ABSTRACT

Cloud computing is the on demand availability of computer system resources, especially data storage and computing power, without direct active management by the user. Storage and Management of files in the cloud is an important problem to solve because in public cloud, the data is accessible via the internet. Secure authentication and authorization is essential in managing files. Hence the mini-project was implemented which takes the availability and durability of S3 to store files and modern encryption techniques to encrypt those files and store securely. Amazon EC2 web server provides reliable compute infrastructure to manage the web server.

Problem Statement:

Build and deploy application to upload and download file on/from public cloud in encrypted form.

Theory:

I. Amazon EC2

1. Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.
2. Amazon Elastic Compute Cloud (EC2) is Infrastructure as a Service.
3. Benefits of using Amazon EC2 :
    a. Quickly scale up and down
    b. Broad Compatibility (Servers and other IT resources of various configurations)
    c. No need of physical IT resource management.
    d. Cheap due to economies of scale.

II. Amazon Simple Storage Service

1. Amazon S3 is an Object based Store. It is used to store static files like images, videos, datasets etc.
2. Data is spread across multiple devices and facilities to ensure durability
3. 99.9% SLA for availability of files
4. Replication of files across multiple zones.
5. Virtually unlimited storage
6. Files are stored in buckets
7. Benefits of Amazon S3 :
    a. Highly durable: Data is replicated across at least two zones.
    b. High Availability: The data is highly available with service-level agreement.
    c. Allows flexible querying of data stored in S3.

III. Amazon DynamoDB

1. Amazon DynamoDB is a fully managed NoSQL store.
2. Supports Document or Key-Value Structures.
3. Durable and Highly Available.

Working:

The program is a web application which is used to host static files. The web application provides user interface to upload local files to the cloud to make it accessible from anywhere.
Files are organized into folders. The files are uploaded to the EC2 instance and are encrypted. These encrypted files are then stored in the S3 bucket. Amazon Simple Storage Service allows to store multiple static files of varying sizes with high durability and availability.
The user can then download files from the web interface provided and served with Amazon EC2 server. The files are decrypted on the server and then returned to the user.

Implementation:

The web server and the backend code which does the processing is implemented in Python and Flask Micro Framework. Flask provides a simple and easy to use APIs for web development. In handles web socket creation and Web Server Gateway Interface. Flask simplifies web development by using web routes through decorators. The flask server serves the static files to the user and accepts static files uploaded by the user as well as handle user management and encryption/decryption.

DynamoDB was used to store the user login information. DynamoDB provides highly scalable document and key-value store. Details like email, the name of the user and password hash of the user were stored in a dynamodb table in the AWS Mumbai Region. Bcrypt library was used in python to hash the password as well as verify passwords.

To connect Python backend with a dynamoDB table, the boto3 library is used. The Boto3 library provides a programmatic interface to AWS services in python.
Information in boto3 is transferred using the JSON format.
We have implemented interfaces for the following in S3 with boto3:

- Adding a user
- Loading a user

These interfaces can be found in “models.py” file in user folder.
The user management was done in flask using decorators to protect routes which require user login. A custom decorator called “login_required” was used to protect routes which require user logins.

To connect Python backend with S3 bucket, the boto3 library is used. The Boto3 library provides a programmatic interface to AWS services in python. The boto client will connect to each individual services provided by AWS and allows us to do actions like launching an instance and putting objects in S3 bucket etc.

Information in boto3 is transferred using the JSON format. Responses received from boto3 contain parameters like HTTP status code, name of instances, type of instance etc.
We have implemented interfaces for the following in S3 with boto3:

- List folders
- List Files in a folder
- Downloading a file
- Uploading a file

These interfaces with S3 are written in “s3_utils.py” file inside the “file” folder.

Flask WTForms was used to handle web forms in the program. This flask extension helps validate forms and protect against Cross-site request forgery. A csrf token is generated for the forms and Must be tied to the user's sessions. It is used to send requests to the server, in which the token validates them.

