SNAPVIBE

Ayush Jadhav
StudentId: 23987654
Cloud DevOpsSec, MSc in Cloud Computing
National College of Ireland Dublin, IRELAND
Email: x23987654@student.ncirl.ie. URL: www.ncirl.ie
Deployed Application URL: http://demosnapvibe-env.eba-xn6fmsnz.eu-west-2.elasticbeanstalk.com/users/login/
Video (Project Presentation, Demo, Q&As) URL: https://www.youtube.com/watch?v=lKJBsQ-dmR4

***Abstract*—This report summarizes the detailed information of a dynamic cloud-based django application SnapVibe that allows users to share images, add captions, and interact with posts from other users through likes and comments. The aim of the application is to provide an interface where users can share the photographs of the newly discovered places and suggest such places for a nice visit and also, providing functionalities like uploading, viewing, liking, and commenting on posts of a large community of users. The application is deployed to the Amazon Web Services (AWS) Elastic Beanstalk and CodePipeline along with code build is used for the DevOps CI/CD automation process. Pylint performs static code analysis and security vulnerabilities are handled by SonarCloud. The back-end is developed using Django, a high-level Python web framework, which manages user registrations, authentication, and database interactions. The database operations are managed by SQLite, a lightweight, transactional, and relational embedded database. The front-end is implemented using HTML for visual designs and Bootstrap 5 for creating responsive components hence providing a user-friendly interface. Increasing user engagement, building a community around photography, providing users to discover new perspectives, and connecting with like-minded individuals, through photo sharing, is what SnapVibe seeks. By employing the cloud services, SnapVibe also aims to provide a scalable, efficient, and highly available application. The report includes the architectural description, implementation details, challenges faced, and results of SnapVibe.**

***Index Terms*—Elastic Beanstalk, SQLite, Code Pipeline, CodeBuild, S3, SonarCloud, DevOps, CI/CD, Django web app, photo sharing, SnapVibe**

I. Architectural Design

The rise of social media platforms has been fueled by the ever-growing desire of sharing experiences through the connectivity with others. Among these platforms, photo-sharing applications have emerged as a popular interface, allowing users to express themselves visually and bring communities together who have common interests. This report details the architecture design and development of a cloud-based photo-sharing application, utilizing modern web development technologies to create a platform for photography enthusiastic users to upload, share, and engage with photos.

Figure 1 gives a glimpse of the main functionalities focused by the application:

- **User Registration and Login:** Users can create accounts by providing basic information and log-in to their existing accounts securely.
- **Photo Uploads and Profile Management:** Users can upload photos from their devices and edit basic details like captions or locations.
- **Browsing User Feeds:** Users can come across and search for photos shared by other users.
- **Social Interactions:** Users can utilize features like liking and commenting.

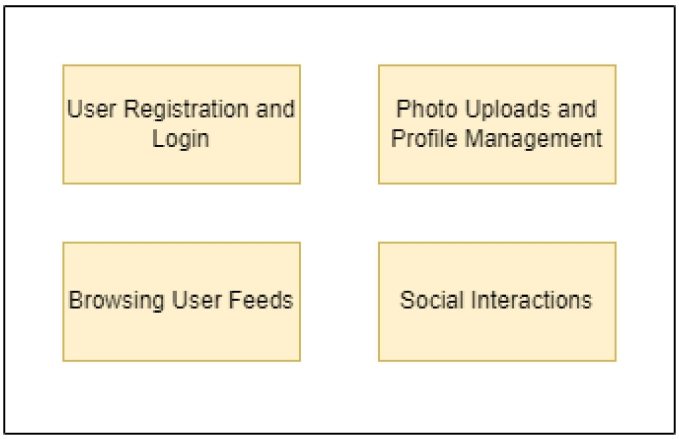

Fig. 1. Functionalities provided by SnapVibe application

The communication between the users and SnapVibe application goes as follows:

1. Users interacts with the application hosted on the cloud.
2. The request is redirected to the Django back-end.
3. Django interacts with the database for data storage and retrievals.
4. Then the request is processed by Django and the response is generated.
5. The response is sent back to the user's display.

Figure 2 presents the **architecture** of the application. The architecture involves various independent components.

