

SnapStream: AWS-Powered Media Upload and

Analysis Platform

Project Description:

At a time when digital content creation and media management are rapidly growing, organizations often struggle with storing, processing, and analyzing large volumes of multimedia data efficiently. To address these challenges, the development team created SnapStream — an AWS-Powered Media Upload and Analysis Platform designed to automate and streamline end-to-end media handling.

SnapStream leverages a Flask-based backend for managing user interactions and API integrations, while AWS EC2 provides scalable hosting for the web application. Uploaded media files — including images, audio, and video — are securely stored in Amazon S3, with each upload triggering AWS Lambda for serverless, event-driven processing. The platform integrates AI/ML services such as Amazon Rekognition, Transcribe, and Comprehend to automatically extract metadata, generate transcripts, and perform intelligent content analysis.

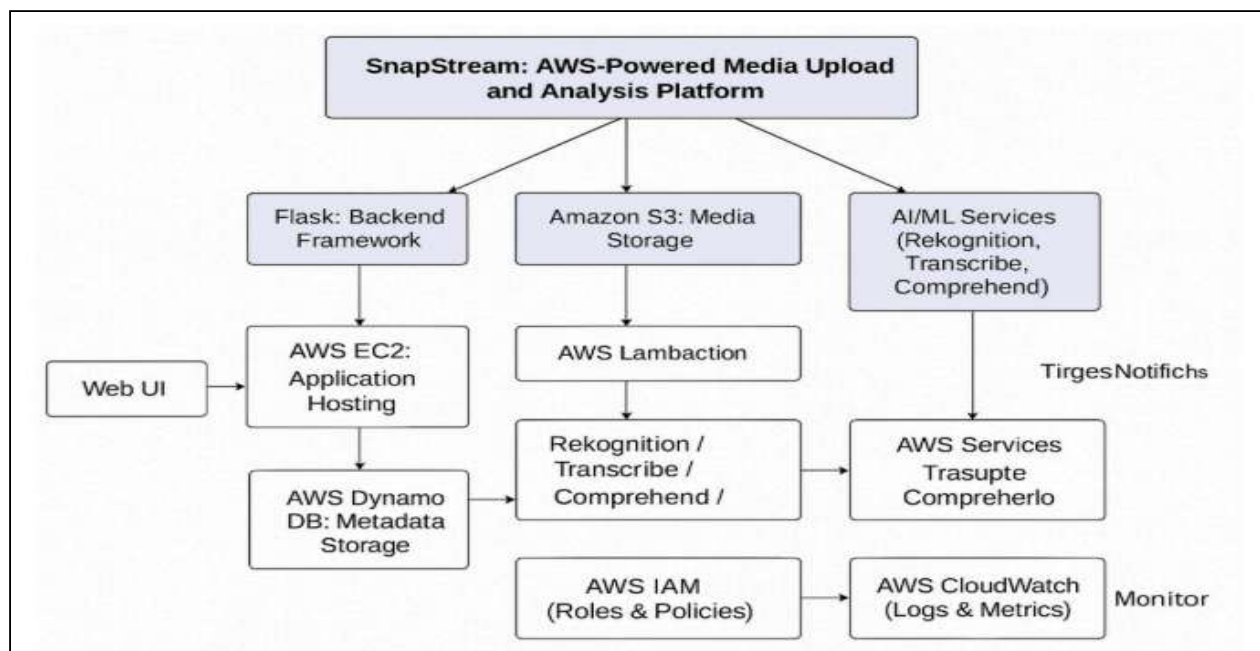
All metadata and analysis results are stored in AWS DynamoDB, ensuring fast, reliable, and schema-flexible data retrieval. AWS SNS provides real-time notifications to users about upload and processing status, while AWS IAM roles ensure secure and role-based access to all cloud resources.

Built with a responsive HTML/CSS frontend, SnapStream delivers a smooth and intuitive user experience, enabling users to upload, manage, and analyze their media effortlessly. This cloud-native platform provides a cost-effective, scalable, and intelligent solution for modern media workflows—reducing manual effort and empowering users with deeper insights into their multimedia content.

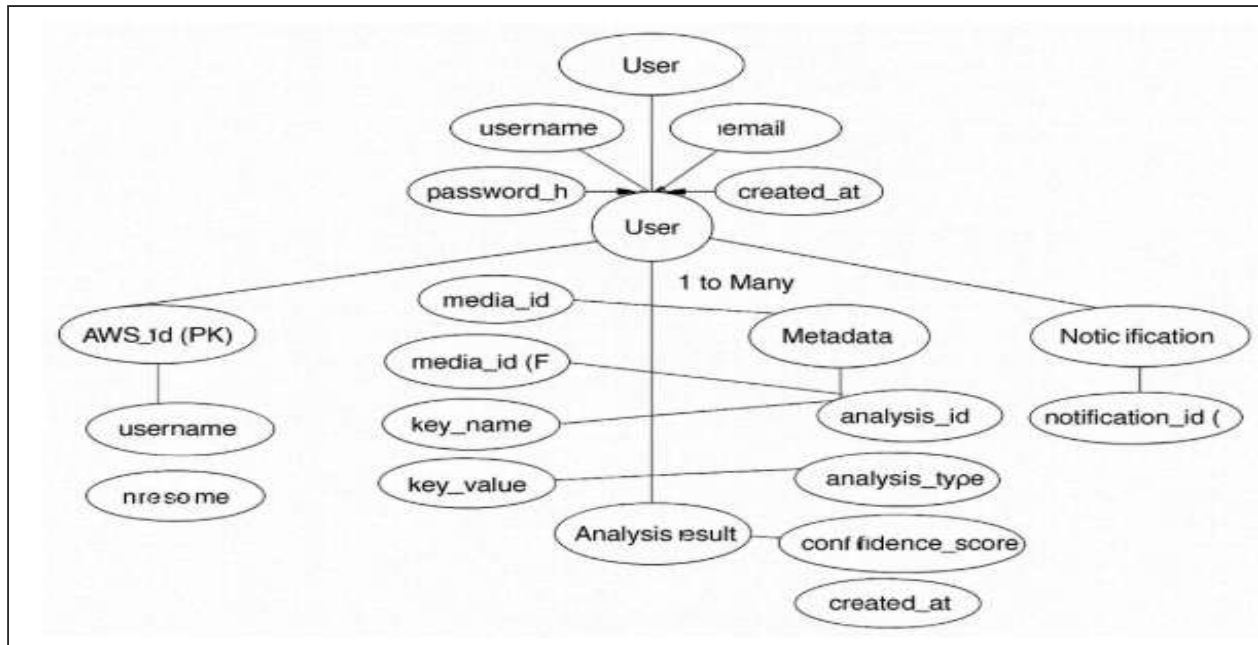
Scenario 1: Media Archiving for a News Agency

A leading digital news agency leverages SnapStream to manage its extensive library of daily news footage. Each time a journalist uploads a video, it is securely stored in Amazon S3, and an AWS Lambda function is automatically triggered to generate preview thumbnails for quick browsing. Using Amazon Rekognition, the platform detects key visual elements such as people, logos, and text appearing in the footage. This automated tagging system allows reporters to search for specific events, individuals, or topics with ease. What once required manual sorting and labeling is now completed in seconds, significantly improving workflow efficiency and freeing up time for content creation and reporting.

Architecture



Entity Relationship (ER)Diagram



Pre-requisites

- 1.AWS Account Setup: [AWS Account Setup](#)
- 2.Understanding IAM: [IAM Overview](#)
- 3.Amazon EC2 Basics: [EC2 Tutorial](#)
- 4.DynamoDB Basics: [DynamoDB Introduction](#)
- 5.Git Version Control: [Git Documentation](#)

Project WorkFlow

1. AWS Account Setup and Login

Activity 1.1: Set up an AWS account if not already done.

Activity 1.2: Log in to the AWS Management Console

2. DynamoDB Database Creation and Setup

Activity 2.1: Create a DynamoDB Table.

Activity 2.2: Configure Attributes for User Data and Book Requests.

3.SNS Notification Setup

Activity 3.1: Create SNS topics for sending email notifications to users and library staff

Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time notifications

4. Backend Development and Application Setup

Activity 4.1:Develop the Backend Using Flask.

Activity 4.2: Integrate AWS Services Using boto3.

5. EC2 Instance Setup

Activity 5.1: Launch an EC2 instance to host the Flask application.

Activity 5.2: Configure security groups for HTTP, and SSH access.

