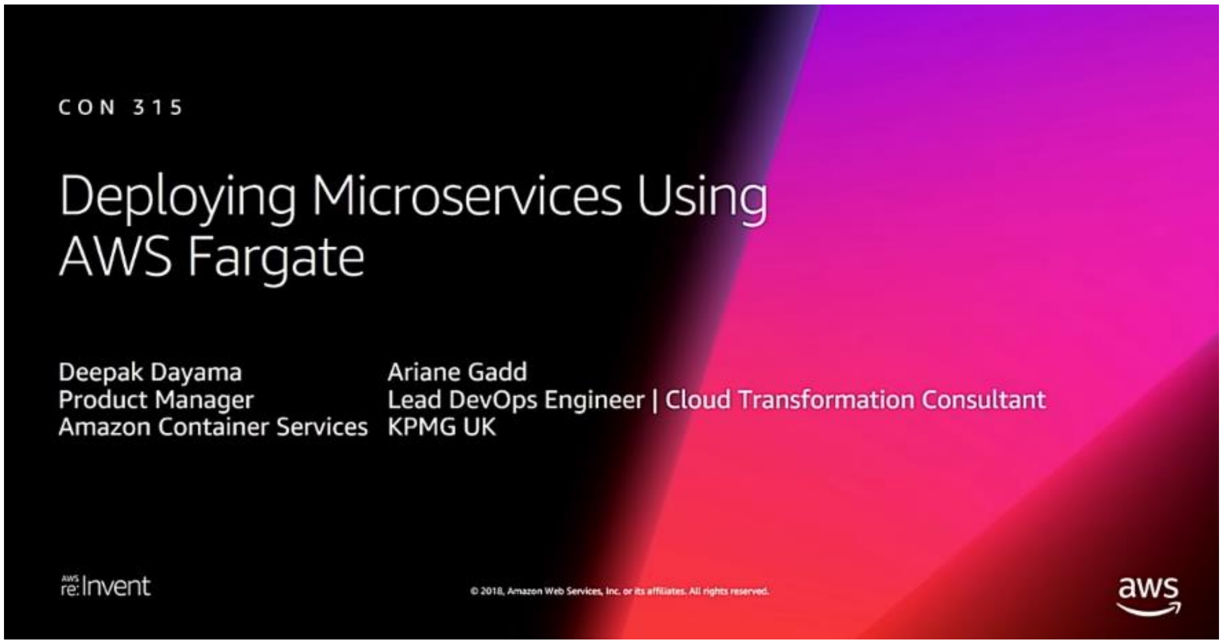

CON 315

Deploying Microservices Using AWS Fargate

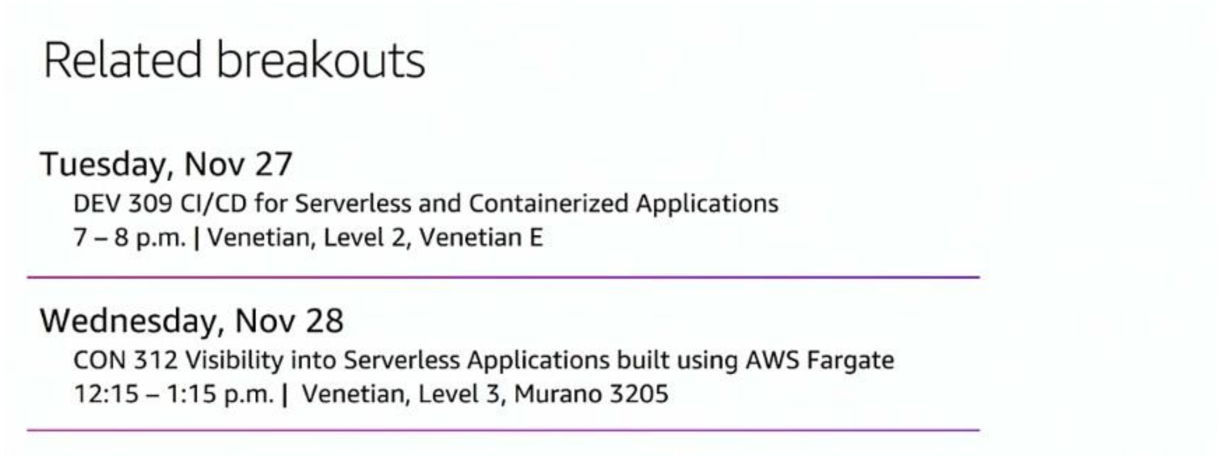
Deepak Dayama
Product Manager
Amazon Container Services

Ariane Gadd
Lead DevOps Engineer | Cloud Transformation Consultant
KPMG UK

re:Invent

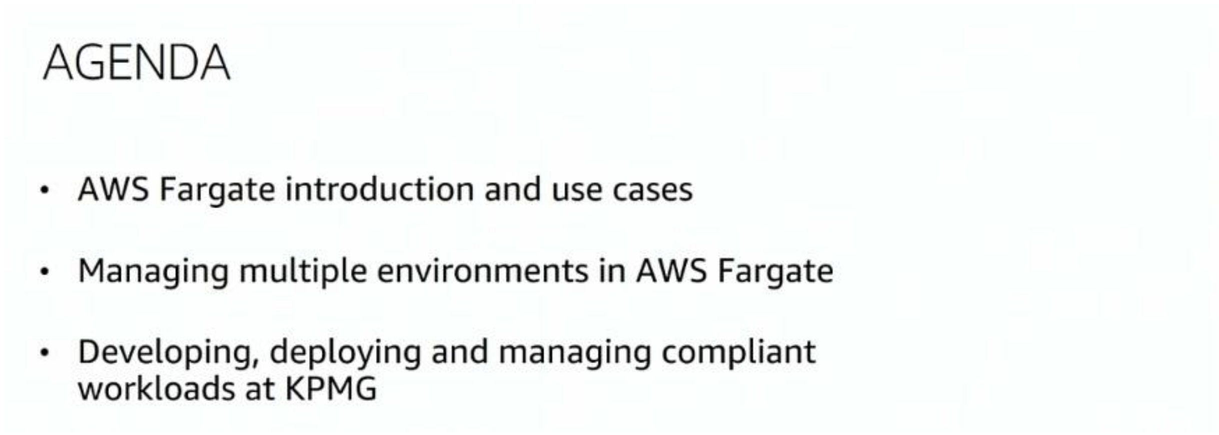
aws

KPMG have built a customer due diligence solution for a high-profile banking client in AWS. **The solution is made up of a number of microservices which are deployed to containers using AWS Fargate**. This presentation will dive into the details of the architecture of the solution, how the infrastructure and applications are deployed using third party tools such as Hashicorp's Terraform and Jenkins, and the best practices when running containers in production workloads. The presentation will cover details on the AWS resources used in the solution, including DynamoDB, ECS, Fargate and S3, CI/CD and automation, with a focus around security to meet banking regulatory requirements. We will look at how KPMG have configured for canary deployments to ECS Fargate, how we manage secrets management and encryption, and how we manage service discovery between the microservices using ECS Service Discovery and Route 53.

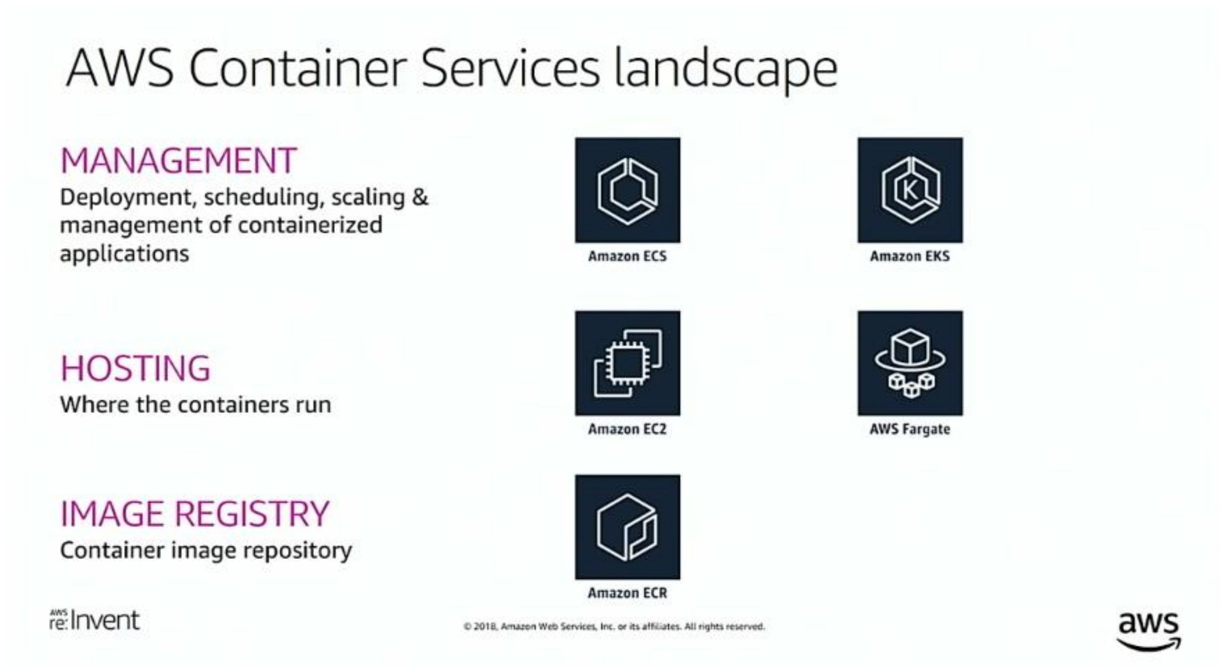

Related breakouts

Tuesday, Nov 27
DEV 309 CI/CD for Serverless and Containerized Applications
7 – 8 p.m. | Venetian, Level 2, Venetian E

