

Cloud Drive Application: Design and Implementation Report

Hsing-Chen (Tony) Lin

October 20, 2024

Contents

1	Introduction	3
2	System Architecture	3
2.1	Web Tier	3
2.2	Load Balancer	3
2.3	Application Tier	4
2.4	Database Tier	4
2.5	Storage Tier	5
3	User Interface	6
3.1	Home Page (Not Logged In)	6
3.2	Login Page	7
3.3	Register Page	8
3.4	Home Page (Logged In)	9
3.5	Upload File Page	9
3.6	File List Page	10
3.7	Profile Page	11
4	Functionality and Implementation	11
4.1	User Authentication and Registration	11
4.2	File Upload and Management	12

5	AWS Integration	13
5.1	EC2 Instance Setup	14
5.2	Elastic Load Balancer	14
5.3	AWS S3 (Storage)	16
5.4	AWS RDS (MySQL)	17
6	Conclusion	19

1 Introduction

This project implements a cloud-based application that allows users to securely store, manage, and retrieve files in the cloud. The Cloud Drive Application is a web-based system built using Django, integrating with AWS services such as Amazon S3 for file storage, Amazon RDS for user information management, and Elastic Load Balancer (ELB) for distributing traffic across multiple EC2 instances. Users can register, log in, and manage their files efficiently, with all data being stored securely in the cloud.

2 System Architecture

2.1 Web Tier

The web tier serves as the front-end of the application, developed using the Django framework, handling user interactions, form submissions, and navigation between different pages.

The key pages provided by the web tier are:

- **Home Page:** Once logged in, the user is presented with the options to upload files, view the list of their files, and access their profile information.
- **Login and Registration Pages:** These pages handle user authentication and account creation, ensuring that only authenticated users can access the file management features.
- **File Management Pages:** These pages allow users to upload files to AWS S3 and manage them. Users can view a list of their uploaded files, along with file size and the option to delete files.
- **View Profile Page:** This page displays the user's personal information, such as username and email. Although editing features are not yet implemented, users can view their basic profile data.

2.2 Load Balancer

The application uses an AWS Elastic Load Balancer (ELB) to distribute incoming HTTP traffic across multiple EC2 instances. The load balancer continuously monitors the health of each instance using health checks. If an instance becomes unhealthy, the ELB automatically routes traffic to the remaining healthy instances, ensuring high availability and reliability.

The load balancer is configured to handle HTTP traffic on port 80 and forwards requests to the appropriate instance based on its health status.

2.3 Application Tier

The application tier is responsible for processing user requests and interfacing with the storage and database tiers. This tier is also implemented using Django, which handles the following:

- **Registration:** The application allows new users to create an account, with their information (username, email, and securely hashed password) stored in the AWS RDS MySQL database.
- **User Authentication:** Django's built-in authentication system manages both the registration and login processes, ensuring that only authenticated users can access file management features.
- **File Upload and Retrieval:** The application allows users to upload files, which are then stored in their personal folder on AWS S3. Each file is stored under a path corresponding to the user's username, ensuring clear organization and separation of files.

2.4 Database Tier

The database tier is implemented using AWS RDS (MySQL). This tier stores the following user-related information:

- **User ID:** A unique identifier for each user, automatically generated by the system.
- **Username and Email Address:** These are used for user authentication and communication purposes.
- **Password (hashed):** Passwords are stored in a securely hashed format using Django's authentication mechanisms.

The RDS database is securely configured to only allow access from the application's EC2 instances, ensuring that the data remains protected.

2.5 Storage Tier

The storage tier is implemented using AWS S3. When a user uploads a file, the application uses the Boto3 library to store the file in an S3 bucket. The storage of files is organized as follows:

- Bucket Name: `cis5517-cloud-storage`
- File Path:
`https://cis5517-cloud-storage.s3.us-east-2.amazonaws.com/username/filename`

Each user's files are stored in a directory named after their username, ensuring that their files are logically separated from other users' files. This also makes it easier to list and manage files for each user. The storage tier also handles the retrieval and deletion of files.

3 User Interface

This section describes the different pages in the Cloud Drive Application's user interface. The application allows users to perform actions such as registering, logging in, uploading files, viewing uploaded files, and managing their profiles.

3.1 Home Page (Not Logged In)

Before a user logs in, they are presented with the home page that displays two options: **Login** and **Register**.

