Containers on AWS

AWS London Loft

Paul Maddox, Amazon Web Services
Ric Harvey, Ticketmaster

September 2017

What to expect from this session

- Choices for container orchestration on AWS
- Amazon EC2 Container Service (Amazon ECS)
    - Deployment
    - Features
    - Ops
- AWS and the Cloud Native Computing Foundation (CNCF)
- Kubernetes
    - Deployment
    - Features
    - Ops
- Who’s using what?
- An inside view of Kubernetes on AWS by Ticketmaster

aws

Running containers in development is easy…

```
$ vi Dockerfile
$ docker build -t mykillerapp:0.0.1 .
$ docker run -it mykillerapp:0.0.1
```


aws

Moving to production is harder…

Common questions

- How do I deploy my containers to hosts?
  - How do I do zero downtime or blue green deployments?
- How do I keep my containers alive?
- How can my containers talk to each other?
  - Linking? Service Discovery?
- How can I configure my containers at runtime?
  - What about secrets?
- How do I best optimise my "pool of compute”?

aws

Container Orchestration

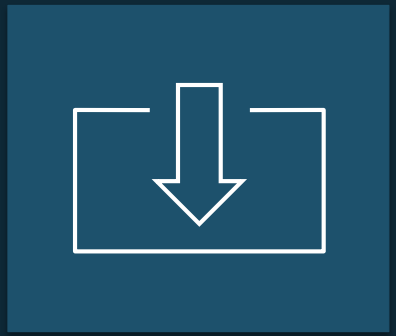
Our goal at AWS, is to be the best place to run containers.

… whatever your choice of orchestration engine is.

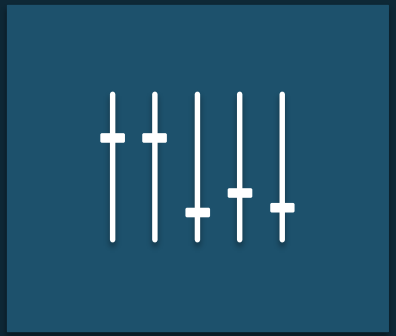
Amazon ECS

Amazon ECS

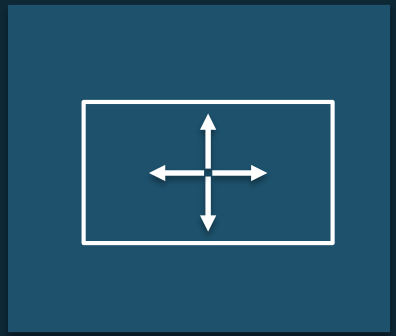
There is no software to:



Install



Manage

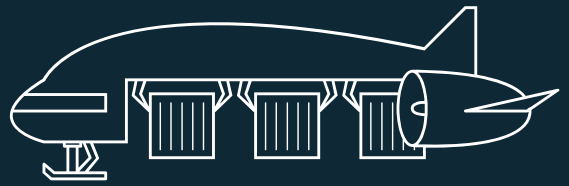


Scale

aws

Deep integration

with the AWS platform



Load Balancing

Auto Scaling

Logging

Networking

ID & Access

Monitoring

Security and compliance

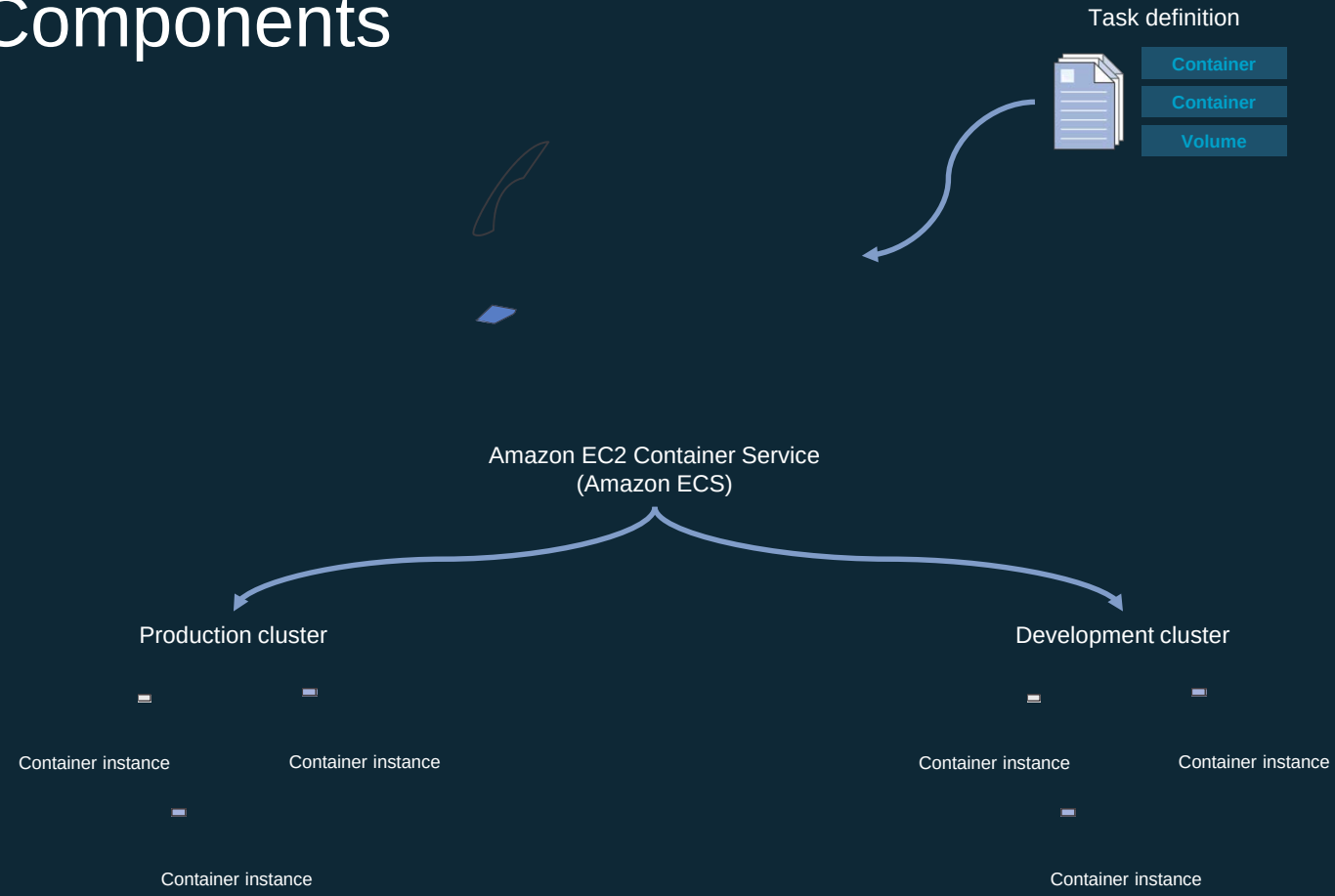
Suitable for regulated workloads

IAM Roles per container/task

ISO 27001

PCI

Key Components

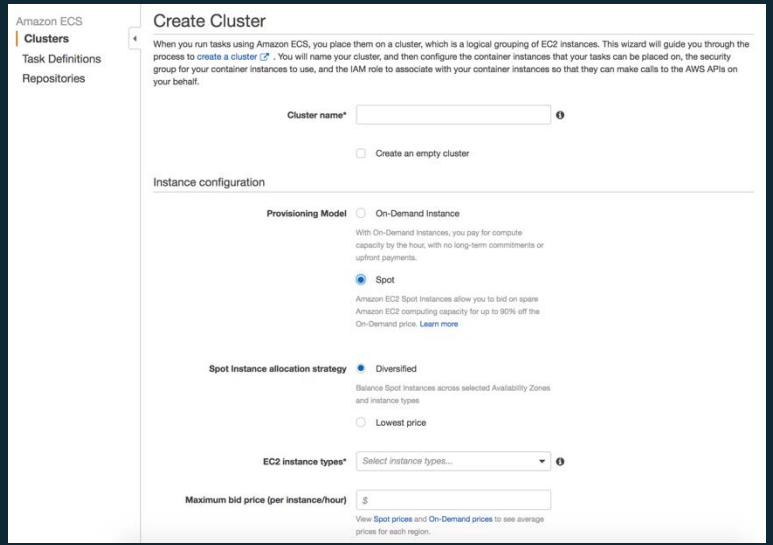


Amazon ECS: Deployment

Creating a cluster

$ ecs-cli up --keypair my-key --capability-iam --size 3

[--verbose]
[--cluster *cluster_name*]
[--capability-iam |
--instance-role *instance-profile-name*]
[--size *n*]
[--azs *availability_zone_1,availability_zone_2*]
[--security-group *security_group_id*[,*security_group_id*[,...]]]
[--cidr *ip_range*]
[--port *port_number*]
[--subnets *subnet_1,subnet_2*]
[--vpc *vpc_id*]
[--instance-type*instance_type*]
[--image-id *ami_id*]
[--no-associate-public-ip-address]
[--force]
[--region *region*]



Deploying a service

```
$ cat > docker-compose.yml

version: '2'
services:
  mykillerapp:
    image: mykillerapp:0.0.1
    cpu_shares: 100
    mem_limit: 524288000
    ports:
    - "8080:80"
```

```
# Run it locally
$ docker-compose up
```

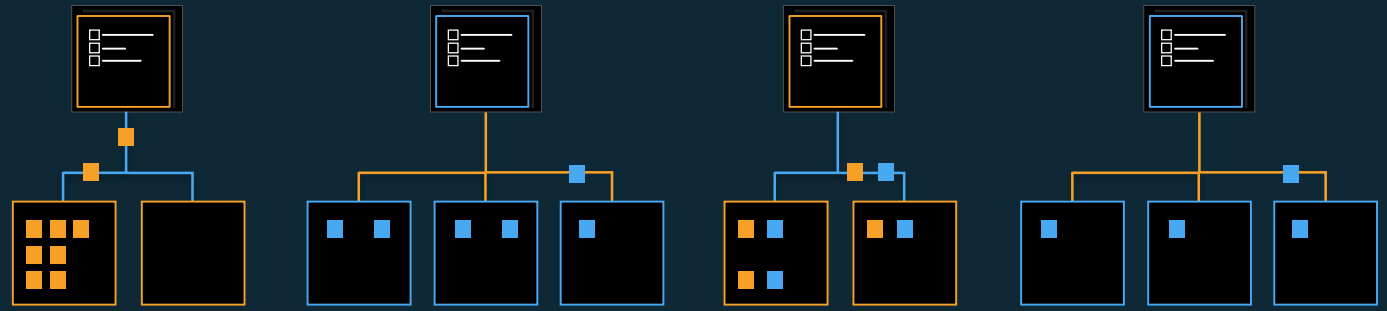
```
# Run it Amazon ECS
$ ecs-cli compose service up
  [--role ""]
  [--load-balancer-name ""]
  [--container-name ""]
  [--container-port ""]
  [--target-group-arn ""]
```

```
# Check the status
$ ecs-cli ps
State                Ports              TaskDefinition
mykillerapp  RUNNING.  0.0.0.0:8080->80/tcp  mykillerapp:10
```


