National College of Ireland

Configuration Manual

MSc Research Project
MSCLOUD

Nikhil Rajendra Puranik
Student ID: X22194771

School of Computing
National College of Ireland

Supervisor: Rashid Mijumbi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Nikhil Rajendra Puranik
Student ID:	X22194771
Programme:	MSCCLOUD **Year:** 2023-2024
Module:	MSCCLOUD Research Project
Lecturer:	Rashid Mijumbi
Submission Due Date:	03-01-2025
Project Title:	Image Security in Cloud using hybrid Compression and Encryption Technique.
Word Count:	
Page Count:	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Nikhil Rajendra Puranik

Date: 03-01-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Nikhil Rajendra Puranik
Student ID: X22194771

1 AWS S3 Bucket Setup

1. Log in to AWS Management Console

- Open the AWS Management Console and sign in with your AWS credentials.

2. Navigate to S3 Dashboard

- From the console dashboard, click on "Services" and then select "S3" under the "Storage" section.

3. Create Two S3 Buckets

Create two S3 buckets for your project:

1. Input Bucket (for uploading raw images).
2. Processed Bucket (for storing meta-embedded images).

3.1. Create Input S3 Bucket

1. Click the "Create bucket" button at the top right of the S3 Dashboard.
2. Bucket Name: Enter a unique name for the input bucket (e.g. *source-image-bucket-folder*).
3. Region: Choose a region closest to user base or other resources.
4. Bucket Settings:
    - Block all public access should be enabled to keep the bucket private.
    - Leave all other settings as default.

3.2. Create Processed S3 Bucket

1. Repeat the process above to create another bucket for storing processed images.
2. Bucket Name: Enter a unique name for the processed bucket (e.g., *processed-images-bucket-folder*).
3. Ensure the same settings as the input bucket and access permissions.

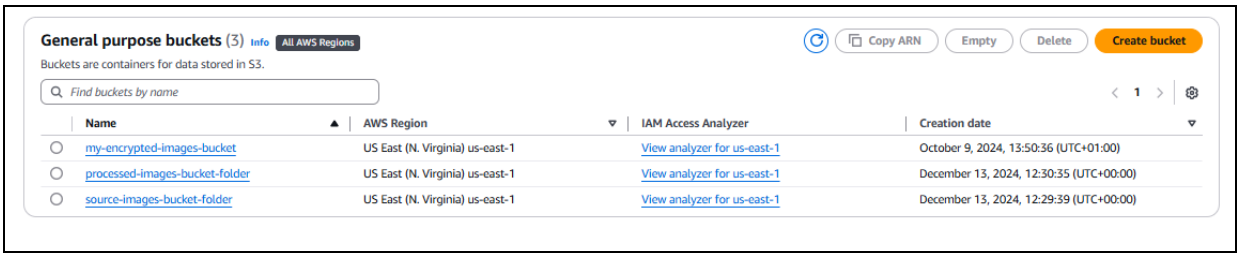

Fig.1 S3 buckets

4. Set Up Event Notification (Lambda Test Preparation)

1. **Configure Event Notification on Input Bucket**:
    - Go to the Properties tab of the input S3 bucket *source-image-bucket-folder*.
    - Scroll to the Event notifications section and click Create event notification.
    - Event Name: Enter a name like image-upload-trigger.
    - Event Type: Choose “All object create events” to trigger the event whenever a new image is uploaded to the bucket.
    - Destination: Leave this empty for now, as the Lambda function will be configured later.

Fig.2 Event notification

5. Set Bucket Policies and Permissions

1. **Set Bucket Policy for Input Bucket**:
    - Ensure that the input bucket *my-encrypted-images-bucket* is ready to be accessed by Lambda once it’s configured.
    - For now, ensure the bucket has permissions for the Lambda function to **“**read**”** images when the trigger is set up later.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
```

```
      ],
      "Resource": "arn:aws:s3:::my-encrypted-images-bucket/*"
    }
  ]
}
```

2. **Set Bucket Policy for Processed Bucket**:
    - Similarly, ensure that the processed bucket *processed-images-bucket-folder* has permissions for Lambda to "**write"** the processed files.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::processed-images-bucket-folder/*"
    }
  ]
}
```

6. Review and Test S3 Setup

1. Verify Bucket Creation: Navigate to the S3 Dashboard and confirm that both the input and processed buckets have been created successfully.
2. Upload a Test Image: Upload a test image to the input S3 bucket (my-encrypted-images-bucket) to prepare for Lambda testing.