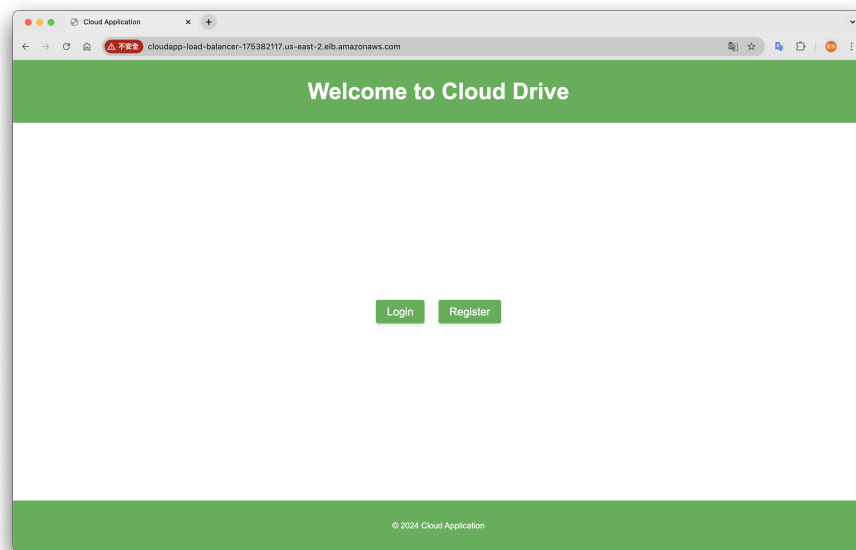


Figure 1: Home Page - Not Logged In

3.2 Login Page

The login page allows users to enter their credentials to access the application.

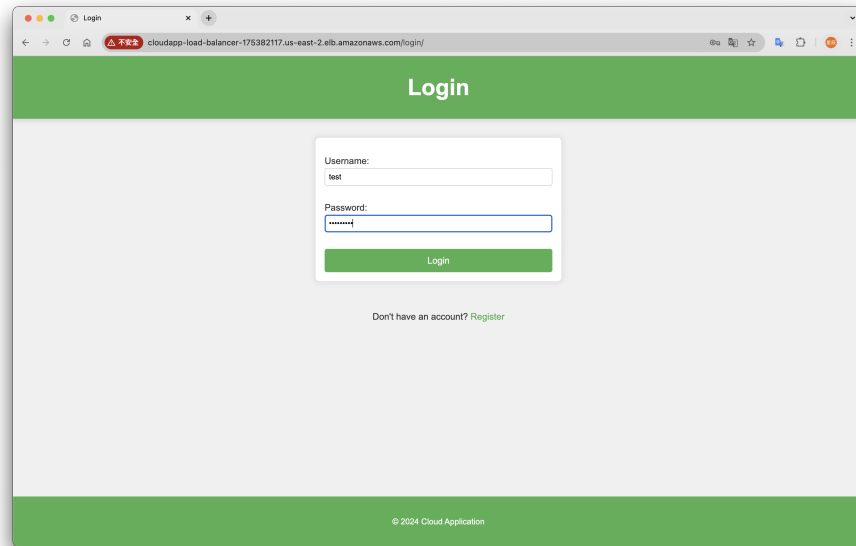
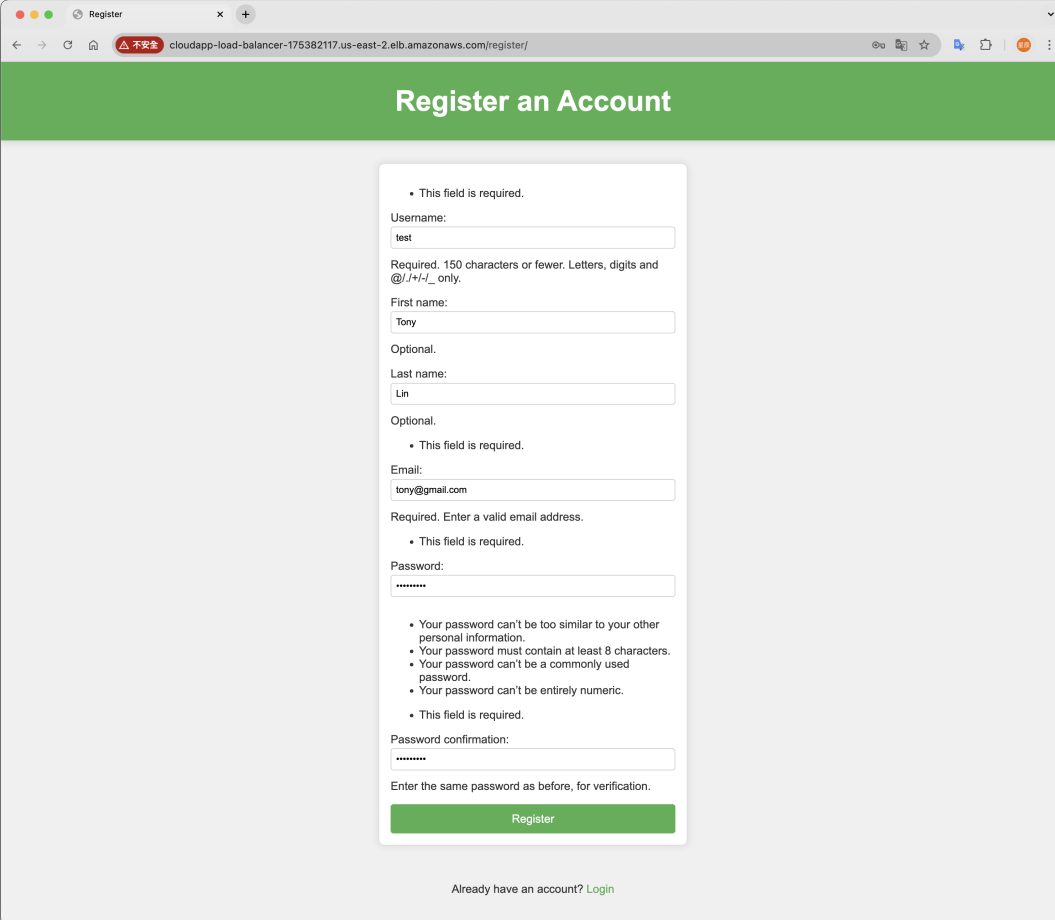


