- We have a building full of Java engineers
  - Riak is in Erlang – a blessing and a curse…
- We want FOSS + Support
  - Voldemort doesn't have a support model
- Writes are intrinsically harder than reads
  - Hbase is optimized for reads, Cassandra for writes
- We tested Cassandra and it works for us
  - Step by step into full production over the last year

Priam – Cassandra Automation

Coming soon to 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

Astyanax

Coming soon to 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
  - Discovery integration
  - Host reconnect (fixed interval or exponential backoff)
  - Token aware (in development) to save a network hop
- Netflix style configuration (INFLibrary)
- 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

Initializing Astyanax

```
// Configuration either set in code or nfastyanax.properties
platform.ListOfComponentsToInit=LOGGING,APPINFO,DISCOVERY
netflix.environment=test
default.astyanax.readConsistency=CL_QUORUM
default.astyanax.writeConsistency=CL_QUORUM
MyCluster.MyKeyspace.astyanax.servers=127.0.0.1

// Must initialize platform for discovery to work
NFLibraryManager.initLibrary(PlatformManager.class, props, false, true);
NFLibraryManager.initLibrary(NFAstyanaxManager.class, props, true, false);

// Open a keyspace instance
Keyspace keyspace = KeyspaceFactory.openKeyspace(”MyCluster”,”MyKeyspace");
```

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) {
}
```

Data Migration to Cassandra

Distributed Key-Value Stores

- 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

DBA

- No schema to change, no scheduled downtime
- Latency for typical queries
  - Memcached is dominated by network latency <1ms
  - Cassandra takes a few milliseconds
  - SimpleDB replication and REST auth overheads >10ms

