

# 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

# Running containers in development is easy...

```
$ vi Dockerfile
```

```
$ docker build -t mykillerapp:0.0.1 .
```

```
$ docker run -it mykillerapp:0.0.1
```

# 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"?

# 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

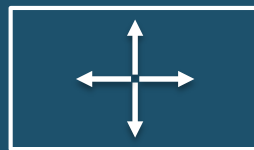
There is **no software** to:



Install

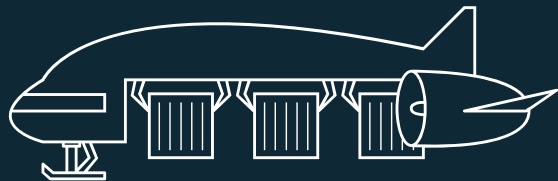


Manage



Scale

# 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

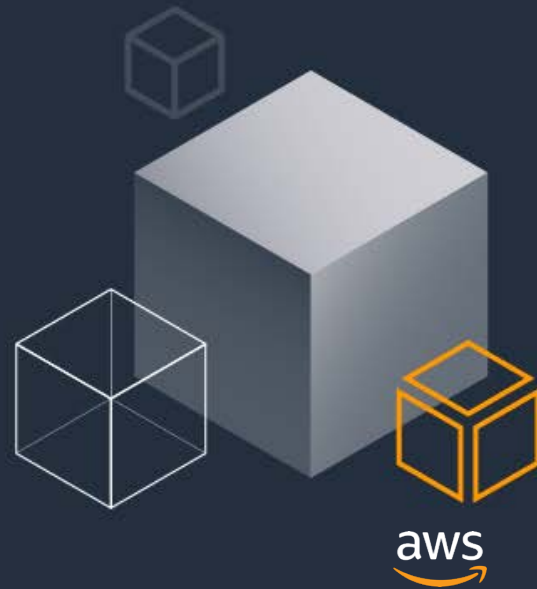


PCI

# 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]
```

The screenshot shows the 'Create Cluster' wizard in the Amazon ECS console. The left sidebar has a navigation menu with 'Clusters' selected, and sub-items for 'Task Definitions' and 'Repositories'. The main content area is titled 'Create Cluster' and includes a descriptive paragraph about clusters. Below this, there are several configuration sections: 'Cluster name\*' with a text input field; a checkbox for 'Create an empty cluster'; 'Instance configuration' with a 'Provisioning Model' section (radio buttons for 'On-Demand Instance' and 'Spot', with 'Spot' selected); a 'Spot instance allocation strategy' section (radio buttons for 'Diversified' and 'Lowest price', with 'Diversified' selected); an 'EC2 instance types\*' dropdown menu; and a 'Maximum bid price (per instance/hour)' input field. At the bottom, there is a link to 'View Spot prices and On-Demand prices to see average prices for each 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
```

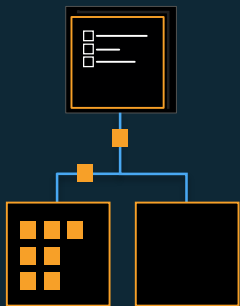
State	Ports	TaskDefinition
mykillerapp	RUNNING.	0.0.0.0:8080->80/tcp mykillerapp:10