Figure 2: Login Page

3.3 Register Page

New users can create an account by entering a username, email, password, and optional first and last names.



The screenshot shows a web browser window with the title "Register" and the URL "cloudapp-load-balancer-175382117.us-east-2.elb.amazonaws.com/register/". The page has a green header with the text "Register an Account". The main content area is a light gray background with a white registration form in the center. The form contains the following fields and labels:

- Username:** A text input field with the value "test". Above it is a bullet point: "• This field is required." Below it is the text: "Required. 150 characters or fewer. Letters, digits and @/+/!/_ only."
- First name:** A text input field with the value "Tony". Above it is the text: "Optional."
- Last name:** A text input field with the value "Lin". Above it is the text: "Optional."
- Email:** A text input field with the value "tony@gmail.com". Above it is a bullet point: "• This field is required." Below it is the text: "Required. Enter a valid email address."
- Password:** A text input field with masked characters "*****". Above it is a bullet point: "• This field is required." Below it are four bullet points: "• Your password can't be too similar to your other personal information.", "• Your password must contain at least 8 characters.", "• Your password can't be a commonly used password.", and "• Your password can't be entirely numeric."
- Password confirmation:** A text input field with masked characters "*****". Above it is a bullet point: "• This field is required." Below it is the text: "Enter the same password as before, for verification."

At the bottom of the form is a green button labeled "Register". Below the form, centered, is the text "Already have an account? [Login](#)".

Figure 3: Register Page

3.4 Home Page (Logged In)

Once logged in, users are presented with the home page where they can upload files, view and manage their uploaded files, and view their profile information.