2 AWS Lambda Setup

1. Navigate to Lambda Dashboard

- From the AWS Management Console, click on "Services" and then select "Lambda" under the "Compute" section.

2. Create a New Lambda Function

1. Click the "Create function" button.
2. Select "Author from scratch".
3. Function Name: Enter a descriptive name for your function, e.g., *lamda_function_1*
4. Runtime: Select Python 3.x as the runtime for your Lambda function.

5. Role: Choose "Use an existing role" and select the IAM role that grants the Lambda function permission to access the necessary AWS services of S3 and CloudWatch.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:HeadObject"
      ],
      "Resource": [
        "arn:aws:s3:::my-encrypted-images-bucket/*",
        "arn:aws:s3:::processed-images-bucket-folder/*"
      ]
    }
  ]
}
```

6. **Configure Lambda Function Timeout and Memory**:
    - Timeout: Set an appropriate timeout (e.g., 5 minutes) based on the expected image processing time.
    - Memory: Set the memory to 512MB for processing.

3. Upload Lambda Function Code

1. Download the code from Github "*https://github.com/nikhilpuranik97/cloud_image_storage.git* "and unzip.
2. Prepare the Lambda function code and dependencies (such as pycryptodome,pillow).
3. **Package the code**:
    1. Include function code (lambda_function.py and libraries in a ZIP file.
    2. Make sure to include the necessary dependencies in the ZIP (e.g., pycryptodome, boto3, Pillow).
4. **Upload the code**:
    1. Under the "Function code" section, select "Upload a .zip file" and upload the ZIP file containing your Lambda function code.
5. If additional dependencies (e.g., pycryptodome, Pillow) are required, install them using pip and include them in the ZIP file.

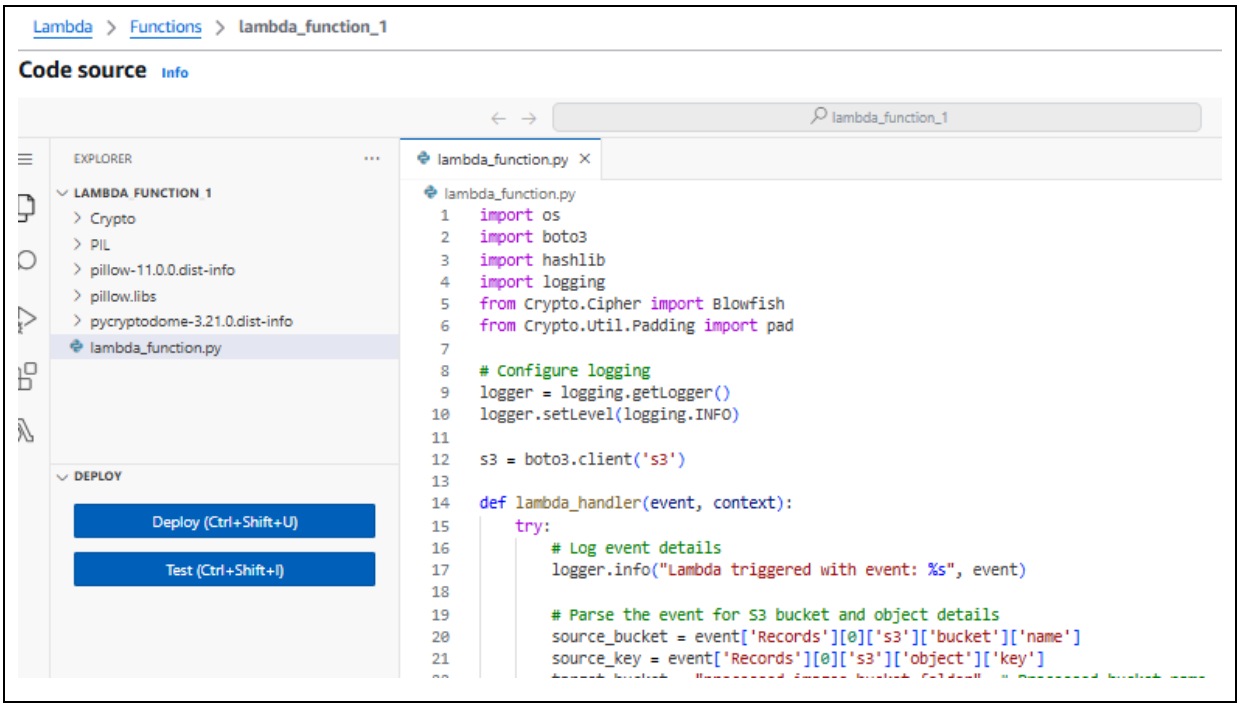

Fig.3 Lamda function code

5. Set Lambda Trigger (S3 Bucket Event)