Wednesday, Nov 28
CON 312 Visibility into Serverless Applications built using AWS Fargate
12:15 – 1:15 p.m. | Venetian, Level 3, Murano 3205


AGENDA

- AWS Fargate introduction and use cases
- Managing multiple environments in AWS Fargate
- Developing, deploying and managing compliant workloads at KPMG



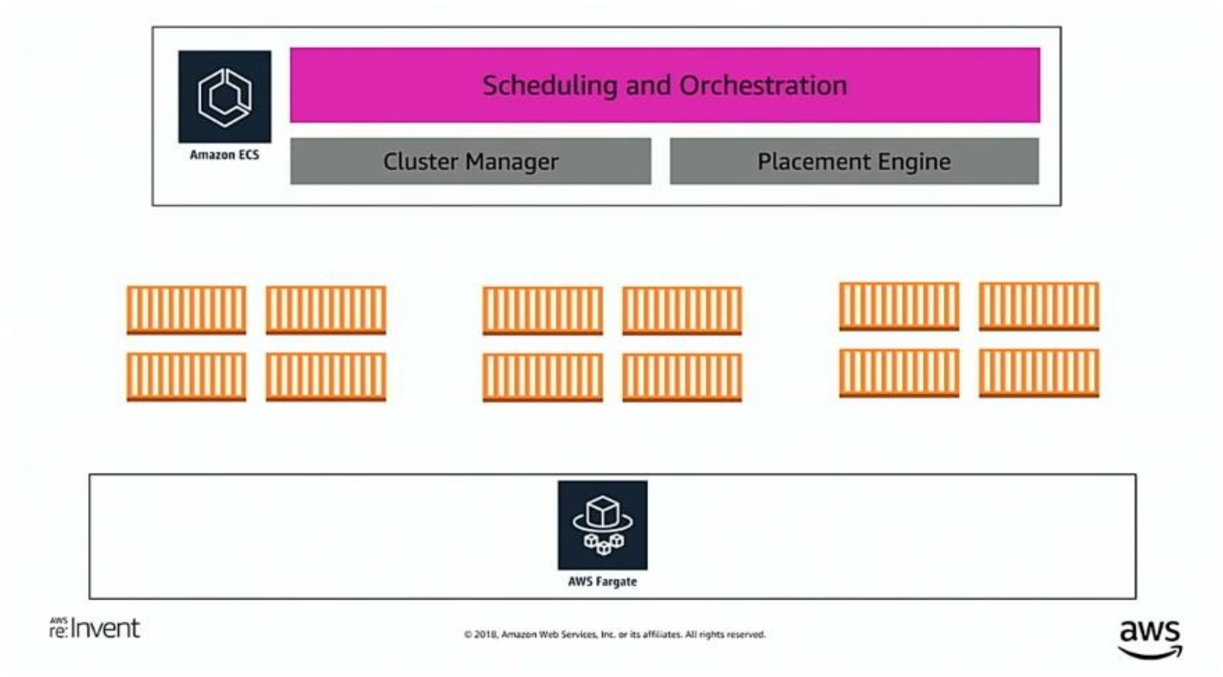
AWS Container Services landscape

MANAGEMENT
Deployment, scheduling, scaling & management of containerized applications

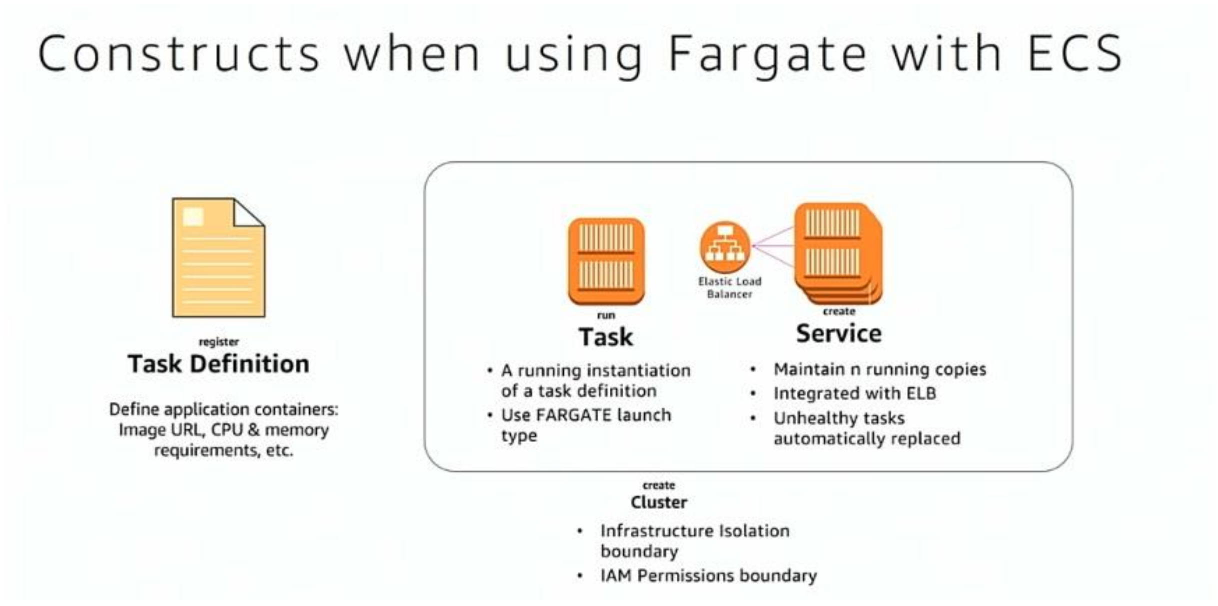
HOSTING
Where the containers run

IMAGE REGISTRY
Container image repository

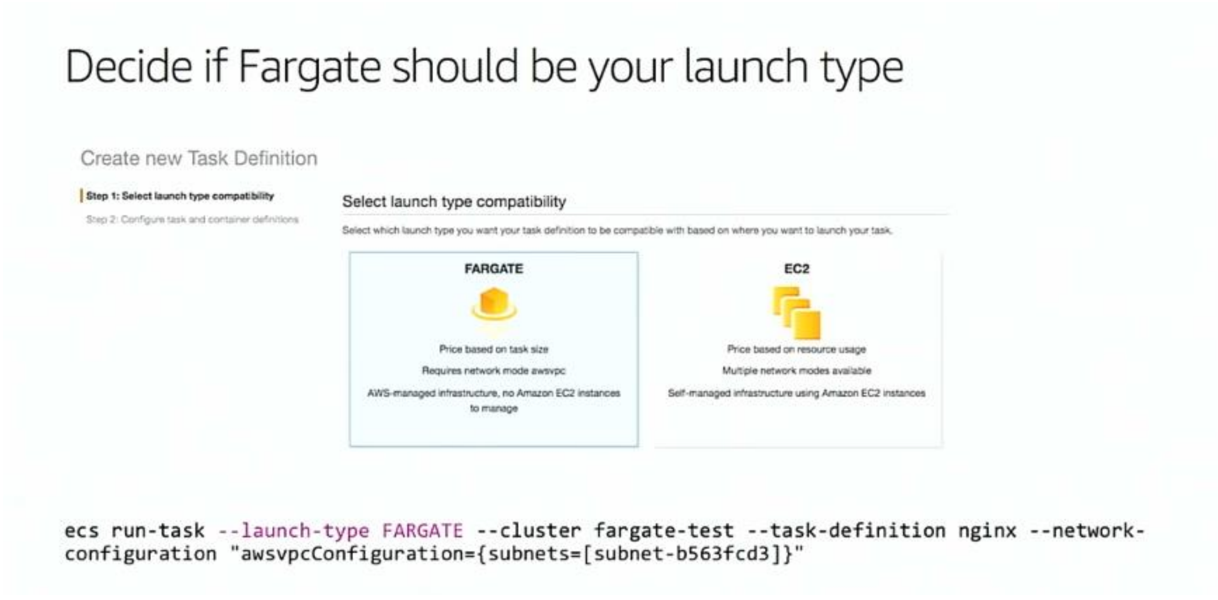
Amazon ECS
Amazon EKS
Amazon EC2
AWS Fargate
Amazon ECR
re:Invent
aws



AWS Fargate is the underlying layer that is not exposed to you but it helps you run the containers.


Constructs when using Fargate with ECS
register
Task Definition
Define application containers: Image URL, CPU & memory requirements, etc.
run
Task
• A running instantiation of a task definition
• Use FARGATE launch type
Elastic Load Balancer
create
Service
• Maintain n running copies
• Integrated with ELB
• Unhealthy tasks automatically replaced
create
Cluster
• Infrastructure Isolation boundary
• IAM Permissions boundary

the Task Definition defines the parameters for running your containers, you can have up to 10 containers defined, what is the networking and ports to be used. The applications will run inside a cluster as a standalone task you manage or run it as a service that will be fronted by a LB.



