Amazon ECS

Amazon Elastic Container Service

ECS **Services** are used to maintain a **desired count** of tasks

An ECS **Task** is created from a **Task Definition**

An Amazon **ECS Cluster** is a logical grouping of **tasks** or **services**

Task Definition

```
{
"containerDefinitions": [
                {
                  "name": "wordpress",
                  "links": [
                    "mysql"
                ],
                "image": "wordpress",
                "essential": true,
                "portMappings": [
                  {
                    "containerPort": 80,
                    "hostPort": 80
                     }
                  ],
                  "memory": 500,
                  "cpu": 10
                }
```

Availability Zone

Availability Zone

ECS Cluster

ECS Service

Auto Scaling group

ECS Container instance

Task

Task

ECS Container instance

Task

Task

An ECS **Task** is a running **Docker container**

Amazon Elastic Container Registry

Registry

Image Image

Docker **images** can be stored in **Amazon ECR**

DigitalCloud TRAINING

Amazon ECS Key Features

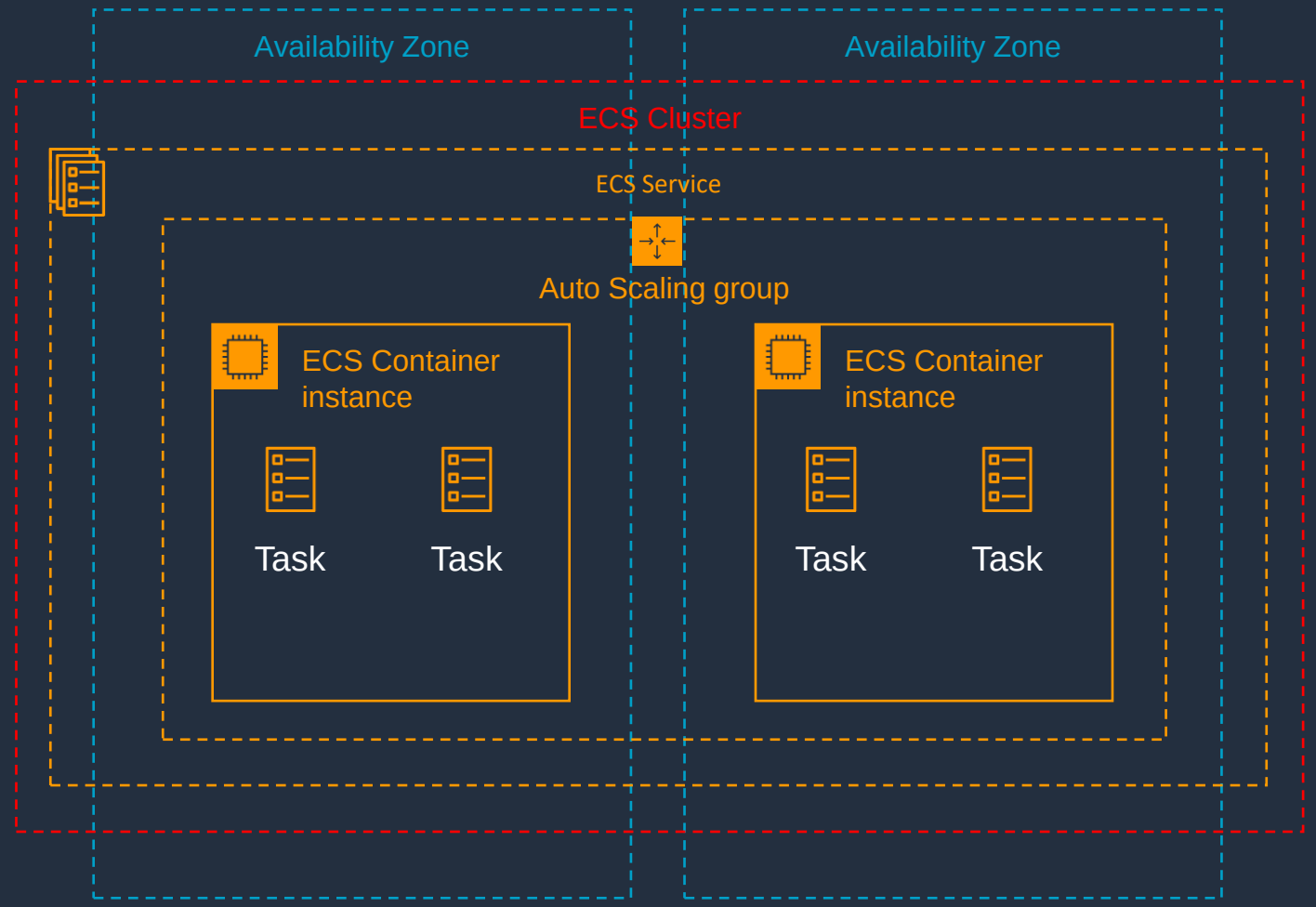
- **Serverless with AWS Fargate** – managed for you and fully scalable
- **Fully managed container orchestration** – control plane is managed for you
- **Docker support** – run and manage Docker containers with integration into the Docker Compose CLI
- **Windows container support** – ECS supports management of Windows containers
- **Elastic Load Balancing integration** – distribute traffic across containers using ALB or NLB
- **Amazon ECS Anywhere (NEW)** – enables the use of Amazon ECS control plane to manage on-premises implementations

Amazon ECS Components

Elastic Container Service (ECS)	Description
Cluster	Logical grouping of tasks or services
Container instance	EC2 instance running the the ECS agent
Task Definition	Blueprint that describes how a docker container should launch
Task	A running container using settings in a Task Definition
Service	Defines long running tasks – can control task count with Auto Scaling and attach an ELB