1. After creating the Lambda function, click "Add trigger" to set up an event source for triggering the Lambda function when a new image is uploaded to your S3 bucket.
2. **Configure the S3 Trigger**:
    - Trigger Type: Choose S3 from the available options.
    - Bucket Name: Select the S3 bucket where images are uploaded (e.g., my-encrypted-images-bucket).
    - Event Type: Choose "All object create events" to trigger the Lambda whenever an image is uploaded.
    - Prefix and Suffix: You can leave this as blank or specify if you want to filter specific objects (e.g., .png files).
    - Destination: Select the Lambda function you just created (e.g., image-processing-lambda).

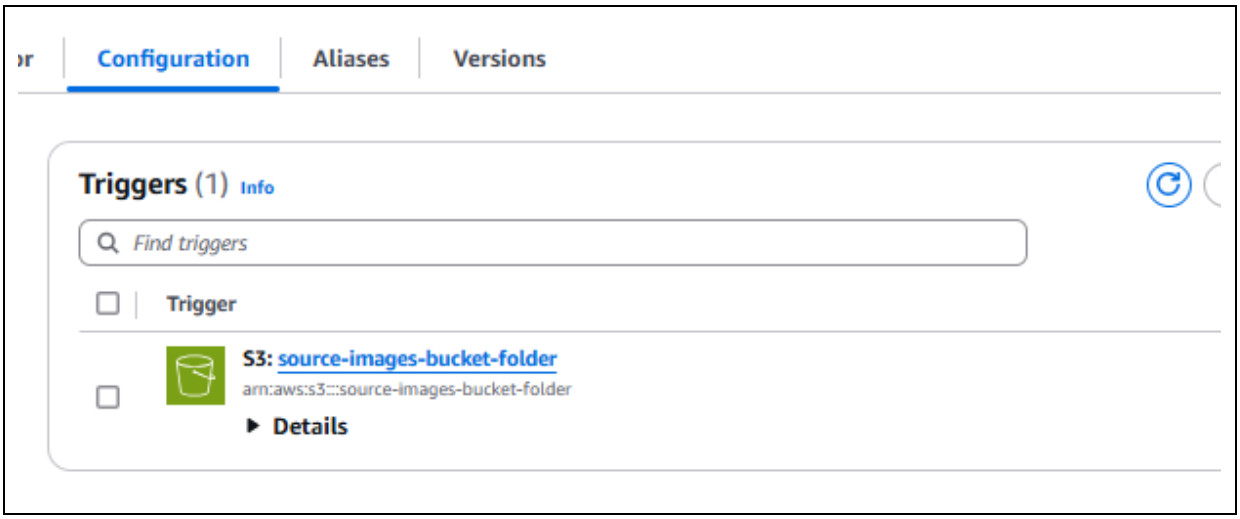

Fig.4 Lamda trigger

6. Test the Lambda Function

1. **Upload a test image** to your S3 bucket (e.g., *source-image-bucket-folder*).
2. **Check Lambda Logs**:
    - Go to CloudWatch Logs to verify that the Lambda function was triggered successfully when the image was uploaded.

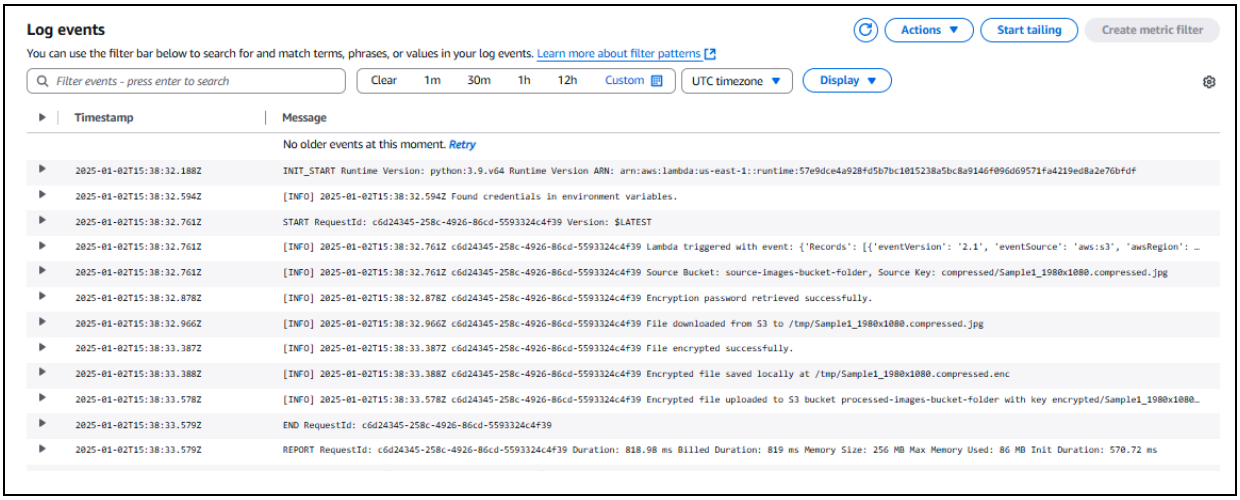

Fig. 5 Log Events in cloud watch

- Look for logs related to downloading the file, encryption, uploading the encrypted file, and metadata. The logs should include success messages like "File encrypted successfully" and "Encrypted file uploaded to S3".

7. Monitor Lambda Function

- Use CloudWatch Logs to monitor the performance and execution of the Lambda function.

3 AWS EC2 Instance Setup

1. **Navigate to EC2 Dashboard**:
   From the console dashboard, click on "Services" and then select "EC2" under the "Compute" section.
2. **Launch an EC2 Instance**:
   - Click the Launch Instance button at the top right of the EC2 Dashboard.
   - Choose an Amazon Machine Image (AMI): In the "Choose an Amazon Machine Image (AMI)" section, search for "Ubuntu 22.04" or select another Linux-based AMI.
