CI AND CD AT SCALE

SCALING JENKINS WITH DOCKER AND APACHE MESOS

Carlos Sanchez

@csanchez csanchez.org

cloudbees®

APACHECON Europe

Watch online at carlossg.github.io/presentations

ABOUT ME

Senior Software Engineer @ CloudBees

Contributor to the Jenkins Mesos plugin and the Java Marathon client

Author of Jenkins Kubernetes plugin

Long time OSS contributor at Apache, Eclipse, Puppet,…

OUR USE CASE

Scaling Jenkins

Your mileage may vary

SCALING JENKINS

Two options:

- More build agents per master
- More masters

SCALING JENKINS: MORE BUILD AGENTS

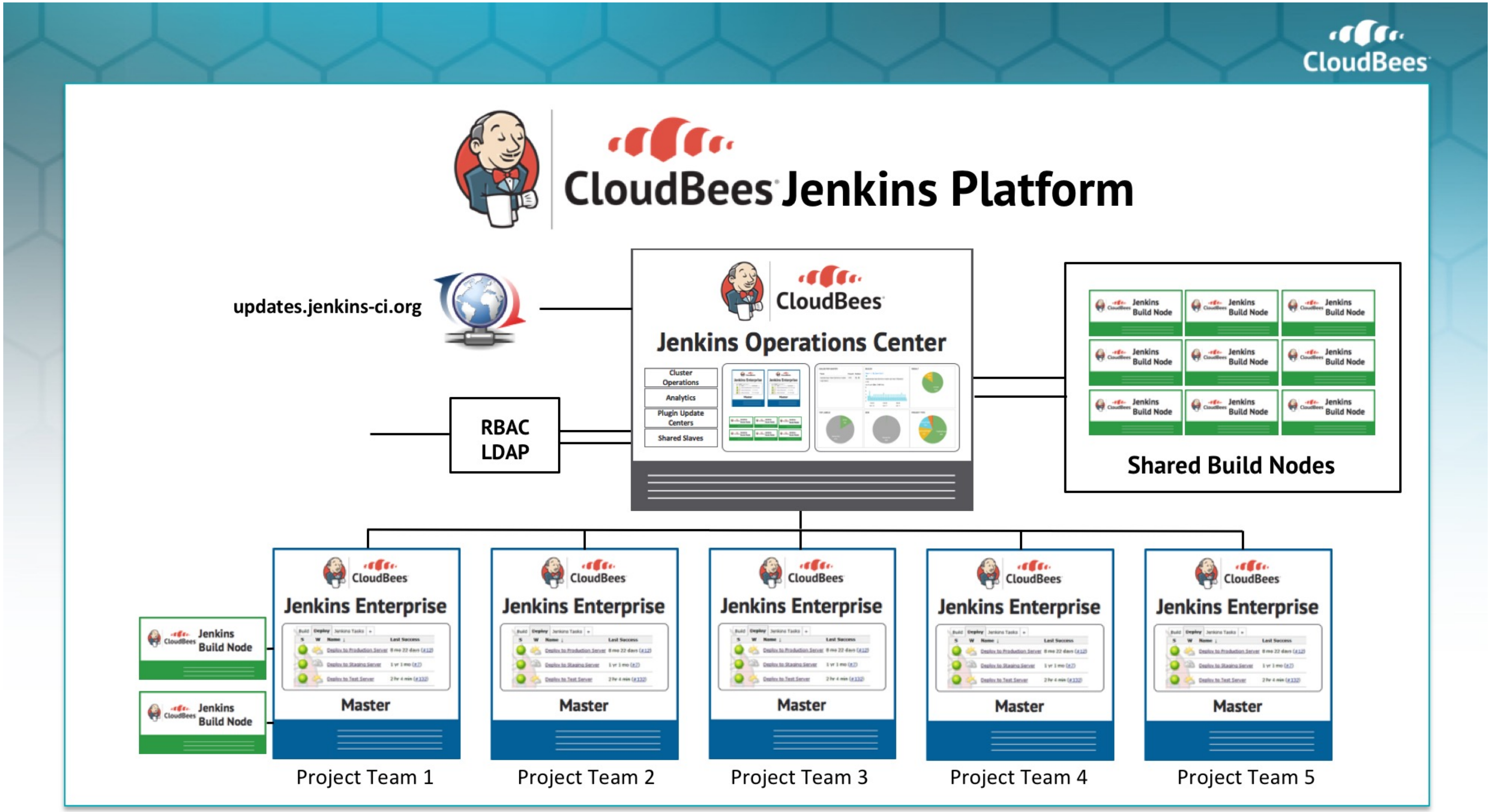
- Pros
  - Multiple plugins to add more agents, even dynamically
- Cons
  - The master is still a SPOF
  - Handling multiple configurations, plugin versions,...
  - There is a limit on how many build agents can be attached

SCALING JENKINS: MORE MASTERS

- Pros
  - Different sub-organizations can self service and operate independently
- Cons
  - Single Sign-On
  - Centralized configuration and operation

CLOUDBEES JENKINS ENTERPRISE EDITION

CloudBees Jenkins Operations Center



CLOUDBEES JENKINS PLATFORM - PRIVATE SAAS EDITION

The best of both worlds

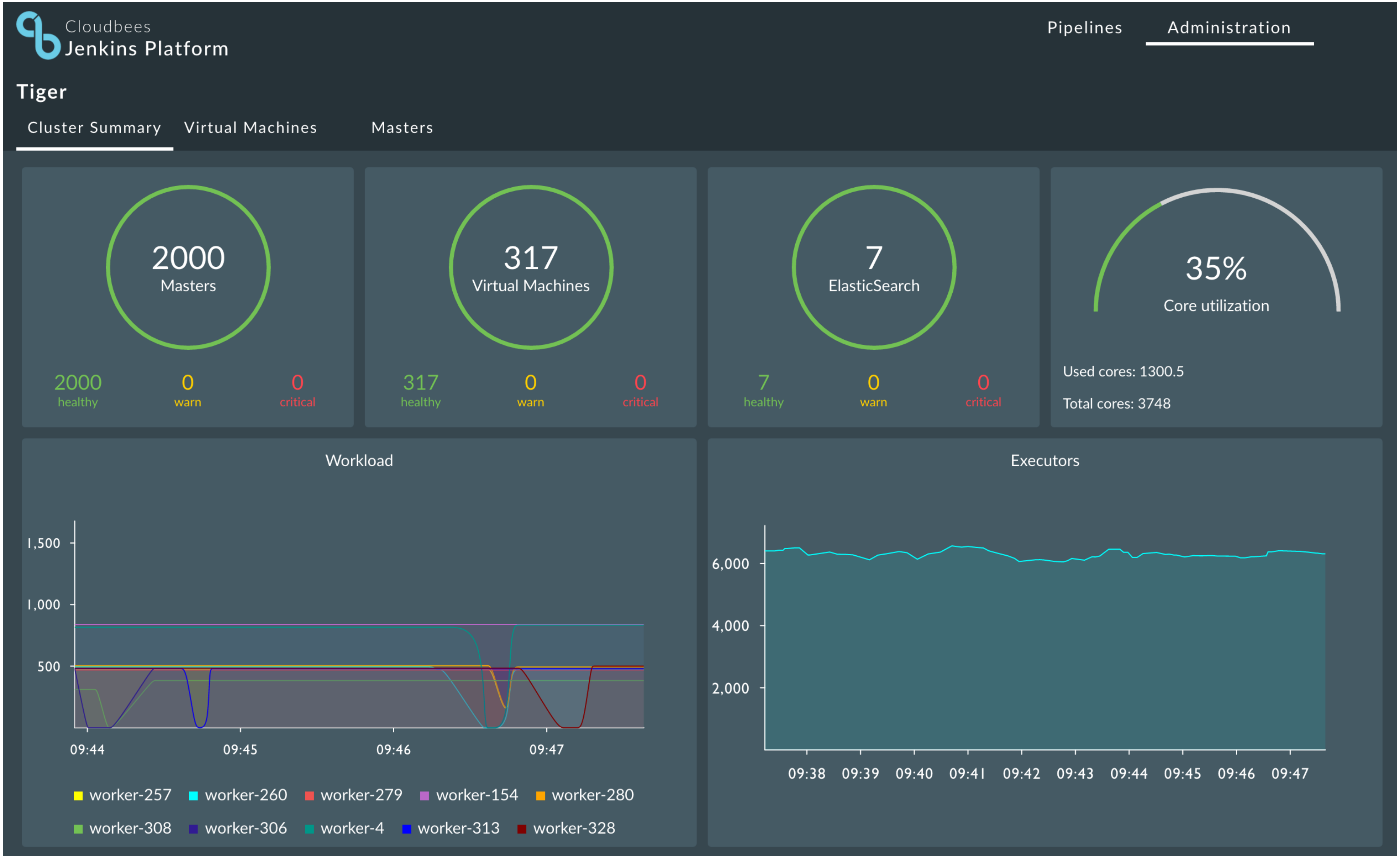
CloudBees Jenkins Operations Center with multiple masters

