Review Radar

Leveraging NLP for Targeted Product Enhancement

CS 816- Software Production Engineering

International Institute of Information Technology, Bangalore

under the guidance of

Prof. B Thangaraju

Submitted by

Anjali J Kumar (MT2023126)

Sambhu S S (MT2023072)

GitHub

CONTENTS

Table of Contents

ABSTRACT

In the realm of consumer electronics, understanding customer sentiments and identifying key areas for improvement is paramount for staying competitive. This project presents a comprehensive solution leveraging DevOps methodologies to enhance the efficiency and reliability of a sentiment analysis and topic modelling system for phone companies.

The project begins with the development phase, wherein a dataset comprising reviews of various phone companies is subjected to topic modelling using Non-Negative Matrix Factorization (NMF). This process identifies latent topics within the reviews, offering insights into the diverse aspects of customer feedback.

Building upon the topic modelling results, sentiment analysis is performed on each review within the identified topics. By gauging the sentiment of individual reviews, the system identifies the areas of concern and excellence for each phone company, facilitating targeted improvements and highlighting strengths.

The core of the project lies in its DevOps implementation, aimed at automating and streamlining the deployment and maintenance processes. Continuous Integration (CI) and Continuous Deployment (CD) pipelines ensure seamless updates and enhancements to the sentiment analysis and topic modelling algorithms. Additionally, robust monitoring and logging mechanisms are integrated to provide real-time insights into system performance and user interactions.

A user-friendly dashboard is provided to phone companies, presenting actionable insights derived from the sentiment analysis and topic modelling results. Moreover, a feedback mechanism allows for the submission of new reviews, which are promptly analysed to provide instant feedback on sentiment and topic relevance.

Through this project, the fusion of advanced data analytics techniques with DevOps practices demonstrates a holistic approach towards enhancing customer experience and driving business success in the dynamic consumer electronics landscape.

PROJECT DETAILS

Introduction

This project aims to analyze customer reviews of Apple and Samsung phones to provide insights into customer sentiments and highlight key areas for improvement. By using topic modelling and sentiment analysis, the system helps companies understand customer feedback comprehensively and take data-driven actions.

Data Collection and Preparation

The initial dataset comprised approximately 400,000 reviews. From this, reviews related to Apple and Samsung were extracted. The preprocessing steps involved:

Cleaning the Text Data: Removing stop words, punctuation, and performing tokenization.

Filtering: Only retaining reviews related to Apple and Samsung for focused analysis.

Topic Modelling

Topic modelling was performed using Non-Negative Matrix Factorization (NMF) to identify the main topics within the reviews. Ten topics were identified, each characterized by the top 15 words associated with them. These topics were manually named for clarity:

- Appreciation Towards Service and Delivery
- Buying Experience
- Condition and Packaging of Items
- Expectations and Product Quality
- Issues with Device Functionality
- Ordering and Customer Service
- Overall Satisfaction with Purchase
- Product Quality and Experience
- Recommendations and Satisfaction
- Satisfaction with Product Condition

Implementation:

NMF Configuration: NMF was configured to generate 10 topics.

Evaluation: Topics were evaluated and named based on the top 15 words identified by the model.

Sentiment Analysis

Sentiment analysis was conducted using the VADER sentiment analysis tool. Each review was classified into one of three categories: positive, neutral, or negative. This analysis provided insights into the overall sentiment distribution across different topics.

Implementation:

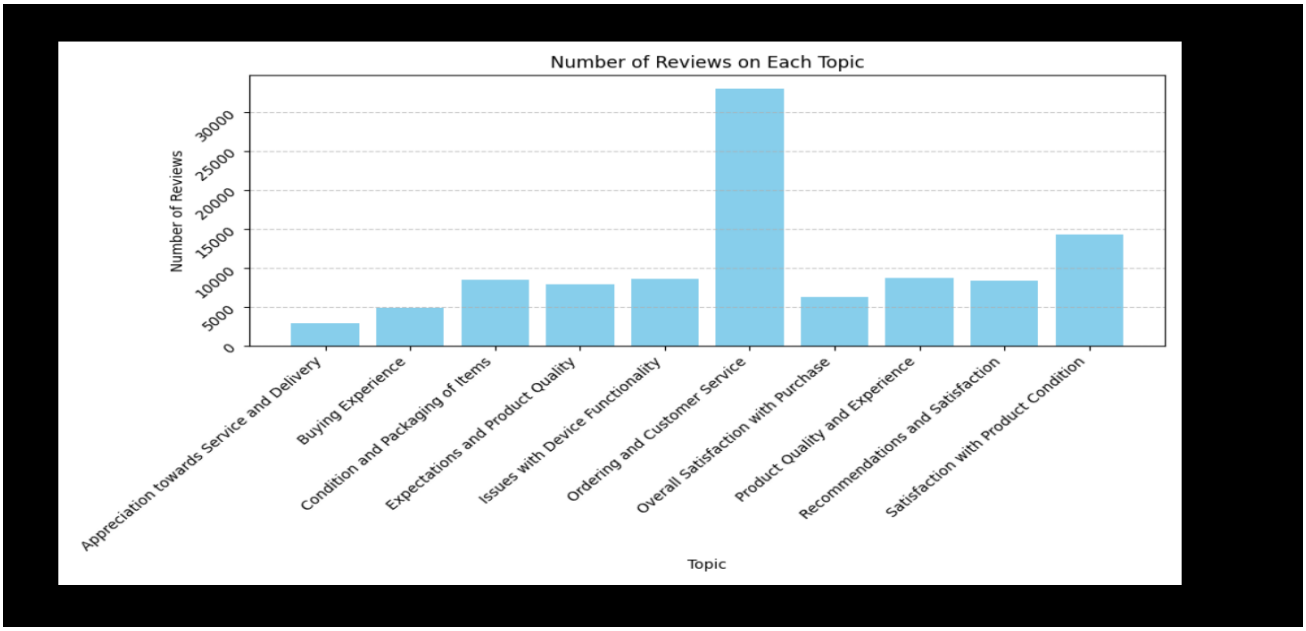
Sentiment Classification: VADER was used to analyze and classify sentiments for each review.

Result Aggregation: Sentiments were aggregated for each topic to understand the distribution.

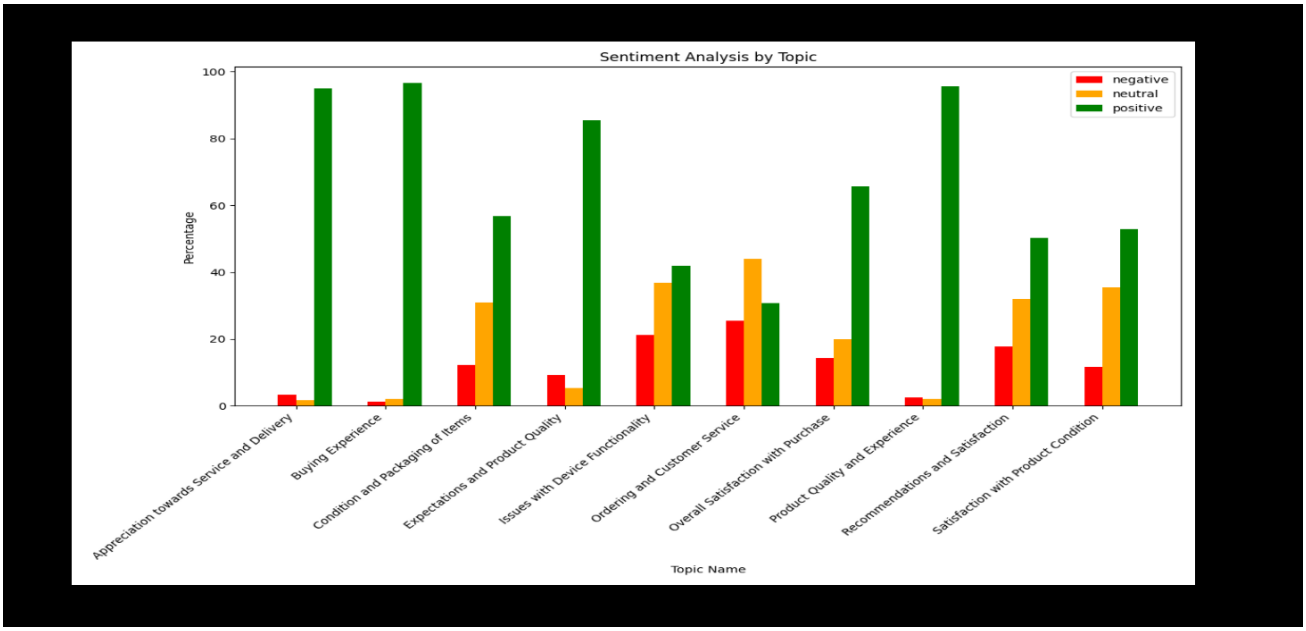
Dashboard

The main features of the dashboard include:

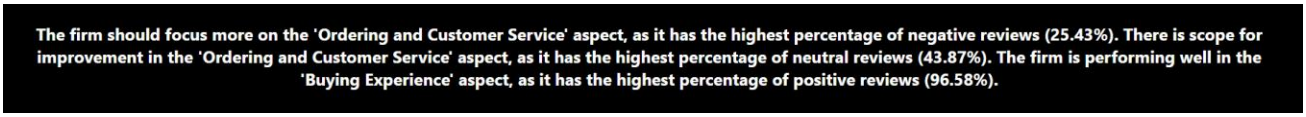
Topic Distribution Bar Plot: This bar plot shows the number of reviews associated with each topic. It helps companies understand which aspects customers frequently comment on.



Sentiment Analysis by Topic: This bar plot displays the percentage of positive, neutral, and negative reviews for each topic. It helps companies identify areas needing attention.

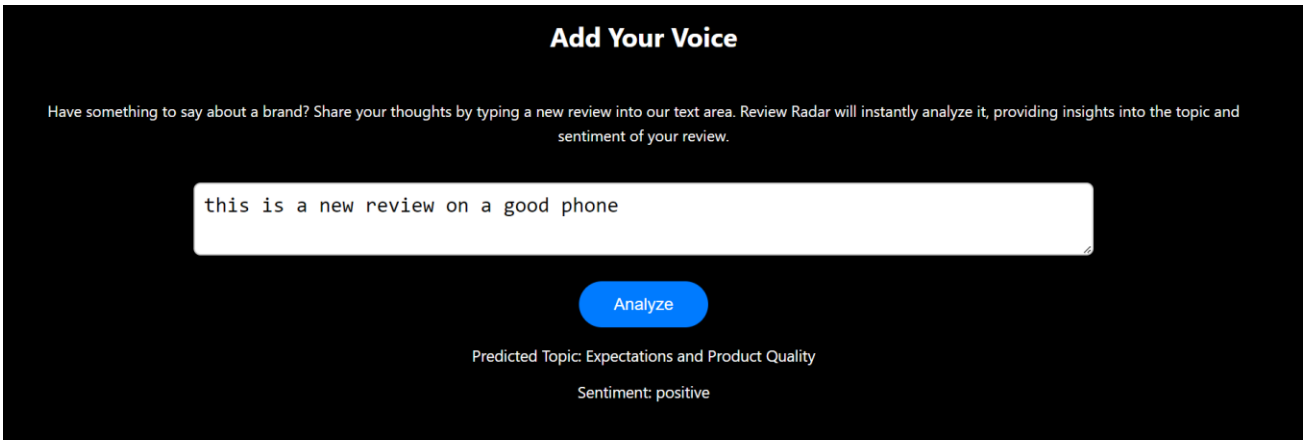


The dashboard provides a clear visual representation of the topics and their corresponding sentiments, making it easy for companies to interpret the data and derive actionable insights.



User Interaction

The dashboard includes a text box for users to submit new reviews. The system analyzes these new reviews in real-time, predicting the relevant topic and sentiment. The results are displayed immediately to the user, providing instant feedback.



Implementation:

Real-Time Analysis: New reviews are analysed in real-time for topic prediction and sentiment analysis.

Feedback Display: The predicted topic and sentiment are displayed instantly to the user.

Data Storage: Newly submitted reviews are stored in a MySQL database for future analysis, although they are not currently used in the live system.

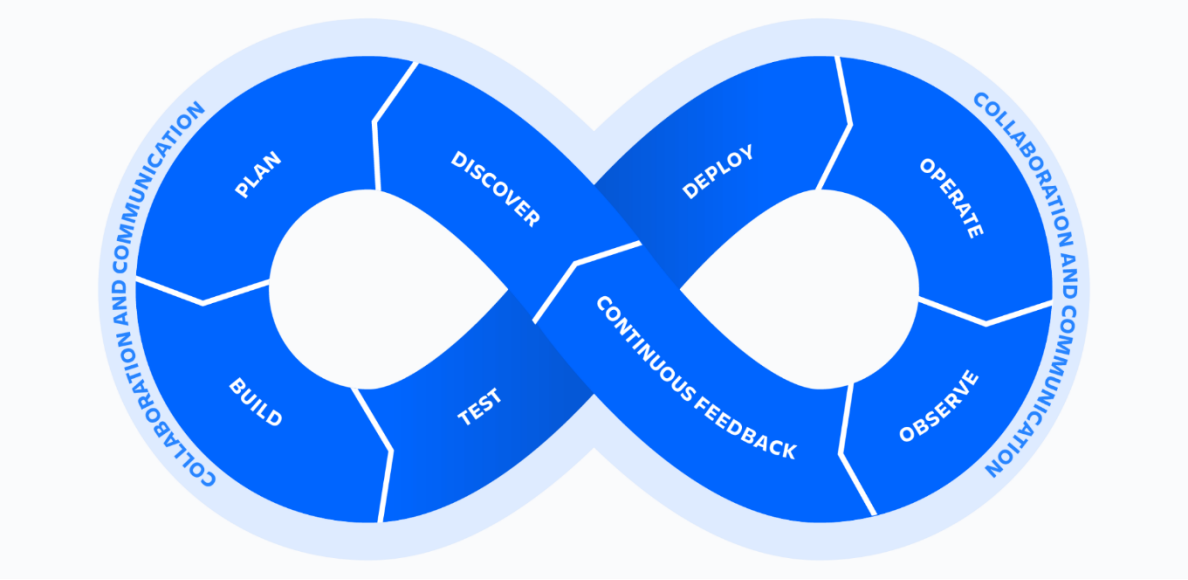
Conclusion

This project effectively utilizes topic modelling and sentiment analysis to provide actionable insights from customer reviews of Apple and Samsung phones. By identifying key topics and their associated sentiments, companies can focus on areas needing improvement and recognize their strengths. The interactive dashboard enhances user experience by allowing real-time analysis of new reviews, ensuring continuous feedback and improvement.

DevOps

DevOps is a set of practices aimed at breaking down silos between software development (Dev) and IT operations (Ops) teams, fostering collaboration, and automating processes throughout the software development lifecycle.

The primary goal of DevOps is to increase the speed, efficiency, and quality of software delivery.



1. **Assessment and Planning**
    - Evaluate current development and operations processes, identifying pain points, bottlenecks, and areas for improvement.
2. **Culture and Organization**
    - Encourage communication and transparency between teams to break down silos and build trust.
3. **Automation**
    - Identify manual and repetitive tasks throughout the software development lifecycle, such as building, testing, deployment, and infrastructure provisioning.
4. **Continuous Integration (CI)**
    - Establish a CI pipeline that automatically builds and tests code changes whenever they are committed to version control.
5. **Continuous Delivery or Deployment (CD)**
    - Extend CI to include continuous delivery practices, enabling automated deployment of code changes to production or staging environments.
6. **Monitoring and Feedback**
    - Implement monitoring and logging solutions to track application performance, infrastructure metrics, and user behaviour in real-time.

Why DevOps?

1. **Faster Time to Market**: By automating processes and breaking down barriers between teams, DevOps enables faster delivery of features and updates to customers, allowing businesses to respond to market demands more quickly.
2. **Improved Quality**: Continuous integration, automated testing, and continuous delivery practices help catch bugs early in the development process, leading to higher-quality software with fewer defects.
3. **Increased Collaboration**: DevOps encourages collaboration between development, operations, and other stakeholders, fostering a culture of shared responsibility and accountability.
4. **Better Scalability and Reliability**: Automation and infrastructure-as-code practices enable teams to easily scale infrastructure and deploy applications across different environments, leading to improved scalability and reliability.
5. **Reduced Risk**: By automating processes and deploying smaller, incremental changes, DevOps reduces the risk of introducing errors or failures into production environments.

