



What are ECS services and how do they ensure high availability?

An **ECS Service** in AWS is like a **watchdog** for your containerized application. It keeps your app running **all the time**, manages **failures**, and handles **updates** smoothly.

What does an ECS Service do?

1. **Keeps tasks running:**
 - If you want 3 copies of your app always running, the ECS Service makes sure exactly 3 are running.
 - If 1 task crashes, ECS automatically replaces it.
2. **Handles traffic via Load Balancers:**
 - ECS Service can connect to an **Application Load Balancer (ALB)** or **Network Load Balancer (NLB)**.
 - It spreads incoming traffic across all healthy containers.
3. **Performs rolling updates:**
 - When updating your app, ECS replaces old tasks one by one with new ones.
 - This avoids downtime.

How ECS Ensures High Availability

ECS Services make your application resilient and fault-tolerant using:

- **Multi-AZ Deployment:**
 - Tasks are spread across different **Availability Zones**. If one zone fails, others still serve traffic.
- **Automatic Health Checks:**
 - ECS and Load Balancer both check if your containers are healthy.
 - If a task becomes unhealthy, ECS replaces it quickly.
- **Auto Scaling:**
 - ECS can add more tasks when traffic increases and remove them when it decreases—saving cost.

Example (STAR Format)

Situation:

We ran an e-commerce site using ECS on EC2.

Task:

Ensure the site is highly available across failures.

Action:

Deployed 3 tasks of the web app across 3 different AZs, connected them to an ALB, and enabled health checks.

Result:

When one AZ had an outage during a load test, ECS launched a replacement in a healthy zone, and traffic continued without downtime.

How do you handle service discovery in ECS? What is AWS Cloud Map?

What is Service Discovery?

Imagine you're at a **hotel**, and you want to order food from the restaurant, but you don't know which room the chef is in.

You call the **reception**, and they tell you:

"The chef is in room 204."

That **reception** is like **Service Discovery** — it helps different services (like the chef and the guest) **find each other**.

In ECS (Elastic Container Service), multiple containers (apps/microservices) often need to talk to each other.

For that, they need to know **where** the other containers are running.

How ECS Enables Service Discovery

There are **two ways** to do Service Discovery in ECS:

Type	How It Works	Example
DNS-based Service Discovery	ECS uses AWS Cloud Map to automatically assign a DNS name (like myapp.local) to your container	Service A can find Service B using serviceb.myapp.local
Using a Load Balancer	ECS puts all tasks behind a Load Balancer (ALB/NLB) . Other services just use the load balancer's DNS	Service A just sends requests to the ALB URL

What is AWS Cloud Map?

AWS Cloud Map is like a **phone directory for your microservices**.

It:

- Registers your services with **DNS names**.
- Keeps track of which **IP addresses or endpoints** belong to those services.
- Keeps this list **updated dynamically** as tasks come and go (ECS handles that).

✅ So, instead of hardcoding IPs, your services use names like:

orders.mycompany.local

users.mycompany.local

Cloud Map resolves those names to **current, running ECS task IPs**.

Let's say:

- You have two ECS services: frontend and backend.
- frontend needs to talk to backend.

Without Service Discovery:

- You'd need to manually find backend's IP or load balancer — not scalable.

With Cloud Map:

- Backend service is registered in Cloud Map as backend.local.
- Frontend simply calls `http://backend.local/api/data`.

ECS + Cloud Map handles everything behind the scenes:

- Tracks task IPs
- Updates DNS records
- Routes traffic correctly

When to Use Cloud Map in ECS

Use Cloud Map when:

- You're using **ECS with Fargate or EC2** and **want services to discover each other dynamically**
- You're **not using a load balancer** (or want more flexibility)
- You need to **register custom names** or **health check endpoints**

Summary

Concept	Meaning
Service Discovery	Mechanism for services to find each other automatically
Cloud Map	AWS service that manages names and locations of your services
Why it matters	No need to hardcode IPs; services scale, fail, and recover easily
How ECS uses it	ECS can register services in Cloud Map so other services can find them via DNS

AWS ECS Fargate + ALB High Availability setup

What are we building?

We're deploying a **Flask web app** (Python-based) using:

- **ECS Fargate** (serverless container hosting)
- **Application Load Balancer** (for traffic distribution and high availability)
- **ECR** (Docker image storage)
- **CloudWatch** (logging)

Prerequisites

Make sure you have:

- **AWS CLI** installed and configured

- **Docker** installed on your machine
- An **AWS account** with admin-level permissions

Setup

Step 1: Create a Simple Flask App

- This is just a small Python web application.
 - It shows a greeting message when you visit the website.
 - It also has a special “health check” page so AWS can check if it's working.
 - This app will be converted into a Docker container and deployed later.
-



Step 1: Create a Simple Flask App

Why? You need a sample app to deploy.