3. **Choose Instance Type**:
   - Select t3.medium as the instance type. This instance type provides a balanced combination of compute, memory, and network resources, suitable for running applications like your image processing system.
4. **Select a Key Pair**:
   - Choose an existing key pair or create a new one. This key will be used to securely connect to your EC2 instance.
     - Download the key pair file (.pem file) and keep it safe.
     - Confirm access to the key pair file, as you'll need it to SSH into the instance.
5. **Configure Instance Details**:
   - Click "Next: Configure Instance Details".
   - Configure the instance as needed. For basic setup, the default settings are usually sufficient.
6. **Add Storage**:
   - Click "Next: Add Storage".
   - Specify the storage size and type. For example, set 30 GB as the root storage for the instance.
7. **Add Tags**:
   - Click "Next: Add Tags".
   - Add key-value pairs to tag your instance. Tags help manage and identify your resources. For example:
     - **Key**: Name, **Value**: ImageProcessingInstance.

Fig.6 Configuring instance

8. **Configure Security Group**:
    - Click **"Next: Configure Security Group"**.
    - Configure the security group to allow SSH (port 22) access and HTTP (port 8501) if your application uses it.
        - **SSH**: For secure access to the instance, open port 22 for SSH.
        - **HTTP**: If you're running a web server (like Streamlit), open port 8501 for HTTP access.

Fig.7 Configuring security group

9. **Review and Launch**:
    - Review instance configuration.
    - Click "Launch" to create the EC2 instance.
    - Select the key pair created or uploaded earlier and acknowledge that have access to it.

10. **Connect to Your EC2 Instance**:
    Once the instance is launched, select it from the EC2 Dashboard and click on the Connect button.

