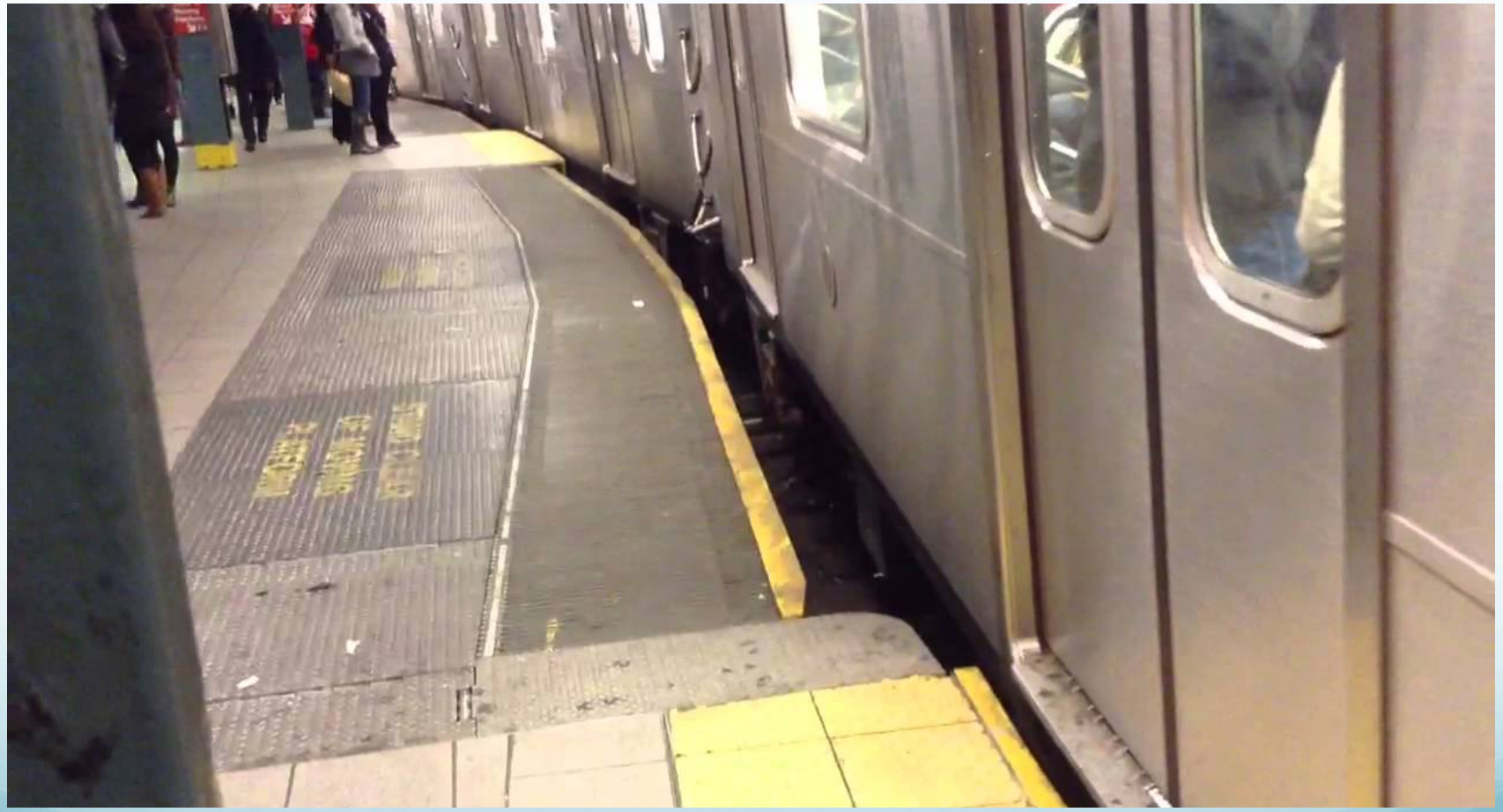
From Zero to Production – Our Mesos Journey

Chien Huey
DevOps Engineer
XO Group, Inc.