Tools used

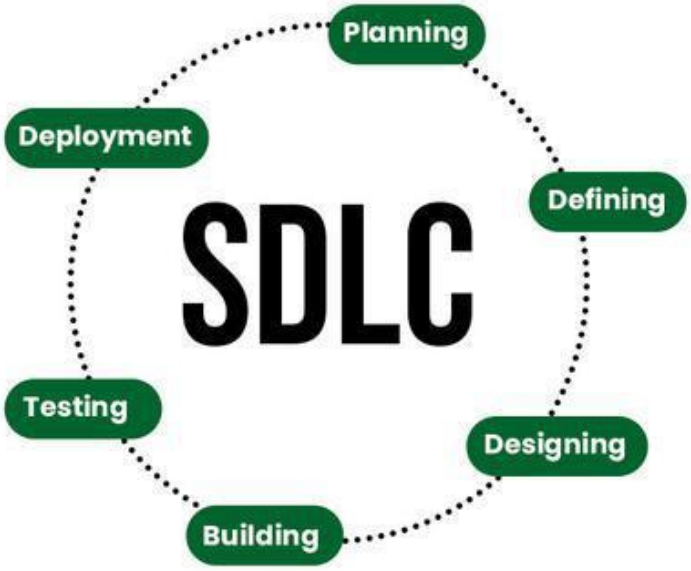
1. **PyCharm/VSCode** - Project IDEs for backend (PyCharm) and frontend (VSCode) development
2. **No formal build tool (Flask-based project)** - Build Tool
3. **unittest** - Testing Tool
4. **Git** - Source Control Management Tool
5. **Jenkins** - Continuous Integration Tool
6. **Docker** - Containerization Tool
7. **Kubernetes** - Container Orchestration Tool
8. **Ansible** - Configuration Management Tool
9. **ELK Stack** - Continuous Monitoring

Software Development Life Cycle (SDLC)

Software development life cycle (SDLC) is a structured process that is used to design, develop, and test good-quality software. SDLC, or software development life cycle, is a methodology that defines the entire procedure of software development step-by-step.

The goal of the SDLC life cycle model is to deliver high-quality, maintainable software that meets the user's requirements.

SDLC in software engineering models outlines the plan for each stage so that each stage of the software development model can perform its task efficiently to deliver the software at a low cost within a given time frame that meets users' requirements.



Steps Followed in the Project

Set Up Development Environment

Install dependencies using pip and create a virtual environment with virtualenv or venv.

Use PyCharm for backend development and VSCode for frontend development.

Test

Write and run unit tests using **unittest** to ensure code quality and functionality.

Source Code Management

Use Git for version control and managing source code repositories.

Continuous Integration

Set up Jenkins to automate testing and integration processes.

Containerization

Use Docker to create container images for the Flask application, ensuring consistency across different environments.

Configuration Management

Utilize Ansible to manage and automate configuration across various environments.

Deployment

Deploy the containerized application using Kubernetes for orchestration and managing scalability.

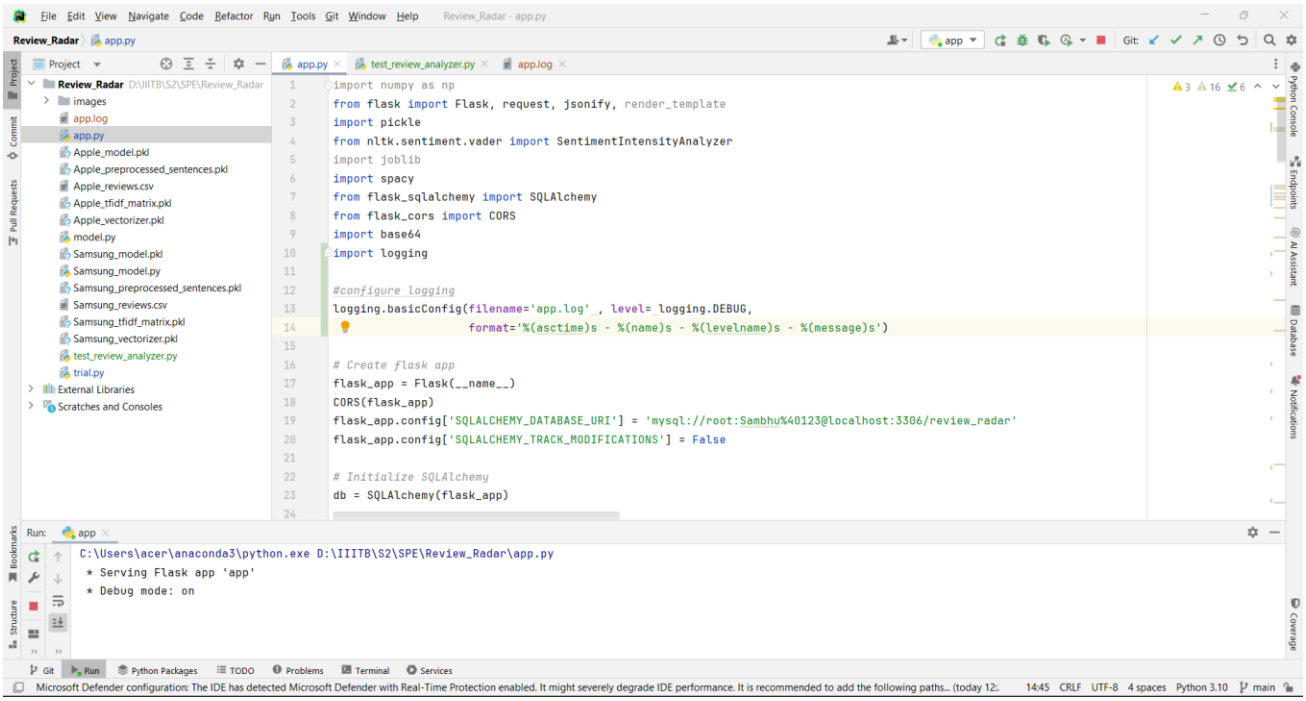
Continuous Monitoring

Implement the ELK Stack (Elasticsearch, Logstash, and Kibana) to monitor application performance and logs continuously.

DEVELOPMENT ENVIRONMENT

Backend Development in PyCharm (IDE):

PyCharm is a feature-rich Integrated Development Environment (IDE) for Python development. It offers tools for code completion, debugging, and version control integration, streamlining the development workflow.



Dependencies Installed with pip:

Project dependencies were installed using pip, ensuring access to required libraries for the Flask application.

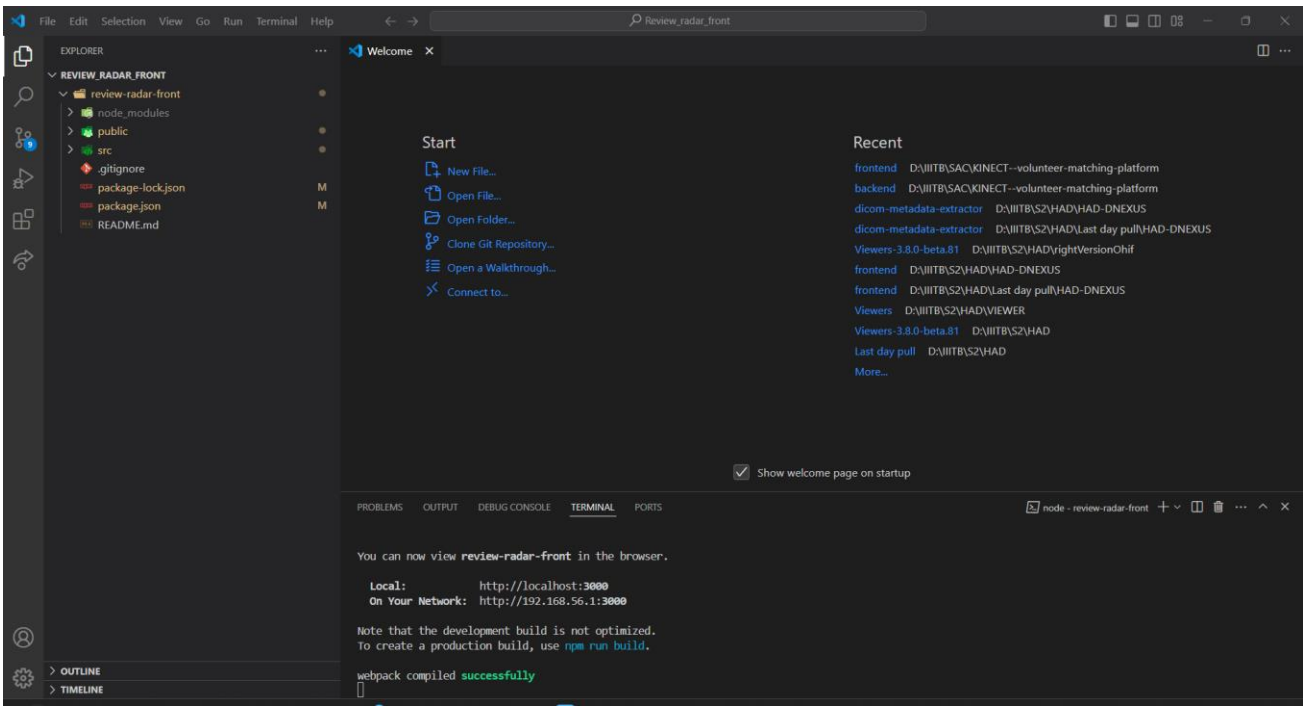
Libraries Used:

- Flask
- NLTK (Natural Language Toolkit) - Specifically, the SentimentIntensityAnalyzer module
- Joblib
- spaCy
- SQLAlchemy
- Flask-CORS
- Base64
- Logging

These libraries serve various purposes in the project, such as web development (Flask), sentiment analysis (NLTK), database interaction (SQLAlchemy), etc.

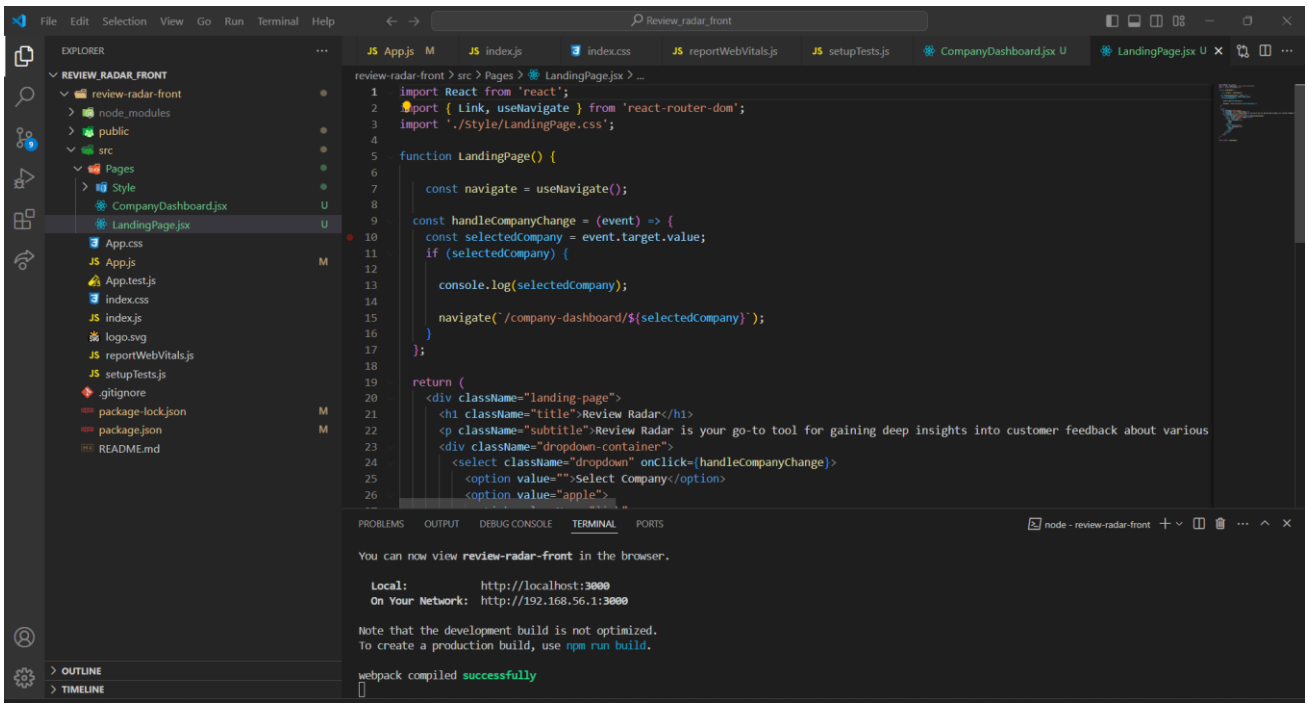
Frontend Development in Visual Studio Code (VSCode):

Visual Studio Code (VSCode) is a versatile code editor used for frontend development, offering features like IntelliSense, debugging, and Git integration.



React:

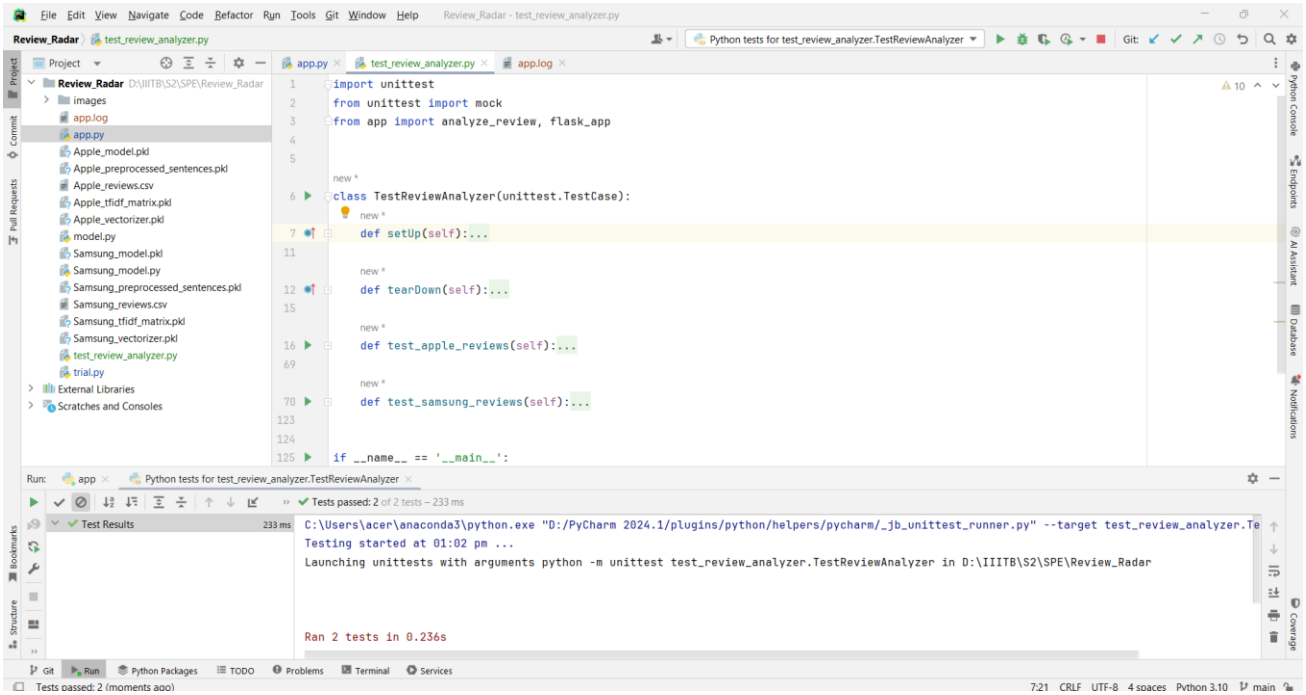
The frontend of the application was developed using React, a JavaScript library for building user interfaces.



TEST

Test Cases Using unittest Module:

Testing of the Flask application is performed using the unittest module, which provides a framework for organizing and running test cases.



- The **TestReviewAnalyzer** class contains test cases for analysing reviews of both Apple and Samsung products.
- The **setUp** and **tearDown** methods ensure that the Flask application context is set up and torn down before and after each test, respectively.
- Within each test method, reviews are simulated for both Apple and Samsung products, and the **analyze_review** function is called with the simulated data.
- Assertions are made to verify that the predicted topic and sentiment match the expected values for each review.

SOURCE CODE MANAGEMENT

Git

Git is a distributed version control system used for tracking changes in source code during software development. It allows multiple developers to collaborate on projects efficiently by providing features like branching, merging, and version history tracking.

Repository Setup:

Begin by initializing a Git repository in your project directory using the git init command. This creates a .git directory to store repository metadata.