Create a file called app.py with a basic "Hello" message and a health check endpoint.

python

```
from flask import Flask
```

```
import socket
```

```
app = Flask(__name__)
```

```
@app.route("/")
```

```
def hello():
```

```
    return f"Hello from {socket.gethostname()} - Version 1.0"
```

```
@app.route("/health")
```

```
def health():
```

```
    return {"status": "healthy", "hostname": socket.gethostname()}
```

```
if __name__ == "__main__":
```

```
    app.run(host="0.0.0.0", port=80)
```



Step 2: Dockerize the App

Step 2: Create a Dockerfile

- A Dockerfile is like a recipe that tells Docker how to package your app.
- It says: use Python, copy the app code, install Flask, and run the app.
- Once we build the Docker image, it can run anywhere—including AWS.

Why? ECS runs Docker containers.

Create a file named Dockerfile:

dockerfile

```
FROM python:3.9-slim
```

```
WORKDIR /app
```

```
COPY app.py /app/
```

```
RUN pip install flask
EXPOSE 80
CMD ["python", "app.py"]
```

Step 3: Build and Push Docker Image to ECR

Why? ECS needs your app image from a registry (ECR in this case).

Step 3: Push Docker Image to AWS ECR

- **ECR (Elastic Container Registry)** is like a storage for your Docker images in AWS.
- First, you find your AWS account and region details.
- Then you create a place (repository) in ECR for your image.
- You **build your image**, **log in to ECR**, and **push it** there.

3.1 Get Account Info

```
bash
```

```
-----
```

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

```
AWS_REGION=$(aws configure get region)
```

3.2 Create ECR Repository

```
bash
```

```
-----
```

```
aws ecr create-repository --repository-name ecs-flask-demo
```

3.3 Build, Tag, and Push Image

```
bash
```

```
-----
```

```
docker build -t ecs-flask-demo .
```

```
aws ecr get-login-password | docker login --username AWS --password-stdin
```

```
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com
```

```
docker tag ecs-flask-demo:latest
```

```
$AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/ecs-flask-demo:latest
```

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

Step 4: Create ECS Cluster

Why? A cluster is the environment where your containers run.

Step 4: Create an ECS Cluster

- An **ECS cluster** is like a container playground where your app will run.
- Even though Fargate manages infrastructure for you, ECS still needs a cluster to organize everything.

```
bash
```

```
-----
```

```
aws ecs create-cluster --cluster-name demo-cluster
```

Step 5: Create Task Execution Role

Why? ECS needs permission to pull from ECR and write logs.

Step 5: Create ECS Task Execution Role

- This is an IAM **permission role** that ECS tasks need to:
 - Pull your app image from ECR.
 - Write logs to CloudWatch.
- You create this role and attach a pre-made AWS policy that gives the right permissions.
- You also get its **ARN** (a unique ID), which you'll need in the next steps.

5.1 Create Trust Policy JSON

bash

```
cat > trust-policy.json <<EOF
```

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": { "Service": "ecs-tasks.amazonaws.com" },
    "Action": "sts:AssumeRole"
  }]
}
EOF
```

5.2 Create Role and Attach Policy

bash

```
aws iam create-role --role-name ecsTaskExecutionRole --assume-role-policy-
document file://trust-policy.json
```

```
aws iam attach-role-policy --role-name ecsTaskExecutionRole --policy-arn
arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
```

5.3 Get Role ARN

bash

```
EXECUTION_ROLE_ARN=$(aws iam get-role --role-name ecsTaskExecutionRole --query
'Role.Arn' --output text)
```

Step 6: Get Network Details (VPC, Subnets, Security Groups)

Step 6: Get Network Setup Details

- ECS needs networking to run—just like a house needs plumbing and electricity. You do three things here:

1. Get VPC info: AWS by default creates a Virtual Private Cloud (VPC)—your isolated network.
2. Find 2 public subnets: For high availability, AWS needs to run your app in two different zones.
3. Create security groups:
 - One for the Load Balancer (allows internet traffic).
 - One for the ECS tasks (allows traffic from the Load Balancer).

6.1 Get Default VPC

bash

```
VPC_ID=$(aws ec2 describe-vpcs --filters "Name=isDefault,Values=true" --query "Vpcs[0].VpcId" --output text)
```

6.2 Get 2 Public Subnets

bash

```
SUBNET_IDS=$(aws ec2 describe-subnets --filters "Name=vpc-id,Values=$VPC_ID" "Name=map-public-ip-on-launch,Values=true" --query "Subnets[*].SubnetId" --output text)
```

6.3 Create Security Groups

For ALB:

bash