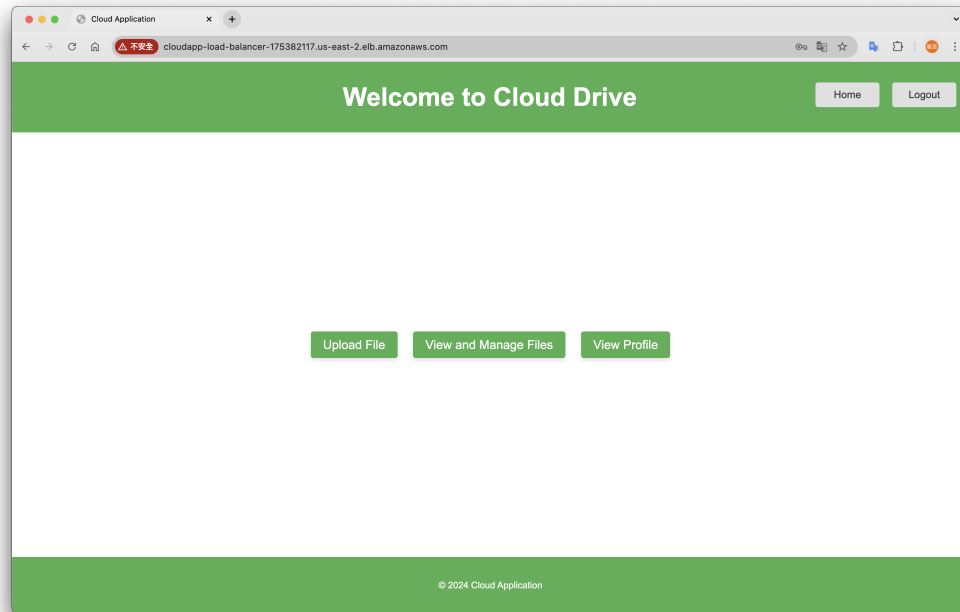


Figure 4: Home Page - Logged In

3.5 Upload File Page

The upload page allows users to choose a file (JPG, PDF, or TXT) and upload it to the cloud storage.

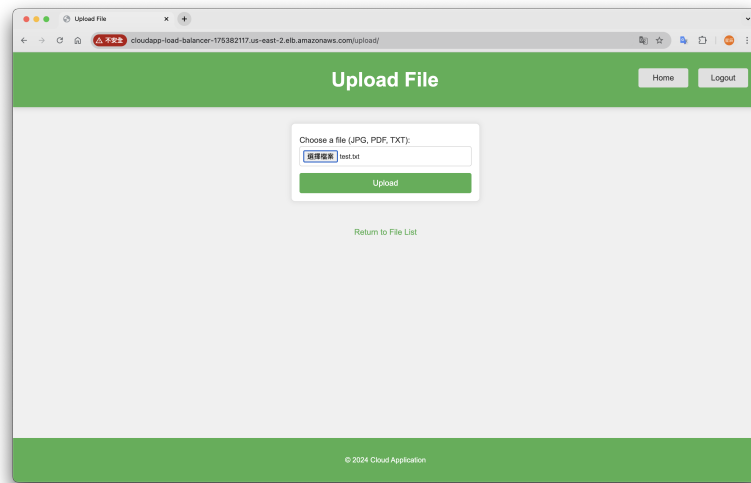


Figure 5: Upload File Page

3.6 File List Page

Users can view a list of their uploaded files on this page, along with the upload date, file size, and an option to delete each file.

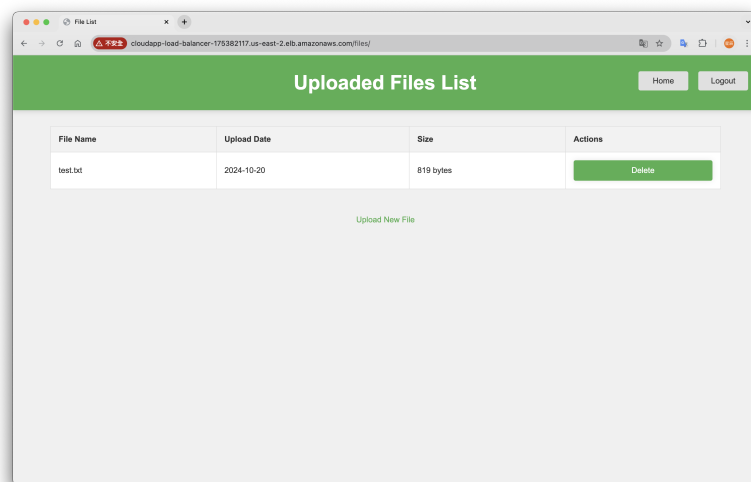


Figure 6: Uploaded Files List Page

3.7 Profile Page

On the profile page, users can view their email, first name, and last name.