6. Deployment on EC2

Activity 6.1:Upload Flask Files

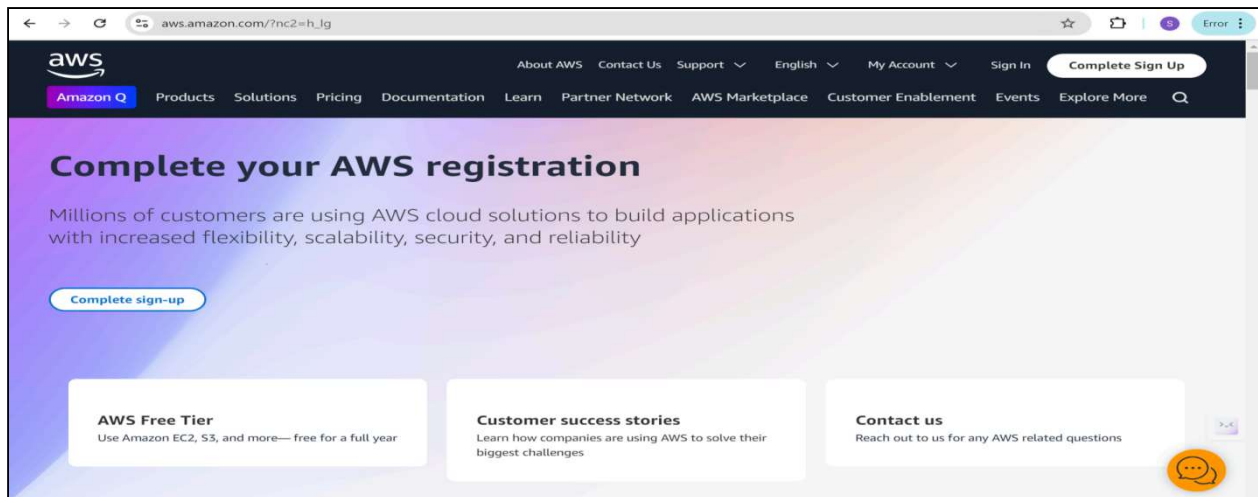
Activity 6.2: Run the Flask App

7. Testing and Deployment

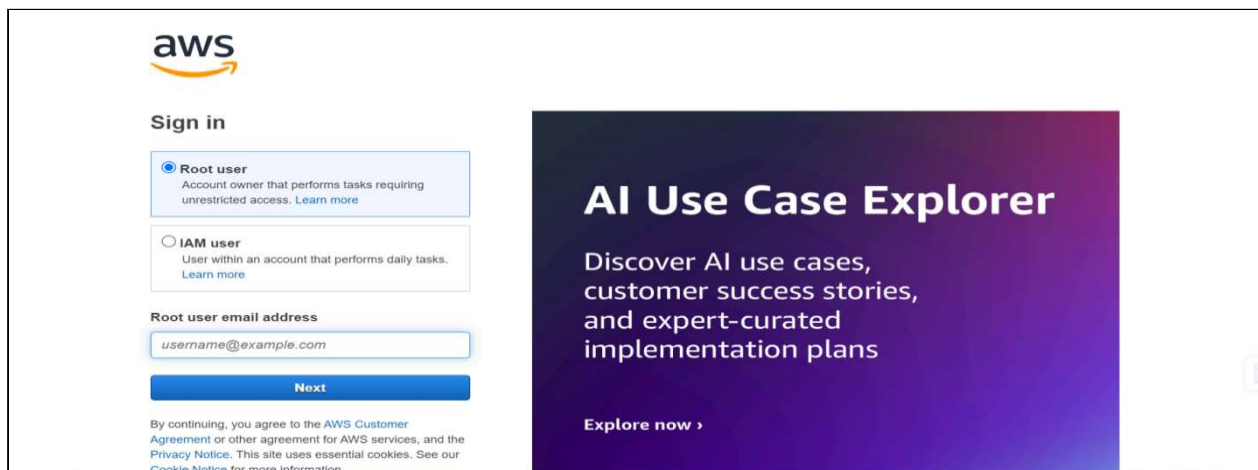
Activity 7.1: Conduct functional testing to verify user registration, login, book requests, and notifications.

Milestone 1: AWS Account Setup And Login

Activity 1.1: Set up an AWS account if not already done.

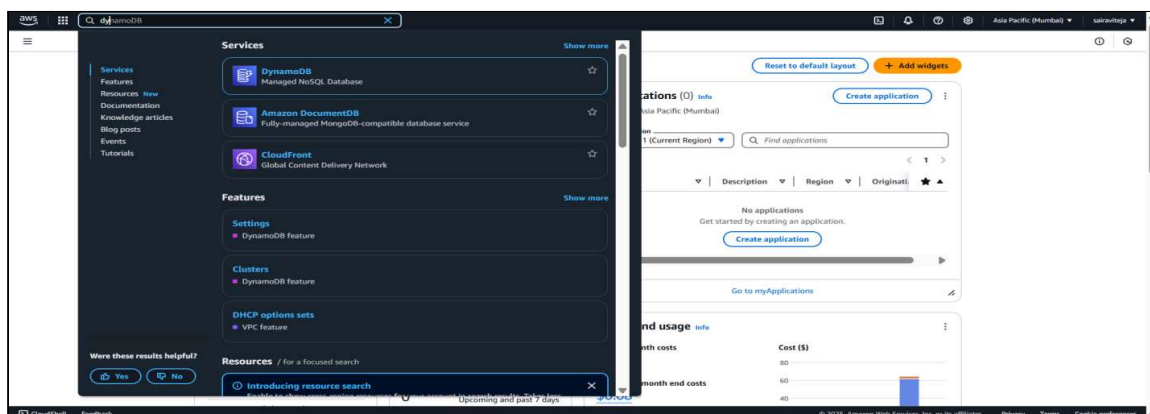


Activity 1.2: Log in to the AWS Management Console

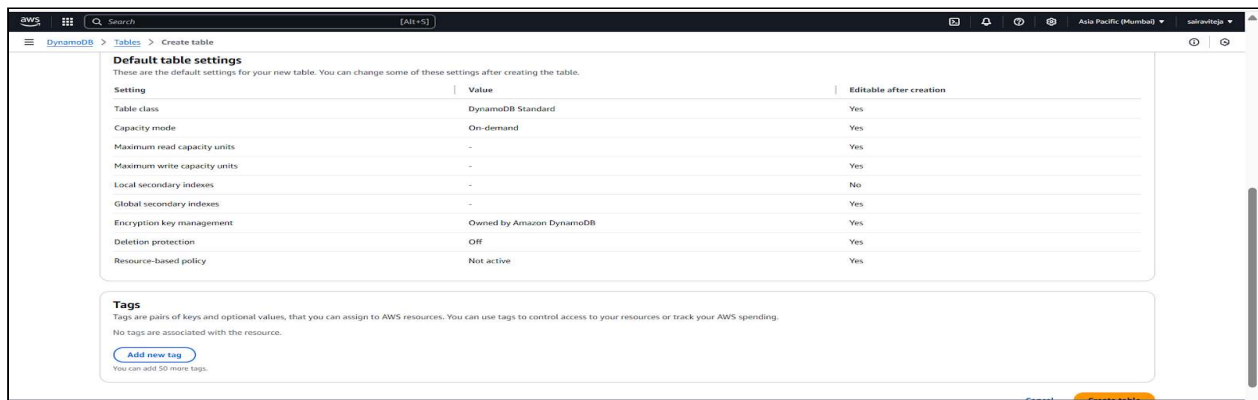
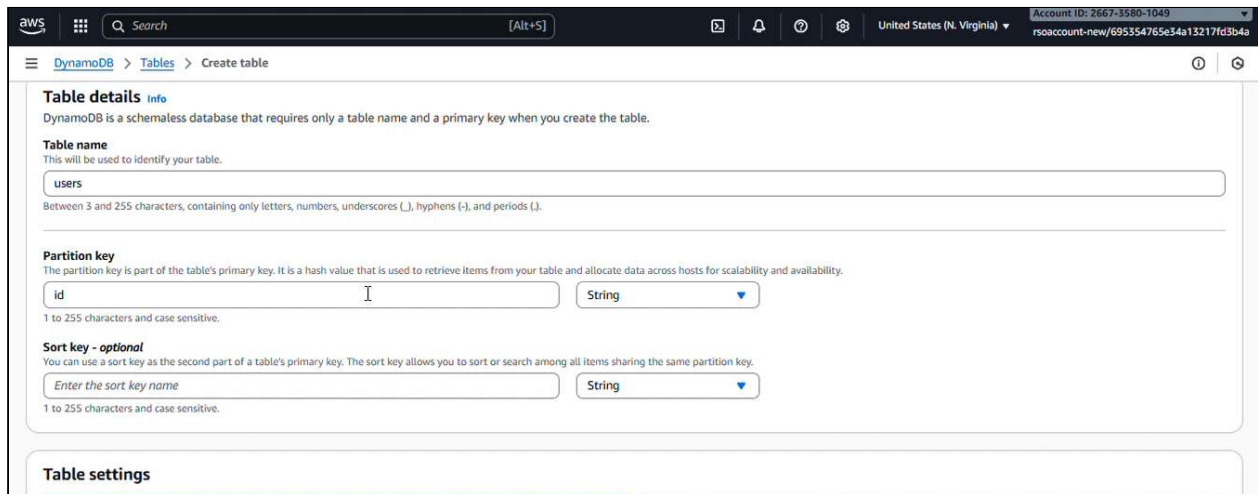
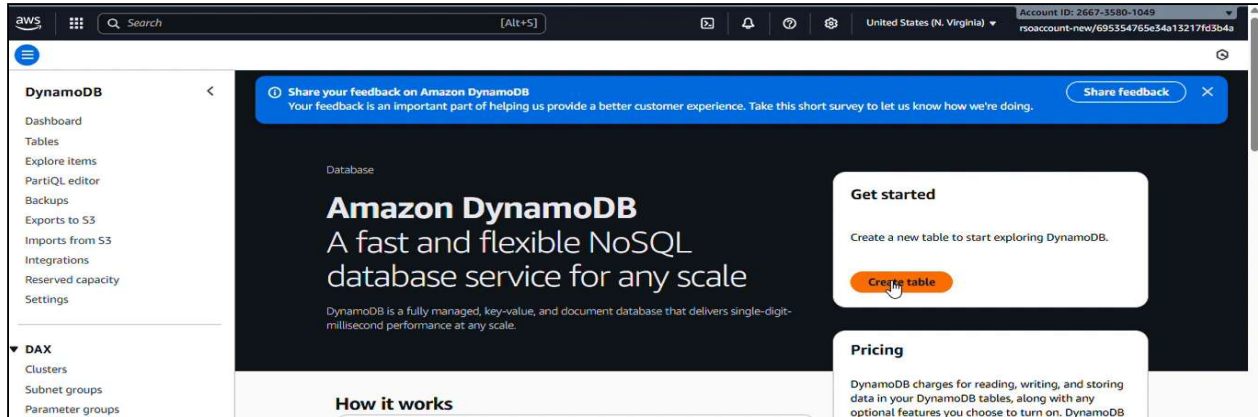


Milestone 2: DynamoDB Database Creation and Setup

Activity 2.1: Create a DynamoDB Table.