The Gap



Gap Items

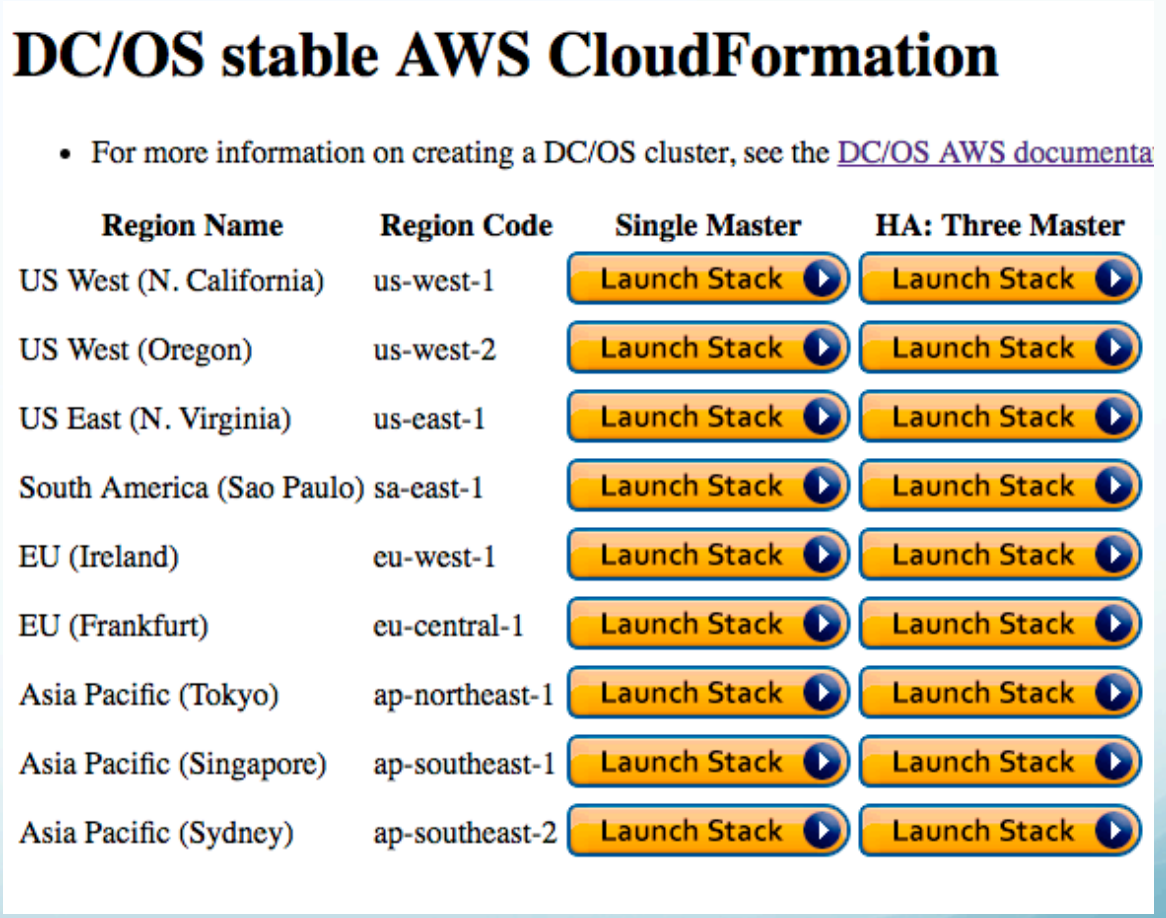
Getting to production

DC/OS:

- CloudFormation
- Advanced Installer

Vanilla Mesos:

- Non-DC/OS version
- Package manager installation
- Home-grown orchestration



DC/OS stable AWS CloudFormation

- For more information on creating a DC/OS cluster, see the DC/OS AWS documenta

Region Name	Region Code	Single Master	HA: Three Master
US West (N. California)	us-west-1	Launch Stack	Launch Stack
US West (Oregon)	us-west-2	Launch Stack	Launch Stack
US East (N. Virginia)	us-east-1	Launch Stack	Launch Stack
South America (Sao Paulo)	sa-east-1	Launch Stack	Launch Stack
EU (Ireland)	eu-west-1	Launch Stack	Launch Stack
EU (Frankfurt)	eu-central-1	Launch Stack	Launch Stack
Asia Pacific (Tokyo)	ap-northeast-1	Launch Stack	Launch Stack
Asia Pacific (Singapore)	ap-southeast-1	Launch Stack	Launch Stack
Asia Pacific (Sydney)	ap-southeast-2	Launch Stack	Launch Stack

Using DC/OS to run Mesos

Advantages

- Fully packaged and integrated
- Appliance-like AMI
- Command-line interface available

Using DC/OS to run Mesos

Disadvantages

- CloudFormation not production-friendly
- Installs all components regardless of node's role
- Advanced installer – infrastructure omission
- Upgrades unclear

