

# Documentation: Deploying to ECS

## Preface:

This documentation provides a step-by-step guide to deploying a simple CRUD application using AWS Elastic Container Service (ECS), with the addition of implementing a Load Balancer to handle incoming traffic. The process includes creating a Docker container, pushing it to Amazon ECR, and deploying it to ECS.

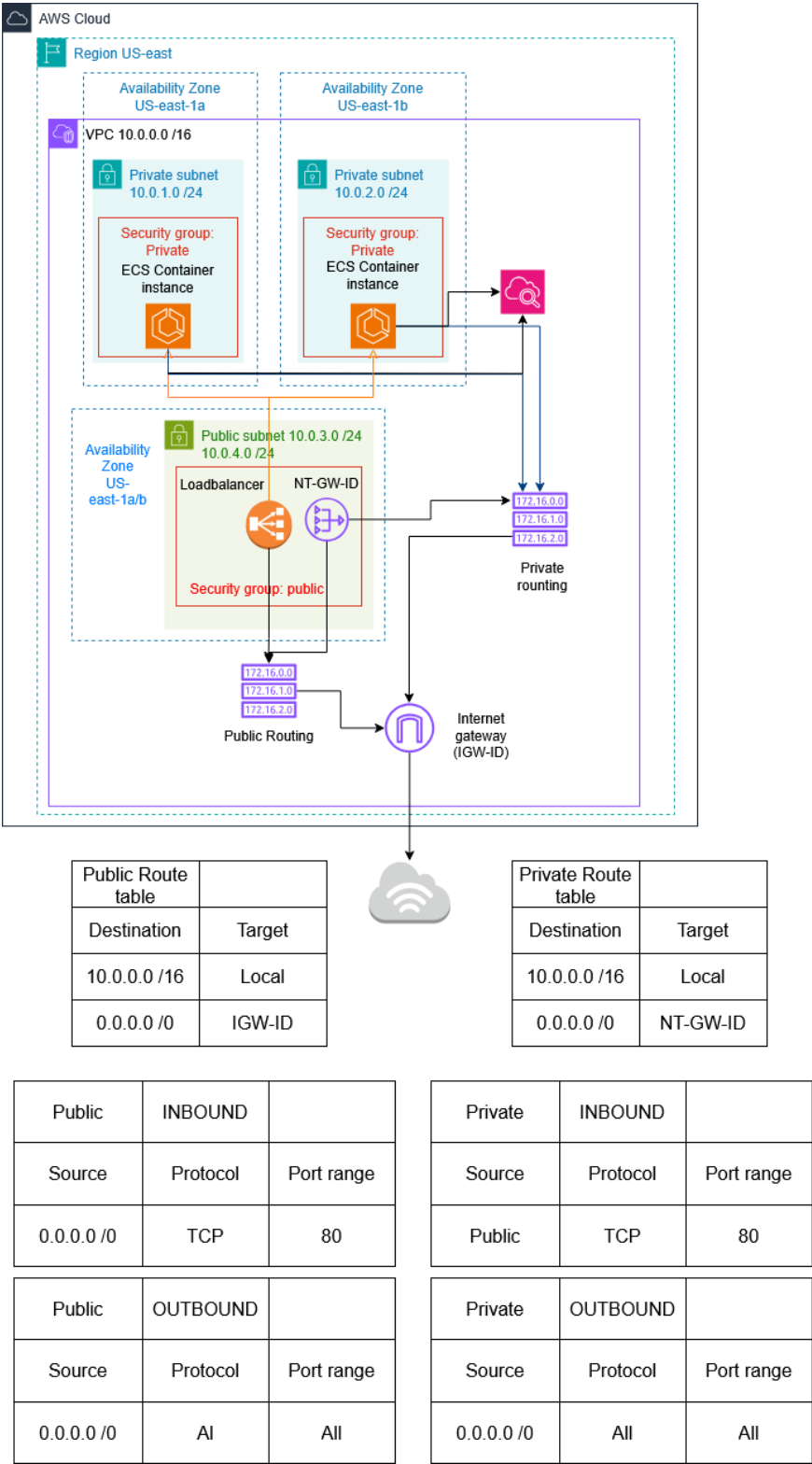
Before configuring the AWS environment, I first designed a setup by creating a deployment diagram. This design was tailored to meet specific requirements, such as running the application in a dedicated subnet with a public IP, serving HTTP on port 80, sending logs to AWS CloudWatch, and incorporating a Load Balancer to distribute traffic evenly across ECS tasks.