- **Users:** These are end users or clients who are going to interact with the django application via Internet. Users can access the application through various devices (phone, tablet, desktop) web browsers.
- **Internet:** It provides public network through which users all over the globe can access the application by sending

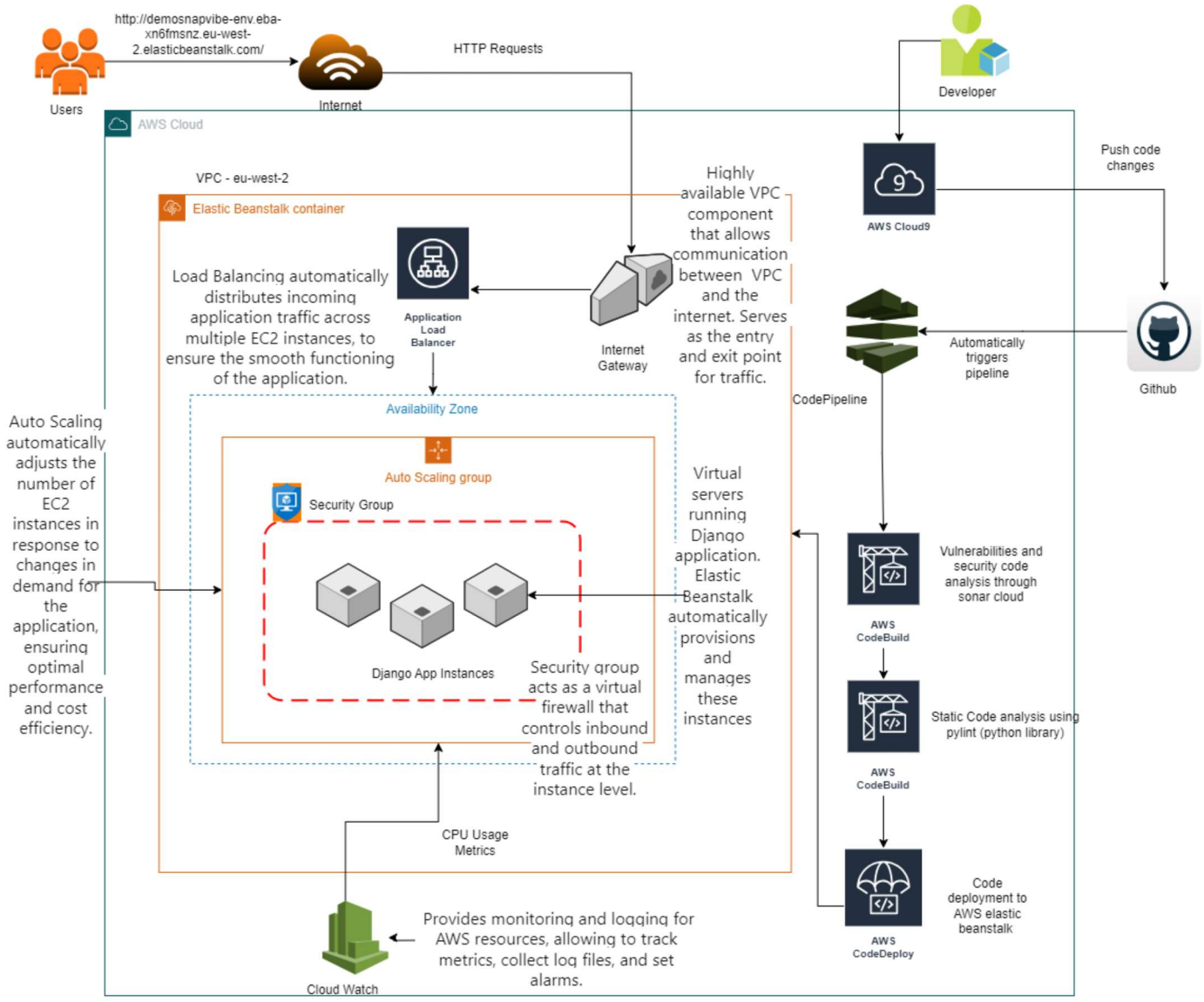

Fig. 2. Architecture Diagram

the http request and get response from the server and renders it into users browser.

- **Internet Gateway:** It is a highly available VPC component that helps in communication between VPC and the internet. It provides the facility for inbound and outbound traffic to django application hosted on AWS Elastic Beanstalk.
- **Application Load Balancer:** It helps to distribute the incoming traffic across various ec2 instances in various availability zones. From this, high availability of application and fault tolerance is ensured.
- **Availability Zones:** AWS infrastructure is divided into various regions and each region is further divided into multiple availabilty zones. These zones ensures high availability of applications and also stable and fault-tolerant application.
- **AutoScaling group:** AutoScaling group helps in auto scaling of the ec2 instances based on load on the server. It again helps in scaling up or down based on traffic patterns.
- **Security Group:** A security group acts as a virtual firewall for all ec2 instances and controls the inbound and outbound traffic to the application. It creates the rules allowing HTTP/HTTPS requests to django application.
- **Django Ec2 Instances:** These are virtual servers running the django application.
- **CloudWatch:** Cloud watch is a monitoring service which provides various health checks.
- **Cloud9:** Cloud9 is an integrated development envi-

ronment or IDE provided by AWS. It removes the need to setup a development environment as it helps to code, build, run, debug and deploy the application code. Django framework is pre-configured and all the libraries and tools. Cloud9 also offers to setup a special environment type known as "EC2 Environment", which creates a new EC2 instance to connect to our development environment.