Committing Changes:

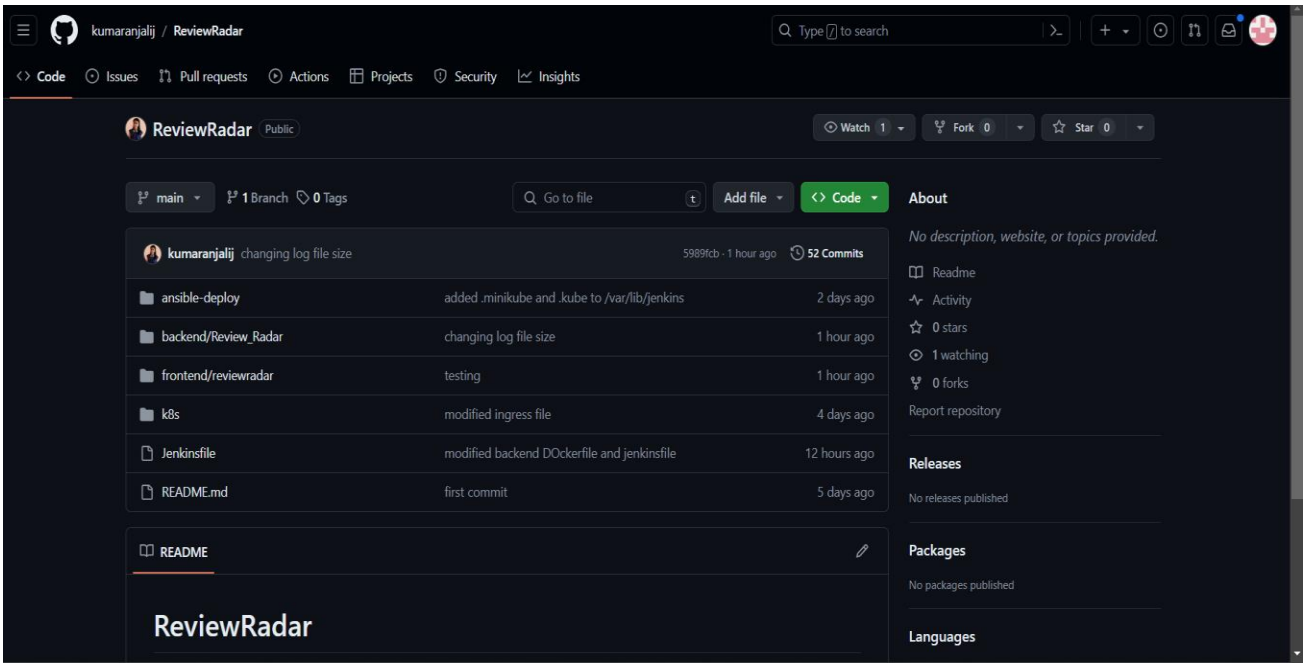
After making changes to your code, use the git add command to stage the modified files for commit. Then, commit the changes to the repository using git commit -m "commit message".

```
anjali@anjali-HP-Pavilion-Laptop-14-ec1xxx:~/SPE_FINAL_PROJECT/REVIEW-RADAR$ git branch
* main
anjali@anjali-HP-Pavilion-Laptop-14-ec1xxx:~/SPE_FINAL_PROJECT/REVIEW-RADAR$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Jenkinsfile

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        backend/.idea/
        backend/Review_Radar/.idea/
        backend/Review_Radar/app.log
        backend/Review_Radar/app.log.1
        backend/Review_Radar/app.log.10
        backend/Review_Radar/app.log.2
        backend/Review_Radar/app.log.3
        backend/Review_Radar/app.log.4
        backend/Review_Radar/app.log.5
        backend/Review_Radar/app.log.6
        backend/Review_Radar/app.log.7
        backend/Review_Radar/app.log.8
        backend/Review_Radar/app.log.9

no changes added to commit (use "git add" and/or "git commit -a")
anjali@anjali-HP-Pavilion-Laptop-14-ec1xxx:~/SPE_FINAL_PROJECT/REVIEW-RADAR$ gitadd Jenkinsfile
gitadd: command not found
anjali@anjali-HP-Pavilion-Laptop-14-ec1xxx:~/SPE_FINAL_PROJECT/REVIEW-RADAR$ git add Jenkinsfile
anjali@anjali-HP-Pavilion-Laptop-14-ec1xxx:~/SPE_FINAL_PROJECT/REVIEW-RADAR$ git commit -m "testing webhook"
[main e6f93f5] testing webhook
 1 file changed, 1 insertion(+), 1 deletion(-)
anjali@anjali-HP-Pavilion-Laptop-14-ec1xxx:~/SPE_FINAL_PROJECT/REVIEW-RADAR$ git push -u origin main
Username for 'https://github.com': kumaranjalij
Password for 'https://kumaranjalij@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 329 bytes | 329.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/kumaranjalij/ReviewRadar.git
   34b4fcb..e6f93f5  main -> main
branch 'main' set up to track 'origin/main'.
```

Version History and Log:

Git maintains a detailed version history of your project, allowing you to view commit logs, changes, and contributors. Use commands like git log to view commit history and git diff to compare changes between commits.

GitHub repository screenshot:



CONTINUOUS INTEGRATION WITH JENKINS

Jenkins is an open-source automation server that facilitates continuous integration and continuous delivery (CI/CD) pipelines. It automates the build, test, and deployment processes, allowing teams to integrate code changes frequently and detect issues early in the development cycle. Jenkins provides a highly extensible architecture with a vast ecosystem of plugins, enabling customization and integration with various tools and technologies.

Key Features of Jenkins:

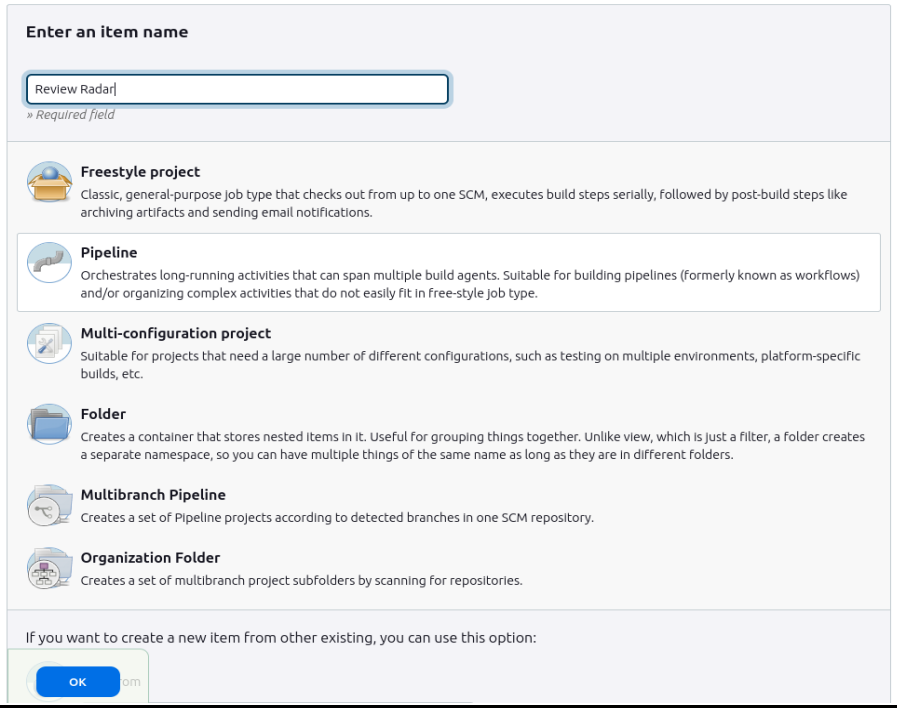
- **Automation**: Jenkins automates repetitive tasks such as code compilation, testing, and deployment, reducing manual effort and improving efficiency.
- **Scalability**: Jenkins scales easily to accommodate projects of any size, from small teams to large enterprises, supporting parallel and distributed builds.
- **Integration**: Jenkins integrates seamlessly with version control systems, issue trackers, and other development tools, enabling end-to-end automation of the software delivery pipeline.
- **Extensibility**: Jenkins offers a rich ecosystem of plugins that extend its functionality, allowing users to customize and adapt Jenkins to suit their specific requirements.

Jenkins CI/CD Pipelines:

Jenkins CI/CD pipelines define the stages and actions to be executed as part of the software delivery process. A typical pipeline includes stages for building, testing, and deploying applications, with each stage consisting of one or more tasks or jobs.

Access Jenkins: http://localhost:8080

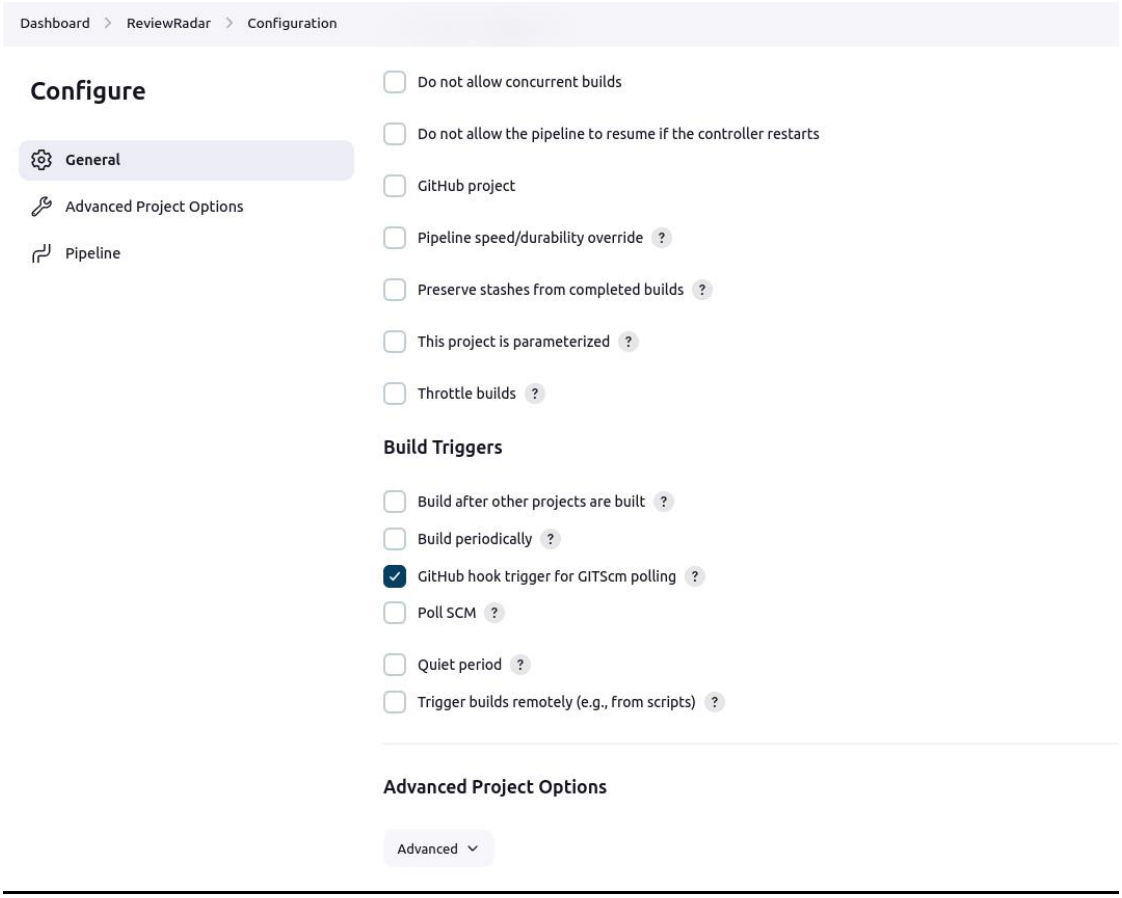
Create a Pipeline Project in Jenkins



Project Configuration Details

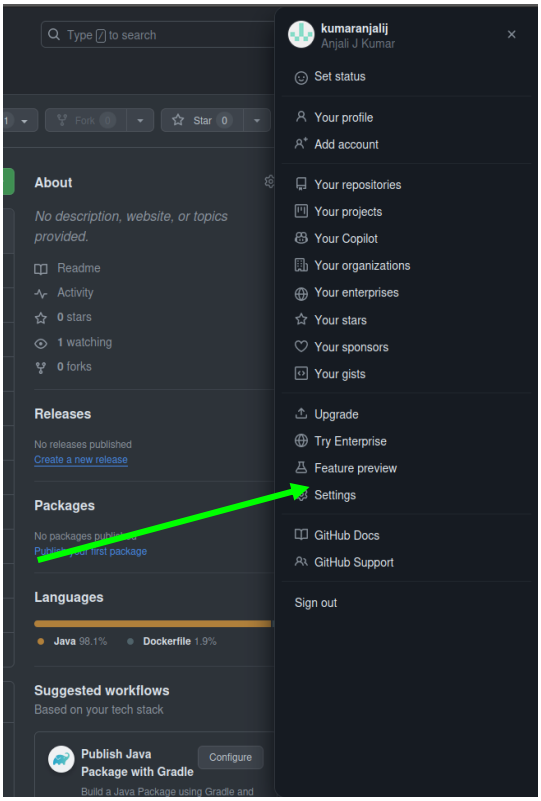
1. GitHub hook trigger for GITScm polling:

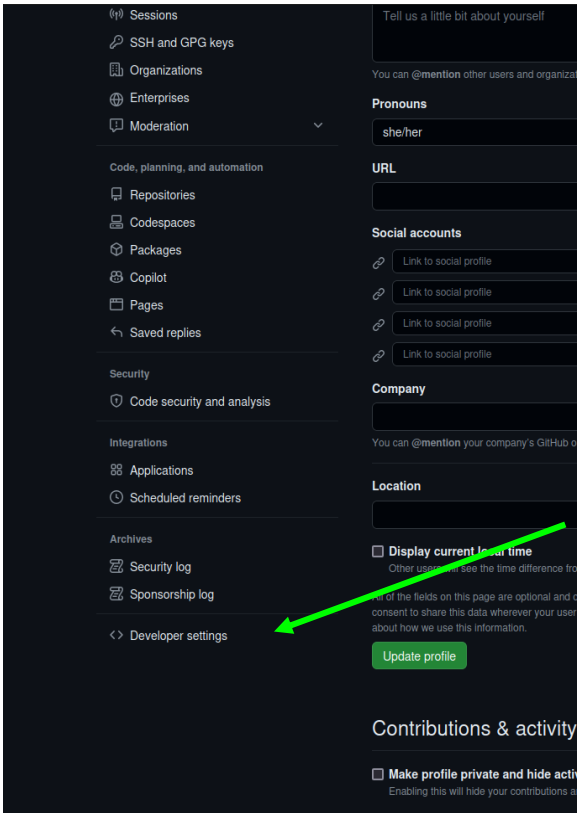
- ➢ Enabled GitHub hook trigger for GITScm polling in Jenkins to enable Git SCM polling for continuous integration.
- ➢ This configuration facilitates automatic triggering of pipeline builds upon code changes in the associated GitHub repository.