Decide if Fargate should be your launch type

Create new Task Definition

Step 1: Select launch type compatibility

Step 2: Configure task and container definitions

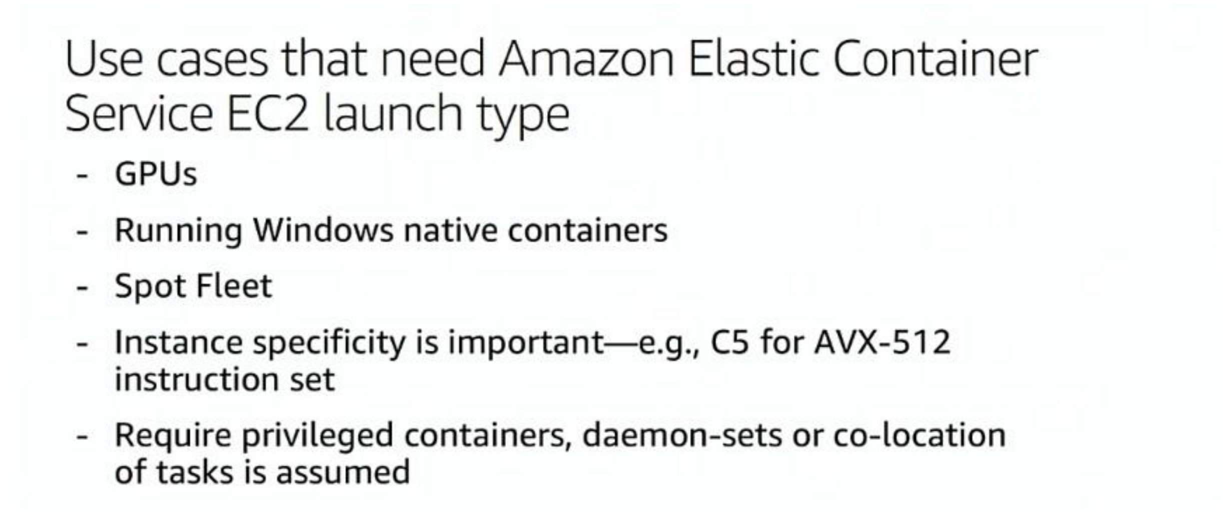
Select launch type compatibility

Select which launch type you want your task definition to be compatible with based on where you want to launch your task.

FARGATE	EC2
Price based on task size	Price based on resource usage
Requires network mode awsvpc	Multiple network modes available
AWS-managed infrastructure, no Amazon EC2 instances to manage	Self-managed infrastructure using Amazon EC2 instances

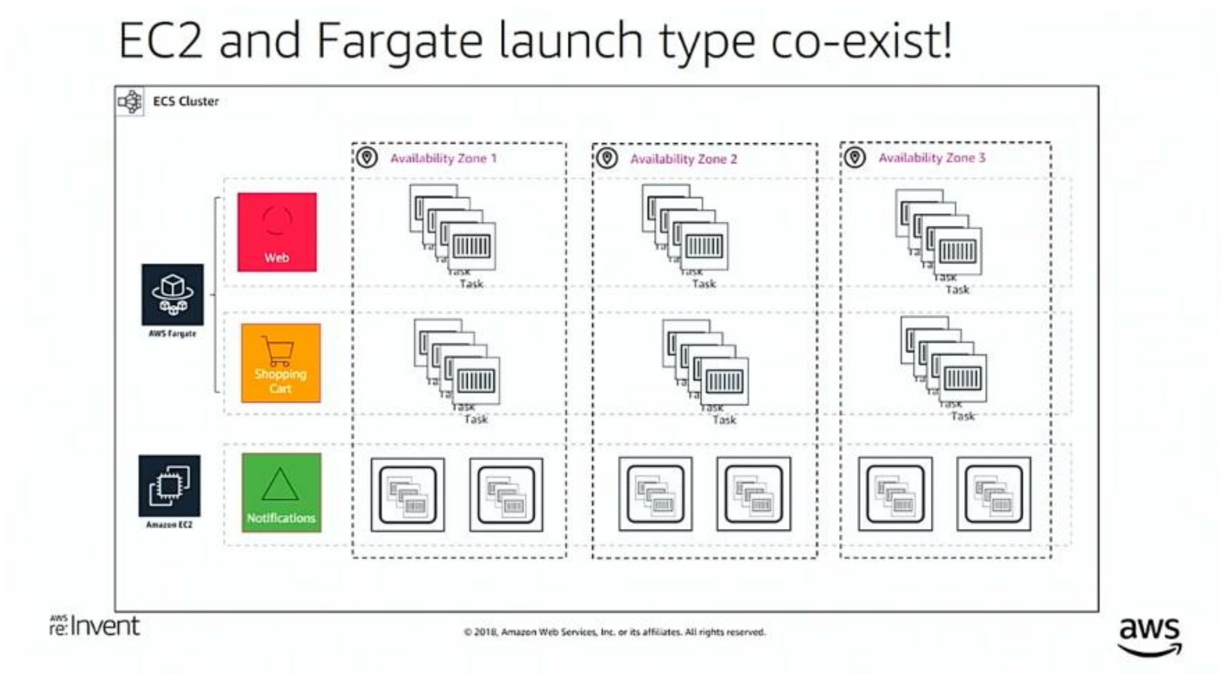
```
ecs run-task --launch-type FARGATE --cluster fargate-test --task-definition nginx --network-
configuration "awsvpcConfiguration={subnets=[subnet-b563fcd3]}"
```

You can run your applications on EC2 instances or on Fargate



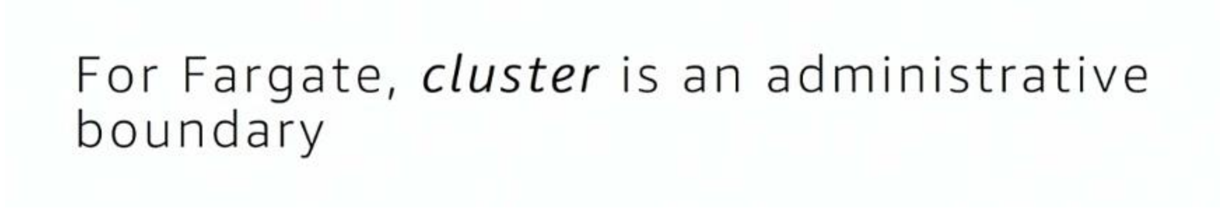
Use cases that need Amazon Elastic Container Service EC2 launch type

- GPUs
- Running Windows native containers
- Spot Fleet
- Instance specificity is important—e.g., C5 for AVX-512 instruction set
- Require privileged containers, daemon-sets or co-location of tasks is assumed



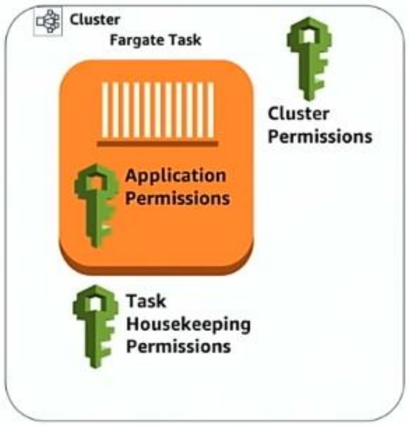
EC2 and Fargate launch type co-exist!

They can run in the same cluster boundary and even within the same VPC subnet, you just need to specify that you want to use the EC2 or Fargate instance launch type.



For Fargate, *cluster* is an administrative boundary

Permission tiers



Cluster permissions:

Who can run/see tasks in the cluster?

Application (task) permissions:

Which of my AWS resources can this application access?

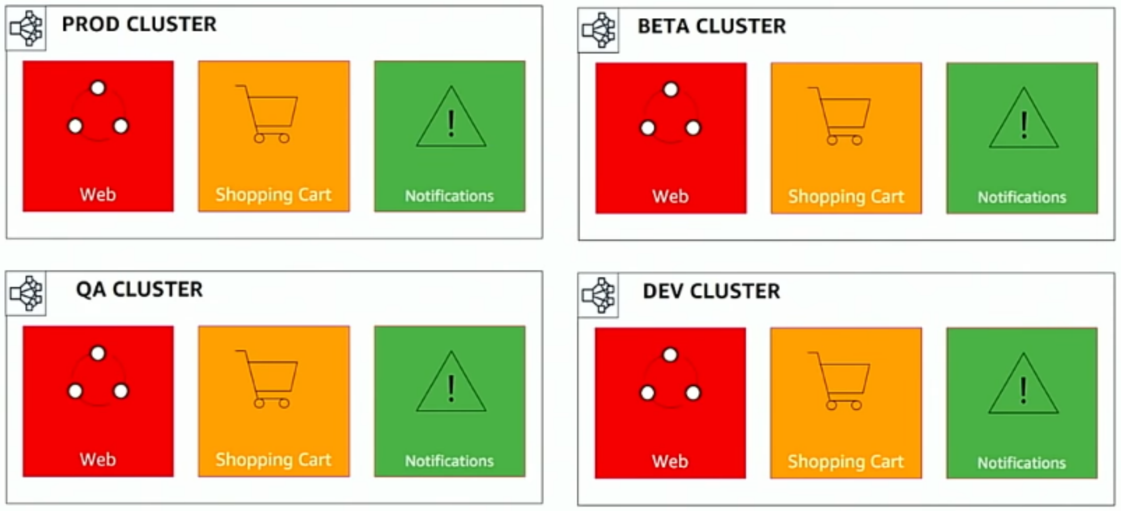
Housekeeping permissions:

What permissions do I want to grant ECS to perform?