Using DC/OS to run Mesos

CF/AMI Drawbacks

- Last 10% of configuration depends on CF userdata
  - Difficult to debug
  - Difficult to parameterize
  - Maintenance nightmare
- Small changes require recreating AMI, lots of time to make configure file change
- No mechanism to distribute files

Master node file structure

```
ubuntu@ip-172-23-9-180:/etc$ tree /etc/mesos-master
/etc/mesos-master
├── advertise_ip
├── cluster
├── hostname
├── ip
├── quorum
└── work_dir

0 directories, 6 files

ubuntu@ip-172-23-9-180:/etc$ tree /etc/mesos-slave
/etc/mesos-slave
└── work_dir
```

Slave node file structure

```
ubuntu@ip-172-23-5-132:~$ tree /etc/mesos-slave
/etc/mesos-slave
├── advertise_ip
├── containerizers
├── docker
├── executor_registration_timeout
├── hostname
├── ip
├── isolation
└── work_dir
0 directories, 8 files

ubuntu@ip-172-23-5-132:~$ tree /etc/mesos-master
/etc/mesos-master
├── quorum
└── work_dir
```

Why Templating

- Components match node role
- Upgrades via package manager
- Self-documenting (source of truth)
- Parameterize configurations across regions/ domains
- Abstract provisioning logic

Region-specific variable files

```
1 aws_region = "us-west-2"
2 aws_private_key_file = "/Users/chuey/.ssh/xo-consul-west.pem"
3 aws_flavor = "t2.small"
4 aws_private_key = "xo-consul-west"
```

```
1 aws_region = "us-east-2"
2 aws_private_key_file = "/Users/chuey/.ssh/xo-consul-east.pem"
3 aws_flavor = "t2.small"
4 aws_private_key = "xo-consul-east"
```

Isn’t templating another word for cloning?

Templating Mesos

Requirements

- Control machine (Red Hat, Debian, CentOS, Mac OS, BSD, etc). Windows not supported
  - Python 2.7
  - Terraform
  - Ansible 2.1.0
  - Cloud provider CLI client (aws-cli)
- Terraform-supported cloud provider (AWS, Google Compute)

Terraform Templating

- Provisions
  - Instances
  - DNS
  - Security Groups
  - S3 buckets
  - IAM roles, policies

Terraform Drawbacks

- Abstraction is not complete; configurations are specific to cloud provider (not portable)
- Managing dependencies can be imprecise/tricky
  - IAM roles do not propagate fully, need pause in order to launch instance with role (show example)
- Cannot depend on order of operations
  - Terraform engine the ultimate decision of exclusion order

```
resource “aws_instance” “aws-web” {
    ami=“${data.aws.ami.ubuntu.id”
    instance_type=“t2.micro”
    tags {
         Name=“Helloworld”
    }

}

resource “google_compute_instance” “google-web” {
    name=“test”
    machine_type=“n1-standard-1”
    zone = “us-central1-a”
    tags {
         Name=“Helloworld”
    }

}
```

```
24
25 resource "aws_iam_role" "swarm_base" {
26   name = "swarm_base"
27   assume_role_policy = <<-EOF
28   {
29       "Version": "2012-10-17",
30       "Statement": [
31           {
32               "Action": "sts:AssumeRole",
33               "Principal": {"Service": "ec2.amazonaws.com"},
34               "Effect": "Allow",
35               "Sid": "AllowAssumeRole"
36           }
37       ]
38   }
39   EOF
40
41   provisioner "local-exec" {
42     command = "sleep 30"
43   }
44 }
```

Ansible

- Ideally suited for updating configuration of nodes
- Ansible order of execution is absolute (top to bottom)
- Declare separate tasks for each component
  - Common
  - Docker
  - Chronos
  - Masters
  - Slave