Dynamic build agent creation in each master

ElasticSearch for Jenkins metrics and Logstash

BUT IT IS NOT TRIVIAL

A 2000 JENKINS MASTERS CLUSTER



2000 Masters

2000 healthy · 0 warn · 0 critical

317 Virtual Machines

317 healthy · 0 warn · 0 critical

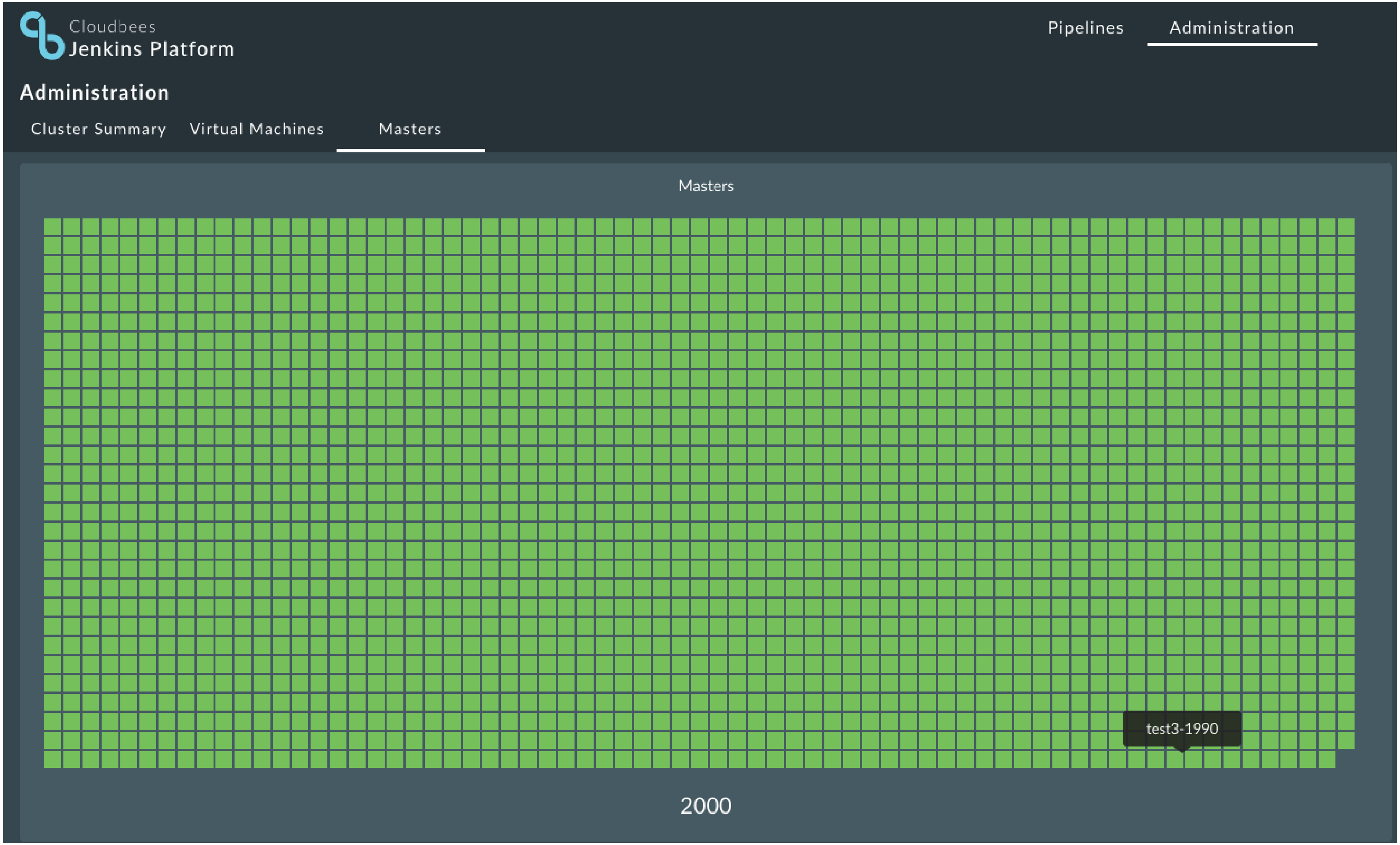
7 ElasticSearch

7 healthy · 0 warn · 0 critical

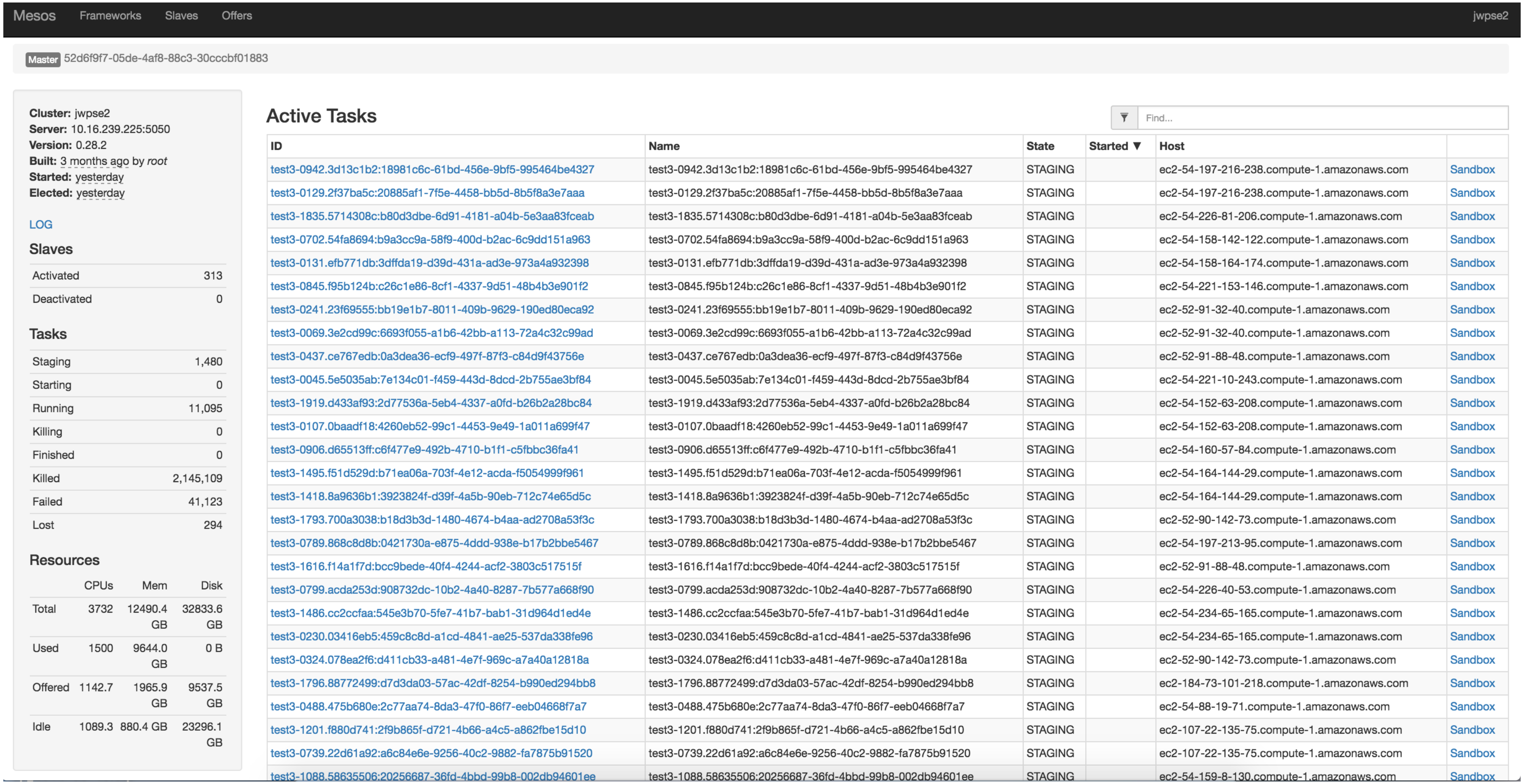
35% Core utilization

Used cores: 1300.5

Total cores: 3748



Cloudbees
Jenkins Platform
Pipelines
Administration
Administration
Cluster Summary
Virtual Machines
