

System Design Document

This is broken down into various tasks for ease of use.

Task 1: Write a summary of each service: AWS services: EC2 (Elastic Compute Cloud), S3 (Simple Storage Service), RDS (Relational Database Service), and CloudFormation. Note: explaining its purpose, key features, and benefits.

- **Amazon EC2 (Elastic Compute Cloud):**

Purpose: Amazon EC2 is a web service that provides elastic compute capacity in the cloud. It allows users to run virtual servers (known as instances) to host applications and services.

Key Features:

Scalability: Easily scale computing resources up or down based on demand.

Variety of Instances: Offers a wide selection of instance types optimized for different use cases.

The cost associated with those instances depends upon various factors including the size and type.

Custom AMIs (Amazon Machine Images): Create custom images with pre-configured software and settings.

Benefits:

Flexibility: EC2 instances offer flexibility and control over computing resources.

Cost-Efficiency: Pay only for the compute capacity you use, with options for on-demand, reserved, or spot instances.

Global Reach: Deploy instances in multiple regions for high availability and low-latency access.

Instant availability: The instances are deployed with a single click providing you instant availability.

- **Amazon S3 (Simple Storage Service):**

Purpose: Amazon S3 is an object storage service designed to store and retrieve any amount of data from anywhere on the web. It provides a highly durable and available storage infrastructure.

Key Features:

Scalability: Scales automatically to handle varying amounts of data.

Data Lifecycle Management: Define rules to transition objects between storage classes.

Security and Compliance: Provides features like access control, encryption, and compliance capabilities.

Benefits:

Durability: Designed for 99.999999999% (11 nines) durability of objects over a given year.

Accessibility: Access data from anywhere on the web through simple HTTP/HTTPS interfaces.

Versatility: Suitable for a wide range of use cases, from backup and restore to big data analytics.

- **Amazon RDS (Relational Database Service):**

Purpose: Amazon RDS is a managed relational database service that simplifies database setup, operation, and scaling. It supports multiple database engines, including MySQL, PostgreSQL, Oracle, SQL Server, and MariaDB.

Key Features:

Automated Backups: Automatically backs up databases and allows point-in-time recovery.

Scalability: Easily scale compute and storage resources as needed.

Multi-AZ Deployment: Provides high availability by replicating databases across multiple Availability Zones.

Benefits:

Managed Service: AWS takes care of database administration tasks, allowing users to focus on application development.

Security: Offers features like encryption at rest and in transit, IAM roles for database access, and network isolation.

Compatibility: Supports popular database engines, making it easy to migrate existing applications.

- **AWS CloudFormation:**

Purpose: AWS CloudFormation is a service that enables users to define and provision AWS infrastructure as code. It allows the creation and management of AWS resources using a declarative template.

Key Features:

Infrastructure as Code (IaC): Define and provision AWS infrastructure using JSON or YAML templates.

Stack Management: Create, update, and delete a collection of resources as a single unit, known as a stack.

Template Designer: Visual tool for creating, viewing, and modifying CloudFormation templates.

Benefits:

Consistency: Ensure consistent and repeatable resource deployments.

Automation: Automate the creation and management of AWS resources.