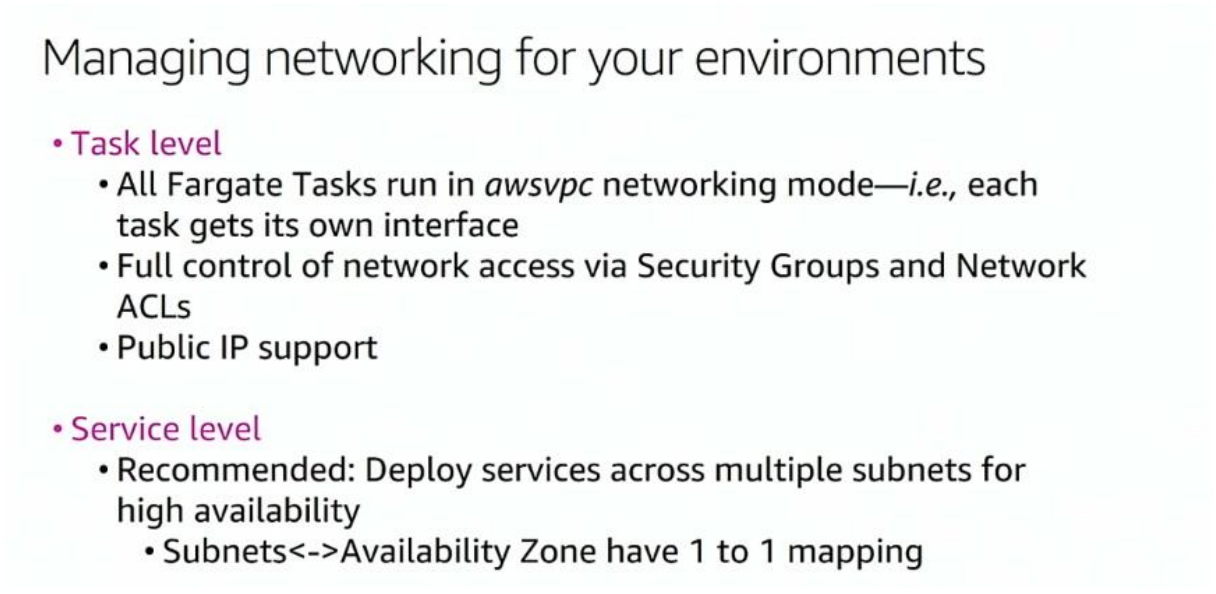
For example

- ECR image pull
- CloudWatch Logs pushing
- ENI creation
- Register/deregister targets into ELB

Managing multiple environments



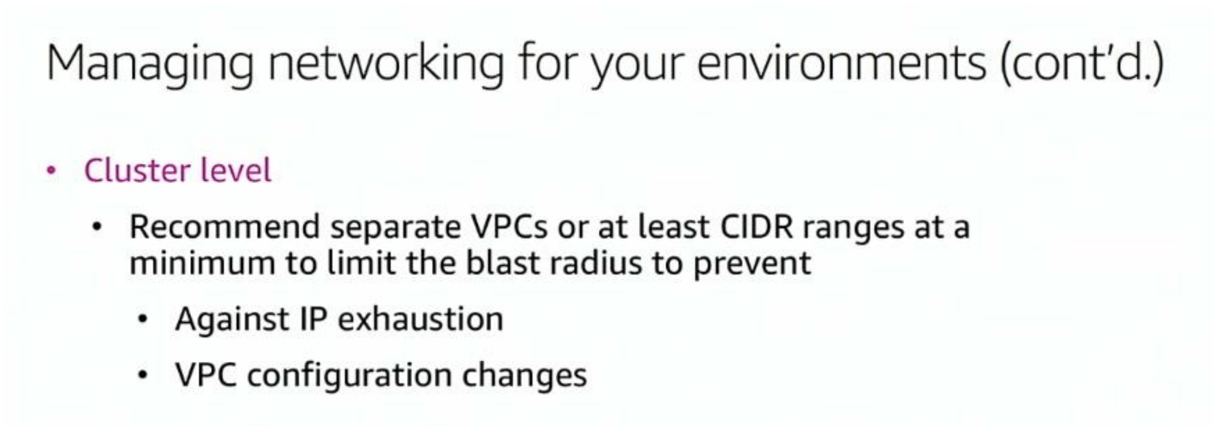
Networking



Managing networking for your environments

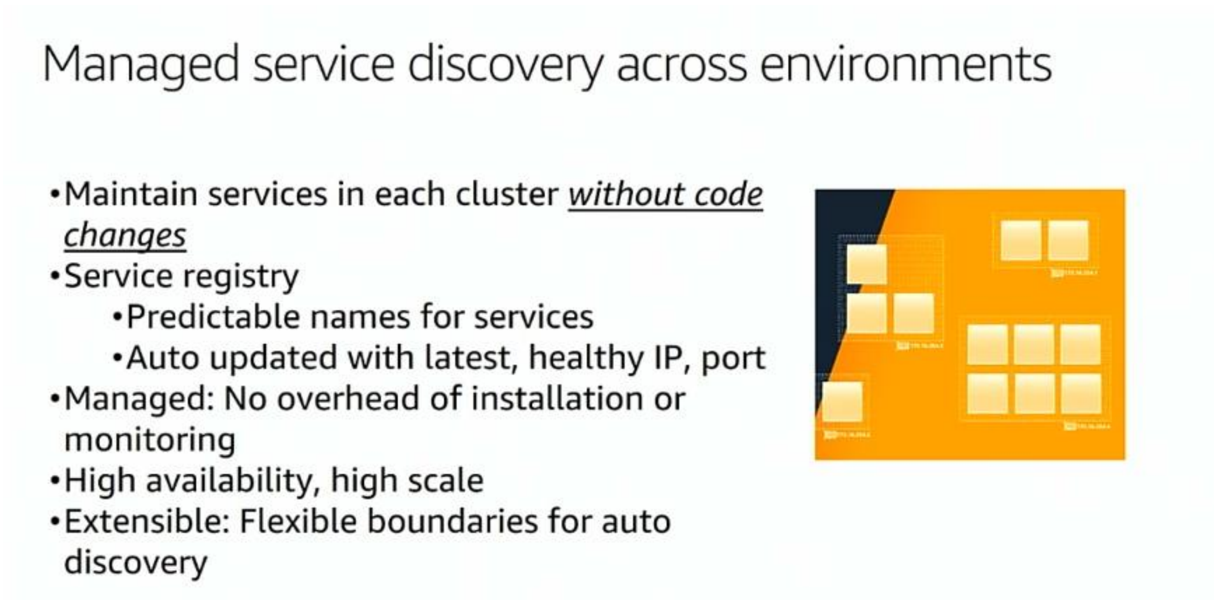
- Task level
  - All Fargate Tasks run in *awsvpc* networking mode—*i.e.*, each task gets its own interface
  - Full control of network access via Security Groups and Network ACLs
  - Public IP support
- Service level
  - Recommended: Deploy services across multiple subnets for high availability
    - Subnets<->Availability Zone have 1 to 1 mapping

Fargate tasks run only in your VPC in the designated subnets you choose for Fargate to use to run your applications in.



Managing networking for your environments (cont'd.)

- Cluster level
  - Recommend separate VPCs or at least CIDR ranges at a minimum to limit the blast radius to prevent
    - Against IP exhaustion
    - VPC configuration changes

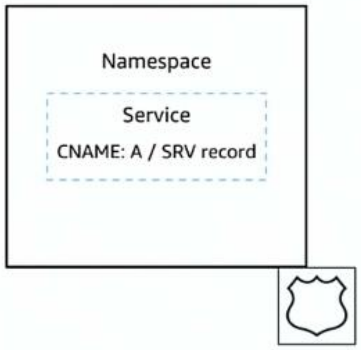


Managed service discovery across environments

- Maintain services in each cluster ***without code changes***
- Service registry
  - Predictable names for services
  - Auto updated with latest, healthy IP, port
- Managed: No overhead of installation or monitoring
- High availability, high scale
- Extensible: Flexible boundaries for auto discovery

For applications within different clusters or environments to talk to each other, they can use service discovery.

Amazon Route 53 auto naming provides service registry



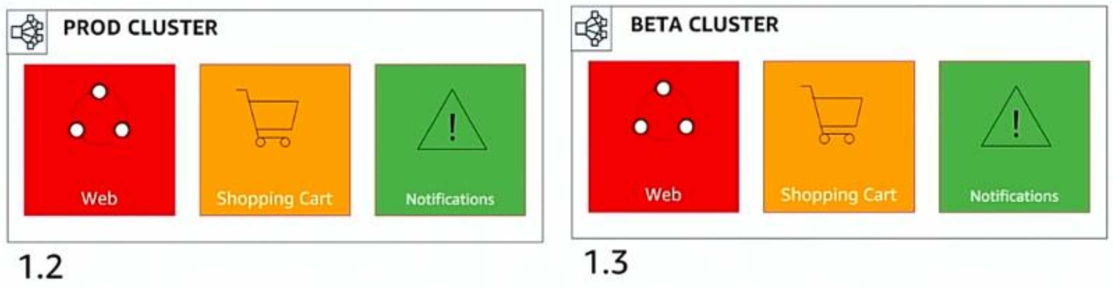
Route 53 provides APIs to create
- Namespace
- CNAME per service auto name
- A records per task IP
- SRV records per task IP + port

Fargate platform versions

Using platform versions for managing environments

- A version refers to specific runtime environment for the task—agent, docker daemon, OS, etc.
  - Available versions—1.0, 1.1 and 1.2
- Features changes go in to new platform versions
- You can specify a version number or as *LATEST* pointer
  - DOES NOT mean we auto-upgrade as soon as new versions are available
  - Only the next user-triggered deployment upgrades to the new version

Qualifying platform versions for your services



- *NOTE:* Versions exist per service and not at the cluster-level
- Choose 'Force New Deployment' when calling Update Service across versions

