# Introduction to Cloud Computing for MLOps

**Overview**

This lab course introduces essential cloud computing concepts for MLOps, using Amazon Web Services (AWS) cloud platform. We will explore key AWS services and their roles in creating a robust MLOps infrastructure.

**Lab plan [10 points + 1 bonus point]**

1. Account setup & IAM configuration [1 point]
2. Amazon S3 setup and file management [1.5 point]
3. Elastic Container Registry (ECR) & Docker management [1.5 point]
4. Virtual Private Cloud (VPC) configuration [3 points]
5. AWS Fargate deployment & CloudWatch configuration [3 points]
6. Application testing and monitoring [1 point]

**AWS services & tools**

1. Identity and Access Management (IAM)
2. AWS CLI
3. Amazon Simple Storage Service (S3)
4. Elastic Container Registry (ECR)
5. Virtual Private Cloud (VPC)
6. Elastic Container Service (ECS)
7. AWS Fargate
8. AWS CloudWatch

**If you are using your own account**, start from the beginning. Cost will be under $1 total, and we will set up billing monitoring and alerts. You will start by configuring security and permissions via IAM.

**If you are using AWS Educate**, start from section 1.3, as it doesn't allow IAM configuration. section 1.4 also can be omitted.

This lab concerns using AWS Console and manual setup, to get a good feeling of how AWS operates visually. This way, you can interactively set up cloud services, experiment with architecture, and introduce experimental infrastructure changes. In the next lab, we will use Infrastructure as Code (IaC) approach to implement modular and versioned configuration for some services used here.

# 1. Account setup & IAM configuration

When starting with a cloud, it is **crucial** to set up your AWS account properly for security and cost control. By default, you use the root account, which is not recommended due to its unrestricted access and auditing difficulties. Thus, we will start with securing access, creating administrative IAM user, and implementing billing alerts. Identity and Access Management (IAM) is the core AWS service for authorization & authentication of users and services.

**Why do we need this?**

1. **Least privilege principle** - you should always use the smallest set of permissions required for work.
2. **Security** - minimize attack surface, particularly in case of compromised credentials.
3. **Auditing** - root account is challenging to control, track and audit.

The root account has full access to all services, and its actions are hard to fully track and audit, clearly breaking the least privilege principle. In fact, AWS recommends removing all access keys for root accounts.

**Step 1.1: Enable Multi-Factor Authentication (MFA) for root account [0.25 point]**

1. Sign in to your root account and navigate to the **Identity and Access Management (IAM)** service.
2. At the top of the **IAM dashboard**, locate the security recommendations.
3. Set up MFA by downloading (or using your already-installed) authenticator app on your mobile device and following the on-screen instructions on the AWS IAM page.

**Step 1.2: Create a new IAM user (admin user) [0.25 point]**

1. In the IAM Console, click **Create User** and choose "I want to create an IAM user".
2. Specify the user details.
3. Create a new IAM group called **admin** and attach the policy `AdministratorAccess`.
4. Add the new user to the **admin** group.
5. Review the user groups and confirm that the user appears with the `AdministratorAccess` policy attached (via the **admin** group).
6. Before signing in with your new user, note the "Sign-in URL for IAM users in this account" on the Dashboard. Since this URL is complicated, customize it by creating an account alias.

7. Use another browser or an incognito window to navigate to the new URL and sign in using your new IAM user.

**Important:** Set up an MFA for your administrator account.

**Note:** Do not lose or forget your root account credentials. Otherwise, you may need to contact AWS support.

**Step 1.3: Connect to AWS using the CLI [0.25 point]**

1. Install the **AWS CLI** by following the AWS CLI installation guide and verify the installation with `aws --version`.
2. In the IAM Console, select your new user and navigate to the **Security Credentials** tab.
3. Create a new access key and store it securely (treat it like a password—do not share it!).
4. When you start your **awsadacemy** session under **AWS Details** the access keys will be displayed. Copy them and paste to ~/.aws/credentials.
5. Test the configuration by running a command, e.g. `aws iam list-users`.
6. *(Optional)* Remove your user from the admin group temporarily and test if you can still list users, confirming that access keys work as expected.

