

# CLOUD COMPUTING PROJECT REPORT

FILE-SHARING WEB APPLICATION

☀️ UPLOAD YOUR FILES, LET THEM SOAR IN THE CLOUD ☀️



## Team Members:

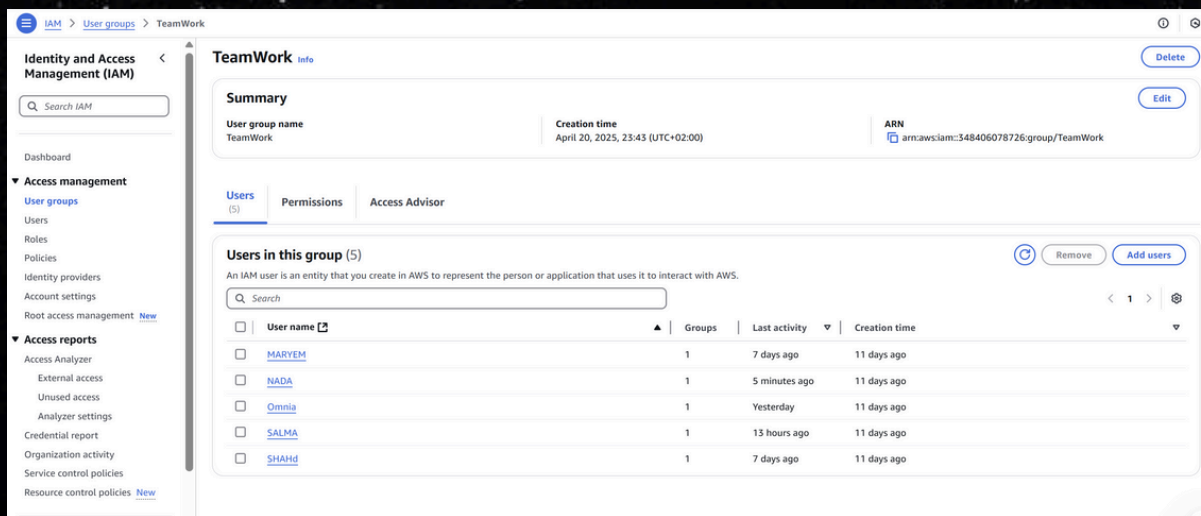
NAME	Roles
Nada Fayez	VPC and Networking
Nouran Hasan	IAM account setup & Billing alerts
Mariam Ramadan	EC2 Server
Omnia mohamed	S3 storage
Shahd Nezar	S3 storage
Salma Ahmed	Web App development

# Project Overview:

We developed a cloud-based web application seamlessly integrated with Amazon EC2 to ensure 24/7 accessibility. Through this platform, users can effortlessly upload their files, which are then securely stored in Amazon S3. The entire system operates within a dedicated network, providing a safe, reliable, and protected environment for data storage and access

## 1- IAM Account & Billing alerts:

We used IAM accounts instead of the root account to enhance security and provide complete control over access privileges. With IAM, we can define precise permissions for each user or application individually allowing them to only have the permissions they need without granting excessive permissions that could compromise the system



First, we created a group called "Team Work," then created accounts for all users working on the project. This way, we can manage the permissions of each team member separately and precisely, enhancing security and facilitating monitoring of activities



# IAM Policies

**TeamWork** Info Delete

**Summary** Edit

User group name: TeamWork | Creation time: April 20, 2025, 23:43 (UTC+02:00) | ARN: arn:aws:iam::348406078726:group/TeamWork

Users (5) | **Permissions** | Access Advisor

**Permissions policies (3)** Info Simulate Remove Add permissions

You can attach up to 10 managed policies.

Filter by Type: All types

Policy name	Type	Attached entities
<a href="#">AdministratorAccess</a>	AWS managed - job function	3
<a href="#">AmazonEC2ContainerRegistryFullAccess</a>	AWS managed	1
<a href="#">AmazonS3FullAccess</a>	AWS managed	6

We have set the following policies:

AdministratorAccess & AmazonEC2ContainerRegistryFullAccess, and AmazonS3FullAccess  
These are AWS-managed policies, similar to the policies AWS provides to users by default

Administrator Access: This policy grants full administrative privileges to all AWS resources

AmazonEC2ContainerRegistryFullAccess: This policy allows full access to Amazon ECR (a container storage service)

AmazonS3FullAccess: This policy allows full access to Amazon S3

# IAM Roles

**Roles (3)** Info Delete Create role

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Search

Role name	Trusted entities	Last activity
<a href="#">AWSServiceRoleForSupport</a>	AWS Service: support (Service-Linked Role)	-
<a href="#">AWSServiceRoleForTrustedAdvisor</a>	AWS Service: trustedadvisor (Service-Linked Role)	-
<a href="#">Maryem-EC2-S3-Role</a>	AWS Service: ec2	37 minutes ago

**Roles Anywhere** Info Manage

Authenticate your non AWS workloads and securely provide access to AWS services.