Masters
Masters
test3-1990
2000



Master 52d6f9f7-05de-4af8-88c3-30cccbf01883

Cluster: jwpse2
Server: 10.16.239.225:5050
Version: 0.28.2
Built: 3 months ago by *root*
Started: yesterday
Elected: yesterday

LOG

Slaves

Activated	313
Deactivated	0

Tasks

Staging	1,480
Starting	0
Running	11,095
Killing	0
Finished	0
Killed	2,145,109
Failed	41,123
Lost	294

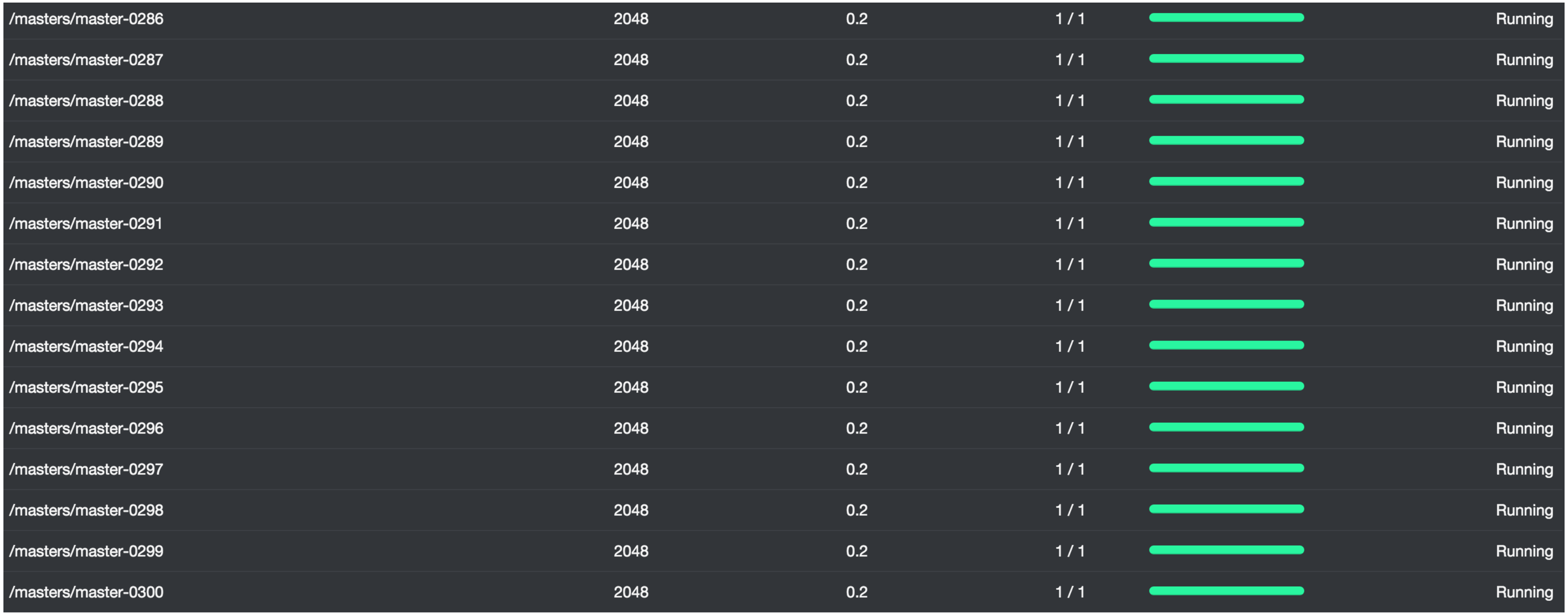
Resources

	CPUs	Mem	Disk
Total	3732	12490.4 GB	32833.6 GB
Used	1500	9644.0 GB	0 B
Offered	1142.7	1965.9 GB	9537.5 GB
Idle	1089.3	880.4 GB	23296.1 GB