Additional references:

- Root User Best Practices
- Root User Tasks

**Step 1.4: Set Up Billing and Cost Alerts [0.25 point]**

1. Navigate to the **Billing and Cost Management dashboard** from the AWS Management Console.
2. Activate access to billing data by navigating to Billing Dashboard -> Preferences, check the box for "Receive Billing Alerts" and save the changes.
3. Set up a billing budget. Choose the "Cost budget" template, set a monthly limit (e.g. $5), and configure it to monitor actual costs.
4. Add an email notification by entering your email address to receive alerts when costs approach or exceed the budget.

Additional references:

- What is cost management
- Creating a budget

## 2. Amazon S3 setup and file management

AWS Simple Storage Service (S3) is an object storage, designed for storing arbitrary objects (bytes) identified by a string key. It is a core AWS storage, and main storage used in MLOps, e.g. for datasets, ML model artifacts, and training logs. It offers scalability, versioning, and integration with other AWS services.

**Why do we need this?**

1. **Object storage** - very convenient, and ideal for storing large ML objects, e.g. dataset files or models.
2. **Scalability & low cost** - S3 scales automatically and basically infinitely. It also only charges for storage used and requests made, and is the cheapest storage available.
3. **Versioning & archiving** - you can configure objects versioning, use long-term archiving options, and set lifecycle policies. Those options allow rich, yet easy to use management.

**Step 2.1: Create a private S3 bucket [0.25 point]**

1. Navigate to the **S3 Console**.
2. Click **Create bucket** and enter a globally unique bucket name. Note that S3 has a global namespace for buckets. Thus, your bucket name must be unique worldwide, across all AWS regions and buckets.
3. Leave all default configurations as-is.

**Step 2.2: Upload required files [0.25 point]**

1. Download the necessary files from Lab 1.
2. Upload these files to the newly created S3 bucket.

**Step 2.3: Integrate S3 into a ML workflow [1 point]**

1. In your sentiment analysis application (from Lab 1 homework), remove any previously downloaded files from Google Drive **OR**

2. Download ready app from Lab6.
3. Read readme.md and run the application and verify that everything works as expected.

Additional references:

- Amazon S3
- Boto3

## 3. Elastic Container Registry (ECR) & Docker management

ECR is a managed Docker registry in AWS, similar in concept to Dockerhub. It provides secure, scalable storage for your Docker images, tightly integrated with other AWS services.

**Why do we need this?**

1. **Versioning -** you can manage and version your Docker images easily.
2. **Security** - it's integrated with IAM and supports KMS encryption, helping with the least principle privilege.
3. **Integration** - seamlessly integrates with other AWS services like ECS, with Docker images being simply "visible" to other services after permissions setup.

**Step 3.1: Create an ECR repository [0.5 point]**

1. Navigate to **ECR** -> **Repositories** in the AWS Console.
2. Click **Create Repository**.
3. Choose to use **AES-256 encryption**
4. Remember, it's a good practice to create separate repositories for each application. This helps with management and versioning (through tagging systems).

**Step 3.2: Authenticate your Docker client to ECR [0.25 point]**

1. Run the following command. Replace <placeholders> with your actual region and account details. You can also find this command in ECR when you click on your repository.


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

**Step 3.3: Build and tag your Docker image [0.5 point]**

1. Build your Docker image:

```
docker build -t sentiment-app .
```

2. Tag the image for ECR:

```
docker tag sentiment-app <aws_account_id>.dkr.ecr.<region>.amazonaws.com/<your-repository-name>:14.01.25
```

**Step 3.4: Push the image to ECR [0.25p]**

1. Push the tagged image:

```
docker push <aws_account_id>.dkr.ecr.<region>.amazonaws.com/<your-repository-name>
```

Additional references:

- What is ECR?
- Docker Hub

## 4. Virtual Private Cloud (VPC) configuration

A VPC provides a logically isolated network environment for your AWS resources. This offers complete control over IP ranges, subnets, routing, and security. Such isolation can often be required for MLOps in regulated industries, e.g. in finance, biomedical applications, or legal applications.