**Access AWS from your non AWS workloads**  
Operate your non AWS workloads using the same authentication and authorization strategy that you use within AWS.

**X.509 Standard**  
Use your own existing PKI infrastructure or use [AWS Certificate Manager Private Certificate Authority](#) to authenticate identities.

**Temporary credentials**  
Use temporary credentials with ease and benefit from the enhanced security they provide.

We created and used IAM Roles to define the permissions granted to different AWS services

**AWSServiceRoleForSupport**

This is a Service-Linked Role automatically created by AWS and used to support management services

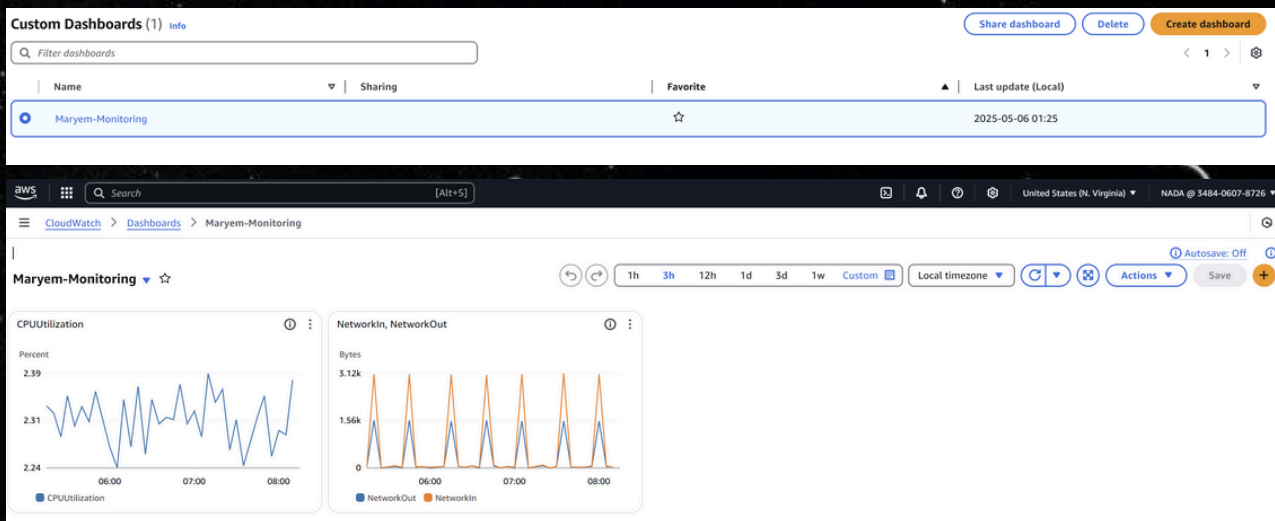
**AWSServiceRoleForTrustedAdvisor**

This is also a Service-Linked Role used by Trusted Advisor to improve performance and enhance security

