**Genie Application**

**Overview**

**Genie** is a powerful code review tool designed to automate code reviews, provide code analysis, and ensure that the code meets quality standards. Genie also offers seamless integration with CI/CD pipelines and cloud environments, enabling developers to improve the quality and security of their codebases.

This document provides detailed instructions on how to set up, configure, and use **Genie**.

**Prerequisites**

Before getting started, ensure that you have the following tools installed:

**1. Docker Desktop**

* Install Docker Desktop to run Genie and its associated services in containers.
* Docker allows for containerization of applications and simplifies deployment and scaling.

**2. Python**

* Genie relies on Python for various backend operations, including code analysis and processing.
* Make sure Python is installed, preferably version 3.9 or higher.

**3. pip (Python Package Installer)**

* pip is required to install Python dependencies for Genie.
* Ensure that pip is installed along with Python.
* Install pip via the command:

python -m ensurepip --upgrade

**4. Nginx**

* Nginx will be used as a reverse proxy and load balancer for Genie.
* Install Nginx to handle web traffic routing to Genie.

**5. Node.js**

* Genie’s frontend relies on Node.js for its build and package management system.
* Ensure that Node.js (LTS version) is installed.

**6. Nexus Artifactory**

* Nexus Artifactory is required if you're using a repository manager for storing your artifacts and dependencies.
* If you plan to use Genie to manage internal libraries or dependencies, make sure you have access to Nexus or Artifactory.

**7. GKE (Google Kubernetes Engine) Cluster**

* If deploying Genie to the cloud, you will need a GKE cluster to host the application.
* Set up a GKE cluster and configure Kubernetes for Genie deployment.
* Google Cloud Kubernetes Engine Guide

**8. Cloud Managed DNS**

* Set up cloud-managed DNS to ensure that the domain for Genie resolves properly in your cloud environment.
* Google Cloud DNS or AWS Route 53 are popular choices.
* Google Cloud DNS Guide

**9. Venafi for Certificate Management**

* Venafi is used to manage and issue certificates for Genie to ensure secure HTTPS communication.
* Configure Venafi to create and manage SSL/TLS certificates.

**10. Google Cloud Artifact Registry**

* **Google Cloud Artifact Registry** is required if you plan to store container images, Helm charts, or other build artifacts used by Review Genie.
* Artifact Registry helps you securely store and manage artifacts in a centralized location.

**Features of Genie**

Genie comes packed with features to streamline the code review process and improve code quality:

* **Automated Code Review**: Automatically analyzes the code and provides suggestions for improvement based on predefined rules.
* **Security Checks**: Identifies potential security vulnerabilities in the codebase.
* **Performance Insights**: Highlights areas of the code that could be optimized for better performance.
* **Integration with Git Repositories**: Seamlessly integrates with popular Git repositories like GitHub, GitLab, and Bitbucket.
* **Code Explanation**: Generates explanations of complex code snippets to aid understanding.
* **Support for Multiple Languages**: Supports various programming languages such as Python, JavaScript, Java, and more.
* **Continuous Integration (CI) Integration**: Easily integrates with Jenkins, CircleCI, GitHub Actions, etc., to automate the review process.
* **Customizable Rules**: Genie comes with default rules, but you can create custom rules tailored to your organization’s coding standards.

**Installation and Setup**

**1. Clone the Genie Repository**

* Start by cloning the Genie repository to your local machine

git clone https://github.com/alm.github.hsbc/wlb-nldp-repo1.git

cd wlb-nldp-repo1

**2. Docker Setup**

1. **Build the Docker Image**: Build the Docker image for the app.

docker build -t frontend .

1. **Run the Docker Container**: Once the image is built, run the container

docker run -d -p 8080:80 Genie

1. **Verify the Setup**: Access Genie on http://localhost:8080 to ensure it's running properly.

**3. Configure Nginx**

1. **Install Nginx**: If not already installed, follow the instructions to install Nginx on your server.
2. **Nginx Configuration**: Set up Nginx as a reverse proxy to forward requests to Genie.