**Why do we need this?**

1. **Isolation** - resources are isolated by default, ensuring only authorized traffic reaches them.
2. **Control** - customize IP ranges, subnets, route tables, and gateways.
3. **Scalability** - plan for future growth with proper IP planning and range management.
4. **Flexibility** - you can configure both public and private subnets, exposing only necessary parts.

**Step 4.1: Create a new VPC [0.5 point]**

1. Go to the **VPC Console** in the AWS Management Console. As you can see, one default VPC is already created by AWS, but we will create our own.
2. Configure the following:
    a. **Name:** `sentiment-app-vpc`.
    b. **IPv4 CIDR Block:** Use `10.0.0.0/16`. This has ~65,536 IP addresses, see e.g. cidr.xyz for details.

c. **Tenancy:** Select **Default** unless dedicated hardware is required.

Additional references:

- What is AWS VPC?

**Step 4.2: Create public and private subnets [0.5 point]**

Subnets divide your VPC into smaller segments with specific roles, e.g. public-facing or private backend. For this setup, we will create two public subnets and two private subnets across multiple Availability Zones (AZs) for high availability.

**Public subnets:**

1. In the VPC Console, navigate to **Subnets** and click **Create Subnet**.
2. Configure for each public subnet:
   a. **Name:** e.g., `public-subnet-1` and `public-subnet-2`.
   b. **Availability Zone:** Choose different AZs (e.g., `eu-west-1a` and `eu-west-1b`) for redundancy.
   c. **IPv4 CIDR Block:** e.g., `10.0.1.0/24` for public-subnet-1 and `10.0.2.0/24` for public-subnet-2.

**Private subnets:**

1. Repeat the process for private subnets:
   a. **Name:** e.g., `private-subnet-1` and `private-subnet-2`.
   b. **IPv4 CIDR Block:** e.g., `10.0.3.0/24` for private-subnet-1 and `10.0.4.0/24` for private-subnet-2.

Additional references:

- Subnets for you VPC

**Step 4.3: Create an Internet Gateway [0.25 point]**

Internet Gateway enables resources in public subnets to send/receive traffic to/from the internet.

1. Navigate to **Internet Gateways** in the VPC Console.
2. Click **Create Internet Gateway**.
3. Name it (e.g., `sentiment-app-igw`).
4. Attach **IGW** to our new **VPC.**

Additional references:

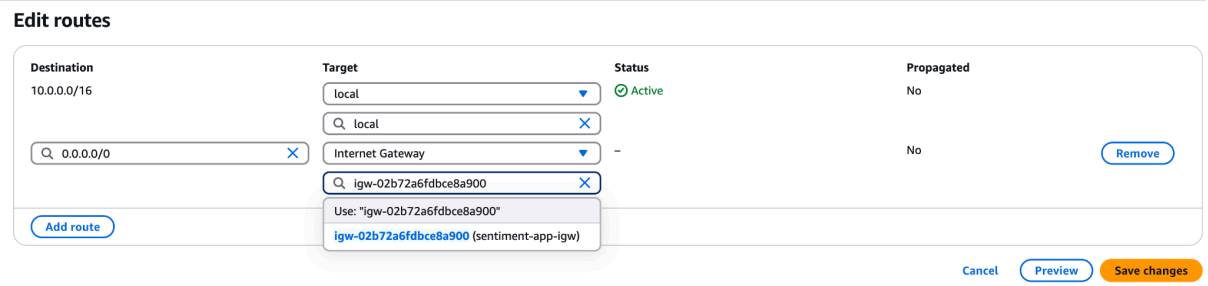
- Enable VPC internet access using internet gateways