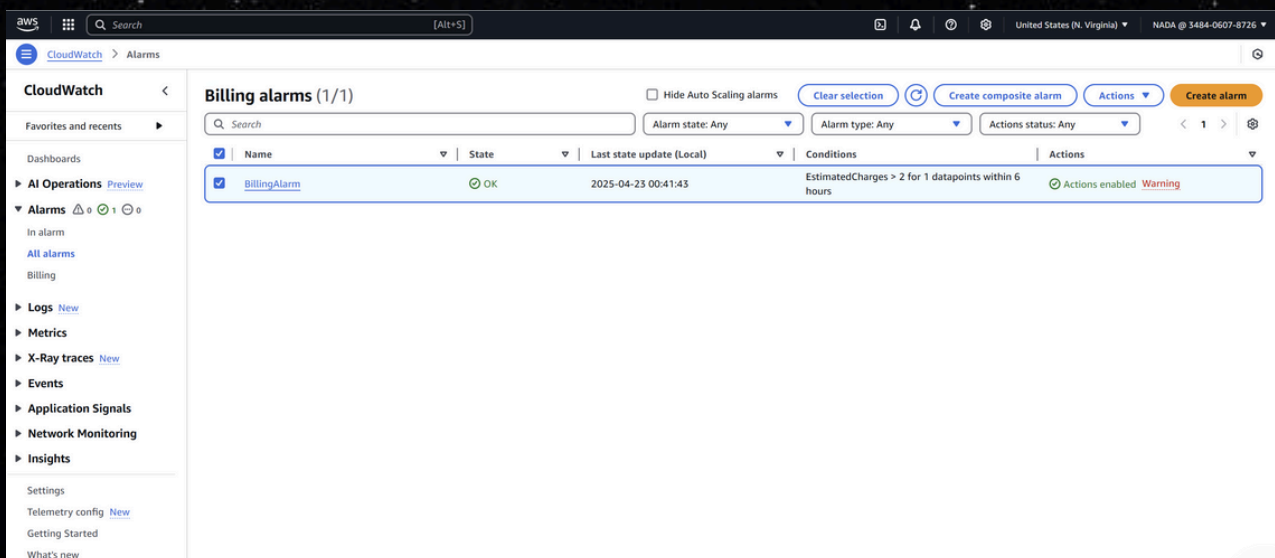
**Maryem-EC2-S3-Role**

This is a custom Role we created to allow EC2 to access the S3 service. This means that only the EC2 instance can use this role

# Billing Alarm



First, we created a dashboard to monitor server performance in real-time displaying vital indicators such as CPU usage and network in/out. This tool helps quickly detect any issues, such as abnormal data traffic. Thanks to this dashboard, actions can be taken to ensure server stability



Then we created a billing alarm using CloudWatch to track the estimated cost of our account on a regular basis



Details

Name

BillingAlarm

Type

Metric alarm

Description

No description

State

OK

Threshold

EstimatedCharges > 2 for 1 datapoints within 6 hours

Last state update

2025-04-23 00:41:43 (Local)

Actions

Actions enabled

Namespace

AWS/Billing

Metric name

EstimatedCharges

Currency

USD

Statistic

Maximum

Period

6 hours

Datapoints to alarm

1 out of 1

Missing data treatment

Treat missing data as missing

Percentiles with low samples

evaluate

ARN

arn:aws:cloudwatch:us-east-1:348406078726:alarm:BillingAlarm

► View EventBridge rule

DetailsTagsActionsHistoryParent alarms

Actions

Actions enabled

Type

▼

Description

▼

Config

▼

Notification

When in alarm, send message to topic "BillingAlarmTopic"

Configuration not verified

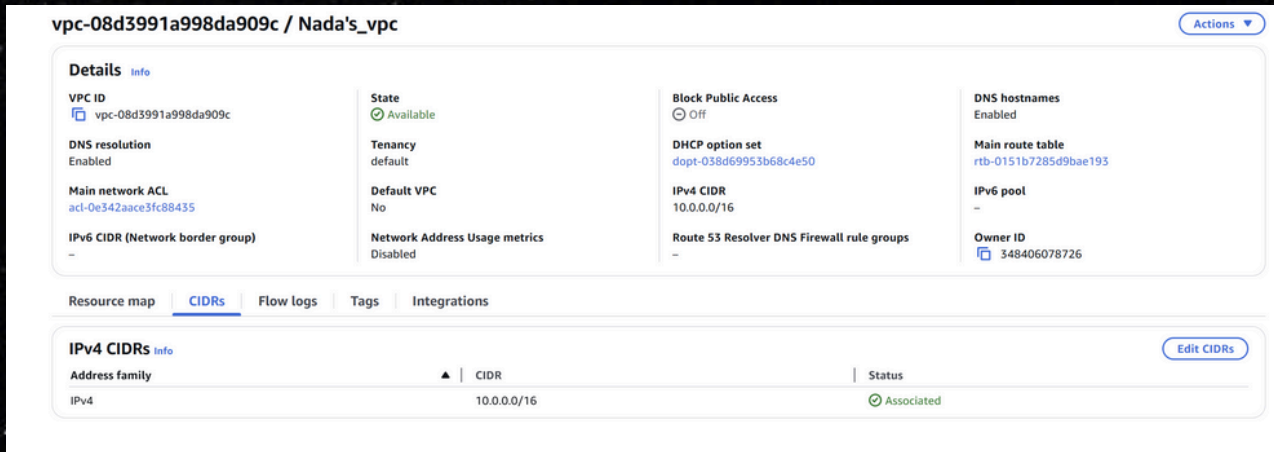
The alarm is set to trigger if the estimated cost exceeds \$2 within a 6 hour period, and an automatic notification is sent to the SNS Topic if this occurs. This step helps us maintain budget and control our project's cloud usage safely and effectively