- **Github:** It is web based platform which helps in source versioning control. It helps various developers to work on same project.
- **Elastic Compute Cloud (EC2):** Elastic Compute Cloud is one of the core services provided by AWS for cloud computing, focused on provisioning virtual machines based on the requirements of the users. It reduces the cost as we have to pay only for the instances that we use. It also provides a variety of EC2 instances to choose from. We can scale up or down the compute resources easily based on the application's needs.
- **Code Pipeline:** Code Pipeline is service of AWS which helps in implementing the CI/CD process. It helps in easy deployment of the application's releases. It allows to setup the different stages of the deployment pipeline. For source stage, GitHub is used as the repository for storing the code and for versioning. AWS CodeBuild tool is used for the build stage. CodeDeploy is used for the deploy stage, target being Elastic Beanstalk. Code Pipeline ensures in automating the build and deploy as soon as the changes are committed to GitHub repository, faster delivery of the new features or bug fixes, and minimizes the downtime of the application by maximizing the availability.
- **Elastic Beanstalk:** Elastic Beanstalk is deployment service provided by AWS that handles the deployment and management of the application[1]. It supports various programming languages like Java, Python, Ruby, .NET, PHP, etc. We just need to provide the EC2 instance (or other AWS resources) and Elastic Beanstalk automatically picks up the supported platform version. It also provides health monitoring, application metrics, load balancing and scaling.

The flow involves various independent development of the presentation (front-end), application (back-end), data access (DAOs) and database (sqlite) layers.

- **Presentational Layer:** This layer focuses on the accessibility of the application to the users. Users can access the application through various devices (phone, tablet, desktop) web browsers. The user interface (UI) is developed using the technologies like HTML and Bootstrap. They have helped in structuring responsive web pages and UI that adapts to different devices.
- **Application Layer:** Django, a high-level Python web framework, is used for the backend development of the application; it handles all core functionalities. Django is used along with SQLite, a lightweight relational database, providing a simple and easy to use interface[2]
  - **User Authentication:** Whenever a user tries to login, the application makes sure that the user is authorized. Django provides a built-in functionality to check the basic authentication of the user. If the user does not exist it allows them to register to the application. Django provides some built-in mechanisms for registration, login, and session handling.
  - **User Management:** Django provides user models with relevant fields like username, email, password, and other user related information. It also manages creation of new users, user data retrieval, and updates, providing a centralized system.
    Figure 3 represents the Django form validations in which it checks whether the username already exists in the database. Similar validations have been implemented for other fields as well.

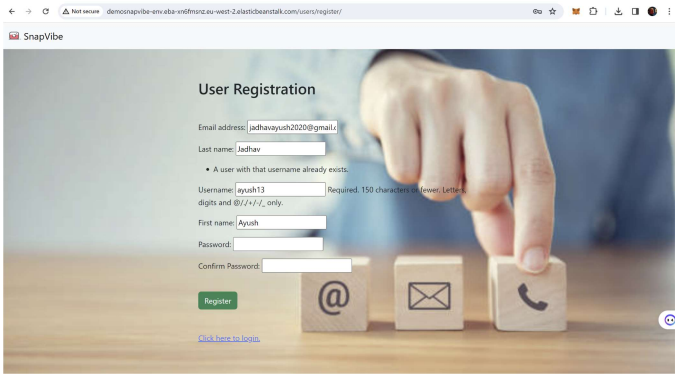

Fig. 3. User Validations

  - **Data Management:** Django allows functionalities to handle the photos uploaded by the users. It validates the uploaded data to ensure that they meet the specified format and size.
  - **Database Interactions:** Django acts as a medium between the database and the user interface. Django provides Object Relational Mapping (ORM) for SQLite, which helps in simplifying the database operations.
  - **API Development:** Well defined Application Programming Interfaces (APIs) are created using Django. These APIs act as the communicating channels between the back-end (Django) and front-end (user interface). The APIs are implemented based on the functionalities provided by the web application, including user registration, authentication, uploading photos, data retrievals, and other user interactions (such as liking, updating, commenting).
  - **Routing:** The different functionalities based on the user requests are handled by the Django URL routing

system. It maps the requests to the appropriate views within the application.

Django provides security features, accelerate developments because of its pre-built features, and can handle the growing user base.