Managing compliant workloads

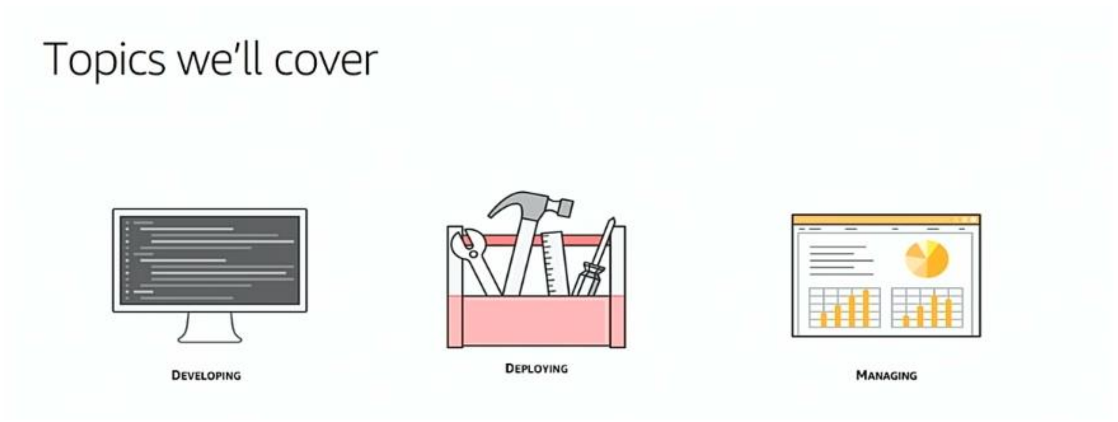
- Both EC2 and Fargate Launch Types in parity for compliance
  - SOC 1,2,3
  - PCI
  - ISO 9001 / 27001 / 27017 / 27018
  - HIPAA

Customers using Fargate at scale

AppZen Turner
GE CATALYTIC Yummly accenture FireEye
HARRY'S ancestry VERITONE Aptiv BEACHBODY
SKYWATCH CBS SPORTS DIGITAL Talroo ÓNICA ZIFF DAVIS
Smartronix CORTEVA agriscience ironSource LEVEL access KPMG

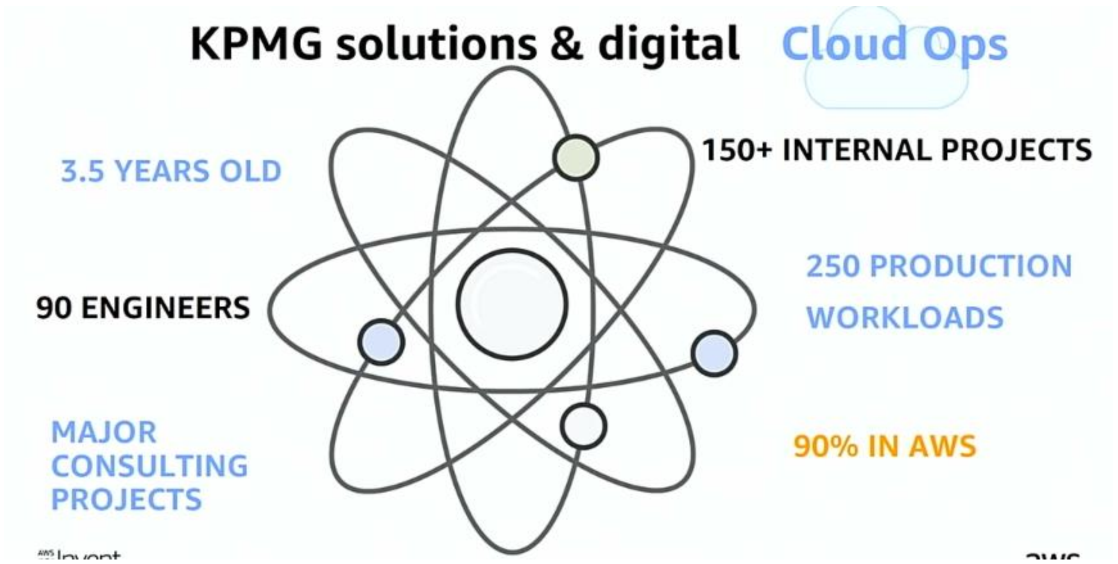
KPMG

Ariane Gadd



Topics we'll cover

DEVELOPING
DEPLOYING
MANAGING



KPMG solutions & digital Cloud Ops

We work globally across all sectors

BANKING SECTOR
80% of transformation projects are global blue-chip banks

TAX & AUDIT
Multiple projects for internal tax & audit functions

GOVERNMENT SECTOR
Largest project to date for civil service

FMCG
Built compensation scheme for Tesco PLC

Customer due diligence (CDD) solution

Know your customer for a multi-national blue-chip bank

Running successfully in the cloud for over a year

Sold onto another blue-chip bank

Current architecture

- A lot of pets, SQL 2008 on EC2
- .NET deployments to Windows EC2
- Have to manage regular patching, maintenance of servers
- Requirement for server level access

re:Invent

aws

Evolution to the new CDD

Amazon EC2 → AWS Fargate

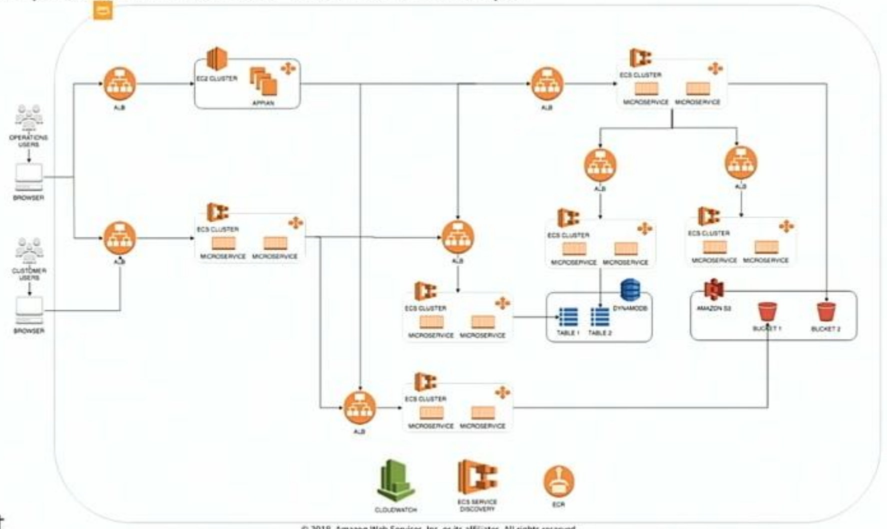
TC + Octopus Deploy → docker + Jenkins

CloudFormation → HashiCorp Terraform

re:Invent

aws

Current platform microservice relationships



re:Invent

aws

Benefits of a managed container service

Immutable deployments, no logging into servers!

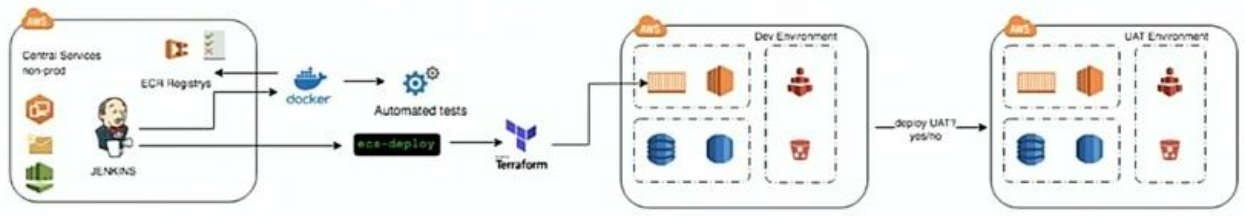
Cluster resource provisioning is handled by AWS Fargate

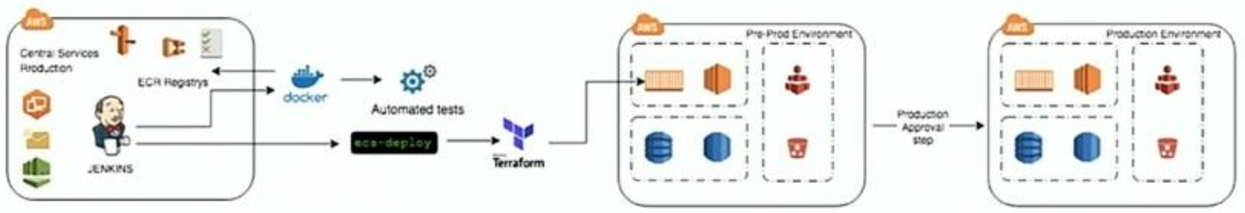
No patching of underlying operating systems