Once the design was finalized, I implemented it in the AWS web console, configuring the load balancer, ECS services, and all necessary components. Each step was thoroughly documented with clear instructions and screenshots for easy understanding and implementation.

## Table of Contents

Preface:.....	2
1. VPC design .....	4
2. Installing AWS CLI .....	5
3. Credentials.....	6
4. Docker image.....	8
4.1. Creating an image .....	8
4.2. Building the image.....	9
5. AWS ECR repository .....	10
5.1. Creating the AWS ECR repository .....	10
5.2. Pushing the image .....	11
6. VPC.....	13
7. Security groups .....	15
7.1. Public security group .....	15
7.2. Private security group .....	16
8. ECS (Elastic Container Service).....	17
8.1. ECS Cluster .....	17
8.2. ECS Task definition.....	18
8.3. Target groups .....	19
8.4. Load balancer.....	20
8.5. ECS service .....	21
9. Testing.....	23
9.1. Testing load balancer.....	23
9.2. CloudWatch .....	24

# 1. VPC design



## 2. Installing AWS CLI

### Prerequisite

The AWS Command Line Interface (CLI) is used to create a repository on Amazon Elastic Container Registry (ECR) and to allow Docker to authenticate with AWS ECR. The installation is only supported on Windows operating systems.

**Step 1:** Open a Windows Terminal.

**Step 2:** Execute the following command:

```
msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi /qn
```

### Breakdown

**msiexec.exe:** The Windows Installer executable, used for installing, modifying, or uninstalling software via .msi (Microsoft Installer) files.

**/i:** Stands for "install" and specifies the .msi package to be installed.

**https://awscli.amazonaws.com/AWSCLIV2.msi:** The direct URL to the AWS CLI v2 installer. This tells msiexec where to fetch the installer from.

**/qn:** Stands for "quiet mode with no user interface." This means the installation happens silently in the background without requiring user interaction.

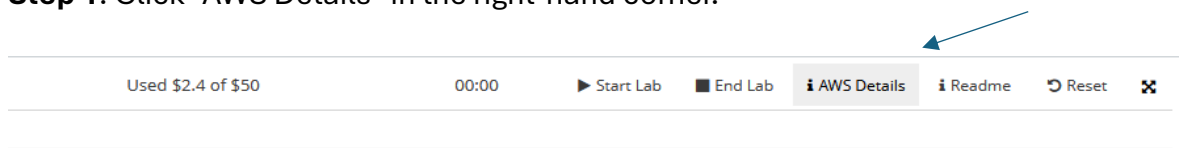
### 3. Credentials

#### Prerequisites:

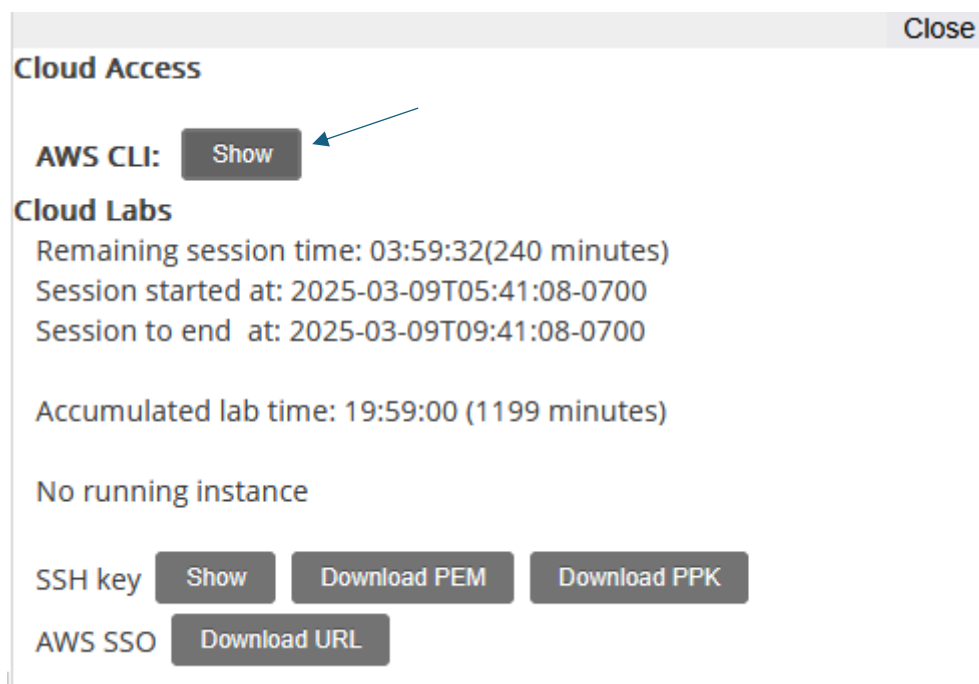
Before searching for your credentials, you must first start an AWS Learner Lab. This can be done by clicking the following [link](#) and clicking "Start Lab" in the top right-hand corner. *(You may need to log in before you can start a lab.)*