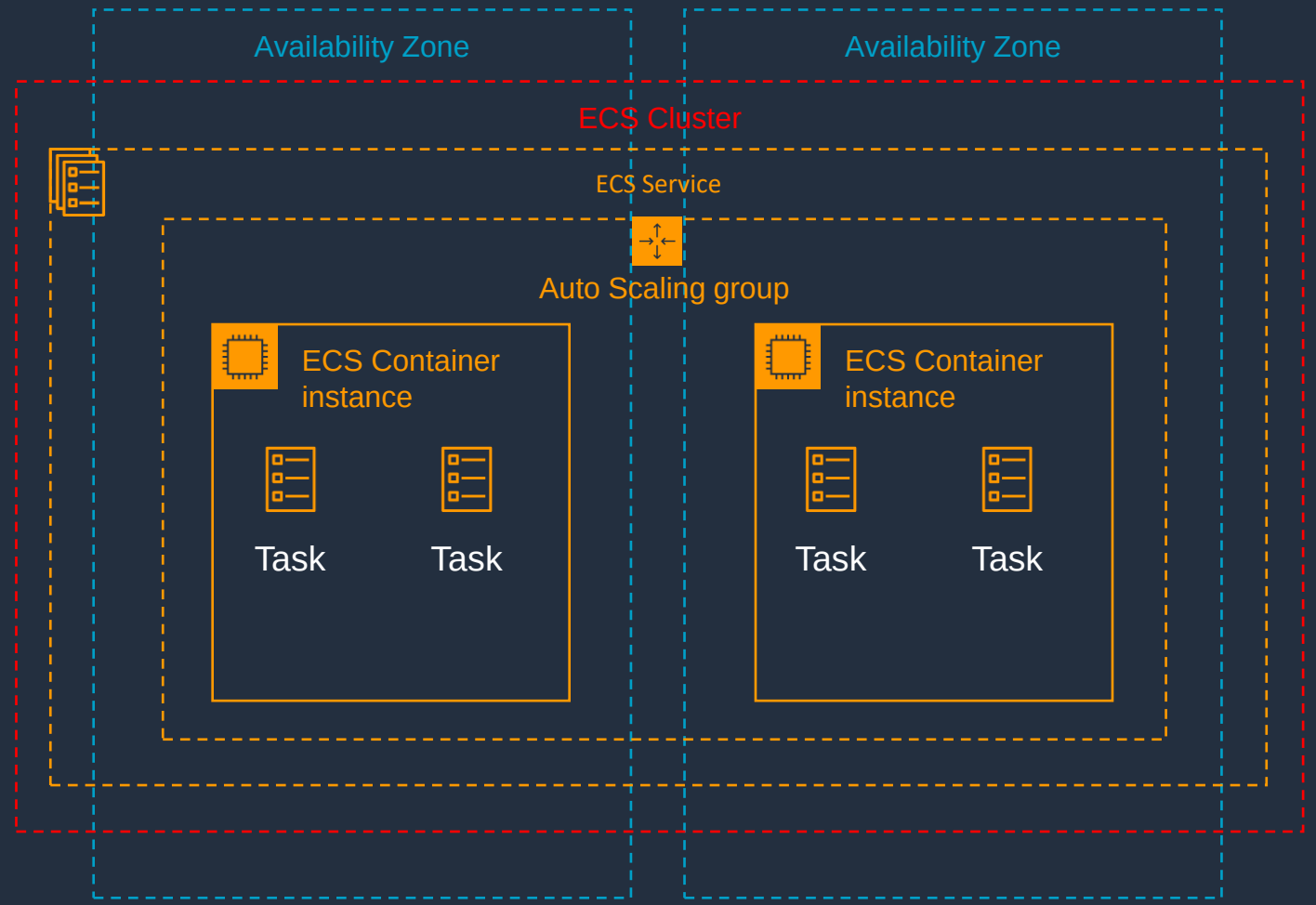
Amazon ECS Clusters

- ECS Clusters are a logical grouping of container instances that you can place tasks on
- A default cluster is created but you can then create multiple clusters to separate resources
- ECS allows the definition of a specified number (desired count) of tasks to run in the cluster



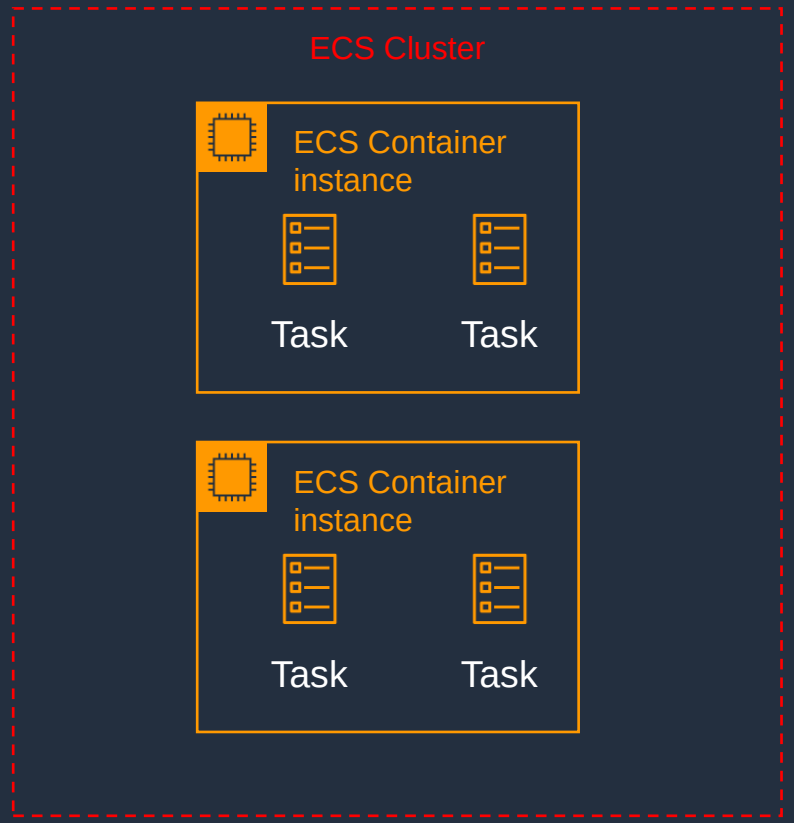
Amazon ECS Clusters

- Clusters can contain tasks using the Fargate and EC2 launch type
- EC2 launch type clusters can contain different container instance types
- Each container instance may only be part of one cluster at a time
- Clusters are region specific
- You can create IAM policies for your clusters to allow or restrict users' access to specific clusters



ECS Container Instances and Container Agent

- You can use any AMI that meets the Amazon ECS AMI specification
- The EC2 instances used as container hosts must run an ECS agent
- The ECS container agent allows container instances to connect to the cluster
- The container agent runs on each infrastructure resource on an ECS cluster