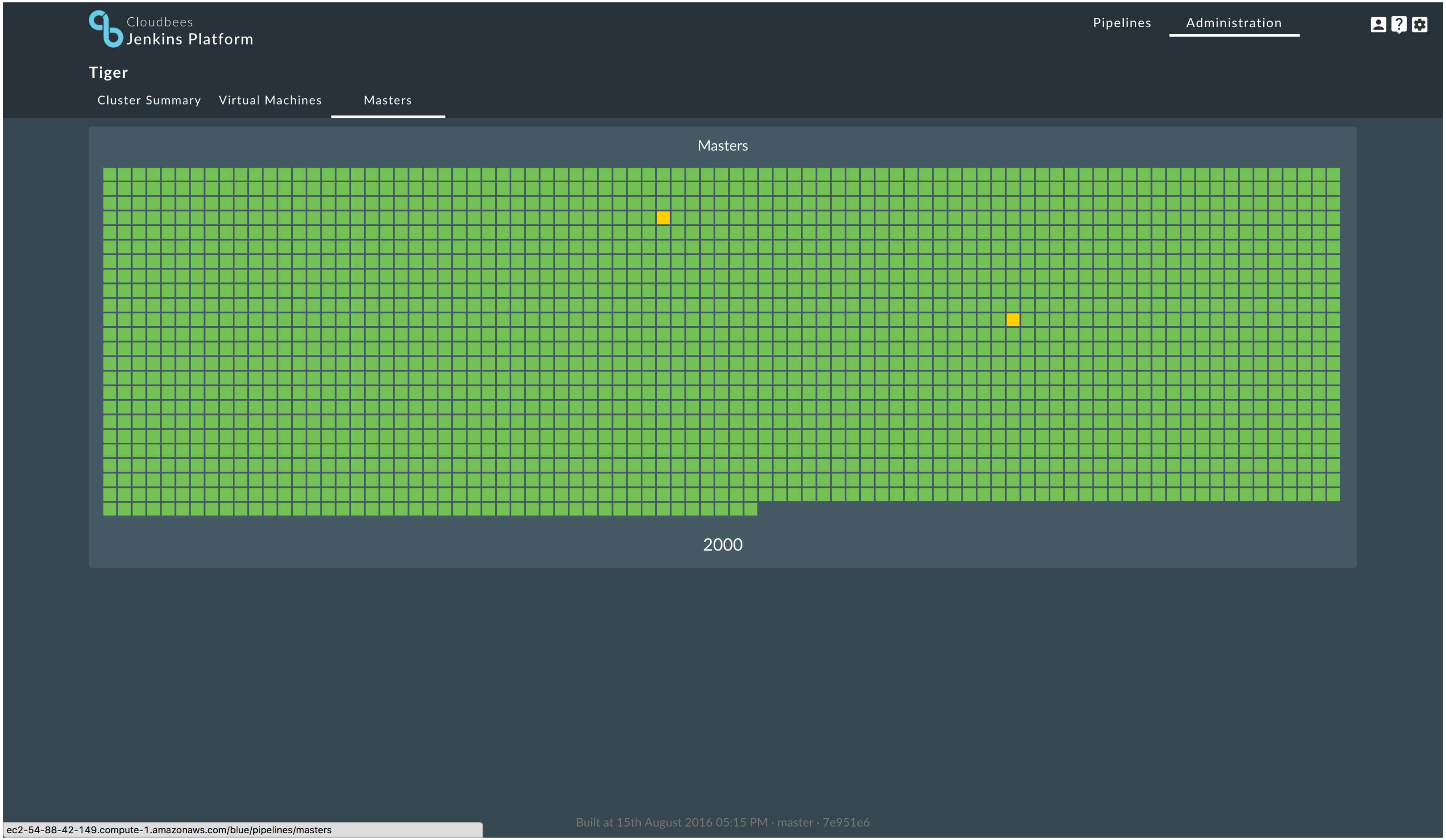
Active Tasks

Find...

ID	Name	State	Started ▼	Host	
test3-0942.3d13c1b2:18981c6c-61bd-456e-9bf5-995464be4327	test3-0942.3d13c1b2:18981c6c-61bd-456e-9bf5-995464be4327	STAGING		ec2-54-197-216-238.compute-1.amazonaws.com	Sandbox
test3-0129.2f37ba5c:20885af1-7f5e-4458-bb5d-8b5f8a3e7aaa	test3-0129.2f37ba5c:20885af1-7f5e-4458-bb5d-8b5f8a3e7aaa	STAGING		ec2-54-197-216-238.compute-1.amazonaws.com	Sandbox
test3-1835.5714308c:b80d3dbe-6d91-4181-a04b-5e3aa83fceab	test3-1835.5714308c:b80d3dbe-6d91-4181-a04b-5e3aa83fceab	STAGING		ec2-54-226-81-206.compute-1.amazonaws.com	Sandbox
test3-0702.54fa8694:b9a3cc9a-58f9-400d-b2ac-6c9dd151a963	test3-0702.54fa8694:b9a3cc9a-58f9-400d-b2ac-6c9dd151a963	STAGING		ec2-54-158-142-122.compute-1.amazonaws.com	Sandbox
test3-0131.efb771db:3dffda19-d39d-431a-ad3e-973a4a932398	test3-0131.efb771db:3dffda19-d39d-431a-ad3e-973a4a932398	STAGING		ec2-54-158-164-174.compute-1.amazonaws.com	Sandbox
test3-0845.f95b124b:c26c1e86-8cf1-4337-9d51-48b4b3e901f2	test3-0845.f95b124b:c26c1e86-8cf1-4337-9d51-48b4b3e901f2	STAGING		ec2-54-221-153-146.compute-1.amazonaws.com	Sandbox
test3-0241.23f69555:bb19e1b7-8011-409b-9629-190ed80eca92	test3-0241.23f69555:bb19e1b7-8011-409b-9629-190ed80eca92	STAGING		ec2-52-91-32-40.compute-1.amazonaws.com	Sandbox
test3-0069.3e2cd99c:6693f055-a1b6-42bb-a113-72a4c32c99ad	test3-0069.3e2cd99c:6693f055-a1b6-42bb-a113-72a4c32c99ad	STAGING		ec2-52-91-32-40.compute-1.amazonaws.com	Sandbox
test3-0437.ce767edb:0a3dea36-ecf9-497f-87f3-c84d9f43756e	test3-0437.ce767edb:0a3dea36-ecf9-497f-87f3-c84d9f43756e	STAGING		ec2-52-91-88-48.compute-1.amazonaws.com	Sandbox
test3-0045.5e5035ab:7e134c01-f459-443d-8dcd-2b755ae3bf84	test3-0045.5e5035ab:7e134c01-f459-443d-8dcd-2b755ae3bf84	STAGING		ec2-54-221-10-243.compute-1.amazonaws.com	Sandbox
test3-1919.d433af93:2d77536a-5eb4-4337-a0fd-b26b2a28bc84	test3-1919.d433af93:2d77536a-5eb4-4337-a0fd-b26b2a28bc84	STAGING		ec2-54-152-63-208.compute-1.amazonaws.com	Sandbox
test3-0107.0baadf18:4260eb52-99c1-4453-9e49-1a011a699f47	test3-0107.0baadf18:4260eb52-99c1-4453-9e49-1a011a699f47	STAGING		ec2-54-152-63-208.compute-1.amazonaws.com	Sandbox
test3-0906.d65513ff:c6f477e9-492b-4710-b1f1-c5fbbc36fa41	test3-0906.d65513ff:c6f477e9-492b-4710-b1f1-c5fbbc36fa41	STAGING		ec2-54-160-57-84.compute-1.amazonaws.com	Sandbox
test3-1495.f51d529d:b71ea06a-703f-4e12-acda-f5054999f961	test3-1495.f51d529d:b71ea06a-703f-4e12-acda-f5054999f961	STAGING		ec2-54-164-144-29.compute-1.amazonaws.com	Sandbox
test3-1418.8a9636b1:3923824f-d39f-4a5b-90eb-712c74e65d5c	test3-1418.8a9636b1:3923824f-d39f-4a5b-90eb-712c74e65d5c	STAGING		ec2-54-164-144-29.compute-1.amazonaws.com	Sandbox
test3-1793.700a3038:b18d3b3d-1480-4674-b4aa-ad2708a53f3c	test3-1793.700a3038:b18d3b3d-1480-4674-b4aa-ad2708a53f3c	STAGING		ec2-52-90-142-73.compute-1.amazonaws.com	Sandbox
test3-0789.868c8d8b:0421730a-e875-4ddd-938e-b17b2bbe5467	test3-0789.868c8d8b:0421730a-e875-4ddd-938e-b17b2bbe5467	STAGING		ec2-54-197-213-95.compute-1.amazonaws.com	Sandbox
test3-1616.f14a1f7d:bcc9bede-40f4-4244-acf2-3803c517515f	test3-1616.f14a1f7d:bcc9bede-40f4-4244-acf2-3803c517515f	STAGING		ec2-52-91-88-48.compute-1.amazonaws.com	Sandbox
test3-0799.acda253d:908732dc-10b2-4a40-8287-7b577a668f90	test3-0799.acda253d:908732dc-10b2-4a40-8287-7b577a668f90	STAGING		ec2-54-226-40-53.compute-1.amazonaws.com	Sandbox
test3-1486.cc2ccfaa:545e3b70-5fe7-41b7-bab1-31d964d1ed4e	test3-1486.cc2ccfaa:545e3b70-5fe7-41b7-bab1-31d964d1ed4e	STAGING		ec2-54-234-65-165.compute-1.amazonaws.com	Sandbox
test3-0230.03416eb5:459c8c8d-a1cd-4841-ae25-537da338fe96	test3-0230.03416eb5:459c8c8d-a1cd-4841-ae25-537da338fe96	STAGING		ec2-54-234-65-165.compute-1.amazonaws.com	Sandbox
test3-0324.078ea2f6:d411cb33-a481-4e7f-969c-a7a40a12818a	test3-0324.078ea2f6:d411cb33-a481-4e7f-969c-a7a40a12818a	STAGING		ec2-52-90-142-73.compute-1.amazonaws.com	Sandbox
test3-1796.88772499:d7d3da03-57ac-42df-8254-b990ed294bb8	test3-1796.88772499:d7d3da03-57ac-42df-8254-b990ed294bb8	STAGING		ec2-184-73-101-218.compute-1.amazonaws.com	Sandbox
test3-0488.475b680e:2c77aa74-8da3-47f0-86f7-eeb04668f7a7	test3-0488.475b680e:2c77aa74-8da3-47f0-86f7-eeb04668f7a7	STAGING		ec2-54-88-19-71.compute-1.amazonaws.com	Sandbox
test3-1201.f880d741:2f9b865f-d721-4b66-a4c5-a862fbe15d10	test3-1201.f880d741:2f9b865f-d721-4b66-a4c5-a862fbe15d10	STAGING		ec2-107-22-135-75.compute-1.amazonaws.com	Sandbox
test3-0739.22d61a92:a6c84e6e-9256-40c2-9882-fa7875b91520	test3-0739.22d61a92:a6c84e6e-9256-40c2-9882-fa7875b91520	STAGING		ec2-107-22-135-75.compute-1.amazonaws.com	Sandbox
test3-1088.58635506:20256687-36fd-4bbd-99b8-002db94601ee	test3-1088.58635506:20256687-36fd-4bbd-99b8-002db94601ee	STAGING		ec2-54-159-8-130.compute-1.amazonaws.com	Sandbox