An AWS Learner Lab is a sandbox environment that allows you to explore and practice AWS services without worrying about costs or setup.

**Step 1:** Click “AWS Details” in the right-hand corner.



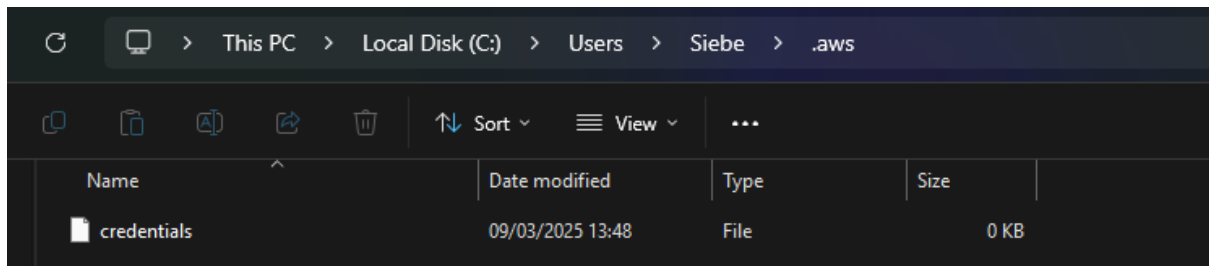
**Step 2:** Under “AWS Details” a new menu should pop-up, Click “Show” next to “AWS CLI:”.



**Step 3:** Copy the text that got revealed. Don't share it with anyone, these are your credentials and are used to authenticate with AWS.

**Step 4:** Open a Windows terminal.

**Step 5:** Create a new file named "credentials" (no extension) in the directory "~/aws/". This directory is in your current user directory.



**Step 6:** Paste your credentials in this file and save it.

## 4. Docker image

### Prerequisites:

This guide does not cover Docker installation or explain container virtualization. Before building a container, it is assumed that you have already installed Docker and have prior knowledge of Docker and how containers work.

### 4.1. Creating an image

**Step 1:** Create a new file named “Dockerfile” (no extension) in a directory this is easily accessible via a terminal.

**Step 2:** Paste the configuration below in the Dockerfile and save.

```
# Use an official Python runtime as the base image
FROM python:3.12

# Set the working directory inside the container
WORKDIR /app

# Install Git to enable repository cloning
RUN apt-get update && apt-get install -y git && rm -rf /var/lib/apt/lists/*

# Clone the application repository from GitHub
RUN git clone https://github.com/gurkanakdeniz/example-flask-crud.git .

# Create and activate a virtual environment, then install dependencies
RUN python3 -m venv venv && \
    . venv/bin/activate && \
    pip install --upgrade pip && \
    pip install -r requirements.txt

# Set environment variable to specify the Flask application entry point
ENV FLASK_APP=crudapp.py

# Initialize and migrate the database to set up required tables
RUN . venv/bin/activate && \
    flask db init && \
    flask db migrate -m "Create entries table" && \
    flask db upgrade

# Expose port 5000 for the Flask application
EXPOSE 80

# Start the Flask application with the virtual environment activated
```



```
CMD ["/bin/sh", "-c", ". venv/bin/activate && flask run --host=0.0.0.0 --port=80"]
```

## 4.2. Building the image

**Step 1:** Open a Windows terminal.

**Step 2:** Execute the following command:

```
Docker build -t flask-app-crud .
```

### Breakdown:

**docker build:** This is the Docker command to build an image.

**-t flask\_app\_crud:** The -t flag assigns a tag (name) to the built image.

**.(dot):** Specifies the build context, meaning Docker will look for a Dockerfile in the current directory to build the image.

