

Documentation: Multi-tier architecture in AWS

Preface:

This document serves as the final deliverable for the third and final assignment of the course, focusing on the design and implementation of a **multi-tier cloud architecture** for a web application. In this assignment, we extend the simple Flask CRUD application introduced in Assignment 1 and elevate it into a production-ready, cloud-hosted solution using a fully separated architecture.

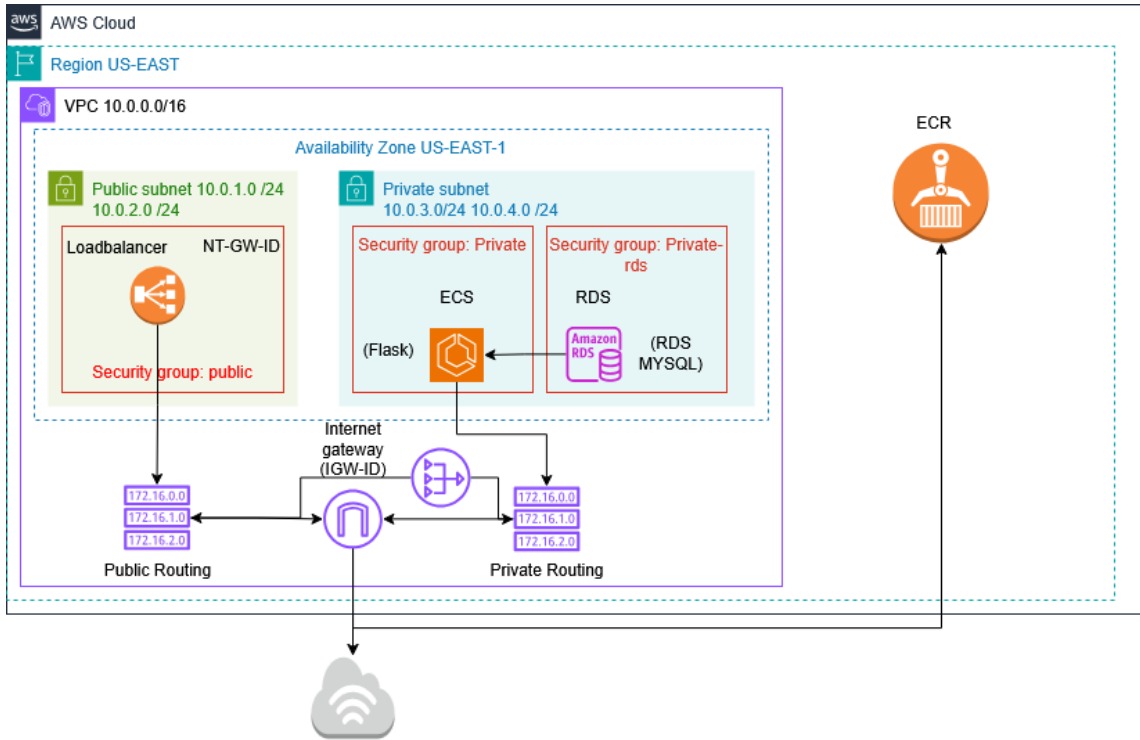
Unlike previous assignments, the application is no longer run in a single development container. Instead, it is deployed across multiple tiers, each responsible for a specific part of the application stack. This includes a front-end tier acting as a public-facing web proxy, a middle-tier WSGI application server, and a secure back-end database tier. All components are provisioned within a realistic and scalable AWS environment, with security, reliability, and maintainability as core principles.

The goal of this project is not just to deploy the application, but to **architect a robust, production-grade solution**. This includes integrating a production WSGI server such as Gunicorn, modifying the application for better configuration management via environment variables, and ensuring services like logging and monitoring are centralized using AWS CloudWatch. Additional enhancements such as HTTPS termination, load balancing, and database replication are encouraged to demonstrate advanced cloud architecture skills.

Table of Contents

Preface:.....	2
1. VPC design	4
2. Installing AWS CLI.....	5
3. Credentials.....	6
4. GitHub repository	8
4.1. Downloading the repository	8
5. AWS ECR repository	9
5.1. Creating the AWS ECR repository	9
6. Docker.....	10
6.1. Building the image	10
6.2. Pushing the image	10
7. VPC.....	12
8. Security groups.....	14
8.1. Public security group	14
8.2. Private security group	15
8.2.1. Middle-end security group.....	15
8.2.2. Back-end security group.....	15
9. RDS (Relational Database Service)	17
10. ECS (Elastic Container Service).....	19
10.1. ECS Cluster	19
10.2. ECS Task definition	19
10.3. Target groups.....	20
10.4. Load balancer	21
10.5. ECS service	22
11. Testing	24
GitHub repository	25