Inventory files

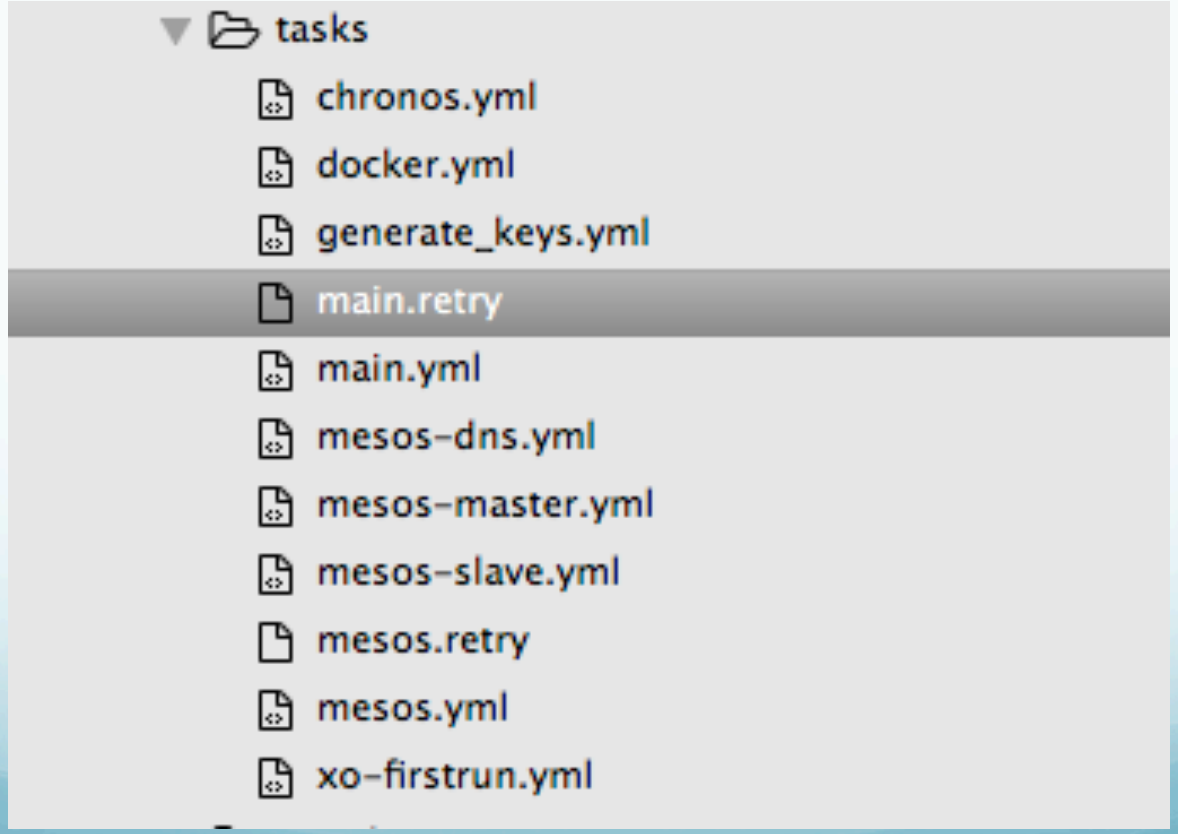
```
Outputs:

ansible_inventory = [mesos_masters_east]
54.227.XX.XX
54.146.XX.XX
54.164.XX.XX

[mesos_agents_east]
54.227.XX.XX
54.89.XXX.XXX
54.157.XX.XXX
```

What userdata can do better

```
"UserData": {
      "Fn::Base64": {
      "Fn::Join": [
            "“,                [
           "#cloud-config\n",
           "\"coreos\":\n“,
           "  \"units\":\n",
           "  - \"command\": |-\n",
           "      start\n",
           "    \"content\": |\n",
           "      [Unit]\n“,
           "      Description=AWS Setup: Formats the /
var/lib ephemeral drive\n“,
```

Role separation

- Show how the various components (Chronos, Docker, etc) can be handled by separate roles