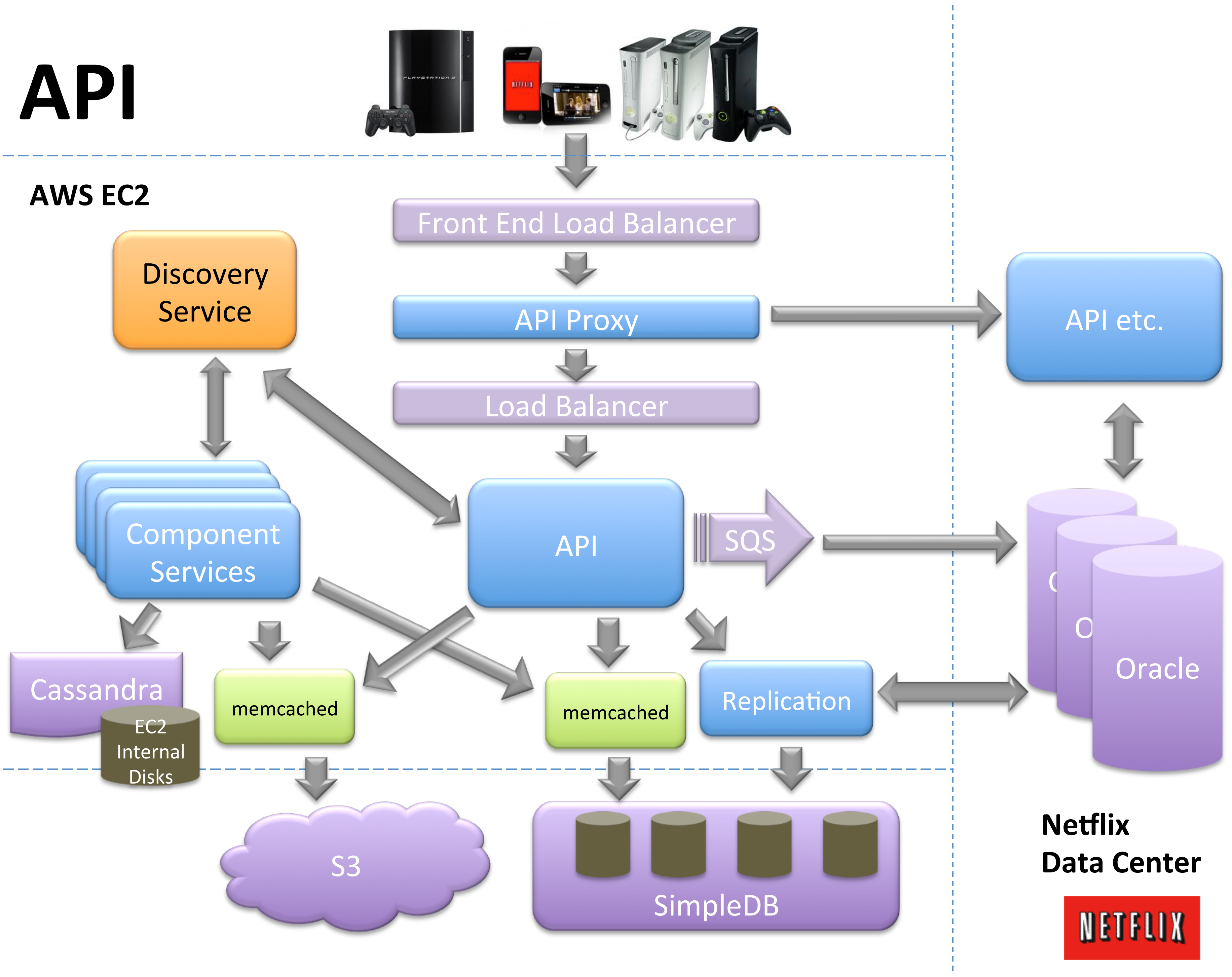
Multi-Regional Data Replication

- IR Framework – Datacenter Item Replicator
  - Built in 2009, first step to the cloud
  - Oracle to SimpleDB or Cassandra via poll and push
  - Return updates to Oracle via SQS message queue
- SimpleDB or S3 to Cassandra
  - Data migration tool for global Netflix
- Global SimpleDB and S3 Replication
  - Cross region async updates USA to Europe

Transitional Steps

- Bidirectional Replication
  - Oracle to SimpleDB
  - Queued reverse path using SQS
  - Backups remain in Datacenter via Oracle
- New Cloud-Only Data Sources
  - Cassandra based
  - No replication to Datacenter
  - Backups performed in the cloud

API

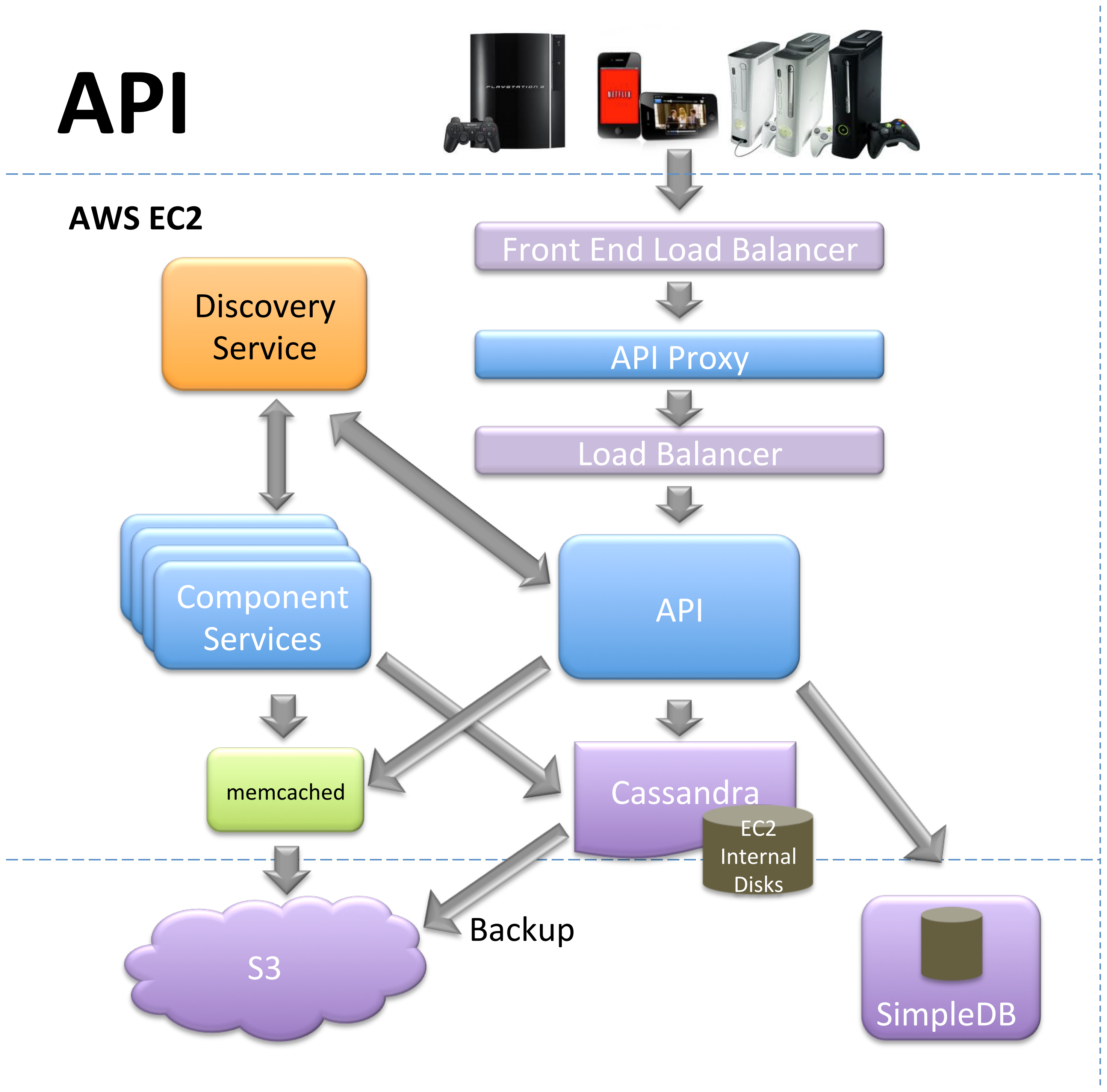


AWS EC2
Front End Load Balancer
Discovery Service
API Proxy
API etc.
Load Balancer
Component Services
API
SQS
Oracle
Cassandra
EC2 Internal Disks
memcached
memcached
Replication
S3
SimpleDB
Netflix Data Center

Cutting the Umbilical

- Transition Oracle Data Sources to Cassandra
  - Offload Datacenter Oracle hardware
  - Free up capacity for growth of remaining services
- Transition SimpleDB+Memcached to Cassandra
  - Primary data sources that need backup
  - Keep simplest small use cases for now
- New challenges
  - Backup, restore, archive, business continuity
  - Business Intelligence integration

API



AWS EC2
Front End Load Balancer
API Proxy
Load Balancer
API
Discovery Service
Component Services
memcached
Cassandra
EC2 Internal Disks
S3
Backup
SimpleDB

High Availability

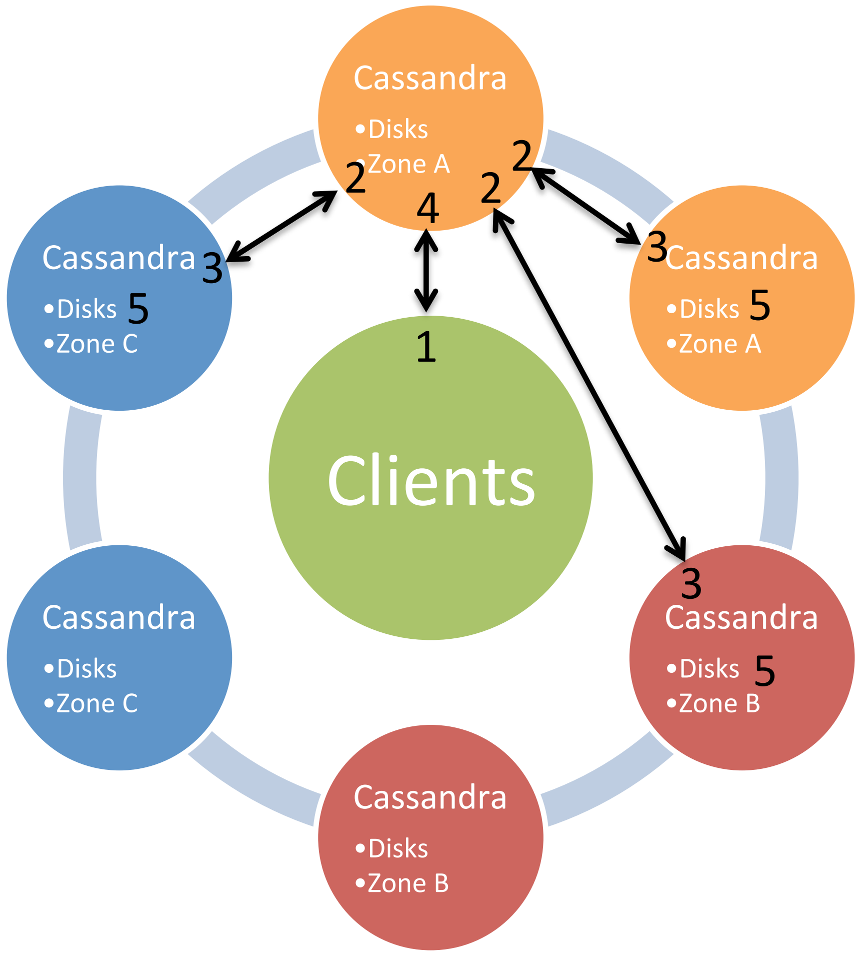
- Cassandra stores 3 local copies, 1 per zone
  - Synchronous access, durable, highly available
  - Read/Write One fastest, least consistent - ~1ms
  - Read/Write Quorum 2 of 3, consistent - ~3ms
- AWS Availability Zones
  - Separate buildings
  - Separate power etc.