aws

Amazon ECS: Resource Optimisation

Placement Strategies

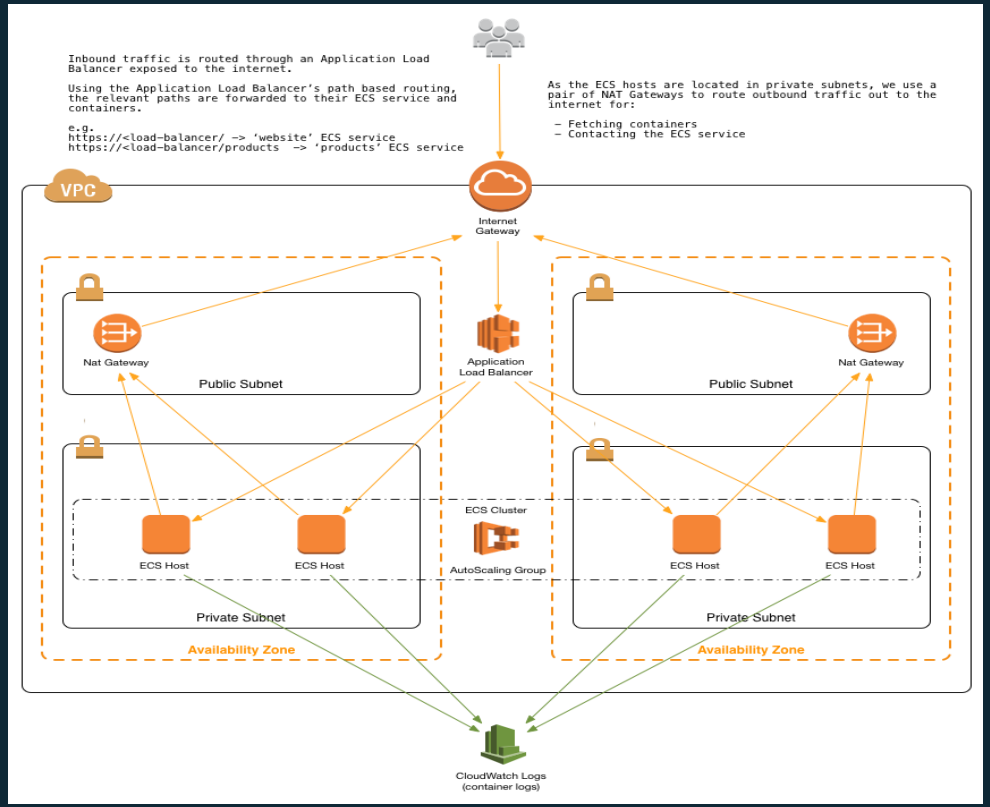


Placement Constraints

Name	Example
AMI ID	attribute:ecs.ami-id == ami-eca289fb
Availability Zone	attribute:ecs.availability-zone == us-east-1a
Instance Type	attribute:ecs.instance-type == t2.small
Distinct Instances	type="distinctInstances"
Custom	attribute:stack == prod

Amazon ECS: Service Discovery

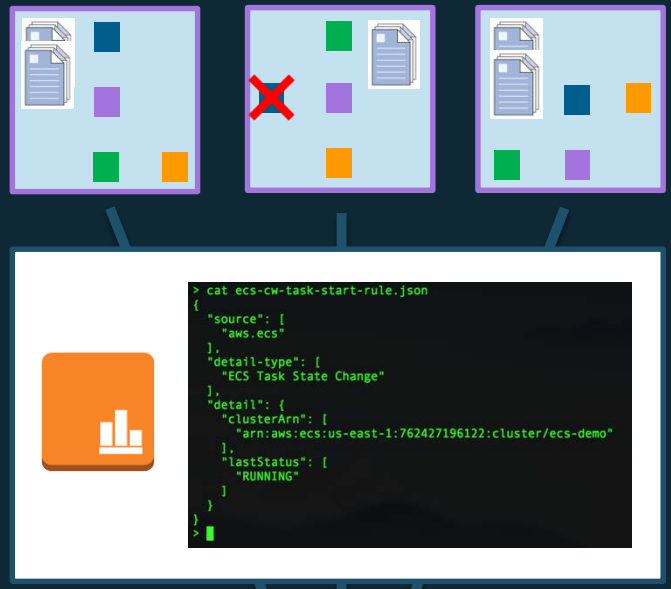
Service Discovery (Load Balancers)



Service Discovery (DNS)

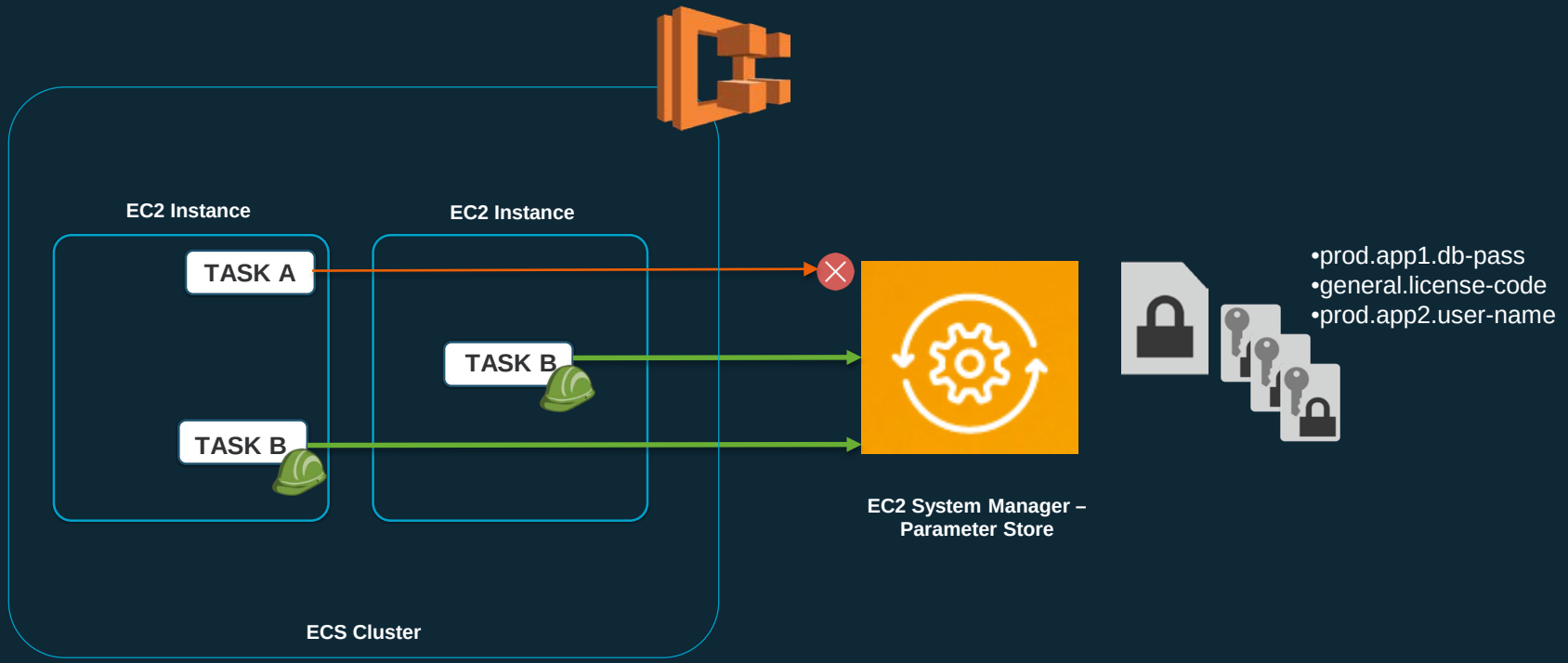
app1-tst → 10.1.0.11
db1-tst → 10.1.0.14
app2 → 10.1.0.16
db2 → 10.1.0.18
my-app → 10.1.0.20
db-dev→ 10.1.0.19
websrv1→ 10.1.0.1
websrv2→ 10.1.0.2
websrv3→ 10.1.0.4
app-dev1→ 10.1.0.9
app-dev2→ 10.1.0.5
app-dev3→ 10.1.0.8

```python
def lambda_handler(event, context):

    container_instance = ecs.describe_container_instances(
        cluster=event['detail']['clusterArn'],
        containerInstances=[
            event['detail']['containerInstanceArn'],
        ]
    )['containerInstances'][0]

    ec2_instance = ec2.describe_instances(
        InstanceIds=[container_instance['ec2InstanceId']]
    )['Reservations'][0]['Instances'][0]

    private_dns = ec2_instance['PrivateDnsName']

    container_name = event['detail']['containers'][0]['name']

    r53.change_resource_record_sets(
        HostedZoneId=HostedZoneId,
        ChangeBatch={
            'Changes': [
                {
                    'Action': 'UPSERT',
                    'ResourceRecordSet': {
                        'Name': '{}.{}'.format(container_name, HostedZoneName),
                        'Type': 'SRV',
                        'TTL': 60,
                        'ResourceRecords': [
                            {
                                'Value': '1 1 {} {}.'.format(80, private_dns)
                            }
                        ]
                    }
                }
            ]
        }
    )
```