# Amazon ECS: Resource Optimisation

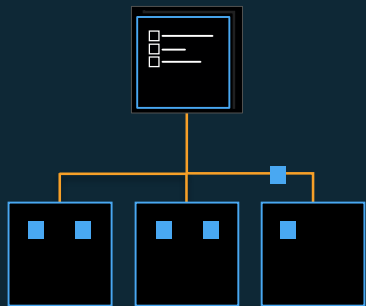




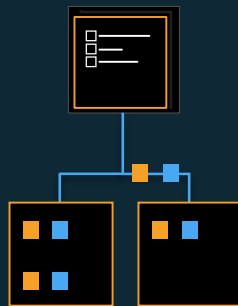
# Placement Strategies



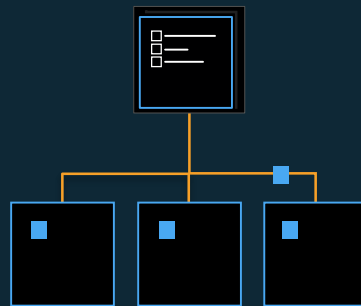
Binpacking



Spread



Affinity

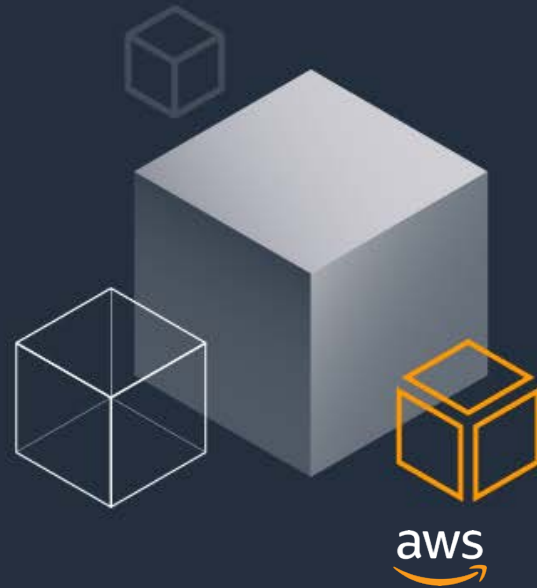


Distinct Instance

# Placement Constraints

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

# Amazon ECS: Service Discovery



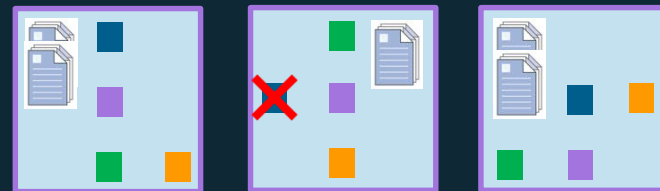
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



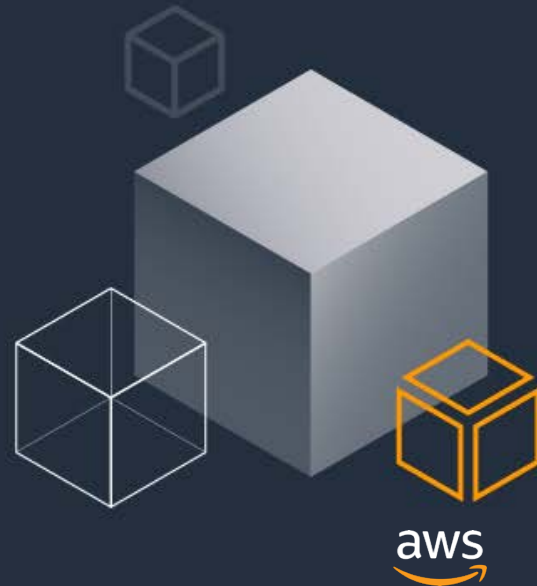
# Service Discovery (DNS)

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

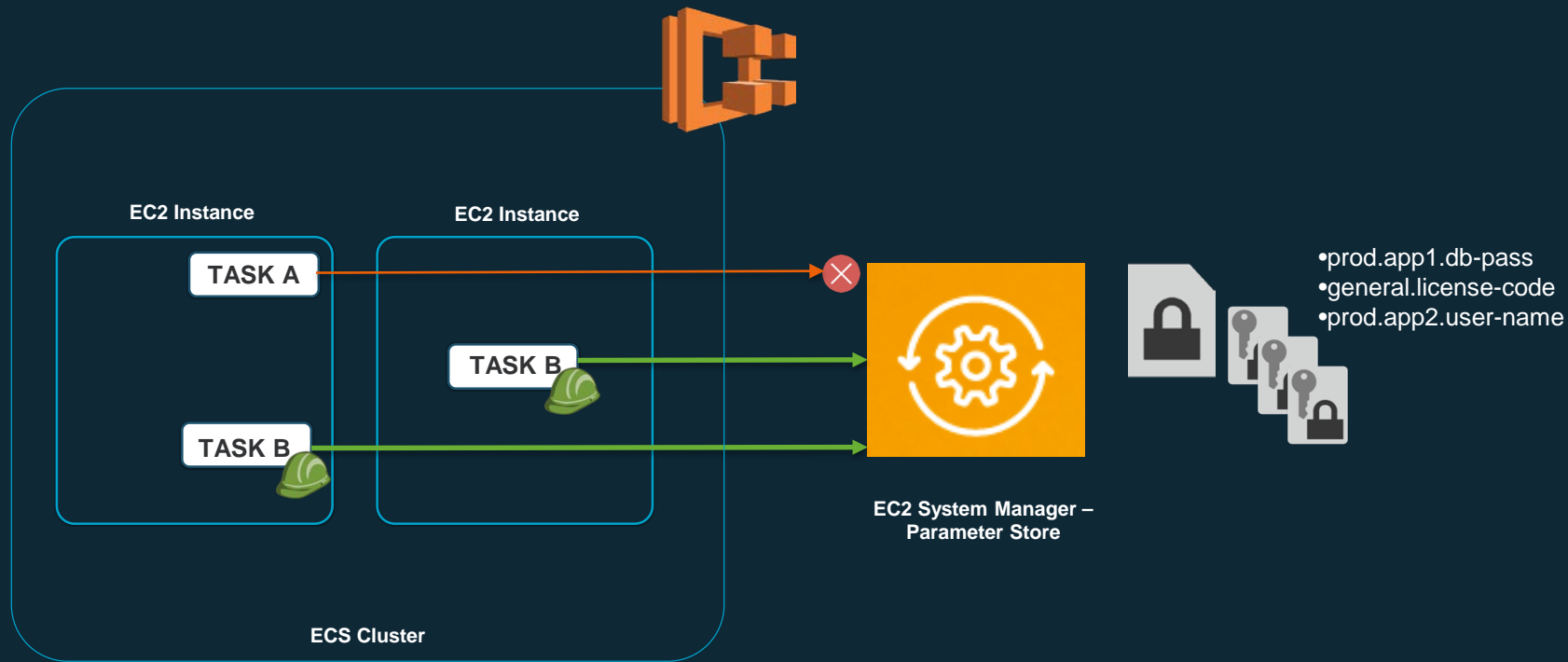
```
def lambda_handler(event, context):  
  
    container_instance = ecs.describe_container_instances(  
        cluster=event['detail']['clusterArn'],  
        containerInstances=[  
            event['detail']['containerInstanceArn'],  
        ]  
    )['containerInstances'][0]  
  
    ec2_instance = 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)  
                            }  
                        ]  
                    }  
                }  
            ]  
        }  
    )
```