---

## 2- VPC and Networking

### VPC

A VPC (Virtual Private Cloud) is an isolated virtual network within the AWS environment that serves as a private and secure network environment, allowing us to have complete control over the network configuration for applications, including IP addresses, network segmentation into subnets, firewall settings, and traffic routing.



The screenshot displays the AWS Management Console interface for a VPC named 'vpc-08d3991a998da909c / Nada's\_vpc'. The 'Details' tab is active, showing various configuration options. The VPC ID is 'vpc-08d3991a998da909c'. The state is 'Available'. DNS resolution is 'Enabled'. The main network ACL is 'acl-0e342aace3fc88435'. The IPv4 CIDR is '10.0.0.0/16'. The DHCP option set is 'dopt-038d69953b68c4e50'. The route 53 Resolver DNS Firewall rule groups are disabled. The DNS hostnames are 'Enabled'. The main route table is 'rtb-0151b7285d9bae193'. The IPv6 pool is empty. The owner ID is '348406078726'. Below the details, there are tabs for 'Resource map', 'CIDRs', 'Flow logs', 'Tags', and 'Integrations'. The 'CIDRs' tab is selected, showing a table with one entry: 'IPv4' with CIDR '10.0.0.0/16' and status 'Associated'.

vpc-08d3991a998da909c / Nada's_vpc			
<b>Details</b>			
VPC ID vpc-08d3991a998da909c	State Available	Block Public Access Off	DNS hostnames Enabled
DNS resolution Enabled	Tenancy default	DHCP option set dopt-038d69953b68c4e50	Main route table rtb-0151b7285d9bae193
Main network ACL acl-0e342aace3fc88435	Default VPC No	IPv4 CIDR 10.0.0.0/16	IPv6 pool -
IPv6 CIDR (Network border group) -	Network Address Usage metrics Disabled	Route 53 Resolver DNS Firewall rule groups -	Owner ID 348406078726

IPv4 CIDRs		
Address family	CIDR	Status
IPv4	10.0.0.0/16	Associated

First, I created an isolated virtual private network (VPC). I defined the network scope using IPv4 CIDR 10.0.0.0/16, providing up to 65,536 internal IP addresses. I also enabled DNS Resolution to translate domain names to IP addresses, enabling resources within the network to connect to the internet, and enabled DNS Hostnames to automatically provide internal DNS names for each instance within the network



## Subnet

A subnet is a part of a VPC that we use to organize the distribution of resources within the network, and separate sensitive resources (such as databases) from public resources (such as servers that receive users), to improve security and communication efficiency

The screenshot displays the AWS console interface for a subnet. The title bar shows 'subnet-09f146ba06596463e / Nada's\_Subnet' and an 'Actions' button. The main content area is divided into several sections:

- Details:** Subnet ID (subnet-09f146ba06596463e), IPv4 CIDR (10.0.2.0/24), Availability Zone (us-east-1a), Route table (rtb-001bda9635dd2994b | Nada's\_route\_table), Auto-assign IPv6 address (No), IPv4 CIDR reservations (None), Resource name DNS A record (Disabled).
- Subnet ARN:** arn:aws:ec2:us-east-1:348406078726:subnet/subnet-09f146ba06596463e
- Available IPv4 addresses:** 250
- Availability Zone ID:** use1-az4
- Network ACL:** acl-0e342aace3fc88435
- Auto-assign customer-owned IPv4 address:** No
- IPv6 CIDR reservations:** None
- Resource name DNS AAAA record:** Disabled
- State:** Available
- IPv6 CIDR:** None
- Network border group:** us-east-1
- Default subnet:** No
- Customer-owned IPv4 pool:** None
- IPv6-only:** No
- DNS64:** Disabled
- Block Public Access:** Off
- IPv6 CIDR association ID:** None
- VPC:** vpc-08d3991a998da909c | Nada's\_vpc
- Auto-assign public IPv4 address:** No
- Outpost ID:** None
- Hostname type:** IP name
- Owner:** 348406078726