Sample Nginx config:

nginx

Copy

server {

listen 80;

server\_name genie.com;

location / {

proxy\_pass http://localhost:8080;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

}

}

1. **Restart Nginx**: After updating the Nginx config, restart the service:

bash

Copy

sudo systemctl restart nginx

**Backend - FastAPI Integration**

Review Genie uses **FastAPI** as the backend framework to handle all the API requests and data processing. The FastAPI application is responsible for processing requests related to code analysis, security checks, and interacting with the frontend to display results.

**FastAPI Backend Overview**

* The backend is built using **FastAPI**, which is a modern, fast (high-performance), web framework for building APIs with Python.
* The **FastAPI** application runs on port **3000**.
* It serves RESTful APIs that are consumed by the frontend.

**Backend Configuration**

1. **Backend Port**: The FastAPI backend listens on port 3000.
   * Example: http://localhost:3000 (if running locally)
2. **Backend Service**:
   * The FastAPI backend exposes endpoints that the frontend can call to trigger code reviews, fetch analysis results, and interact with other services.
   * The frontend (typically running on a different port) will make HTTP requests to the FastAPI backend to retrieve and submit data.

**Frontend and Backend Communication**

To establish the connection between the frontend and the FastAPI backend:

1. **Frontend Configuration**:
   * The frontend of Review Genie interacts with the backend through API calls. This can be done via **Axios** or **Fetch** API (in JavaScript).
   * The frontend sends HTTP requests to the backend API to initiate code reviews, fetch results, and handle other actions.
2. **API Requests**:
   * The frontend sends requests to the backend at the FastAPI server endpoint (e.g., http://localhost:3000/api/code-review).
   * The backend processes the requests and returns JSON data containing the results or status updates.

Example of an API request from the frontend to the backend:

**Deployment Considerations**

In a production environment, the backend might be deployed as a containerized service using Docker, Kubernetes, or a cloud service. When deploying, ensure the following:

* **Port Mapping**: Ensure that port **3000** is exposed for the FastAPI backend container or server.
* **CORS (Cross-Origin Resource Sharing)**: Since the frontend and backend are often served from different origins in development or production environments, **CORS** needs to be configured on the FastAPI backend to allow requests from the frontend.

This configuration will allow the frontend (running on http://localhost:8080 or any other specified domain) to interact with the backend running on **port 3000**.

**Configuration**

Genie allows you to customize various settings, such as:

* **Code Review Rules**: Configure default or custom rules for code analysis.
* **Integration Settings**: Set up integrations with GitHub, GitLab, or Bitbucket.
* **CI/CD Integration**: Configure Genie to trigger reviews in your CI pipeline.

Refer to the **configuration** file in the app repository for more detailed configuration options.

**Usage**

Once everything is set up, Genie will automatically review code when pushed to your Git repository or triggered via a CI/CD pipeline.

* **Submit a Code Review**: Push your code to your configured repository. Genie will automatically begin the review process.
* **View the Results**: After the review, you can view detailed reports on code quality, security, and performance.

**Genie Application Folder Structure**

The folder structure of the **Genie** application is organized to separate the frontend and backend code, Kubernetes deployment configurations, and other necessary components for smooth deployment and management.:

**Folder Structure Overview**

review-genie/

│

├── backend/ # Backend (FastAPI) code

│ ├── app/ # FastAPI application code

│ ├── Dockerfile # Dockerfile to containerize backend

│ ├── requirements.txt # Python dependencies for FastAPI

│ └── config/ # Configuration files

│

├── frontend/ # Frontend code (e.g., React, Vue.js, etc.)

│ ├── public/ # Public files like HTML templates

│ ├── src/ # Frontend source code (components, hooks, etc.)

│ ├── Dockerfile # Dockerfile to containerize frontend

│ ├── package.json # Node.js dependencies and scripts

│ └── .env # Environment variables for the frontend

│

├── k8s/ # Kubernetes configuration files

│ ├── backend-deployment.yaml # Kubernetes deployment for the backend

│ ├── backend-service.yaml # Kubernetes service for the backend

│ ├── frontend-deployment.yaml # Kubernetes deployment for the frontend

│ ├── frontend-service.yaml # Kubernetes service for the frontend

│ └── ingress.yaml # Optional Ingress configuration for routing

│

├── docker-compose.yml # Docker Compose for local multi-container setup

└── README.md # Project documentation and setup instructions

<Need to update the information accordingly >