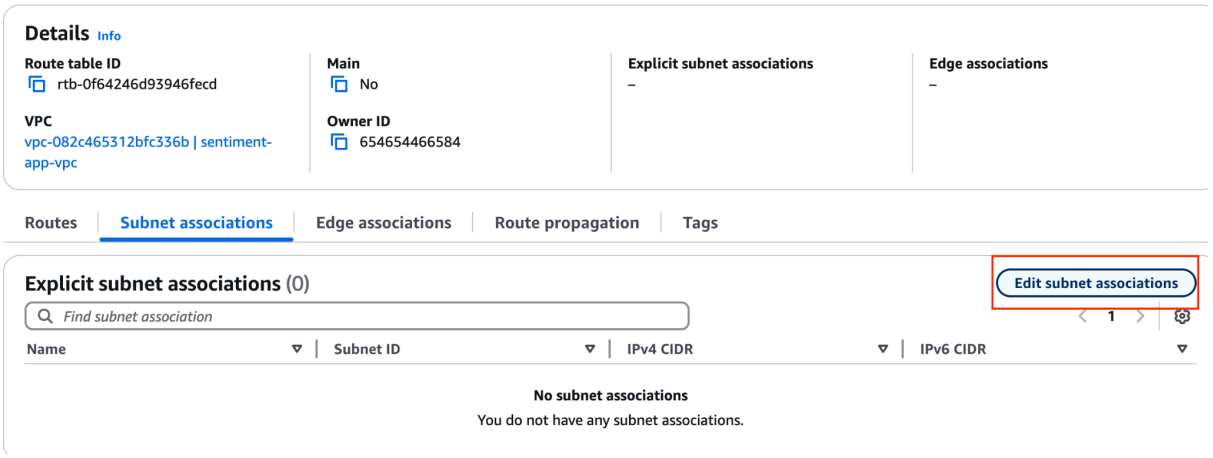
**Step 4.4: Configure route tables [0.25 point]**

Public route tables enable external communication for public subnets, while private route tables keep backend resources isolated.

**Public route table:**

1. In the VPC Console, go to **Route Tables** and create a new route table (e.g., `public-route-table`).
2. Associate this table with your public subnets (`public-subnet-1` and `public-subnet-2`).




3. Add a route:
    - **Destination:** `0.0.0.0/0`
    - **Target:** The Internet Gateway (e.g., `igw-<id>`)

**Private route table:**

1. Create another route table (e.g., `private-route-table`) and associate it with your private subnets (`private-subnet-1` and `private-subnet-2`).

2. Leave it without routes for now until you configure a NAT Gateway.

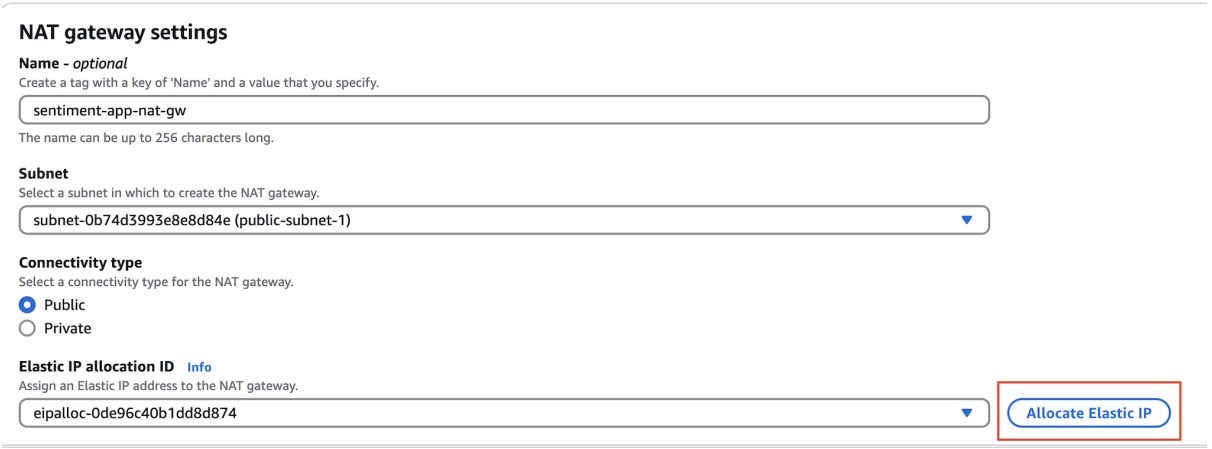
Additional references:

- Configure route tables

**Step 4.5: Create a NAT Gateway [0.25 point]**

NAT Gateway allows instances in private subnets to access the internet without exposing them to inbound traffic. It ensures that private instances can initiate outbound connections without being directly exposed, allowing e.g. accessing online APIs, models, or updating other web-based services with predictions.