No need for provisioning, configuring or scaling of machines

- ➢ Smaller surface area for attacks

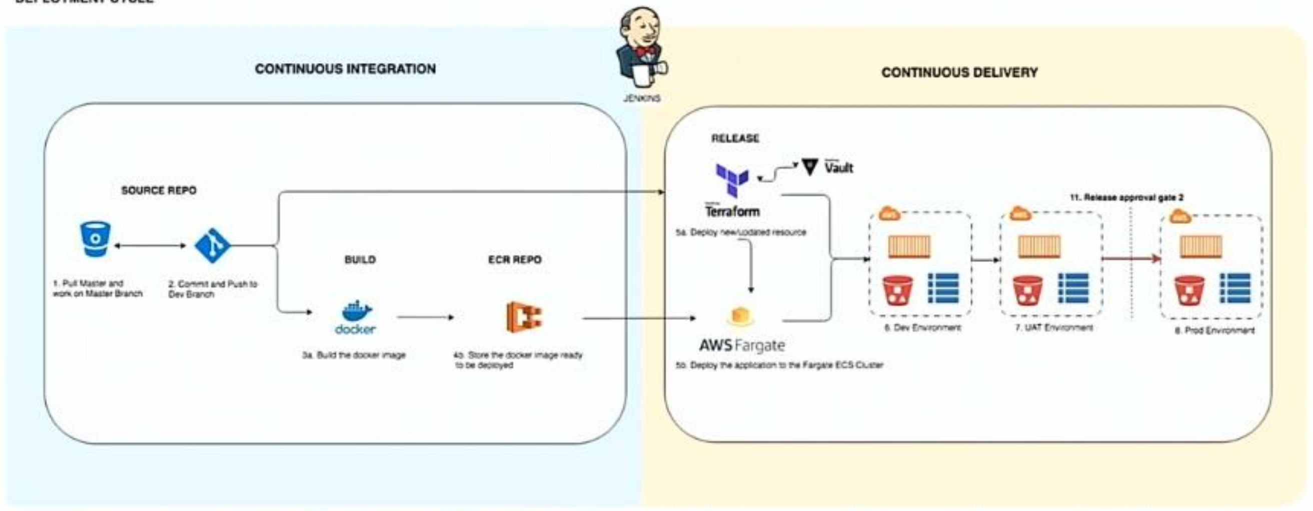
Path to production

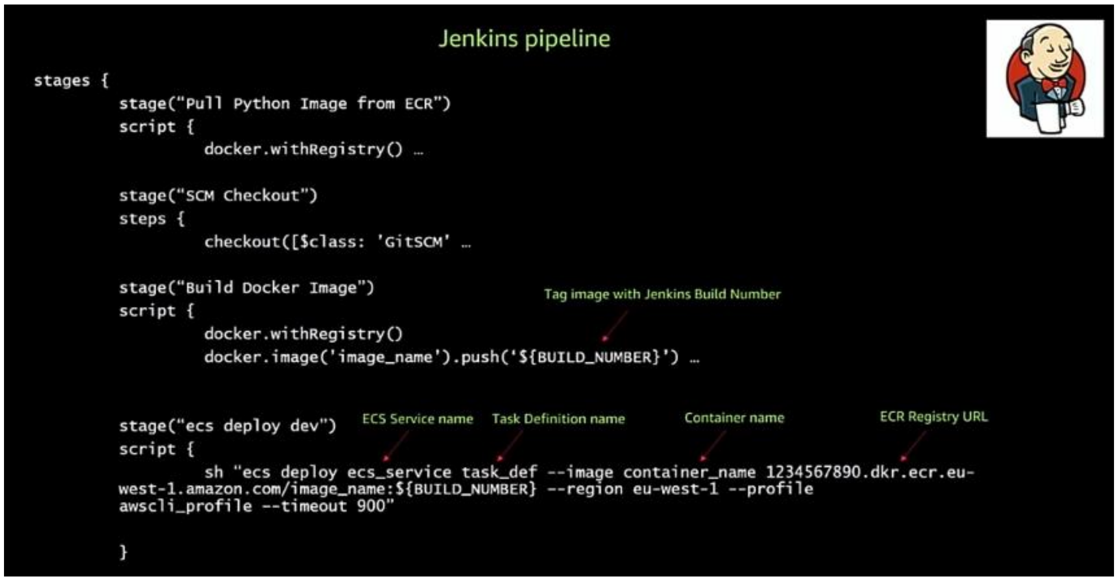




Deployment pipeline

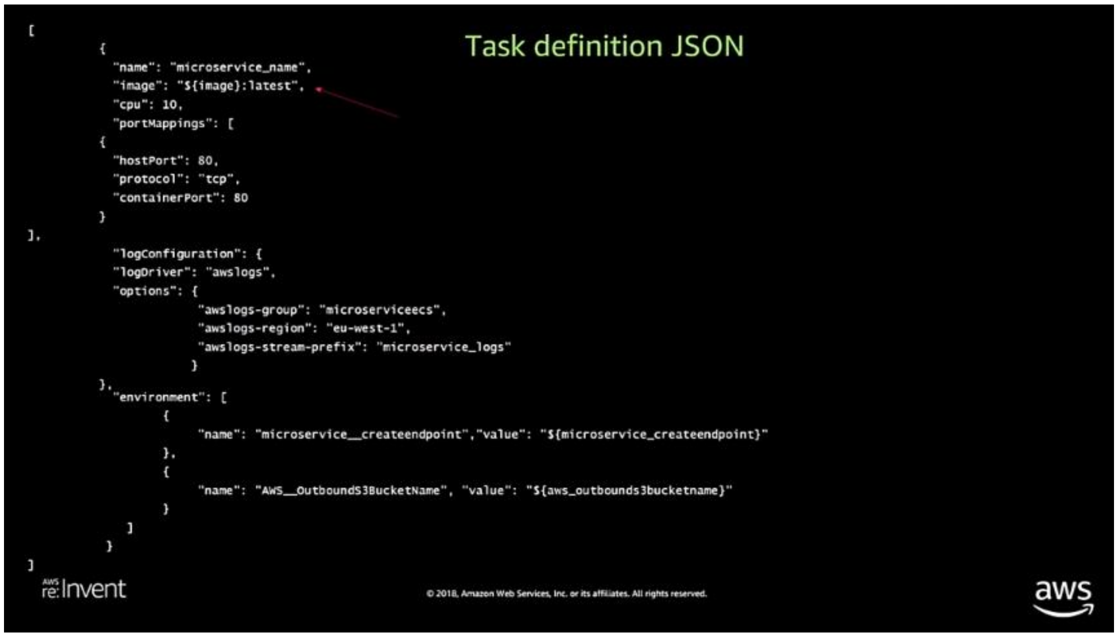
DEPLOYMENT CYCLE


CONTINUOUS INTEGRATION
JENKINS
CONTINUOUS DELIVERY
SOURCE REPO
1. Pull Master and work on Master Branch
2. Commit and Push to Dev Branch
BUILD
docker
3a. Build the docker image
ECR REPO
4b. Store the docker image ready to be deployed
RELEASE
Terraform
Vault
5a. Deploy new/updated resource
AWS Fargate
5b. Deploy the application to the Fargate ECS Cluster
11. Release approval gate 2
6. Dev Environment
7. UAT Environment
8. Prod Environment



Jenkins pipeline

```
stages {
        stage("Pull Python Image from ECR")
        script {
                docker.withRegistry() …

        stage("SCM Checkout")
        steps {
                checkout([$class: 'GitSCM' …

        stage("Build Docker Image")
        script {
                docker.withRegistry()
                docker.image('image_name').push('${BUILD_NUMBER}') …


        stage("ecs deploy dev")
        script {
                sh "ecs deploy ecs_service task_def --image container_name 1234567890.dkr.ecr.eu-
        west-1.amazon.com/image_name:${BUILD_NUMBER} --region eu-west-1 --profile
        awscli_profile --timeout 900"

        }
```

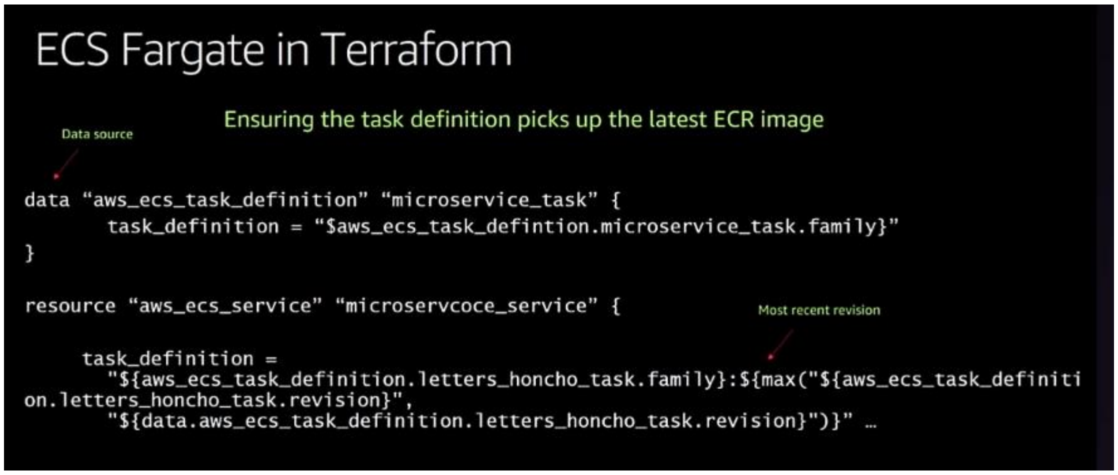


Task definition JSON

```
[
        {
          "name": "microservice_name",
          "image": "${image}:latest",
          "cpu": 10,
          "portMappings": [
        {
          "hostPort": 80,
          "protocol": "tcp",
          "containerPort": 80
        }
],
          "logConfiguration": {
          "logDriver": "awslogs",
          "options": {
                    "awslogs-group": "microserviceecs",
                    "awslogs-region": "eu-west-1",
                    "awslogs-stream-prefix": "microservice_logs"
                   }
        },
          "environment": [
                {
                    "name": "microservice__createendpoint","value": "${microservice_createendpoint}"
                },
                {
                    "name": "AWS__OutboundS3BucketName", "value": "${aws_outbounds3bucketname}"
                }
            ]
         }
]
```

re:Invent © 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved. aws

This is the task definition JSON file that is deployed by Terraform infrastructure-as-code pipeline