- **Data Access Layer:** Django uses components like the Data Access Objects (DAOs) and Object Relational Mappers (ORMs) to interact with the database. Data attributes are represented by the Django models, and define various field types like TextField, CharField, ImageField, DateField, etc. Django also allows to establish the model relationships between objects. For example, each Profile can have a OneToOne relationship with a User, each Post can have a ManyToMany relation with User for the number of likes attribute and so on. There is no need to write the SQL queries for all functionalities as the ORM helps in translating the model operations, making the data access and manipulation simpler. Hence, Django makes it easier to reduce code redundancy, validation of data, simplifying the operations and provides flexibility.
- **Database Layer:** This layer acts as the backbone for the application as it handles all the data related operations. Each Django model represents to a corresponding table in the SQLite.

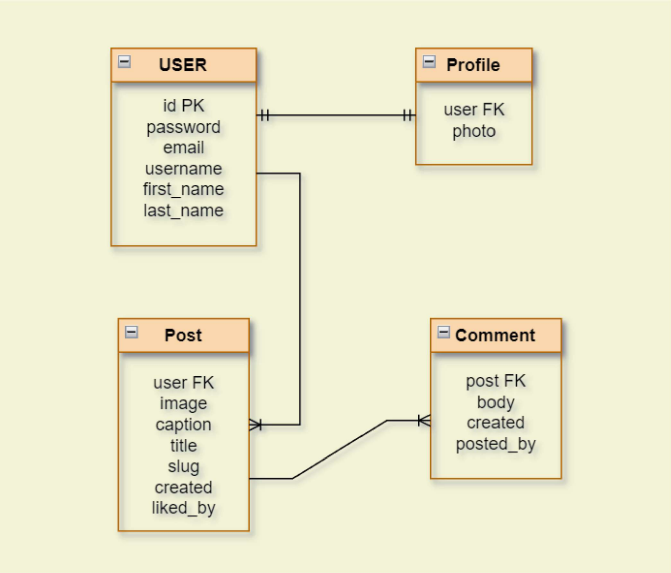

Fig. 4. Database models used in SnapVibe application

Figure 4 shows the various models used in the application

- **User Model:** This model includes user related information, such as username, email, password, firstname, lastname and an auto incremented id.
- **Profile Model:** This model has a OneToOne mapping/relationship with the User model, indicating that one user can have one account. The model also has a photo attribute.
- **Post Model:** This model consists of the details related to a post, including image, caption, title, created