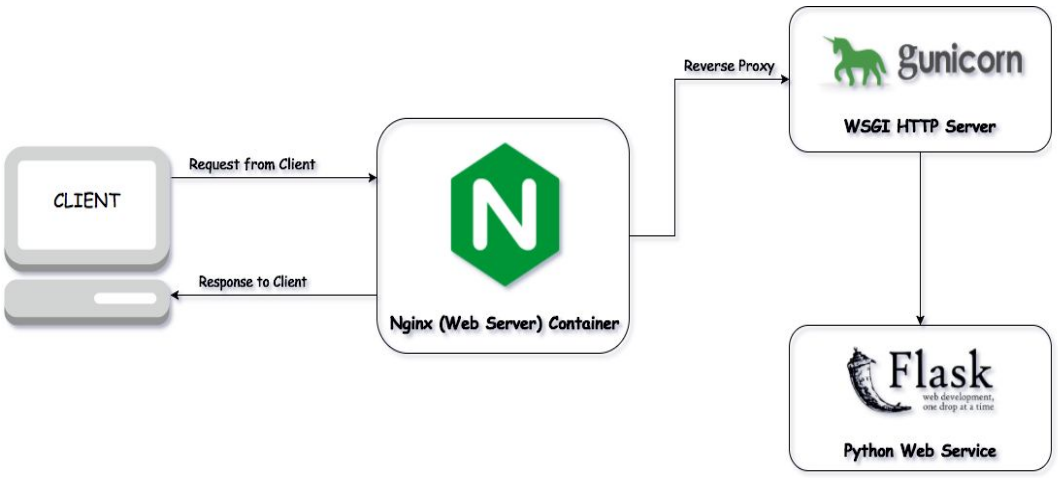
Whenever a new user is created, a new folder in a specified S3 bucket is created. This folder will contain all the files and folders uploaded by the user.
Before uploading the files to the S3 bucket, the first 32 characters of the password hash are used as key to encrypt the files.
Symmetric encryption namely Advanced Encryption Standard is used to encrypt the files. The mode of encryption is Cipher FeedBack(CFB).

File encryption was done using the pycrypto library. An initialization vector was chosen randomly and was the same for encryption and decryption.

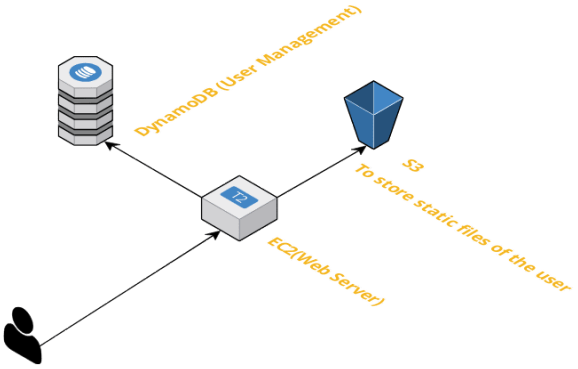
Encrypting files in such a manner ensures that files can only be decrypted by logging in the owner of the file. Encryption (HTTPS) can be used to deliver the files to the client.

Bootstrap and JQuery was used as the frontend technology to improve responsiveness of the web application.

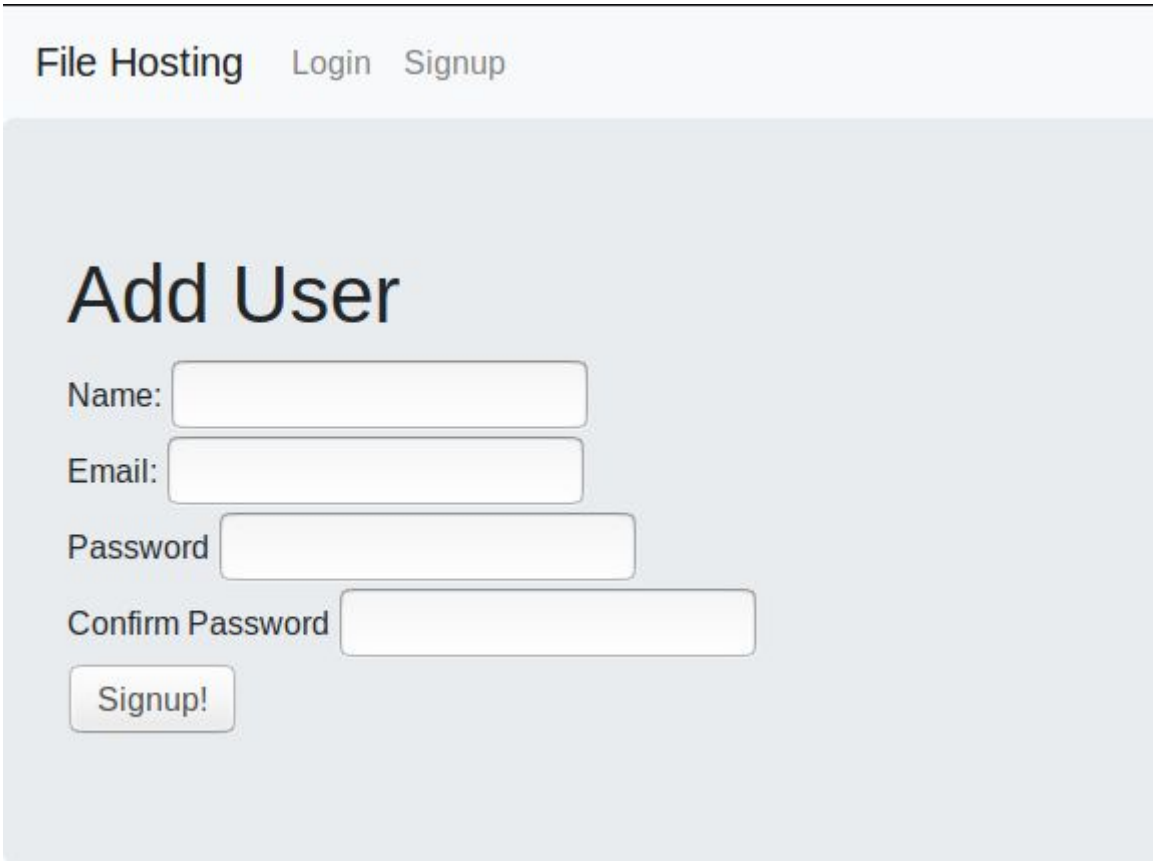
EC2 Software Architecture:

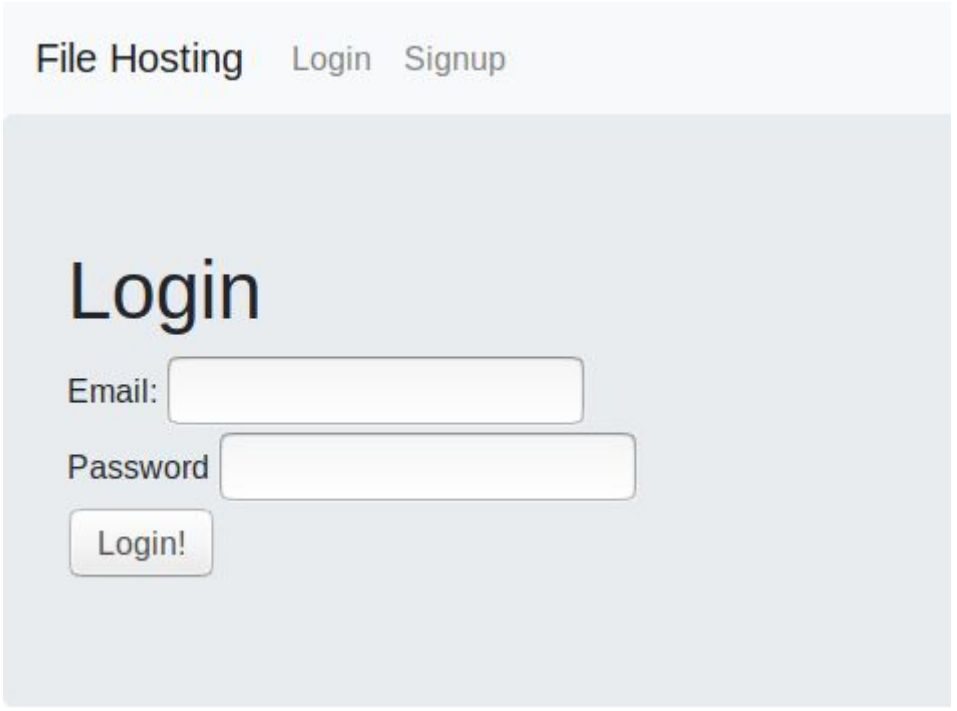


Deployment Architecture:



Screenshots:





Files Uploaded

Files

- 192782-unsplash.jpg
- 521042-unsplash.jpg
- test.png
- test.py

Folders

- Test Folder

Upload

File Browse… No files selected.

Upload!

New Folder

Folder Name Create!

Upload and encrypt code:

```
@files_blueprint.route('/browse/', methods=['POST'])
@login_required
def upload():
    email = session['email']
    _, pwd_hash, _ = load_user(email)
    form = UploadForm()
    if form.validate_on_submit():
        for file in request.files.getlist("file_field"):
            filename = secure_filename(file.filename)
            f = file.read()
            obj = AES.new(pwd_hash[:32], AES.MODE_CFB, 'This is an IV456')
            ciphertext = obj.encrypt(f)
            path = form.target.data
            upload_file(email+ '/'+path +filename, ciphertext)
            file.close()

        flash('Files Uploaded')
        return redirect(url_for('file.browse',p=path))

    return render_template('upload.html', form=form)
```

Download and decrypt code:

```
@files_blueprint.route('/download/', methods=['GET'])
@login_required
def download():
    email = session['email']
    _, pwd_hash, _ = load_user(email)
    prefix = email + '/'
    path = request.args.get('p')
    if path:
        if path.endswith('/'):
            return "Bad Params"
        path = prefix+path
    else:
        return redirect(url_for('file.browse'))

    filename = path.split('/')[-1]
    f = download_file(path)
    obj2 = AES.new(pwd_hash[:32], AES.MODE_CFB, 'This is an IV456')
    decrypted_file = obj2.decrypt(f.read())
    response = make_response(decrypted_file)
    response.headers.set('Content-Disposition', 'attachment', filename=filename)
    return response
```