AWS CLI Setup and Configuration Guide

1. Installing AWS CLI on Mac M3

To install AWS CLI on Mac M3:

Step 1: Download and Install AWS CLI

Open the **Terminal** and run the following commands:

```
curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o
"AWSCLIV2.pkg"
sudo installer -pkg AWSCLIV2.pkg -target /
```

Step 2: Verify AWS CLI Installation

Run the following command to check if AWS CLI is installed:

```
aws --version
```

Expected output:

```
aws-cli/2.x.x Python/3.x.x Darwin/x86_64
```

If you see this, AWS CLI is installed successfully.

2. Setting Up AWS IAM User and Access Keys

We need to create an **IAM User** and generate access keys to configure AWS CLI.

Step 1: Create an IAM User

Go to **AWS Console** → Open **IAM (Identity and Access Management)**.
Click on **Users** (left panel) → **Add user**.
Name the user (e.g., `aws-cli-user`).
Select **Programmatic Access** to enable CLI usage.
Click **Next**.

Step 2: Attach Policies to IAM User

To allow full access to AWS services, attach the following policies:

Select "**Attach policies directly**".
Search for and select:

`AdministratorAccess` (Full AWS access) **OR**

`AmazonS3FullAccess`
`AmazonRDSFullAccess`
`AmazonSQSFullAccess`

Click **Next** → **Add Permissions**.

Step 3: Generate Access Key and Secret Key

Click on the created **IAM User**.
Navigate to **Security credentials**.
Scroll down to **Access Keys** → Click **Create Access Key**.
Select **"Command Line Interface (CLI)"**.
Click **Next** → **Create Access Key**.
Download the .CSV file (contains **Access Key ID** & **Secret Access Key**).

3. Configuring AWS CLI

Now, configure AWS CLI with your IAM user credentials.

Run the following command in **Terminal**:

```
aws configure
```

It will prompt for the following details:

AWS Access Key ID → Paste from `.csv` file
AWS Secret Access Key → Paste from `.csv` file
Default Region Name → `us-east-1` (or your AWS region)
Default Output Format → `json` (or leave blank)

Verify AWS CLI Connection

Run the command:

```
aws s3 ls
```

If it lists buckets (or shows an empty list), AWS CLI is **configured successfully!**
If you get an `AccessDenied` error, ensure IAM permissions are set correctly.

4. Setting Up AWS Lambda

Step 1: Create an AWS Lambda Function (Using AWS Console)

Go to **AWS Console** → Open **Lambda.**
Click "**Create Function**" (Top Right Corner).
Choose "**Author from Scratch**".
Function Name: `hello-world-lambda`
Runtime: Select **Python 3.11** (or latest version).
Permissions: Choose "**Create a new role with basic Lambda permissions**".
Click "**Create Function**".

Step 2: Deploy Lambda Using AWS CLI

Create a local directory: `mkdir lambda_function && cd lambda_function`

Create a new Python file: `nano lambda_function.py`

Add this code: `import json`

```
def lambda_handler(event, context):
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from AWS Lambda deployed
via CLI!')
    }
```

Save the file and create a ZIP package: `zip function.zip lambda_function.py`

Deploy it using AWS CLI: `aws lambda create-function --function-name cli-hello-world \
--runtime python3.11 --role arn:aws:iam::YOUR_ACCOUNT_ID:role/YOUR_LAMBDA_ROLE \
--handler lambda_function.lambda_handler --zip-file fileb://function.zip`

Invoke and test: `aws lambda invoke --function-name cli-hello-world response.json
cat response.json`

AWS Lambda + RDS Integration

1. AWS CLI Setup & Configuration

Step 1: Install AWS CLI on Mac M3

```
curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o
"AWSCLIV2.pkg"

sudo installer -pkg AWSCLIV2.pkg -target /
```

Step 2: Verify AWS CLI Installation

```
aws --version
```
Expected Output:

```
aws-cli/2.x.x Python/3.x.x Darwin/x86_64
```

Step 3: Configure AWS CLI with IAM User Credentials

```
aws configure
```

Provide the following details:

- **AWS Access Key ID**
- **AWS Secret Access Key**
- **Default Region Name** (e.g., us-east-1)
- **Default Output Format** (json)

Verify AWS Connection:

```
aws s3 ls
```

2. AWS RDS Setup

Step 1: Create a VPC

```
aws ec2 create-vpc --cidr-block 10.0.0.0/16 --query
"Vpc.VpcId" --output text
```

Output:

```
vpc-05f0ced5798300f57
```

Step 2: Create Subnets

```
aws ec2 create-subnet --vpc-id vpc-05f0ced5798300f57 --
cidr-block 10.0.1.0/24 --availability-zone us-east-1a --
query "Subnet.SubnetId" --output text

aws ec2 create-subnet --vpc-id vpc-05f0ced5798300f57 --
cidr-block 10.0.2.0/24 --availability-zone us-east-1b --
query "Subnet.SubnetId" --output text
```