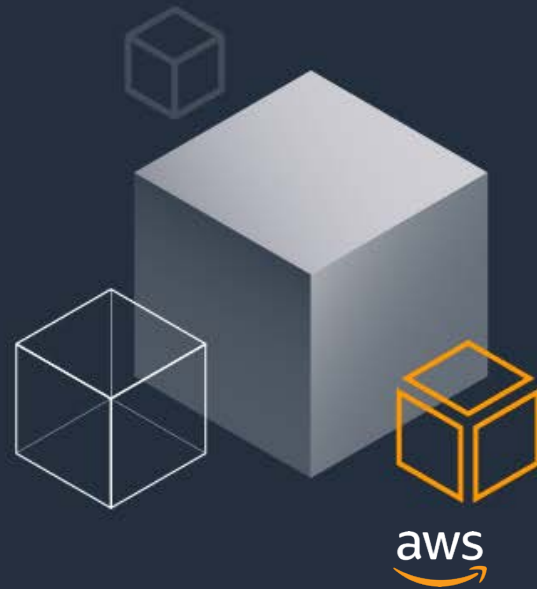
# Amazon ECS: Config / Secrets



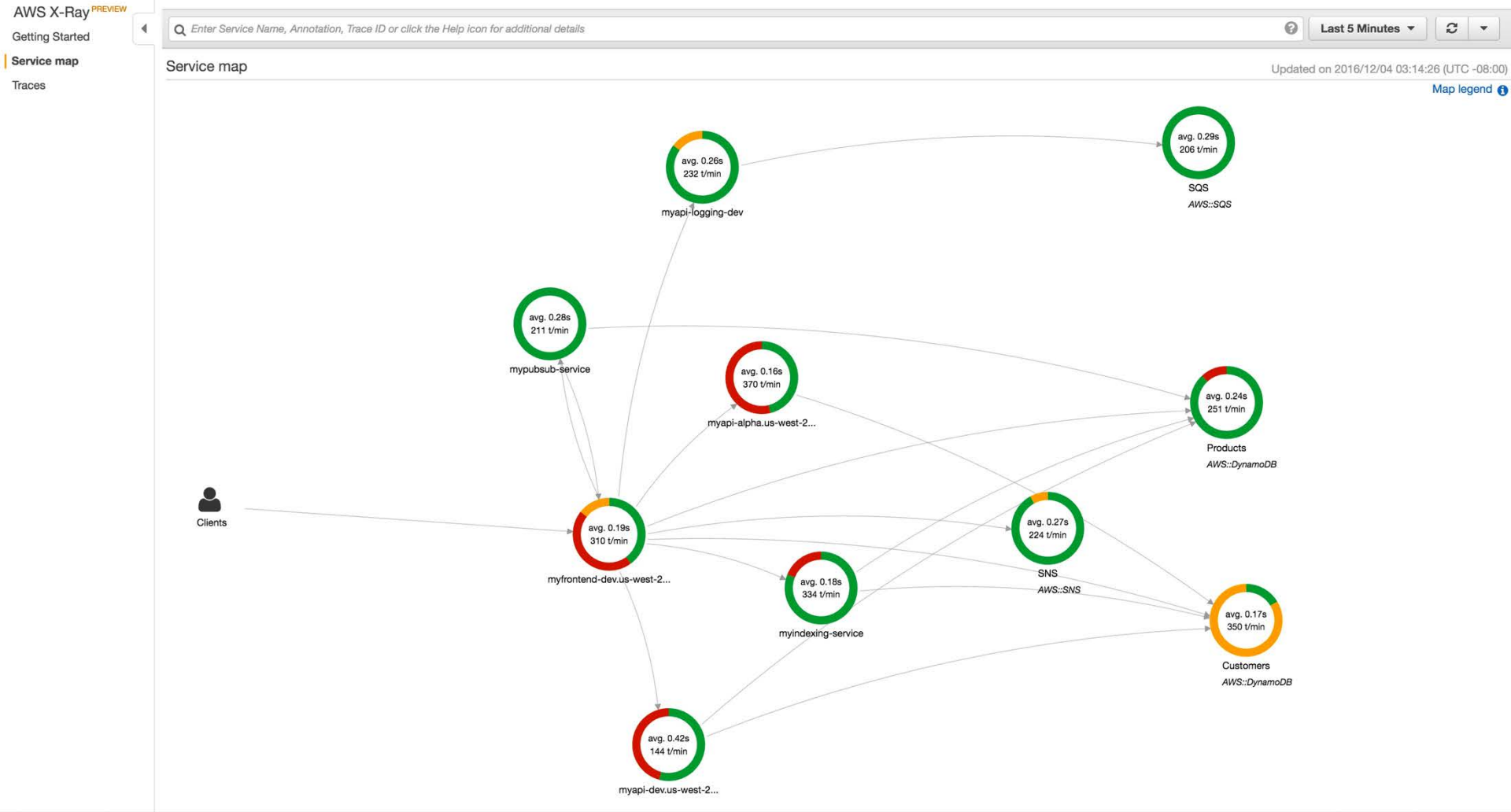
# Configuration / Secrets



# Amazon ECS: Monitoring/logging







# AWS X-Ray: Identify performance bottlenecks

? Last 1 Minute ▾ ↺

Trace overview

Group by: URL ▾

URL ▴	Avg Latency ▴	% of Traces ▾	Response ▴
<a href="http://myfrontend-dev.us-east-1.elasticbeanstalk.com">http://myfrontend-dev.us-east-1.elasticbeanstalk.com</a>	1.2 ms	89.95%	510 OK, 0 Throttled, 0 Errors, 0 Faults
<a href="http://myfrontend-dev.us-east-1.elasticbeanstalk.com">http://myfrontend-dev.us-east-1.elasticbeanstalk.com</a>	900 ms	8.29%	47 OK, 0 Throttled, 0 Errors, 0 Faults