Leverage mesosphere packages

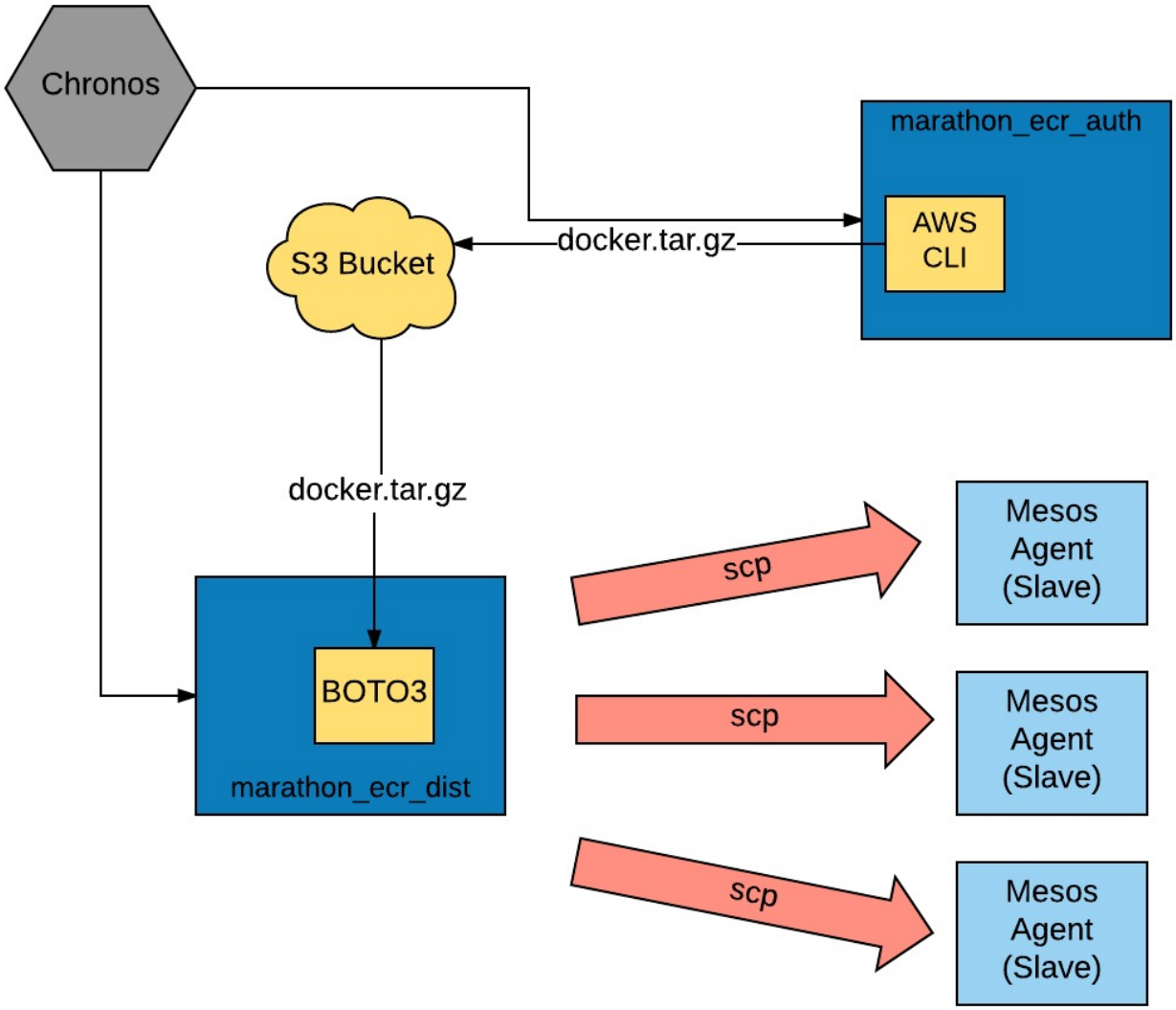
- Utilize the package manager to handle installs/ upgrades

```
ubuntu@ip-172-23-2-229:~$ sudo apt list | grep mesos

WARNING: apt does not have a stable CLI interface yet. Use with caution in scripts.

mesos/trusty,now 1.0.1-2.0.93.ubuntu1404 amd64 [installed]
mesosphere/trusty 0.1.1~ubuntu14.04-1 all
ubuntu@ip-172-23-2-229:~$
```

Live Demo

Private Docker Registries

- Generate Docker auth token
- Distribute auth token onto all the slaves
- Refresh auth token every 6 hours


Chronos
marathon_ecr_auth
AWS
CLI
S3 Bucket
docker.tar.gz
docker.tar.gz
BOTO3
marathon_ecr_dist
scp
scp
scp
Mesos
Agent
(Slave)
Mesos
Agent
(Slave)
Mesos
Agent
(Slave)