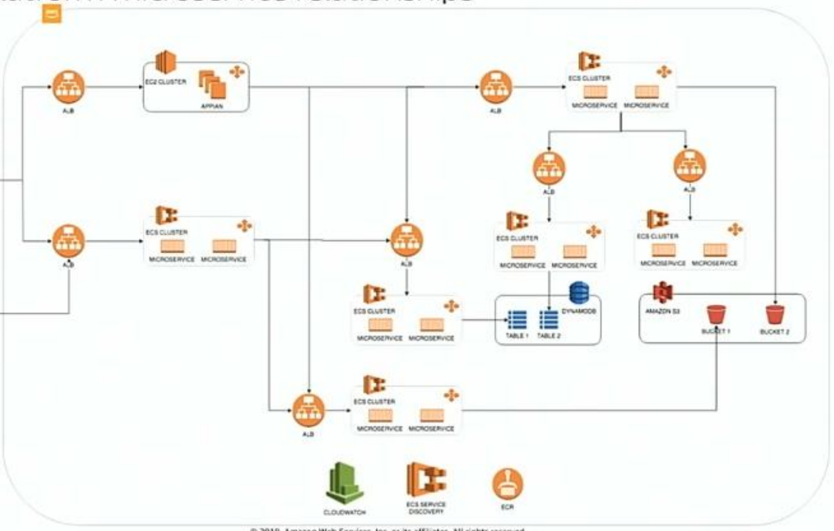


ECS Fargate in Terraform

Ensuring the task definition picks up the latest ECR image

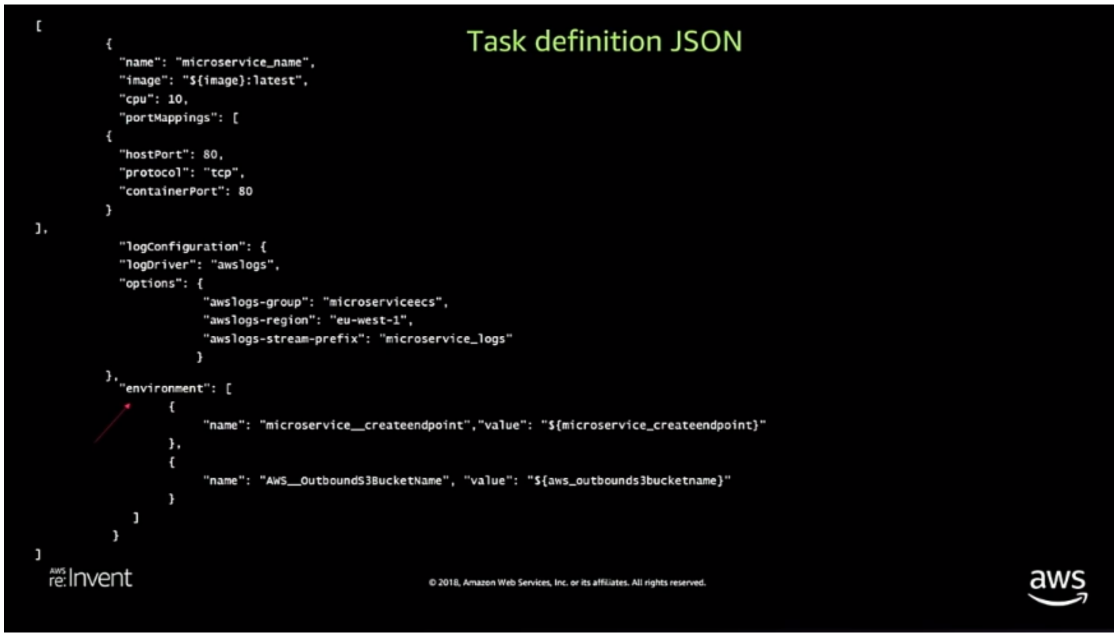
```
data "aws_ecs_task_definition" "microservice_task" {
       task_definition = "$aws_ecs_task_defintion.microservice_task.family}"
}

resource "aws_ecs_service" "microservcoce_service" {

     task_definition =
       "${aws_ecs_task_definition.letters_honcho_task.family}:${max("${aws_ecs_task_definiti
on.letters_honcho_task.revision}",
       "${data.aws_ecs_task_definition.letters_honcho_task.revision}")}" …
```

Current platform microservice relationships



re:Invent

aws

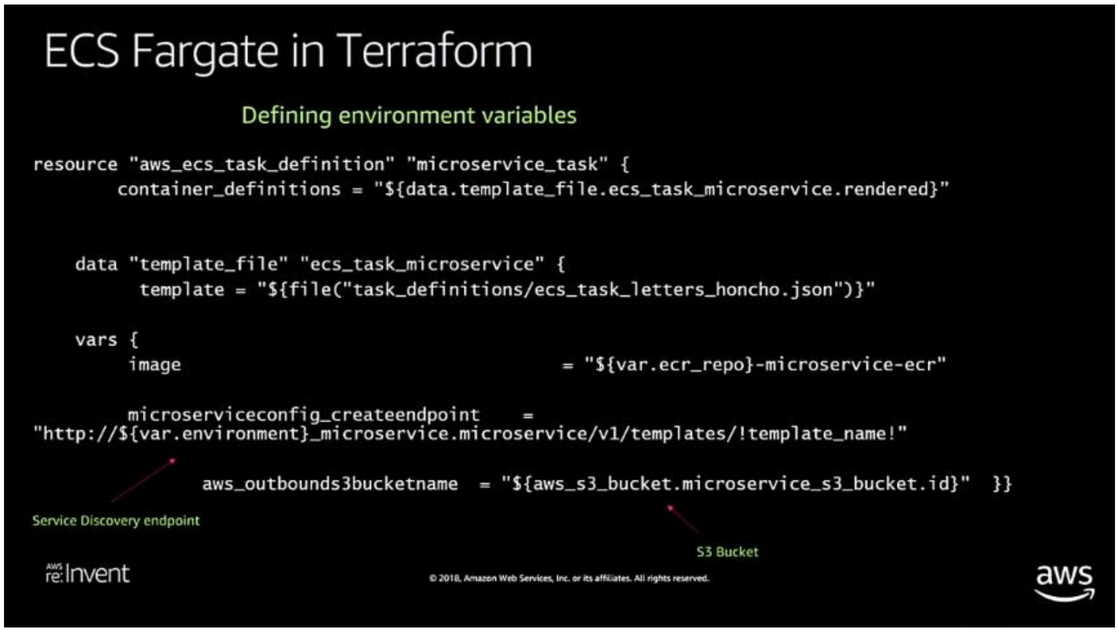


Task definition JSON

```
[
        {
          "name": "microservice_name",
          "image": "${image}:latest",
          "cpu": 10,
          "portMappings": [
        {
          "hostPort": 80,
          "protocol": "tcp",
          "containerPort": 80
        }
],
          "logConfiguration": {
          "logDriver": "awslogs",
          "options": {
                     "awslogs-group": "microserviceecs",
                     "awslogs-region": "eu-west-1",
                     "awslogs-stream-prefix": "microservice_logs"
                    }
        },
          "environment": [
                {
                     "name": "microservice__createendpoint","value": "${microservice_createendpoint}"
                },
                {
                     "name": "AWS__OutboundS3BucketName", "value": "${aws_outbounds3bucketname}"
                }
             ]
         }
]
```

re:Invent

aws



ECS Fargate in Terraform

Defining environment variables

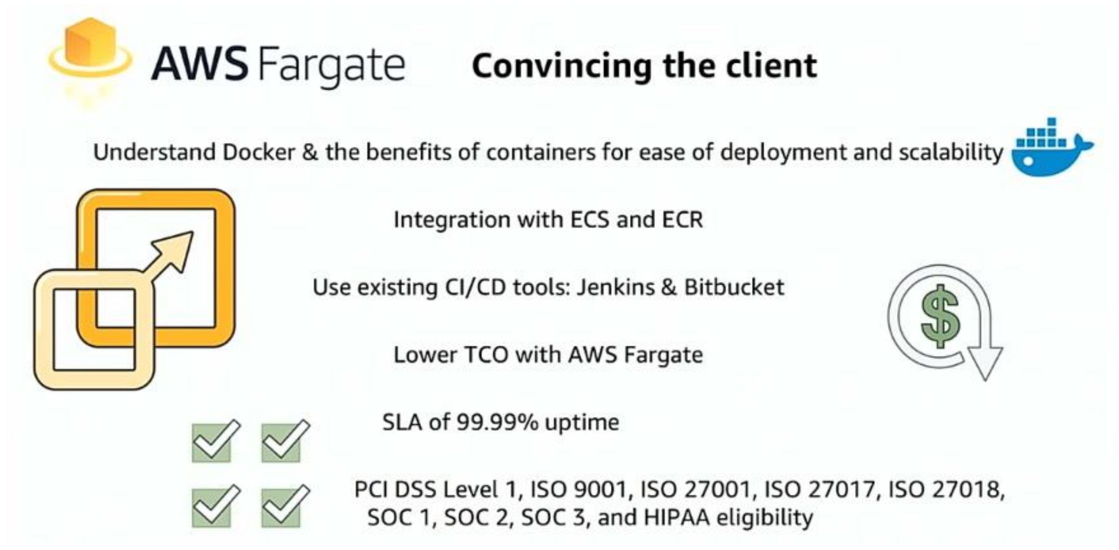
```
resource "aws_ecs_task_definition" "microservice_task" {
       container_definitions = "${data.template_file.ecs_task_microservice.rendered}"


    data "template_file" "ecs_task_microservice" {
         template = "${file("task_definitions/ecs_task_letters_honcho.json")}"

    vars {
        image                                  = "${var.ecr_repo}-microservice-ecr"

        microserviceconfig_createendpoint     =
"http://${var.environment}_microservice.microservice/v1/templates/!template_name!"

              aws_outbounds3bucketname  = "${aws_s3_bucket.microservice_s3_bucket.id}"  }}
```

