AWS File Sharing App

Team Information

Name	ID	Role
عمر محمد خميس	23011384	IAM & Testing & Documentation
محمد اكرم على	23011462	AWS Account Setup
يوسف وائل هارون	23011660	S3 & File Handling
يوسف عثمان محمد	23011645	Web App Development
مصطفى محمد مرشدي	23011156	VPC
بدر احمد محمد	23011235	EC2

Project Summary

The AWS File Sharing App is a web application hosted on AWS EC2, utilizing Amazon S3 for secure file storage. Users can upload files (.pdf, .jpg, .png, .txt) through a simple web interface built with HTML, CSS, and JavaScript, and store them in an S3 bucket. The application, developed using the Flask framework and Boto3 for AWS interactions, provides download links for retrieving files. The project is deployed within a custom VPC, with IAM roles ensuring secure access between EC2 and S3. This project demonstrates cloud deployment, file handling, and secure AWS resource management.

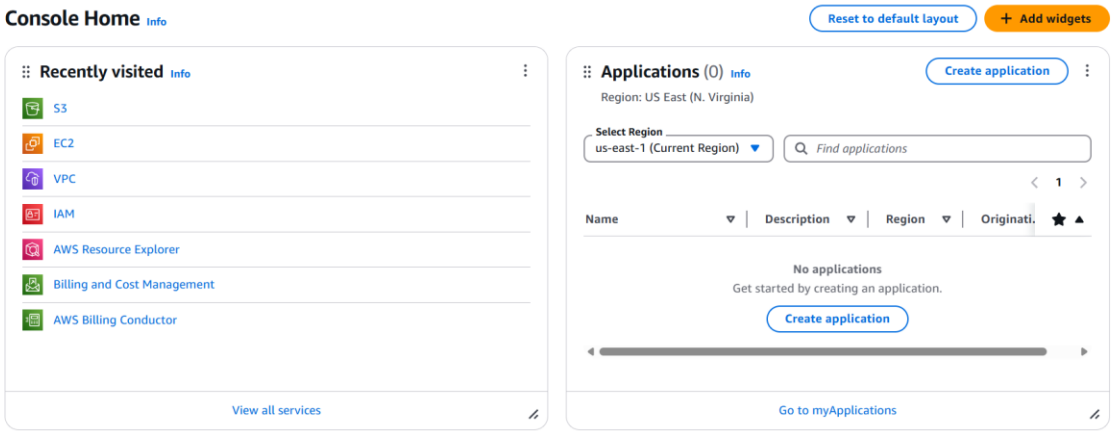
Project Components and Setup Steps

1. AWS Account Setup

Steps:

- Created an AWS Free Tier account at AWS Free Tier.
- Selected the us-east-1 (N. Virginia) region for consistency across services.
- Configured billing management, setting a $1 budget.
- Created three CloudWatch billing alarms at 20%, 50%, and 80% of the budget to monitor usage.

Importance: The AWS account setup ensures access to the Free Tier, minimizing costs. Selecting us-east-1 ensures compatibility with most AWS services. Billing alarms are critical for cost control, alerting the team to potential overages.

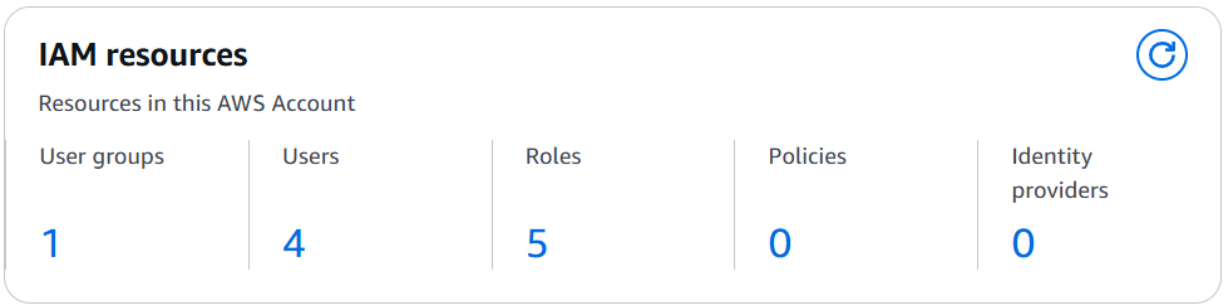


2. IAM Configuration

Steps:

- Created four IAM users with specific permissions:
  - Admin User: Full AWS permissions for account management.
  - Network User: **VPCFullAccess** for VPC configuration.
  - EC2 User: EC2FullAccess for instance management.
  - S3 User: S3FullAccess for bucket operations.
- Created an IAM role for EC2 with a policy allowing access to the S3 bucket (s3:PutObject, s3:GetObject).
- Attached the IAM role to the EC2 instance to enable secure S3 interactions.