```
ALB_SG_ID=$(aws ec2 create-security-group --group-name flask-alb-sg --description "ALB SG" --vpc-id $VPC_ID --query 'GroupId' --output text)
aws ec2 authorize-security-group-ingress --group-id $ALB_SG_ID --protocol tcp --port 80 --cidr 0.0.0.0/0
```

For ECS Tasks:

bash

```
ECS_SG_ID=$(aws ec2 create-security-group --group-name flask-ecs-sg --description "ECS SG" --vpc-id $VPC_ID --query 'GroupId' --output text)
aws ec2 authorize-security-group-ingress --group-id $ECS_SG_ID --protocol tcp --port 80 --source-group $ALB_SG_ID
```

Step 7: Set Up Load Balancer (ALB)

Step 7: Set Up the Application Load Balancer (ALB)

- ALB spreads incoming web traffic across multiple tasks (copies of your app).
- It makes sure traffic only goes to healthy, running instances of your app.

You do 3 things here:

1. Create the ALB and note its DNS name (your app's public URL).
2. Create a Target Group: This tells ALB where to send the traffic (your ECS tasks).
3. Create a Listener: This listens for HTTP requests and forwards them to the Target Group.

7.1 Create ALB

bash

```
ALB_ARN=$(aws elbv2 create-load-balancer --name flask-alb --subnets $SUBNET_1 $SUBNET_2 --security-groups $ALB_SG_ID --query 'LoadBalancers[0].LoadBalancerArn' --output text)
```

```
ALB_DNS=$(aws elbv2 describe-load-balancers --load-balancer-arns $ALB_ARN --
query 'LoadBalancers[0].DNSName' --output text)
```

7.2 Create Target Group

```
bash
```

```
-----
```

```
TARGET_GROUP_ARN=$(aws elbv2 create-target-group --name flask-targets --protocol
HTTP --port 80 --vpc-id $VPC_ID --target-type ip --health-check-path /health --query
'TargetGroups[0].TargetGroupArn' --output text)
```

7.3 Add Listener to ALB

```
bash
```

```
-----
```

```
aws elbv2 create-listener --load-balancer-arn $ALB_ARN --protocol HTTP --port 80 --
default-actions Type=forward,TargetGroupArn=$TARGET_GROUP_ARN
```

Step 8: Create ECS Task Definition

Step 8: Create ECS Task Definition

- Think of a task definition as a blueprint for running your app.
- It describes:
 - Which Docker image to use.
 - What CPU and memory it needs.
 - Where logs go.
 - Which port the app runs on.
- You create this definition in a JSON file and upload it to ECS.

Also:

- You create a log group in CloudWatch to store your app logs.
- Then you register the task definition with ECS.

8.1 Use this template to define your container

Save this JSON (or use envsubst for auto-replacing variables):

```
json
```

```
-----
```

```
{
  "family": "flask-task",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["FARGATE"],
  "cpu": "256",
  "memory": "512",
  "executionRoleArn": "<EXECUTION_ROLE_ARN>",
  "containerDefinitions": [
    {
      "name": "flask-app",
      "image": "<ECR_IMAGE_URI>",
      "portMappings": [{ "containerPort": 80 }],
      "essential": true,
      "logConfiguration": {
        "logDriver": "awslogs",
```



```

    "options": {
      "awslogs-group": "/ecs/flask-task",
      "awslogs-region": "<REGION>",
      "awslogs-stream-prefix": "ecs"
    }
  }
}
]
}

```

8.2 Create CloudWatch Log Group

bash

```
aws logs create-log-group --log-group-name /ecs/flask-task
```

8.3 Register Task Definition

bash

```
aws ecs register-task-definition --cli-input-json file://ecs-task-definition.json
```

Step 9: Create ECS Service

- The service is the actual thing that runs and maintains your app.
- You tell it:
 - Use 2 copies (for high availability).
 - Use Fargate launch type (no servers to manage).
 - Use the subnets and security groups you set up.
 - Connect to the ALB and target group.
- ECS then launches your app tasks on Fargate.

bash

```

aws ecs create-service \
  --cluster demo-cluster \
  --service-name flask-service \
  --task-definition flask-task \
  --desired-count 2 \
  --launch-type FARGATE \
  --network-configuration
"awsvpcConfiguration={subnets=[$SUBNET_1,$SUBNET_2],securityGroups=[$ECS_SG_ID],assignPublicIp=ENABLED}" \
  --load-balancers "targetGroupArn=$TARGET_GROUP_ARN,containerName=flask-app,containerPort=80"

```

Step 10: Verify and Test

Step 10: Verify Your App is Working

- You check if the ECS service is running properly.
- You check if the tasks are healthy.
- You access the app using the ALB URL in your browser.

- You can also use tools like curl to test it from command line.

10.1 Check if service is running