## 5. AWS ECR repository

### Prerequisites:

The built image must be pushed to an Amazon Elastic Container Registry (ECR) repository to make it accessible within AWS.

### 5.1. Creating the AWS ECR repository

**Step 1:** Remain in the same Windows terminal as the previous step.

**Step 2:** Execute the following command:

```
aws ecr create-repository --repository-name flask-app-crud-repository --region us-east-1
```

### Output:

```
C:\Users\Siebe>aws ecr create-repository --repository-name flask_app_crud_repository --region us-east-1
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:539982272675:repository/flask_app_crud_repository",
    "registryId": "539982272675",
    "repositoryName": "flask_app_crud_repository",
    "repositoryUri": "539982272675.dkr.ecr.us-east-1.amazonaws.com/flask_app_crud_repository",
    "createdAt": "2025-03-07T18:42:53.616000+01:00",
    "imageTagMutability": "MUTABLE",
    "imageScanningConfiguration": {
      "scanOnPush": false
    },
    "encryptionConfiguration": {
      "encryptionType": "AES256"
    }
  }
}
```

### Breakdown:

**aws ecr create-repository:** Calls the AWS CLI to create a new ECR repository.

**--repository-name flask-app-crud-repository:** Specifies the name of the repository being created (flask-app-crud-repository).

**--region us-east-1:** Creates the repository in the us-east-1 AWS region.

## 5.2. Pushing the image

**Step 1:** Execute the following command, replacing <aws-account> with your registry ID, which can be found in the output of the previous command.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <aws-account>.dkr.ecr.us-east-1.amazonaws.com
```

### Breakdown:

**aws ecr get-login-password --region us-east-1:** Retrieves a temporary authentication password for ECR in the us-east-1 region.

**| (Pipe) :** Passes the output (password) from the first command into the next command.

**docker login --username AWS --password-stdin <aws-account>.dkr.ecr.us-east-1.amazonaws.com:** Logs Docker into the specified ECR registry using:

**--username AWS:** AWS is the username for ECR authentication.

**--password-stdin:** Reads the password from the standard input (stdin) instead of typing it manually.

**<aws-account>.dkr.ecr.us-east-1.amazonaws.com:** The URL of the private ECR registry for your AWS account. Replace <aws-account> with your actual AWS account ID.

**Step 2:** Execute the following command, replacing <aws-account> with your registry ID.

```
docker tag flask-app-crud <aws-account>.dkr.ecr.us-east-1.amazonaws.com/flask-app-crud-repository
```

### Breakdown:

**docker tag:** This command creates a new tag for an existing Docker image.

**flask-app-crud:** This is the **name of the locally built Docker image** that needs to be tagged.

**<aws-account>.dkr.ecr.us-east-1.amazonaws.com/flask-app-crud-repository:** This is the new tag (the fully qualified repository name in ECR) where the image will be stored.

**Step 3:** Execute the following command, replacing <aws-account> with your registry ID:

```
docker push <aws-account>.dkr.ecr.us-east-1.amazonaws.com/flask-app-crud-  
repository:latest
```

**Breakdown:**

**docker push:** This command uploads a Docker image to a container registry.

**<aws-account>.dkr.ecr.us-east-1.amazonaws.com/flask-app-crud-  
repository:latest:** This specifies the fully qualified image name.

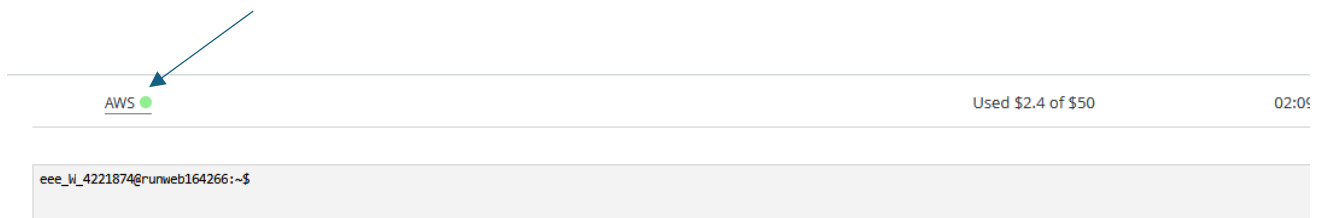
## 6. VPC

### Prerequisites:

An Amazon Virtual Private Cloud (VPC) is a logically isolated network within AWS where you can launch and manage AWS resources, such as EC2 instances, databases, and container services. It allows you to control networking settings, including IP addressing, subnets, routing, and security.

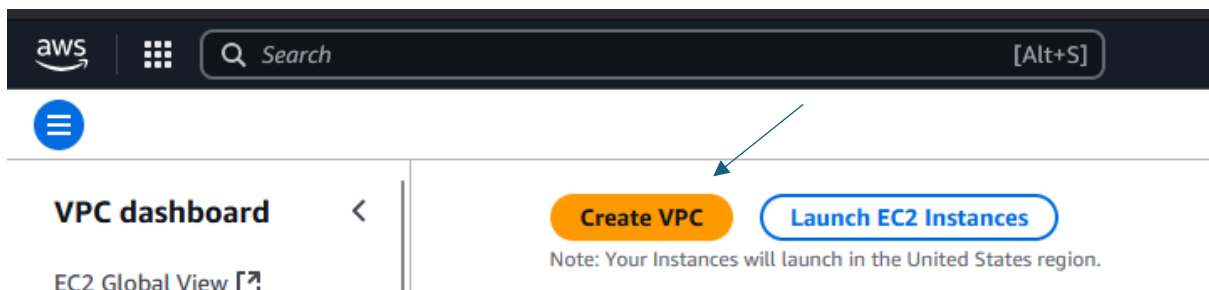
**Step 1:** Open the AWS learner lab again, make sure the lab is still active.

**Step 2:** Click “AWS”



**Step 3:** Search “VPC” in the search bar in the top left-hand corner.

**Step 4:** Click “Create VPC”



**Step 5:** Configure the VPC as followed: (when a setting is not described leave it default)

### VPC setting:

#### Resource to create:

- “VPC and more”

#### Name tag auto-generation:

- ☒ Auto-generate
- Text box: “flask-app-crud”

#### Number of availability Zones (AZs)

- 2

### Click “customize AZs”

- First availability zone: “us-east-1a”
- Second availability zone: “us-east-1b”

### Number of public subnets

- 2

### Number of private subnets

- 2

### Click “Customize subnets CIDR blocks”

Public subnet CIDR block in us-east-1a: 10.0.3.0/24

Public subnet CIDR block in us-east-1b: 10.0.4.0/24

Private subnet CIDR block in us-east-1a: 10.0.1.0/24

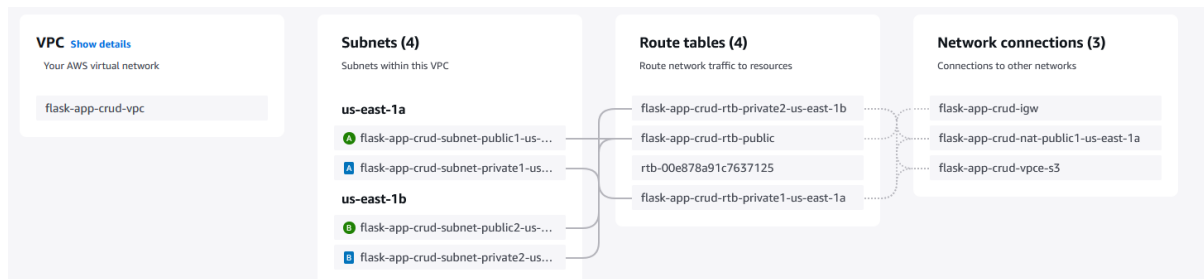
Private subnet CIDR block in us-east-1b: 10.0.2.0/24

### Nat gateway (\$)

- “In 1 AZ”

### Click “create VPC”

### Your resource map should look like this



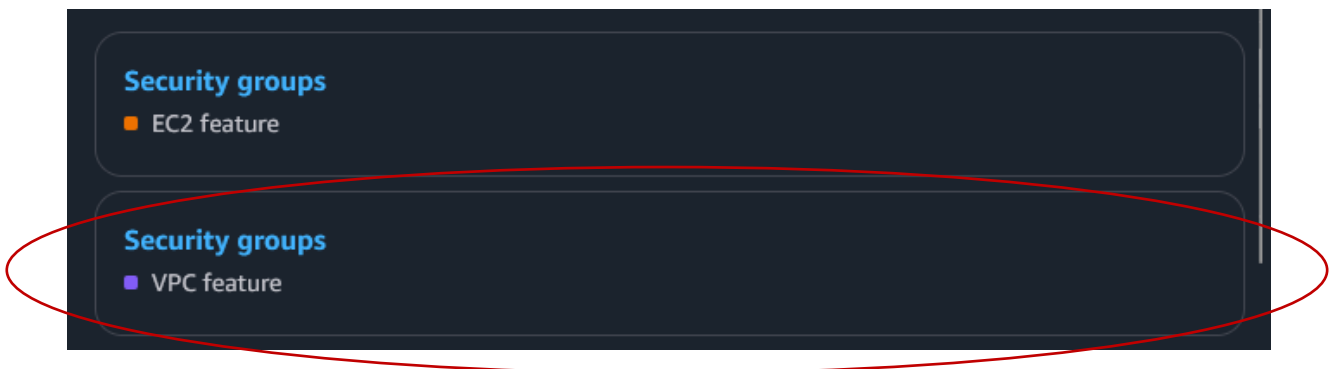
## 7. Security groups

### Prerequisites:

A Security Group (SG) in AWS acts as a virtual firewall for your Amazon EC2 instances, containers, and other AWS resources. It controls inbound and outbound traffic based on rules that you define.

### 7.1. Public security group

**Step 1:** Search “Security Groups” in the search bar in the top left-hand corner. There will be multiple, choose the second one.



**Step 2:** In the right-hand corner, click “Create security group”.

**Step 3:** Configure the security group as followed: (when a setting is not described leave it default)

#### Basic details:

- Security group name: “flask-app-crud-public”
- Description: “Allow http access from internet”
- VPC: vpc-xxxxx (flask-app-crud-vpc)

#### Inbound rules:

##### Click “Add rules”

Type: “http”

Source type: “Anywhere-IPv4”

Description: “Allow http access from internet”

Click “Create security group”.

## 7.2. Private security group

**Step 1:** Create a second security group.

**Step 2:** Configure the security group as followed: (when a setting is not described leave it default)

**Basic details:**

- Security group name: “flask-app-crud-private”
- Description: “Allow http access from load balancer”
- VPC: vpc-xxxxx (flask-app-crud-vpc)

**Inbound rules:**

Click “Add rules”

Type: “http”

Source type: “Custom”

Source: “flask-app-crud-public”