<a href="http://myfrontend-dev.us-east-1.elasticbeanstalk.com">http://myfrontend-dev.us-east-1.elasticbeanstalk.com</a>	0.4 ms	1.76%	10 OK, 0 Throttled, 0 Errors, 0 Faults

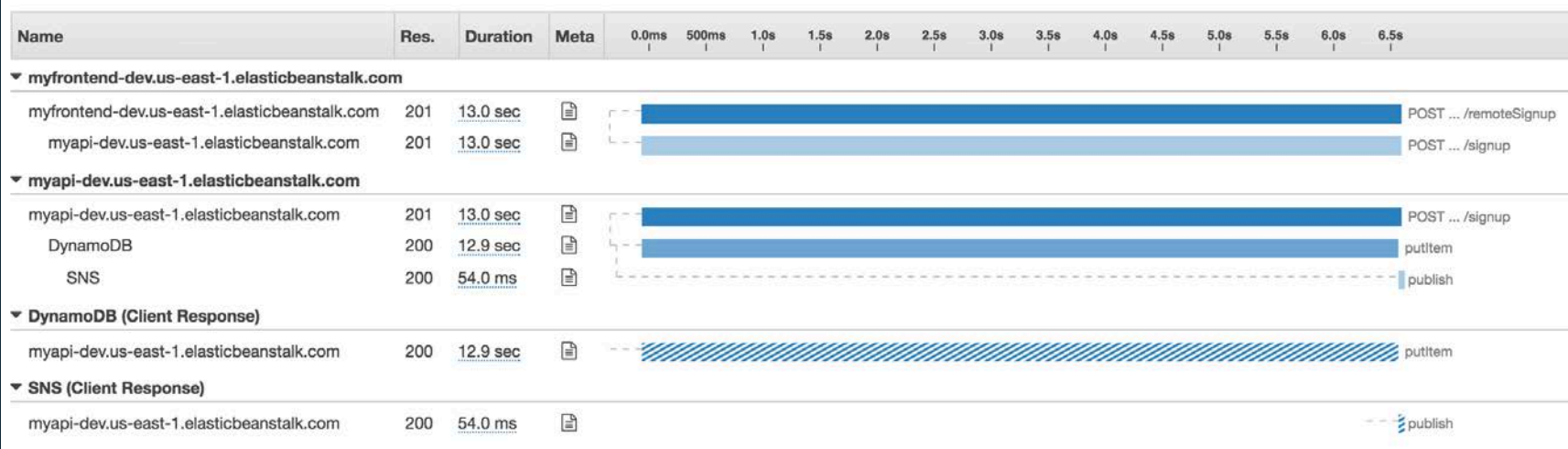
Trace list (567) ⓘ

ID ▴	Age ▾	Method ▴	Response ▴	Latency ▴	URL ▴	Client IP ▴	Annotations ▴
<a href="#">...3635bdce</a>	4.6 sec	GET	404	1.0 ms	<a href="#">http://myfrontend...</a>	54.92.225.233	0
<a href="#">...cf999840</a>	4.6 sec	GET	200	1.0 ms	<a href="#">http://myfrontend...</a>	54.92.225.233	0
<a href="#">...b64c9192</a>	4.6 sec	GET	200	2.0 ms	<a href="#">http://myfrontend...</a>	54.92.225.233	0
<a href="#">...9a96c931</a>	4.6 sec	GET	200	1.0 ms	<a href="#">http://myfrontend...</a>	54.92.225.233	0