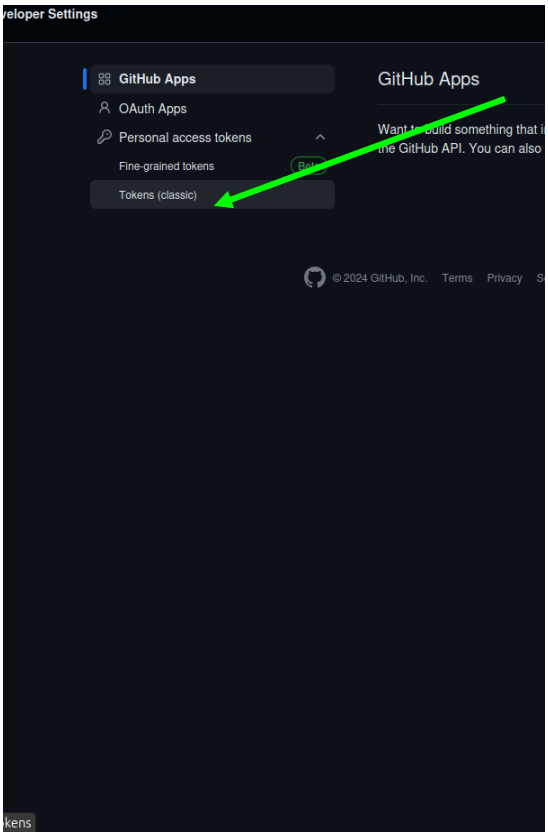


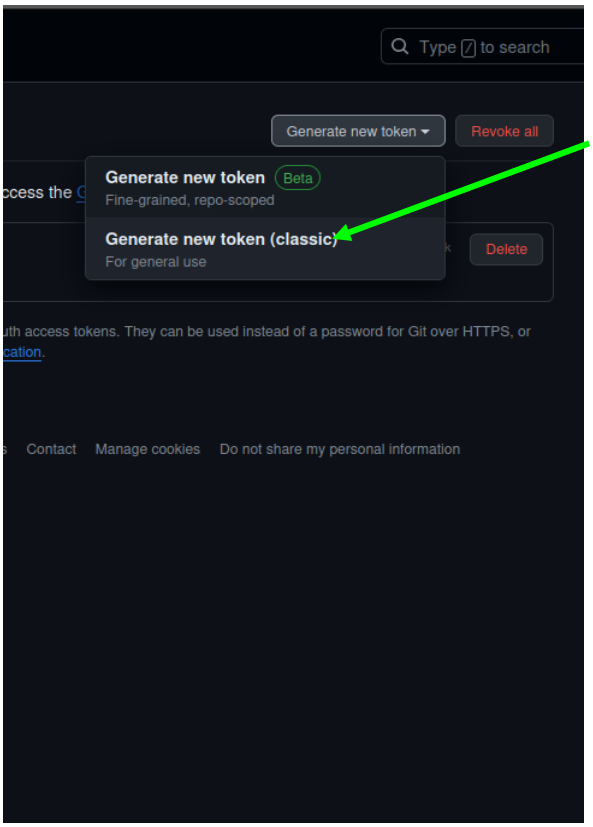
2. Integration with GitHub Repository with Jenkins:

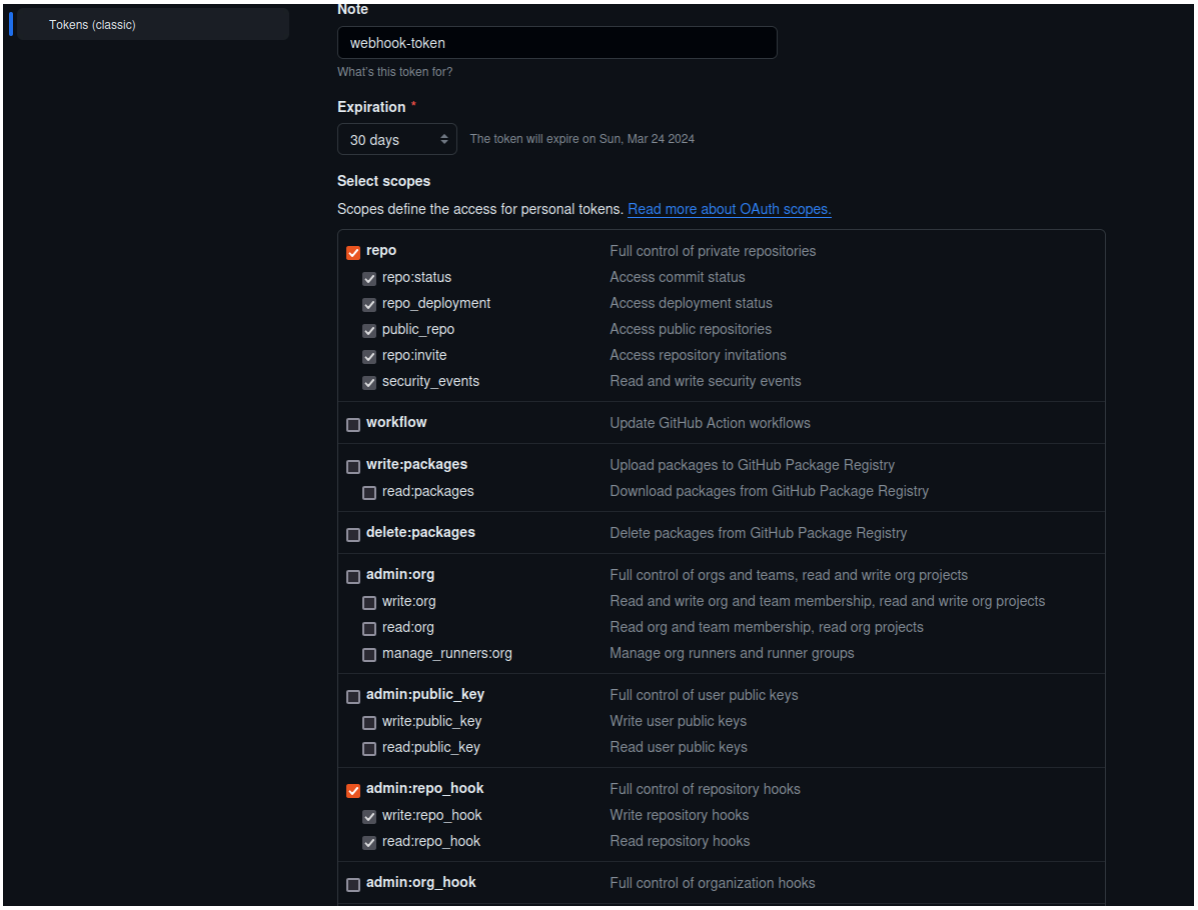
- ➢ Established seamless integration between Jenkins and project GitHub repository.
- ➢ Git is the Source Control Management tool for version control.
- ➢ Secured the integration by generating a token within GitHub with requisite permissions (repo and admin: repo hook).
- ➢ Used the token as credentials for authentication, ensuring access to the private repository.

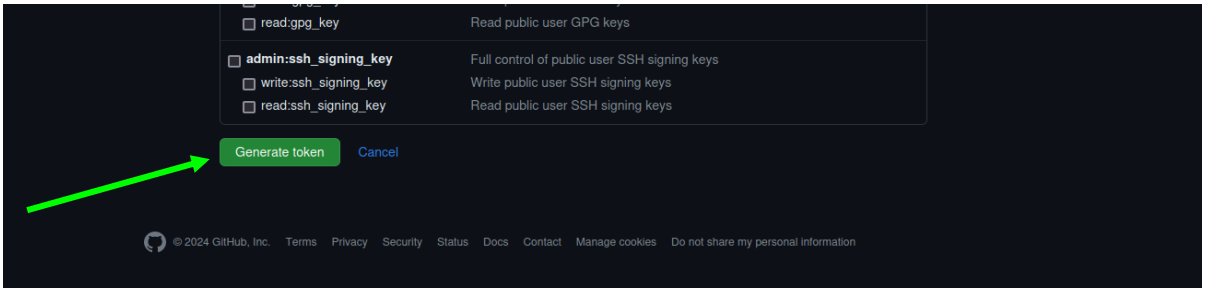

Type / to search
kumaranjalij
Anjali J Kumar
Set status
Your profile
Add account
Your repositories
Your projects
Your Copilot
Your organizations
Your enterprises
Your stars
Your sponsors
Your gists
Upgrade
Try Enterprise
Feature preview
Settings
GitHub Docs
GitHub Support
Sign out
About
No description, website, or topics provided.
Readme
Activity
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
Languages
Java 98.1%
Dockerfile 1.9%
Suggested workflows
Based on your tech stack
Publish Java Package with Gradle
Configure

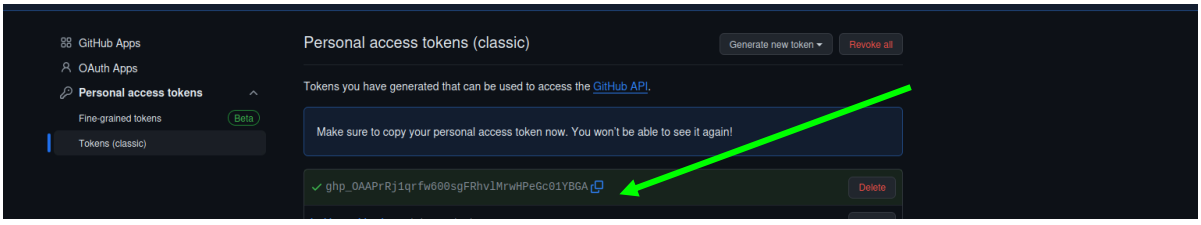

Sessions
SSH and GPG keys
Organizations
Enterprises
Moderation
Code, planning, and automation
Repositories
Codespaces
Packages
Copilot
Pages
Saved replies
Security
Code security and analysis
Integrations
Applications
Scheduled reminders
Archives
Security log
Sponsorship log
Developer settings
Tell us a little bit about yourself
Pronouns
she/her
URL
Social accounts
Link to social profile
Company
Location
Update profile
Contributions & activity


eloper Settings
GitHub Apps
OAuth Apps
Personal access tokens
Fine-grained tokens
Tokens (classic)
GitHub Apps
© 2024 GitHub, Inc.
Terms
Privacy


Type / to search
Generate new token
Revoke all
Generate new token Beta
Fine-grained, repo-scoped
Generate new token (classic)
For general use
Delete
Contact
Manage cookies
Do not share my personal information



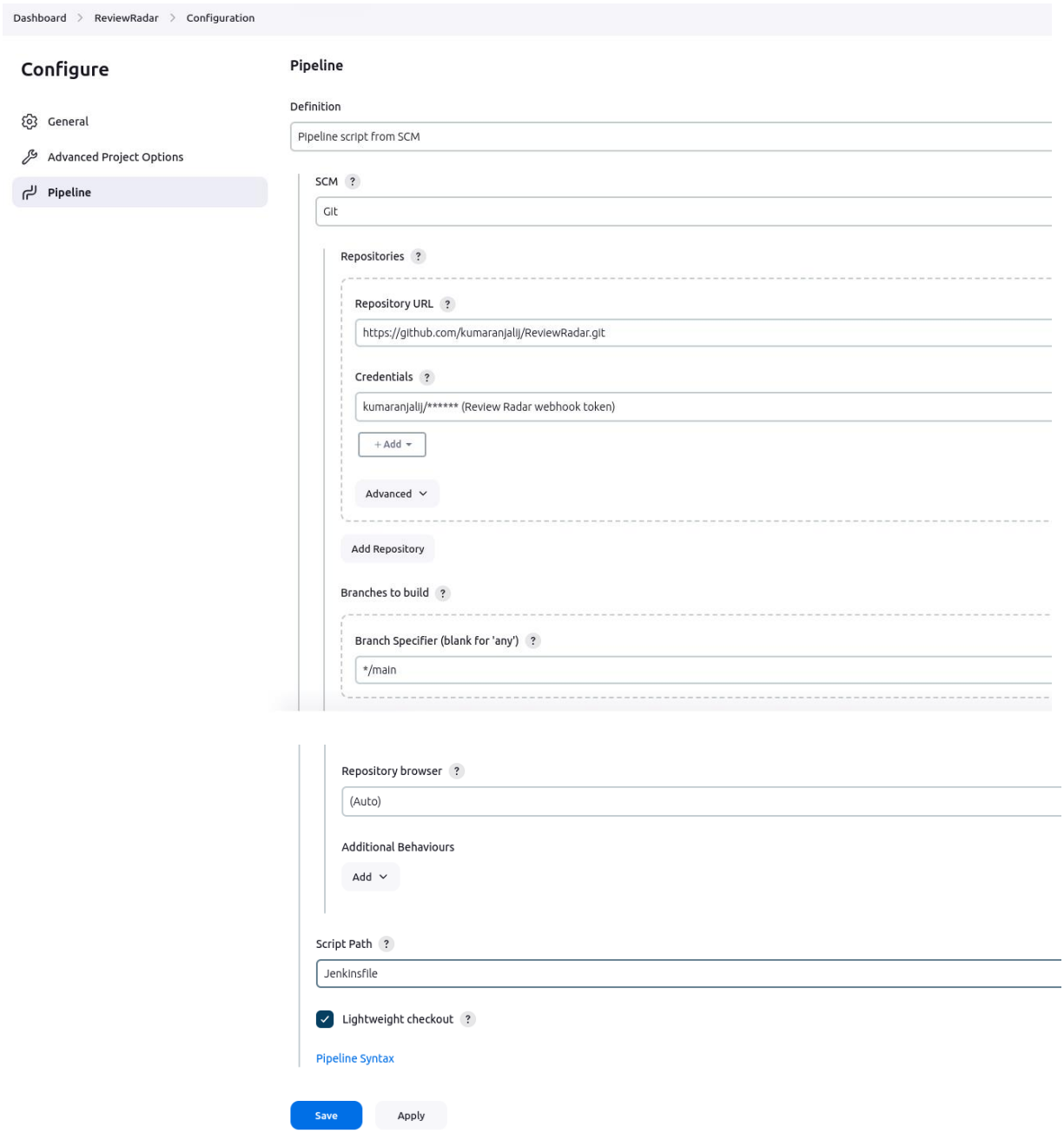




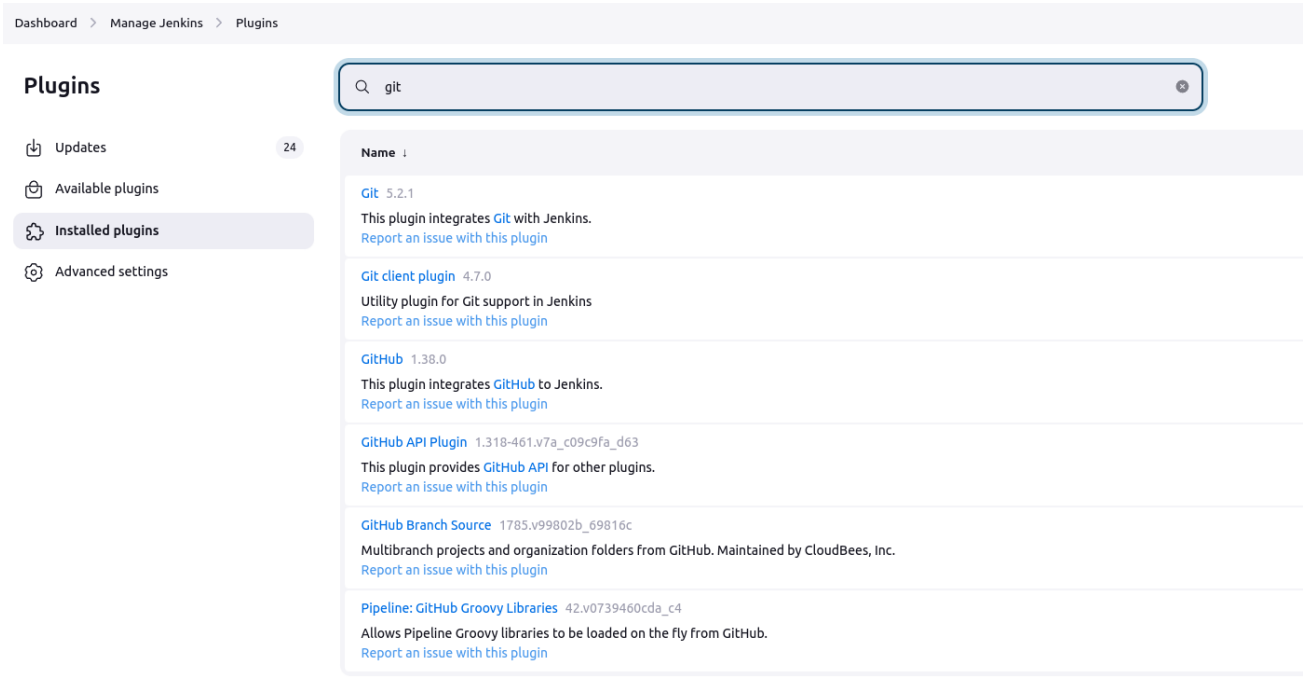
Copy and save the token for future use.

3. Pipeline script from SCM:

- ➢ Selected the "Pipeline script from SCM" option in Jenkins.
- ➢ Specified Git as the SCM tool.
- ➢ Provided the GitHub repository URL in Jenkins configuration.
- ➢ Configured Jenkins to authenticate with the GitHub repository using the created webhook-token.
- ➢ Defined the script path by naming the pipeline script file (jenkinsfile), ensuring proper execution of the pipeline upon triggering.