1. VPC design



Public Route table	
Destination	Target
10.0.0.0 /16	Local
0.0.0.0 /0	IGW-ID

Private Route table	
Destination	Target
10.0.0.0 /16	Local
0.0.0.0 /0	Nginx

Public	INBOUND	
Source	Protocol	Port range
0.0.0.0 /0	TCP	80

Public	OUTBOUND	
Source	Protocol	Port range
0.0.0.0 /0	All	All

Private	INBOUND	
Source	Protocol	Port range
Public	TCP	80

Private-middle-end	OUTBOUND	
Source	Protocol	Port range
0.0.0.0 /0	All	All

Private-rds	INBOUND	
Source	Protocol	Port range
Public	TCP	3306

Private-back-end	OUTBOUND	
Source	Protocol	Port range
0.0.0.0 /0	All	All

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. This 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.

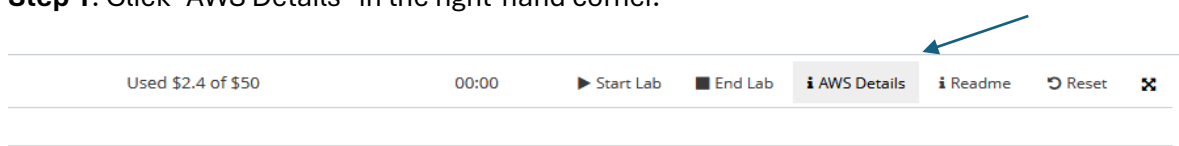
3. Credentials

Prerequisites:

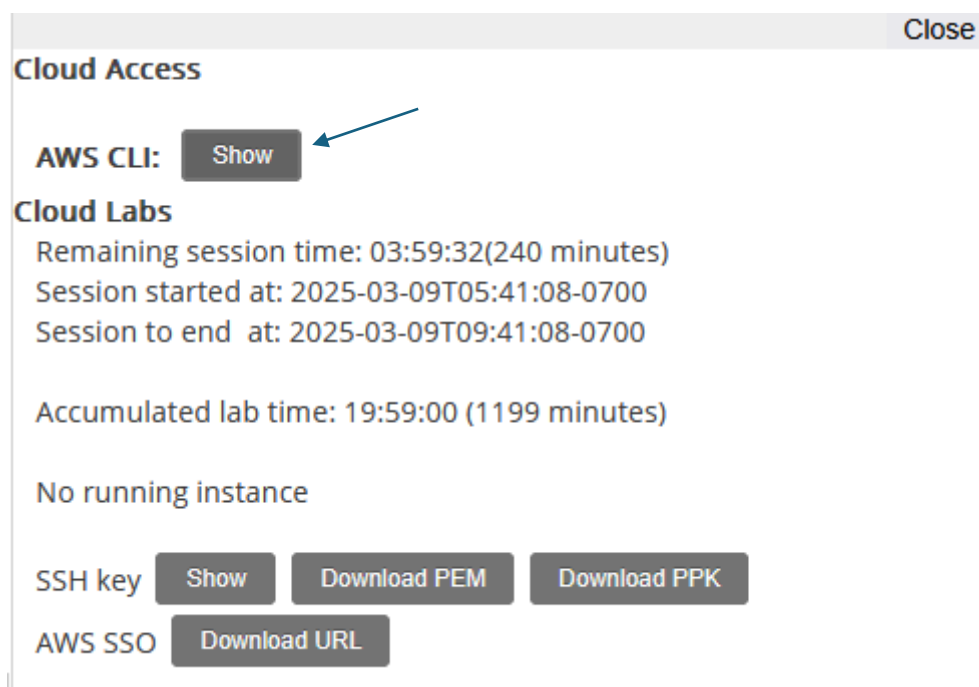
Before searching for your credentials, you must first start an AWS Learner Lab. This can be done by clicking the following [link](#) and clicking "Start Lab" in the top right-hand corner. *(You may need to log in before you can start a lab.)*

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.