Importance: IAM users and roles enforce the principle of least privilege, ensuring team members only access necessary resources. The EC2-S3 role secures file operations without hardcoding credentials, enhancing security.

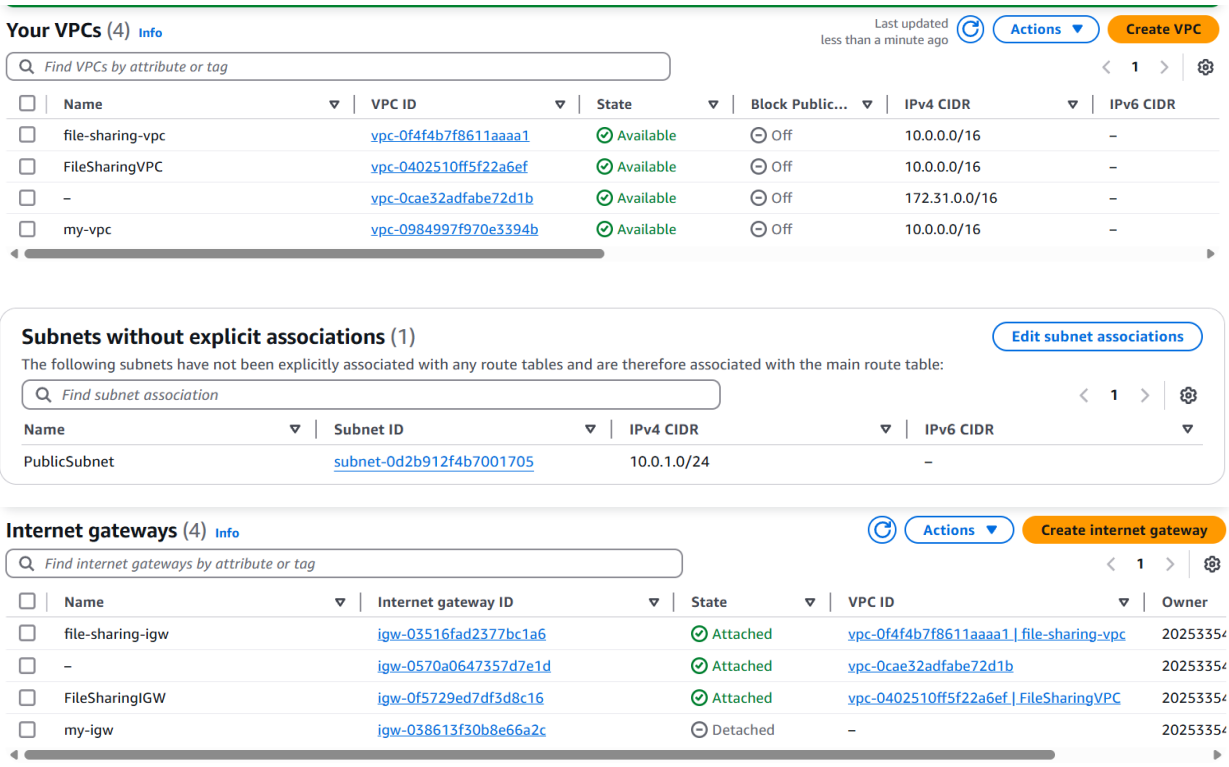


3. VPC Configuration

Steps:

- Created a custom VPC with a CIDR block of 10.0.0.0/16.
- Added a public subnet (10.0.1.0/24) in us-east-1a.
- Created an Internet Gateway and attached it to the VPC.
- Updated the route table to route 0.0.0.0/0 to the Internet Gateway.
- Configured a security group for the EC2 instance, allowing inbound HTTP (port 80) and SSH (port 22) from 0.0.0.0/0.
- Deployed the EC2 instance in the public subnet.

Importance: The VPC isolates the application in a secure network. The public subnet and Internet Gateway enable internet access for the web app. Security groups restrict traffic, protecting the instance from unauthorized access.