Git Plugins



Pipeline Script for Continuous Integration

This Jenkins pipeline automates the process of checking out code, building and testing the backend, building the frontend, creating Docker images, pushing these images to Docker Hub, and finally deploying the application using Ansible. The stages of the pipeline are as follows:

1. **GIT CHECKOUT**:
    - This stage checks out the code from the specified GitHub repository. It uses the **git** command with the branch **main** and credentials identified by **ReviewRadar-webhook**. This ensures that the latest code from the repository is available for subsequent stages.
2. **BACKEND BUILD AND TEST**:
    - In this stage, the backend application is built and tested. It activates a Python virtual environment and installs necessary dependencies listed in the **requirements.txt** file. The Natural Language Toolkit (NLTK) **vader_lexicon** and spaCy's **en_core_web_sm** model are also downloaded. After the dependencies are installed, the backend application is started using **nohup** to run in the background, with logs directed to **app.log**.
3. **FRONTEND BUILD**:
    - This stage handles the building of the frontend application. It changes to the frontend project directory (**./frontend/reviewradar/**) and runs **npm install** to install

dependencies. Subsequently, **npm run build** is executed to create a production build of the frontend application.

```
pipeline {
    agent any

    environment{
        DOCKER_IMAGE_NAME_FRONTEND = 'anjalijkumar/review-radar-frontend'
        DOCKER_IMAGE_NAME_BACKEND = 'anjalijkumar/review-radar-backend'
        GITHUB_REPO_URL = 'https://github.com/kumaranjalij/ReviewRadar.git'
        DOCKERHUB_CREDENTIALS = credentials('DockerHubCred')
    }

    stages {
        stage('GIT CHECKOUT') {
            steps {
                script {
                        // Checkout the code from the GitHub repo
                        git branch: 'main', credentialsId: 'ReviewRadar-webhook', url: "${GITHUB_REPO_URL}"
                }
            }
        }


        stage('BACKEND BUILD AND TEST') {
            steps {
                // Activate the virtual environment
                withEnv(['PATH+VENV=/var/lib/jenkins/workspace/ReviewRadar/rrenv/bin']) {
                    // Install packages from requirements.txt within the virtual environment
                    sh 'python3 -m nltk.downloader vader_lexicon'
                    sh 'python3 -m spacy download en_core_web_sm'

                    sh 'pip install -r ./backend/Review_Radar/requirements.txt'

                    // Run your Python script within the virtual environment
                    sh 'nohup python3 ./backend/Review_Radar/app.py > app.log 2>&1 &'
                }
            }
        }

```

```
        stage('FRONTEND BUILD') {
            steps {
                // Change directory to your specific directory
                dir('./frontend/reviewradar/') {
                    // Run npm install and npm run build commands
                    sh 'npm install'
                    sh 'npm run build'
                }
            }
        }


        stage('BUILD DOCKER IMAGES') {
            steps {
                script {
                     // Build and tag backend Docker image
                     //docker.build("${DOCKER_IMAGE_NAME_BACKEND}", './backend/Review_Radar/')
                     sh 'docker build -t "${DOCKER_IMAGE_NAME_BACKEND}" ./backend/Review_Radar/'

                     // Build and tag frontend Docker image
                     //docker.build("${DOCKER_IMAGE_NAME_FRONTEND}", './frontend/reviewradar/')
                     sh 'docker build -t "${DOCKER_IMAGE_NAME_FRONTEND}" ./frontend/reviewradar/'
                }
            }
        }

        stage('PUSH DOCKER IMAGE') {
            steps {
                script{
                    sh 'echo "Logging in to Docker Hub"'
                    sh """
                    echo \$DOCKERHUB_CREDENTIALS_PSW | docker login -u \$DOCKERHUB_CREDENTIALS_USR --password-stdin
                    """

                    sh 'echo "pushing frontend"'
                    sh 'docker push anjalijkumar/review-radar-frontend:latest'

                    sh 'echo "pushing backend"'
                    sh 'docker push anjalijkumar/review-radar-backend:latest'

                }
            }
        }


        stage('RUN ANSIBLE PLAYBOOK') {
            steps {
                script {
                    ansiblePlaybook(
                        playbook: './ansible-deploy/deploy.yml',
                        inventory: './ansible-deploy/inventory'
                    )
                }
            }
        }
    }
```

4. **BUILD DOCKER IMAGES**:
    - In this stage, Docker images for both the backend and frontend applications are built. The backend image is created using the Dockerfile located in **./backend/Review_Radar/**, and the frontend image is built from the Dockerfile in **./frontend/reviewradar/**. Each image is tagged appropriately using the environment variables **DOCKER_IMAGE_NAME_BACKEND** and **DOCKER_IMAGE_NAME_FRONTEND**.
5. **PUSH DOCKER IMAGE**:
    - This stage pushes the built Docker images to Docker Hub. First, it logs into Docker Hub using credentials stored in the environment variables **DOCKERHUB_CREDENTIALS_USR** and **DOCKERHUB_CREDENTIALS_PSW**. Then, it pushes the frontend and backend images to their respective repositories on Docker Hub.
6. **RUN ANSIBLE PLAYBOOK**:
    - The final stage runs an Ansible playbook to deploy the application onto a Kubernetes cluster. The **ansiblePlaybook** function is used to execute the playbook defined in **./ansible-deploy/deploy.yml** with the inventory file located at **./ansible-deploy/inventory**. This stage ensures that the latest versions of the application are deployed and running in the specified environment.

Ngrok

Ngrok provides a secure way to share and test web applications by creating temporary, publicly accessible URLs that can be accessed from anywhere.

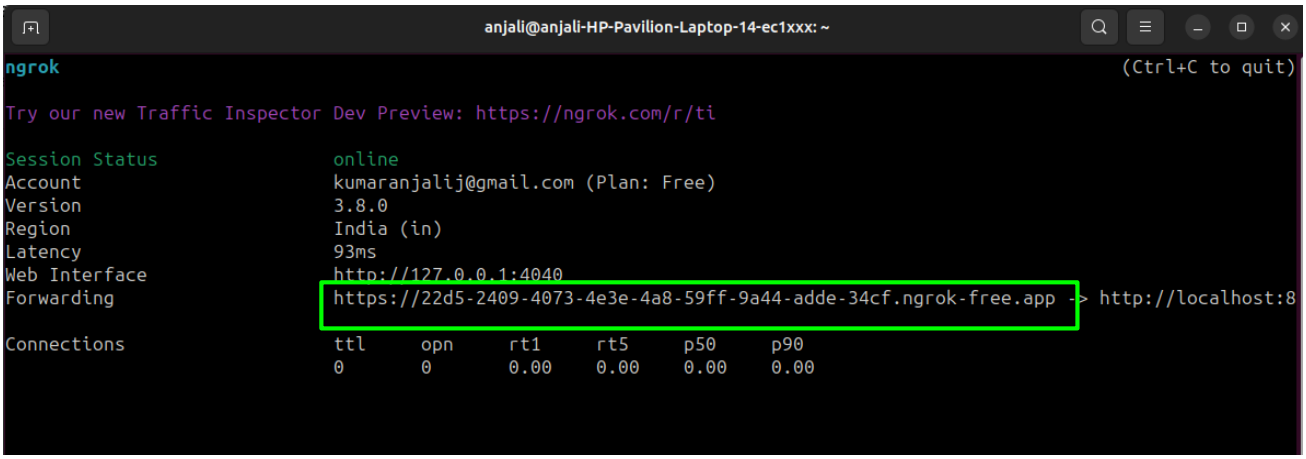
Installation

1. Sign up in https://ngrok.com/
2. Download ngrok from:
   https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.tgz
3. Then extract ngrok from the terminal:
   $sudo tar xvzf ~/Downloads/ngrok-stable-linux-amd64.tgz -C /usr/local/bin
4. Copy Authtoken from:
   https://dashboard.ngrok.com/get-started/your-authtoken
5. Add Authtoken:
   $ngrok authtoken <token>

How to get public IP address

Execute the following command and copy the public ip address for your local host.

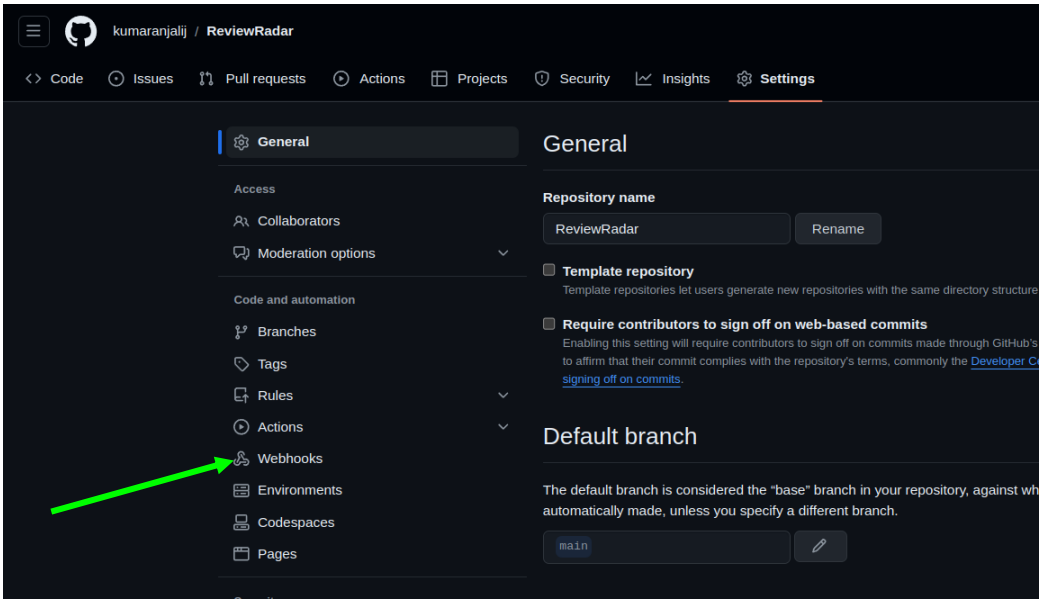
$ngrok http 8080

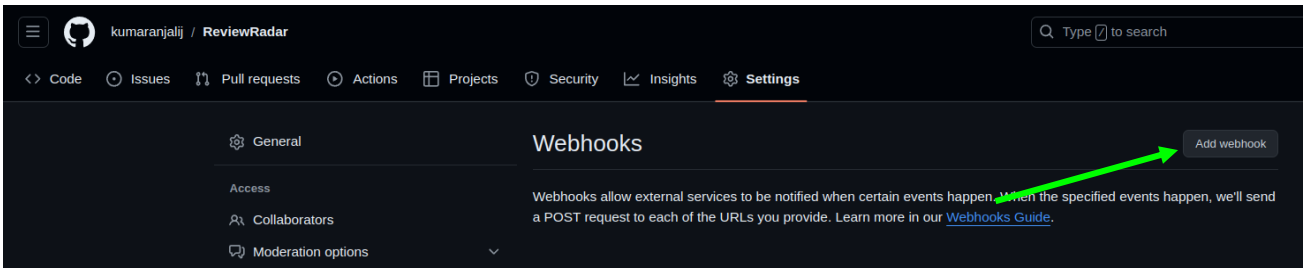


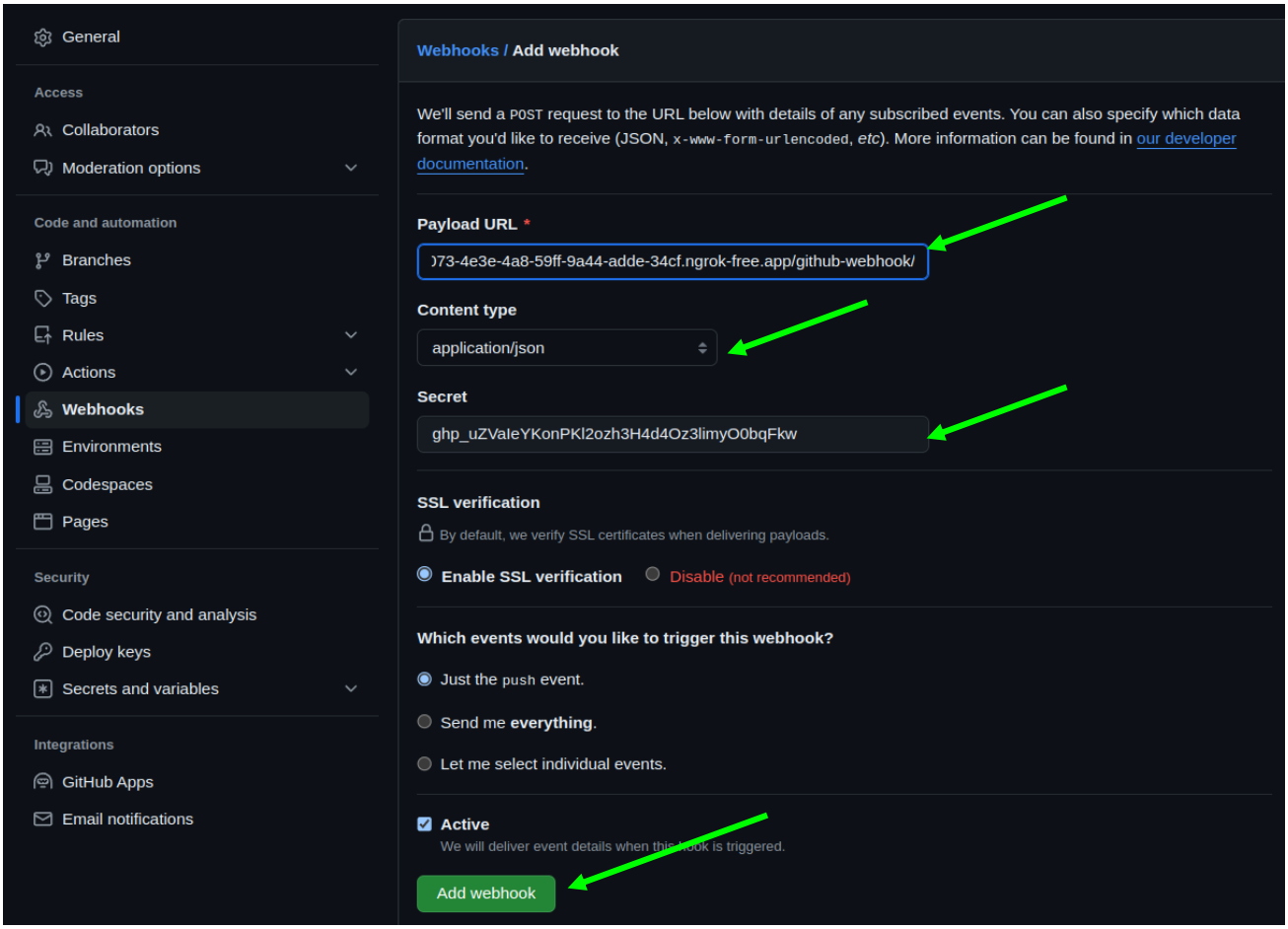
GitHub Webhook

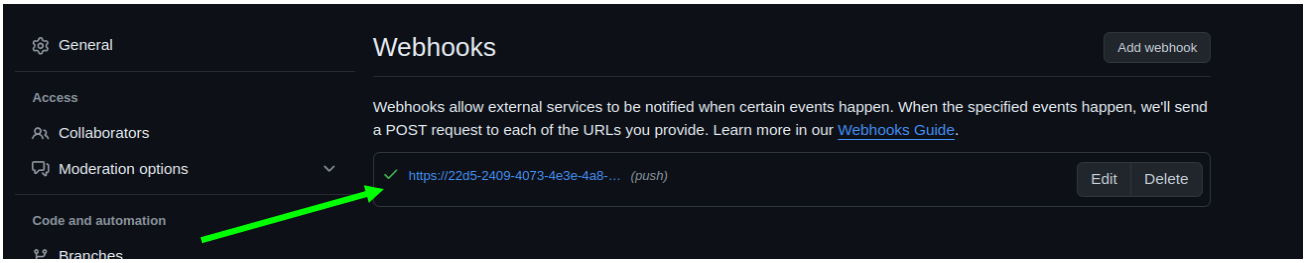
Add a webhook in the GitHub repository with ngrok's public URL .The webhook-token created earlier is used as the secret key.

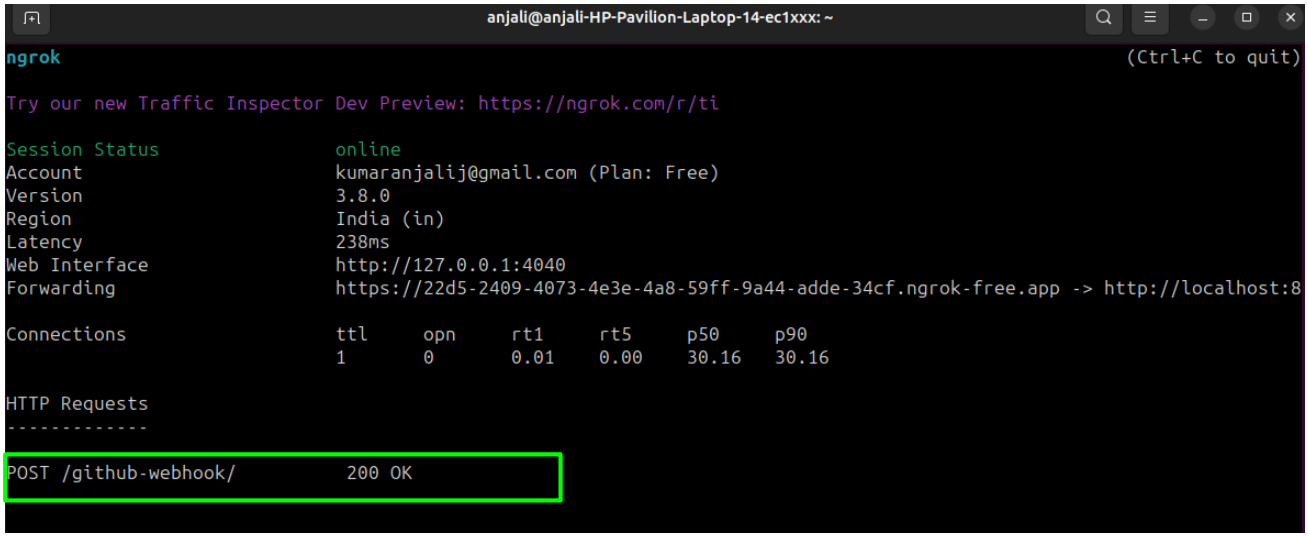
- Navigate to your GitHub repository.
- Go to the repository's settings.
- Select "Webhooks" or "Webhooks & services" from the menu.
- Click on "Add webhook".
- In the "Payload URL" field, paste the ngrok public URL.
- Set the "Content type" to application/json.
- In the "Secret" field, paste the webhook-token.
- Choose the events you want the webhook to trigger on. Typically, this includes events like push, pull request, etc.
- Optionally, you can set other configurations such as SSL verification and webhook status.




kumaranjalij / ReviewRadar
Type / to search
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Settings
General
Access
Collaborators
Moderation options
Webhooks
Add webhook
Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.