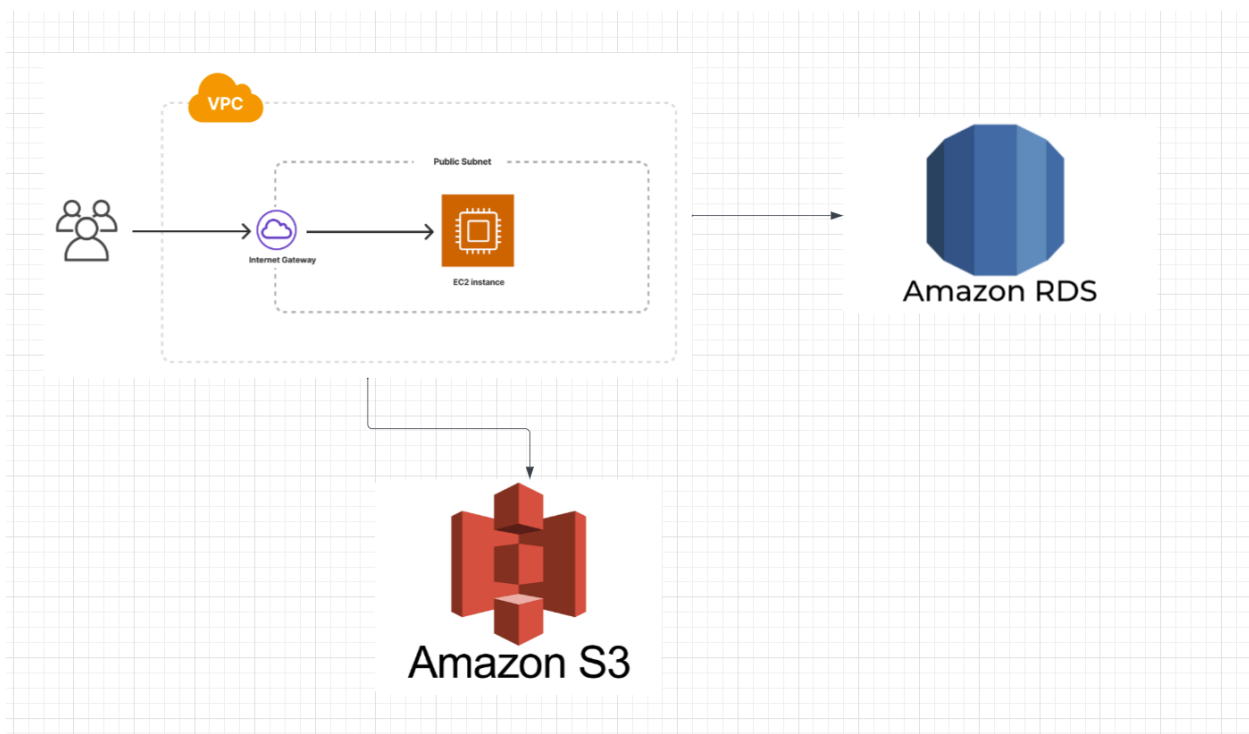
Version Control: Templates can be version-controlled, providing a history of changes.

Task 2: Create a scenario where you need to deploy a web application on AWS. Consider the following requirements: a. The application should be scalable to handle high traffic. b. The application requires a relational database for storing user data. c. The application needs to store and retrieve large files. d. The infrastructure should be managed and provisioned using code.

- **Scenario:** Grand File sharing application. This application is a file-sharing application where any user can upload files and they can download existing files uploaded by the community.
- **Scalability for High Traffic:** The application is deployed on the EC2 instance. The EC2 instance is vertically scalable to handle high traffic scenarios.
- **Relational Database for User Data:** Amazon RDS (Relational Database Service): Set up an RDS instance (MySQL) to store user file upload data. This managed service simplifies database administration tasks and provides scalability.
- **Storage and Retrieval of Large Files:** Amazon S3 (Simple Storage Service): Store large files in S3, which is designed for scalable and secure object storage. Flask-S3 extension: Integrate Flask-S3 extension to interact with S3 directly from your Flask application. This allows for seamless file uploads and downloads.
- **Infrastructure as Code (IaC):** AWS CloudFormation: Define the entire infrastructure as code using CloudFormation templates. This includes EC2 setup, RDS settings, and S3 bucket setup.

Task 3: Based on the scenario, design and implement the architecture using AWS Compute, Storage, Database, and Infrastructure Management Services.

Architecture Diagram:

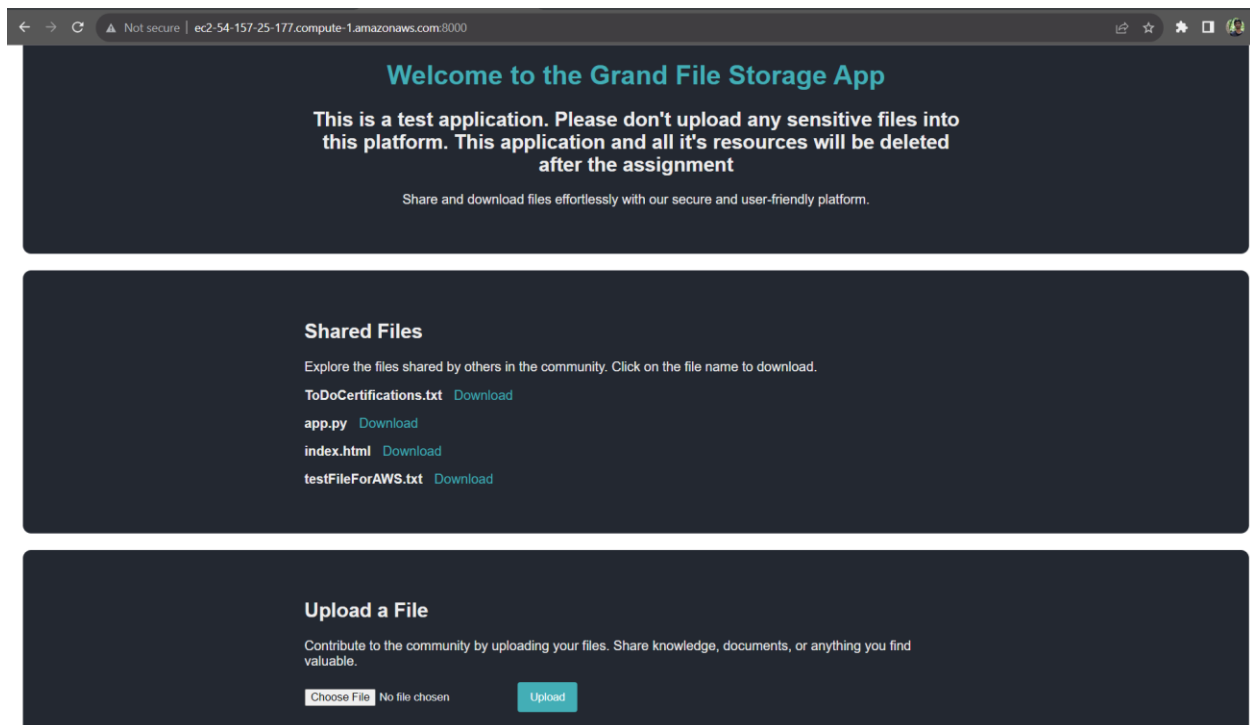


Components:

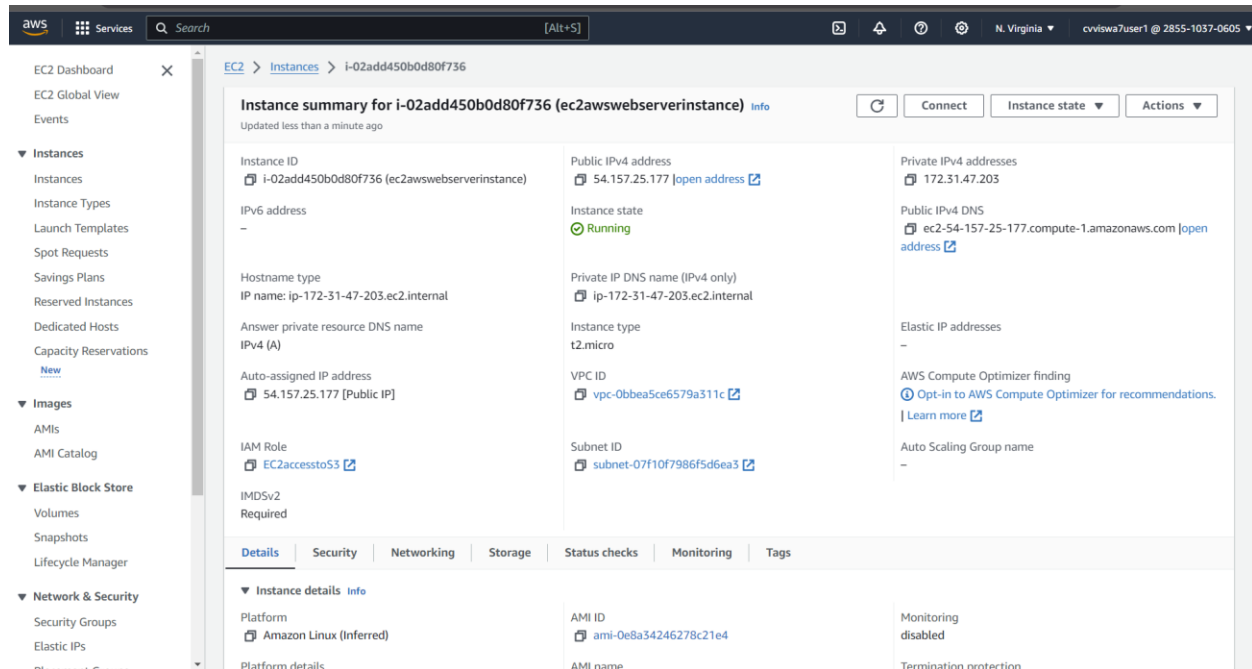
- The Ec2 instance hosts the web application. It has a public IP address to which anyone can connect on port 8000, to access the web application and interact with it. They can upload files to the community or download the existing files uploaded by the community. The web application uses the Python Flask for it's backend and HTML, CSS for it's frontend.
- Amazon S3 houses the files uploaded by the users through the application. The EC2 instance connects with Amazon S3 and stores the files in here. The connection between the EC2 instance and the storage is established using the boto3 library.
- We have an RDS instance. Here, we use the MySQL DBMS on the RDS. The metadata of the files uploaded by the users is safely stored on this RDS instance. The EC2 instance connects with the RDS using mysql-connector.