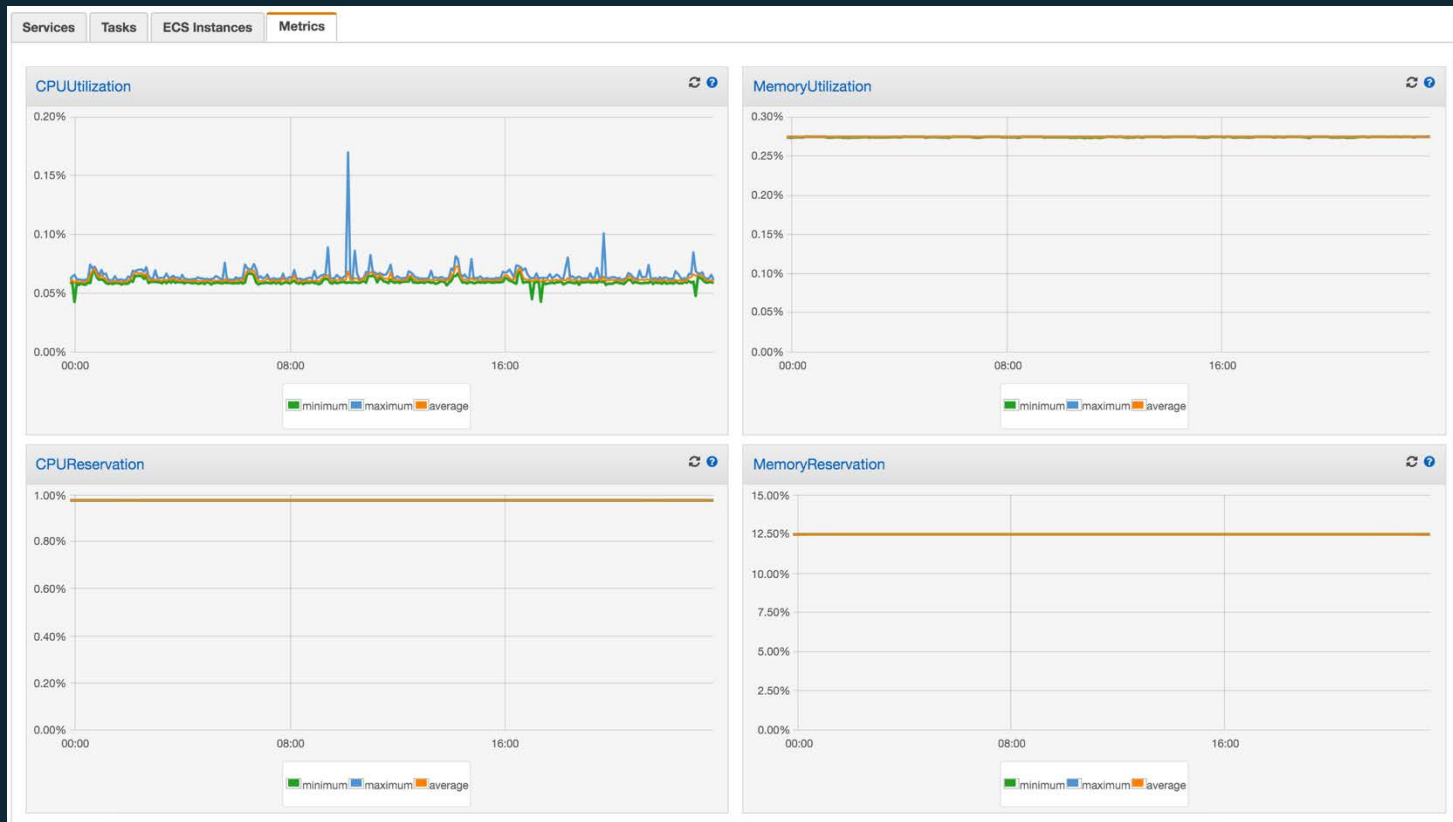
# AWS X-Ray: Identify performance bottlenecks