Activity 2.2: Configure Attributes for User Data and Book Requests.



aws Search [Alt+S] United States (N. Virginia) Account ID: 2667-3580-1049 rsoaccount-new/695354765e34a13217fd3b4a

DynamoDB Tables Create table

Share your feedback on Amazon DynamoDB
Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing. Share feedback

Create table

Table details info
DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.
media_items
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
id String
1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
Enter the sort key name String

aws Search [Alt+S] Create table

Default table settings
These are the default settings for your new table. You can change some of these settings after creating the table.

Setting	Value	Editable after creation
Table class	DynamoDB Standard	Yes
Capacity mode	On-demand	Yes
Maximum read capacity units	-	Yes
Maximum write capacity units	-	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Deletion protection	Off	Yes
Resource-based policy	Not active	Yes

Tags
Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending. No tags are associated with the resource.
Add new tag
You can add 50 resource tags.

Milestone 3: SNS Notification Setup

Activity 3.1: Create SNS topics for sending email notifications to users and library staff.

Search results for 'sns'

Services Show more

- Simple Notification Service** ☆
SNS managed message topics for Pub/Sub
- Route 53 Resolver**
Resolve DNS queries in your Amazon VPC and on-premises network.
- Route 53** ☆
Scalable DNS and Domain Name Registration
- AWS End User Messaging** ☆
Engage your customers across multiple communication channels

Features Show more

- Events**
ElastiCache feature
- SMS**
AWS End User Messaging feature
- Hosted zones**
Route 53 feature

Activity 3.2: Subscribe users and staff to relevant SNS topics to receive real-time

The screenshot shows the Amazon SNS console with a 'Create topic' wizard. The 'Topic name' field is filled with 'snapstream'. A blue banner at the top reads 'New Feature: Amazon SNS now supports High Throughput FIFO topics. Learn more'. The main heading is 'Amazon Simple Notification Service' with the subtitle 'Pub/sub messaging for microservices and serverless applications'. A description of Amazon SNS is provided below the heading.

Create topic

Topic name
A topic is a message channel. When you publish a message to a topic, it fans out the message to all subscribed endpoints.

snapstream

Next step

[Start with an overview](#)

The screenshot shows the 'snapstream' topic details page. The 'Details' tab is active, showing the topic name 'snapstream', its ARN 'arn:aws:sns:us-east-1:26673580-1049:snapstream', and the topic owner '266735801049'. There are buttons for 'Edit', 'Delete', and 'Publish message'. Below the details, there are tabs for 'Subscriptions', 'Access policy', 'Data protection policy', 'Delivery policy (HTTP/S)', 'Delivery status logging', 'Encryption', and 'Tags'. The 'Subscriptions' tab shows one subscription with buttons for 'Edit', 'Delete', 'Request confirmation', 'Confirm subscription', and 'Create subscription'.

Amazon SNS

Dashboard
Topics
Subscriptions
Mobile
Push notifications
Text messaging (SMS)

snapstream [Edit](#) [Delete](#) [Publish message](#)

Details

Name
snapstream

ARN
arn:aws:sns:us-east-1:26673580-1049:snapstream
[Copy ARN](#)

Display name
-

Type
Standard

Topic owner
266735801049

[Subscriptions](#) [Access policy](#) [Data protection policy](#) [Delivery policy \(HTTP/S\)](#) [Delivery status logging](#) [Encryption](#) [Tags](#)

Subscriptions (1) [Edit](#) [Delete](#) [Request confirmation](#) [Confirm subscription](#) [Create subscription](#)

The screenshot shows a confirmation message from AWS Simple Notification Service. It states 'Subscription confirmed!' and 'You have successfully subscribed.' It provides the subscription's ID as 'arn:aws:sns:ap-south-1:940482422578:next_gen_sns:0b981451-9895-4437-94c7-db270d4e9102'. It also includes a link to unsubscribe if it was not the intention.

aws

Simple Notification Service

Subscription confirmed!

You have successfully subscribed.

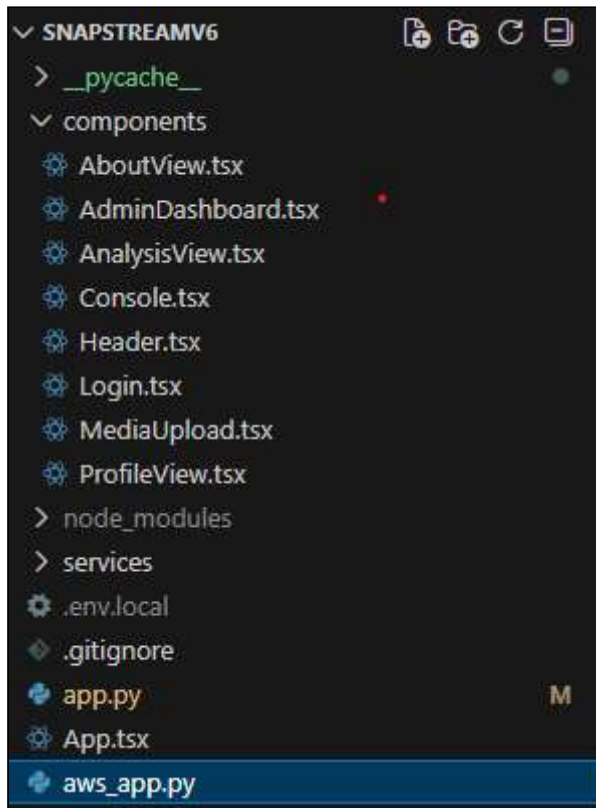
Your subscription's id is:
arn:aws:sns:ap-south-1:940482422578:next_gen_sns:0b981451-9895-4437-94c7-db270d4e9102

If it was not your intention to subscribe, [click here to unsubscribe](#).

Milestone 4: Backend Development and Application Setup

Activity 4.1: Develop the Backend Using Flask

- File explorer



Activity 4.2: Integrate AWS Services Using boto3.

```
aws_app.py > ...
1  from flask import Flask, request, jsonify
2  from flask_cors import CORS
3  import boto3
4  import uuid
5  import json
6  import os
7  from boto3.dynamodb.conditions import Key, Attr
8  from botocore.exceptions import ClientError
```

Description: This project uses Flask for routing, session management, and user authentication with secure password hashing. It integrates AWS services via Boto3 for handling data storage,

notifications, and unique user operations

```
app = Flask(__name__)
```

Description: A new Flask application instance is initialized, and a secret key is set to securely manage user sessions and protect against cookie tampering.

- **DynamoDB Setup**

```
# --- AWS Configuration ---
REGION = 'us-east-1'
SNS_TOPIC_ARN = 'arn:aws:sns:us-east-1:604665149129:aws_capstone_topic'

# Initialize AWS Resources
# In EC2, this uses the IAM Role. Locally, it uses ~/.aws/credentials.
dynamodb = boto3.resource('dynamodb', region_name=REGION)
sns = boto3.client('sns', region_name=REGION)

# Table Definitions (Match the 'id' partition keys from your app.py logic)
users_table = dynamodb.Table('users')
media_table = dynamodb.Table('media_items')
```

- **Routes for web pages**

Register User: Collecting registration data, hashes the password, and stores user details in the database.

```
@app.route('/api/register', methods=['POST'])
def register():
    data = request.json
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    user_id = str(uuid.uuid4())
    try:
        cursor.execute(
            "INSERT INTO users (id, name, email, password, role) VALUES (?, ?, ?, ?, ?)",
            (user_id, data['name'], data['email'], data['password'], data['role'])
        )
        conn.commit()
        return jsonify({"id": user_id, "name": data['name'], "email": data['email'], "role": data['role']}), 201
    except sqlite3.IntegrityError:
        return jsonify({"error": "Email already exists"}), 400
    finally:
        conn.close()
```

- **login Route** : Verifies user credentials, increments login count, and redirects to the dashboard on success.

```
@app.route('/api/login', methods=['POST'])
def login():
    data = request.json

    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()

    # Ensure email comparison is case-insensitive for better UX
    cursor.execute("SELECT id, name, email, password, role FROM users WHERE LOWER(email) = LOWER(?)", (data['email'],))

    user = cursor.fetchone()

    conn.close()
```