  - Fairly close together

NETFLIX

Cassandra Write Data Flows

Single Region, Multiple Availability Zone

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



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

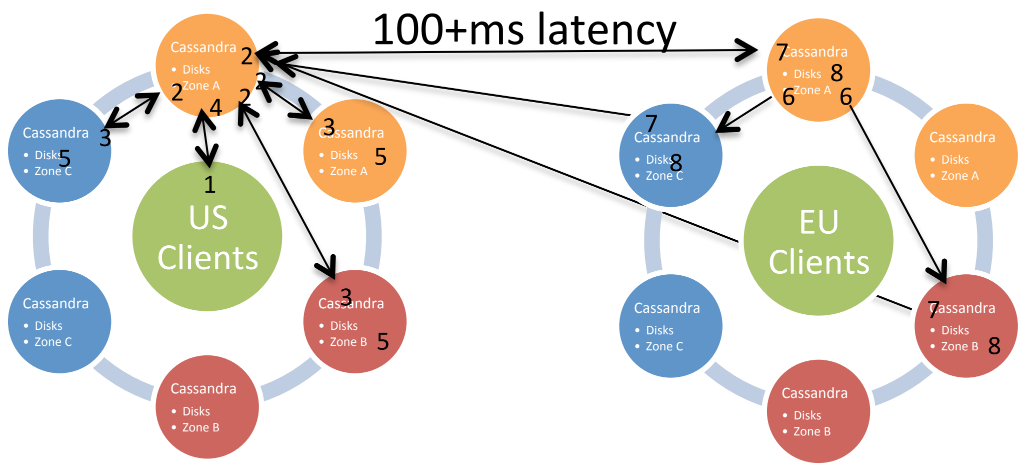
SSTable disk writes and compactions occur asynchronously

Data Flows for Multi-Region Writes

Consistency Level = Local Quorum

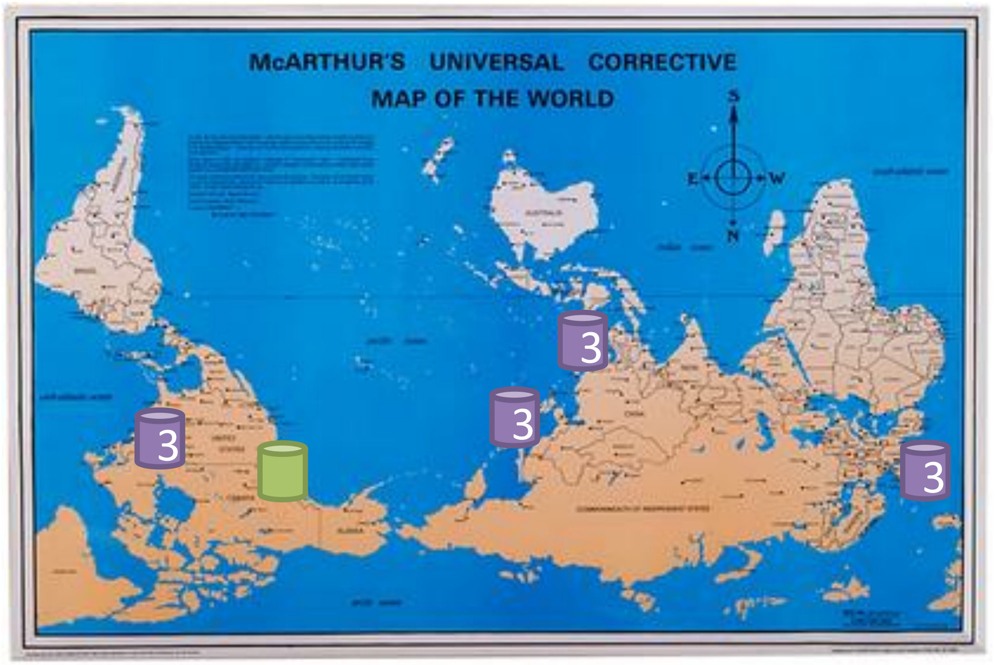
1. Client Writes to any Cassandra Node
2. Coordinator node replicates to other nodes Zones and regions
3. Local write acks returned to coordinator
4. Client gets ack when 2 of 3 local nodes are committed
5. Data written to internal commit log disks
6. When data arrives, remote node replicates data
7. Ack direct to source region coordinator
8. Remote copies written to commit log disks

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.