Amazon ECS: Config / Secrets

Configuration / Secrets



Amazon ECS: Monitoring/logging

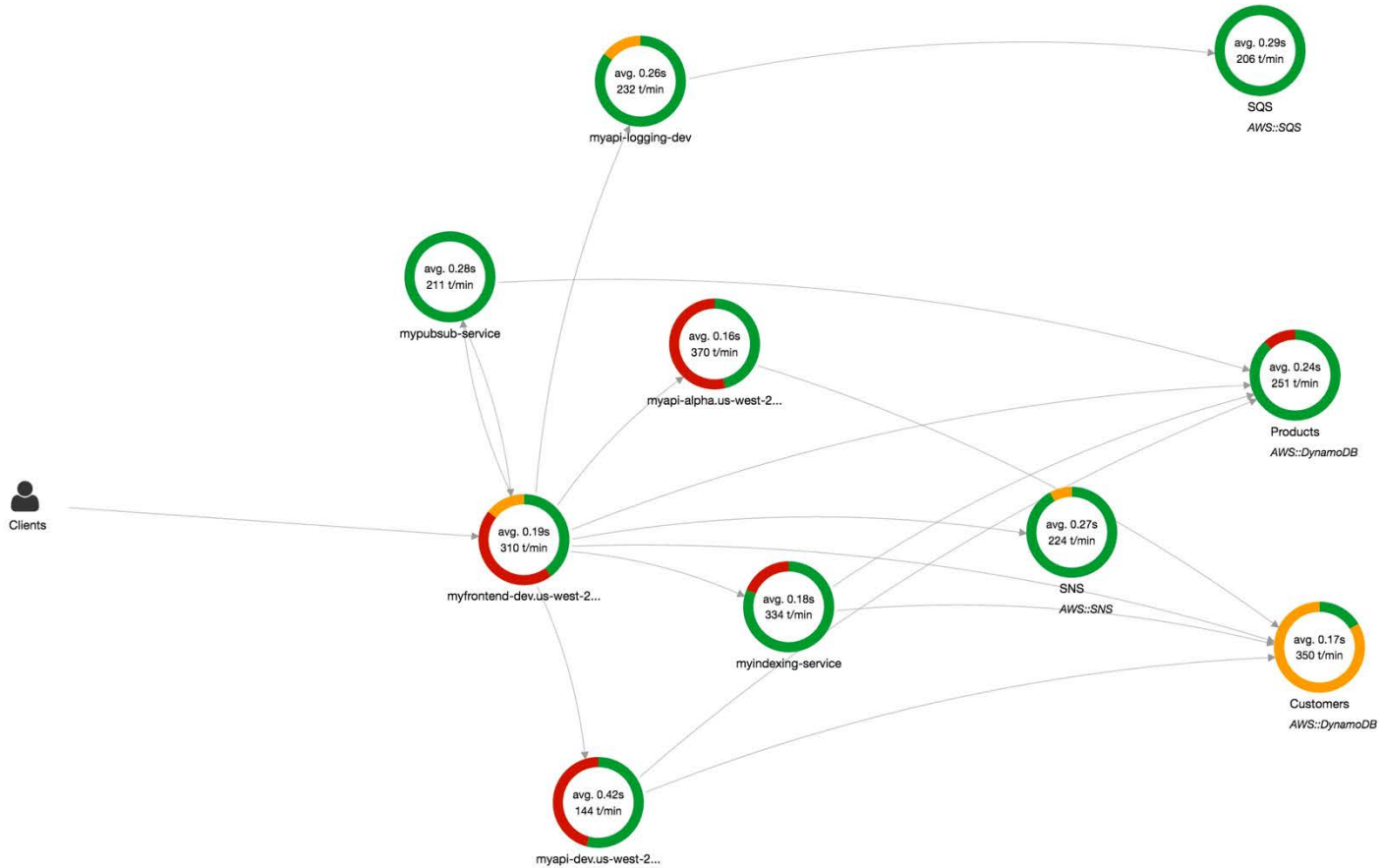
Enter Service Name, Annotation, Trace ID or click the Help icon for additional details