/masters/master-0286	2048	0.2	1 / 1	▬	Running
/masters/master-0287	2048	0.2	1 / 1	▬	Running
/masters/master-0288	2048	0.2	1 / 1	▬	Running
/masters/master-0289	2048	0.2	1 / 1	▬	Running
/masters/master-0290	2048	0.2	1 / 1	▬	Running
/masters/master-0291	2048	0.2	1 / 1	▬	Running
/masters/master-0292	2048	0.2	1 / 1	▬	Running
/masters/master-0293	2048	0.2	1 / 1	▬	Running
/masters/master-0294	2048	0.2	1 / 1	▬	Running
/masters/master-0295	2048	0.2	1 / 1	▬	Running
/masters/master-0296	2048	0.2	1 / 1	▬	Running
/masters/master-0297	2048	0.2	1 / 1	▬	Running
/masters/master-0298	2048	0.2	1 / 1	▬	Running
/masters/master-0299	2048	0.2	1 / 1	▬	Running
/masters/master-0300	2048	0.2	1 / 1	▬	Running



Cloudbees Jenkins Platform

Pipelines Administration

Tiger

Cluster Summary Virtual Machines Masters

Masters

2000

Built at 15th August 2016 05:15 PM · master · 7e951e6

ec2-54-88-42-149.compute-1.amazonaws.com/blue/pipelines/masters

A 2000 JENKINS MASTERS CLUSTER

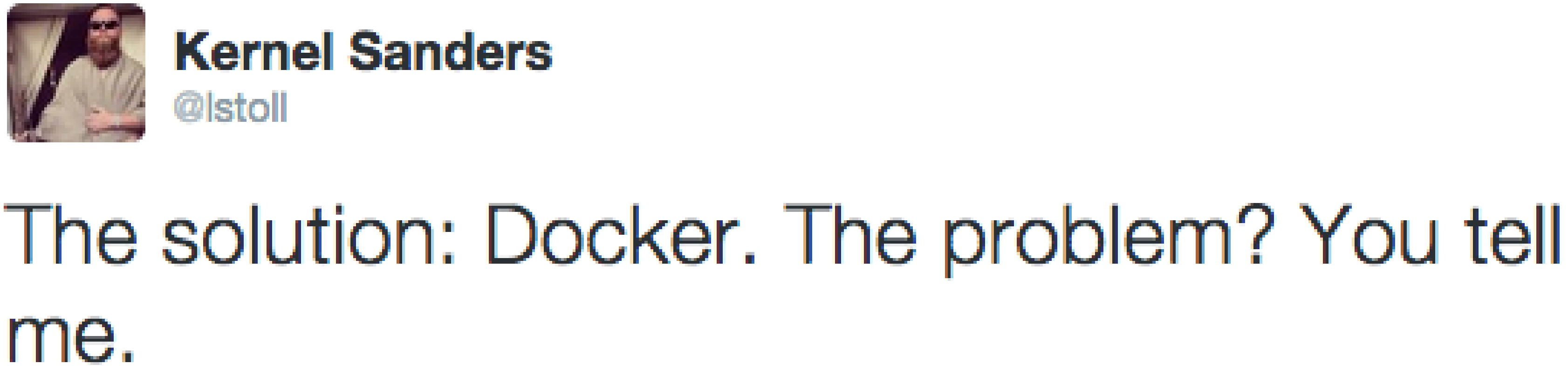
- 3 Mesos masters (m3.xlarge: 4 vCPU, 15GB, 2x40 SSD)
- 317 Mesos slaves (c3.2xlarge, m3.xlarge, m4.4xlarge)
- 7 Mesos slaves dedicated to ElasticSearch: (c3.8xlarge: 32 vCPU, 60GB)

12.5 TB - 3748 CPU

Running 2000 masters and ~8000 concurrent jobs

ARCHITECTURE

Docker Docker Docker



Kernel Sanders
@lstoll

The solution: Docker. The problem? You tell me.

Isolated Jenkins masters

Isolated build agents and jobs

Memory and CPU limits

> *How would you design your infrastructure if you couldn't login? Ever.*
>
> *Kelsey Hightower*

EMBRACE FAILURE!