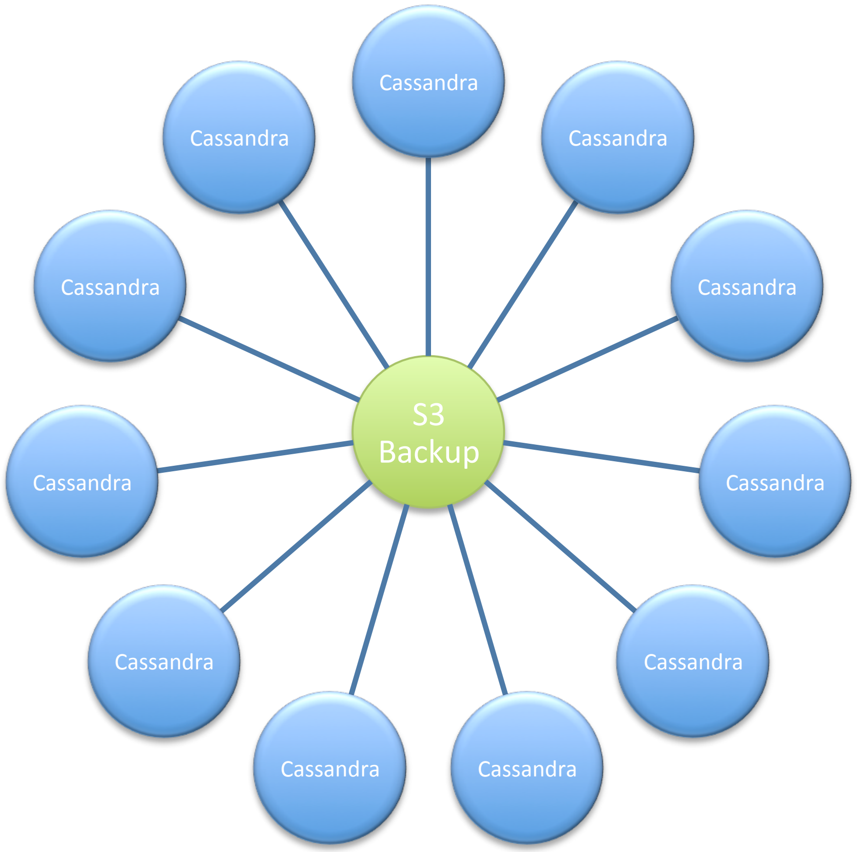
Remote Copies

- Cassandra duplicates across AWS regions
  - Asynchronous write, replicates at destination
  - Doesn't directly affect local read/write latency
- Global Coverage
  - Business agility
  - Follow AWS…
- Local Access
  - Better latency
  - Fault Isolation



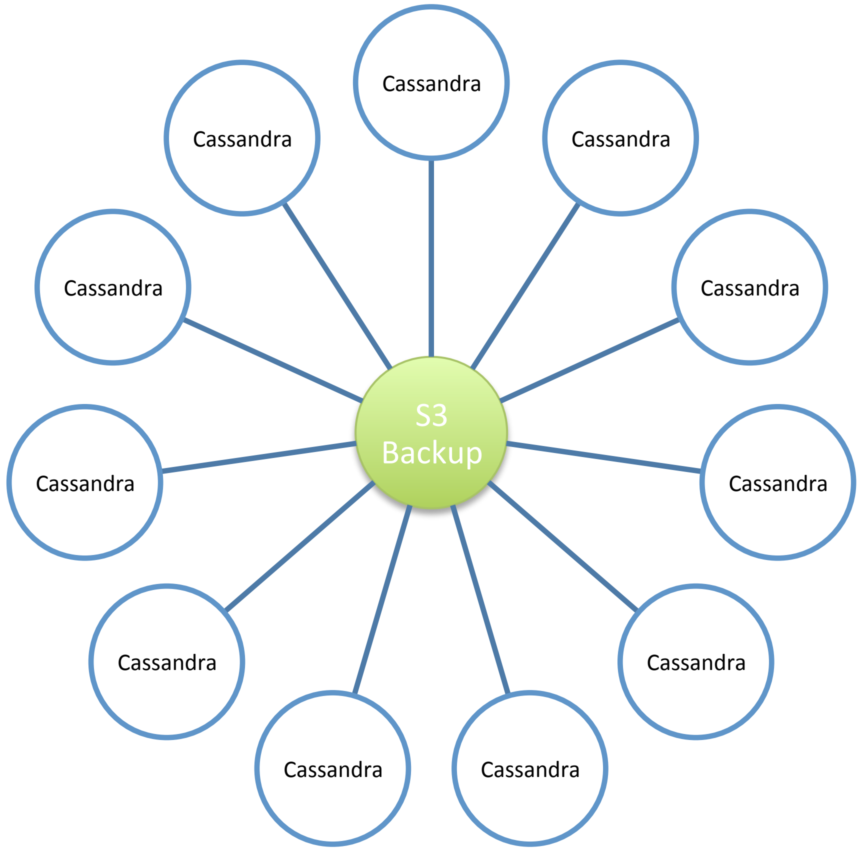
Cassandra Backup

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



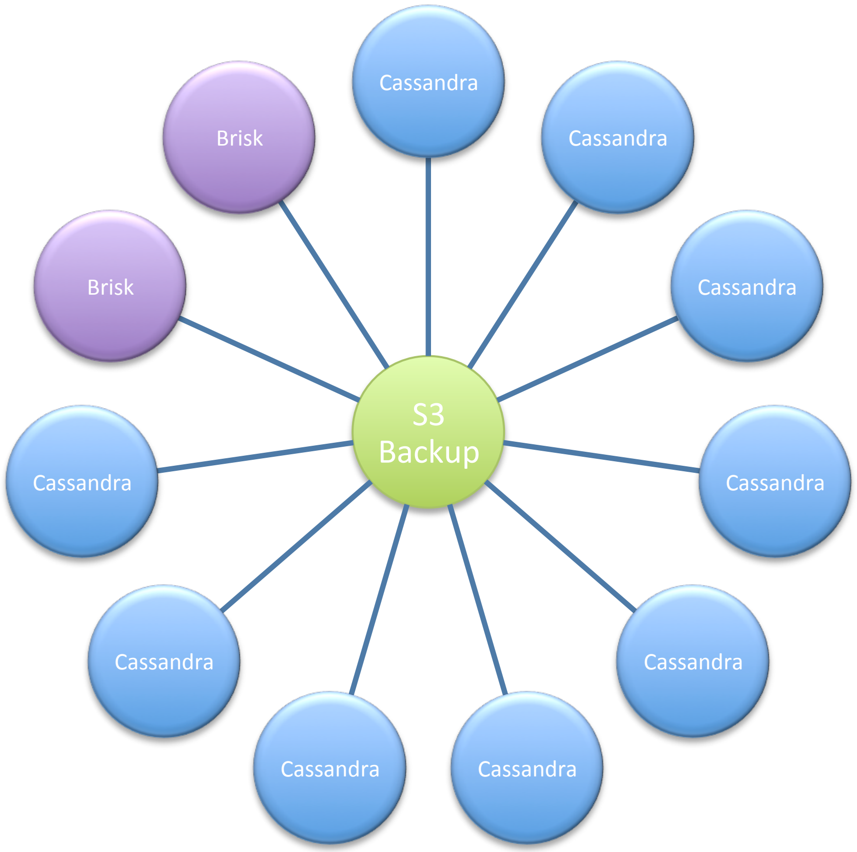
Cassandra Restore

- Full Restore
  - Replace previous data
- New Ring from Backup
  - New name old data
- Scripted
  - Create new instances
  - Parallel load - fast



Cassandra Online Analytics

- Brisk = Hadoop + Cass
  - Use split Brisk ring
  - Size each separately
- Direct Access
  - Keyspaces
  - Hive/Pig/Map-Reduce
  - Hdfs as a keyspace
  - Distributed namenode



NETFLIX

Cassandra Archive

Appropriate level of paranoia needed…

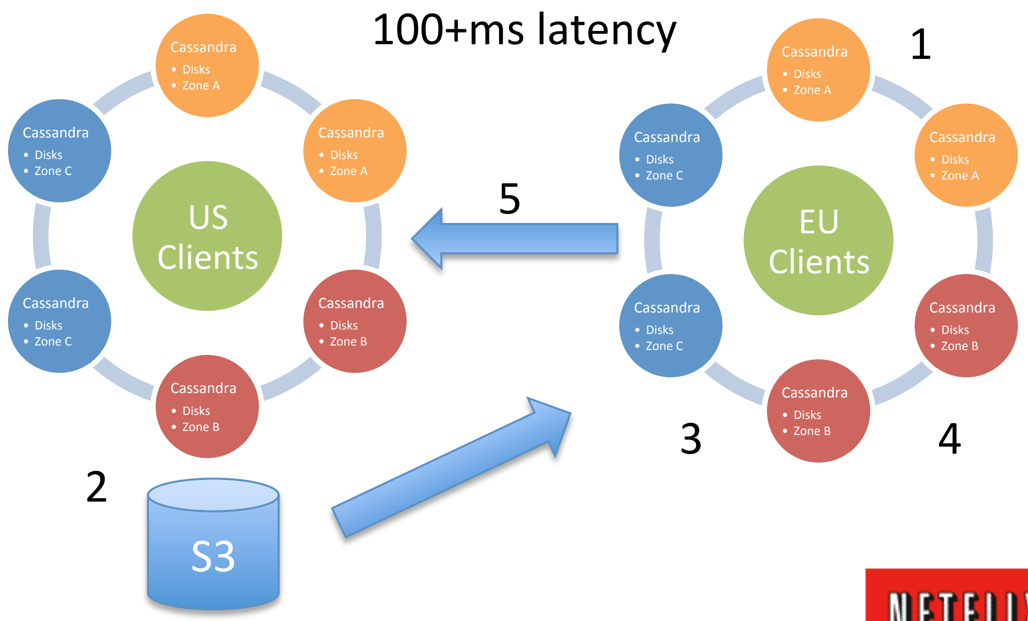
- Archive could be un-readable
  - Restore S3 backups weekly from prod to test
- Archive could be stolen
  - PGP Encrypt archive
- AWS East Region could have a problem
  - Copy data to AWS West
- Production AWS Account could have an issue
  - Separate Archive account with no-delete S3 ACL
- AWS S3 could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

Extending to Multi-Region

In production last week for UK/Eire support!