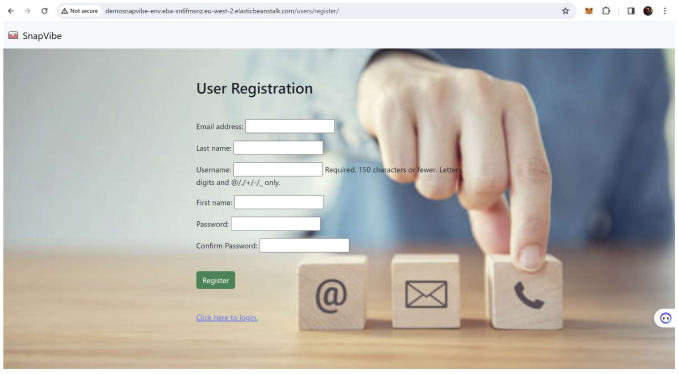

Fig. 5. Registration Page

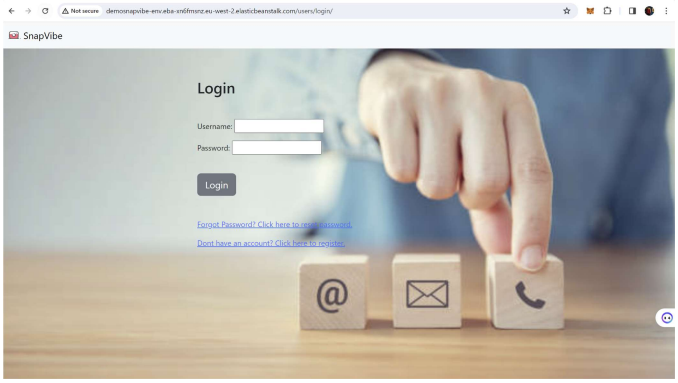

Fig. 6. Login Page

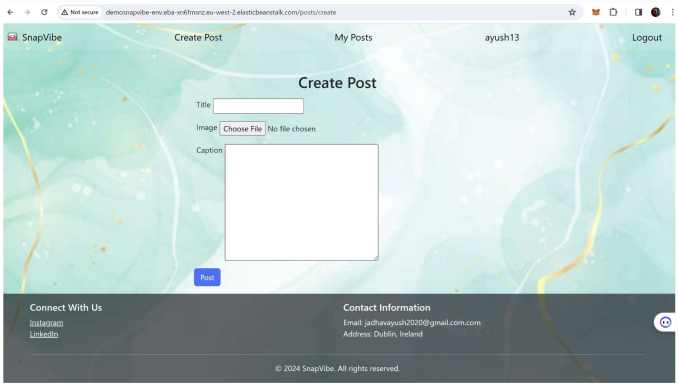

Fig. 7. Creating the Post

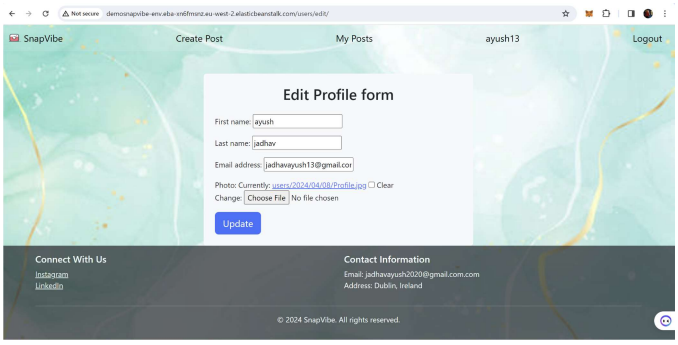

Fig. 8. Updating form

date, liked_by. This model also has a ManyToOne relationship with the User model stating that a single User can have a lot of posts. The liked_by parameter also has a ManyToMany relationship as any user can like the photo posted.

- **Comment Model:** This model consists of data related to the comments of a post, which includes details of the post, comment body, creation date and posted by the user. It has a ManyToOne mapping/relationship with the post model indicating that a post can have multiple comments.
- **User Model:** This model includes user related information, such as username, email, password, firstname, lastname and an auto incremented id.
- **Profile Model:** This model has a OneToOne mapping/relationship with the User model, indicating that one user can have one account. The model also has a photo attribute.

Fig 5 to 8 shows various snapshots of snapvibe application incorporating functionalities that has been implemented. Create, read, update, delete (CRUD) functionalities has been successfully implemented in snapvibe application.

II. Continuous Integration, Continuous Delivery and Deployment

Continuous Integration and Continuous Delivery/Deployment or CI/CD plays a crucial part when hosting software applications to the cloud. CI/CD automates the software development process and creates a delivery pipeline for software development, from writing code to deployment. The CI/CD pipeline provides the following benefits:

- It reduces the possibility of manual errors happening at the time of deployments.
- It speeds up the build and deployment process as we do not have to manually configure it.
- It finds out the errors/bugs in the system at an early stage ensuring smooth production releases.

Implementation of CI/CD in SnapVibe

The tools and services used to implement the CI/CD pipeline in the application are:

- **Version Control:** Git is used for controlling the versions of code releases, maintain the codebase, track the commits and also helps to revert to the previous versions if required.
- **AWS CodePipeline:** CodePipeline is used for build, deploy and deliver the service to the cloud architecture[3]. It gets triggered by the commits in the Git.

Pipeline Stages

There are several action types available in AWS CodePipeline. Figure 9 shows the action types that have been used for our application SnapVibe:

- **Source:** This is the first stage of the pipeline. Git repository is used to fetch the code changes. The CodePipeline gets triggered automatically after a commit is made to the main branch.

