



### 19. What is Amazon ECS and how does it work?

**Amazon ECS (Elastic Container Service)** is a fully managed container orchestration service by AWS. It allows you to **run and manage Docker containers on a cluster of virtual machines without setting up your own container infrastructure.**

**How it works:**

- You define a **Task Definition** (like a blueprint for your containers).
- You deploy it to an **ECS Cluster**.
- ECS schedules the containers to run either on EC2 instances (EC2 launch type) or on AWS-managed compute (Fargate launch type).

---

### 20. What are the main components of ECS architecture?

The key components of ECS are:

Component	Purpose
<b>Cluster</b>	Logical grouping of resources (EC2 or Fargate) to run containers
<b>Task Definition</b>	Blueprint that defines which containers to run, resource needs, ports, etc.
<b>Task</b>	Running instance of a Task Definition
<b>Service</b>	Manages long-running tasks and ensures the desired count is maintained
<b>Container</b>	Docker container that runs your app
<b>Launch Type</b>	Defines if compute is managed by you (EC2) or AWS (Fargate)

---

### 21. What is the difference between ECS clusters, services, and tasks?

**Component Description**

<b>Cluster</b>	A group of infrastructure (either EC2 or Fargate) to run your tasks
<b>Task</b>	A single running copy of your container(s), based on a Task Definition
<b>Service</b>	A way to run and maintain a specified number of tasks simultaneously. It replaces failed tasks to ensure high availability

---

### 22. What are Task Definitions and what do they contain?

A **Task Definition** is like a Docker Compose file in ECS. It defines how your containers should run.

**It includes:**

- Docker image name

- CPU and memory requirements
- Port mappings
- Environment variables
- Volume mounts
- Networking mode
- IAM role (if needed)
- Logging configuration (like CloudWatch)

### 23. What are the two launch types in ECS?

ECS supports two main **launch types**:

#### Launch Type Managed By Infrastructure

<b>EC2</b>	You	Your EC2 instances
<b>Fargate</b>	AWS	Fully managed by AWS (no EC2 required)

### 24. What is the difference between EC2 launch type and Fargate launch type?

Feature	EC2 Launch Type	Fargate Launch Type
<b>Compute managed by</b>	You (self-managed EC2 instances)	AWS (serverless, no EC2 needed)
<b>Provisioning</b>	Manual provisioning and scaling of EC2	Auto-scaling by AWS
<b>Billing</b>	Pay for EC2 instance uptime	Pay per task (vCPU and memory used)
<b>Use case</b>	More control, custom AMIs, spot instances	Simpler setup, quick to deploy, no infra mgmt
<b>Scaling</b>	Needs custom Auto Scaling setup	Handled by AWS automatically

**Summary:** Use **Fargate** for simplicity and faster deployments. Use **EC2** for cost optimization and custom configurations.

### ECS EC2 DEMO (Using Your Own Flask Image)

#### 1. Build & Push Docker Image to Amazon ECR

##### 1.1 Create a basic Flask app & Dockerfile

```
bash
```

```
-----
```

```
mkdir ecs-demo && cd ecs-demo
```

```
# Create app.py
cat <<EOF > app.py
from flask import Flask
app = Flask(__name__)
@app.route('/')

```

```
def hello():
    return "Hello from ECS EC2!"
EOF
```

```
# Create Dockerfile
cat <<EOF > Dockerfile
FROM python:3.9
WORKDIR /app
COPY app.py .
RUN pip install flask
CMD ["python", "app.py"]
EOF
```

## 1.2 Build and Push to ECR

```
bash
```

```
-----
```

```
# Set region and repo name
AWS_REGION=us-east-1
REPO_NAME=ecs-demo
```

```
# Create ECR repo
```

```
aws ecr create-repository --repository-name $REPO_NAME --region $AWS_REGION
```

```
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-2:906253564515:repository/ecs-demo",
    "registryId": "906253564515",
    "repositoryName": "ecs-demo",
    "repositoryUri": "906253564515.dkr.ecr.us-east-2.amazonaws.com/ecs-demo",
    "createdAt": 1753695663.935,
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": false
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

```
# Get AWS Account ID
```

```
ACCOUNT_ID=$(aws sts get-caller-identity --query Account --output text)
```

```
# Login to ECR
```

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

```
# Build Docker image
```

```
docker build -t $REPO_NAME .
```

```
# Tag and push
```

```
docker tag ecs-demo:latest
$ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/ecs-demo:latest
```

```
docker push $ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/ecs-demo:latest
```

✅ Your image is now in ECR.

---

## 2. Create ECS Cluster with EC2 Launch Type

**From AWS Console:**

- Go to **Amazon ECS > Clusters > Create Cluster**
- Cluster name: ecs-demo-cluster
- EC2 Instance type: t2.micro
- Key pair: ecs-key (create if needed)
- Number of instances: 1
- Use default VPC and subnets
- Click **Create**

---

## 3. Register ECS Task Definition

```
{
  "family": "ecs-demo-task",
  "networkMode": "bridge",
  "containerDefinitions": [
    {
      "name": "flask-app",
      "image": "906253564515.dkr.ecr.us-east-2.amazonaws.com/ecs-demo",
      "cpu": 256,
      "memory": 512,
      "essential": true,
      "portMappings": [
        {
          "containerPort": 5000,
          "hostPort": 5000
        }
      ]
    }
  ],
  "requiresCompatibilities": ["EC2"]
}
```

**% aws ecs register-task-definition --cli-input-json file://ecs-task-def.json**