1. Create cluster in EU
2. Backup US cluster to S3
3. Restore backup in EU
4. Local repair EU cluster
5. Global repair/join

Take a Boeing 737 on a domestic flight, upgrade it to a 747 by adding more engines and fly it to Europe without landing it on the way…



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, OpenJDK, CentOS
  - Datastax support for Cassandra, AWS support for Hadoop via EMR
- Monitoring Tools
  - Datastax Opscenter for monitoring Cassandra
  - AppDynamics – Developer focus for cloud http://appdynamics.com

Developer Migration

- Detailed SQL to NoSQL Transition Advice
  - Sid Anand - QConSF Nov 5th – Netflix' Transition to High Availability Storage Systems
  - Blog - http://practicalcloudcomputing.com/
  - Download Paper PDF - http://bit.ly/bhOTLu
- Mark Atwood, "Guide to NoSQL, redux"
  - YouTube http://youtu.be/zAbFRiyT3LU

Cloud Operations

Cassandra Use Cases

Model Driven Architecture

Performance and Scalability

Cassandra Use Cases

- Key by Customer – Cross-region clusters
  - Many app specific Cassandra clusters, read-intensive
  - Keys+Rows in memory using m2.4xl Instances
- Key by Customer:Movie – e.g. Viewing History
  - Growing fast, write intensive – m1.xl instances
  - Keys cached in memory, one cluster per region
- Large scale data logging – lots of writes
  - Column data expires after time period
  - Distributed counters, one cluster per region

Model Driven Architecture

- Datacenter Practices
  - Lots of unique hand-tweaked systems
  - Hard to enforce patterns

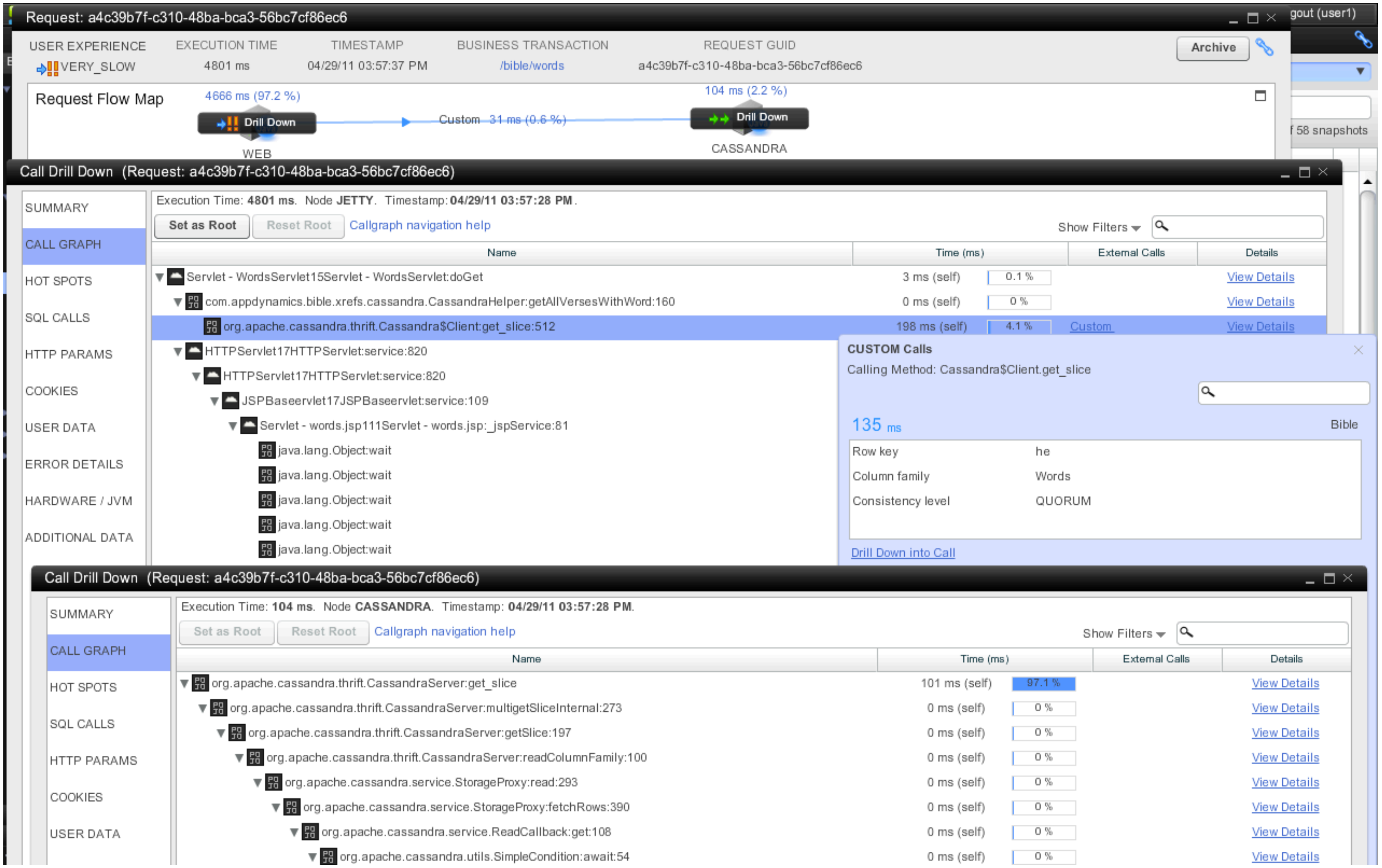
- Model Driven Cloud Architecture
  - Perforce/Ivy/Jenkins based builds for *everything*
  - Every production instance is a pre-baked AMI
  - Every application is managed by an Autoscaler

Every change is a new AMI

Chaos Monkey

- 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
- Computers (Datacenter or AWS) randomly die
  - Fact of life, but too infrequent to test resiliency

AppDynamics Monitoring of Cassandra – Automatic Discovery



Request: a4c39b7f-c310-48ba-bca3-56bc7cf86ec6

USER EXPERIENCE	EXECUTION TIME	TIMESTAMP	BUSINESS TRANSACTION	REQUEST GUID
VERY_SLOW	4801 ms	04/29/11 03:57:37 PM	/bible/words	a4c39b7f-c310-48ba-bca3-56bc7cf86ec6

Call Drill Down (Request: a4c39b7f-c310-48ba-bca3-56bc7cf86ec6)

Execution Time: **4801 ms**. Node **JETTY**. Timestamp: **04/29/11 03:57:28 PM**.