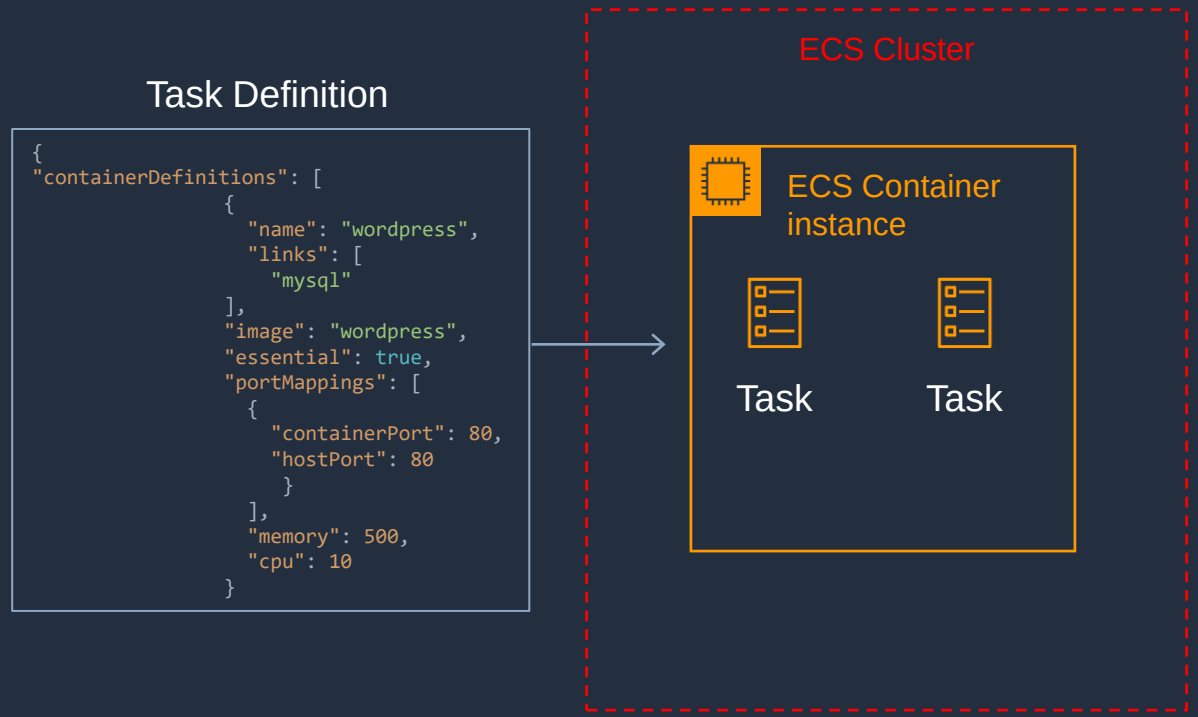


Amazon ECS Images

- Containers are created from a read-only template called an **image** which has the instructions for creating a Docker container
- Images are built from a **Dockerfile**
- Only Docker containers are supported on ECS
- Images are stored in a registry such as DockerHub or Amazon Elastic Container Registry (ECR)
- ECR is a managed AWS Docker registry service that is secure, scalable and reliable
- ECR supports private Docker repositories with resource-based permissions using AWS IAM in order to access repositories and images
- You can use the Docker CLI to push, pull and manage images

Amazon ECS Tasks and Task Definitions

- A task definition is required to run Docker containers in Amazon ECS
- A task definition is a text file in JSON format that describes one or more containers, up to a maximum of 10
- Task definitions use Docker images to launch containers
- You specify the number of tasks to run (i.e. the number of containers)



Amazon ECS Tasks and Task Definitions

Some of the parameters you can specify in a task definition include:

- Which Docker images to use with the containers in your task
- How much CPU and memory to use with each container
- Whether containers are linked together in a task
- The Docker networking mode to use for the containers in your task
- What (if any) ports from the container are mapped to the host container instances
- Whether the task should continue if the container finished or fails
- The commands the container should run when it is started
- Environment variables that should be passed to the container when it starts
- Data volumes that should be used with the containers in the task
- IAM role the task should use for permissions

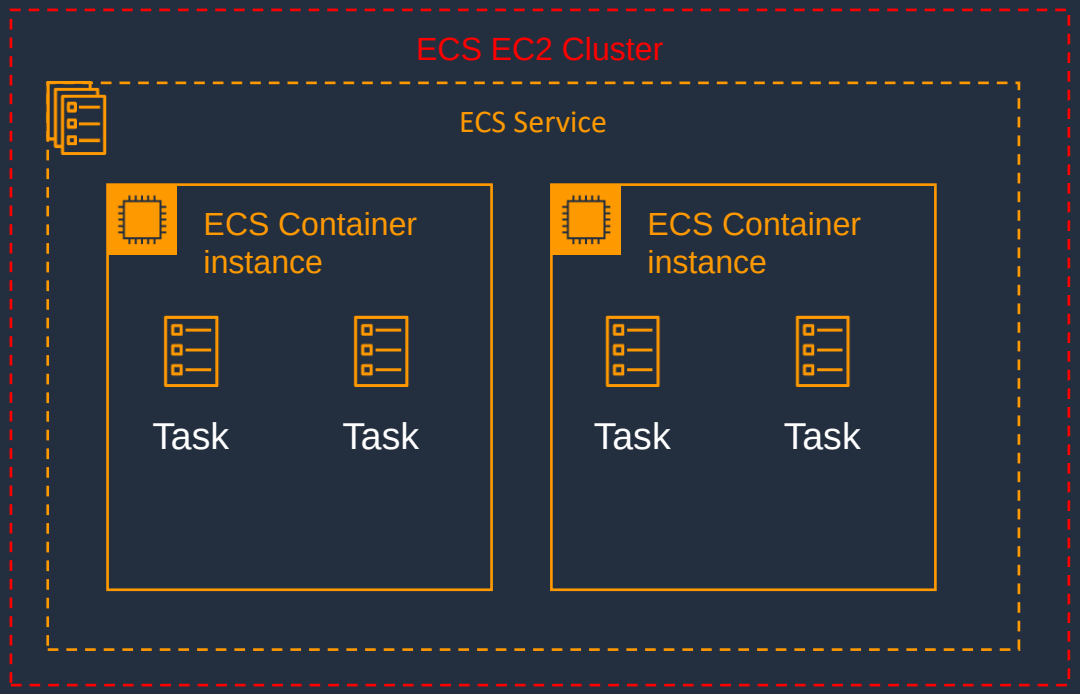
Amazon ECS Launch Types

DigitalCloud TRAINING

Launch Types – EC2 and Fargate

Registry:
ECR, Docker Hub, Self-hosted

Registry:
ECR, Docker Hub



EC2 Launch Type

- You explicitly provision EC2 instances
- You're responsible for managing EC2 instances
- Charged per running EC2 instance
- EFS and EBS integration
- You handle cluster optimization
- More granular control over infrastructure

Fargate Launch Type

- Fargate automatically provisions resources
- Fargate provisions and manages compute
- Charged for running tasks
- No EBS integration
- Fargate handles cluster optimization
- Limited control, infrastructure is automated

Amazon ECS Launch Types

Fargate Launch Type

- Run containers without the need to provision and manage the backend infrastructure
- AWS Fargate is the serverless way to host your Amazon ECS workloads