Last 5 Minutes

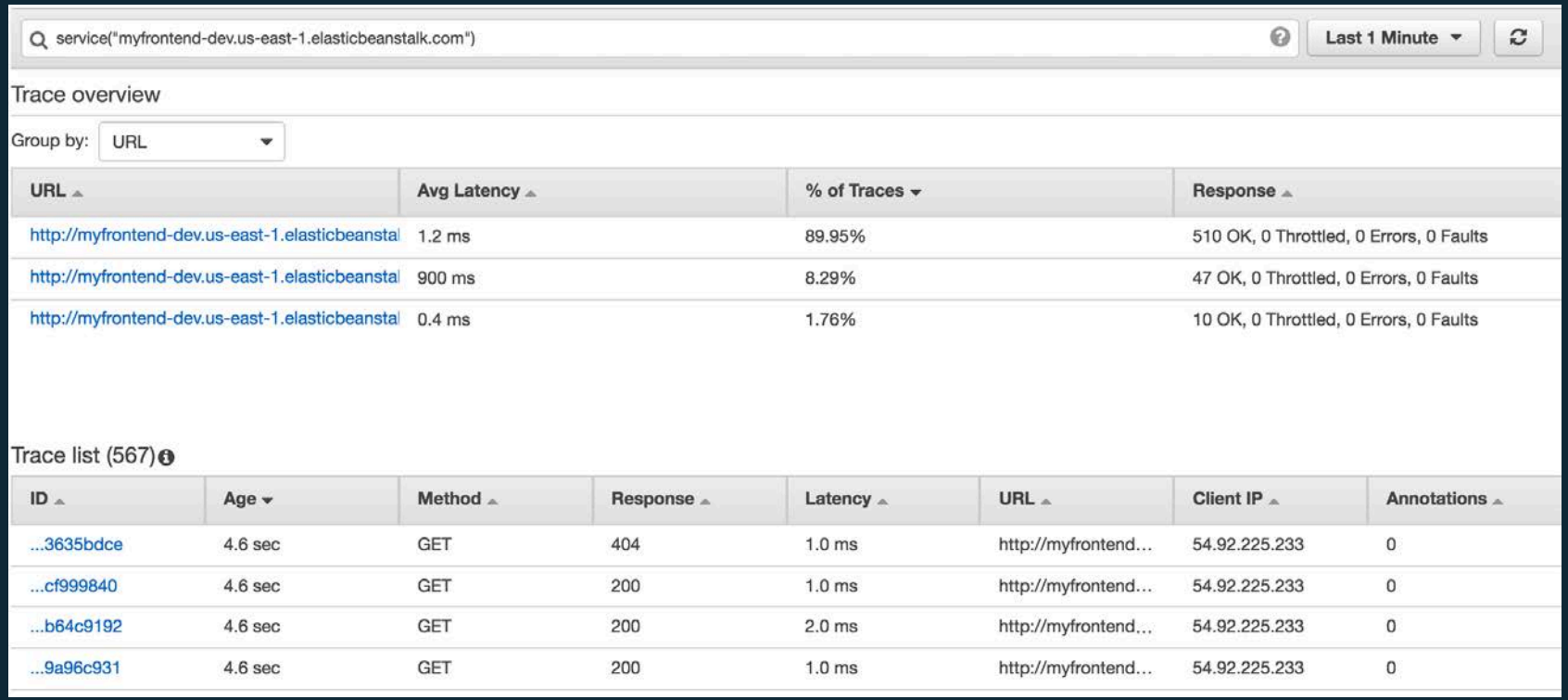
Service map

Updated on 2016/12/04 03:14:26 (UTC -08:00)