User Interface Front-end:

- Users can connect to the application on <http://54.157.25.177:8000/>
This IP address is subjected to change whenever the EC2 instance is stopped and relaunched, since the IP address would be released on every stop. However, the port on which the web app runs is always 8000.
- The users can then see the list of available files and download them. They can also upload their files to the storage service connected to the backend of this application.



Task 4: Launch an EC2 instance with the appropriate configuration to host your web application. Choose an Amazon Machine Image (AMI) that suits your application's requirements.



EC2 instance template:

AWSTemplateFormatVersion: "2010-09-09"

Parameters:

InstanceName:

Type: String

Description: Name for the EC2 instance

Default: ec2awswebserverinstance

Resources:

EC2Instance:

Type: AWS::EC2::Instance

Properties:

InstanceType: t2.micro

SecurityGroupIds:

- launch-wizard-1

- ec2-rds-2

KeyName: mykeypair

ImageId: ami-0c55b159cbf1f0 # Amazon Linux 2 AMI

IamInstanceProfile: EC2accesstoS3

Tags:

- Key: Name

Value: !Ref InstanceName

NetworkInterfaces:

- AssociatePublicIpAddress: true

DeviceIndex: 0

SubnetId: subnet-07f10f7986f5d6ea3

Task 5: Set up an S3 bucket to store the large files used by your web application. Configure appropriate permissions to ensure secure access.

S3:

The screenshot displays the Amazon S3 console interface. On the left, a sidebar lists various S3 services and tools. The main panel shows the 'Buckets' section, including an account snapshot and a list of existing buckets. The bucket 's3awsstorageinstance' is listed with its region, access settings, and creation date.

Name	AWS Region	Access	Creation date
s3awsstorageinstance	US East (N. Virginia) us-east-1	Objects can be public	November 17, 2023, 22:36:15 (UTC-05:00)

The application has `AWS_ACCESS_KEY` and `AWS_SECRET_KEY` configured. These credentials are used to securely access the S3 using the IAM user credentials.

```
# AWS S3 credentials
AWS_ACCESS_KEY = [REDACTED]
AWS_SECRET_KEY = [REDACTED]
S3_BUCKET = 's3awsstorageinstance'
```

The IAM user has the necessary permissions to operate on the S3:

cvviswa7user1

Info

Delete

Summary

ARN

arn:aws:iam::285510370605:user/cvviswa7user1

Created

October 23, 2023, 14:44 (UTC-04:00)

Console access

Enabled without MFA

Last console sign-in

Today

Access key 1

Used 20 hours ago. 30 days old.

Access key 2

Used today. 2 days old.

Permissions

Groups (1)

Tags (2)

Security credentials

Access Advisor

Permissions policies (3)

Permissions are defined by policies attached to the user directly or through groups.

Search

Filter by Type

All types


< 1 >

<input type="checkbox"/>	Policy name	Type	Attached via
<input type="checkbox"/>	AdministratorAccess	AWS managed - job function	Group bigdataaawsgroup
<input checked="" type="checkbox"/>	AmazonS3FullAccess	AWS managed	Group bigdataaawsgroup
<input type="checkbox"/>	s3ManagementUser	Customer inline	Inline

AmazonS3FullAccess [Info](#)

Provides full access to all buckets via the AWS Management Console.

Policy details

Type AWS managed	Creation time February 06, 2015, 13:40 (UTC-05:00)	Edited time September 27, 2021, 16:16 (UTC-04:00)	ARN  arn:aws:iam::aws:policy/AmazonS3FullAccess
---------------------	---	--	---

[Permissions](#) | [Entities attached](#) | [Policy versions \(2\)](#) | [Access Advisor](#)

Permissions defined in this policy [Info](#)

[Summary](#) [JSON](#)

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

Allow (2 of 386 services)

☐ Show remaining 384 services

Service	Access level	Resource	Request condition
S3	Full access	All resources	None
S3 Object Lambda	Full access	All resources	None

S3 instance template:

AWSTemplateFormatVersion: "2010-09-09"

Parameters:

BucketName:

Type: String

Description: Name for the S3 bucket

Default: s3awsstorageinstance

Resources:

S3Bucket:

Type: AWS::S3::Bucket

Properties:

BucketName: !Ref BucketName

AccessControl: Private

PublicAccessBlockConfiguration:

BlockPublicAcls: false

IgnorePublicAcls: false

BlockPublicPolicy: false

RestrictPublicBuckets: false

Task 6: Use CloudFormation to create a VPC Using CloudFormation.