EC2 Launch Type

- Run containers on a cluster of Amazon EC2 instances that you manage

External Launch Type

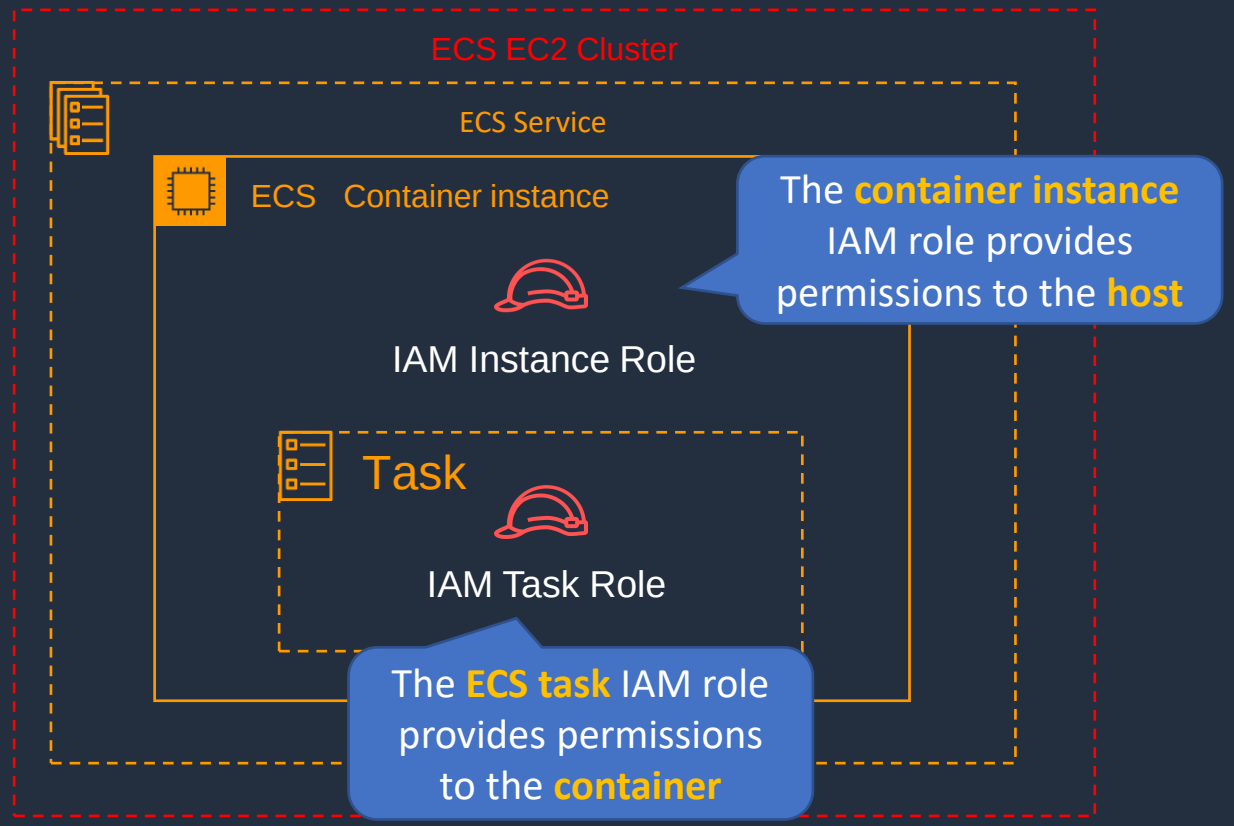
- Run containers on your on-premises servers or virtual machines (VMs) – uses Amazon ECS Anywhere

Amazon ECS Launch Types

Amazon EC2	Amazon Fargate
You explicitly provision EC2 instances	The control plane asks for resources and Fargate automatically provisions
You're responsible for upgrading, patching, care of EC2 pool	Fargate provisions compute as needed
You must handle cluster optimization	Fargate handles cluster optimization
More granular control over infrastructure	Limited control, as infrastructure is automated

Amazon ECS and IAM Roles

ECS and IAM Roles

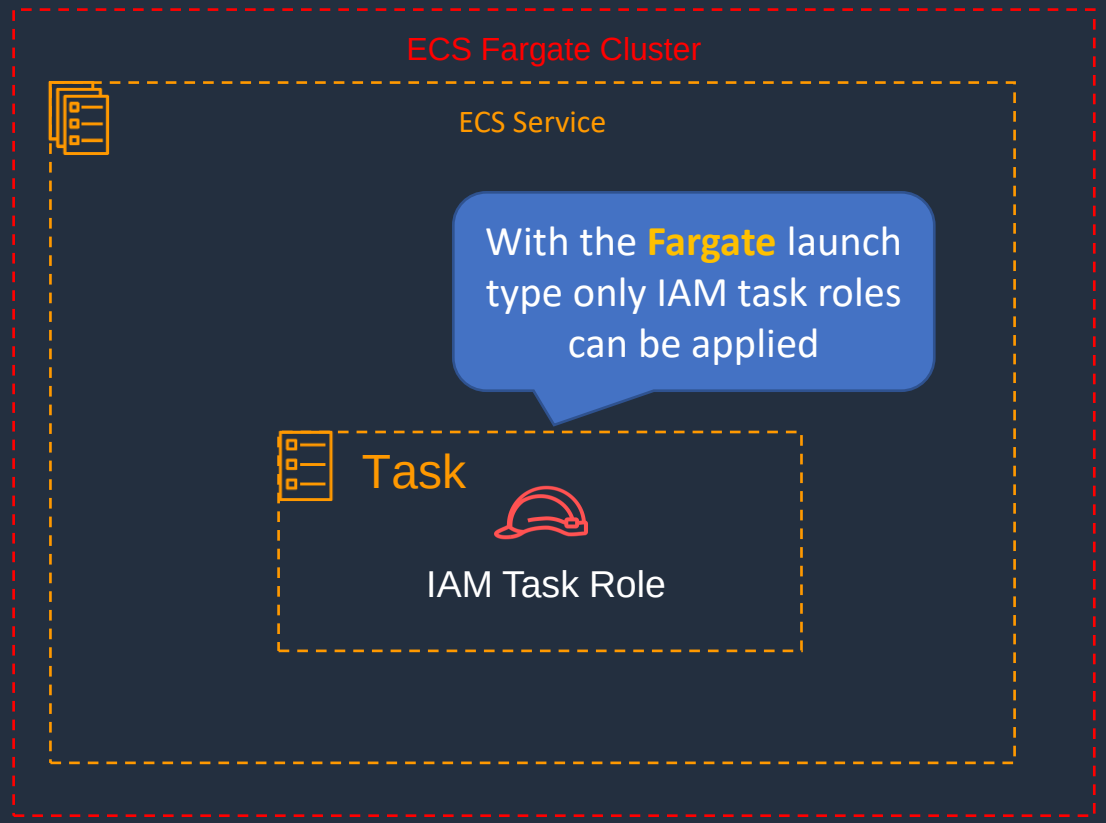


AmazonEC2ContainerServiceforEC2Role

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeTags",
                "ecs:CreateCluster",
                "ecs:DeregisterContainerInstance",
                "ecs:DiscoverPollEndpoint",
                "ecs:Poll",
                "ecs:RegisterContainerInstance",
                "ecs:StartTelemetrySession",
                "ecs:UpdateContainerInstancesState",
                "ecs:Submit*",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```