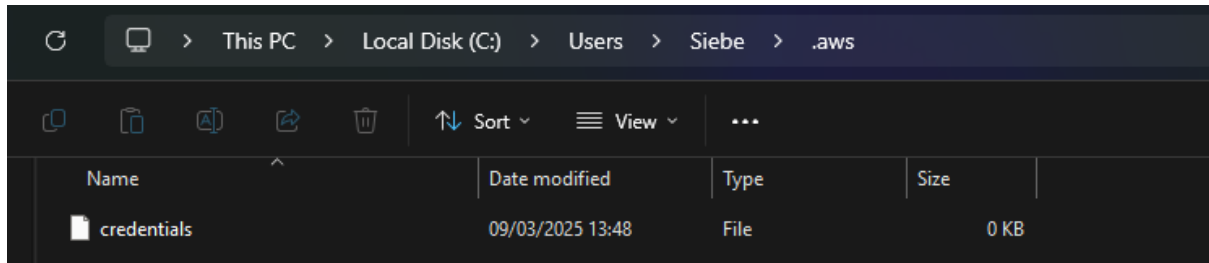
Step 2: Under “AWS Details” a new menu should pop-up, Click “Show” next to “AWS CLI:”.



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. GitHub repository

Prerequisites:

This guide does not cover Git installation or explain how Git or GitHub works.

4.1. Downloading the repository

Step 1: Open a terminal on the location you want the project to be stored.

Step 2: Execute the following command:

```
git clone "https://github.com/Sgeyskens/CloudAssignment3.git"
```


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-crud-repository --region us-east-1
```

Output:

```
C:\Users\Siebe>aws ecr create-repository --repository-name flask_app_repository --region us-east-1
{
  "repository": {
    "repositoryArn": "arn:aws:ecr:us-east-1:539982272675:repository/flask_app_repository",
    "registryId": "539982272675",
    "repositoryName": "flask_app_repository",
    "repositoryUri": "539982272675.dkr.ecr.us-east-1.amazonaws.com/flask_app_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-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.

6. Docker

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.

6.1. Building the image

Step 1: Open a Windows terminal in on the start folder of the project.

Step 2: Execute the following command:

```
docker build -t flask-crud docker\frontend\.
```

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.

docker\frontend\. : Specifies the build context, meaning Docker will look for a Dockerfile in that directory to build the image.

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

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-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-repository:latest: This specifies the fully qualified image name.

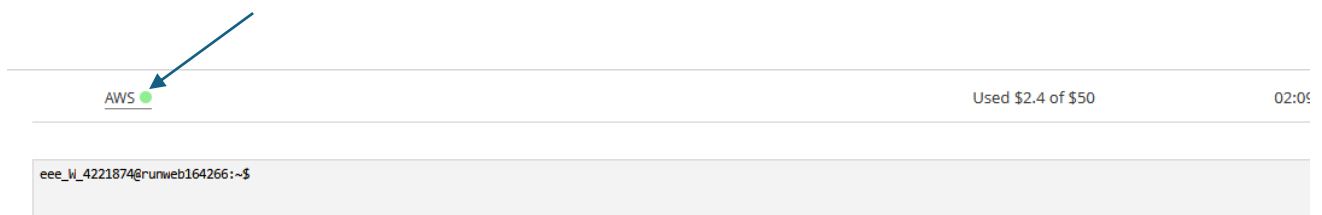
7. VPC

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.

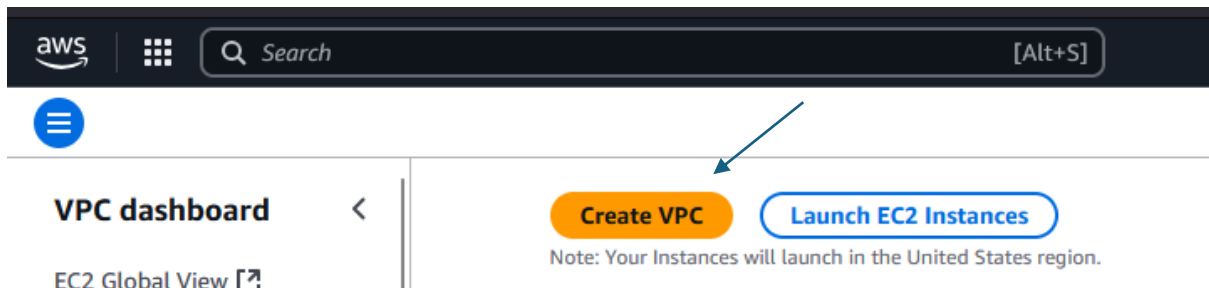
Step 1: Open the AWS learner lab again, make sure the lab is still active.