- **Upload media**: users able to upload their objects safe and securely

```
@app.route('/api/media', methods=['GET'])
def get_media():
    owner_id = request.args.get('owner_id')
    is_admin = request.args.get('is_admin') == 'true'

    if is_admin:
        response = media_table.scan()
    else:
        response = media_table.scan(FilterExpression=Attr('owner_id').eq(owner_id))

    items = response.get('Items', [])
    # Sort by upload_date descending (manual sort since Scan is unordered)
    items.sort(key=lambda x: x.get('upload_date', ''), reverse=True)

    # Format analysis results back to JSON for the frontend
    for item in items:
        if 'analysis_results' in item and isinstance(item['analysis_results'], str):
            item['analysis'] = json.loads(item['analysis_results'])

    return jsonify(items)

@app.route('/api/media', methods=['POST'])
def create_media():
    data = request.json
    media_item = {
        'id': data['id'],
        'owner_id': data.get('owner_id'),
        'name': data['name'],
        'type': data['type'],
        'size': data['size'],
        'status': data['status'],
        'url': data['url'],
        'thumbnail_url': data.get('thumbnailUrl'),
        'profile': data.get('profile'),
        'upload_date': str(uuid.uuid1())
    }
    # Using a timestamp-based ID or current time string
```

```

@app.route('/api/media', methods=['POST'])
def create_media():
    data = request.json
    media_item = {
        'id': data['id'],
        'owner_id': data.get('owner_id'),
        'name': data['name'],
        'type': data['type'],
        'size': data['size'],
        'status': data['status'],
        'url': data['url'],
        'thumbnail_url': data.get('thumbnailUrl'),
        'profile': data.get('profile'),
        'upload_date': str(uuid.uuid1())
        # Using a timestamp-based ID or current time string
    }

    media_table.put_item(Item=media_item)
    return jsonify({"status": "created"}), 201

@app.route('/api/media/<item_id>/analysis', methods=['PUT'])
def update_analysis(item_id):
    data = request.json
    media_table.update_item(
        Key={'id': item_id},
        UpdateExpression="SET analysis_results = :val, #s = :stat",
        ExpressionAttributeValues={
            ':val': json.dumps(data),
            ':stat': 'completed'
        },
        ExpressionAttributeNames={
            "#s": "status"
            # 'status' is a reserved keyword in DynamoDB
        }
    )
    return jsonify({"status": "updated"})

@app.route('/api/media/<item_id>', methods=['DELETE'])
def delete_media(item_id):
    media_table.delete_item(Key={'id': item_id})
    return jsonify({"status": "deleted"})

@app.route('/api/users', methods=['GET'])
def get_users():
    response = users_table.scan()
    items = response.get('Items', [])
    # Filter out passwords before sending to frontend
    users = [{"id": u['id'], "name": u['name'], "email": u['email'], "profile": u['profile']} for u in items]

```

- Application Entry point:

```

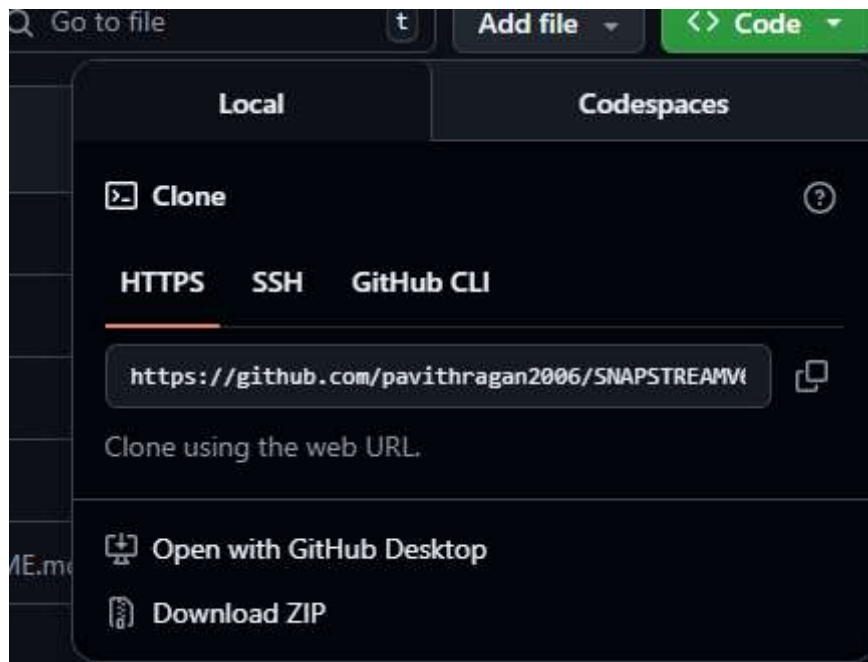
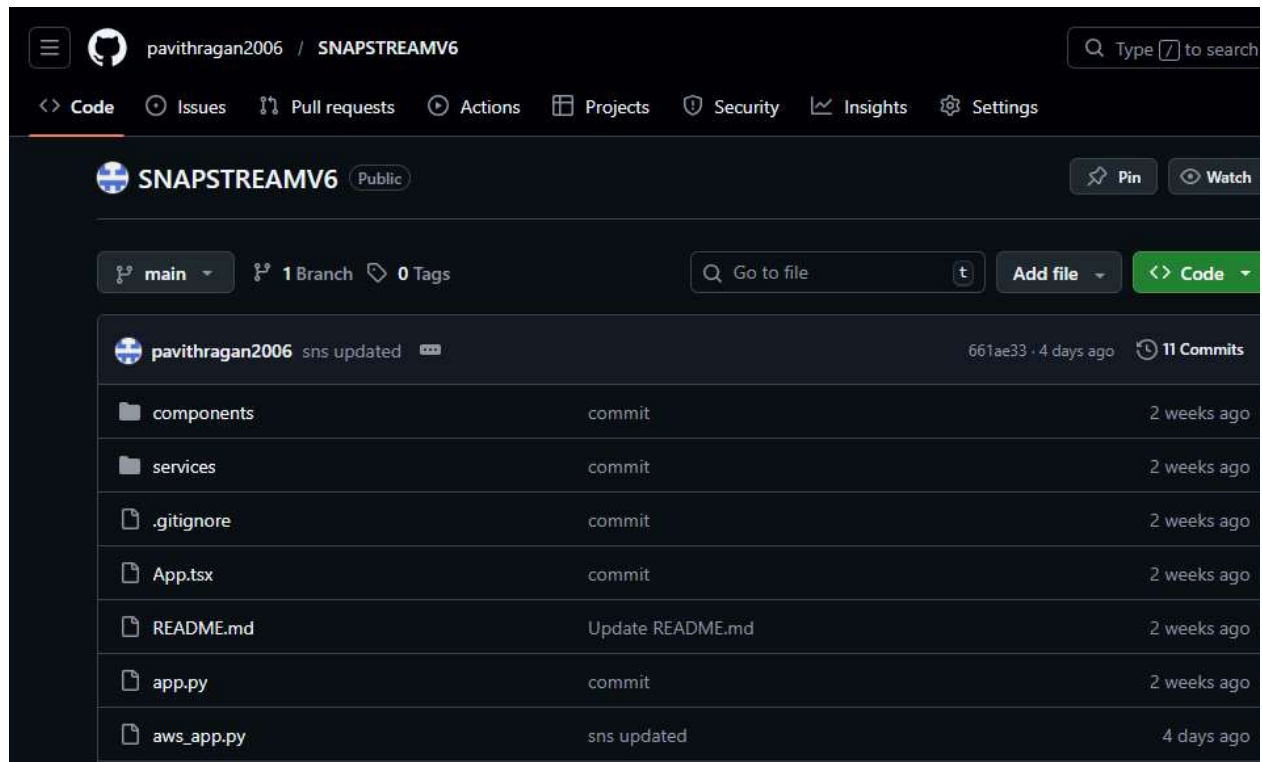
if __name__ == '__main__':

    app.run(port=5000, debug=True)

```

Milestone 5: EC2 Instance Setup

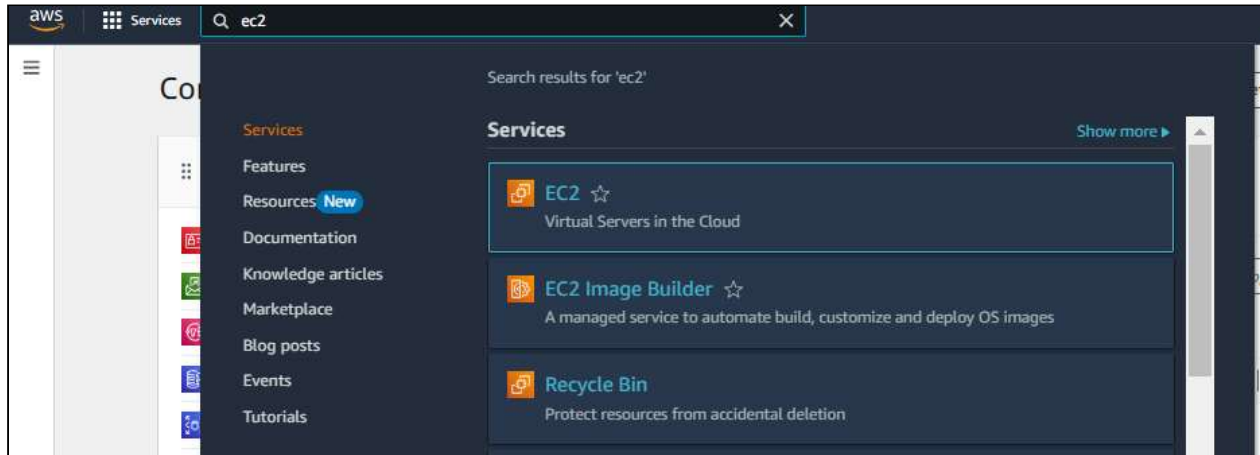
- **Note:** Load your Flask app and Html files into GitHub repository.