Description: “Allow http access from load balancer”

Click “Create security group”.

**You should now have 3 security groups:**

<input type="checkbox"/>	-	<a href="#">sg-082e4859d52bc8df4</a>	flask-app-crud-private	<a href="#">vpc-0490e51fb47f61d5f</a>
<input type="checkbox"/>	-	<a href="#">sg-0a27336705b4f808f</a>	flask-app-crud-public	<a href="#">vpc-0490e51fb47f61d5f</a>
<input type="checkbox"/>	-	<a href="#">sg-03f6acea0b1f81cb7</a>	default	<a href="#">vpc-0490e51fb47f61d5f</a>



## 8. ECS (Elastic Container Service)

### Prerequisites:

Amazon Elastic Container Service (ECS) is a fully managed container orchestration service that allows you to run, manage, and scale Docker containers on AWS. It eliminates the need to manually manage container scheduling, scaling, and networking.

### 8.1. ECS Cluster

#### Prerequisites:

An Amazon ECS Cluster is a logical grouping of EC2 instances or Fargate tasks that are running containerized applications using Amazon Elastic Container Service (ECS). It is where you organize and manage your resources to run containers.

**Step 1:** Search “ecs” in the search bar in the top left-hand corner.

**Step 2:** Click “Create cluster” in the right-hand corner.

**Step 3:** Configure the ecs cluster as followed: (when a setting is not described leave it default)

#### Cluster configuration:

- Cluster name: “flask-app-crud-cluster”

#### Infrastructure:

- !!! make sure that only “AWS Fargate (serverless)” is enabled.

#### Click “Create”

Cluster	Services
<a href="#">flask-app-crud-cluster</a>	1

## 8.2. ECS Task definition

### Prerequisites:

An **ECS Task Definition** is a blueprint used by Amazon Elastic Container Service (ECS) to run Docker containers on an ECS cluster. It is a JSON or YAML file that describes how Docker containers should run, including the configuration for containers, volumes, networking, and resource requirements. In other words, it tells ECS how to launch and manage containers.

**Step 1:** Search “ecs” in the search bar in the top left-hand corner.

**Step 2:** In the left panel, click “Task definitions”.

**Step 3:** click “Create new task definition”.

**Step 3:** Configure the ecs task definition as followed: (when a setting is not described leave it default)

### Task definition configuration

- Task definition family: “flask-app-crud-definition”

### Infrastructure requirements

- **!!! make sure that only “AWS Fargate (serverless)” is enabled.**
- Task size can be configured (not necessary)

### Click “Task roles”

- Task execution rule: “LabRole”

### Container – 1

- Name: “flask-app-crud”
- Image URI: “<aws-account>.dkr.ecr.us-east-1.amazonaws.com/flask-app-crud-repository:latest (replace <aws-account> with your registry ID)”
- Container port: 80
- Protocol: “TCP”
- Port name: “flask-app-crud-80-tcp”

### Logging

- Log collection: “Amazon CloudWatch”

### Click “Create”

## 8.3. Target groups

### Prerequisites:

A Target Group is a logical grouping of resources (such as EC2 instances, Lambda functions, or IP addresses) that a load balancer directs traffic to. It is used in conjunction with AWS Elastic Load Balancing (ELB) to route incoming traffic to one or more registered targets, based on specific rules or conditions.