1. Navigate to **NAT Gateways** in the VPC Console.
2. Click **Create NAT Gateway**.
3. Configure:
    a. Place it in one of your public subnets (e.g., `public-subnet-1`) so it can access the Internet Gateway.
    b. Allocate an Elastic IP address for the NAT Gateway.




4. Update the private route table by adding a route:
    a. **Destination:** `0.0.0.0/0`
    b. **Target:** The NAT Gateway (e.g., `nat-<id>`)

Additional references:

- NAT gateways

**Step 4.6: Configure security groups [0.25 point]**

Security groups act as virtual firewalls at the instance level. Their stateful nature means that when an inbound rule permits traffic, the corresponding outbound response is automatically allowed.

We need two **Security Groups. It** is good practice to keep all private resources in a private firewall, but to address public access to our app we need to place ALB in the public Security Group.

**Load Balancer Security Group**

- **Inbound Rules:**
    1. Allow TCP traffic (port80) from anywhere (`0.0.0.0/0`).
- **Outbound Rules:**
    1. By default, allow all outbound traffic, so instances can reach external services.

**Resource Security group**

- **Inbound Rules:**
    1. Allow TCP traffic (port 8000) from **Load Balancer Security Group.**
- **Outbound Rules:**
    1. By default, allow all outbound traffic, so instances can reach external services.

Additional references:

- Control traffic to your AWS resources using security groups

**Step 4.7: Set up an Application Load Balancer (ALB) [1p]**

Application Load Balancer (ALB) distributes incoming traffic across multiple targets to avoid overloading a single instance with traffic, which minimizes latency and maximizes throughput. Targets can be e.g. EC2 instances, ECS Fargate containers, and other traffic-serving servers.

**Create a target group:**

1. **Go to EC2 > Load Balancing > Target Groups**
2. Create a target group that includes the resources (instances, containers, IP addresses) that will receive traffic.
3. Configuration:
    a. **Target Type:** `IP addresses`
    b. **Protocol & Port:** `http & 8000`
    c. **Health Checks:** `/health`
    d. **Target Registration:** Add your targets (e.g., ECS tasks). We will update this after the ECS task is registered.

**Create the Application Load Balancer:**

1. Go to **EC2** > **Load Balancers** and click **Create Load Balancer**.
2. Choose **Application Load Balancer**.
3. Configure:
    a. **Name:** e.g., `sentiment-app-alb`
    b. **Scheme:** "Internet-facing" if public, or "Internal" if only for VPC access. We are choosing internet-facing, because we need to call it outside our VPC.
    c. **VPC:** Select the VPC you created.
    d. **Availability Zones:** Choose at least two public subnets in different AZs.
    **e.** **Security groups:** Choose our ALB security group.
    **f.** **Listeners & Routing:**
        i. Add a listener for HTTP (port 80). It allows the world to communicate with our ALB.
        ii. Forward traffic to the target group.

Additional references:

- AWS Load Balancer Behind the Scenes

## 5. AWS Fargate deployment & CloudWatch configuration

**AWS Fargate** is a serverless compute engine that lets you run containers without managing underlying EC2 instances. It allows you to automatically scale up and down based on demand, which is very useful for MLOps, where we typically must scale horizontally, since model inference requires the whole CPU.

**AWS ECS** is a fully managed container orchestration service that enables you to deploy, manage, and scale containerized applications on AWS.