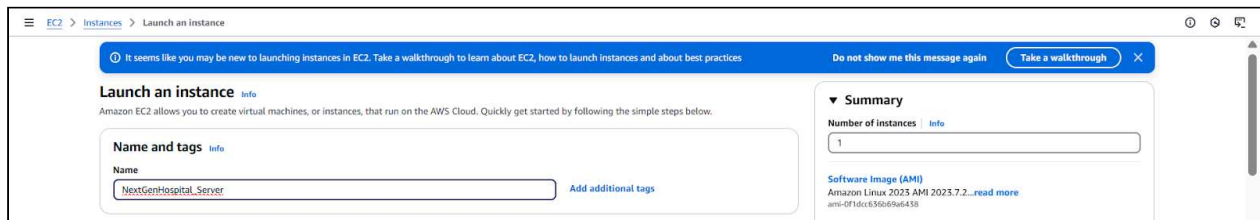
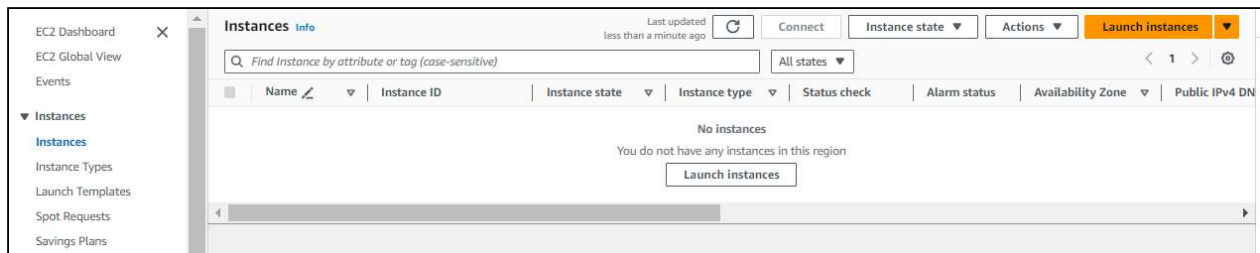
Activity 5.1: Launch an EC2 instance to host the Flask application.

- **Launch EC2 Instance**

In the AWS Console, navigate to EC2 and launch a new instance.



- **Click on Launch instance to launch EC2 instance**



- **Choose Amazon Linux 2 or Ubuntu as the AMI and t2.micro as the instance type (free-tier eligible)**

Amazon Linux
aws

macOS
Mac

Ubuntu
ubuntu

Windows
Microsoft

Red Hat
Red Hat

Browse more AMIs

Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Amazon Linux 2023 AMI

Free tier eligible

ami-02b49a24cfb95941c (64-bit (x86), uefi-preferred) / ami-04ad8c7fcc828fad4 (64-bit (Arm), uefi)

Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Amazon Linux 2023 is a modern, general purpose Linux-based OS that comes with 5 years of long term support. It is optimized for AWS and designed to provide a secure, stable and high-performance execution environment to develop and run your cloud applications.

Architecture	Boot mode	AMI ID	
64-bit (x86)	uefi-preferred	ami-02b49a24cfb95941c	Verified provider

- Create and download the key pair for Server access.

▼ Instance type Info | Get advice

Instance type

t2.micro

Free tier eligible

Family: t2 1 vCPU 1 GiB Memory Current generation: true
 On-Demand Linux base pricing: 0.0124 USD per Hour
 On-Demand Windows base pricing: 0.017 USD per Hour
 On-Demand RHEL base pricing: 0.0268 USD per Hour
 On-Demand SUSE base pricing: 0.0124 USD per Hour

☐ All generations

[Compare instance types](#)

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

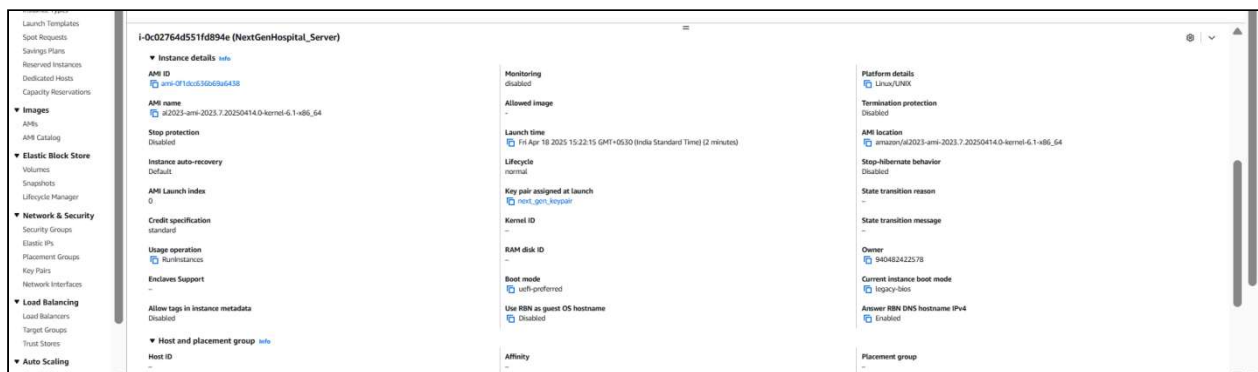
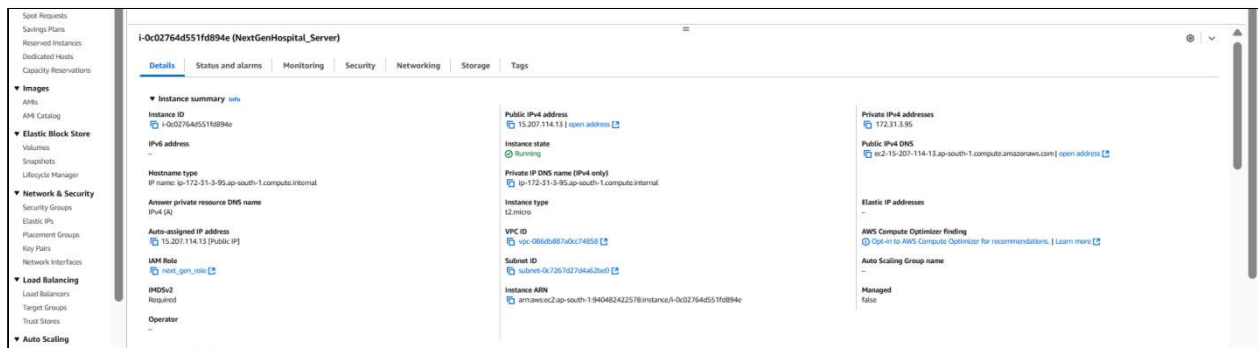
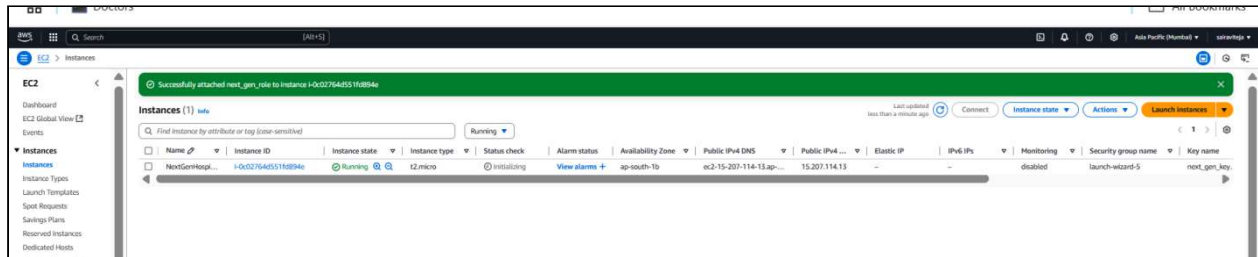
Select

Create new key pair

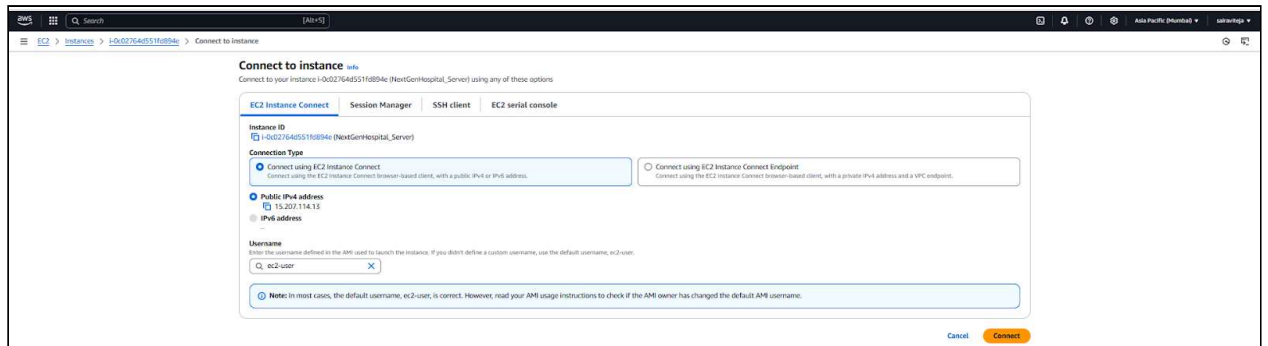
Activity 5.2: Configure security groups for HTTP, and SSH access.

to your EC2 instance. You can do this by selecting your instance, clicking on Actions, then navigating to Security and

- navigate to the EC2 section in the AWS Management Console. Select the EC2 instance you wish to connect to. At the top of the EC2 Dashboard, click the Connect button. From the connection methods presented, choose EC2 Instance Connect. Finally, click Connect again,



- Now connect the EC2 with the files



Milestone 6: Deployment on EC2

Activity 6.1: Install Software on the EC2 Instance

Install Python3, Flask, and Git:

On Amazon Linux 2:

```
sudo yum update -y
```

```
sudo yum install python3 git
```

```
sudo pip3 install flask boto3
```

Verify Installations:

```
flask --version
```

```
git --Version
```

Activity 6.2: Clone Your Flask Project from GitHub

Clone your project repository from GitHub into the EC2 instance using Git.

