Pipeline Environment Builder Tool

Reproducing Made Easy

Prepared by: Aymen Maqsood Mulbagal

1. Project Overview

This project aims to develop the Pipeline Environment Builder Tool that automates the process of scanning GitHub repositories, extracting and installing dependencies, and generating Docker containers. This will significantly benefit computational biology scientists by streamlining environment setup and ensuring reproducibility.

2. Core Features

- Repository Scanning: Extract dependencies from environment files (requirements.txt, environment.yml, package.json).
- Dependency Installation: Install dependencies using appropriate package managers (pip, conda, npm)
- Docker Container Generation: Create Dockerfiles based on the extracted dependencies.
- User Interface: Web interface for inputting GitHub repository URLs and displaying process status.
- Notifications: Inform users about the process status (success, failure, issues).

3. Technology Stack

- Frontend:
  - React.js: For building the user interface.
  - Bootstrap: For styling and responsive design.

- Backend:
  - Python (Flask/Django): For handling API requests and processing.
  - Docker SDK for Python: To interact with Docker and create containers programmatically.

- Database:
  - PostgreSQL/MySQL: To store user data, logs, and configuration files.

- Other Tools:
  - GitHub API: To fetch repository data.
  - Docker: For containerization.
  - CI/CD Tools: GitHub Actions for continuous integration and deployment.

4. Implementation Steps

Step 1: Setup Project Structure
- Initialize a new Git repository.
- Set up frontend and backend directories.

Step 2: Frontend Development
- Build a form for inputting GitHub repository URLs.
- Display the status of scanning and container creation processes.

Step 3: Backend Development
- API Endpoints:
  - /scan-repo: Accepts GitHub URL, clones the repository, and scans for dependencies.
  - /generate-docker: Generates a Dockerfile based on detected dependencies.
  - /status: Checks the status of the process.

Step 4: Dependency Extraction
- Develop scripts to parse common environment files (requirements.txt, environment.yml, package.json).
- Use package managers (pip, conda, npm) to install dependencies.

Step 5: Dockerfile Generation
- Create Dockerfile templates.
- Populate templates with extracted dependencies.
- Use Docker SDK to build and test the Docker image.

Step 6: Integration with GitHub API
- Fetch repository details using GitHub API.
- Handle authentication and rate limiting.

Step 7: Notifications and Logging
- Implement a logging mechanism to track the process.
- Send notifications to users via email or webhooks.

5. Testing and Deployment

Testing
- Unit tests for individual components.
- Integration tests for the entire workflow.
- User acceptance testing (UAT) with selected labs.

Deployment
- Containerize the application using Docker.

- Deploy on cloud platforms like AWS, GCP, or Azure.
- Set up CI/CD pipelines using GitHub Actions.

6. Documentation and Support

Documentation
- Create comprehensive documentation for users and developers.
- Include examples and troubleshooting guides.

Support
- Set up a support channel (e.g., Slack, Discord) for user queries and feedback.

7. Future Enhancements

- Support for More Package Managers: Extend support to other languages and package managers.
- Advanced Dependency Analysis: Integrate tools like Snyk for vulnerability scanning.
- Customizable Dockerfiles: Allow users to customize Dockerfile templates.