NOTE: container instances have access to **all of the permissions** that are supplied to the **container instance role** through instance metadata

ECS and IAM Roles



Create an ECS Cluster with EC2 Launch Type

DigitalCloud TRAINING

Launch Task and Service with ALB

DigitalCloud TRAINING

ECS Task Placement Strategies

ECS Task Placement Strategies

- A task placement strategy is an algorithm for selecting instances for task placement or tasks for termination
- Task placement strategies can be specified when either running a task or creating a new service
- This is relevant only to the EC2 launch type
- Amazon ECS supports the following task placement strategies:

binpack - place tasks based on the least available amount of CPU or memory. This minimizes the number of instances in use

random - place tasks randomly

ECS Task Placement Strategies

spread - place tasks evenly based on the specified value

- Accepted values are instanceId or host (same effect)
- Or any platform or custom attribute that is applied to a container instance, such as attribute:ecs.availability-zone
- Service tasks are spread based on the tasks from that service
- Standalone tasks are spread based on the tasks from the same task group

ECS Task Placement Strategies

The following strategy distributes tasks evenly across **Availability Zones**:

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    }
]
```

ECS Task Placement Strategies

The following strategy distributes tasks evenly across **all instances**:

```
"placementStrategy": [
    {
        "field": "instanceId",
        "type": "spread"
    }
]
```

ECS Task Placement Strategies

The following strategy **bin packs** tasks based on **memory**:

```
"placementStrategy": [
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

ECS Task Placement Strategies

The following strategy distributes tasks evenly across **Availability Zones** and then **bin packs** tasks based on **memory** within each **Availability Zone**:

```
"placementStrategy": [
    {
        "field": "attribute:ecs.availability-zone",
        "type": "spread"
    },
    {
        "field": "memory",
        "type": "binpack"
    }
]
```

ECS Task Placement Strategies

- A task placement constraint is a rule that is considered during task placement
- Amazon ECS supports the following types of task placement constraints:
  - distinctInstance - Place each task on a different container instance
  - memberOf - Place tasks on container instances that satisfy an expression

Cluster Query Language

- Cluster queries are expressions that enable you to group objects
- For example, you can group container instances by attributes such as **Availability Zone**, **instance type**, or **custom metadata**
- Expressions have the following syntax: subject operator [argument]
- **Example 1:** The following expression selects instances with the specified instance type:

```
attribute:ecs.instance-type == t2.small
```

- **Example 2:** The following expression selects instances in the us-east-1a or us-east-1b Availability Zone:

```
attribute:ecs.availability-zone in [us-east-1a, us-east-1b]
```

Cluster Query Language

- **Example 3:** The following expression selects instances that are hosting tasks in the service:production group:

```
task:group == service:production
```

Scaling Amazon ECS

Auto Scaling for ECS

Two types of scaling:

1. Service auto scaling
2. Cluster auto scaling

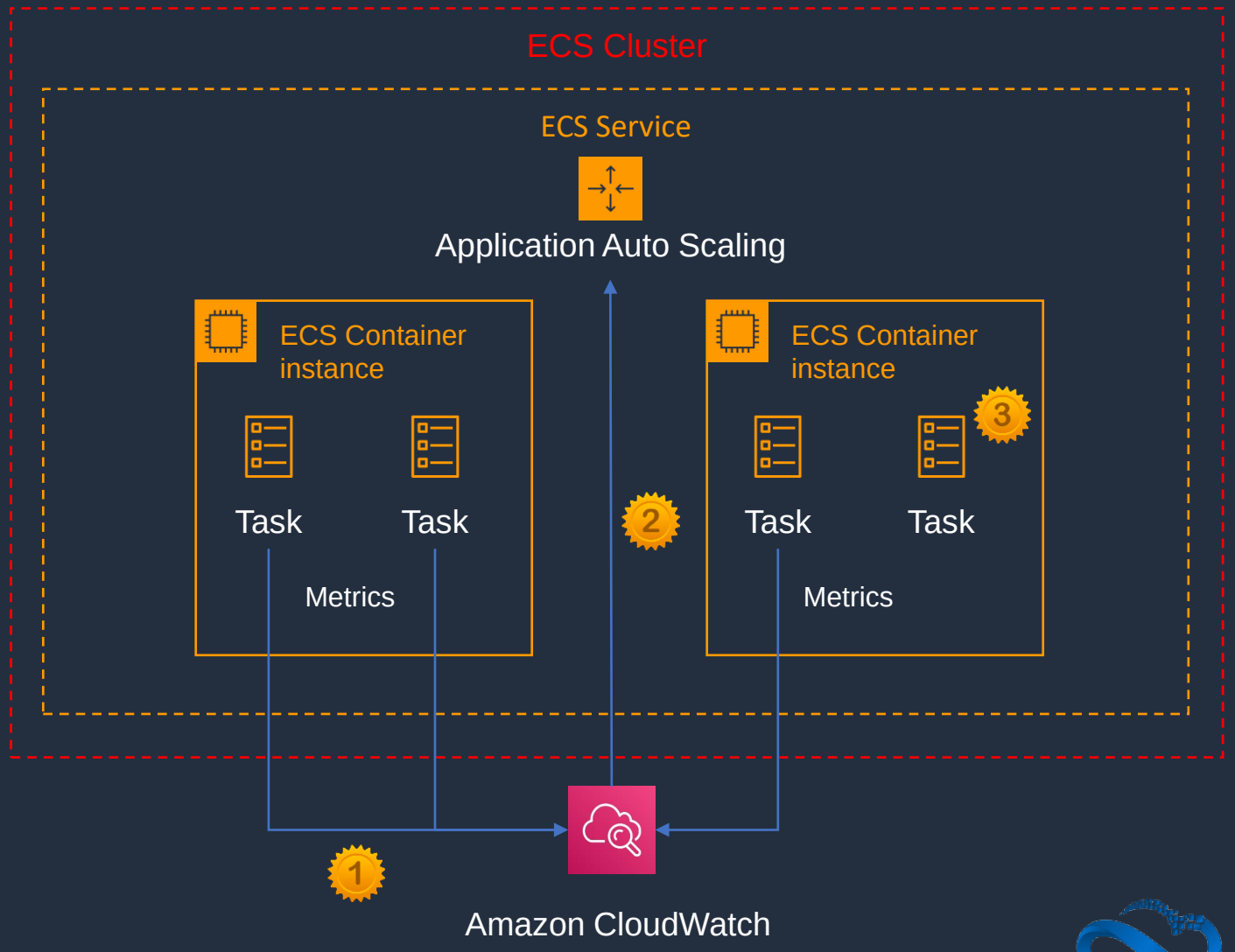
- **Service auto scaling** automatically adjusts the desired task count up or down using the Application Auto Scaling service