For our subnet, we chose IPv4 CIDR 10.0.2.0/24, which allows approximately 256 IP addresses to be distributed to resources such as EC2 instances. The subnet is available for use and is linked to the created VPC (Nada's\_vpc). It is not available for public access by default, as "Block Public Access" is disabled in the network settings

## Route Table

The screenshot displays the AWS console interface for a route table. The title bar shows 'rtb-001bda9635dd2994b / Nada's\_route\_table' and an 'Actions' button. The main content area is divided into several sections:

- Details:** Route table ID (rtb-001bda9635dd2994b), VPC (vpc-08d3991a998da909c | Nada's\_vpc), Main (No), Owner ID (348406078726), Explicit subnet associations (subnet-09f146ba06596463e / Nada's\_Subnet), Edge associations (None).
- Routes (2):** A table with columns: Destination, Target, Status, and Propagated.

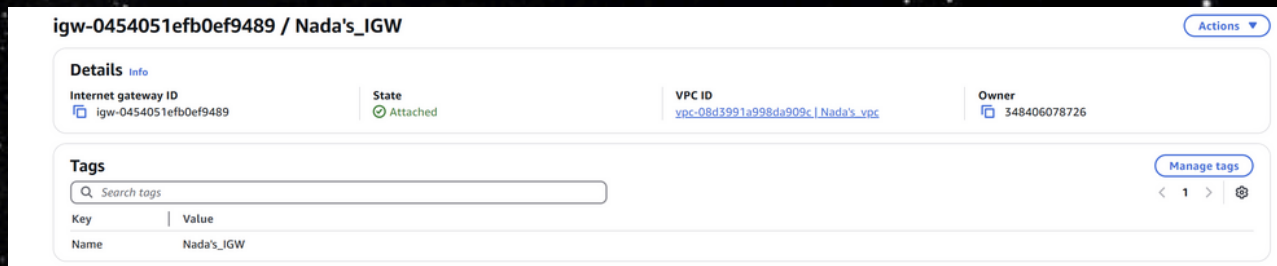
Destination	Target	Status	Propagated
0.0.0.0/0	igw-0454051efb0ef9489	Active	No
10.0.0.0/16	local	Active	No

It's a table that defines the paths for data traffic in and out of the VPC network. I linked it to our subnet, and it contains two rules: one for internal communication between network resources (10.0.0.0/16), and the other for routing internet connections through the Internet Gateway (0.0.0.0/0). This rule not only allows the server to go online, but also allows users to log in to it



## Internet Getaway

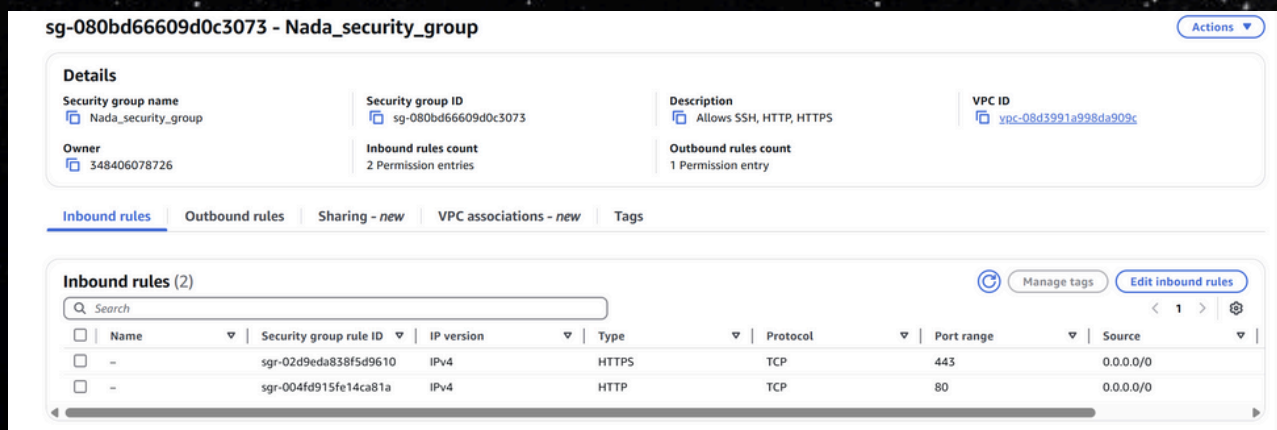
The Internet Gateway (IGW) is a core component of AWS that allows resources within a VPC to communicate with the internet