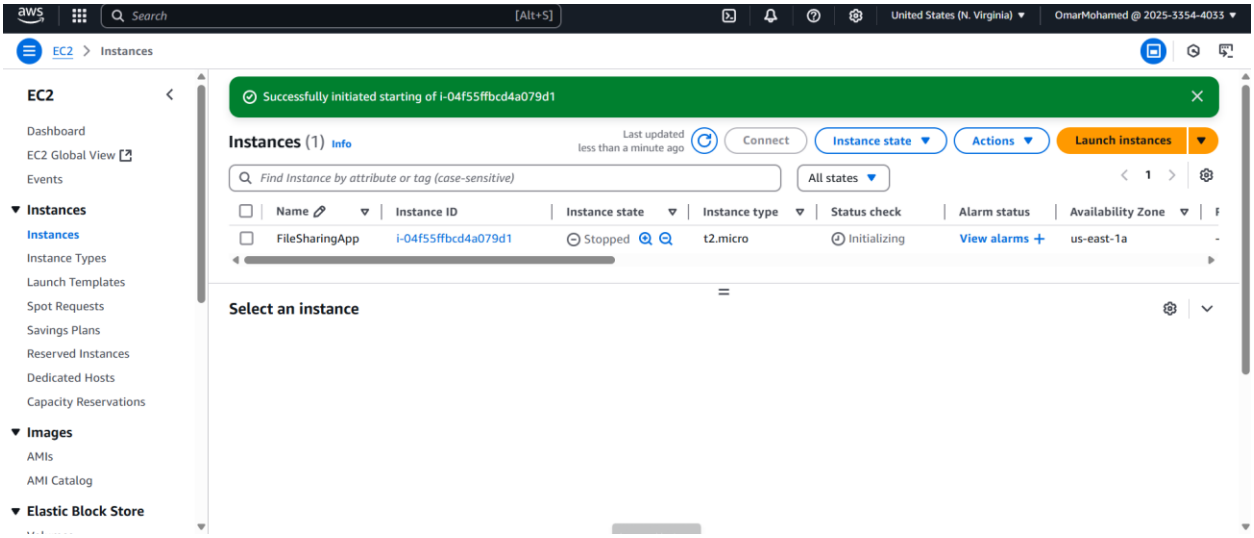
4. EC2 Setup

Steps:

- Launched a t2.micro instance (Free Tier eligible) with Amazon Linux 2 AMI.

- Assigned the IAM role created for S3 access.
- Used a user data script to install dependencies (Python, Flask, Boto3) and deploy the Flask app on launch.
- Configured the instance to run the Flask app on port 80.
- Tested SSH access and verified the web app was accessible via the public IP.

Importance: The EC2 instance hosts the Flask web application, serving the user interface and handling file upload/download logic. The t2.micro instance ensures cost efficiency, and the user data script automates deployment, reducing manual configuration.

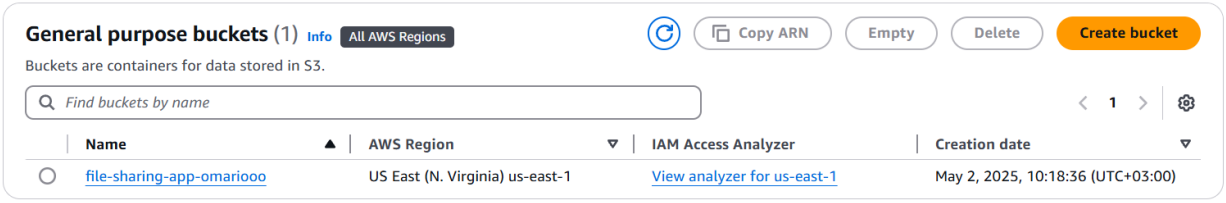


5. S3 Setup

Steps:

- Created an S3 bucket in us-east-1 with a unique name (`file-sharing-bucket`).
- Configured bucket permissions to block public access, ensuring security.
- Added a bucket policy to allow the EC2 IAM role to perform s3:PutObject and s3:GetObject actions.
- Enabled versioning to protect against accidental file deletion.
- Tested file uploads and downloads via the Flask app.

Importance: The S3 bucket securely stores user-uploaded files, providing scalable and durable storage. Proper permissions and versioning ensure data security and integrity, critical for a file-sharing application.