Traces > 1-583f0cba-642468fbaaafce837e212631

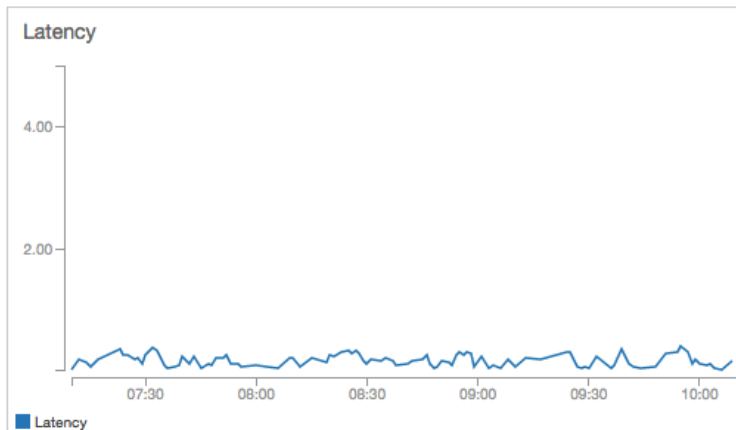
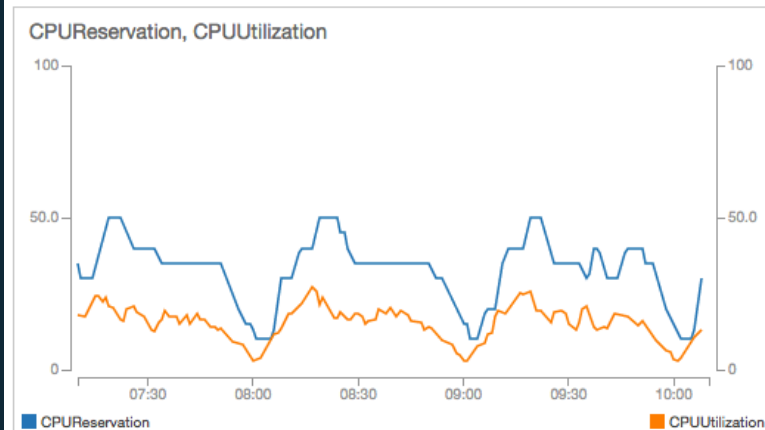
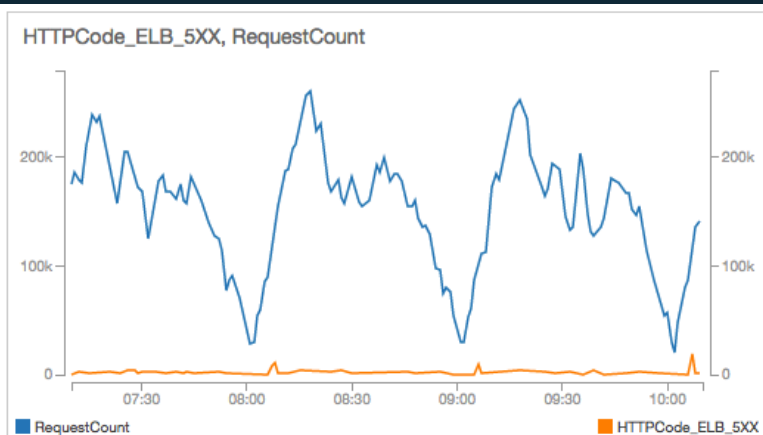
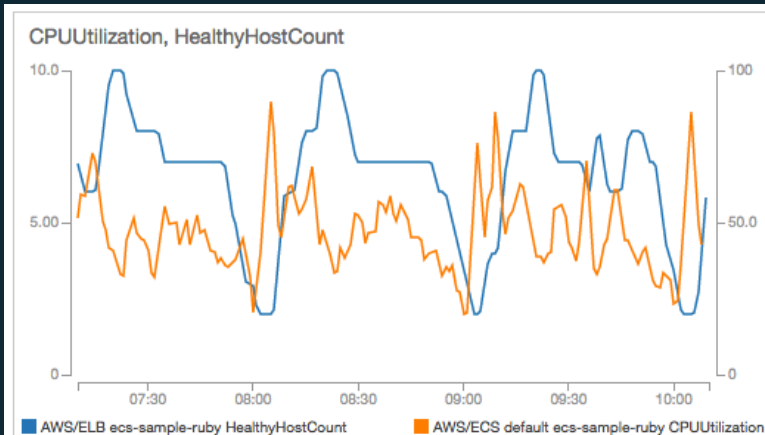
Timeline Raw



# 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](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

# Centralized Logging with CloudWatch Logs

CloudWatch > Log Groups > Production-WebsiteService > 8a2d684337b2cf37c324f221c697ea0c28d8bd841a40a709a540b8fc7957a360