The screenshot shows the AWS Management Console interface for a VPC. On the left is a navigation sidebar with categories like 'Virtual private cloud', 'Security', and 'DNS firewall'. The main content area shows the 'vpc-0bbea5ce6579a311c' details. Below the details is a 'Resource map' section with tabs for 'Resource map', 'CIDRs', 'Flow logs', 'Tags', and 'Integrations'. The 'Resource map' tab is active, showing a diagram of the VPC's resources and their connections.

Details			
VPC ID vpc-0bbea5ce6579a311c	State Available	DNS hostnames Enabled	DNS resolution Enabled
Tenancy Default	DHCP option set dopt-04a8e70a47e6f6c6e	Main route table rtb-00590004b1ba86e9f	Main network ACL acl-0c65620b658dc5e81
Default VPC Yes	IPv4 CIDR 172.31.0.0/16	IPv6 pool -	IPv6 CIDR (Network border group) -
Network Address Usage metrics Disabled	Route 53 Resolver DNS Firewall rule groups -	Owner ID 285510370605	

Resource map

- VPC: vpc-0bbea5ce6579a311c
- Subnets (11): us-east-1a, subnet-077bcf0eaf27d531b, us-east-1b
- Route tables (2): rtb-00590004b1ba86e9f, RDS-Pvt-rt
- Network ACLs: acl-0c65620b658dc5e81
- Internet gateways: ig

VPC template:

AWSTemplateFormatVersion: '2010-09-09'

Parameters:

VpcId:

Type: String

Description: VPC ID

Default: vpc-0bbea5ce6579a311c

DhcpOptionSetId:

Type: String

Description: DHCP option set ID

Default: dopt-04a8e70a47e6f6c6e

DnsHostnames:

Type: String

Description: Enable DNS hostnames

Default: Enabled

DnsResolution:

Type: String

Description: Enable DNS resolution

Default: Enabled

Tenancy:

Type: String

Description: VPC Tenancy

Default: Default

Resources:

MyVPC:

Type: AWS::EC2::VPC

Properties:

CidrBlock: 172.31.0.0/16

VpcId: !Ref VpcId

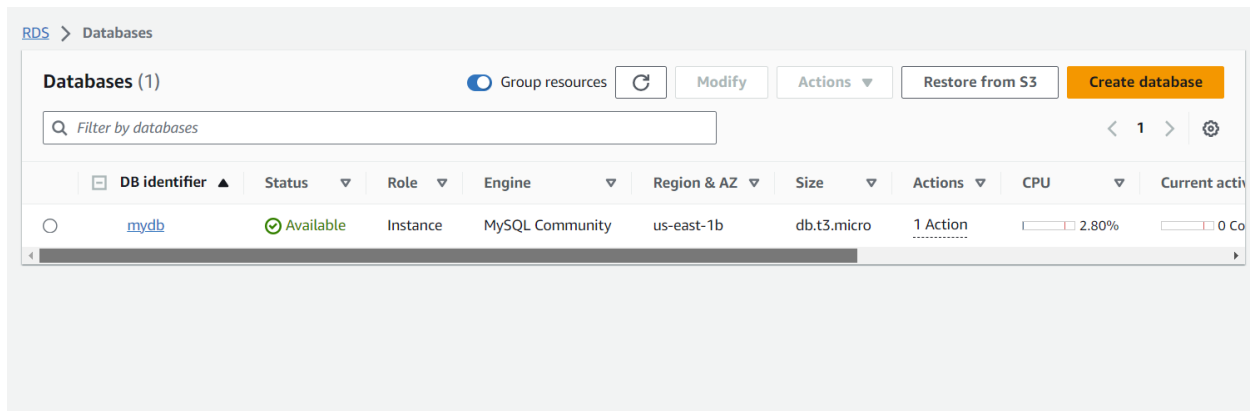
EnableDnsHostnames: !If [IsDNSHostnamesEnabled, true, false]

EnableDnsSupport: !If [IsDNSResolutionEnabled, true, false]

InstanceTenancy: !Ref Tenancy

DhcpOptionsId: !Ref DhcpOptionSetId

RDS using this VPC:



RDS template:

AWSTemplateFormatVersion: '2010-09-09'

Parameters:

DBInstanceName:

Type: String

Description: RDS instance name

Default: mydb

DBEngine:

Type: String

Description: RDS database engine

Default: MySQL

DBInstanceClass:

Type: String

Description: RDS instance size

Default: db.t3.micro

VPCId:

Type: String

Description: VPC ID

Default: vpc-0bbea5ce6579a311c

SubnetId:

Type: String

Description: Subnet ID

Default: subnet-07f10f7986f5d6ea3

Region:

Type: String

Description: AWS region

Default: us-east-1b

Resources:

MyDBInstance:

Type: AWS::RDS::DBInstance

Properties:

DBInstanceIdentifier: !Ref DBInstanceName

Engine: !Ref DBEngine

DBInstanceClass: !Ref DBInstanceClass

AllocatedStorage: 20

VPCSecurityGroups:

- sg-0123456789abcdef0

AvailabilityZone: !Ref Region

MultiAZ: false

PubliclyAccessible: true

StorageType: gp2

MasterUsername: admin

MasterUserPassword: adminpassword

VPCSecurityGroups:

- sg-0123456789abcdef0

DBSubnetGroupName: !Sub MyDBSubnetGroup

Tags:

- Key: Name

Value: !Ref DBInstanceName

MyDBSubnetGroup:

Type: AWS::RDS::DBSubnetGroup

Properties:

DBSubnetGroupName: !Sub MyDBSubnetGroup

DBSubnetGroupDescription: Subnet group for RDS

SubnetIds:

- !Ref SubnetId

Task 7: Deploy your web application on the EC2 instance. Ensure that it can access and utilize the storage services (S3) and the database service (RDS).

- The web application code is uploaded to the Canvas. It's in app.py file. The frontend HTML and CSS files are in the static folder named index.html and style.css respectively.
- Ensuring that EC2 can access the RDS instance. Below is the screenshot which shows that I can successfully connect to my RDS instance and check the available databases using "SHOW DATABASES;" command.

Command used: `mysql -h mydb.cu0cmwngbm5d.us-east-1.rds.amazonaws.com -u admin -p -P 3306`

```
[ec2-user@ip-172-31-47-203 mywebapp]$ mysql -h mydb.cu0cmwngbm5d.us-east-1.rds.amazonaws.com -u admin -p -P 3306
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 720
Server version: 8.0.33 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| seproject |
| sys |
+-----+
5 rows in set (0.02 sec)
```

- Ensuring that EC2 can access the S3 instance. Below is the screenshot which shows that I can successfully connect to my S3 instance and view the objects uploaded in the s3 bucket.

Command used: `aws s3 ls s3://s3awsstorageinstance/`

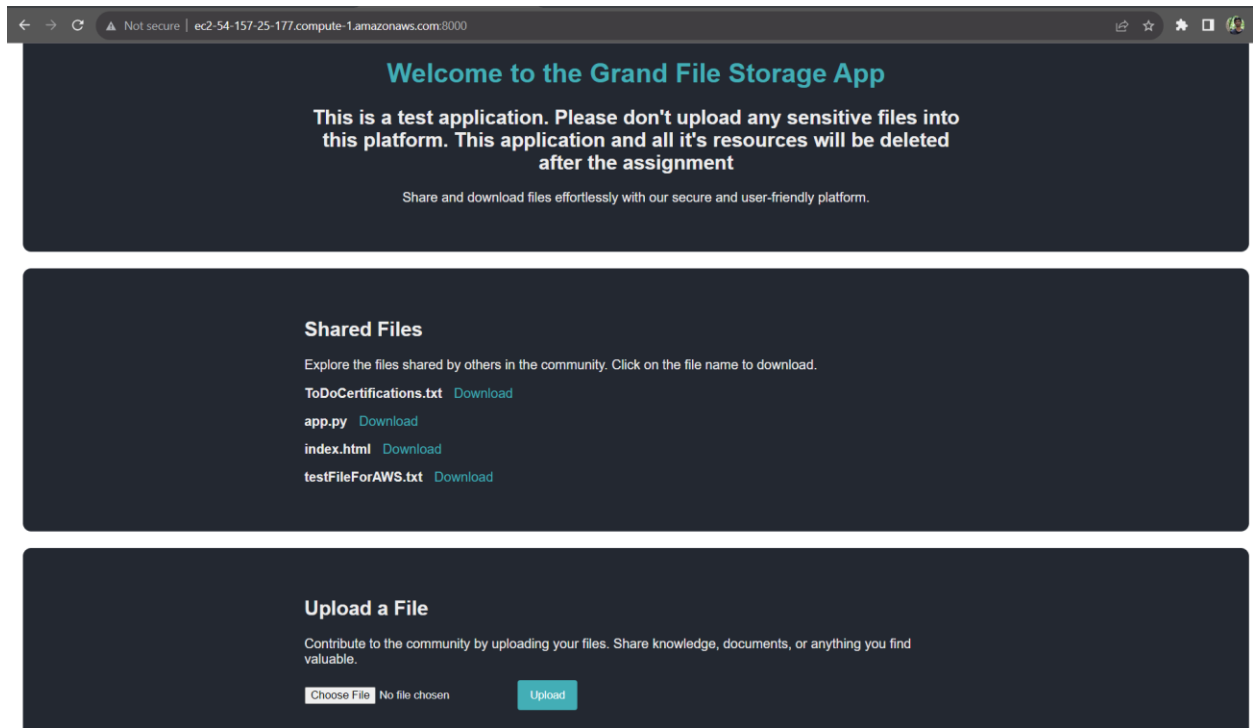
```
[ec2-user@ip-172-31-47-203 mywebapp]$ aws s3 ls s3://s3awsstorageinstance/
2023-11-22 15:40:33      PRE test/
2023-11-22 15:51:59      122 ToDoCertifications.txt
2023-11-22 17:37:41      2017 app.py
2023-11-22 00:22:33      2536 index.html
2023-11-22 17:49:23      973 style.css
2023-11-22 17:49:23      17 testFileForAWS.txt
```