- **Service auto scaling** supports target tracking, step, and scheduled scaling policies

- **Cluster auto scaling** uses a Capacity Provider to scale the number of EC2 cluster instances using EC2 Auto Scaling

Service Auto Scaling

- 1. Metric reports CPU > 80%
- 2. CloudWatch notifies Application Auto Scaling
- 3. ECS launches additional task

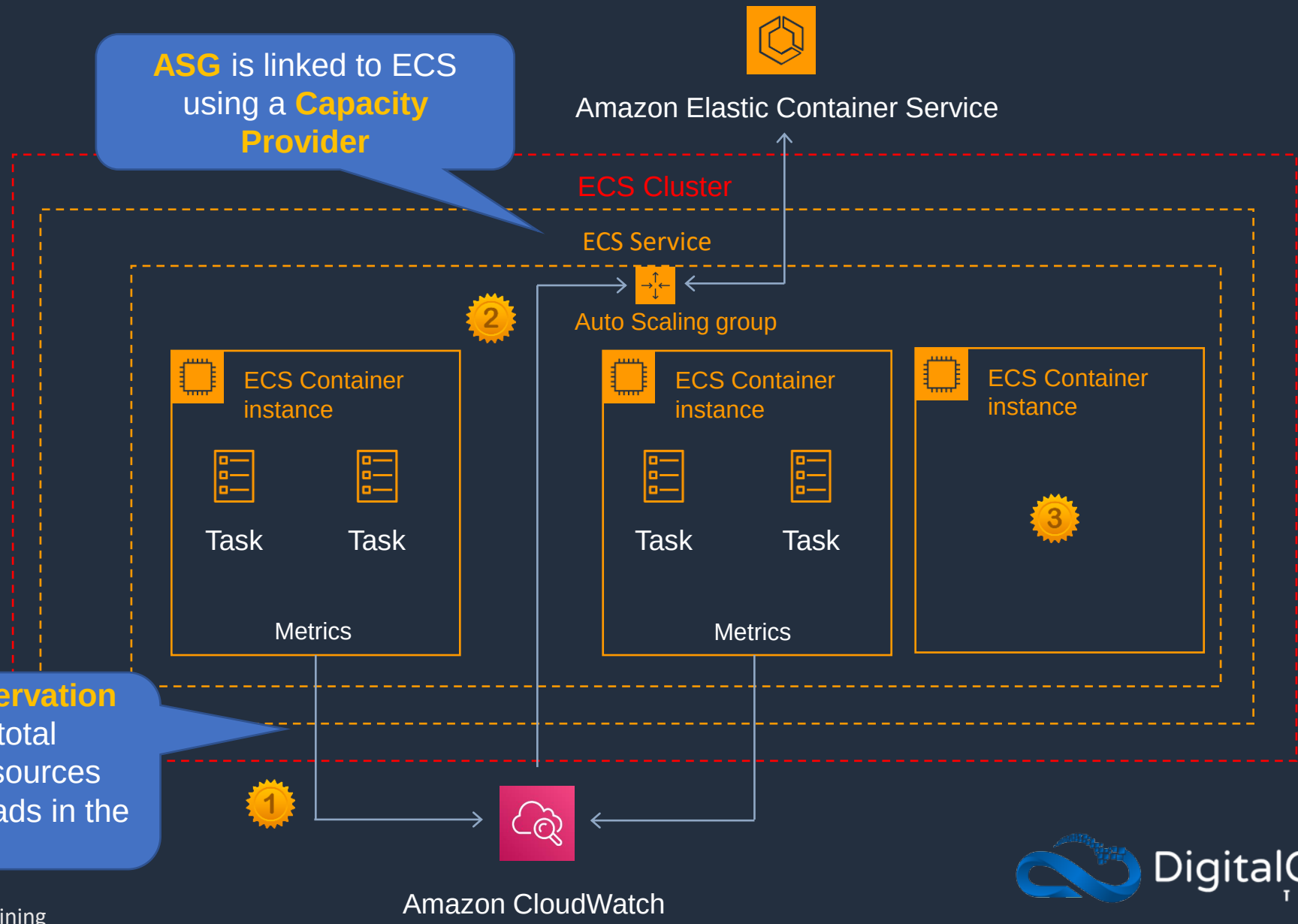


Service Auto Scaling

- Amazon ECS Service Auto Scaling supports the following types of scaling policies:
  - **Target Tracking Scaling Policies**—Increase or decrease the number of tasks that your service runs based on a target value for a specific CloudWatch metric
  - **Step Scaling Policies**—Increase or decrease the number of tasks that your service runs in response to CloudWatch alarms. Step scaling is based on a set of scaling adjustments, known as step adjustments, which vary based on the size of the alarm breach
  - **Scheduled Scaling**—Increase or decrease the number of tasks that your service runs based on the date and time

Cluster Auto Scaling

- 1. Metric reports target capacity > 80%
- 2. CloudWatch notifies ASG
- 3. AWS launches additional container instance