**Step 1:** Search “Target groups (EC2)” in the search bar in the top left-hand corner.

**Step 2:** Click “Create target group” in the top right corner.


**Step 3:** Configure the Target group as followed: (when a setting is not described leave it default)

### Basic Configuration:

- Ip addresses
- Target group name: “flask-app-crud-target-group”
- VPC: choose: “flask-app-crud-vpc”

**Click “Next”**

**Click “Create target group”**

<input type="checkbox"/>	Name	ARN	Port	Protocol	Target type
<input type="checkbox"/>	<a href="#">flask-app-crud-target-group</a>	 <a href="#">arn:aws:elasticloadbalancin...</a>	80	HTTP	IP

## 8.4. Load balancer

### Prerequisites:

A Load Balancer in AWS is a service that automatically distributes incoming traffic across multiple targets, such as EC2 instances, containers, or IP addresses, to ensure that no single resource is overwhelmed. It helps increase the availability and fault tolerance of your application by balancing traffic, ensuring optimal resource utilization, and reducing the risk of downtime.

**Step 1:** Search “Load balancers” in the search bar in the top left-hand corner.

**Step 2:** Click “Create load balancer” in the top right corner.

**Step 3:** Configure the load balancer as followed: (when a setting is not described leave it default)

### Load balancer types:

#### Application Load Balancer

- Click “Create”

### Basic configuration:

- Load balancer name: “flask-app-crud-load-balancer”
- Scheme: “Internet-facing”

### Network mapping:

- VPC: choose: “flask-app-crud-vpc”
- Availability Zones and subnets
- ☒ us-east-1a (use1-az2)
- Subnet: flask-app-crud-subnet-public1-us-east-1a
- ☒ us-east-1b (use1-az4)
- Subnet: flask-app-crud-subnet-public2-us-east-1b

### Security groups

- Only enable: “flask-app-crud-public”



### Listeners and routing

- Protocol: “http”
- Port: “80”
- Default action: “flask-app-crud-target-group”

### Click “Create Load balancer”

Wait for the state of the load balancer to change to “Active”. This can take a while.

You should see this.

<input type="checkbox"/>	Name	DNS name	State	VPC ID
<input type="checkbox"/>	<a href="#">flask-app-crud-load-balancer</a>	 flask-app-crud-load-balanc...	 Active	vpc-0376fc863c058ed95

## 8.5. ECS service

### Prerequisites:

An ECS Service is a configuration that allows you to run and maintain a specified number of tasks or containers (which are instances of your containerized application) in an Amazon ECS Cluster. It ensures that your desired number of containers are always running and can automatically replace any failed or stopped tasks to maintain the desired state of your application.

**Step 1:** Search “ecs” in the search bar in the top left-hand corner.

**Step 2:** Click on the cluster “flask-app-crud-cluster”.

**Step 3:** Click the tab “Services”.

**Step 4:** Click “Create”.

**Step 5:** Configure the service as followed: (when a setting is not described leave it default)

### Compute configuration (advanced):

- Compute options: “Launch type”
- Launch type: “FARGATE”

### Deployment configuration:

- Application type: “service”
- Family: “flask-app-crud-definition”