The IGW which I've named Nada's\_IGW is connected to our VPC Nada's\_vpc. It allows all resources within the VPC (such as EC2 instances) to access and exit the internet based on rules defined in the Route Table. So its primary function is to connect the VPC network to the internet

## Security Group

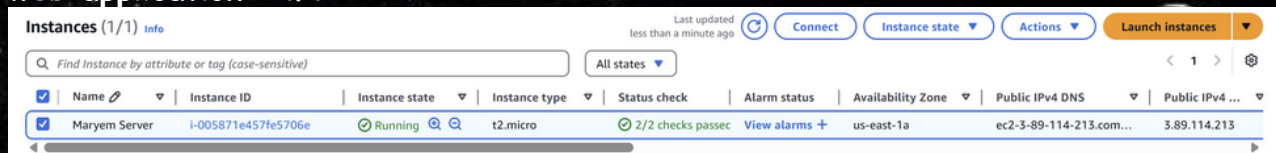
A Security Group is a set of security rules that defines the type of communication allowed between EC2 instances and other resources in a VPC. Nada\_security\_group is the Security Group we are using in this project



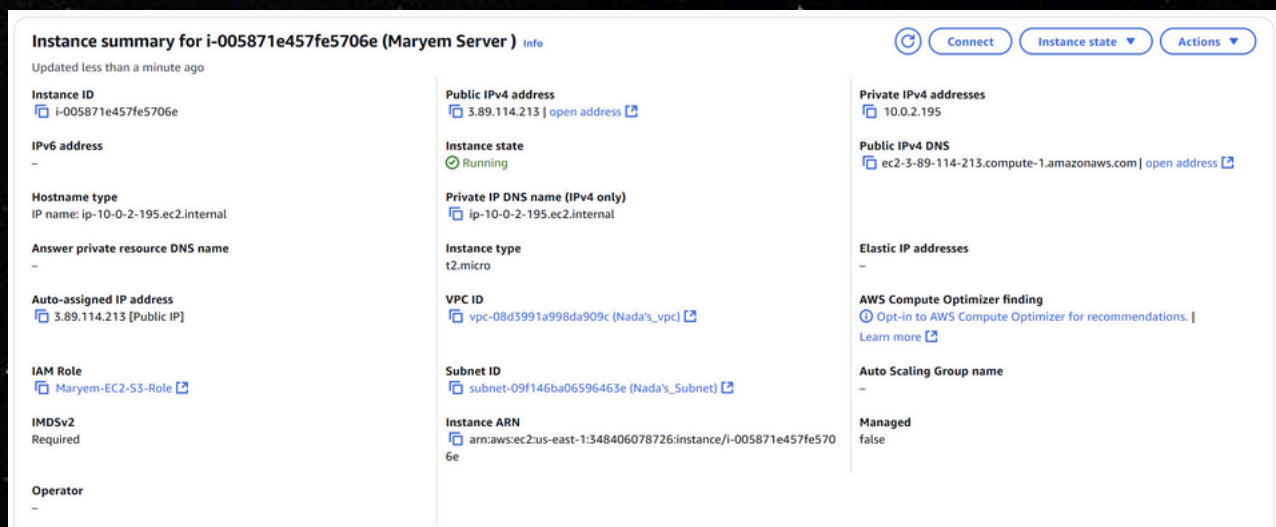
The two existing rules allow access to the server from anywhere in the world (0.0.0.0/0) via HTTP (port 80). HTTP is used for browsing websites and is an unencrypted protocol, meaning that data exchanged between the client and the server is sent in plain text. HTTPS is used for browsing secure websites and relies on the same HTTP protocol with an additional layer of encryption using SSL/TLS to ensure the protection of the exchanged data

### 3- EC2

Amazon EC2 is a service from AWS that provides virtual servers (instances) to run applications. It allows you to choose the type of server based on your processing, storage, and memory needs, with the flexibility to resize as needed. The service is pay-as-you-go and provides continuous hosting for your web application 24/7



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
Maryem Server	i-005871e457fe5706e	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1a	ec2-3-89-114-213.com...	3.89.114.213