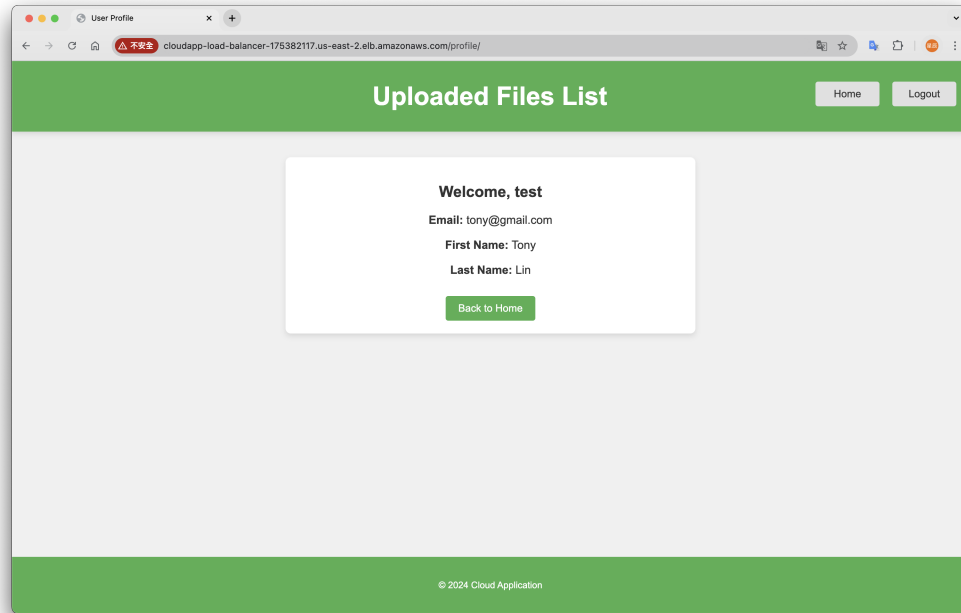


Figure 7: User Profile Page

4 Functionality and Implementation

4.1 User Authentication and Registration

User authentication and registration are implemented using Django's built-in authentication system. When a new user registers, their information, including user-name, email, and password (hashed for security), is stored in the AWS RDS MySQL database.

- User registration is managed by a custom Django form.
- User login is handled by Django's built-in authentication system.
- Upon successful registration or login, the user is redirected to the home page.

Below is the implementation of the user registration and login functionality:

```
# Registration view
def register(request):
    if request.method == 'POST':
        form = CustomUserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            return redirect('home')
    else:
        form = CustomUserCreationForm(request.POST)

    return render(request, 'register.html', {'form': form})

# Login view
def login_view(request):
    return render(request, 'login.html')
```

4.2 File Upload and Management

The application allows users to upload files, which are stored in their dedicated folder on AWS S3. Each user's folder is named after their username, ensuring proper organization of files. The Boto3 library is used to handle communication with AWS S3 for file upload, retrieval, and deletion. Only authenticated users can perform these operations.

- Users can view files through the file management page. The system lists the files under a folder corresponding to the user's username, ensuring each user's files are organized separately. Additionally, the file sizes are dynamically converted from bytes to a more readable format (KB or MB) based on the file size. Here is the core logic for listing files:

```
@login_required
def file_list(request):
    s3 = boto3.client('s3')
    bucket_name = settings.AWS_STORAGE_BUCKET_NAME
    user_directory = request.user.username
    try:
        response = s3.list_objects_v2(Bucket=bucket_name,
            ↪ Prefix=f'{user_directory}/')
        files = response.get('Contents', [])
        for file in files:
```

```

        file['url'] = f"https://{bucket_name}.s3.amazonaws.
        ↪ com/{file['Key']}"
        file['name'] = os.path.basename(file['Key'])
        if file['Size'] < 1000:
            file['size_display'] = f"{file['Size']} bytes"
        elif file['Size'] < 1000000:
            file['size_display'] = f"{file['Size'] /
            ↪ 1000:.2f} KB"
        else:
            file['size_display'] = f"{file['Size'] /
            ↪ 1000000:.2f} MB"
    except ClientError as e:
        files = []
    return render(request, 'file_list.html', {'files': files})

```

- Users can also delete files through the file management page. The system ensures that only files under the folder corresponding to the user's username are deleted. Below is the core logic for deleting files:

```

@login_required
def delete_file(request, file_name):
    s3 = boto3.client('s3')
    bucket_name = settings.AWS_STORAGE_BUCKET_NAME
    user_directory = request.user.username
    if request.method == 'POST':
        try:
            file_key = f'{user_directory}/{file_name}'
            s3.delete_object(Bucket=bucket_name, Key=file_key)
            return redirect('file_list')
        except ClientError as e:
            return HttpResponse("Error deleting file.", status
            ↪ =500)
    return redirect('file_list')