Config.ini

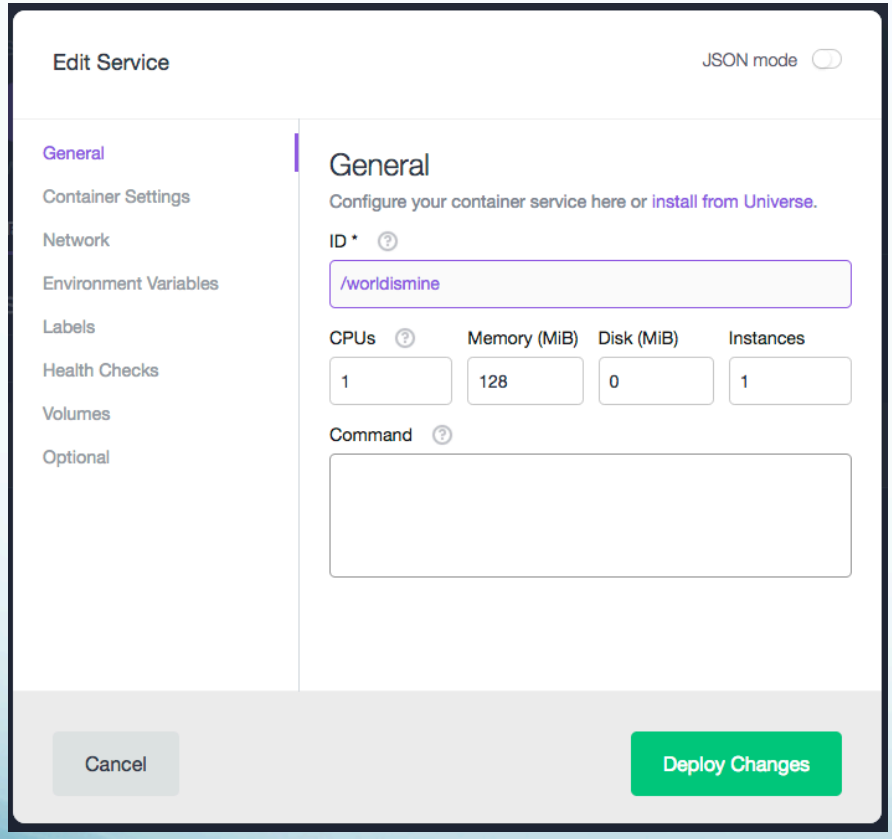
```
[ECRDist]
s3Bucket: mesos-marathon-repo-authentication

RoleARN: arn:aws:iam::352362988575:role/
mesos_base

PrivateKeyFileName: marathon-ecr-dist.pem

MesosClusterURL:
http://mesos-masters.xyz.xogrp.com:5050/
master/slaves

region: us-east-1
```

Generate Docker token

```
#!/bin/ash

eval `aws ecr get-login --region=us-
east-1`

cd ~

tar czf $WORKDIR/docker.tar.gz ~/.docker/
config.json
```

```
def download_auth_token():

    client = boto3.client('sts')
    assumedRoleObject = client.assume_role
(RoleArn=RoleARN,
    RoleSessionName='mesos_base_role’)
    aws_access_key = assumedRoleObject['Credentials']
['AccessKeyId’]
    aws_secret_key = assumedRoleObject['Credentials']
['SecretAccessKey’]
    aws_session_token = assumedRoleObject['Credentials']
['SessionToken']

    aws_session = boto3.session.Session(aws_access_key,
aws_secret_key, aws_session_token, region)
    s3 = aws_session.resource('s3’)
    s3.meta.client.download_file('mesos-marathon-repo-
authentication','docker.tar.gz','/auth/marathon/
docker.tar.gz’)
    s3.meta.client.download_file('mesos-marathon-repo-
authentication',privateKeyFileName,'/auth/marathon/' +
privateKeyFileName)
```

Marathon interface




```
{
  "id": "/shipyard",
  "cmd": "docker run -v /etc/docker:/etc
    ssl:ro --name shipyard --link rethin
    swarm-manager:swarm-manager -p $PORT
    shipyard:latest --debug server -d  t
    --tls-ca-cert=/etc/docker/ca.pem --t
    server-cert.pem --tls-key=/etc/docke
  "cpus": 0.25,
  "mem": 512,
  "disk": 0,
  "instances": 1,
  "constraints": [
    [
      "hostname",
      "UNIQUE"
    ],
    [
      "hostname",
      "LIKE",
      "mesos-agent-a1-us-east-1.xyz.xogr
    ]
  ],
  "labels": {
```

Autoscale implementation

- Design phase
- Metrics:
  - master/cpus_* (percent|used|total)
  - slave/mem_* (used|total)
  - system/load_* (15min|5min|10min)
- IAM roles
- Chronos
- Marathon

Conclusion

- What XO is doing
  - DCOS for Elasticsearch
  - Containers use Docker Swarm
- Why
  - No available non-proprietary CLI
  - Awkward Marathon containerizer interface
  - Zookeeper's latency sensitivity


Elastic Search