Instance summary for i-005871e457fe5706e (Maryem Server)	
<b>Instance ID</b> i-005871e457fe5706e	<b>Public IPv4 address</b> 3.89.114.213   open address
<b>IPv6 address</b> -	<b>Instance state</b> Running
<b>Hostname type</b> IP name: ip-10-0-2-195.ec2.internal	<b>Private IP DNS name (IPv4 only)</b> ip-10-0-2-195.ec2.internal
<b>Answer private resource DNS name</b> -	<b>Instance type</b> t2.micro
<b>Auto-assigned IP address</b> 3.89.114.213 [Public IP]	<b>VPC ID</b> vpc-08d3991a998da909c (Nada's_vpc)
<b>IAM Role</b> Maryem-EC2-S3-Role	<b>Subnet ID</b> subnet-09f146ba06596463e (Nada's_Subnet)
<b>IMDSv2</b> Required	<b>Instance ARN</b> arn:aws:ec2:us-east-1:348406078726:instance/i-005871e457fe5706e
<b>Operator</b> -	<b>Managed</b> false

Details

Status and alarms

Monitoring

Security

Networking

Storage

Tags

▼ Networking details

Info

Public IPv4 address

3.89.114.213 | [open address](#)

Public IPv4 DNS

ec2-3-89-114-213.compute-1.amazonaws.com | [open address](#)

Subnet ID

subnet-09f146ba06596463e (Nada's\_Subnet)

Availability zone

us-east-1a

Use RBN as guest OS hostname

Disabled

Private IPv4 addresses

10.0.2.195

Private IP DNS name (IPv4 only)

ip-10-0-2-195.ec2.internal

IPv6 addresses

-

Carrier IP addresses (ephemeral)

-

Answer RBN DNS hostname IPv4

Disabled

VPC ID

vpc-08d3991a998da909c (Nada's\_vpc)

Secondary private IPv4 addresses

-

Outpost ID

-

▼ Network Interfaces (1)

Info

Q

Filter network interfaces

Interface ID

Device index

Card index

Description

Public IPv4 address

Private IPv4 address

Private IPv4 DNS

IPv6 addresses

Pr

eni-0528464cd8f961423

0

0

-

3.89.114.213

10.0.2.195

ip-10-0-2-195.ec2.in...

-

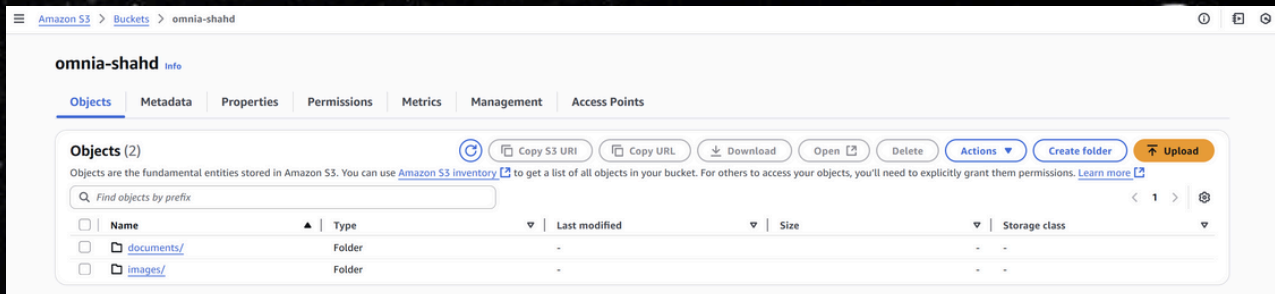
-

The EC2 instance "Maryem Server" is a t2.micro server with a public IP address (3.89.114.213) and a private IP address (10.0.2.195) inside the VPC "Nada's\_vpc" and subnet "Nada's\_Subnet". The server is associated with the IAM role "Maryem-EC2-S3-Role" which allows it to access the S3 service. In the security details, the security group "Nada\_security\_group" is assigned to the server we create



## 4- S3 Storage

Amazon S3 is one of the most important cloud storage services provided by AWS, as it allows files and data to be stored securely and easily accessible over the Internet



An S3 Bucket named "omnia-shahd" was created in AWS to store the required files and data. The bucket's settings were configured, with the "Block all public access" option disabled, to allow access via EC2 and its associated IAM role. The bucket has internal file organization by creating folders such as Documents, Photos, and Videos to facilitate data management.



A container-specific policy was defined by navigating to the Permissions tab in the container settings, where a policy was added that allows access to upload and download files during the test period. This policy ensures flexible ways to interact with the data stored within the container.