Run: `git clone https://github.com/pavithragan2006/SNAPSTREAMV6.git`

Note: change your-github-username and your-repository-name with your credentials

- This will download your project to the EC2 instance.

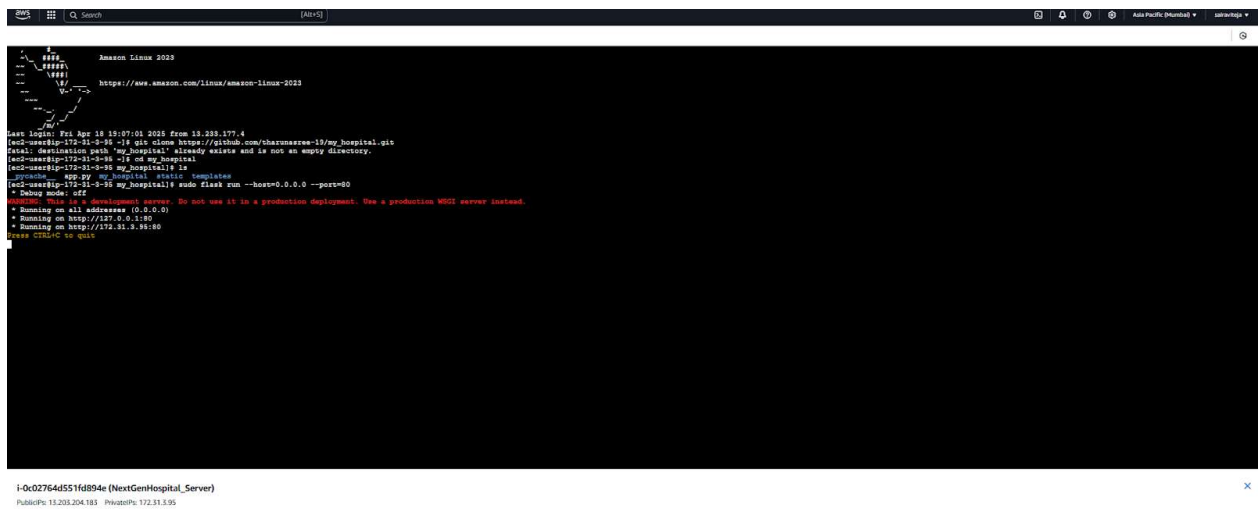
To navigate to the project directory, run the following command:

```
cd snapstream
```

- Once inside the project directory, configure and run the Flask application by executing the following command with elevated privileges:

Run the Flask Application

```
sudo flask run --host=0.0.0.0 --port=80
```



```
Amazon Linux 2023
https://aws.amazon.com/linux/amazon-linux-2023

Last login: Fri Apr 18 19:07:01 2025 from 13.233.177.4
[ec2-user@ip-172-31-3-96 ~]$ git clone https://github.com/sharunasree-19/my_hospital.git
fatal: destination path 'my_hospital' already exists and is not an empty directory.
[ec2-user@ip-172-31-3-96 ~]$ cd my_hospital
[ec2-user@ip-172-31-3-96 my_hospital]$ ls
__pycache__  app.py  my_hospital  static  templates
[ec2-user@ip-172-31-3-96 my_hospital]$ sudo flask run --host=0.0.0.0 --port=80
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:80
 * Running on http://172.31.3.96:80
Press CTRL+C to quit
```



```
100 120.51.20 - - [13/Apr/2025 10:57:30] "GET / HTTP/1.1" 200 -
100 120.51.20 - - [13/Apr/2025 10:57:31] "GET /static/css/style.css HTTP/1.1" 200 -
100 120.51.20 - - [13/Apr/2025 10:57:31] "GET /static/js/script.js HTTP/1.1" 200 -
100 120.51.20 - - [13/Apr/2025 10:57:31] "GET /static/images/testimonial2.webp HTTP/1.1" 200 -
100 120.51.20 - - [13/Apr/2025 10:57:31] "GET /static/images/testimonial1.jpg HTTP/1.1" 200 -
100 120.51.20 - - [13/Apr/2025 10:57:31] "GET /static/images/testimonial3.webp HTTP/1.1" 200 -
100 120.51.20 - - [13/Apr/2025 10:57:31] "
```

Verify the Flask app is running:

`http://your-ec2-public-ip`

- Run the Flask app on the EC2 instance

```
[ec2-user@ip-172-31-3-95 my_hospital]$ sudo flask run --host=0.0.0.0 --port=80
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:80
* Running on http://172.31.3.95:80
Press CTRL+C to quit
103.120.51.20 - - [19/Apr/2025 10:37:30] "GET / HTTP/1.1" 200 -
103.120.51.20 - - [19/Apr/2025 10:37:31] "GET /static/css/style.css HTTP/1.1" 200 -
103.120.51.20 - - [19/Apr/2025 10:37:31] "GET /static/js/script.js HTTP/1.1" 200 -
103.120.51.20 - - [19/Apr/2025 10:37:31] "GET /static/images/testimonial2.webp HTTP/1.1" 200 -
103.120.51.20 - - [19/Apr/2025 10:37:31] "GET /static/images/testimonial1.jpg HTTP/1.1" 200 -
103.120.51.20 - - [19/Apr/2025 10:37:31] "GET /static/images/testimonial3.webp HTTP/1.1" 200 -
103.120.51.20 - - [19/Apr/2025 10:37:31] "GET /static/img/hero-banner.jpg HTTP/1.1" 404 -

[ec2-user@ip-172-31-29-154 ~]$ cd SNAPSTREAMV6 venv/
bash: cd: too many arguments
[ec2-user@ip-172-31-29-154 ~]$ cd SNAPSTREAMV6
[ec2-user@ip-172-31-29-154 SNAPSTREAMV6]$ ls
app.py components index.tsx package-lock.json services tsconfig.json vite.config.ts
README.md aws_app.py index.html metadata.json package.json snapstream.db types.ts
[ec2-user@ip-172-31-29-154 SNAPSTREAMV6]$
[ec2-user@ip-172-31-29-154 SNAPSTREAMV6]$ python3 aws_app.py
/home/ec2-user/venv/lib64/python3.9/site-packages/boto3/compat.py:89: PythonDeprecationWarning: Boto3 will no longer support Python 3.9 starting April 29, 2026. To continue receiving service updates, bug fixes, and security updates please upgrade to Python 3.10 or later. More information can be found here: https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
warnings.warn(warning, PythonDeprecationWarning)
* Serving Flask app 'aws_app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.31.29.154:5000
Press CTRL+C to quit
* Restarting with stat
/home/ec2-user/venv/lib64/python3.9/site-packages/boto3/compat.py:89: PythonDeprecationWarning: Boto3 will no longer support Python 3.9 starting April 29, 2026. To continue receiving service updates, bug fixes, and security updates please upgrade to Python 3.10 or later. More information can be found here: https://aws.amazon.com/blogs/developer/python-support-policy-updates-for-aws-sdks-and-tools/
warnings.warn(warning, PythonDeprecationWarning)
* Debugger is active!
* Debugger PIN: 145-276-357

i-098465b82a47f93f3 (snapstream_backend)
```