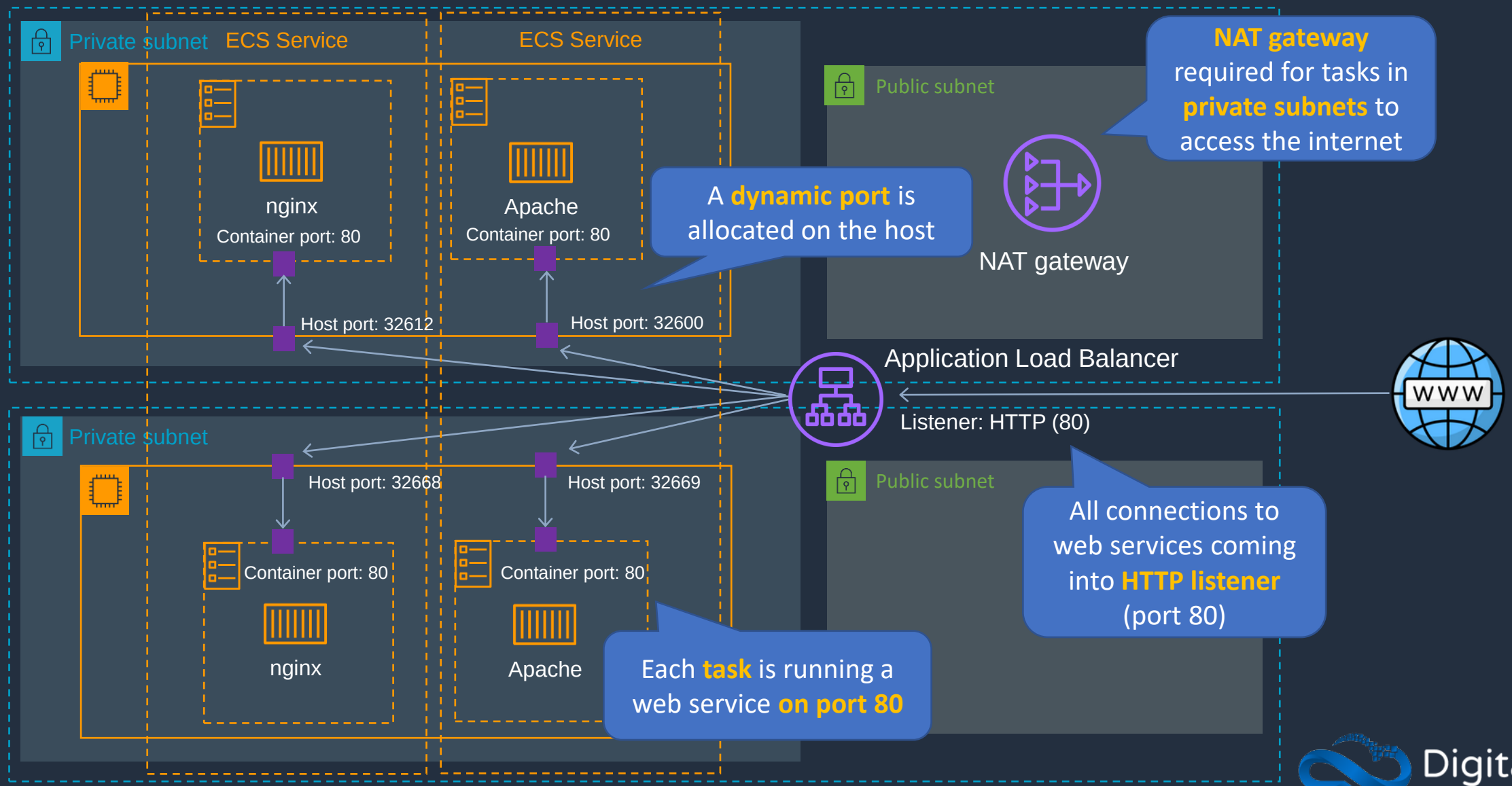


Cluster Auto Scaling

- Uses an ECS resource type called a **Capacity Provider**
- A Capacity Provider can be associated with an EC2 **Auto Scaling Group** (ASG)
- ASG can automatically scale using:
  - **Managed scaling** - with an automatically-created scaling policy on your ASG
  - **Managed instance termination protection** - which enables container-aware termination of instances in the ASG when scale-in happens

Amazon ECS with ALB

Amazon ECS with ALB



Amazon Elastic Container Registry (ECR)

Amazon Elastic Container Registry (ECR)

- Amazon ECR is a fully-managed container registry
- Integrated with Amazon ECS and Amazon EKS
- Supports Open Container Initiative (OCI) and Docker Registry HTTP API V2 standards
- You can use Docker tools and Docker CLI commands such as push, pull, list, and tag
- Can be accessed from any Docker environment – in the cloud, on-premises, or on you machine

Amazon Elastic Container Registry (ECR)

- Container images and artifacts are stored in S3
- You can use namespaces to organize repositories
- Public repositories allow everyone to access container images
- Access control applies to private repositories:
  - **IAM access control** - Set policies to define access to container images in private repositories
  - **Resource-based policies** - Access control down to the individual API action such as create, list, describe, delete, and get

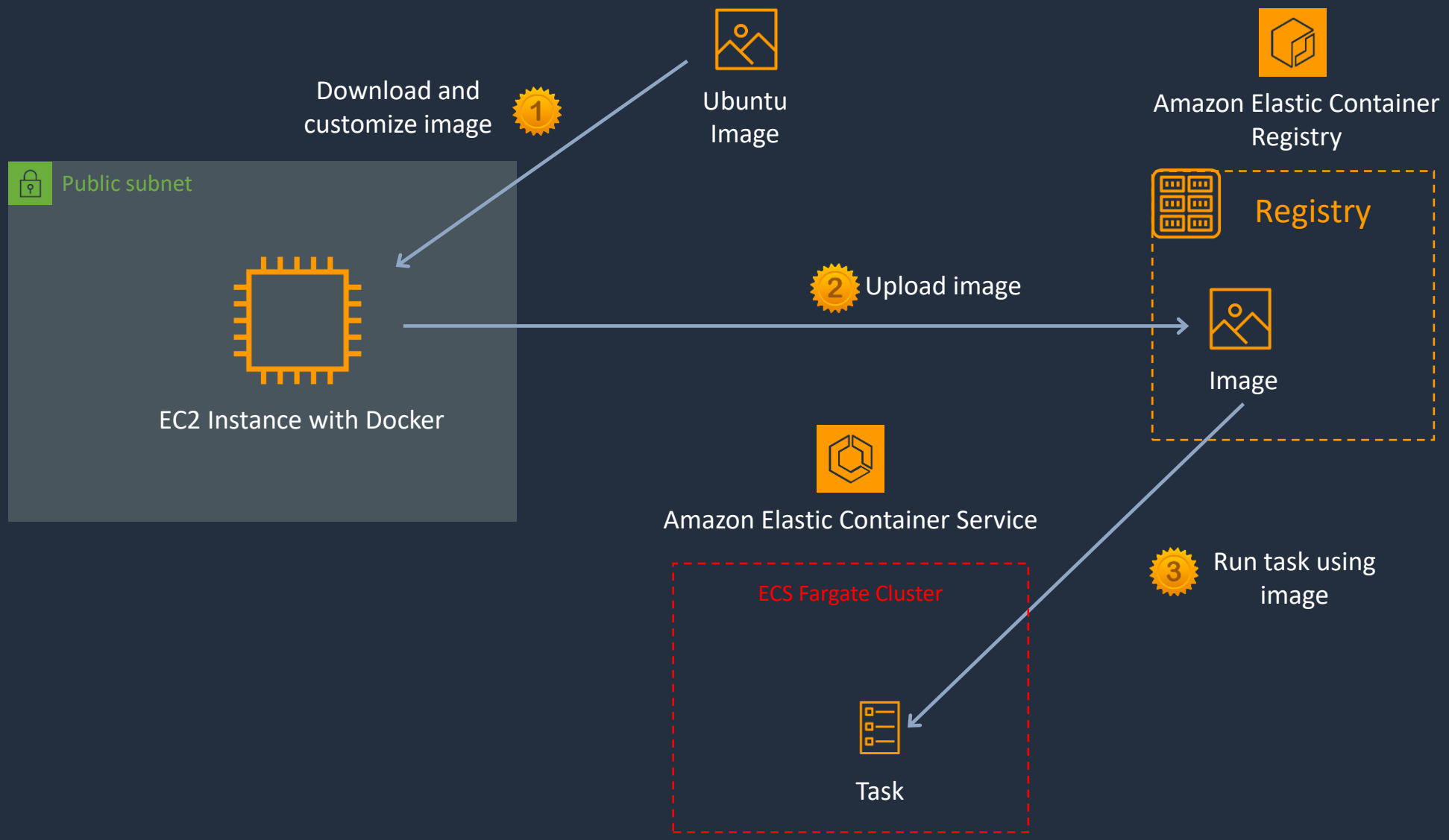
Amazon ECR Components

ECR Component	Description
Registry	An Amazon ECR private registry is provided to each AWS account; you can create one or more repositories in your registry and store images in them
Authorization token	Your client must authenticate to Amazon ECR registries as an AWS user before it can push and pull images
Repository	An Amazon ECR repository contains your Docker images, OCI images, and OCI compatible artifacts
Repository policy	You can control access to your repositories and the images within them with repository policies
Image	You can push and pull container images to your repositories