Step 3: Create an Internet Gateway and Attach to VPC

```
aws ec2 create-internet-gateway --query
"InternetGateway.InternetGatewayId" --output text

aws ec2 attach-internet-gateway --internet-gateway-id
igw-0cf189ca83d68898c --vpc-id vpc-05f0ced5798300f57
```

Step 4: Create and Associate Route Table

```
aws ec2 create-route-table --vpc-id vpc-05f0ced5798300f57
--query "RouteTable.RouteTableId" --output text

aws ec2 associate-route-table --route-table-id
rtb-0ac2b9d4f331c8152 --subnet-id subnet-0c25331abb5a2e7ec
```

```
aws ec2 associate-route-table --route-table-id
rtb-0ac2b9d4f331c8152 --subnet-id subnet-0d2795451f88620df
```

Step 5: Create Security Group for RDS

```
aws ec2 create-security-group --group-name my-lambda-db-sg
--description "Security group for Lambda RDS" --vpc-id
vpc-05f0ced5798300f57 --query "GroupId" --output text
```

Allow MySQL Access:

```
aws ec2 authorize-security-group-ingress --group-id
sg-0fd9db6bd7f5fa641 --protocol tcp --port 3306 --cidr
YOUR_IP/32
```

Step 6: Create an RDS MySQL Database

```
aws rds create-db-instance --db-instance-identifier my-
lambda-db --db-instance-class db.t3.micro --engine mysql --
allocated-storage 20 --master-username admin --master-user-
password MySecurePassword123 --vpc-security-group-ids
sg-0fd9db6bd7f5fa641 --db-subnet-group-name my-rds-subnet-
group --backup-retention-period 7 --storage-type gp2 --
publicly-accessible
```

Wait until the database status becomes "available":

```
aws rds describe-db-instances --query "DBInstances[*].
[DBInstanceIdentifier,DBInstanceStatus]"
```

Retrieve the RDS Endpoint:

```
aws rds describe-db-instances --query "DBInstances[*].
[Endpoint.Address]"
```

3. AWS Lambda Setup & RDS Integration

Step 1: Create Lambda Function

```
aws lambda create-function --function-name lambda-rds-
integration --runtime python3.11 --role

arn:aws:iam::710271938127:role/service-role/hello-world-
lambda-role-yrsenrbk --handler

lambda_function.lambda_handler --zip-file fileb://
function.zip
```

Step 2: Allow Lambda to Access RDS

```
aws iam attach-role-policy --role-name hello-world-lambda-
role-yrsenrbk --policy-arn arn:aws:iam::aws:policy/service-
role/AWSLambdaVPCAccessExecutionRole
```

Update Lambda Function Configuration:

```
aws lambda update-function-configuration --function-name
lambda-rds-integration --timeout 15
```

Test Lambda Execution:

```
aws lambda invoke --function-name lambda-rds-integration
response.json
```

4. API Gateway Integration

Step 1: Create API Gateway

```
aws apigateway create-rest-api --name "LambdaRDSAPI" --
query "id" --output text
```

Retrieve API Resource ID:

```
aws apigateway get-resources --rest-api-id d9jcu0kxek --
query "items[*].[id]" --output text
```

Create API Resource:

```
aws apigateway create-resource --rest-api-id d9jcu0kxek --
parent-id fxhvn8lu3l --path-part users --query "id" --
output text
```

Define GET Method:

```
aws apigateway put-method --rest-api-id d9jcu0kxek --
resource-id 84zb1b --http-method GET --authorization-type
"NONE"
```

Integrate with Lambda:

```
aws apigateway put-integration --rest-api-id d9jcu0kxek --
resource-id 84zb1b --http-method GET --type AWS_PROXY --
integration-http-method POST --uri arn:aws:apigateway:us-
east-1:lambda:path/2015-03-31/functions/arn:aws:lambda:us-
east-1:710271938127:function:lambda-rds-integration/
invocations
```

Deploy API Gateway:

```
aws apigateway create-deployment --rest-api-id d9jcu0kxek
--stage-name prod --query "id" --output text
```

Test API:

```
curl -X GET "https://d9jcu0kxek.execute-api.us-
east-1.amazonaws.com/prod/users"
```

Obstacles Faced

1. **RDS Connection Issues**: Resolved by adjusting security group ingress rules.
2. **IAM Role Permissions**: Attached correct IAM policies for Lambda to access VPC.
3. **Subnet Group Assignment Errors**: Ensured correct VPC and subnets were used.