Name	Time (ms)		External Calls	Details
Servlet - WordsServlet15Servlet - WordsServlet:doGet	3 ms (self)	0.1 %		View Details
com.appdynamics.bible.xrefs.cassandra.CassandraHelper:getAllVersesWithWord:160	0 ms (self)	0 %		View Details
org.apache.cassandra.thrift.Cassandra$Client:get_slice:512	198 ms (self)	4.1 %	Custom	View Details
HTTPServlet17HTTPServlet:service:820				
HTTPServlet17HTTPServlet:service:820				
JSPBaseervlet17JSPBaseervlet:service:109				
Servlet - words.jsp111Servlet - words.jsp:_jspService:81				
java.lang.Object:wait				
java.lang.Object:wait				
java.lang.Object:wait				
java.lang.Object:wait				
java.lang.Object:wait				

CUSTOM Calls

Calling Method: Cassandra$Client.get_slice

135 ms Bible

Row key	he
Column family	Words
Consistency level	QUORUM

Drill Down into Call

Call Drill Down (Request: a4c39b7f-c310-48ba-bca3-56bc7cf86ec6)

Execution Time: **104 ms**. Node **CASSANDRA**. Timestamp: **04/29/11 03:57:28 PM**.

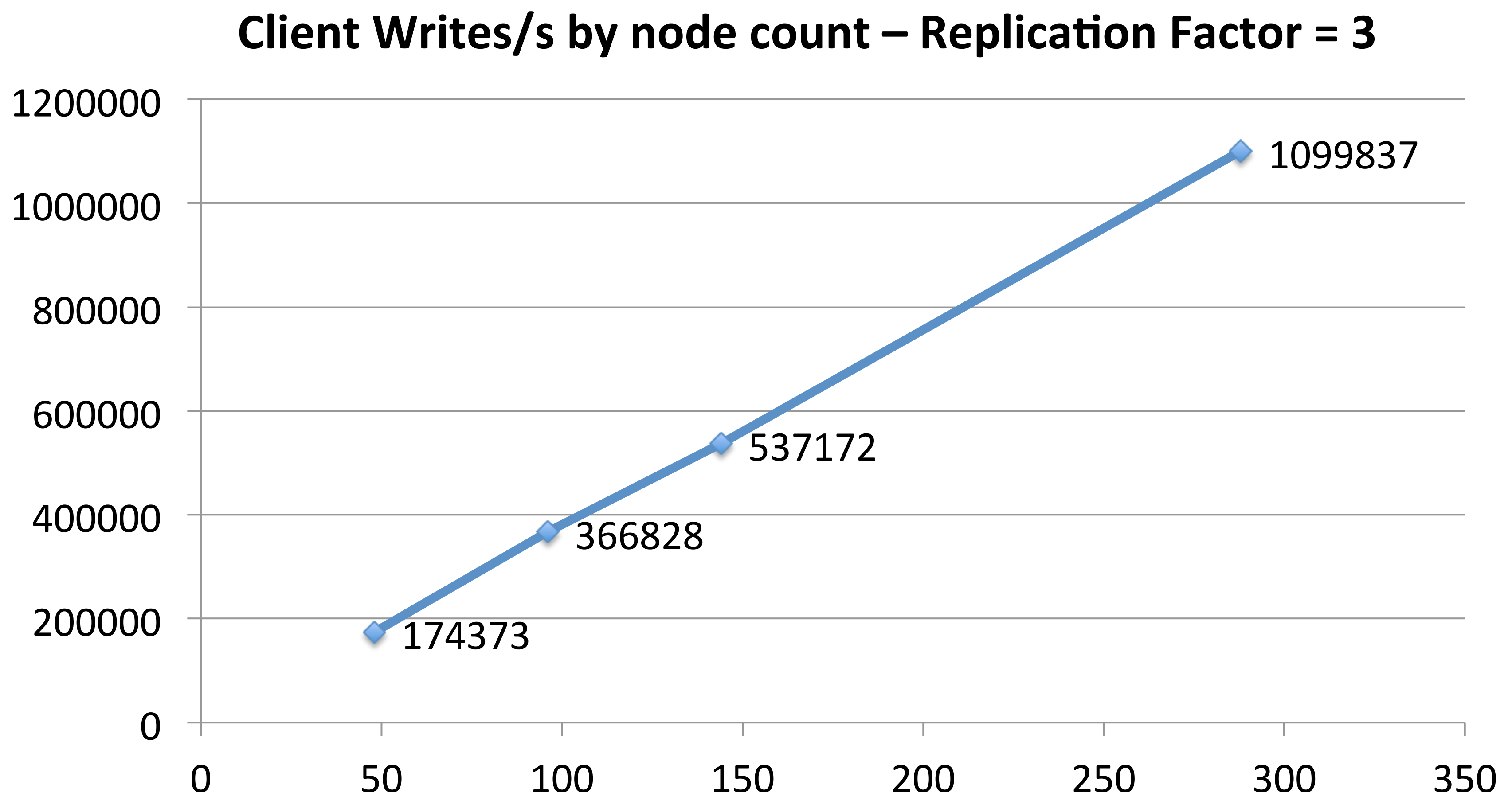
Name	Time (ms)		External Calls	Details
org.apache.cassandra.thrift.CassandraServer:get_slice	101 ms (self)	97.1 %		View Details
org.apache.cassandra.thrift.CassandraServer:multigetSliceInternal:273	0 ms (self)	0 %		View Details
org.apache.cassandra.thrift.CassandraServer:getSlice:197	0 ms (self)	0 %		View Details
org.apache.cassandra.thrift.CassandraServer:readColumnFamily:100	0 ms (self)	0 %		View Details