```
ssh -i /path/to/your-key.pem ubuntu@your-instance-public-ip
```

- Use SSH to connect to your EC2 instance:

  - Make sure to replace /path/to/your-key.pem with the actual path to private key and your-instance-public-ip with the public IP of your instance.

9. **Clone the GitHub Repository:**
   - Install Git (if not already installed):

```
sudo apt install git
```

   - Clone project repository from GitHub:

```
git clone https://github.com/nikhilpuranik97/cloud_image_storage.git
```

   - Navigate to the project directory:

```
cd cloud-image-storage
```

   - Install Dependencies from requirements.txt:

```
pip3 install -r requirements.txt
```

10.Verify Instance Configuration:

   - After installation, test the instance to ensure it is working as expected. For example, if running Streamlit, start the application:

```
python3 -m streamlit run app.py
```

   - Open a browser and access the app via the public IP of the EC2 instance on the configured port (e.g., *http://your-instance-public-ip:8501*).

4 Locally Setup

To setup application locally follow the process till step 2 AWS lamda setup.

1. Install Visual Studio Code

If Visual Studio Code (VS Code) not installed:

1. Go to the official Visual Studio Code download page.
2. Download the appropriate installer for your operating system (Windows, macOS, or Linux).

2. Install Git

Git is required for cloning the repository and version control:

1. Visit the official Git download page and download the installer for operating system.
2. Install Git and follow the default installation prompts.

3. Clone the GitHub Repository

Need to clone the project repository from GitHub:

1. Open a terminal (or use the **VS Code terminal**).
2. Navigate to the folder where you want to store your project:

```
cd /path/to/your/project/directory
```

Clone the project repository:

```
git clone https://github.com/nikhilpuranik97/cloud_image_storage.git
```

Once the repository is cloned, navigate to the project directory:

```
cd cloud-image-storage
```

3. **Set Up Virtual Environment**

In project directory (cloud-image-storage), create a virtual environment:

```
python -m venv venv
```

Activate the virtual environment:

```
.\venv\Scripts\activate
```

5. Install Required Dependencies

The project dependencies should be listed in the requirements.txt file. To install them:

1. Run the following command to install all dependencies:

```
pip install -r requirements.txt
```

6. Configure AWS Credentials

For local testing and interacting with AWS services (S3), need to set up AWS credentials:

1. Install the AWS CLI:

```
pip install awscli
```

2. Configure your AWS credentials by running:

```
aws configure
```

Need to provide:

- AWS Access Key ID
- AWS Secret Access Key
- Default region name
- Default output format

7. Run the Application Locally

Now that everything is set up, run your application locally using Streamlit:

1. In the terminal, run the following command to start application:
   streamlit run app.py
2. Open browser and navigate to the URL (*http://localhost:8501*) to view the app running locally

Fig.7 Application UI.

Fig.8 Application result.

References

Amazon Web Services (AWS) (2025). *AWS S3 Documentation*. Available at: https://aws.amazon.com/console/ (Accessed: 3 January 2025).

Visual Studio Code (n.d.). *Download Visual Studio Code*. Available at: https://code.visualstudio.com/ (Accessed: 3 January 2025).

Amazon Web Services (AWS) (n.d.). *Amazon EC2 Documentation*. Available at: https://docs.aws.amazon.com/ec2/ (Accessed: 2 January 2025).

Amazon Web Services (AWS) (n.d.). *AWS Lambda Documentation*. Available at: https://docs.aws.amazon.com/lambda/ (Accessed: 2 January 2025).

Streamlit (n.d.). *Deploy Streamlit Applications*. Available at: https://docs.streamlit.io/streamlit-cloud (Accessed: 2 January 2025).

Amazon Web Services (AWS) (n.d.). *Deploying Streamlit with AWS EC2*. Available at: https://aws.amazon.com/blogs/machine-learning/deploying-streamlit-applications-on-aws/ (Accessed: 2 January 2025).