Task 8: Test the functionality of your web application, ensuring that it can handle high traffic and efficiently retrieve and store large files and data in the database.

- Below code shows the connection details of the web application launched on the EC2 instance with RDS and S3 via web application.

[illegible]

- As demonstrated, The “Shared Files” section below connects to the RDS instance to retrieve all the user file details and display in the front-end for the users to download. The “Upload a File” section connects to the S3 instance in the backend to upload a file from the user.



Task 9: Write a report documenting the architecture design, configuration details, and the steps taken to implement and test the web application on AWS. Explain the benefits and advantages of using AWS Compute, Storage, Database, and Infrastructure Management Services for your scenario.

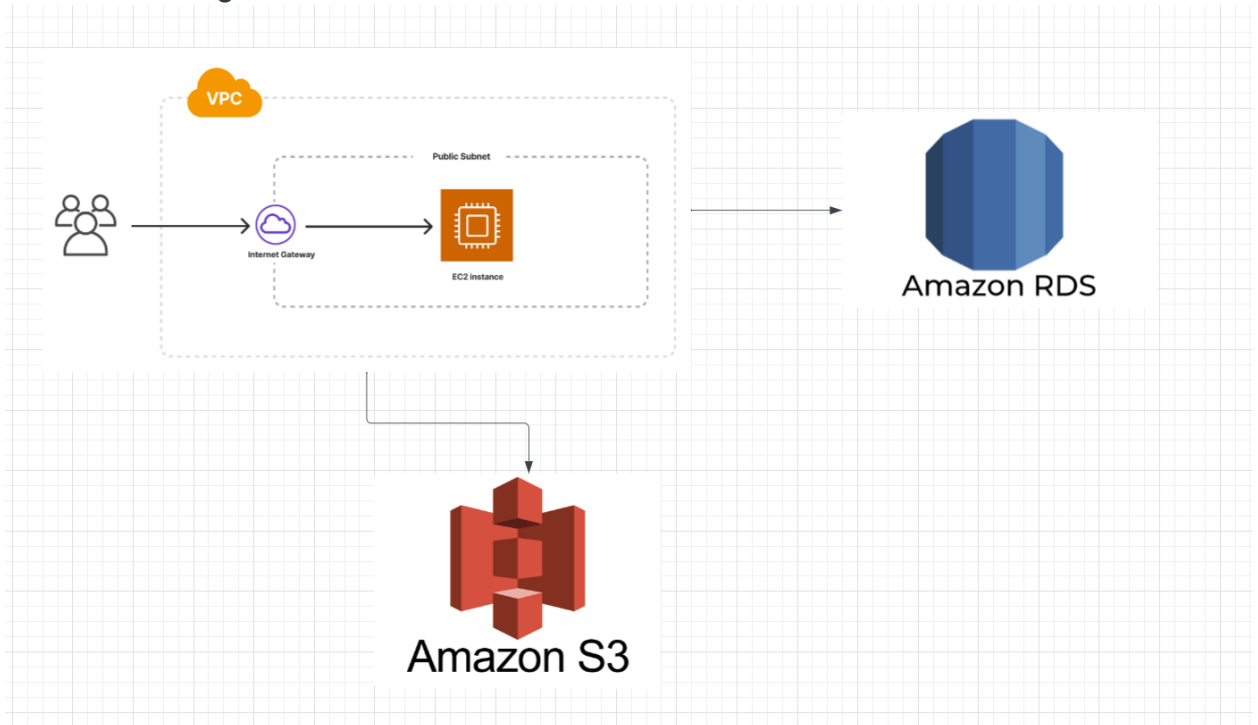
Title: *Grand File Sharing Application on AWS - Architecture Design and Implementation Report*