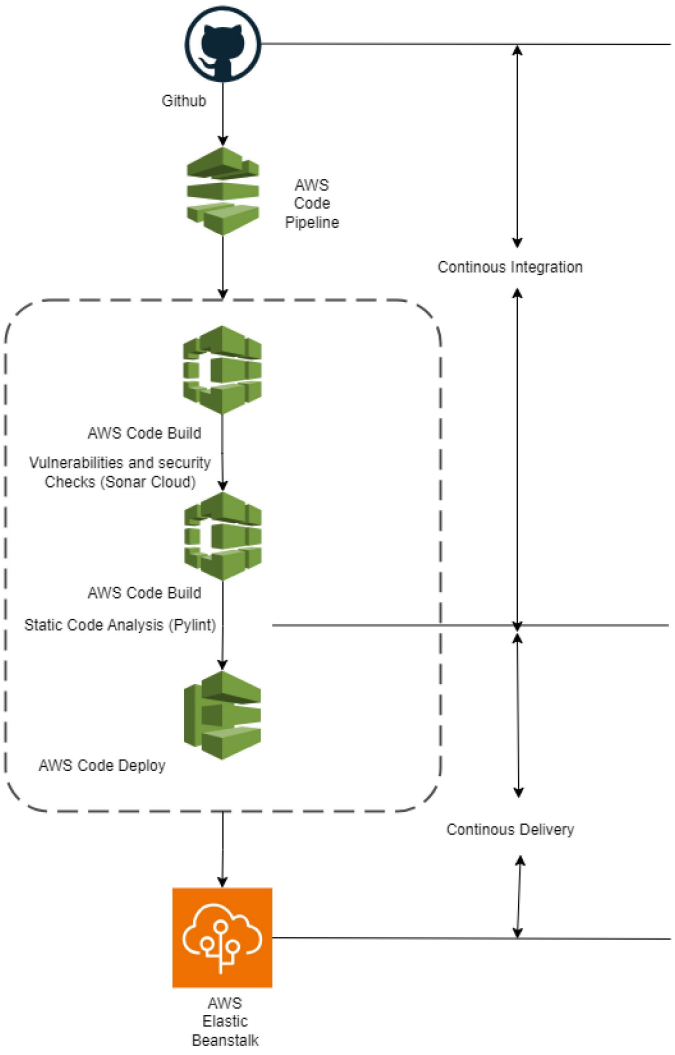

Fig. 9. Pipeline Stages

- **Code Review:** This is the second stage, vulnerabilities check is performed here with the help of sonar cloud. The detailed description is covered in the next section.
- **Code Build:** This is the third step, static code analysis is performed by pylint. The detailed description is covered in the next section.
- **Deploy:** This is the final step of the pipeline where the django application is deployed to elastic beanstalk.

Implementation and Results:

Figure 10 represents successfully implemented code pipeline and the order in which various actions are performed for running the application on the cloud. It helps in automating any new release of a feature or code changes to the production quickly and reliably. The code pipeline uses a predefined model, it looks for any changes in GitHub using webhook, if it encounters a change or new commit, it notifies code pipeline by triggering code

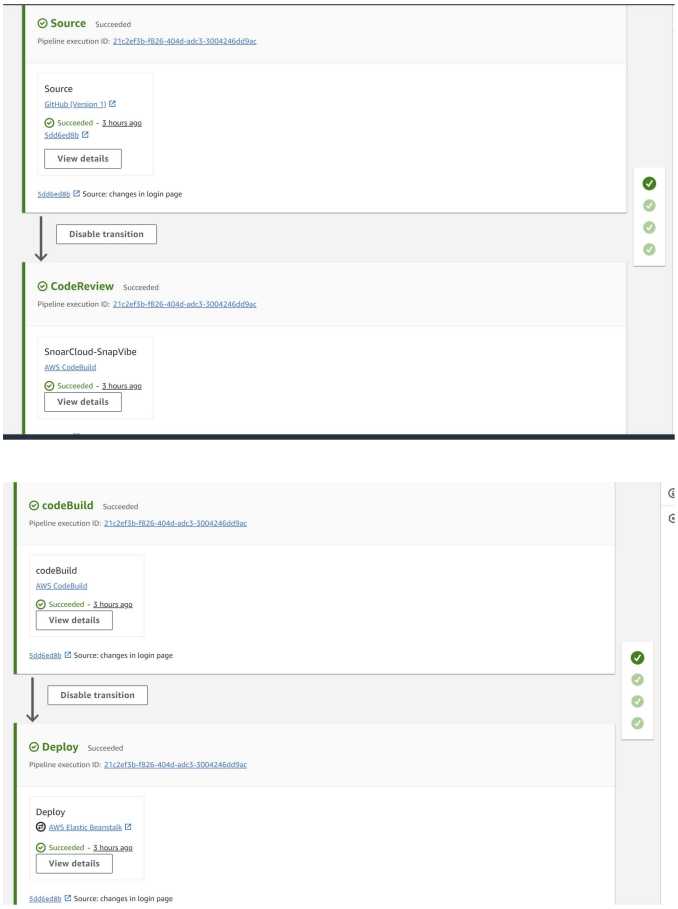

Fig. 10. AWS CodePipeline

pipeline. Code pipeline then starts building and deploying the changes. Firstly, CodeBuild is done where the code is fetched from the version control system which in this case is Git and then it is compiled, unit tests are executed and is packaged for deployment. In this step, the vulnerabilities and security checks are performed by SonarCloud and then the static code analysis is done using Pylint. After that the packaged application is deployed with the help of CodeDeploy. It is responsible for pushing the application to AWS Elastic beanstalk.

III. Static Code Analysis

The building of reliable, bug-free, vulnerable, and stable code involves the use of many tools. Security is an important aspect to consider during the development of any application. The quality of any software is dependent on the quality and structure of the source code. Static code analysis works on improving code style by embedding or suggesting a better code style and design. On the other hand, security vulnerability analysis focuses on identifying security concerns. Both combined work together to deliver a reliable, high-quality, secure application. SnapeVibe is tested for static code and vulnerability using PyLint as a static analysis tool and Sonar Cloud as a vulnerability analysis tool. The detailed explanation of each term and implementation part is discussed below.