```

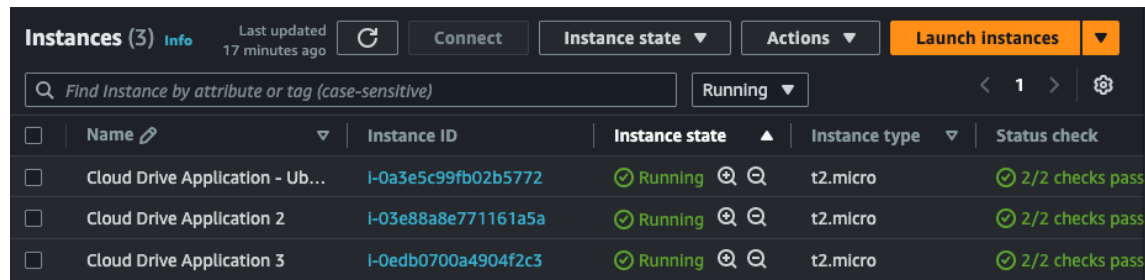
The delete functionality removes the file from the user's personal folder in the S3 bucket by combining the username and the file name.

5 AWS Integration

This application leverages several AWS services for hosting (EC2), traffic distribution (ELB), storage (S3), and database management (RDS). The following sections detail how each service is integrated into the system, including important configurations and screenshots.

5.1 EC2 Instance Setup

The Django application is deployed on three EC2 instances within the same AWS region and VPC. After launching the instances, all required software and libraries, such as Python, Django, and Boto3, are installed, with each instance running the same application code.



The screenshot shows the AWS Management Console 'Instances' page. At the top, there are buttons for 'Connect', 'Instance state', 'Actions', and 'Launch instances'. A search bar is present with the text 'Find Instance by attribute or tag (case-sensitive)'. Below the search bar, there is a table of instances. The table has columns for 'Name', 'Instance ID', 'Instance state', 'Instance type', and 'Status check'. Three instances are listed, all with a 'Running' state and '2/2 checks pass' status.

Name	Instance ID	Instance state	Instance type	Status check
Cloud Drive Application - Ub...	i-0a3e5c99fb02b5772	Running	t2.micro	2/2 checks pass
Cloud Drive Application 2	i-03e88a8e771161a5a	Running	t2.micro	2/2 checks pass
Cloud Drive Application 3	i-0edb0700a4904f2c3	Running	t2.micro	2/2 checks pass

5.2 Elastic Load Balancer

An Elastic Load Balancer (ELB) is used to distribute traffic across three EC2 instances, ensuring availability and fault tolerance. The load balancer monitors the health of the instances and forwards traffic only to the healthy ones. The DNS name for the load balancer is:

- **DNS Name:** `cloudapp-load-balancer-175382117.us-east-2.elb.amazonaws.com`

Key configurations include:

- The load balancer listens on port 80 (HTTP).
- The load balancer is associated with a target group containing the EC2 instances.

cloudapp-TargetGroup

Actions

Details

arn:aws:elasticloadbalancing:us-east-2:345594591702:targetgroup/cloudapp-TargetGroup/21fb90cb4603aa1a

Target type	Protocol : Port	Protocol version	VPC
Instance	HTTP: 80	HTTP1	vpc-0d990f3742293ec86
IP address type	Load balancer		
IPv4	cloudapp-load-balancer		

3	3	0	0	0	0
Total targets	Healthy	Unhealthy	Unused	Initial	Draining
	0 Anomalous				

Distribution of targets by Availability Zone (AZ)

Select values in this table to see corresponding filters applied to the Registered targets table below.

Targets
Monitoring
Health checks
Attributes
Tags

Registered targets (3)

Anomaly mitigation: Not applicable

Deregister

Register targets

Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registered targets according to the target group's health check settings. Anomaly detection is automatically applied to HTTP/HTTPS target groups with at least 3 healthy targets.

Filter targets

1

	Instance ID	Name	Port	Zone	Health status	Health status details
<input type="checkbox"/>	i-0edb0700a4904f2c3	Cloud Drive Ap...	8000	us-east-2b	Healthy	-
<input type="checkbox"/>	i-03e88a8e771161a5a	Cloud Drive Ap...	8000	us-east-2b	Healthy	-
<input type="checkbox"/>	i-0a3e5c99fb02b5772	Cloud Drive Ap...	8000	us-east-2b	Healthy	-

Figure 8: Load Balancer Target Group Configuration

- Health checks are configured to ping the root path / on each instance to ensure it is functioning properly.