Expand all ☒ Row ☐ Text   

200  all 30s 5m 1h 6h 1d 1w custom ▾

	Time (UTC +00:00)	Message
2016-11-03		
▶	21:48:59	[GIN] 2016/11/03 - 21:48:59   97;42m 200 [0m] 20.560916ms   10.180.23.86:1605   97;44m [0m GET /
▶	21:48:59	[GIN] 2016/11/03 - 21:48:59   97;42m 200 [0m] 20.170986ms   10.180.14.27:63074   97;44m [0m GET /
▶	21:49:09	[GIN] 2016/11/03 - 21:49:09   97;42m 200 [0m] 3.23317ms   10.180.23.86:1625   97;44m [0m GET /
▶	21:49:09	[GIN] 2016/11/03 - 21:49:09   97;42m 200 [0m] 853.79µs   10.180.14.27:63096   97;44m [0m GET /
▶	21:49:19	[GIN] 2016/11/03 - 21:49:19   97;42m 200 [0m] 918.472µs   10.180.23.86:1633   97;44m [0m GET /
▶	21:49:19	[GIN] 2016/11/03 - 21:49:19   97;42m 200 [0m] 879.181µs   10.180.14.27:63102   97;44m [0m GET /
▶	21:49:29	[GIN] 2016/11/03 - 21:49:29   97;42m 200 [0m] 1.113159ms   10.180.23.86:1643   97;44m [0m GET /
▶	21:49:29	[GIN] 2016/11/03 - 21:49:29   97;42m 200 [0m] 921.021µs   10.180.14.27:63110   97;44m [0m GET /
▶	21:49:39	[GIN] 2016/11/03 - 21:49:39   97;42m 200 [0m] 2.395537ms   10.180.23.86:1653   97;44m [0m GET /
▶	21:49:39	[GIN] 2016/11/03 - 21:49:39   97;42m 200 [0m] 971.799µs   10.180.14.27:63118   97;44m [0m GET /
▶	21:49:49	[GIN] 2016/11/03 - 21:49:49   97;42m 200 [0m] 1.163781ms   10.180.23.86:1661   97;44m [0m GET /
▶	21:49:49	[GIN] 2016/11/03 - 21:49:49   97;42m 200 [0m] 970.934µs   10.180.14.27:63128   97;44m [0m GET /
▶	21:49:59	[GIN] 2016/11/03 - 21:49:59   97;42m 200 [0m] 1.10307ms   10.180.23.86:1667   97;44m [0m GET /
▶	21:49:59	[GIN] 2016/11/03 - 21:49:59   97;42m 200 [0m] 870.338µs   10.180.14.27:63136   97;44m [0m GET /
▶	21:50:09	[GIN] 2016/11/03 - 21:50:09   97;42m 200 [0m] 1.058436ms   10.180.23.86:1681   97;44m [0m GET /
▶	21:50:09	[GIN] 2016/11/03 - 21:50:09   97;42m 200 [0m] 898.432µs   10.180.14.27:63148   97;44m [0m GET /



# Tip: Use Metric Filters with CloudWatch Logs

## Define Logs Metric Filter

**Filter for Log Group: Production-WebsiteService**

You can use metric filters to monitor events in a log group as they are sent to CloudWatch Logs. You can monitor and count specific terms or extract values from log events and associate the results with a metric. [Learn more about pattern syntax](#).

**Filter Pattern**

Internal Error

[Show examples](#)

**Select Log Data to Test**

5039d050486cd713a90f150fab00d06a278a0264816b6b7f394510cb62223224

Test Pattern

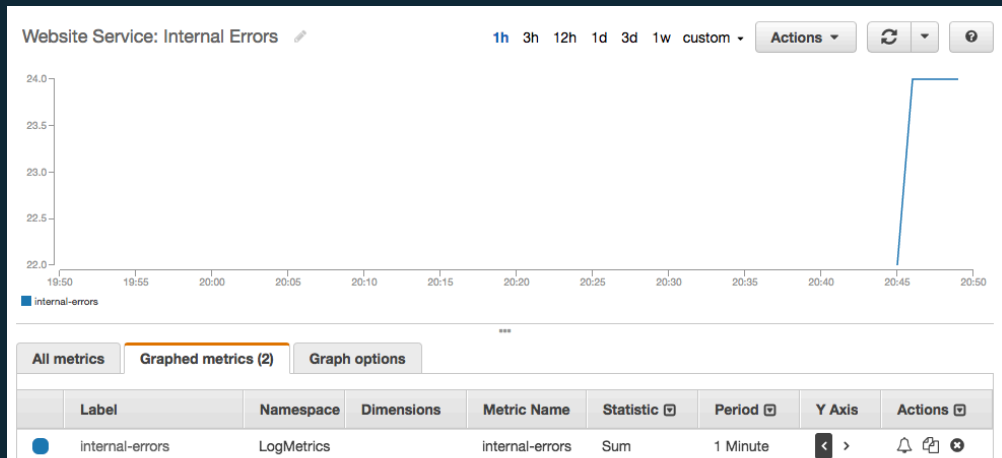
[Clear](#)

[GIN-debug] [WARNING] Running in "debug" mode. Switch to "release" mode in production.  
- using env: export GIN\_MODE=release  
- using code: gin.SetMode(gin.ReleaseMode)  
[GIN-debug] GET / --> main.main.func1 (3 handlers)  
[GIN-debug] Listening and serving HTTP on :8000  
[GIN] 2016/11/03 - 21:48:59 [[97;42m 200 [0m] 17.829212ms | 10.180.14.27:13114 [[97;44m [0m GET /  
[GIN] 2016/11/03 - 21:48:59 [[97;42m 200 [0m] 20.674425ms | 10.180.23.86:13576 [[97;44m [0m GET

**Results**

Found 5 matches out of 50 event(s) in the sample log.