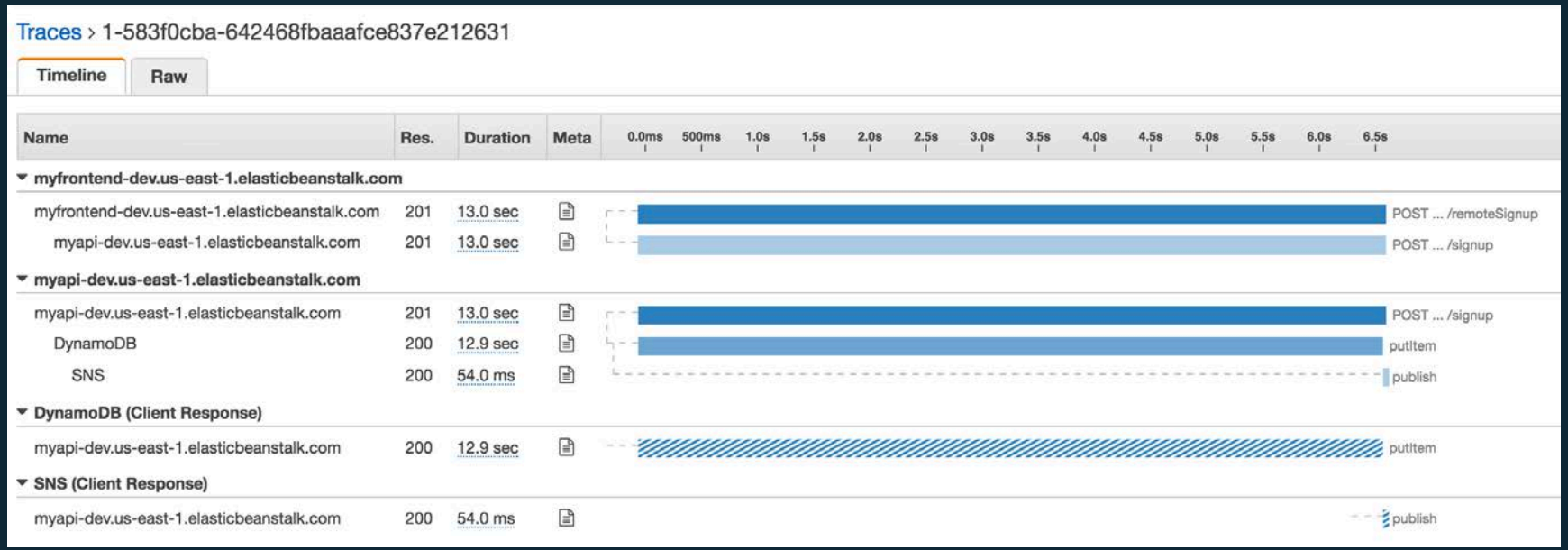
Map legend



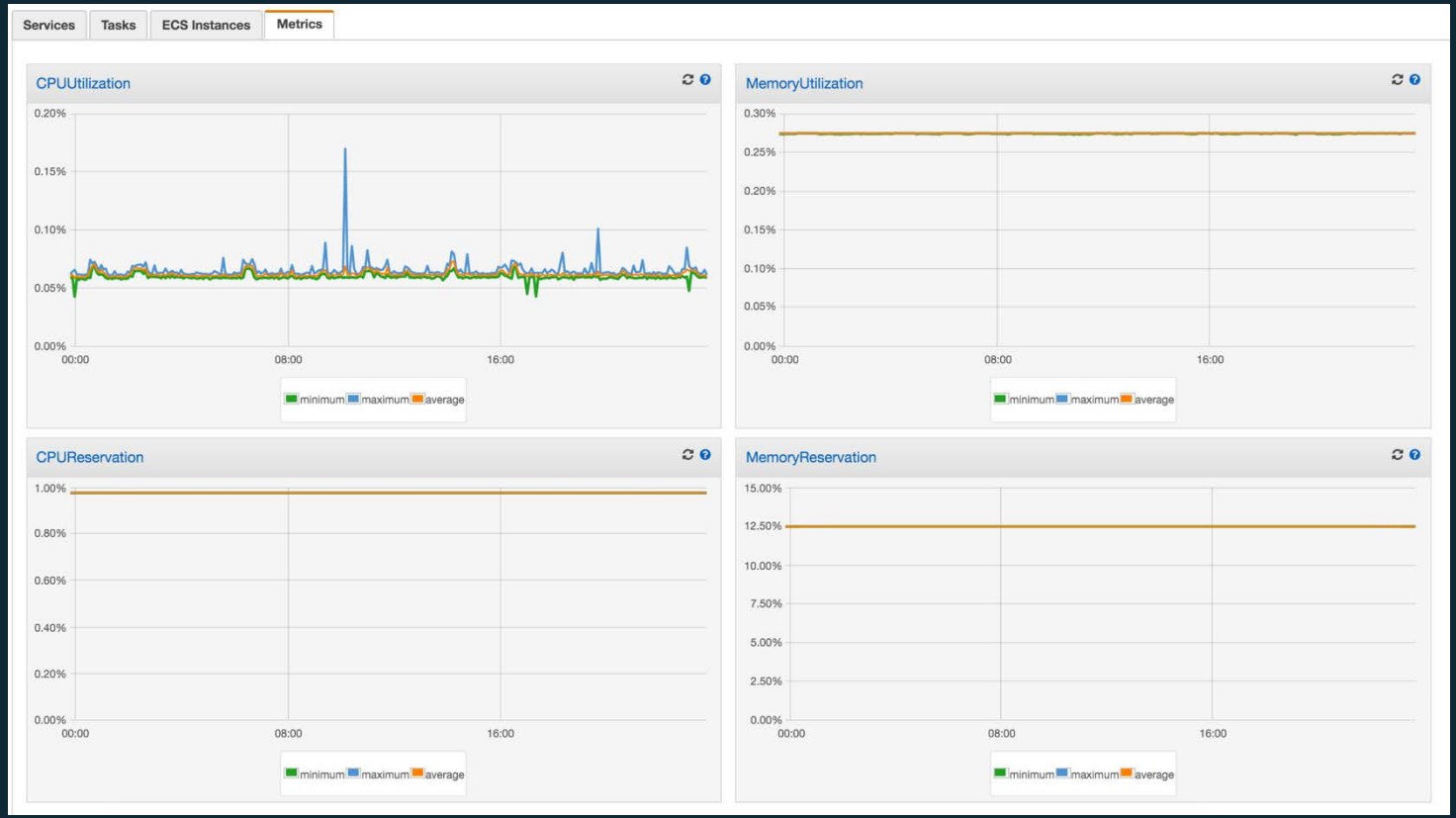
AWS X-Ray: Identify performance bottlenecks



AWS X-Ray: Identify performance bottlenecks



Monitoring with CloudWatch



Monitoring with CloudWatch

Exporting metrics to other tooling

Prometheus:

https://github.com/slok/ecs-exporter

ElasticSearch:

http://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_ES_Stream.html

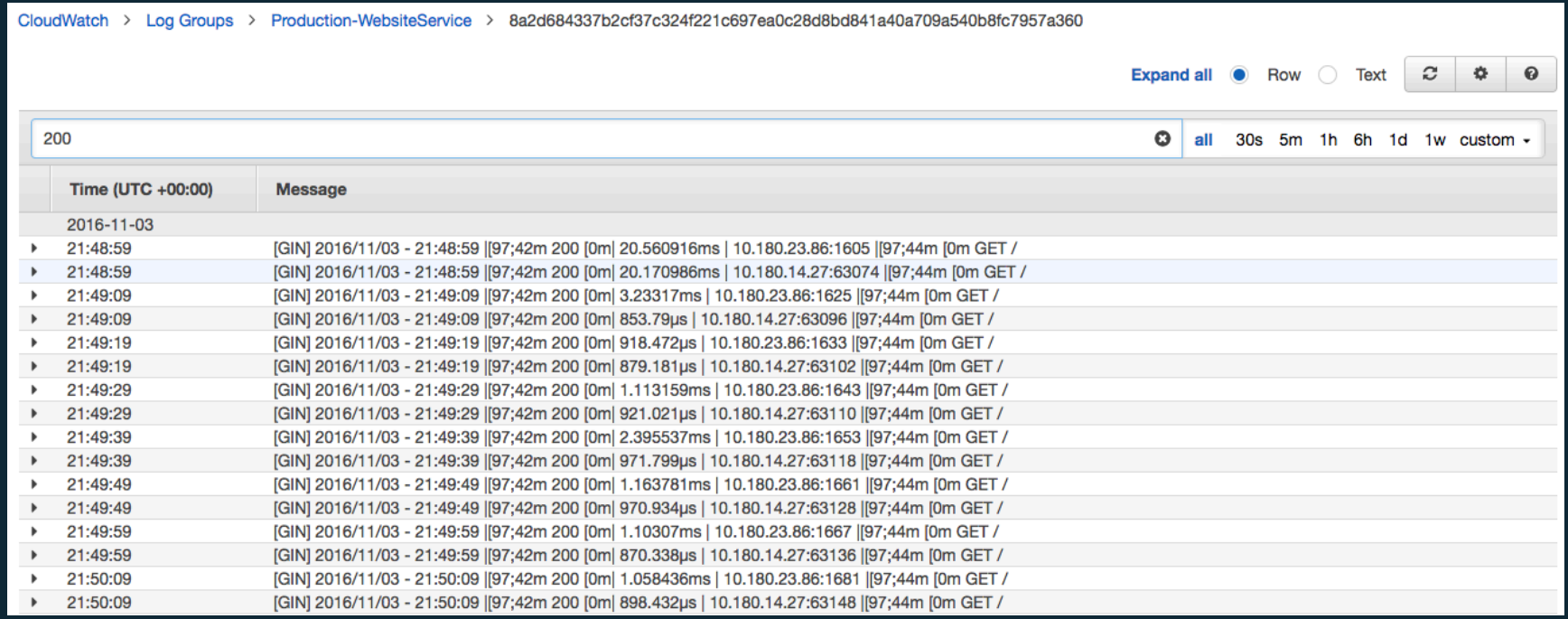
Centralized Logging with CloudWatch Logs

```
{
  "image": ”nginx:latest",
  …
  "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
      "awslogs-group": ”nginx",
      "awslogs-region": "us-east-1"
    }
  }
{
```