Service Discovery endpoint

S3 Bucket

re:Invent

aws



AWS Fargate **Convincing the client**

Understand Docker & the benefits of containers for ease of deployment and scalability

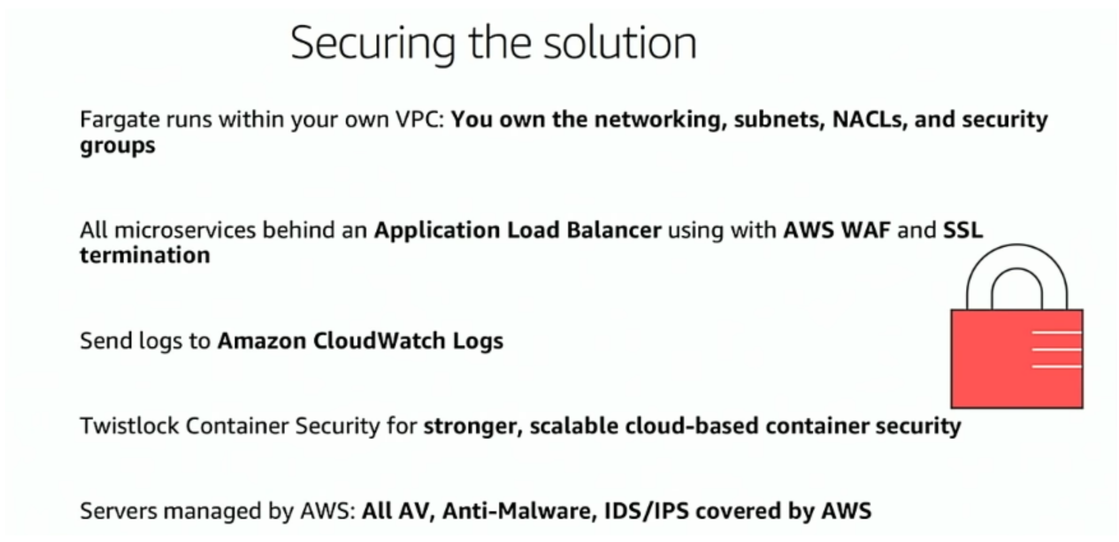
Integration with ECS and ECR

Use existing CI/CD tools: Jenkins & Bitbucket

Lower TCO with AWS Fargate

SLA of 99.99% uptime

PCI DSS Level 1, ISO 9001, ISO 27001, ISO 27017, ISO 27018, SOC 1, SOC 2, SOC 3, and HIPAA eligibility



Securing the solution

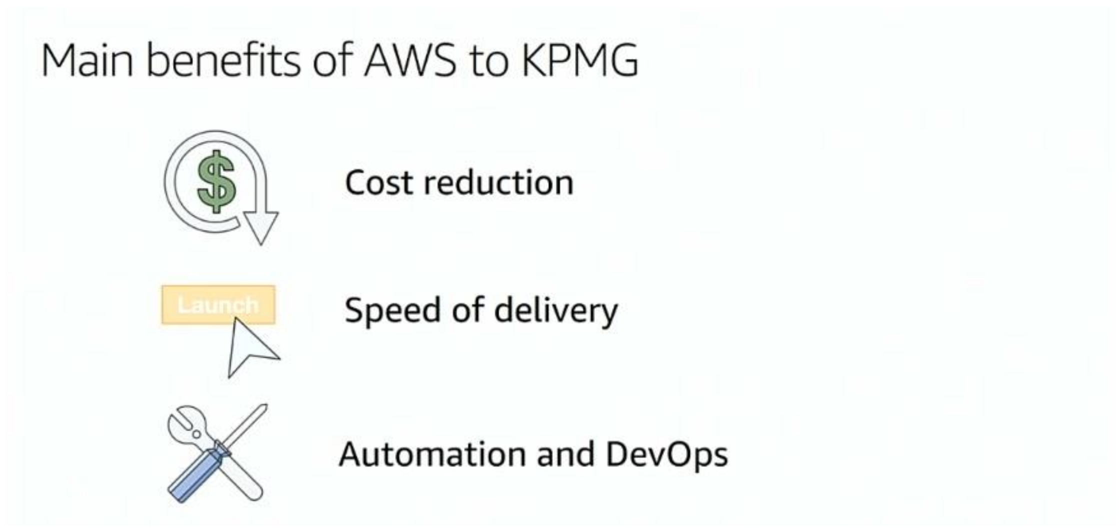
Fargate runs within your own VPC: **You own the networking, subnets, NACLs, and security groups**

All microservices behind an **Application Load Balancer** using with **AWS WAF** and **SSL termination**

Send logs to **Amazon CloudWatch Logs**

Twistlock Container Security for **stronger, scalable cloud-based container security**

Servers managed by AWS: **All AV, Anti-Malware, IDS/IPS covered by AWS**



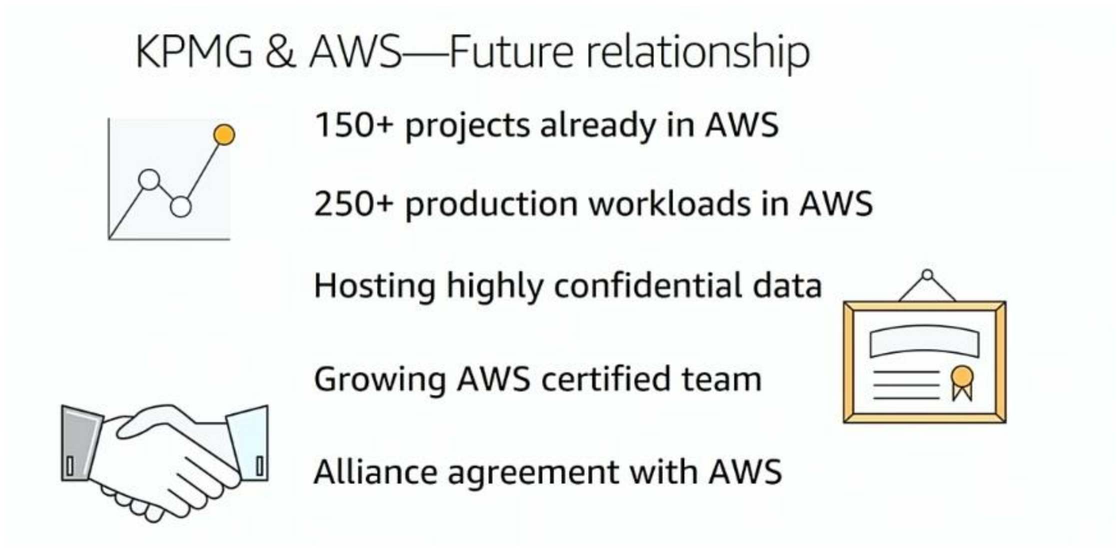
Main benefits of AWS to KPMG

Cost reduction

Launch

Speed of delivery

Automation and DevOps



KPMG & AWS—Future relationship

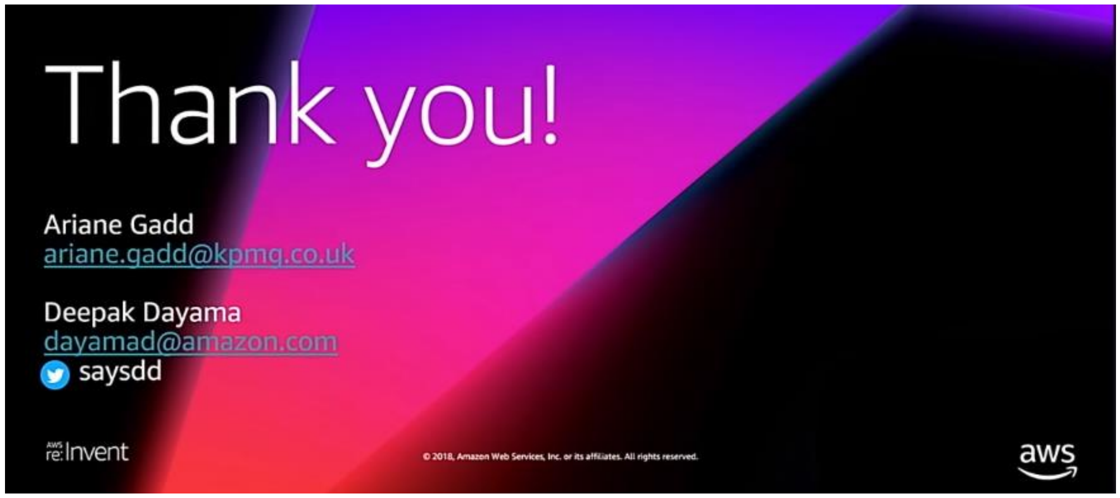
150+ projects already in AWS

250+ production workloads in AWS

Hosting highly confidential data

Growing AWS certified team

Alliance agreement with AWS



Thank you!

Ariane Gadd
ariane.gadd@kpmg.co.uk

Deepak Dayama
dayamad@amazon.com
saysdd

re:Invent

aws