org.apache.cassandra.service.StorageProxy:read:293	0 ms (self)	0 %		View Details
org.apache.cassandra.service.StorageProxy:fetchRows:390	0 ms (self)	0 %		View Details
org.apache.cassandra.service.ReadCallback:get:108	0 ms (self)	0 %		View Details
org.apache.cassandra.utils.SimpleCondition:await:54	0 ms (self)	0 %		View Details

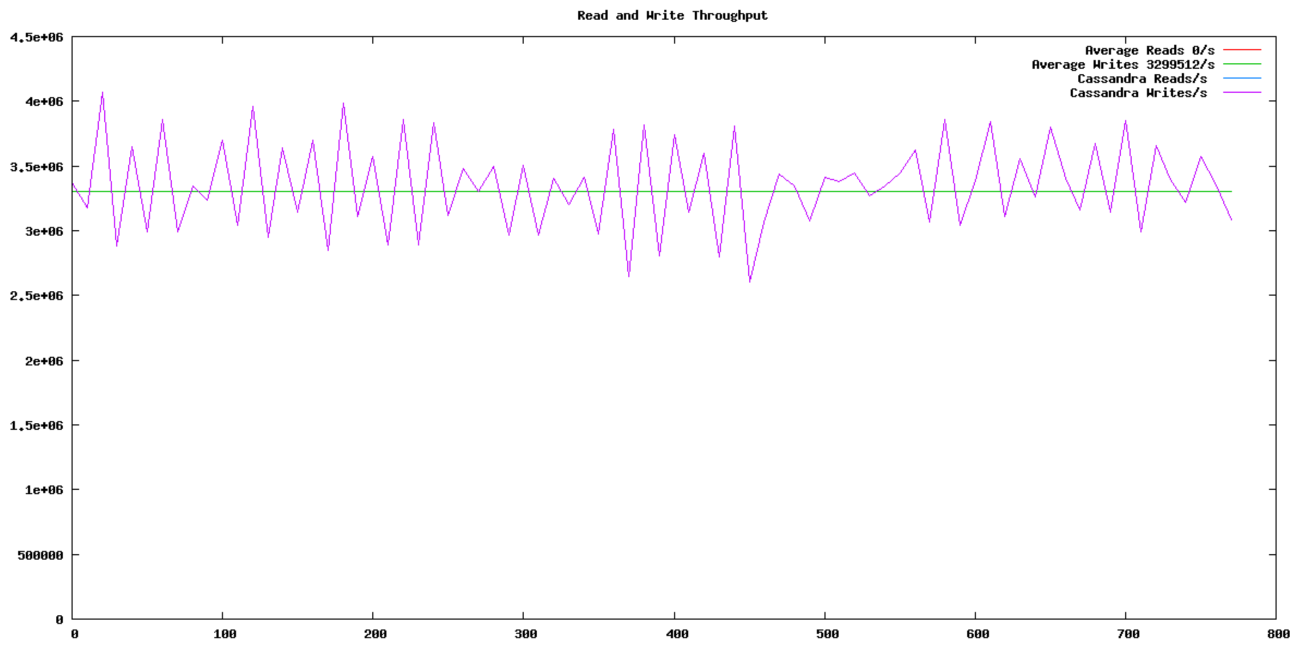
Scalability Testing

- 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?

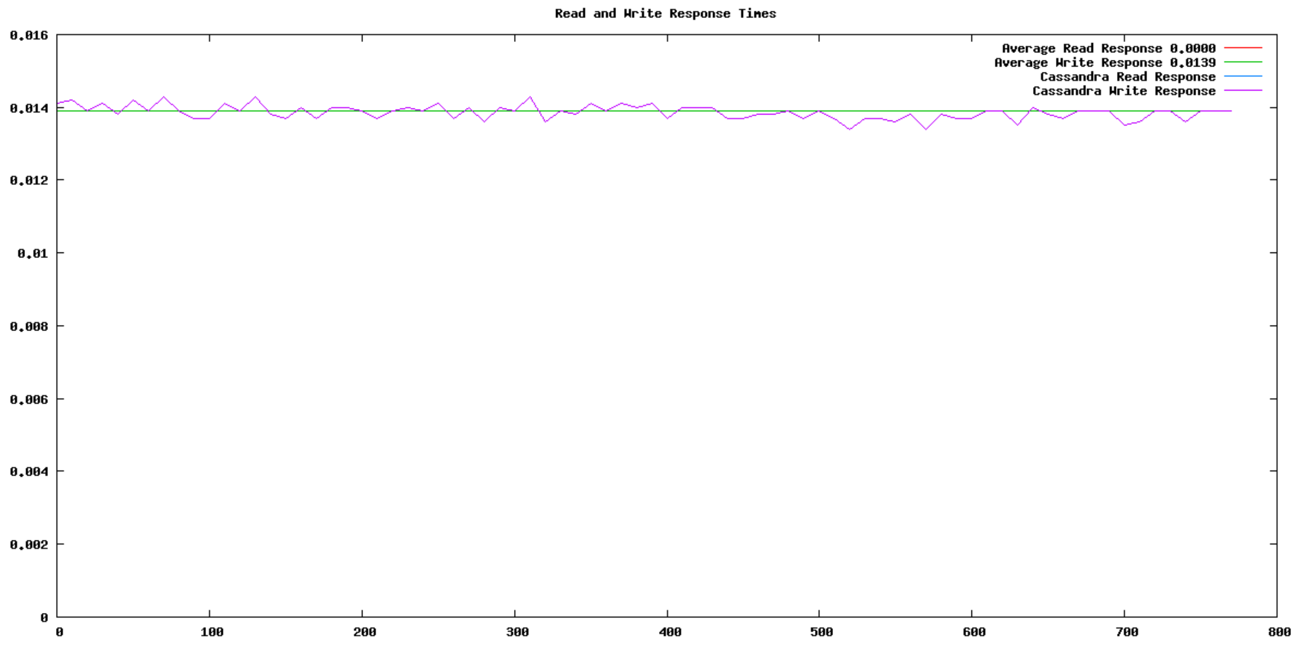
<DrEvil>ONE MILLION</DrEvil>

Scale-Up Linearity



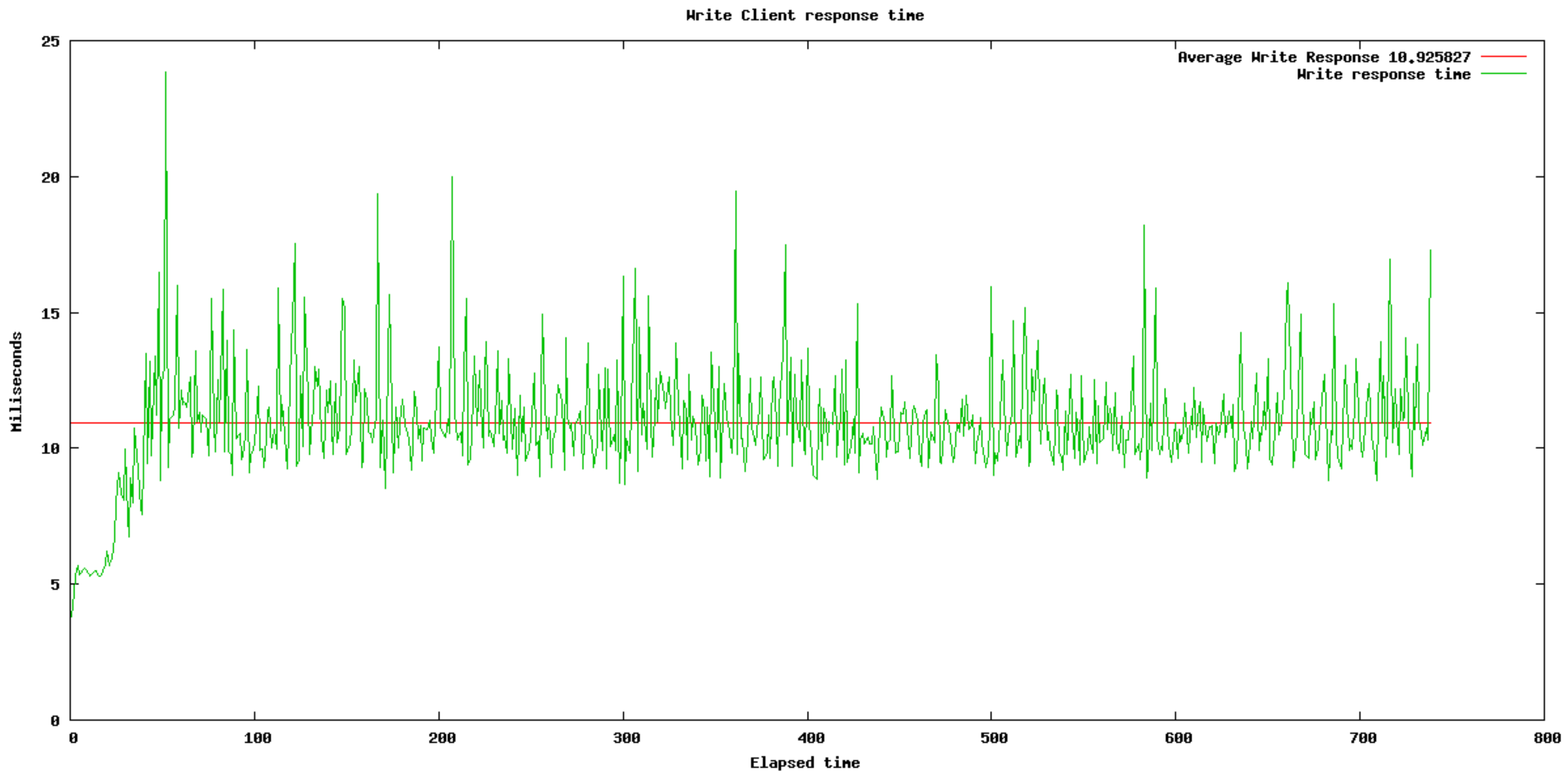


Read and Write Throughput
Average Reads 0/s
Average Writes 3299512/s
Cassandra Reads/s
Cassandra Writes/s
4.5e+06
4e+06
3.5e+06
3e+06
2.5e+06
2e+06
1.5e+06
1e+06
500000
0
0
100
200
300
400
500
600
700
800



Read and Write Response Times
Average Read Response 0.0000
Average Write Response 0.0139
Cassandra Read Response
Cassandra Write Response
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0
100
200
300
400
500
600
700
800

NETFLIX


Write Client response time
Average Write Response 10.925827
Write response time
Miliseconds
Elapsed time
0
5
10
15
20
25
0
100
200
300
400
500
600
700
800

Per Node Activity

Per Node	48 Nodes	96 Nodes	144 Nodes	288 Nodes
Per Server Writes/s	10,900 w/s	11,460 w/s	11,900 w/s	11,456 w/s
Mean Server Latency	0.0117 ms	0.0134 ms	0.0148 ms	0.0139 ms