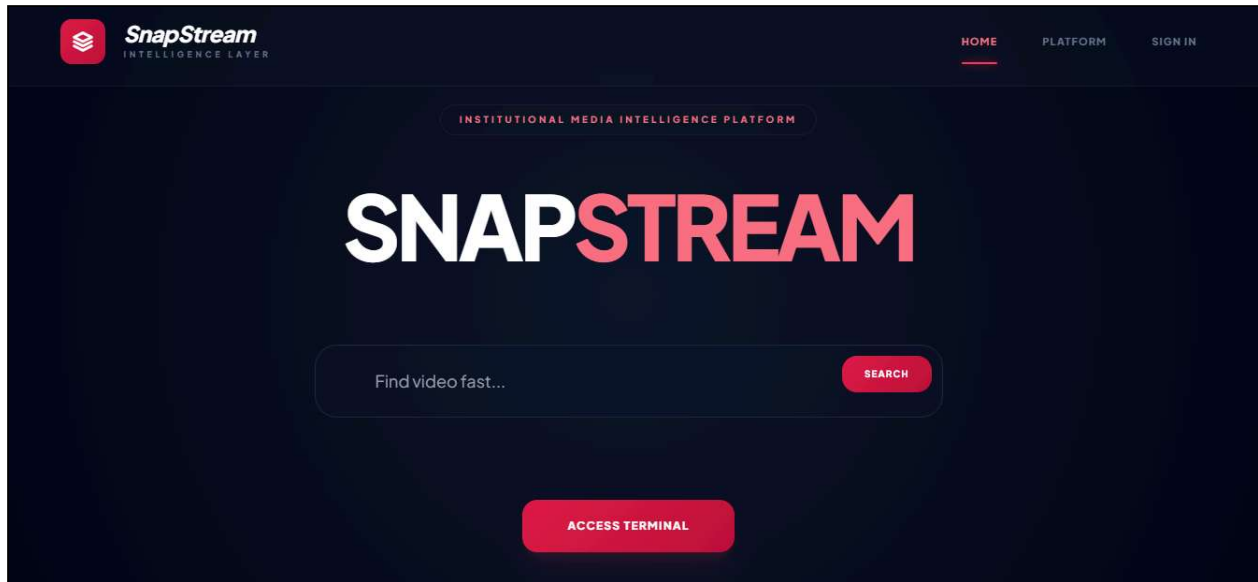
Access the website through:

PublicIPs:`http://13.203.204.183/80`

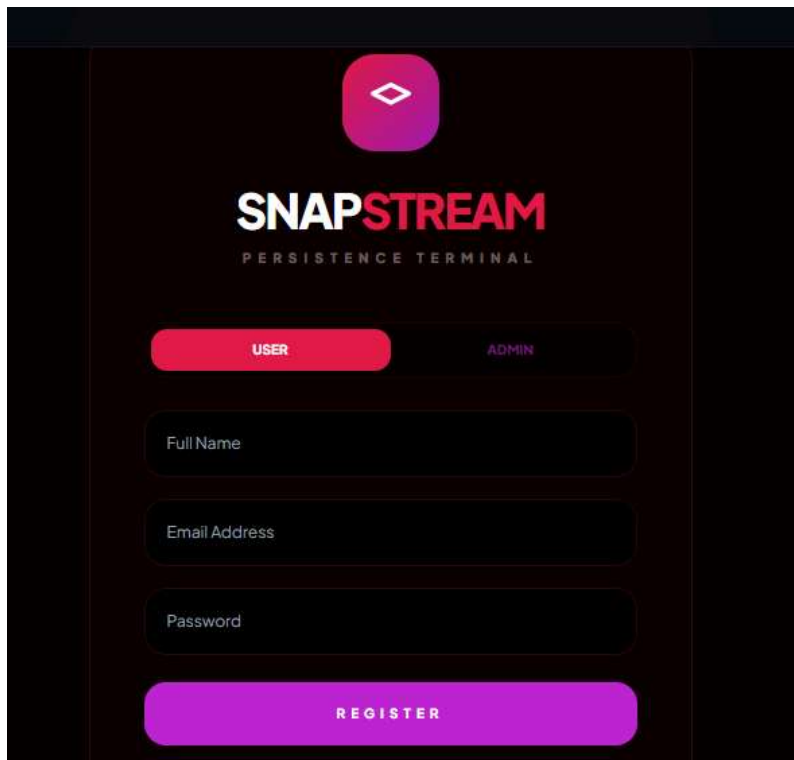
Milestone 7: Testing and Deployment

Activity 7.1: Conduct functional testing to verify user registration, login, book requests, and notifications.

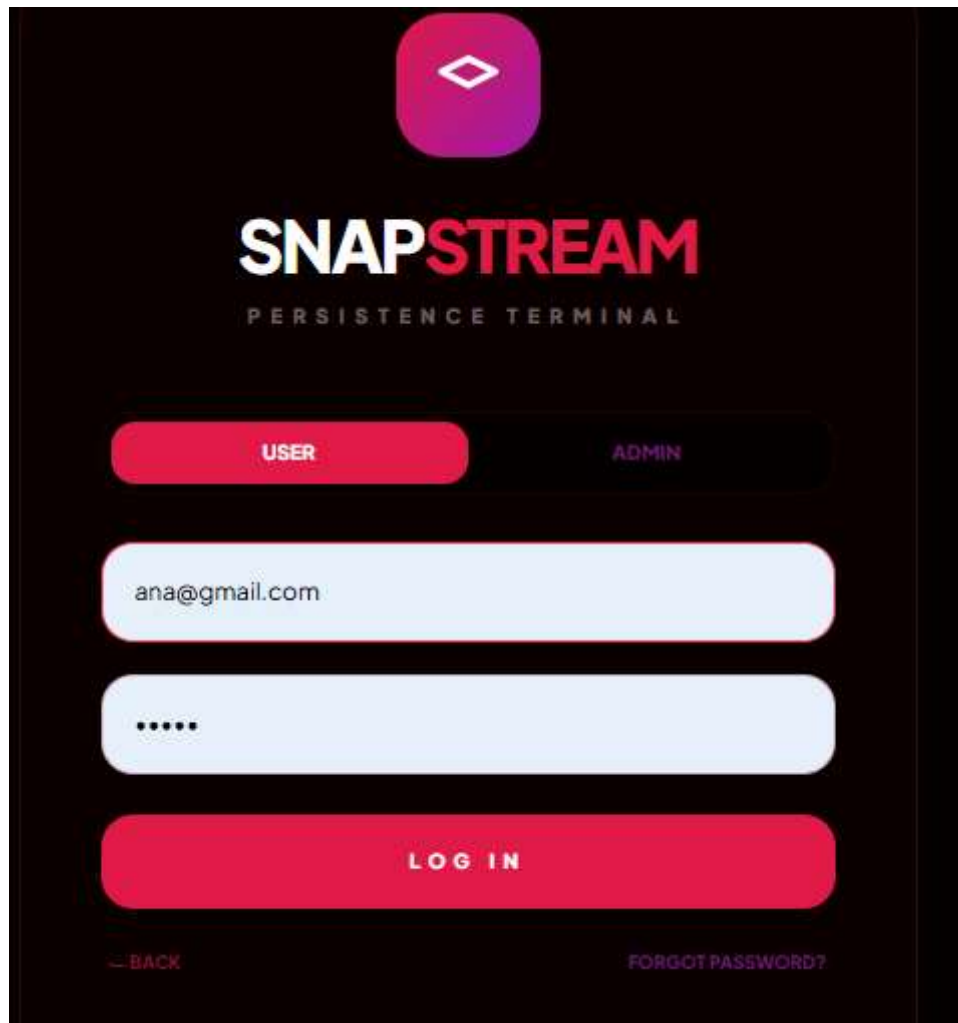
Home page




Register Page



Login page



The image shows a login page for 'SNAPSTREAM PERSISTENCE TERMINAL'. At the top center is a purple rounded square icon with a white diamond shape inside. Below the icon, the text 'SNAPSTREAM' is displayed in a large, bold, sans-serif font, with 'SNAP' in white and 'STREAM' in red. Underneath this, 'PERSISTENCE TERMINAL' is written in a smaller, white, all-caps font. The page features two toggle buttons: 'USER' (highlighted in red) and 'ADMIN' (in purple). Below these are two light blue input fields. The first field contains the email 'ana@gmail.com'. The second field contains five dots, indicating a password. A large red 'LOG IN' button is positioned below the input fields. At the bottom left is a purple '← BACK' link, and at the bottom right is a purple 'FORGOT PASSWORD?' link.



SNAPSTREAM
PERSISTENCE TERMINAL

USER ADMIN

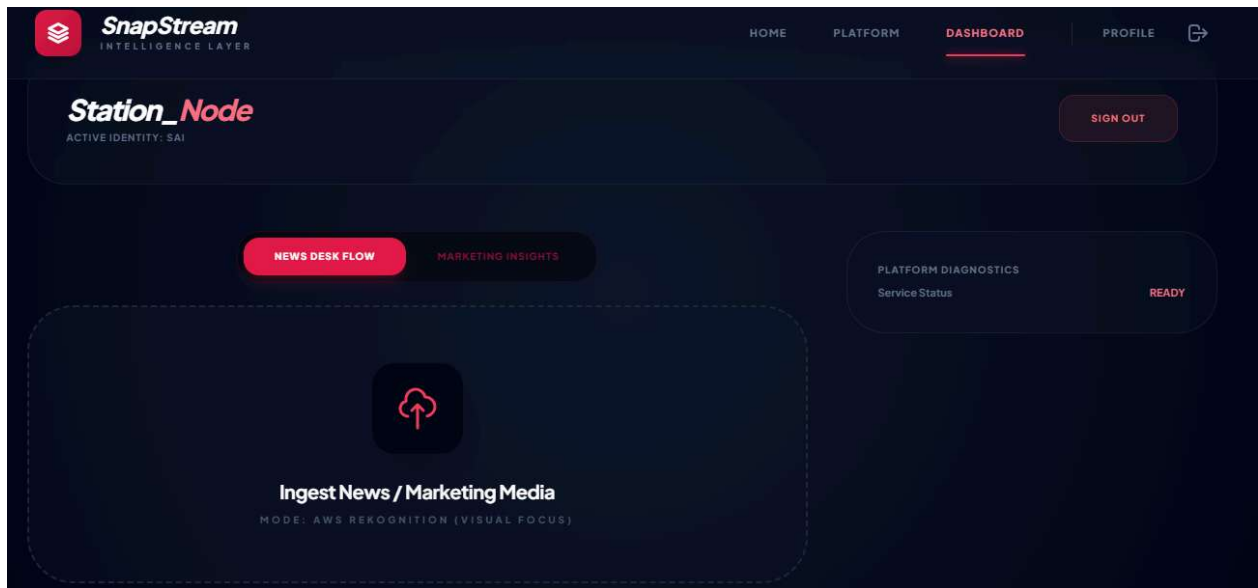
ana@gmail.com

.....