## 5- Web Application

### Backend

The backend is the part responsible for processing data and performing operations in the background

```
app = Flask(__name__)
CORS(app, resources={r"/": {"origins": "*"}}, supports_credentials=True)

s3 = boto3.client(
    's3',
    region_name='us-east-1',
    aws_access_key_id='AKIAVCHU63UDHOUANH46',
    aws_secret_access_key='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'
)

BUCKET_NAME = 'omnia-shahd'

def get_file_folder(filename):
    extension = os.path.splitext(filename)[1].lower()

    if extension in ['.jpg', '.jpeg', '.png', '.gif', '.bmp', '.webp']:
        return 'images/'
    elif extension in ['.pdf', '.doc', '.docx', '.xls', '.xlsx', '.ppt', '.pptx', '.txt']:
        return 'documents/'
    elif extension in ['.mp4', '.mov', '.avi', '.mkv', '.wmv']:
        return 'videos/'
    else:
        return 'others/'

@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return jsonify({'error': 'No file part'}), 400

    file = request.files['file']

    if file.filename == '':
        return jsonify({'error': 'No selected file'}), 400

    try:
        folder = get_file_folder(file.filename)
        filename = f"{folder}{uuid.uuid4().hex}_{file.filename}"

        s3.upload_fileobj(file, BUCKET_NAME, filename)

        file_url = f"https://{BUCKET_NAME}.s3.amazonaws.com/{filename}"

        return jsonify({
            'message': 'File uploaded successfully',
            'url': file_url,
            'folder': folder[:-1]
        }), 200

    except NoCredentialsError:
        return jsonify({'error': 'Credentials not available'}), 500
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

The application was built using Flask to power the web application, while the boto3 library interacts with AWS and stores files in an S3 Bucket. Files are uploaded using the upload\_fileobj function, which uploads the file to the Bucket using the specified file name and content type. After the file is successfully uploaded and stored, a download link is returned to the user to access the saved file

Pretty-print ☐

```
{"download_link": "https://omnia-shahd.s3.amazonaws.com/Salma-Ahmed-Mohamed-Tabana-FlowCV-Resume-20250428.pdf"}
```



## Frontend

The front end is the part that interacts with the user and displays data visually

```

<DOCTYPE html>
<html lang="en"
<head>
  </meta charset="UTF-8"
</title>Cloudy Files ✨</title>
<style>
  } html, body
  ;height: 100%
  ;margin: 0
;font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif
{
  } body
  ;background: linear-gradient(135deg, #667eea 0%, #764ba2 100%)
  ;display: flex
  ;justify-content: center
  ;align-items: center
  ;flex-direction: column
  ;overflow-x: hidden
  ;text-align: center
  ;position: relative
  {
    } body::before
    "" :content
    ;position: absolute
    ;top: 0
    ;left: 0
    ;width: 100%
    ;height: 100%
    ;background-size: cover
    ;background-position: center
    ;opacity: 0.3
    ;z-index: -1
  {
    } h1
    ;color: #fff
    ;text-shadow: 2px 2px 5px #00000000
    ;white-space: nowrap
    ;font-size: 2.5rem
    ;margin-bottom: 2rem
  {
    } form
    ;margin: 20px auto
    ;padding: 30px
    ;background-color: #ffffffee
    ;border-radius: 20px
    ;width: 350px
    ;box-shadow: 0 10px 30px #00000000
    ;backdrop-filter: blur(5px)
    ;border: 1px solid #00000000
  {
    } input[type="file"]
    ;margin-bottom: 15px
    ;width: 100%
  }
}

;animation: float 3s ease-in-out infinite
{
  } media (max-width: 768px)@
  {
    } form
    ;width: 80%
    ;padding: 20px
  {
    } h1
    ;font-size: 1.8rem
    ;white-space: normal
  {
    {
    }
  }
}

<h1> ✨ Upload your files, let them soar in the cloud ✨ </h1>

<form enctype="multipart/form-data" method="POST" action="http://127.0.0.1:5000/upload"
</input type="file" name="file" accept=".pdf,.jpg,.png,.txt,.doc,.docx,.xls,.xlsx" required><br>
<button type="submit">Upload to Cloud</button>
</form>

<div class="link" id="downloadLink"></div>

<script>
} document.querySelector('form').addEventListener('submit', function(e)
{
  ;alert('File is being uploaded to the cloud...')
  ;
})
</script>

```

The front-end is based on an HTML template that allows users to upload files of various types, such as .pdf, .jpg, .png, and .txt. Data is sent via a POST request to the server, and when the upload is complete, the server returns a download link to the file stored in S3, allowing the user to download the file directly.