Cancel Assign Metric



# 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)



kubernetes



Prometheus



OPENTRACING



fluentd



linkerd



CoreDNS



rkt



CNI



envoy



JAEGER

# **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



ON-PREMISE  
SERVERS



AMAZON WEB  
SERVICES (AWS)



GOOGLE CLOUD  
ENGINE (GCE)



MICROSOFT  
AZURE



GOOGLE CONTAINER  
ENGINE (GKE)

[www.kubernetes.io](http://www.kubernetes.io)  
[cncf.io](http://cncf.io)

SOURCE: CNCF Survey, March 2017  
[cncf.io/k8smar17survey](http://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





# Kubernetes on AWS

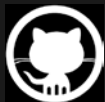
*ticketmaster*®

# \$ 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 with lots of clusters)
- Rolling updates that just work
- Kubernetes community led project

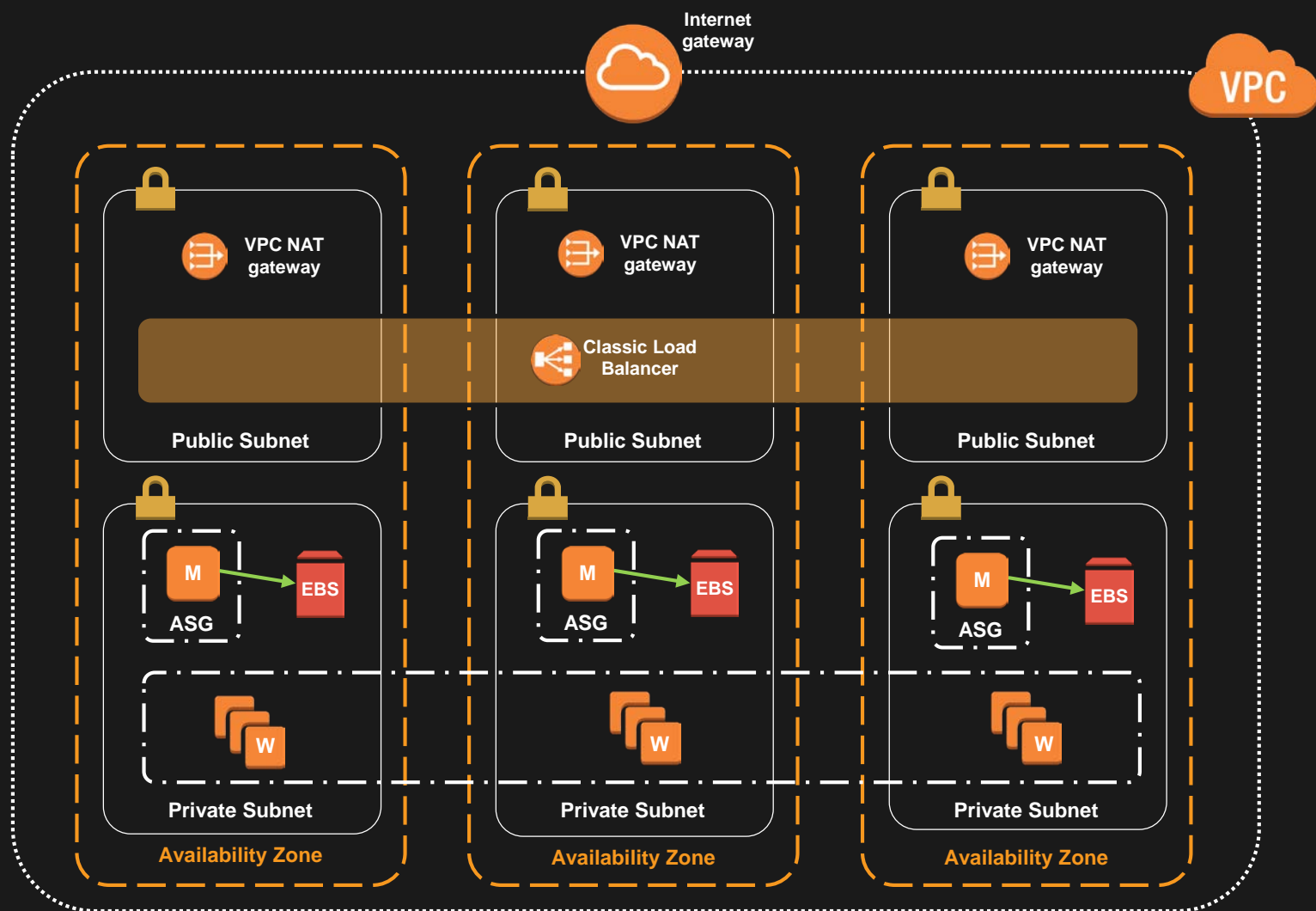
10 minute cluster build

---

# Zero to Kubernetes



Kubernetes  
Cluster State



# Integrating with AWS



# What to do Next

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

***ticketmaster***<sup>®</sup>

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