- **Introduction:**

The Grand File Sharing Application is a robust and user-friendly web application designed for efficient file sharing. Leveraging AWS services, this report outlines the architecture design, configuration details, and steps taken to implement and test the application.

-

Architecture Design:



The architecture is divided into key components:

Frontend:

HTML, CSS, and Jinja2 for the user interface.
unicorn for hosting the web application.

Backend:

Python with Flask for the server-side logic.
AWS S3 SDK for file uploads and downloads.
AWS RDS (Relational Database Service) for storing user data.

- **Configuration Details:** Detailed summary of configurations has been shared in the previous tasks

Amazon EC2:

Created an EC2 instance and deployed the web application.
Created the VPC for the EC2 instance.

Configured the security group and access to the EC2 instance.

AWS S3:

Created an S3 bucket for storing files.

Configured permissions for secure file access.

Utilized AWS SDK for Python (Boto3) for seamless integration.

AWS RDS:

Provisioned an RDS instance with MySQL as the database engine.

Configured security groups for controlled access.

Used mysql-connector for database interactions within the Flask application.

- **Implementation Steps:** Implementation steps are shared in the previous task along with the configuration files.

- **Testing:**
Conducted extensive testing on the application:
Uploading and downloading files of various sizes.
Verifying data integrity in the MySQL database.
Load testing to ensure scalability.

- **Benefits and Advantages of AWS Services:**

EC2:

Provides the infrastructure for a virtual machine.

Easily scalable.

Single-click setup.

Storage (S3):

Provides scalable and durable object storage.

Object life cycle management

Securely stores large files with fine-grained access control.

Database (RDS):

Managed relational database service with automated backups.

Scalable and highly available architecture.

Infrastructure Management:

Infrastructure as Code (IaC) with AWS CloudFormation for reproducible deployments.

Simplifies resource management and provisioning.

Conclusion:

The Grand File Sharing Application demonstrates the power and flexibility of AWS services. Leveraging EC2, S3, and RDS, the application achieves scalability, reliability, and efficient storage.

AWS's managed services streamline deployment, enhance security, and provide a robust foundation for future enhancements.

Task 10: Reflect on the challenges and considerations encountered during the implementation process and discuss how AWS services addressed those challenges.

Below are few notable challenges and considerations in Grand File Sharing App Implementation:

- **Scalability:**
Challenge: Managing the compute instance
AWS Solution: EC2 can be scaled vertically as and when the load increases.
- **File Storage and Retrieval:**
Challenge: Managing large file uploads and downloads efficiently.
AWS Solution: Utilizing AWS S3 for scalable and durable object storage. S3's low-latency retrieval and high-throughput capabilities ensure efficient handling of large files.
- **Database Management:**
Challenge: Setting up and managing a relational database for user data.
AWS Solution: Employing AWS RDS with MySQL for a managed database service. This simplifies database provisioning, maintenance, and backups, ensuring data integrity and availability.
- **Security:**
Challenge: Ensuring secure access to files and user data.
AWS Solution: Configuring IAM roles, S3 bucket policies, and RDS security groups for fine-grained access control. AWS's security features help in implementing the principle of least privilege.
- **Infrastructure Provisioning:**
Challenge: Efficiently managing and provisioning infrastructure resources.
AWS Solution: Implementing Infrastructure as Code (IaC) using AWS CloudFormation. This allows for consistent and repeatable infrastructure deployments, reducing the risk of configuration errors.
- **Monitoring and Optimization:**
Challenge: Ensuring optimal performance and monitoring application health.

AWS Solution: Implementing AWS CloudWatch for monitoring key metrics. CloudWatch provides insights into application performance, helping in identifying and addressing issues proactively.

- **Cost Management:**

Challenge: Optimizing costs associated with infrastructure resources.

AWS Solution: Leveraging AWS's pay-as-you-go model and using services like S3, which automatically adjusts resources based on demand. This helps in cost efficiency by avoiding over-provisioning.