CLUSTER SCHEDULING

- Running in public cloud, private cloud, VMs or bare metal
  - Starting with AWS and OpenStack
- HA and fault tolerant
- With Docker support of course

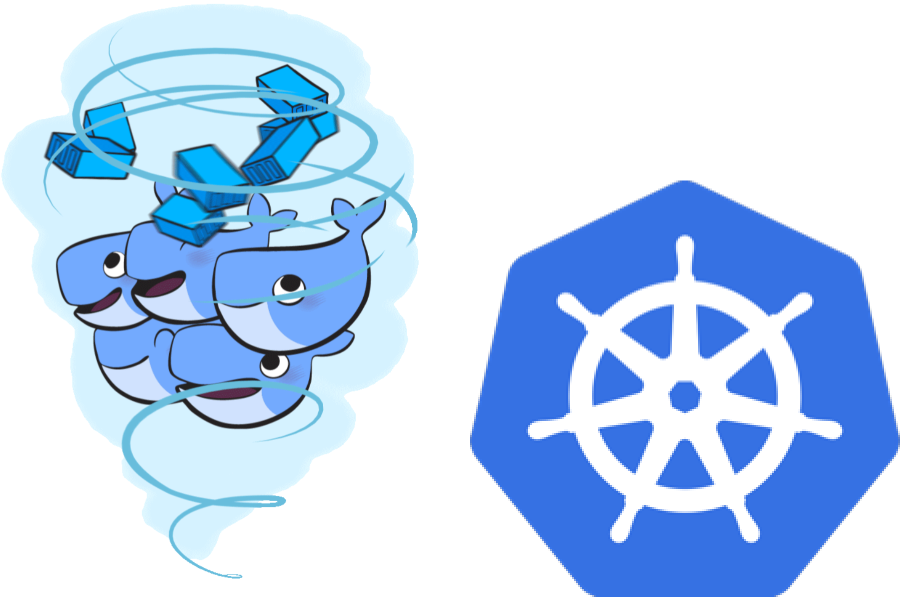
APACHE MESOS

MESOS

A distributed systems kernel

hadoop Spark™ kafka

ALTERNATIVES



Docker Swarm / Kubernetes

MESOSPHERE MARATHON

For long running Jenkins masters

<1.4 does not scale with the number of apps

App definitions hit the ZooKeeper node limit

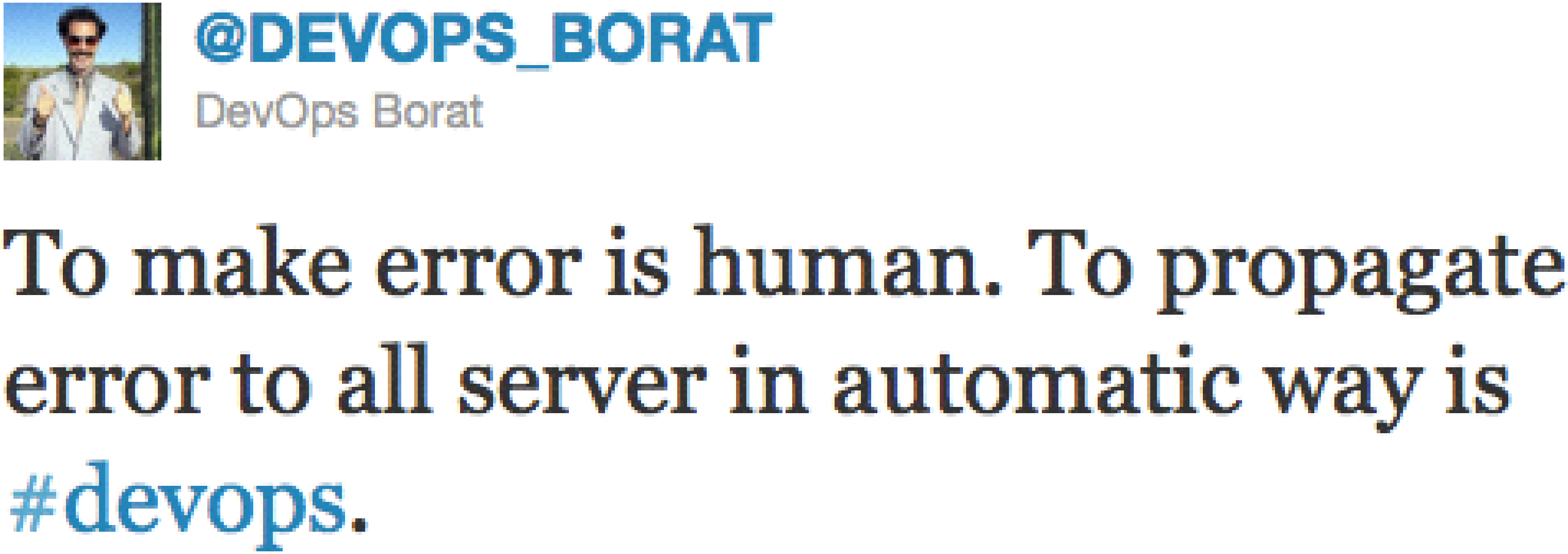
TERRAFORM

TERRAFORM

```
resource "aws_instance" "worker" {
    count = 1
    instance_type = "m3.large"
    ami = "ami-xxxxxx"
    key_name = "tiger-csanchez"
    security_groups = ["sg-61bc8c18"]
    subnet_id = "subnet-xxxxxx"
    associate_public_ip_address = true
    tags {
        Name = "tiger-csanchez-worker-1"
        "cloudbees:pse:cluster" = "tiger-csanchez"
        "cloudbees:pse:type" = "worker"
    }
    root_block_device {
        volume_size = 50
    }
}
```

TERRAFORM

- State is managed
- Runs are idempotent
  - `terraform apply`
- Sometimes it is too automatic
  - Changing image id will restart all instances
- Had to fix a number of bugs, ie. retry AWS calls



@DEVOPS_BORAT

DevOps Borat

To make error is human. To propagate error to all server in automatic way is #devops.

PACKER

- Preinstall packages: Mesos, Marathon, Docker
- Cached docker images
- Other drivers: XFS, NFS,...
- Enhanced networking driver (AWS)

MESOS FRAMEWORK

Started with Jenkins Mesos plugin

Means one framework per Jenkins master, does not scale

If master is restarted all jobs running get killed

OUR NEW MESOS FRAMEWORK

Using Netflix Fenzo

Runs under Marathon, exposes REST API that Jenkins masters call

- Reduce number of frameworks
- Faster to spawn new build agents because framework is not started
- Pipeline durable builds, can survive a restart of the master
- Dedicated workers for builds
- Affinity

STORAGE

Handling distributed storage

Servers can start in any host of the cluster

And they can move when they are restarted

Jenkins masters need persistent storage, agents (*typically*) don't

Supporting EBS (AWS) and external NFS

SIDEKICK CONTAINER

A privileged container that manages mounting for other containers

Can execute commands in the host and other containers

SIDEKICK CONTAINER *CASTLE*

Running in Marathon in each host

```
"constraints": [
  [
    "hostname",
    "UNIQUE"
  ]
]
```

A lot of magic happening with `nsenter`

both in host and other containers

- Jenkins master container requests data on startup using *entrypoint*
    - REST call to Castle
- Castle checks authentication
- Creates necessary storage in the backend
    - EBS volumes from snapshots
    - Directories in NFS backend

- Mounts storage in requesting container
  - EBS is mounted to host, then bind mounted into container
  - NFS is mounted directly in container
- Listens to Docker event stream for killed containers

CASTLE: BACKUPS AND CLEANUP

Periodically takes snapshots from EBS volumes in AWS

Cleanups happening at different stages and periodically

EMBRACE FAILURE!

PERMISSIONS

Containers should not run as root

Container user id != host user id

i.e. `jenkins` user in container is always 1000 but matches `ubuntu` user in host

CAVEATS

Only a limited number of EBS volumes can be mounted

Docs say `/dev/sd[f-p]`, but `/dev/sd[q-z]` seem to work too

Sometimes the device gets corrupt and no more EBS volumes can be mounted there

NFS users must be centralized and match in cluster and NFS server

MEMORY

Scheduler needs to account for container memory requirements and host available memory

Prevent containers for using more memory than allowed

Memory constrains translate to Docker `--memory`

WHAT DO YOU THINK HAPPENS WHEN?