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-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: 10.0.1.0/24

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

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

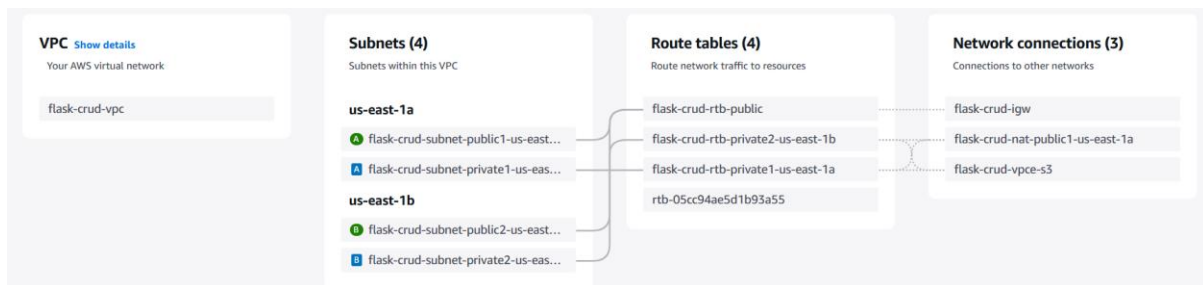
Private subnet CIDR block in us-east-1b: 10.0.4.0/24

Nat gateway (\$)

- “In 1 AZ”

Click “create VPC”

Your resource map should look like this



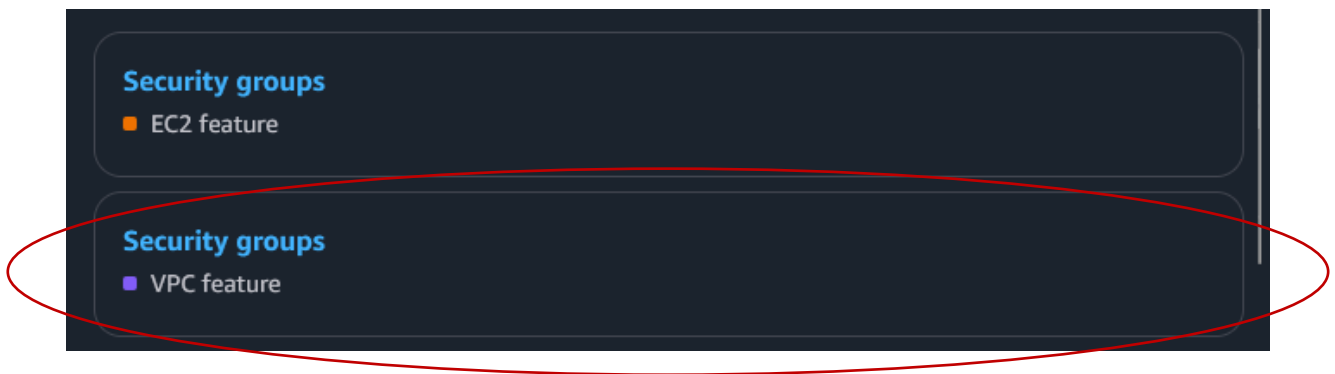
8. Security groups

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.

8.1. 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: “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”.

8.2. Private security group

8.2.1. Middle-end 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: “private”
- Description: “Allow http access from load balancer”
- VPC: vpc-xxxxx (flask-crud-vpc)

Inbound rules:

Click “Add rules”

Type: “http”

Source type: “Custom”

Source: “flask-crud-public”

Description: “Allow http access from load balancer”

Click “Create security group”.

8.2.2. Back-end 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: “private-rds”
- Description: “Allow MYSQL access from middle-end”
- VPC: vpc-xxxxx (flask-crud-vpc)

Inbound rules:

Click “Add rules”

Type: “MYSQL/Aurora”

Source type: “Custom”

Source: “flask-crud-private”

Description: “Allow http access from middle-end”

Click “Create security group”.

You should now have 4 security groups:

<input type="checkbox"/>	-	sg-0c0602d650cd42603	default	vpc-045f1b4cad988d80e
<input type="checkbox"/>	-	sg-02e0b880f2eb51997	private-rds	vpc-045f1b4cad988d80e
<input type="checkbox"/>	-	sg-01c250e1d30f52751	public	vpc-045f1b4cad988d80e
<input type="checkbox"/>	-	sg-0ddac9f8a12f80a2c	private	vpc-045f1b4cad988d80e

9. RDS (Relational Database Service)

Prerequisites:

Amazon RDS (Relational Database Service) is a managed cloud service by AWS that simplifies the setup, operation, and scaling of relational databases. It supports several database engines, including MySQL, PostgreSQL, MariaDB, Oracle, and SQL Server. With RDS, AWS handles tasks like backups, patching, and replication, allowing developers to focus on their applications.

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

Step 2: Under database instances, click DB instances (0/40).

Step 3: Click “Create database”.