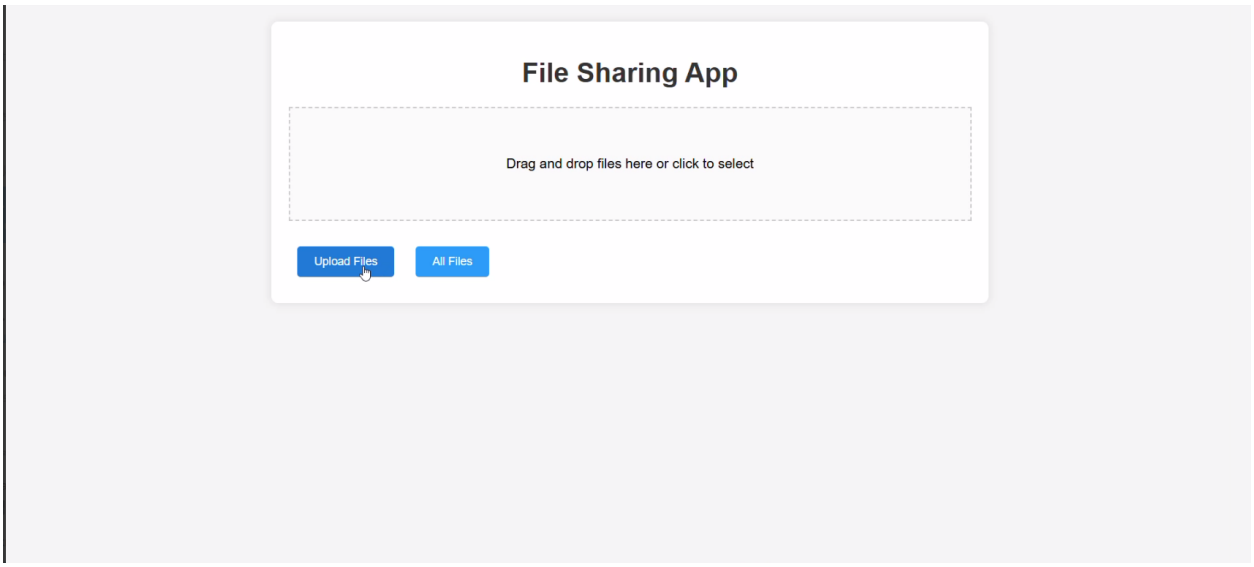


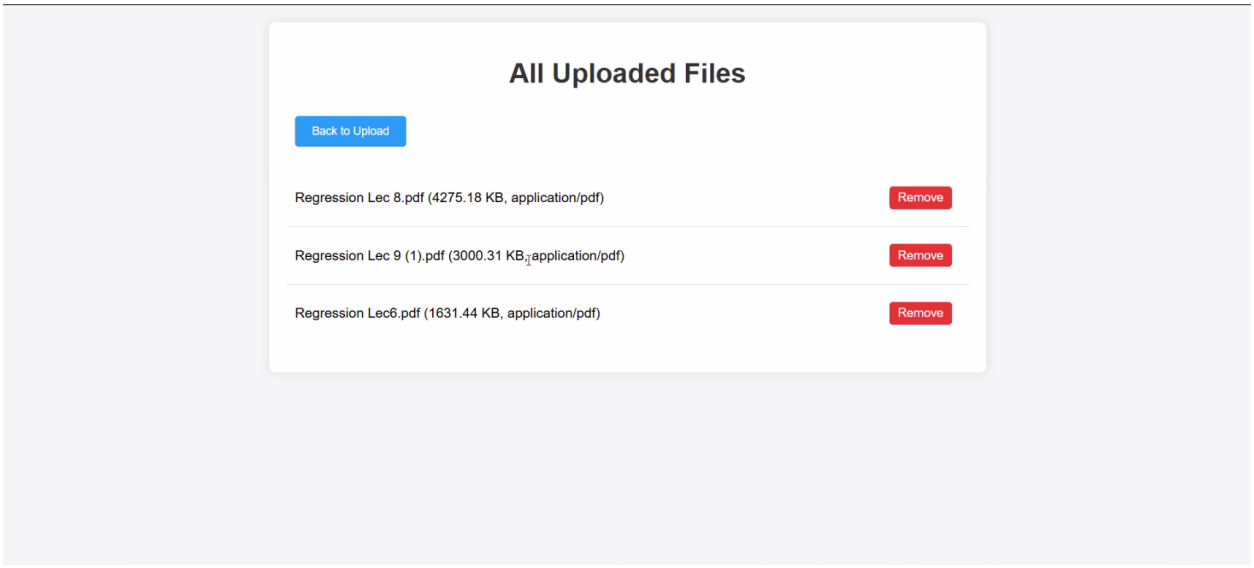
6. Web Application Development

Steps:

- Developed a Flask backend with APIs for file upload and download, using Boto3 to interact with the S3 bucket.
- Created a frontend with HTML, CSS, and JavaScript, featuring two pages:
  - Upload page: A form for selecting and uploading files.
  - Files page: A list of uploaded files with download links.
- Styled the interface with CSS for a clean, user-friendly design.
- Deployed the app on the EC2 instance, ensuring it runs on port 80.
- Tested file uploads, storage in S3, and download link functionality.

Importance: The web application provides the user interface and logic for file sharing. Flask and Boto3 enable seamless integration with AWS, while the frontend ensures accessibility and usability for end users.





Challenges and Resolutions

- **Challenge:** Initial EC2 instance failed to access S3 due to incorrect IAM role permissions.
    - **Resolution:** Updated the IAM role policy to include s3:PutObject and s3:GetObject, and reattached it to the instance.
- **Challenge:** Web app was inaccessible due to security group misconfiguration.
    - **Resolution:** Added inbound rules for port 80 (HTTP) and verified connectivity.
- **Challenge:** File uploads exceeded S3 bucket size limits during testing.