Targets	Monitoring	Health checks	Attributes	Tags
Health check settings				Edit
Protocol HTTP	Path /	Port Traffic port	Healthy threshold 5 consecutive health check successes	
Unhealthy threshold 2 consecutive health check failures	Timeout 5 seconds	Interval 30 seconds	Success codes 200	

Figure 9: Health Check Configuration

5.3 AWS S3 (Storage)

AWS S3 is used to store user-uploaded files. Each user has their own folder, organized under the corresponding username, and files are uploaded to S3 using the Boto3 library. Key configurations include:

- The IAM role attached to the EC2 instances must have permissions to read and write to the S3 bucket.
- Files are uploaded under the directory structure `cloudapp-bucket/username/filename`.

Here is a screenshot of the S3 bucket structure, showing files organized by username:

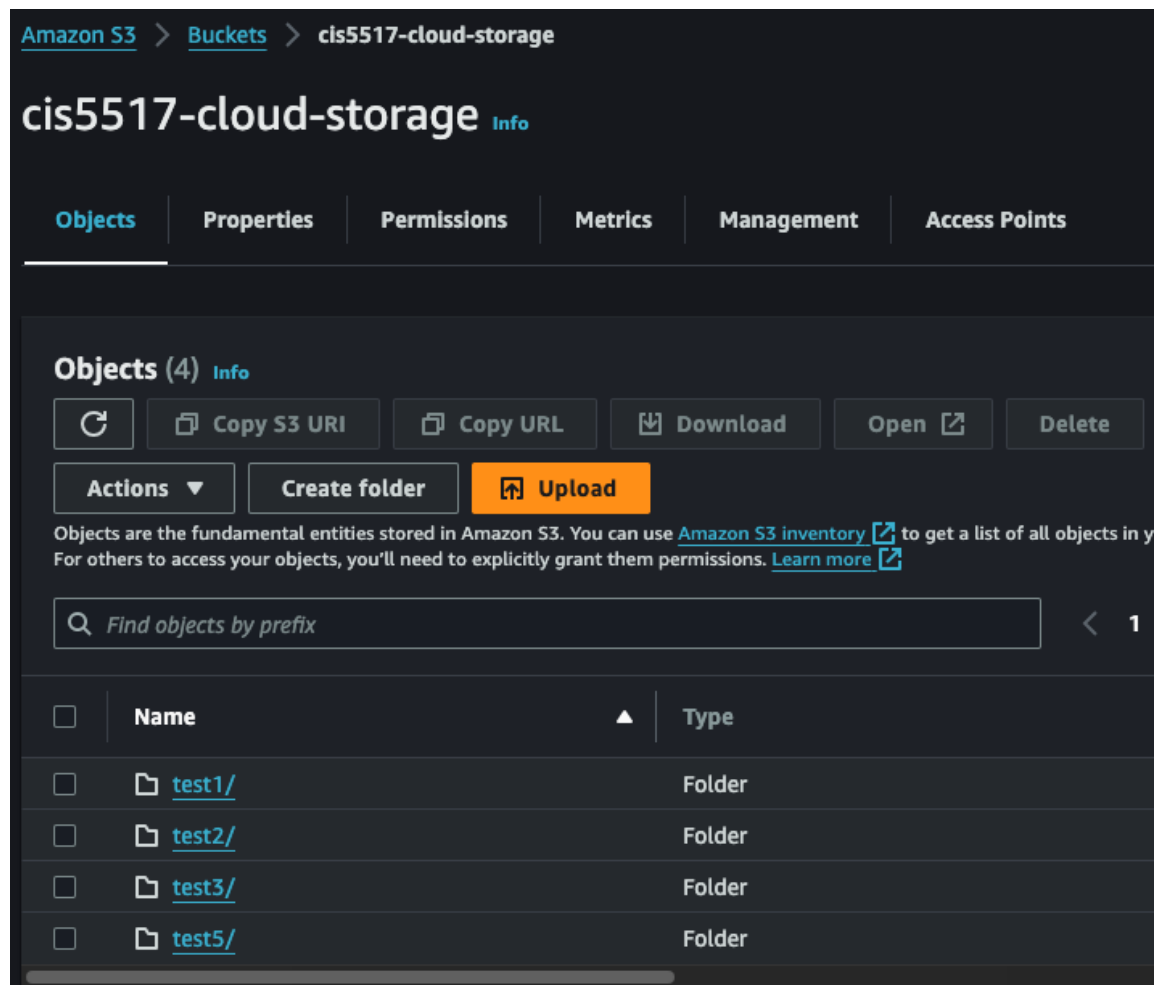


Figure 10: S3 Bucket with User Folders

5.4 AWS RDS (MySQL)

User information, including usernames, emails, and hashed passwords, is stored in an AWS RDS MySQL database. The RDS database is integrated with the Django application, and the following configurations are key:

- The RDS instance is hosted within the same VPC as the EC2 instances to ensure secure and efficient communication. Additionally, the RDS instance is associated with a Security Group that controls network access.

- Inbound rules in the RDS Security Group are configured to allow MySQL traffic (port 3306) from the subnet CIDR range of the EC2 instances. This ensures that only the EC2 instances within the same VPC can access the RDS database.

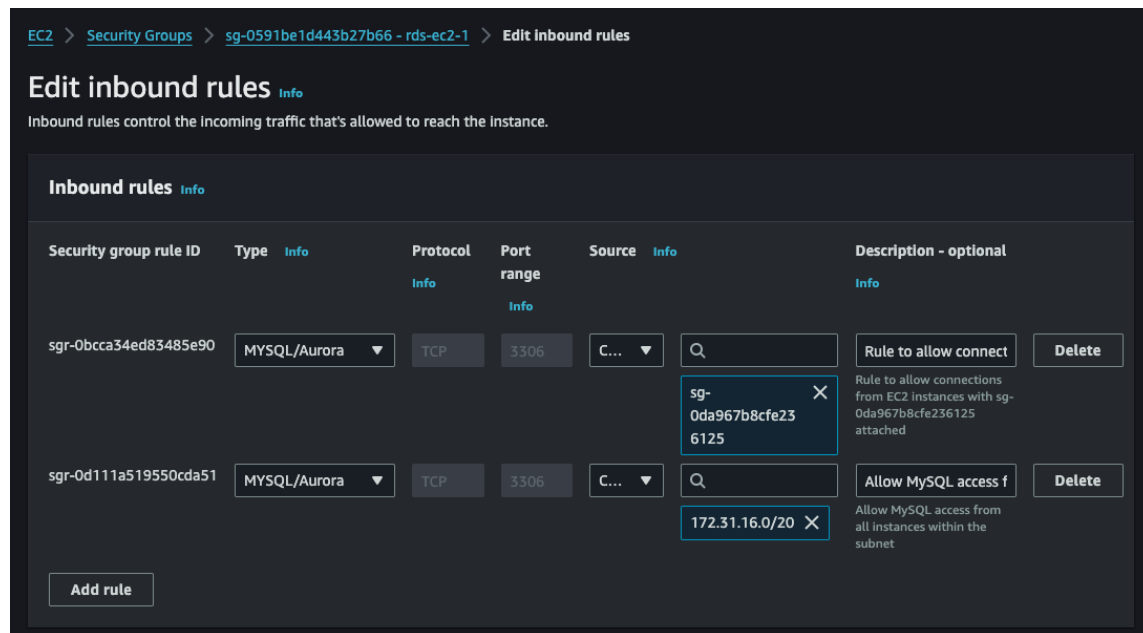


Figure 11: RDS Security Group configuration allowing EC2 instance traffic

- Database connection settings are managed in Django's `settings.py` file, which includes the database endpoint, username, and password. Here is an example of the connection configuration:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'cloudapp_db',
        'USER': 'db_user',
        'PASSWORD': 'db_password',
        'HOST': 'cloudapp-db-instance.rds.amazonaws.com',
        'PORT': '3306',
    }
}
```

6 Conclusion

In conclusion, the Cloud Drive Application successfully demonstrates the integration of multiple AWS services to provide a scalable, secure, and efficient cloud-based file storage solution. By leveraging EC2 instances for hosting, an Elastic Load Balancer (ELB) for traffic distribution, S3 for file storage, and RDS MySQL for managing user information, the application ensures high availability, fault tolerance, and data security.

The use of Django as the application framework simplifies user authentication, file management, and database interactions, while the AWS infrastructure supports seamless scalability. The architecture ensures that each user's files are stored securely in their respective S3 folder, with access restricted to authorized users only. Overall, the project showcases how cloud services can be combined to build a reliable web application for managing files in the cloud.