Step 4: Configure the database as followed: (when a setting is not described leave it default)

Choose a database creation method:

- Check: “Standard create”

Engine options:

- Check: “MySQL”

Templates:

- Check: “Free tier”

Settings:

- DB instance identifier: “db-flask-crud”

Credentials Settings:

- Master username: “flaskcrud”
- Master password: “flaskcrud”
- Confirm master password: “flaskcrud”

Connectivity:

- Check “Don’t connect to an EC2 compute resource”
- Check “IPv4”
- Virtual Private cloud “VPC”: “flask-crud-vpc”
- Public access: “No”

VPC security group (firewall):

- Check: Choose existing
- Existing VPC security groups: “private-rds”

Additional configuration:

- Initial database name: “flaskcrud”

Backup:

- Uncheck “Enable automated backups”

Click “Create database”

10. 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.

10.1. 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-crud”

Infrastructure:

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

Click “Create”

Cluster	Services
flask-crud	1

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

Click “Create”

	Task definition	▼	Status of last revision
<input type="radio"/>	flask-crud		✓ ACTIVE

10.3. 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-crud”
- VPC: choose: “flask-crud-vpc”

Click “Next”

Click “Create target group”

<input type="checkbox"/>	flask-cruf	 arn:aws:elasticloadbalancin...	80	HTTP	IP	flask-crud
--------------------------	------------	--	----	------	----	------------

10.4. 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-crud”
- Scheme: “Internet-facing”

Network mapping:

- VPC: choose: “flask-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-crud”

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"/>	flask-app-crud-load-balancer	 flask-app-crud-load-balanc...	 Active	vpc-0376fc863c058ed95

10.5. 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-crud”.

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-crud”
- Service name: “flask-app-crud”
- Desired tasks: 1
- ☒ Availability Zone rebalancing

Networking:

- VPC: “flask-crud-vpc”
- Subnets: only enable private subnets.
- Security group: ☒ “Use an existing security group”
- Security group name: “flask-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-crud”
- Listener: ☒ “Use an existing listener”
- Target group: ☒ “Use an existing target group”
- Target group name: “flask-grud-target”

Click “create”.

Wait for the process to finish.

Then you should have a cluster looking like this:

Cluster	Services	Tasks	Container instances	CloudWatch monitoring	Cap
flask-crud	1	<div><div></div>0 Pen... 1 Run...</div>	0 EC2	Default	No

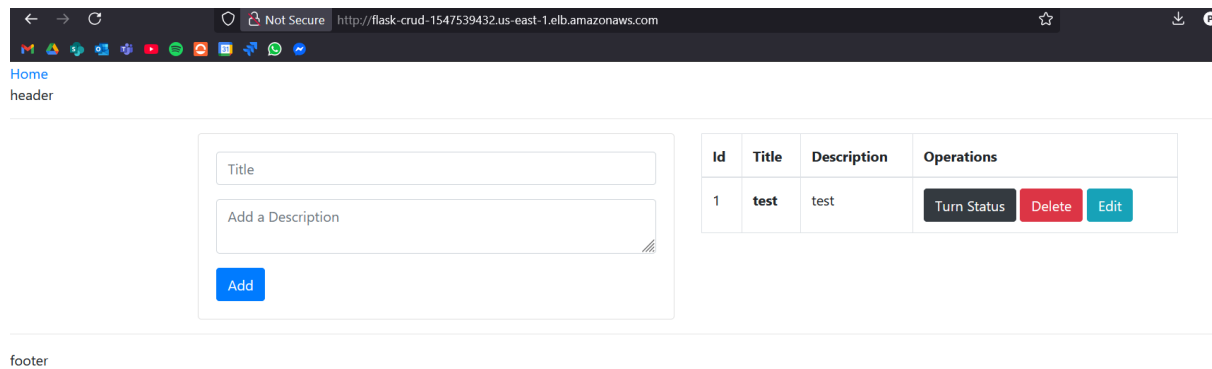
11. Testing

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-crud”.

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



GitHub repository

<https://github.com/Sgeyskens/CloudAssignment3>

Changes made to default configuration

Config.py

```
import os
basedir = os.path.abspath(os.path.dirname(__file__))

class Config(object):
    SECRET_KEY = os.environ.get('SECRET_KEY', 'do-or-do-not-there-is-no-try')

    db_url = os.environ.get('DATABASE_URL')
    db_user = os.environ.get('DATABASE_USERNAME')
    db_pass = os.environ.get('DATABASE_PASSWORD')

    SQLALCHEMY_DATABASE_URI =
f"mysql+pymysql://{db_user}:{db_pass}@{db_url}:3306/flaskcrud"
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

I updated the configuration to use environment variables for the database URL, username, and password, so these can be provided when creating the container. The application now connects to a MySQL database instead of using the default SQLite database.