**Together** ECS decides what to run and when, while Fargate supplies the compute resources and execution environment in a pay-as-you-go model, with automatic scaling and strong task isolation.

| Feature | EC2 launch type | Fargate launch type |
|---|---|---|
| **Server management** | You manage EC2 instances | AWS manages the infrastructure |
| **Task scaling** | Limited by available EC2 capacity | Automatic and flexible |

| | | |
|---|---|---|
| **Scale-up speed** | Depends on ASG and EC2 startup, but slower | Fast – no EC2 provisioning needed |
| **Billing mode** | Pay for EC2 instances, even if idle | Pay only for actual usage (CPU & memory time) |

**Why use AWS Fargate?**

1. **Serverless** - no need to manage servers, fully managed, automatically scaled, cost-efficient.
2. **Simplified management** - AWS handles patching, scaling, and infrastructure maintenance.

**Step 5.1: Create an ECS cluster [0.25 point]**

1. Navigate to the **ECS Console**.
2. Click **Create Cluster**.
3. Infrastructure: skip it for now, we will define fargate later in task definition.
4. Provide a cluster name (e.g., `sentiment-app-cluster`) and click **Create**.

**Step 5.2: Create a task definition [0.5 point]**

A task definition is a blueprint for your application. It tells ECS how to run your containers and encapsulates all settings needed for deployment.

1. Infrastructure requirements:
    a. Launch type: AWS Fargate
    b. Operating system: `linux/X86_64`
    c. CPU/Memory: `1 vCPU / 3GB`
    d. `Task role: labrole`
    e. Task execution role: `labrole`
2. Define the container settings
    a. `use the image url from ECR`
    b. `expose valid port`
3. **awsvpc** networking mode ensures each task gets its own elastic network interface (ENI).
4. Logging: `Use log collection`
5. Leave rest options as default. Including the Logging section.

**Step 5.3: Create an ECS Service [0.75 point]**

An ECS service ensures your application runs continuously and scales automatically based on demand.

1. In the ECS Console, select your cluster and go to the **Services** tab.
2. Select the Task Definition from Step 5.2.
3. Click **Create** and choose **Fargate** as the launch type.
4. Specify the desired number of tasks (instances). `1 no of task is enough`
5. Choose the VPC and private subnets where tasks will be deployed.
6. Configure security groups: `chose our resource security group`
7. Under **Load Balancing**, select "Application Load Balancer", then choose your ALB, listener and target group.

**Step 5.4: Running the Service [0.25 point]**

1. Review all the settings and click **Create Service**.
2. ECS will launch the specified tasks using your Task Definition and associate them with the selected target group and ALB.
3. If you wait suspiciously long for the deployment > 10 minutes. Then go to the events section and check out potential issues.

**Step 5.5: Access the Service [0.25 point]**

1. Retrieve the ALB DNS name from the EC2 Console (under **Load Balancers**).
2. Open a browser and navigate to the ALB URL to test your application.

Additional references:

- AWS Fargate Getting Started

## 6. Application testing and monitoring

After deploying your service, ensure everything is working properly. At this point, all cloud-related work is done, and from the outside perspective, the service is just another REST service.

**Step 6.1 Access the Application [0.25 point]**

Use the ALB DNS name to open the application in a web browser or API client. Make sure that it is working as manual testing.

**Step 6.2 Test Endpoints [0.5 point]**

Validate that all endpoints respond as expected by writing a Python script with appropriate tests for automated testing. You can either navigate to /docs endpoint to display all available options and make requests or just make curl POST call.

**Step 6.3 Monitor with CloudWatch [0.25 point]**

Check CloudWatch for logs, metrics, and any alarms that indicate issues. This is the basic monitoring service at AWS, and the basis of its observability services and platforms.

## 7. Documenting lab

As homework, you are expected to complete this entire lab. You can take screenshots or just record your screen where you are showing all the resources you created with the correct configuration and the working API with valid responses.

# Additional references

- Root User Best Practices
- Root User Tasks
- What is cost management
- Creating a budget
- AWS CLI Installation
- Amazon S3
- Boto3
- Amazon ECR
- Docker Hub
- What is AWS VPC?
- Subnets for you VPC
- CIDR Reference
- Enable VPC internet access using internet gateways
- Configure route tables
- NAT gateways
- Control traffic to your AWS resources using security groups
- AWS Load Balancer Behind the Scenes
- AWS Fargate Getting Started