```
{
  "taskDefinition": {
    "taskDefinitionArn": "arn:aws:ecs:us-east-2:906253564515:task-definition/ecs-demo-task:1",
    "containerDefinitions": [
      {
        "name": "flask-app",
        "image": "906253564515.dkr.ecr.us-east-2.amazonaws.com/ecs-demo",
        "cpu": 256,
        "memory": 512,
```

```

    "portMappings": [
      {
        "containerPort": 5000,
        "hostPort": 5000,
        "protocol": "tcp"
      }
    ],
    "essential": true,
    "environment": [],
    "mountPoints": [],
    "volumesFrom": [],
    "systemControls": []
  }
],
"family": "ecs-demo-task",
"networkMode": "bridge",
"revision": 1,
"volumes": [],
"status": "ACTIVE",
"requiresAttributes": [
  {
    "name": "com.amazonaws.ecs.capability.ecr-auth"
  }
],
"placementConstraints": [],
"compatibilities": [
  "EXTERNAL",
  "EC2"
],
"requiresCompatibilities": [
  "EC2"
],
"registeredAt": 1753696869.895,
"registeredBy": "arn:aws:iam::906253564515:user/DevopsChallengeHub"
}
}

```

---

## 4. Run the Task in ECS Cluster

### From AWS Console:

- Go to **ECS > Clusters > ecs-demo-cluster > Tasks > Run new Task**
- Launch type: EC2
- Task definition: ecs-demo-task
- Cluster: ecs-demo-cluster
- Number of tasks: 1
- Click **Run Task**

 Wait until the task is **RUNNING**

---

## 5. Access the Flask App via EC2 Public IP

### Steps:

1. Go to **EC2 > Instances**
2. Find the instance running the task (check cluster name)
3. Click on the instance, copy the **Public IPv4 address**
4. Find **Port** where container is mapped:
  - Go to **ECS > Cluster > ecs-demo-cluster > Tasks**
  - Click on the task → click container → see **Port Mappings**

Now open in browser:

php-template

-----

http://<EC2\_PUBLIC\_IP>:<HOST\_PORT>

✓ You'll see: **Hello from ECS EC2!**

---

#### **Optional Cleanup**

bash

-----

# To delete ECS and EC2 resources

aws ecs delete-cluster --cluster ecs-demo-cluster

aws ecr delete-repository --repository-name ecs-demo --force

---

**What is the primary purpose of Amazon ECS?**

- A) To run serverless functions like AWS Lambda
- B) To manage and orchestrate Docker containers
- C) To store container images
- D) To monitor container logs only

✓ **Correct Answer:** B) To manage and orchestrate Docker containers

---

**What is the role of a *Task Definition* in ECS?**

- A) It defines IAM permissions for ECS clusters
- B) It schedules ECS containers on EC2
- C) It acts as a blueprint for defining container configurations
- D) It stores logs generated by ECS

✓ **Correct Answer:** C) It acts as a blueprint for defining container configurations

---

**Q3. Which of the following is **not** a launch type in ECS?**

- A) EC2
- B) Fargate
- C) Lambda

✓ **Correct Answer:** C) Lambda

---

**In ECS, what is a **Service** responsible for?**

- A) Storing Docker images
- B) Ensuring that a specified number of tasks are always running
- C) Managing the security group of containers
- D) Collecting logs from containers

✓ **Correct Answer:** B) Ensuring that a specified number of tasks are always running

---

**What does a **Cluster** represent in ECS?**

- A) A group of Dockerfiles
- B) A group of running containers
- C) A logical grouping of compute resources to run containers
- D) A collection of S3 buckets for container storage

✓ **Correct Answer:** C) A logical grouping of compute resources to run containers

---

**What is a Task in ECS?**

- A) A background process for managing EC2
- B) A long-running daemon in the ECS cluster
- C) A running instance of a Task Definition
- D) A CloudFormation template for ECS resources

✓ **Correct Answer:** C) A running instance of a Task Definition

---

**Which of the following is **not** part of a Task Definition in ECS?**

- A) Docker image name
- B) EC2 instance type
- C) Port mappings
- D) Environment variables

✓ **Correct Answer:** B) EC2 instance type

---

**A Task Definition in ECS is most similar to which tool?**

- A) AWS IAM Policy
- B) Docker Compose file
- C) CloudTrail
- D) EC2 Auto Scaling Group

✓ **Correct Answer:** B) Docker Compose file

---

**Which of the following is an **advantage** of using Fargate over EC2?**

- A) Requires manual EC2 scaling
- B) Lets you use custom EC2 AMIs
- C) Serverless, no infrastructure to manage
- D) Pay for full EC2 instance uptime

✓ **Correct Answer:** C) Serverless, no infrastructure to manage

---

**When would you **prefer EC2 launch type** over Fargate?**

- A) You want fast deployments with minimal configuration
- B) You want to avoid managing any servers
- C) You need custom AMIs or use spot instances for cost optimization
- D) You want AWS to manage compute scaling

✓ **Correct Answer:** C) You need custom AMIs or use spot instances for cost optimization

---