Your container goes over memory quota?

iCACHONDEO.COM

WHAT ABOUT THE JVM?

WHAT ABOUT THE CHILD PROCESSES?

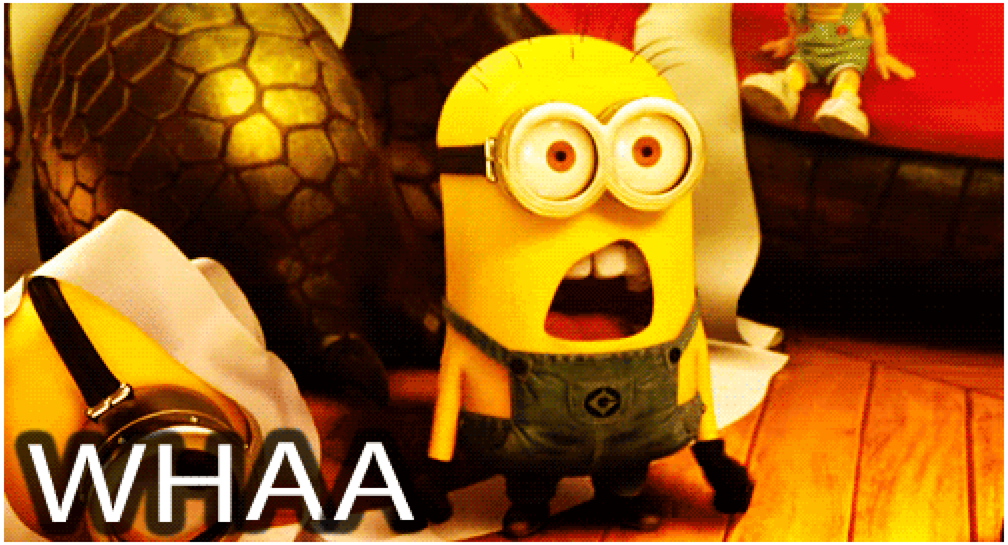
CPU

Scheduler needs to account for container CPU requirements and host available CPUs

WHAT DO YOU THINK HAPPENS WHEN?

Your container tries to access more than one CPU

Your container goes over CPU limits

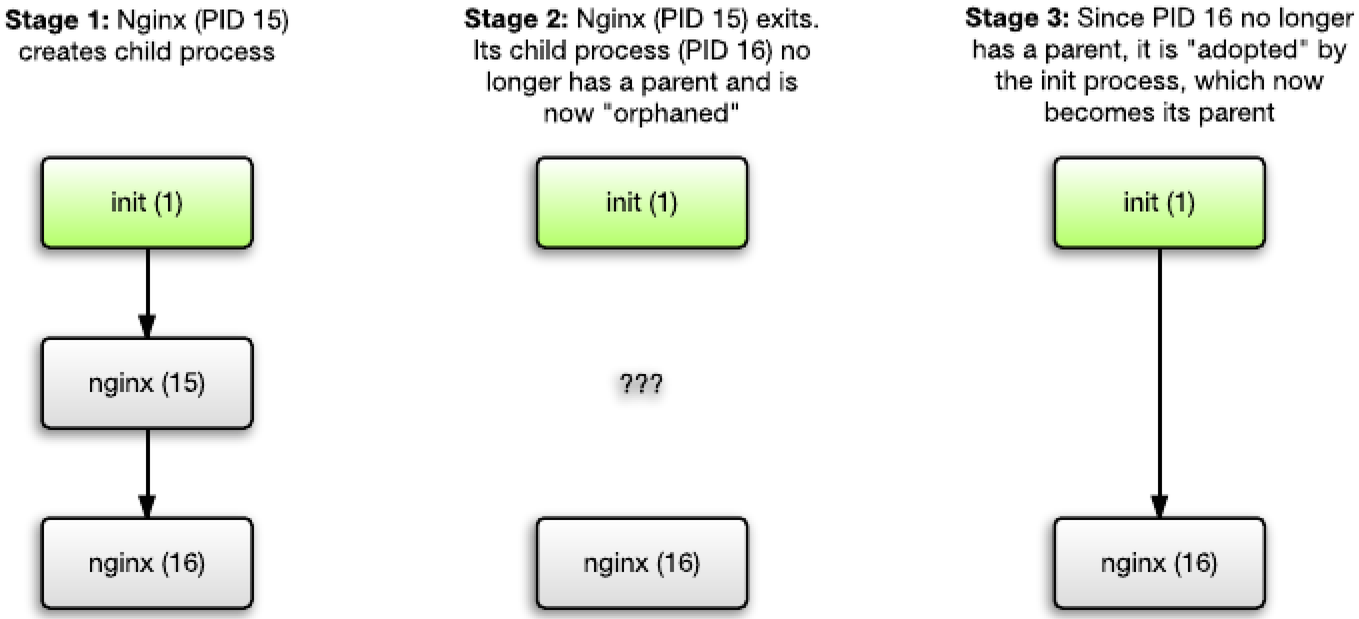


Totally different from memory

CPU translates into Docker `\-\-cpu-shares`

OTHER CONSIDERATIONS

ZOMBIE REAPING PROBLEM


Stage 1: Nginx (PID 15) creates child process
init (1)
nginx (15)
nginx (16)
Stage 2: Nginx (PID 15) exits. Its child process (PID 16) no longer has a parent and is now "orphaned"
init (1)
???
nginx (16)
Stage 3: Since PID 16 no longer has a parent, it is "adopted" by the init process, which now becomes its parent
init (1)
nginx (16)

Zombie processes are processes that have terminated but have not (yet) been waited for by their parent processes.

The init process -- PID 1 -- task is to "adopt" orphaned child processes

source

THIS IS A PROBLEM IN DOCKER

Jenkins build agent run multiple processes

But Jenkins masters too, and they are long running

`TINI`

Systemd or SysV init is too heavyweight for containers

> *All Tini does is spawn a single child (Tini is meant to be run in a container), and wait for it to exit all the while reaping zombies and performing signal forwarding.*

PROCESS REAPING

Docker 1.9 gave us trouble at scale, rolled back to 1.8

Lots of *defunct* processes

NETWORKING

Jenkins masters open several ports

- HTTP
- JNLP Build agent
- SSH server (Jenkins CLI type operations)

NETWORKING: HTTP

We use a simple `nginx` reverse proxy for

- Mesos
- Marathon
- ElasticSearch
- CJOC
- Jenkins masters

Gets destination host and port from Marathon

NETWORKING: HTTP

Doing both

- domain based routing `master1.pse.example.com`
- path based routing `pse.example.com/master1`
  - because not everybody can touch the DNS or get a wildcard SSL certificate

NETWORKING: JNLP

Build agents started dynamically in Mesos cluster can connect to masters internally

Build agents manually started outside cluster get host and port destination from HTTP, then connect directly