```
bash
```

```
-----
```

```
aws ecs describe-services --cluster demo-cluster --services flask-service
```

10.2 Access the App

```
bash
```

```
-----
```

```
echo "http://$ALB_DNS"
```

```
curl http://$ALB_DNS
```

```
curl http://$ALB_DNS/health
```



Step 11: Test High Availability

Step 11: Test High Availability

- You manually stop one task (simulate a failure).
- ECS will automatically start a new one in a few seconds.
- Your app remains live and unaffected—this proves high availability.

11.1 Kill a Task

```
bash
```

```
-----
```

```
TASK_ARN=$(aws ecs list-tasks --cluster demo-cluster --service-name flask-service --  
query 'taskArns[0]' --output text)
```

```
aws ecs stop-task --cluster demo-cluster --task $TASK_ARN --reason "HA Test"
```

11.2 ECS Will Start a New Task Automatically

Check with:

```
bash
```

```
-----
```

```
aws ecs describe-services --cluster demo-cluster --services flask-service
```



Cleanup (Optional)

Remove all created resources to avoid AWS charges.

```
bash
```

```
-----
```

```
# ECS
```

```
aws ecs update-service --cluster demo-cluster --service flask-service --desired-count 0
```

```
aws ecs delete-service --cluster demo-cluster --service flask-service
```

```
# ALB & Target Group
```

```
aws elbv2 delete-load-balancer --load-balancer-arn $ALB_ARN
```

```
aws elbv2 delete-target-group --target-group-arn $TARGET_GROUP_ARN
```

```
# Security Groups
```

```
aws ec2 delete-security-group --group-id $ALB_SG_ID
```

```
aws ec2 delete-security-group --group-id $ECS_SG_ID
```






```
# ECS Cluster
aws ecs delete-cluster --cluster demo-cluster
```

```
# ECR
aws ecr delete-repository --repository-name ecs-flask-demo --force
```

```
# CloudWatch Logs
aws logs delete-log-group --log-group-name /ecs/flask-task
```



Key Benefits of This Setup

Benefit	Description
 High Availability	ECS runs multiple tasks in different subnets/availability zones
 Load Balancing	ALB distributes traffic to healthy tasks automatically
 Auto-Recovery	If one task crashes, ECS restarts another automatically
 Monitoring	Logs go to CloudWatch; health checks are handled by ALB
 Scalability	Easily scale task count from CLI or UI

What is the primary role of an ECS Service in AWS?

- A. To store container images in the cloud
- B. To manage the VPC configuration
- C. To ensure that a specified number of tasks are always running
- D. To manually restart tasks after failure



Correct Answer: C

How does an ECS Service respond when a task crashes?

- A. It waits for the user to restart the task manually
- B. It replaces the failed task automatically
- C. It deletes all other tasks
- D. It sends a termination signal to all services



Correct Answer: B

How does ECS distribute incoming traffic across containers?

- A. By using IAM Roles
- B. By storing data in S3
- C. By using a Load Balancer like ALB or NLB
- D. By launching more EC2 instances



Correct Answer: C

What is a rolling update in ECS Services?

- A. All containers are updated at once, risking downtime
- B. ECS removes all old tasks before launching new ones
- C. ECS gradually replaces old tasks with new ones to avoid downtime
- D. ECS only updates the ECS cluster and not the tasks

 **Correct Answer: C**

Which of the following contributes to high availability in ECS? (Select all that apply)

- A. Multi-AZ deployment
- B. CloudFront distribution
- C. Health checks
- D. Auto Scaling

 **Correct Answers: A, C, D**

What is the purpose of service discovery in ECS?

- A. To find the size of Docker images
- B. To help containers find and communicate with each other
- C. To assign IAM roles to containers
- D. To backup container logs

 **Correct Answer: B**

What is AWS Cloud Map?

- A. A visual map of your cloud infrastructure
- B. A VPC subnet planning tool
- C. A service that maps service names to IPs dynamically
- D. A monitoring dashboard

 **Correct Answer: C**

How does DNS-based service discovery work in ECS with Cloud Map?

- A. ECS assigns fixed IPs to each service
- B. Cloud Map gives a DNS name that resolves to a task's current IP
- C. ECS pushes logs to DNS servers
- D. Each task registers its logs with Route 53

 **Correct Answer: B**

What happens if a container's IP changes in ECS with Cloud Map?

- A. You must manually update the DNS record
- B. You need to restart all services
- C. Cloud Map automatically updates the DNS mapping
- D. The service becomes unavailable

 **Correct Answer: C**