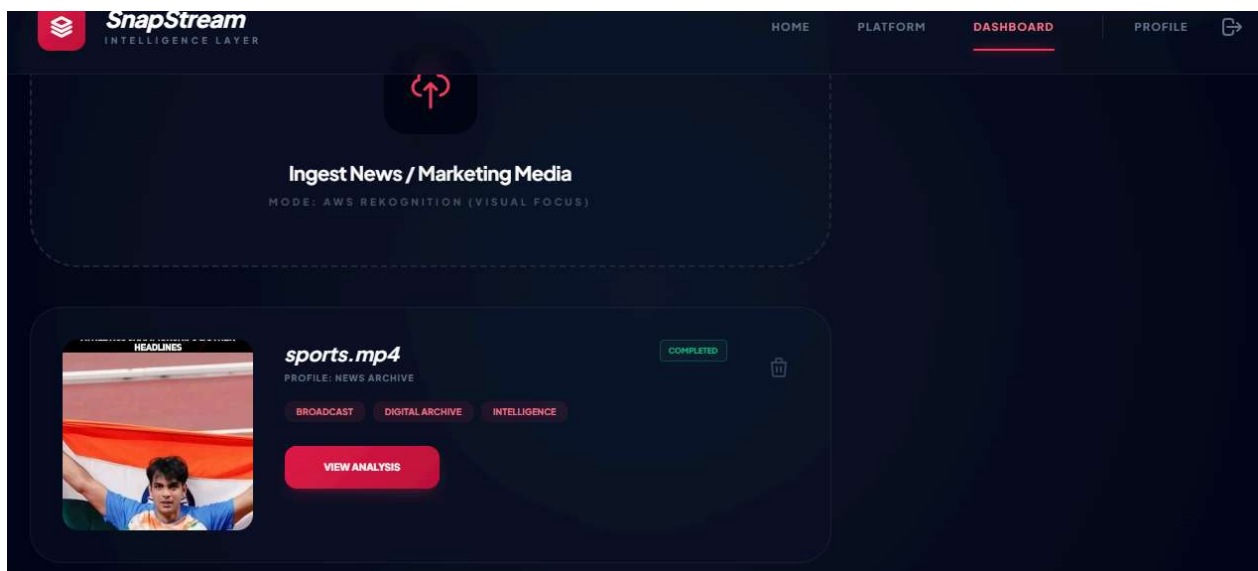
LOG IN

← BACK FORGOT PASSWORD?

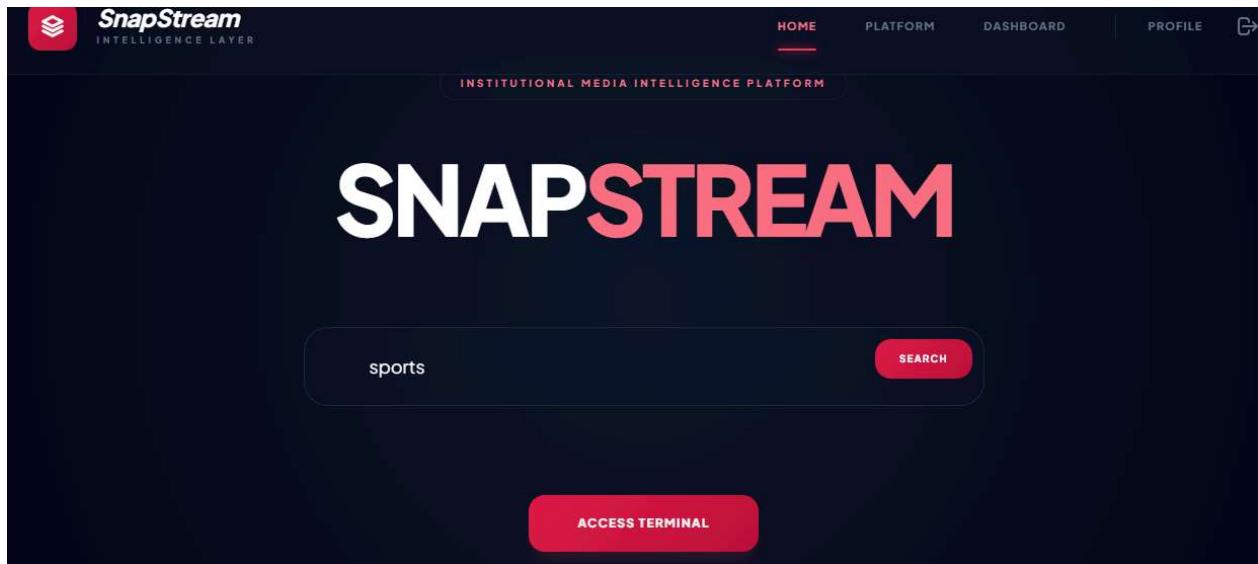
User Dashboard



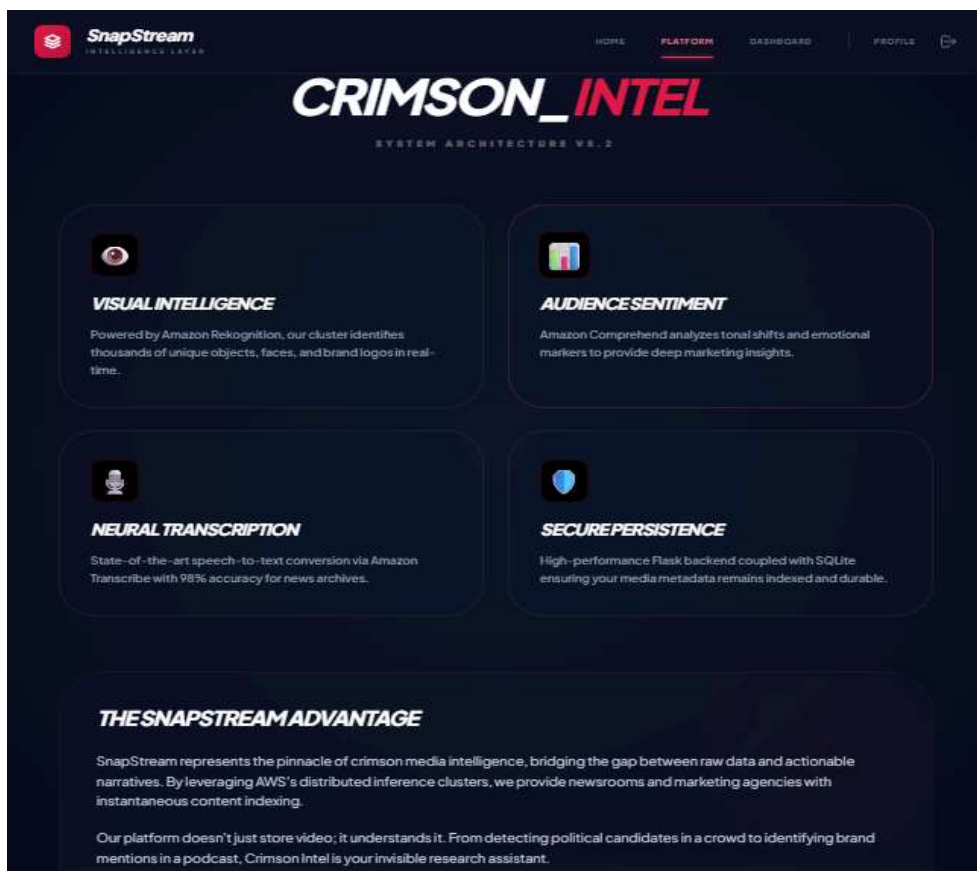
upload media



View Media



about page



CONCLUSION

The SnapStream platform has been successfully implemented as a cloud-native media management and analysis solution powered by AWS services. By integrating scalable and serverless cloud components, the system delivers a secure, efficient, and intelligent framework for handling large volumes of multimedia content. This implementation demonstrates how modern cloud technologies can simplify complex workflows associated with media storage, processing, and analysis.

Through the seamless integration of Amazon S3, AWS Lambda, and AI/ML services such as Amazon Rekognition, Transcribe, and Comprehend, SnapStream automates time-intensive tasks like tagging, transcription, and content analysis. This not only enhances operational productivity but also empowers users with data-driven insights and faster access to valuable media information.

The platform's modular design ensures scalability and cost efficiency, while AWS IAM, DynamoDB, and SNS collectively guarantee security, reliability, and real-time communication. Extensive testing has validated all critical functionalities, from secure uploads to automated analysis pipelines, confirming SnapStream's readiness for production-level deployment.

In conclusion, SnapStream represents a significant step forward in intelligent media management, showcasing the power of AWS cloud infrastructure to transform traditional media workflows into an automated, scalable, and insight-driven ecosystem.