General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
Security
Code security and analysis
Deploy keys
Secrets and variables
Integrations
GitHub Apps
Email notifications
Webhooks / Add webhook
We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our developer documentation.
Payload URL *
)73-4e3e-4a8-59ff-9a44-adde-34cf.ngrok-free.app/github-webhook/
Content type
application/json
Secret
ghp_uZValeYKonPKl2ozh3H4d4Oz3limyO0bqFkw
SSL verification
By default, we verify SSL certificates when delivering payloads.
Enable SSL verification
Disable (not recommended)
Which events would you like to trigger this webhook?
Just the push event.
Send me everything.
Let me select individual events.
Active
We will deliver event details when this hook is triggered.
Add webhook


General
Access
Collaborators
Moderation options
Code and automation
Branches
Webhooks
Add webhook
Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.
https://22d5-2409-4073-4e3e-4a8-... (push)
Edit
Delete

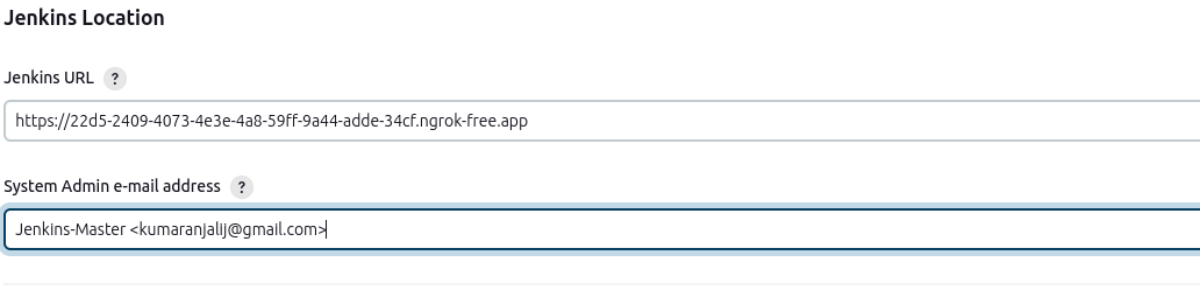


Configure ngrok in jenkins

Configure ngrok URL, GitHub API URL, and provide webhook token as credentials in Jenkins, to ensure that Jenkins is correctly set up to communicate with GitHub via ngrok, and it has the necessary authentication credentials to interact with GitHub's webhook system securely. To configure ngrok in Jenkins and manage webhook settings for GitHub integration, you would follow these steps:

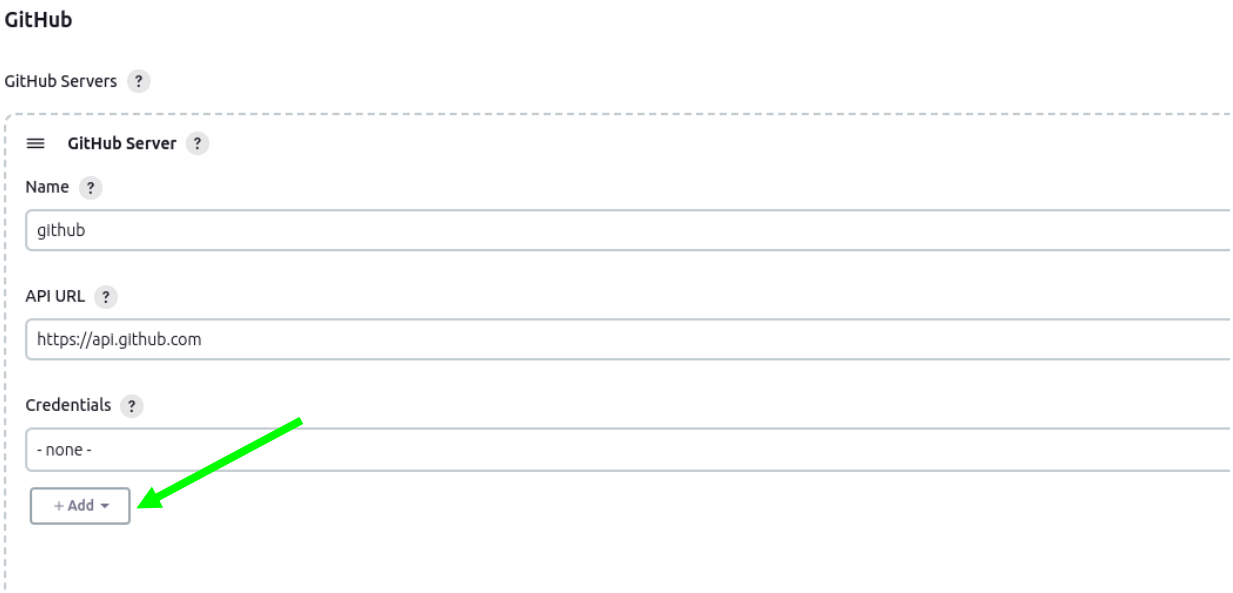
1. Configure Ngrok URL in Jenkins:

- Navigate to "Manage Jenkins" > "System".
- Look for the "Jenkins URL" field and paste the ngrok public URL.



2. GitHub API URL Configuration:

- On the same page, you may have an option to configure GitHub related settings.
- Find the field for "GitHub API URL" and ensure it's correctly configured.
- Usually, it should be set to https://api.github.com.

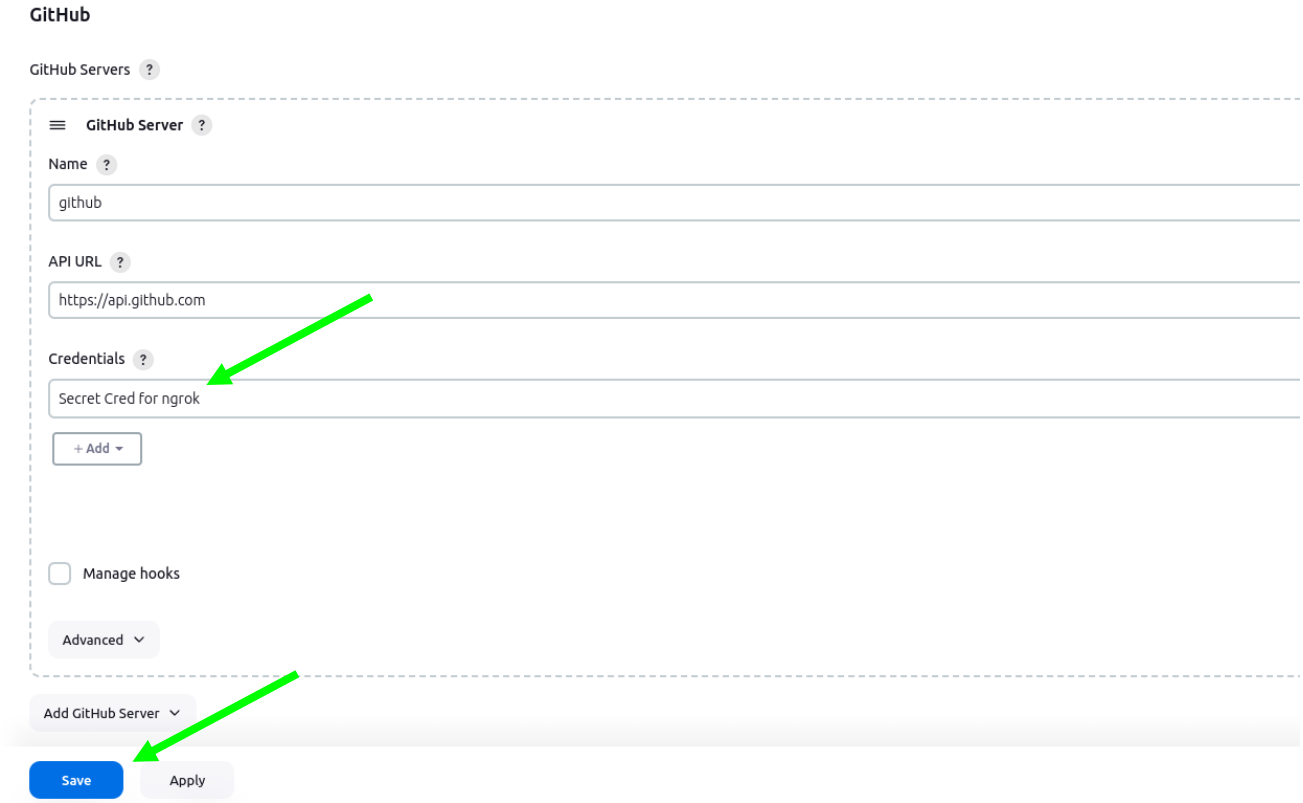


3. Configure Credentials:

- Click on "Add" to add new credentials.
- Choose the type of credentials as “Secret Text”.
- Enter the webhook token in the "Secret" field.
- Optionally, provide an ID and description for the credentials.

4. Save Changes:

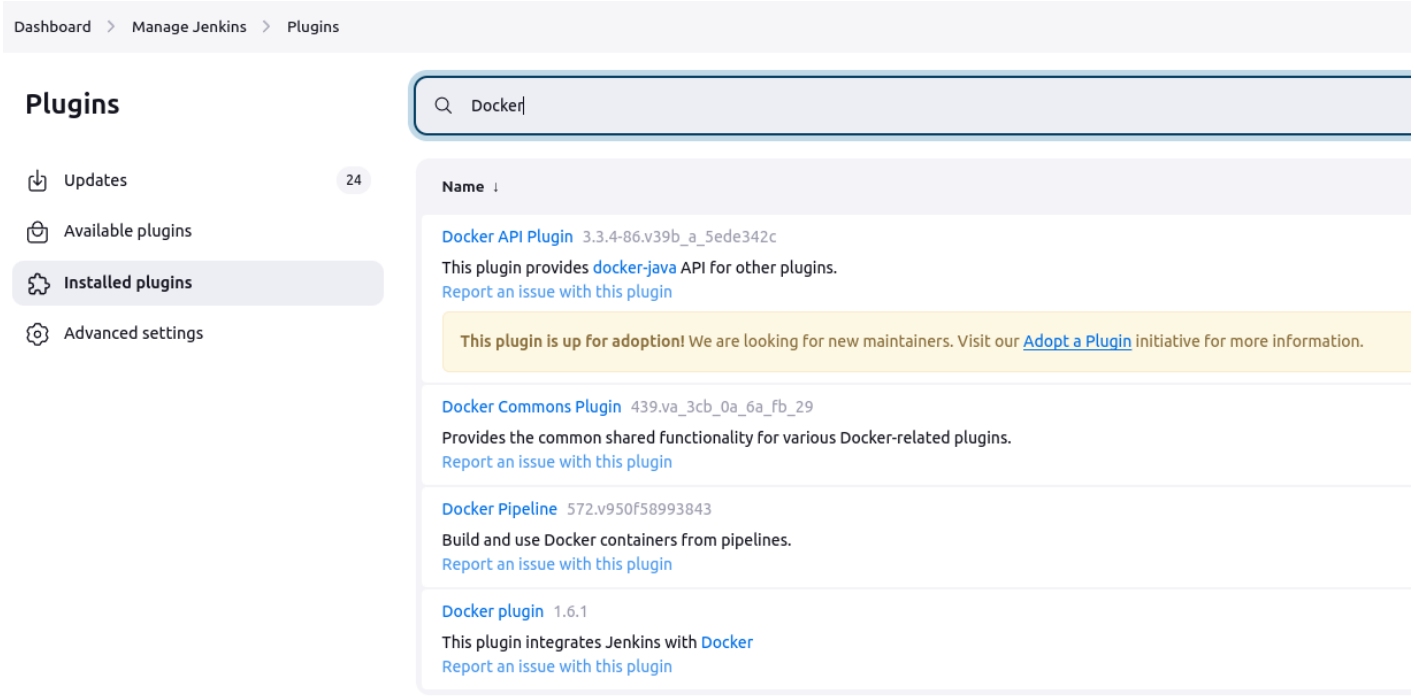
- Once you've configured the ngrok URL, GitHub API URL, and added the webhook token as credentials, click on "Save" or "Apply" to save your changes.



CONTAINERIZATION WITH DOCKER

Docker is a platform for developing, shipping, and running applications inside containers. Containerization allows applications and their dependencies to be packaged together in a consistent environment, ensuring that they run reliably across different computing environments. By containerizing both the frontend and backend components of your application, you can streamline deployment, improve scalability, and enhance portability.

Docker Plugins



Creating Docker Containers:

Backend Container:

Docker containers encapsulate the backend Flask application along with its dependencies, such as libraries and configurations. This container exposes the Flask application's endpoints, allowing it to receive requests and respond accordingly.

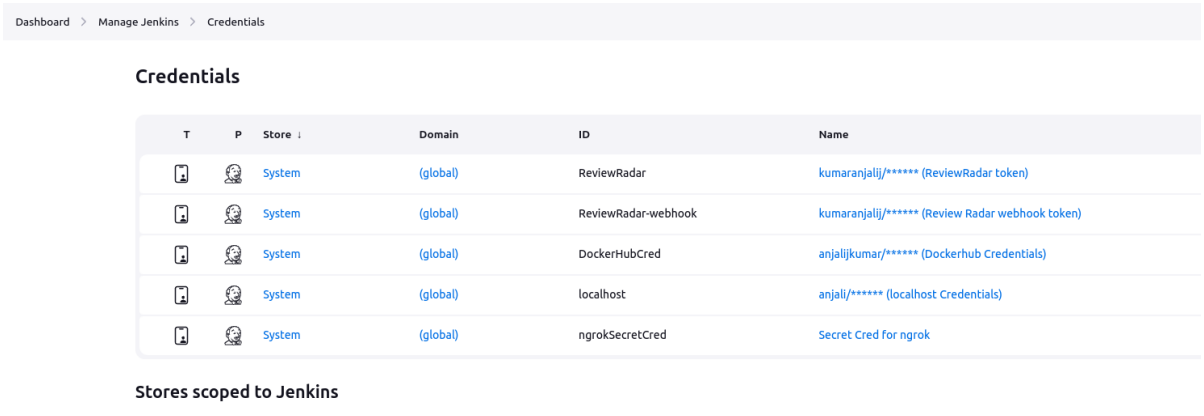
```dockerfile
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Install system-level dependencies
RUN apt-get update \
    && apt-get install -y pkg-config \
                          libmariadb-dev-compat \
                          gcc \
    && rm -rf /var/lib/apt/lists/*

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file into the container at /app
COPY requirements.txt /app/

# Install any needed dependencies specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

RUN python -c "import nltk; nltk.download('vader_lexicon')"

RUN python -m spacy download en_core_web_sm

# Copy the rest of the application code into the container at /app
COPY . /app/

# Make port 5001 available to the world outside this container
EXPOSE 5001

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python3", "app.py"]
```

Frontend Container:

The frontend React application is packaged into a separate Docker container, which contains the necessary libraries, assets, and configurations. This container serves the static files and assets required to render the frontend user interface.

```
# Use an official Node.js runtime as a parent image
FROM node:14-alpine

# Set the working directory in the container
WORKDIR /app

# Copy package.json and package-lock.json to the container
COPY package*.json ./

# Install dependencies
RUN npm install
RUN npm install axios
RUN npm install react-router-dom

# Copy the rest of the application code to the container
COPY . .

# Expose port 3000
EXPOSE 3000

# Command to start serving the React application
CMD ["npm", "start"]
```