Mean CPU %Busy	74.4 %	75.4 %	72.5 %	81.5 %
Disk Read	5,600 KB/s	4,590 KB/s	4,060 KB/s	4,280 KB/s
Disk Write	12,800 KB/s	11,590 KB/s	10,380 KB/s	10,080 KB/s
Network Read	22,460 KB/s	23,610 KB/s	21,390 KB/s	23,640 KB/s
Network Write	18,600 KB/s	19,600 KB/s	17,810 KB/s	19,770 KB/s

Node specification – Xen Virtual Images, AWS US East, three zones

- Cassandra 0.8.6, CentOS, SunJDK6
- AWS EC2 m1 Extra Large – Standard price $ 0.68/Hour
- 15 GB RAM, 4 Cores, 1Gbit network
- 4 internal disks (total 1.6TB, striped together, md, XFS)

Time is Money

	48 nodes	96 nodes	144 nodes	288 nodes
Writes Capacity	174373 w/s	366828 w/s	537172 w/s	1,099,837 w/s
Storage Capacity	12.8 TB	25.6 TB	38.4 TB	76.8 TB
Nodes Cost/hr	$32.64	$65.28	$97.92	$195.84
Test Driver Instances	10	20	30	60
Test Driver Cost/hr	$20.00	$40.00	$60.00	$120.00
Cross AZ Traffic	5 TB/hr	10 TB/hr	15 TB/hr	30[1] TB/hr
Traffic Cost/10min	$8.33	$16.66	$25.00	$50.00
Setup Duration	15 minutes	22 minutes	31 minutes	66[2] minutes
AWS Billed Duration	1hr	1hr	1 hr	2 hr
Total Test Cost	$60.97	$121.94	$182.92	$561.68

[1] Estimate two thirds of total network traffic
[2] Workaround for a tooling bug slowed setup

Takeaway

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

Also, benchmarking in the cloud is fast, cheap and scalable

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

@adrianco #netflixcloud

acockcroft@netflix.com