NETWORKING: SSH

SSH Gateway Service

Tunnel SSH requests to the correct host

Simple configuration needed in client

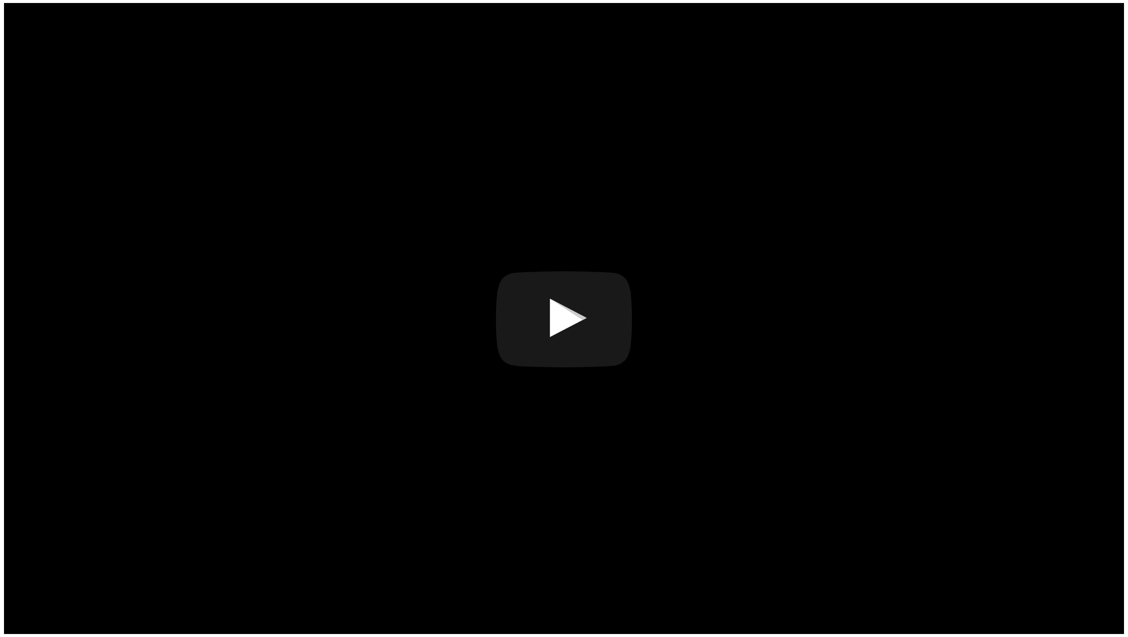
```
Host=*.ci.cloudbees.com
ProxyCommand=ssh -q -p 22 ssh.ci.cloudbees.com tunnel %h
```

allows to run

```
ssh master1.ci.cloudbees.com
```

SCALING

New and interesting problems



If you never used Docker in production, leave the room now

TERRAFORM AWS

- Instances
- Keypairs
- Security Groups
- S3 buckets
- ELB
- VPCs

AWS

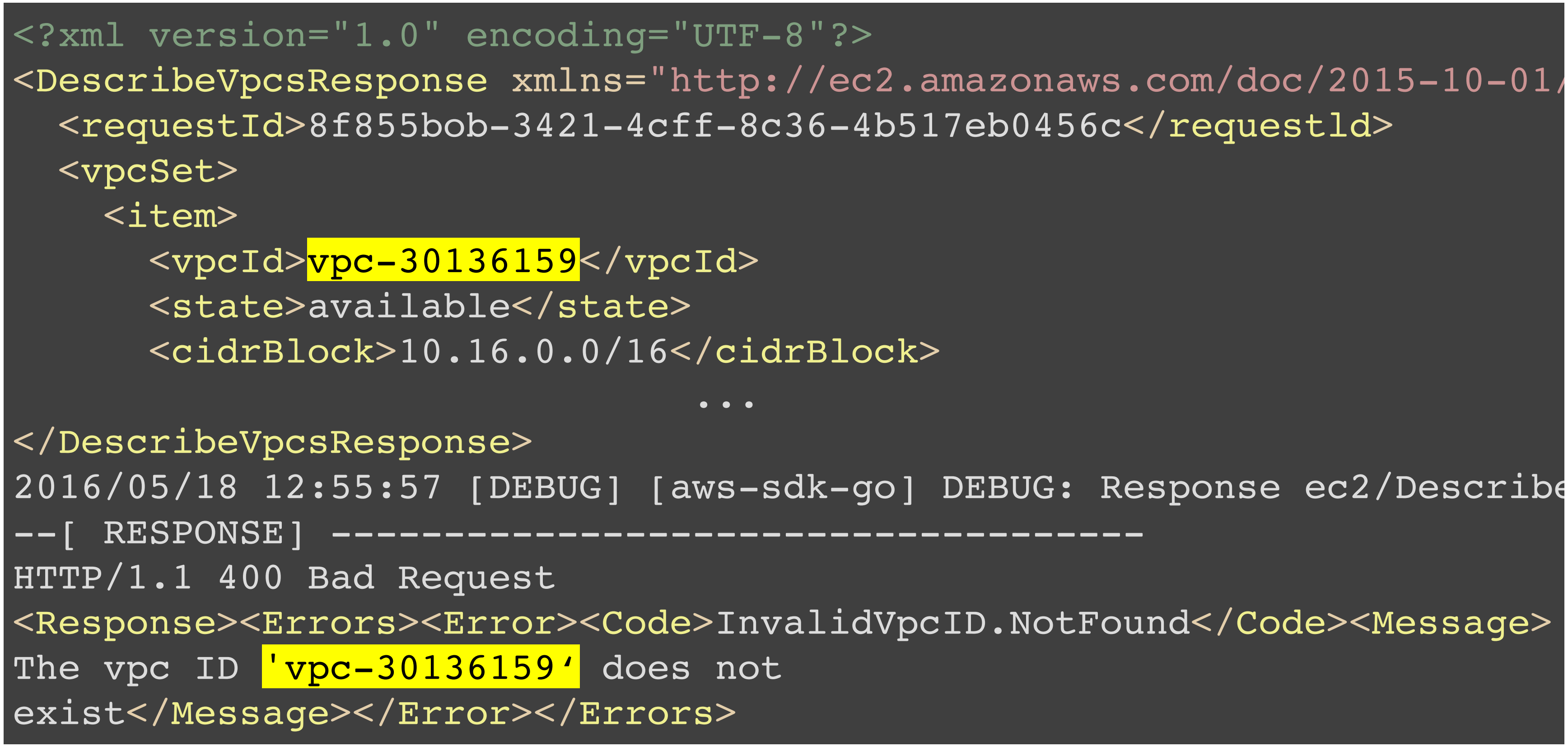
Resource limits: VPCs, S3 snapshots, some instance sizes

Rate limits: affect the whole account

Retrying is your friend, but with exponential backoff

AWS

Running with a patched Terraform to overcome timeouts and AWS *eventual consistency*



```
<?xml version="1.0" encoding="UTF-8"?>
<DescribeVpcsResponse xmlns="http://ec2.amazonaws.com/doc/2015-10-01/
  <requestId>8f855bob-3421-4cff-8c36-4b517eb0456c</requestld>
  <vpcSet>
    <item>
      <vpcId>vpc-30136159</vpcId>
      <state>available</state>
      <cidrBlock>10.16.0.0/16</cidrBlock>
                                      ...
</DescribeVpcsResponse>
2016/05/18 12:55:57 [DEBUG] [aws-sdk-go] DEBUG: Response ec2/Describe
--[ RESPONSE] ------------------------------------
HTTP/1.1 400 Bad Request
<Response><Errors><Error><Code>InvalidVpcID.NotFound</Code><Message>
The vpc ID 'vpc-30136159‘ does not
exist</Message></Error></Errors>
```

TERRAFORM OPENSTACK

- Instances
- Keypairs
- Security Groups
- Load Balancer
- Networks

OPENSTACK

Custom flavors

Custom images

Different CLI commands

There are not two OpenStack installations that are the same

GRACIAS

csanchez.org

csanchez

carlossg

cloudbees®