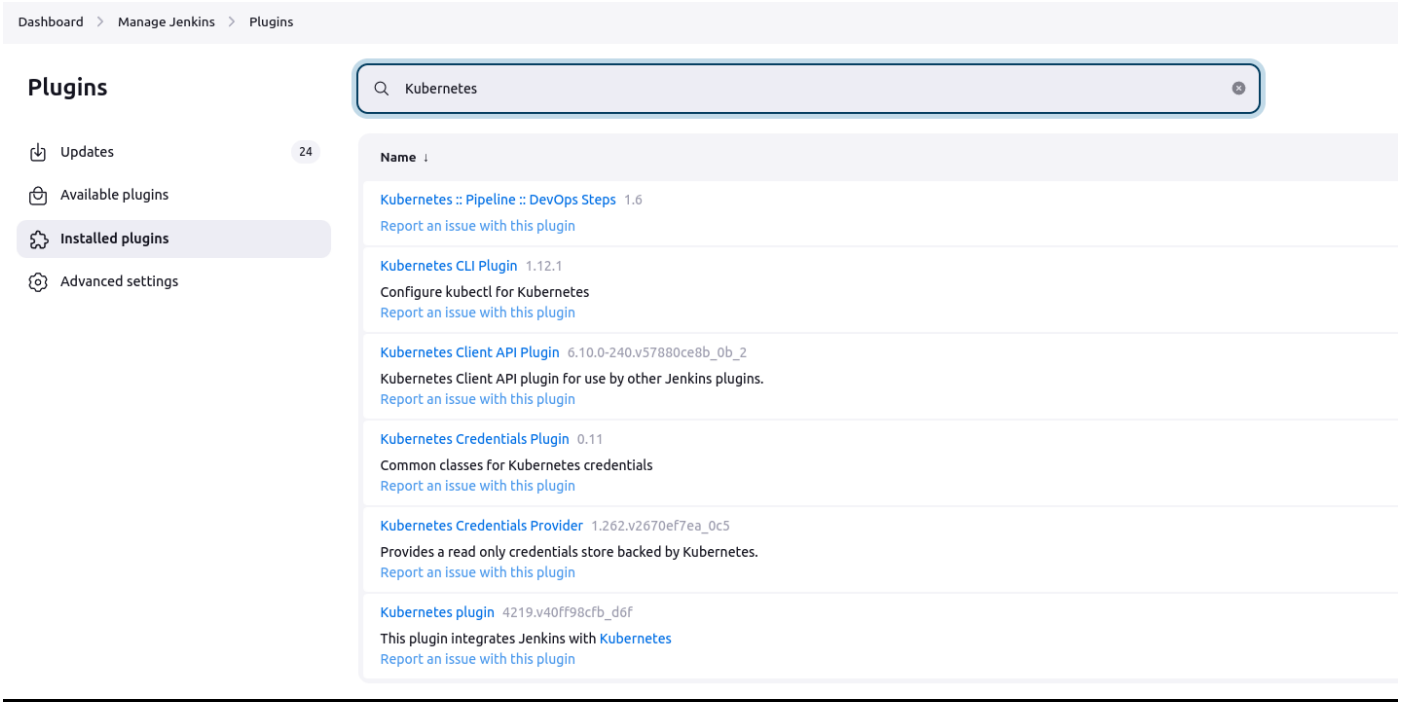
Setting Global Credentials



DEPLOYMENT WITH KUBERNETES

Kubernetes is a powerful container orchestration platform that automates the deployment, scaling, and management of containerized applications. It provides features such as service discovery, load balancing, and automated rollouts and rollbacks, which are essential for deploying applications in production environments. By using Kubernetes, you can ensure high availability, scalability, and reliability for your containerized application.

Kubernetes Plugins



Kubernetes Deployment Configuration:

Backend deploy

This Kubernetes Deployment configuration defines a deployment named "review-radar-backend" with one replica, meaning it ensures that one instance of the backend application is running. Within this deployment, a container named "review-radar-backend" is specified. This container is based on the Docker image "anjalijkumar/review-radar-backend:latest", which contains the necessary code and dependencies for the backend application. The deployment exposes port 5001, allowing other components within the Kubernetes cluster to communicate with the backend application. This configuration ensures the reliable deployment and availability of the backend application within the Kubernetes environment, facilitating seamless communication and interaction with other services or components.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: review-radar-backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: review-radar-backend
  template:
    metadata:
      labels:
        app: review-radar-backend
    spec:
      containers:
        - name: review-radar-backend
          image: anjalijkumar/review-radar-backend:latest
          ports:
            - containerPort: 5001
```

Frontend Deploy

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: review-radar-frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: review-radar-frontend
  template:
    metadata:
      labels:
        app: review-radar-frontend
    spec:
      containers:
        - name: review-radar-frontend
          image: anjalijkumar/review-radar-frontend:latest
          ports:
            - containerPort: 3000
```

This Kubernetes Deployment configuration defines a deployment named "review-radar-frontend" with one replica, ensuring the availability of one instance of the frontend application. Within this deployment, a container named "review-radar-frontend" is specified, which is based on the Docker image "anjalijkumar/review-radar-frontend:latest". This Docker image contains the necessary code and dependencies for the frontend application. The deployment exposes port 3000, allowing other components within the Kubernetes cluster to communicate with the frontend application. Overall,

this configuration facilitates the reliable deployment and accessibility of the frontend application within the Kubernetes environment, enabling seamless interaction with users and other services.

Kubernetes Service Configuration:

Backend service config

```yaml
apiVersion: v1
kind: Service
metadata:
  name: review-radar-backend-service
spec:
  selector:
    app: review-radar-backend
  ports:
    - protocol: TCP
      port: 5001
      targetPort: 5001
  #type: NodePort
```

This Kubernetes Service configuration defines a service named "review-radar-backend-service" that acts as a bridge between clients and the backend application deployed within the Kubernetes cluster. The service ensures that external or internal components can communicate with the backend application seamlessly.

The service is configured to route traffic to pods with the label "app: review-radar-backend", meaning it directs requests to instances of the backend application. This selector ensures that the service forwards traffic specifically to pods running the backend application.

The service exposes port 5001, which is the port on which the backend application is listening inside the pods. When requests are received on port 5001, the service forwards them to the corresponding pods.

Additionally, the service specifies that it uses the TCP protocol and maps port 5001 of the service to port 5001 of the backend application pods using the "targetPort" attribute.

This configuration ensures that the backend application is accessible within the Kubernetes cluster through the "review-radar-backend-service", allowing other components or services to interact with it reliably.

Frontend service config

```yaml
apiVersion: v1
kind: Service
metadata:
  name: review-radar-frontend-service
spec:
  selector:
    app: review-radar-frontend
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
  #type: NodePort
```

This Kubernetes Service configuration defines a service named "review-radar-frontend-service" responsible for routing traffic to the frontend application deployed within the Kubernetes cluster. The service acts as an entry point, enabling external or internal clients to communicate with the frontend application seamlessly.

The service is configured to select pods with the label "app: review-radar-frontend", ensuring that it directs incoming traffic specifically to instances of the frontend application.

It exposes port 3000, which is the port on which the frontend application is listening inside the pods. Incoming requests received on port 3000 are forwarded to the corresponding pods running the frontend application.

This service uses the TCP protocol to communicate with the backend application, and it maps port 3000 of the service to port 3000 of the frontend application pods using the "targetPort" attribute.

The configuration facilitates reliable access to the frontend application within the Kubernetes cluster through the "review-radar-frontend-service", allowing other components or services to interact with it as needed.

Ingress File

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: reviewradar-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    kubernetes.io/ingress.class: "nginx"
spec:
  ingressClassName: nginx
  rules:
    - host: localhost
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: review-radar-backend-service
                port:
                  number: 5001
          - path: /
            pathType: Prefix
            backend:
              service:
                name: review-radar-frontend-service
                port:
                  number: 3000
```

This Kubernetes Ingress configuration facilitates external access to services within the cluster through defined routing rules:

Annotations: These provide additional configuration options for the Ingress controller. For instance:

- "ingress.kubernetes.io/rewrite-target": Rewrites incoming request paths to "/".
- "kubernetes.io/ingress.class": Specifies the Ingress controller class as "nginx".

Spec

- **IngressClassName**: Specifies the Ingress controller class as "nginx".
- **Rules**: These define routing based on hostname and path:
  - **Host**: Identifies the hostname to which the rules apply, in this case, "localhost".
  - **HTTP Paths**: Define routing based on path prefix:
    - **Path**: Specifies the path prefix.

- **PathType**: Indicates the type of matching used for the path, here, "Prefix".
- **Backend**: Specifies the backend service to which requests are forwarded:
    - **Service Name**: Name of the backend service.
    - **Port Number**: Port of the backend service

This setup ensures that incoming HTTP requests are directed to the appropriate backend services based on defined rules, enabling efficient external access to services within the Kubernetes cluster.

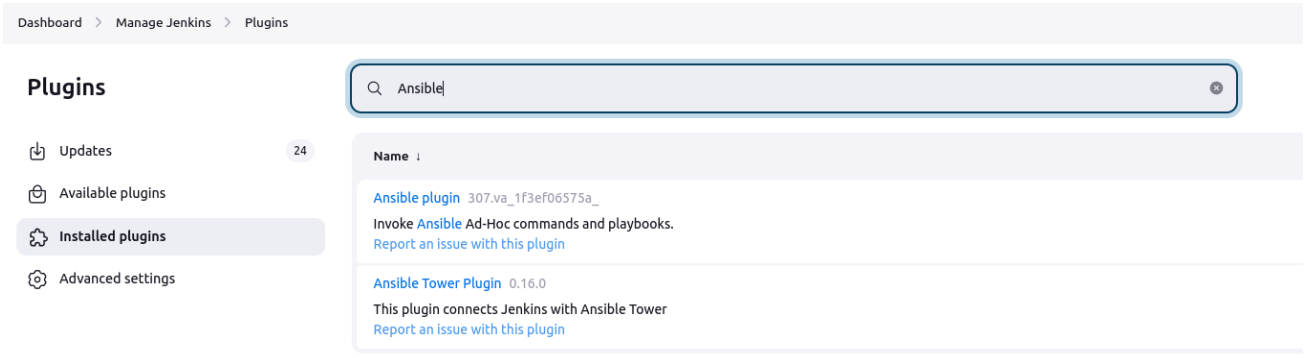
CONFIGURATION MANAGEMENT WITH ANSIBLE

Ansible is an open-source automation tool that enables configuration management, application deployment, and orchestration. It allows you to automate tasks such as software installation, configuration changes, and system updates across multiple servers or environments. Ansible uses simple YAML-based playbooks to define automation tasks, making it easy to understand and use.

Benefits of Ansible:

- **Consistency**: Ansible ensures consistency across environments by automating configuration tasks and enforcing desired states on target systems.
- **Scalability**: Ansible scales easily to manage configurations for large infrastructure deployments, enabling efficient management of multiple servers.
- **Reduced Manual Overhead:** By automating repetitive tasks, Ansible reduces manual overhead and frees up valuable time for IT teams to focus on more strategic initiatives.
- **Idempotent Operations**: Ansible playbooks are idempotent, meaning they can be run multiple times without causing unexpected changes, ensuring predictable and reliable configuration management.

Ansible Plugins



Ansible Playbook:

Ansible playbooks are YAML files that define a set of tasks to be executed on remote systems. Each task specifies the desired state of a system, such as installing packages, configuring services, or updating configuration files.

```yaml
---
- name: Deploying Review Radar on the kubernetes cluster
  hosts: localhost
  remote_user: anjali
  become: false
  tasks:
    - name: Deleting older application files
      ansible.builtin.shell: |
        kubectl delete -f ../k8s/deploy/backend-deploy.yml --ignore-not-found=true
        kubectl delete -f ../k8s/deploy/frontend-deploy.yml --ignore-not-found=true
        kubectl delete -f ../k8s/services/backend-service.yml --ignore-not-found=true
        kubectl delete -f ../k8s/services/frontend-service.yml --ignore-not-found=true
        kubectl delete -f ../k8s/ingress/ingress.yml --ignore-not-found=true

    - name: Deploying application to kubernetes cluster
      ansible.builtin.shell: |
        kubectl apply -f ../k8s/deploy/backend-deploy.yml
        kubectl apply -f ../k8s/deploy/frontend-deploy.yml
        kubectl apply -f ../k8s/services/backend-service.yml
        kubectl apply -f ../k8s/services/frontend-service.yml
        kubectl apply -f ../k8s/ingress/ingress.yml
```

This is an Ansible playbook named "Deploying Review Radar on the Kubernetes cluster". Ansible is being used to automate the deployment process of the Review Radar application onto a Kubernetes cluster.

name: Describes the name of the playbook.

hosts: Specifies the target hosts or machines where the tasks will be executed. Here, it's set to "localhost", indicating that the tasks will be executed on the local machine.

remote_user: Specifies the user to use for SSH connections. Here, it's set to "anjali".

become: Indicates whether to escalate privileges (become root) to execute tasks. Here, it's set to "false", meaning privileges won't be escalated.

tasks: This section contains the list of tasks to be executed:

Deleting older application files: This task uses the kubectl delete command to delete older application files from the Kubernetes cluster. It includes deleting deployment files (backend-deploy.yml and frontend-deploy.yml), service files (backend-service.yml and frontend-service.yml), and the ingress file (ingress.yml). The --ignore-not-found=true option ensures that the command does not fail if the resources are not found.

Deploying application to Kubernetes cluster: This task uses the kubectl apply command to apply new application files to the Kubernetes cluster. It includes applying deployment files for backend and

frontend (backend-deploy.yml and frontend-deploy.yml), service files for backend and frontend (backend-service.yml and frontend-service.yml), and the ingress file (ingress.yml).

Overall, this playbook automates the process of deploying the Review Radar application to a Kubernetes cluster by deleting old resources and applying new ones using kubectl commands.

Inventory file

In Ansible, an inventory file is a crucial component that defines the hosts or groups of hosts on which Ansible commands, modules, and playbooks will operate. The inventory file can be a simple text file listing IP addresses or hostnames, organized into groups for easier management. It allows users to specify variables and configuration settings for each host or group, enabling efficient and targeted automation.