- Static Code Analysis is usually done with automated tools or in-built libraries. It examines the complete code for code quality, design of code, inclusion of doc string, comments or any grammatical violation of standard coding conventions. Static analysis helps to maintain improved readability, quality code, and improved readability of code. PyLint is a static code analysis tool that is mainly used for python[4]. PyLint is customizable and can easily integrated with IDE's and CI platforms. Pylint supports various extension and plugins. For this, configuration file is been created. which is integrated with code build to download and install dependencies of PyLint [5]. The detail analysis of finding and improvement is mentioned in below figure 11.
  Following steps are followed to perform static code analysis using pylint:
  - Installation of pylint library in the project. The command for it as follows: "pip install pylint"
  - Navigate to the the project directory.
  - Creation of pylint configuration file in which all configurations can be specified. The command for it as follows "pylint –generate-rcfile pylintrc"
  - To finally perform static code analysis following command needs to be executed which will analyze the project. "pylint SnapVibe.py"
  - If any particular file needs to be analysed then a specific file name can be given in the command. "pylint SnapVibe.py". The above command will perform static code analysis on the manage.py file.
  - Pylint will provide an output file of all the findings including any issues in the code along with line number the error code and a brief description.
  - It also gives code rating out of 10.
  - If in buildspec.yml configuration file for pylint, multiple directories are given in commands then it is necessary to get 10 code rating to go ahead to next stage.
  - After this necessary action needs to be taken to resolve those issues. Issues like improper arrangements of import statements, missing docstring, missing object methods, too long line etc.
  - By following the above steps standard coding practices can be achieved.
- Vulnerability analysis is the process of identifying security vulnerabilities in a web application. There are many tools which carry out vulnerability tests. Applications, including source code and artifacts are scanned for vulnerability to check for security flaws. It helps to identify weak libraries, framework and application components by using security rules defined. Vulnerability tools can easily identify weak libraries, unsafe setups, code smells, or bug fixes in code and suggest the required changes accordingly. SnapVibe web app is tested for vulnerability

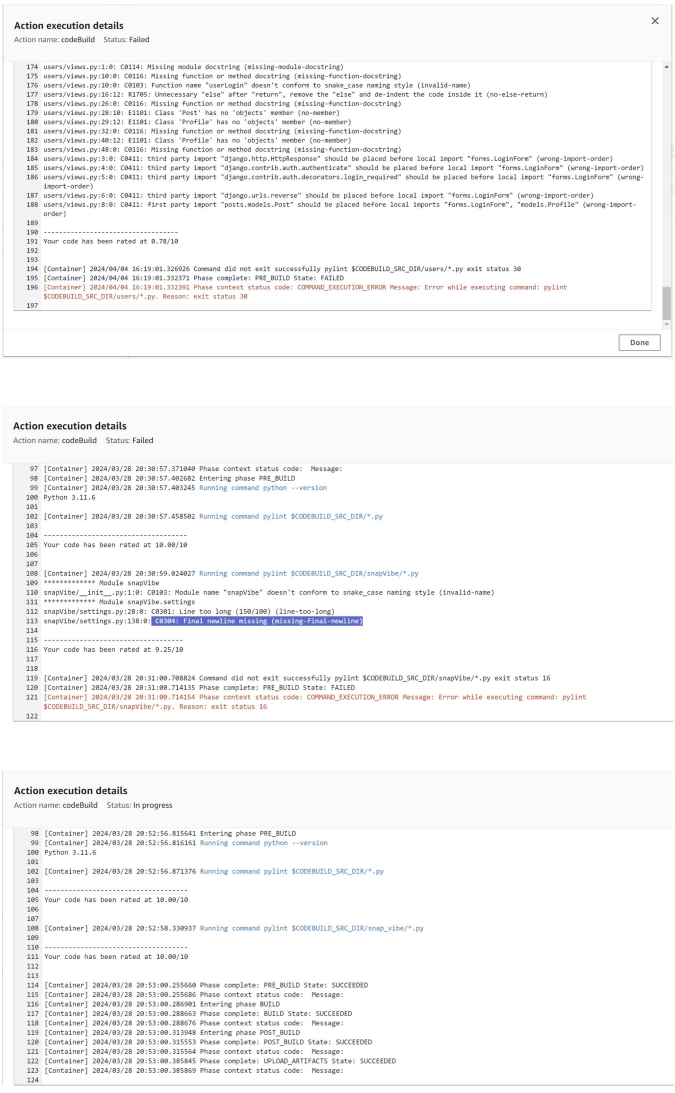

Fig. 11. Static Analysis using PyLint

using Sonar Cloud. SonarCloud is a cloud-based tool to continuously inspect the code quality and security for any code present in version control platforms[5].
The following steps are followed to integrate sonar cloud with AWS code build:

- Signing up to sonar cloud with github account.
- Need to provide repository access to sonar cloud to perform analysis. For this repository needs to be public to use free access of the sonar cloud.
- After this need to configure AWS code build. From AWS console management, created code build project and configured that based on buildspec.yml file which needs to be stored in the project root directory.
- This buildspec.yml file contains all the pre-build and build phase commands that need to be performed.
- Project details need to pass in the sonar-scanner command which includes project key and organization key.
- After this pipeline needs to be triggered.
- Sonar cloud will do quality checks which are configured in quality gate configuration.
- After a successful run, a detailed analysis of the project can be viewed on sonar cloud console as well as in code build logs.
- Quality of code, code smell, bugs, vulnerabilities, security issues, and many more issues can be identified and appropriate steps to resolve them can be taken.
- On sonar cloud users can add and modify the quality gates as per their project requirements. For this project quality gates include duplicate lines percentage, maintainability ratings, bugs, code smell, comments, security, critical issues, percentage of unit test cases covered are taken into consideration.

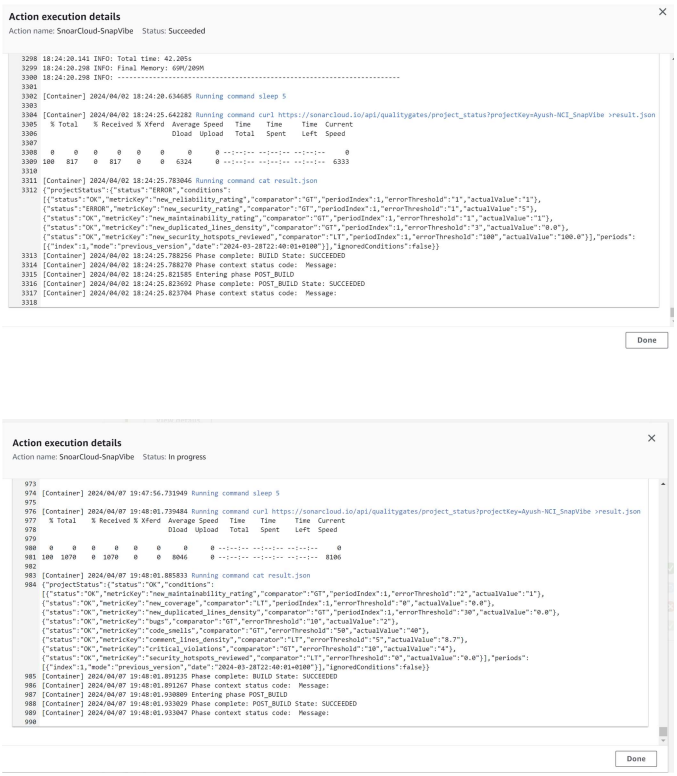

Fig. 12. Vulnerability Analysis using Sonar Cloud

Sonar cloud does this analysis using sonar scanner, it is a inspection tool that continuously inspects the GitHub for new changes. Sonar scanner works on rules defined in quality gate; it suggests changes for all the failures in rules. One can easily add or remove quality gate metrics from sonar cloud. For every project in sonar cloud there is an associated quality profile that make use of the quality gate defined. It is integrated with the sonar cloud with code build AWS service using buildspec2.yml configuration file. Complete build process and installation of dependencies is mentioned in this file. The detail analysis and security improvement suggested by sonar

cloud is mentioned below. According to the suggestion of sonar cloud respective changes has been made in code to improve code quality. Below are the attached screenshots of analysis made by sonar scanner and improvement made.

IV. Conclusions

Successfully implemeneted CRUD operations in the social media "SnapVibe" web application. All functional and non-functional requirements have been implemented. Also deployed the application through the implementation of CICD pipeline to AWS elastic beanstalk. Static analysis and vulnerability issues are identified using pylint and sonar cloud along with AWS code build, and necessary actions to fix them have been performed. While in the development and deployment phases, learned various things including various coding standards including doc-String, coding structure, security analysis, code smells, bug fixes etc.

If this project again needs to be implemented then I would have taken care of all standard coding practices that need to be followed because after using pylint got to know Python uses snake case and making changes afterward sometimes becomes a hectic task. Also, I would like to incorporate additional features to it like integrating with Google Maps API where users can get directions to places. Also, a functionality can be added where groups can be formed in this app and people can make plans for outings together.

V. References

[1] https://docs.aws.amazon.com/elastic-beanstalk/.

[2] https://realpython.com/get-started-with-django-1/.

[3] https://aws.amazon.com/codepipeline/.

[4] https://pylint.readthedocs.io/en/stable/.

[5] https://medium.com/codex.