Amazon Elastic Container Registry (ECR)

- **Lifecycle policies** - manage the lifecycle of the images in your repositories
- **Image scanning** - identify software vulnerabilities in your container images
- **Cross-Region and cross-account replication** – replicate images across accounts/Region
- **Pull through cache rules** - cache repositories in remote public registries in your private Amazon ECR registry

Amazon Elastic Container Registry (ECR)



Pushing an Image to a Private Repository

- Users must have the following IAM permissions:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:CompleteLayerUpload",
                "ecr:GetAuthorizationToken",
                "ecr:UploadLayerPart",
                "ecr:InitiateLayerUpload",
                "ecr:BatchCheckLayerAvailability",
                "ecr:PutImage"
            ],
            "Resource": "*"
        }
    ]
}
```

The **Resource** element can also be used to scope to a specific repository ARN

Pushing an Image to a Private Repository

- First, authenticate the Docker client to ECR:

```
aws ecr get-login-password --region region | docker login --username AWS --password-stdin
aws_account_id.dkr.ecr.region.amazonaws.com
```

- Tag your image with the Amazon ECR registry, repository, and image tag name to use:

```
docker tag e9ae3c220b23 aws_account_id.dkr.ecr.region.amazonaws.com/my-repository:tag
```

- Push the image using the docker push command:

```
docker push aws_account_id.dkr.ecr.region.amazonaws.com/my-repository:tag
```

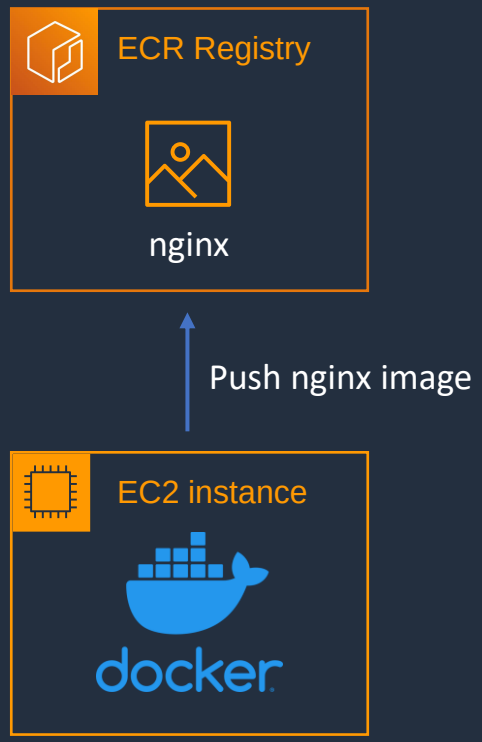
ECS-LAB-1 - Create Image and Push to ECR Repository

DigitalCloud TRAINING

ECS Lab Part 1

1. **Create Image and Push to ECR Repository**
2. Create Task Definition and ALB
3. Create Fargate Cluster and Service
4. CodeDeploy Application and Pipeline
5. Implement Blue/Green Update to ECS

ECS Lab Part 1

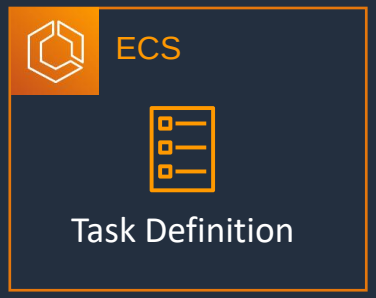


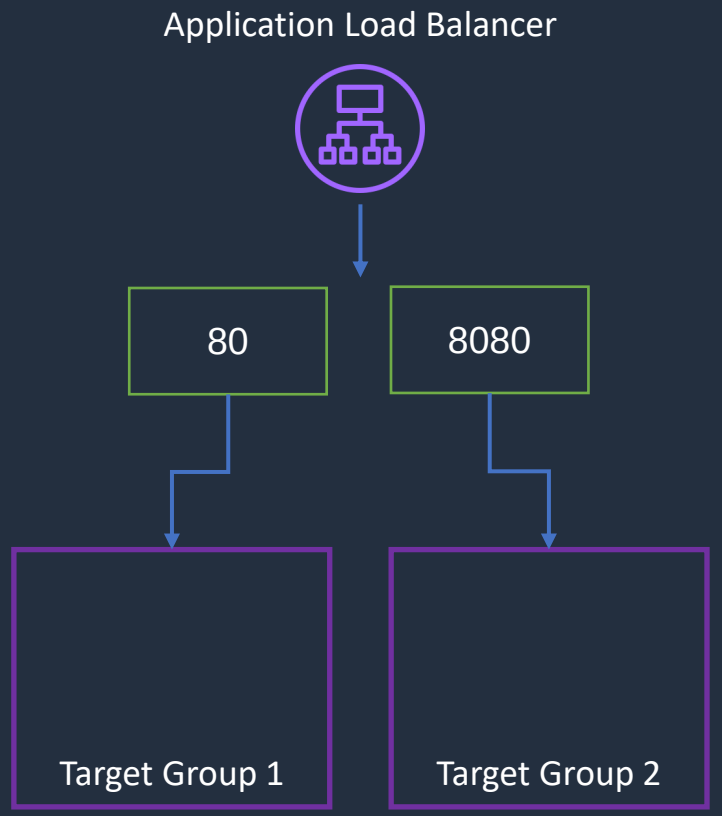
ECS-LAB-2 - Create Task Definition and ALB

ECS Lab Part 2

1. Create Image and Push to ECR Repository
2. **Create Task Definition and ALB**
3. Create Fargate Cluster and Service
4. CodeDeploy Application and Pipeline
5. Implement Blue/Green Update to ECS

ECS Lab Part 2





ECS-LAB-3 - Create Fargate Cluster and Service

ECS Lab Part 3

1. Create Image and Push to ECR Repository
2. Create Task Definition and ALB
3. **Create Fargate Cluster and Service**
4. CodeDeploy Application and Pipeline
5. Implement Blue/Green Update to ECS

ECS Lab Part 3