```
ansible_host=localhost ansible_user=anjali ansible_ssh_pass=12345
ansible_ssh_common_args = '-o StrictHostKeyChecking=no'
```

Here's a breakdown of each part:

ansible_host=localhost: Specifies the address of the host to connect to. In this case, localhost indicates the local machine.

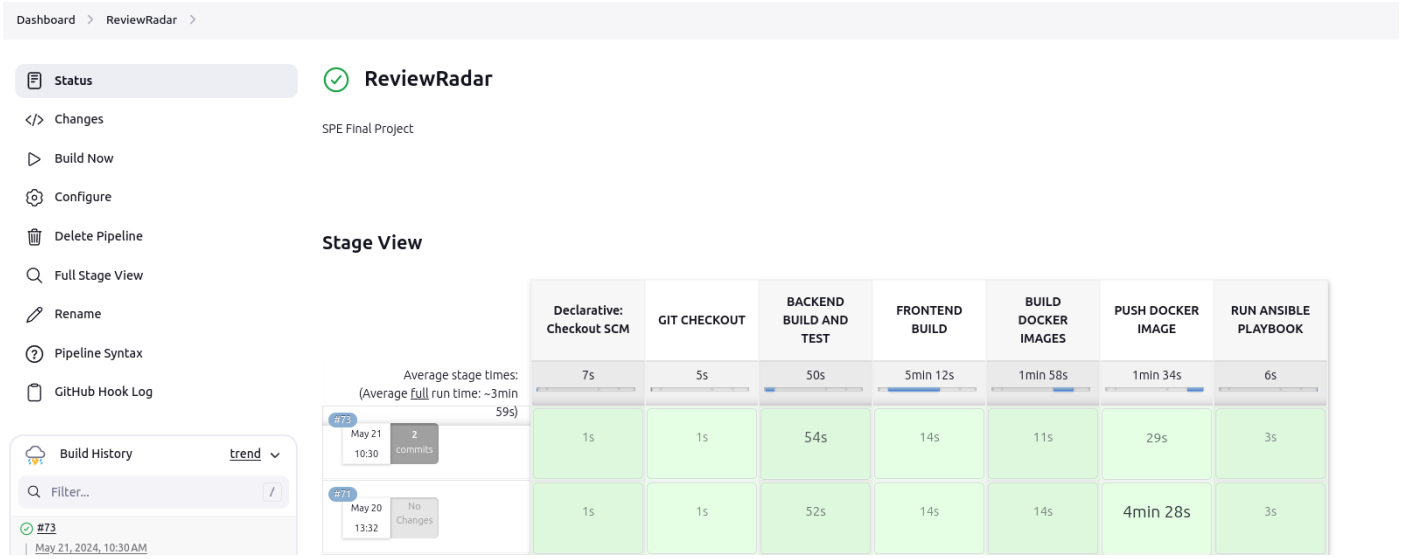
ansible_user=anjali: Indicates the username to use when connecting to the host. Here, the user is anjali.

ansible_ssh_pass=12345: Provides the SSH password for the user anjali. In this example, the password is 12345

ansible_ssh_common_args='-o StrictHostKeyChecking=no': Sets additional SSH arguments. The **-o StrictHostKeyChecking=no** option disables SSH's host key checking, preventing SSH from prompting for confirmation when connecting to a new host for the first time. This is useful for automation but should be used with caution as it can expose the system to man-in-the-middle attacks.

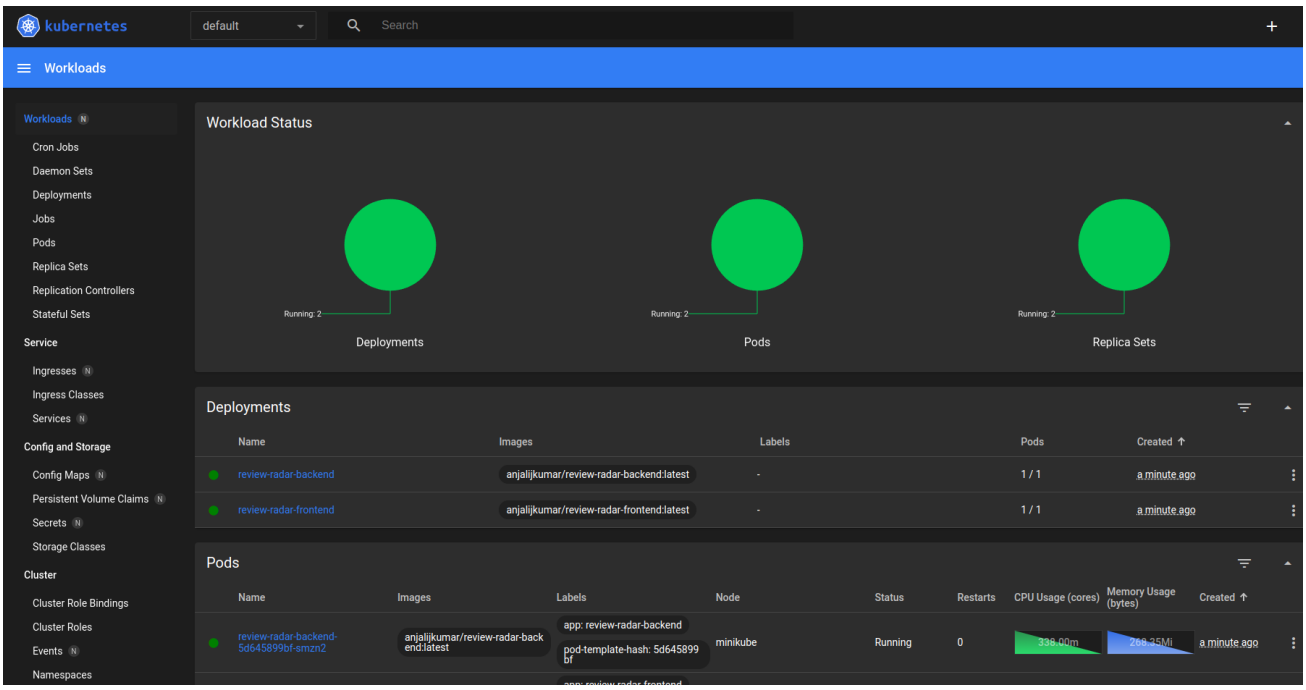
This configuration provides Ansible with the necessary details to connect to localhost using the specified user credentials and SSH options.

Jenkins Pipeline Output



This Jenkins pipeline provides an automated, consistent, and repeatable process for building, testing, containerizing, and deploying the Review Radar application. By breaking down the process into distinct stages, it ensures each step is executed in sequence, with clear checkpoints and validations along the way. This approach enhances efficiency, reduces the potential for human error, and facilitates continuous integration and continuous deployment (CI/CD) practices.

Kubernetes Dashboard



CONTINUOUS MONITORING WITH THE ELK STACK

The ELK Stack is a popular open-source log management platform that combines Elasticsearch, Logstash, and Kibana. It provides a comprehensive solution for collecting, indexing, searching, analyzing, and visualizing log data in real-time. The ELK Stack is widely used for monitoring applications, infrastructure, and system logs, enabling organizations to gain insights into their systems' health, performance, and security.

Components of the ELK Stack:

- **Elasticsearch**: Elasticsearch is a distributed search and analytics engine that stores and indexes log data efficiently. It enables fast and scalable searches across large volumes of log data, supporting real-time analysis and visualization.
- **Logstash**: Logstash is a data processing pipeline that ingests, processes, and enriches log data before indexing it into Elasticsearch. It provides input plugins to collect logs from various sources, filter plugins to parse and transform log data, and output plugins to send processed data to Elasticsearch.
- **Kibana**: Kibana is a web-based visualization and dashboarding tool that allows users to explore, analyze, and visualize log data stored in Elasticsearch. It provides interactive dashboards, charts, and graphs for monitoring and troubleshooting purposes, making it easy to identify trends, anomalies, and patterns in log data.

Continuous Monitoring Workflow:

- **Data Collection**: Log data is collected from various sources such as servers, applications, and network devices using Logstash input plugins. These plugins support protocols like syslog, filebeat, and beats for collecting logs in different formats.
- **Data Processing**: Logstash processes the collected log data by parsing, filtering, and enriching it as required. This includes extracting relevant fields, converting timestamps, and applying transformations to normalize the data.
- **Indexing:** Processed log data is indexed into Elasticsearch, where it is stored and made available for search and analysis. Elasticsearch indexes the data in a distributed and scalable manner, ensuring fast and efficient retrieval.
- **Visualization and Analysis**: Kibana is used to create visualizations and dashboards that provide insights into log data. Users can create custom dashboards with charts, graphs, and tables to monitor system performance, track error rates, and detect security incidents in real-time.

Key Features of the ELK Stack:

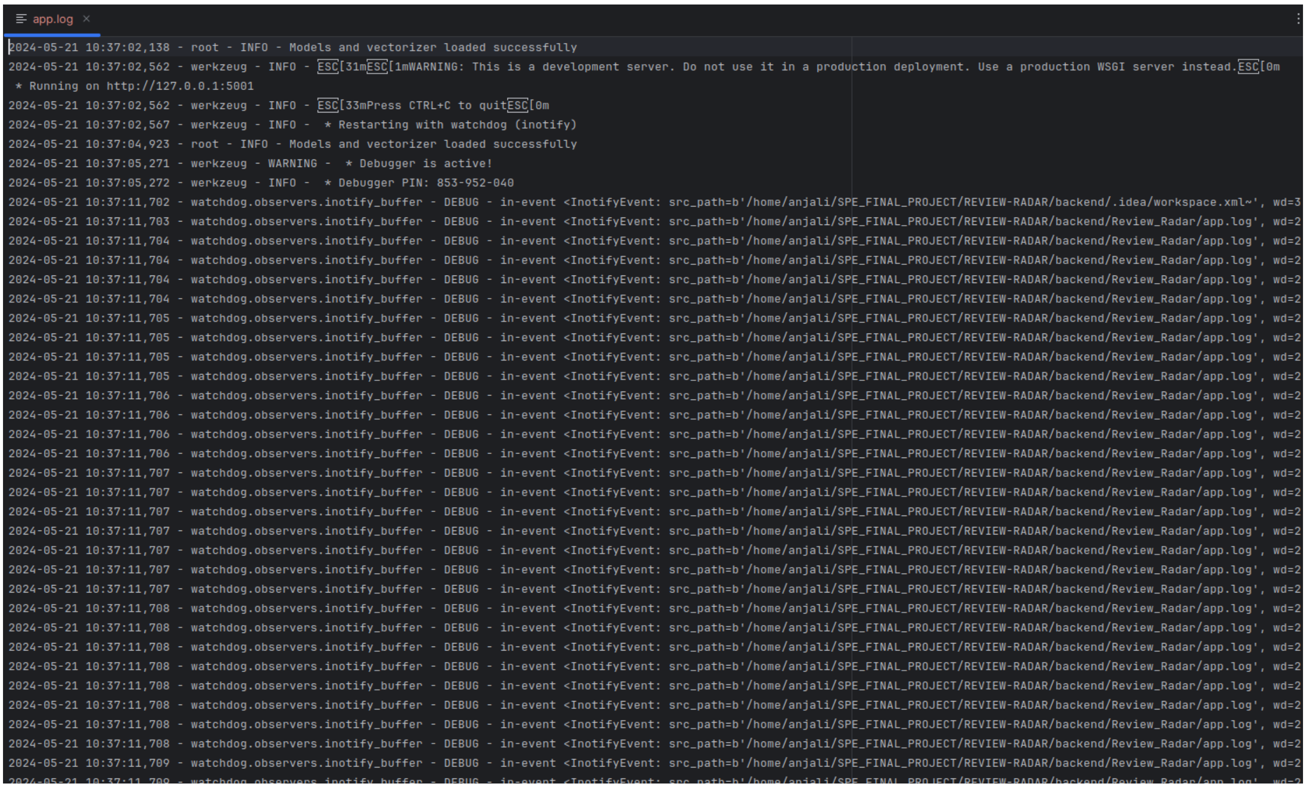
- **Real-time Monitoring:** The ELK Stack supports real-time monitoring of log data, allowing organizations to detect and respond to issues as they occur.
- **Scalability**: Elasticsearch and the ELK Stack can scale horizontally to handle large volumes of log data, ensuring performance and reliability even in high-traffic environments.

- **Alerting**: Kibana provides alerting capabilities to notify users of important events or anomalies in log data. Alerts can be configured based on predefined conditions and thresholds.
- **Search and Analysis**: Elasticsearch offers powerful search and analytics capabilities, including full-text search, aggregations, and filtering, enabling users to analyze log data in-depth and extract valuable insights.

Deployment Considerations:

- **Infrastructure Requirements**: Deploying the ELK Stack requires sufficient hardware resources, especially for Elasticsearch nodes, to ensure optimal performance and scalability.
- **Security**: Proper security measures should be implemented to protect the ELK Stack from unauthorized access and data breaches. This includes securing network communication, enabling authentication and authorization mechanisms, and encrypting sensitive data.
- **Maintenance and Monitoring**: Ongoing maintenance and monitoring are essential for the smooth operation of the ELK Stack. Regular updates, backups, and performance monitoring help ensure the reliability and availability of the monitoring platform.

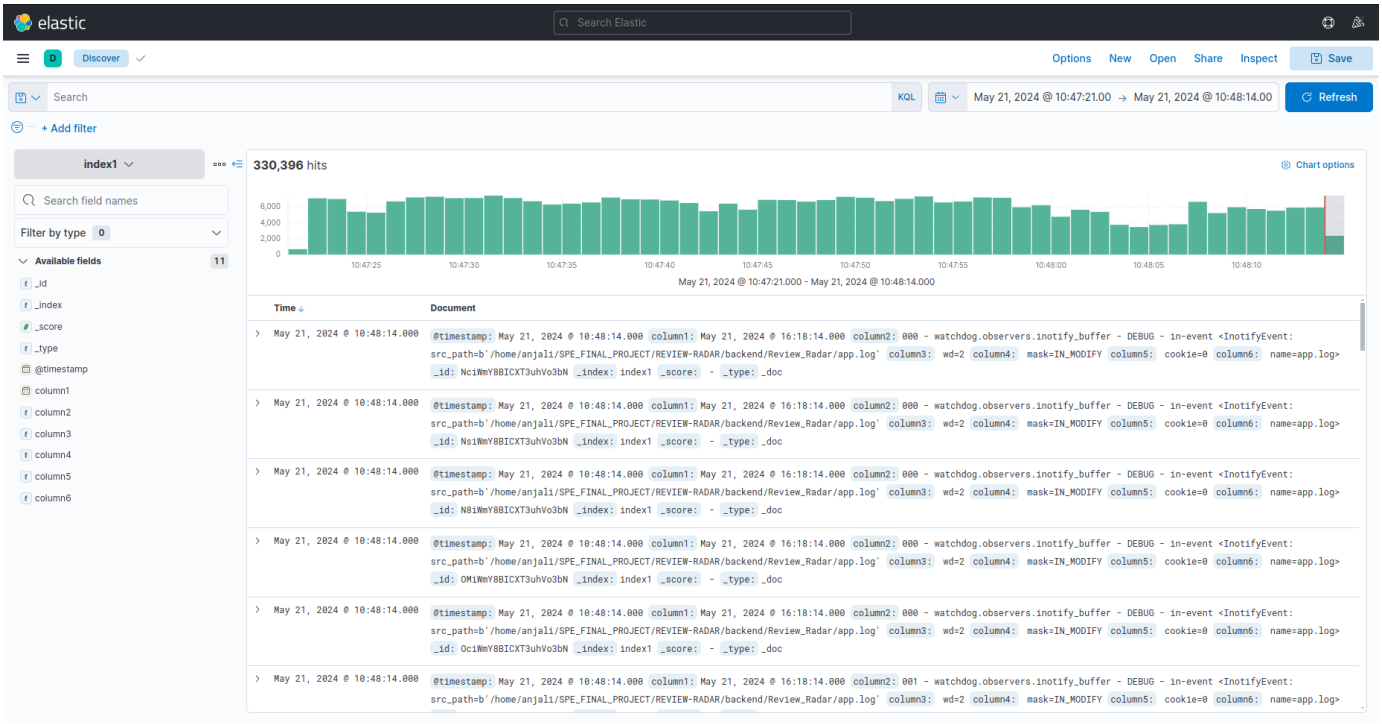
Log File



```
app.log
2024-05-21 10:37:02,138 - root - INFO - Models and vectorizer loaded successfully
2024-05-21 10:37:02,562 - werkzeug - INFO - ESC[31mESC[1mWARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.ESC[0m
 * Running on http://127.0.0.1:5001
2024-05-21 10:37:02,562 - werkzeug - INFO - ESC[33mPress CTRL+C to quitESC[0m
2024-05-21 10:37:02,567 - werkzeug - INFO -  * Restarting with watchdog (inotify)
2024-05-21 10:37:04,923 - root - INFO - Models and vectorizer loaded successfully
2024-05-21 10:37:05,271 - werkzeug - WARNING -  * Debugger is active!
2024-05-21 10:37:05,272 - werkzeug - INFO -  * Debugger PIN: 853-952-040
2024-05-21 10:37:11,702 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/.idea/workspace.xml~', wd=3
2024-05-21 10:37:11,703 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,704 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,704 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,704 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,704 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,705 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,705 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,705 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,705 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,706 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,706 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,706 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,706 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,707 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,707 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,707 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,707 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,707 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,707 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,707 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,708 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,708 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,708 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,708 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,708 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,708 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,708 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,708 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
2024-05-21 10:37:11,709 - watchdog.observers.inotify_buffer - DEBUG - in-event <InotifyEvent: src_path=b'/home/anjali/SPE_FINAL_PROJECT/REVIEW-RADAR/backend/Review_Radar/app.log', wd=2
```

Kibana Dashboard

Access Kibana Dashboard: http://localhost:5601



CONCLUSION

In conclusion, the implementation of various technologies and methodologies such as topic modeling, sentiment analysis, Docker containerization, Ansible configuration management, Jenkins continuous integration, and the ELK Stack for continuous monitoring has significantly enhanced our ability to analyze customer sentiments, streamline development workflows, and ensure the reliability and performance of our applications.

Through topic modeling and sentiment analysis, we have gained valuable insights into customer feedback, enabling us to identify key areas for improvement and prioritize actionable tasks. By leveraging Docker containerization, we have achieved consistency and portability in our deployment process, allowing us to deploy applications efficiently across different environments.

Ansible configuration management has enabled us to automate repetitive tasks and enforce desired states across our infrastructure, improving consistency and reducing manual overhead. Jenkins continuous integration has facilitated the automation of build, test, and deployment processes, enabling us to deliver high-quality software faster and more reliably.

Finally, the ELK Stack for continuous monitoring has provided us with real-time visibility into our system's health, performance, and security posture. By collecting, analyzing, and visualizing log data, we can proactively detect and respond to issues, ensuring the reliability and availability of our applications.

In summary, the adoption of these technologies and methodologies has empowered us to better understand customer sentiments, accelerate development cycles, and maintain the stability and security of our applications in an ever-evolving technological landscape. As we continue to innovate and improve, these tools and practices will remain invaluable assets in our pursuit of excellence.