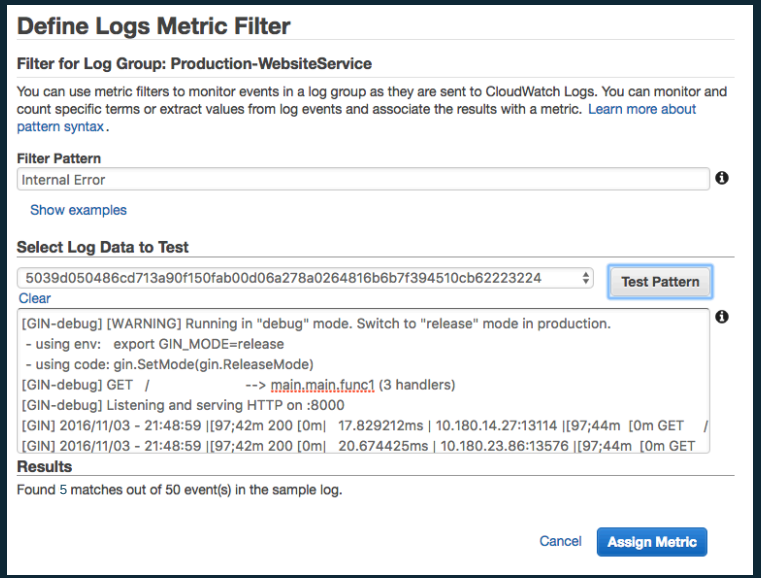
- Defined within the task definition
- Available log drivers
  - awslogs
  - fluentd
  - gelf
  - journald
  - json-file
  - splunk
  - Syslog
- Submit a pull request on ECS agent GitHub repo if you would like others

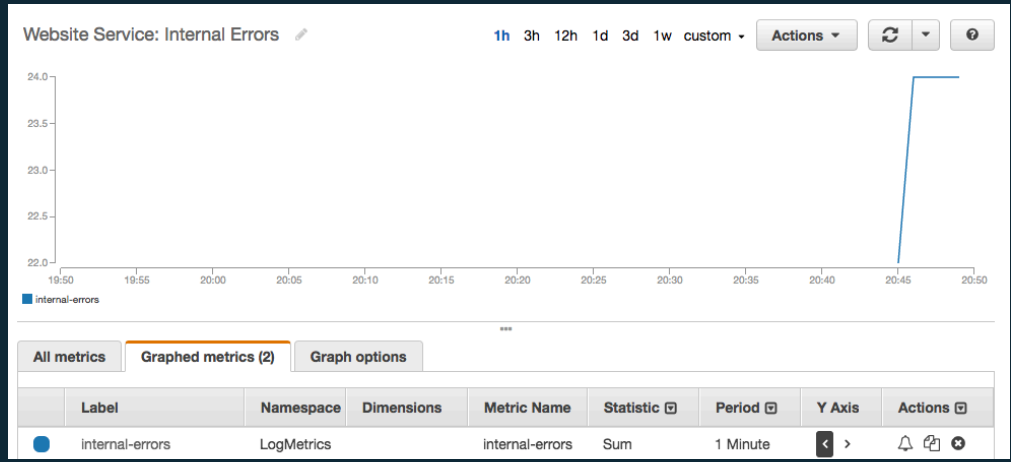
aws

Centralized Logging with CloudWatch Logs



Tip: Use Metric Filters with CloudWatch Logs





Amazon ECS wrap-up

- Fully managed container orchestration
- Service discovery with load balancers or DNS
- Flexibility with ECS event stream
- Placement strategies / constraints
- Conf/secrets management with env vars + SSM Parameter Store
- Monitoring with X-Ray, CloudWatch, or your own tooling
- Free! Just pay for the container instances (EC2)

AWS and the CNCF

Amazon Web Services Joins Cloud Native Computing Foundation as Platinum Member

By cncf | August 9, 2017 | Announcement

AWS Formalizes Collaboration with the Foundation as More and More Cloud Native Workloads Run in the AWS Cloud

San Francisco – August 9, 2017 – The Cloud Native Computing Foundation (CNCF), which is sustaining and integrating open source technologies to orchestrate containers as part of a microservices architecture, today announced that Amazon Web Services (AWS) has joined the CNCF as a platinum member to accelerate the development and deployment of cloud native technologies in its industry-leading public cloud. As part of the membership, Adrian Cockcroft, Vice President of Cloud Architecture Strategy at AWS, will join CNCF's Governing Board.

According to a recent survey, 63 percent of respondents run containers on Amazon Web Services; up from 44 percent a year ago. Many well-known companies are already running Kubernetes in production on AWS, including CNCF End User Community participants NCSOFT, Ticketmaster, Vevo, and Zalando. AWS has also been an early and important contributor to containerd, the CNCF industry-standard container runtime that provides increased consistency between container orchestration platforms. AWS plans to take an active role in the cloud native community, contributing to Kubernetes and other cloud native technologies such as containerd, CNI, and linkerd.

Cloud Native Computing Foundation (CNCF)

aws

kubernetes on Amazon Web Services

RECENT SURVEY SAYS

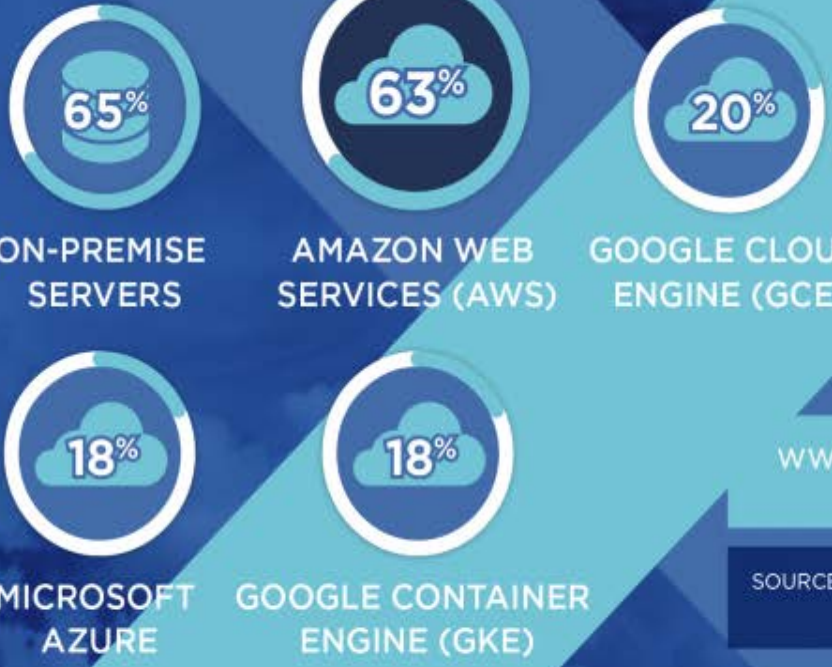
63%

host Kubernetes on Amazon Web Services

A 19 PERCENT INCREASE IN ONE YEAR

Major Companies Run Kubernetes on Amazon Web Services: **NCSOFT, Ticketmaster, Vevo, and Zalando**

Deployments By Environments



www.kubernetes.io
cncf.io

SOURCE: CNCF Survey, March 2017
cncf.io/k8smar17survey

Note: % totals to more than 100 because of companies using multiple environments

Who’s using what?