- Service name: “flask-app-crud”
- Desired tasks: 2
- ☒ Availability Zone rebalancing

### Networking:

- VPC: “flask-app-crud”
- Subnets: only enable private subnets.
- Security group: ☒ “Use an existing security group”
- Security group name: “flask-app-crud-private”


### Load balancing (optional):

- ☒ Use load balancing
- Load balancer type: "Application Load Balancer"
- Application Load Balancer: "Use an existing load balancer"
- Load balancer: "flask-app-crud-load-balancer"
- Listener: ☒ "Use an existing listener"
- Target group: ☒ "Use an existing target group"
- Target group name: "flask-app-grud-target-group"

Click "create".

Wait for the process to finish.

Then you should have a cluster looking like this:

Cluster	Services	Tasks	Container instances
<a href="#">flask-app-crud-cluster</a>	1	 0 Pending   2 Running	0 EC2

## 9. Testing

### 9.1. Testing load balancer

**Step 1:** Search “Load balancers” in the search bar in the top left-hand corner.

**Step 2:** Click “Create load balancer” in the top right corner.

**Step 3:** Click “flask-app-crud-load-balancer”.

**Step 4:** Copy the DNS name and past it in a browser.

You should see this:

[home](#)  
header

---

Title

Add a Description

Add

Id	Title	Description	Operations
----	-------	-------------	------------

---

footer



**Step 5:** Enter a tile and description and click add.

**Step 6:** Refresh the page, the message should appear.

Each time you refresh the page; the message appears and then disappears. This happens because the message is only available on one web server and not on the others. With each refresh, you are switching between different servers.

# 9.2. CloudWatch

2b70caf7c3044c81ae9a3830a1abe6e1

Last updated  
March 09, 2025 at 18:24 (UTC+1:00)  

Configuration

Logs

Networking

Volumes (0)

Tags

Logs (201+)

View in CloudWatch

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

Filter container

Any container

Filter date time range

Since 1 hour ago

<

1

2

3

...

>

Timestamp (UTC+01:00)	Message	Container
March 09, 2025 at 18:24 (UTC+1:00)	10.0.4.161 - - [09/Mar/2025 17:24:36] "GET / HTTP/1.1" 200 -	flask-app-crud
March 09, 2025 at 18:24 (UTC+1:00)	10.0.3.213 - - [09/Mar/2025 17:24:35] "GET / HTTP/1.1" 200 -	flask-app-crud
March 09, 2025 at 18:24 (UTC+1:00)	10.0.4.161 - - [09/Mar/2025 17:24:06] "GET / HTTP/1.1" 200 -	flask-app-crud
March 09, 2025 at 18:24 (UTC+1:00)	10.0.3.213 - - [09/Mar/2025 17:24:05] "GET / HTTP/1.1" 200 -	flask-app-crud
March 09, 2025 at 18:23 (UTC+1:00)	10.0.4.161 - - [09/Mar/2025 17:23:36] "GET / HTTP/1.1" 200 -	flask-app-crud
March 09, 2025 at 18:23 (UTC+1:00)	10.0.3.213 - - [09/Mar/2025 17:23:35] "GET / HTTP/1.1" 200 -	flask-app-crud
March 09, 2025 at 18:23 (UTC+1:00)	10.0.4.161 - - [09/Mar/2025 17:23:06] "GET / HTTP/1.1" 200 -	flask-app-crud
March 09, 2025 at 18:23 (UTC+1:00)	10.0.3.213 - - [09/Mar/2025 17:23:05] "GET / HTTP/1.1" 200 -	flask-app-crud
March 09, 2025 at 18:22 (UTC+1:00)	10.0.4.161 - - [09/Mar/2025 17:22:36] "GET / HTTP/1.1" 200 -	flask-app-crud
March 09, 2025 at 18:22 (UTC+1:00)	10.0.3.213 - - [09/Mar/2025 17:22:35] "GET / HTTP/1.1" 200 -	flask-app-crud

Activate Windows

Go to Settings to activate Windows.