Some great stories…

Amazon ECS

Kubernetes on AWS

aws

The Ticketmaster way

Kubernetes on AWS

$ whoami

Contractor with a long history of AWS migrations and linux systems. High traffic environments: Channel 4, Ticketmaster

@ric_harvey

https://github.com/richarvey

Deployments

- Two deployment methods for kubernetes at Ticketmaster
  - Tectonic (large multi tenanted clusters)
  - KOPS (product specific clusters)

- Deployment Environments
  - Terraformed VPC’s
  - Everything tagged with a janitor process to clean up
  - Best practice such as only ELB’s in public subnets etc etc

Why Kubernetes and why KOPS?

Kubernetes

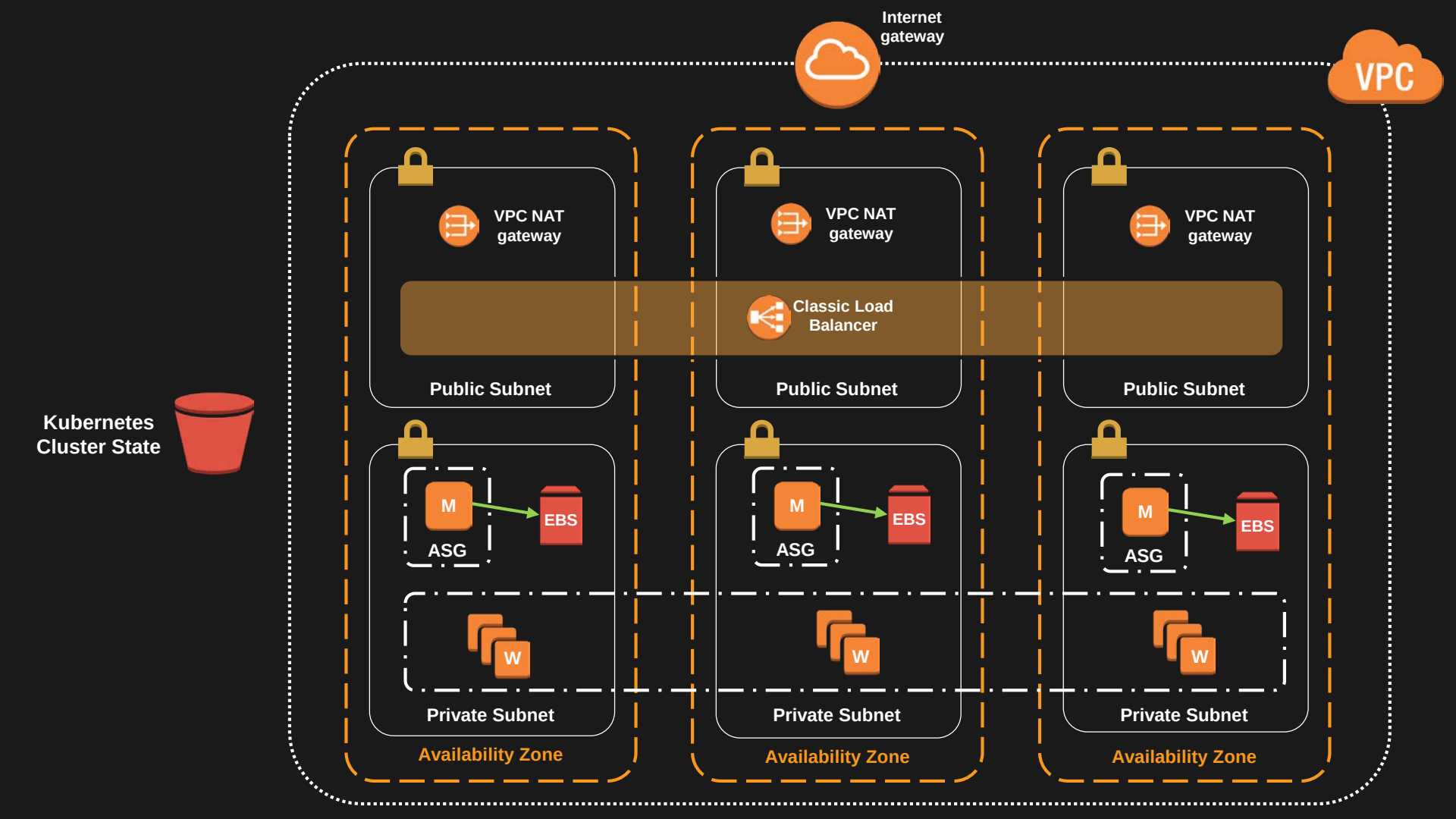
- Community
- Array of tools and features (kubectl exec for example)
- Simplified Orchestration for large amount of products
- Agile and fast deployments from product teams

KOPS

- Ability to deploy into existing VPC’s
- Cost savings with ETCD deployment (important wit lots of clusters)
- Rolling updates that just work
- Kubernetes community led project

10 minute cluster build

Zero to Kubernetes


Internet gateway
VPC
VPC NAT gateway
VPC NAT gateway
VPC NAT gateway
Classic Load Balancer
Public Subnet
Public Subnet
Public Subnet
Kubernetes Cluster State
M
EBS
ASG
M
EBS
ASG
M
EBS
ASG
W
W
W
Private Subnet
Private Subnet
Private Subnet
Availability Zone
Availability Zone
Availability Zone

Cluster Addons

Integrating with AWS

What to do Next

- Dashboard
- Cluster Autoscaler
- kube2IAM
- Logging to ElasticSearch Service via Fluentd

We’re Hiring

Links

- KOPS: https://github.com/kubernetes/kops
- Kube2IAM: https://github.com/jtblin/kube2iam
- Autoscaler: https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler
- Files for this Demo: https://github.com/richarvey/kops-contain-demo

Summary

- AWS is a great choice to run your containers

Thank you