Trunk-Based Development Score Application Documentation

Table of Contents

[Introduction 2](#_Toc174879791)

[Architecture 4](#_Toc174879792)

[High-Level Overview 4](#_Toc174879793)

[Architecture Diagram 4](#_Toc174879794)

[Frontend 5](#_Toc174879795)

[Technology Stack 5](#_Toc174879796)

[Functionality 5](#_Toc174879797)

[Components 5](#_Toc174879798)

[API Integration 5](#_Toc174879799)

[Error Handling 5](#_Toc174879800)

[Backend Service 6](#_Toc174879801)

[Technology Stack 6](#_Toc174879802)

[Functionality 6](#_Toc174879803)

[API Endpoints 6](#_Toc174879804)

[Bitbucket API Integration 6](#_Toc174879805)

[Score Calculation <INCLUDE THIS ONLY IF WE ARE PROVIDING SCORE> 6](#_Toc174879806)

[Error Handling 7](#_Toc174879807)

[Environment & Deployment 8](#_Toc174879808)

[OpenShift Overview 8](#_Toc174879809)

[Deployment Pipeline 8](#_Toc174879810)

[Configuration 8](#_Toc174879811)

# Introduction

This application evaluates a project’s adherence to trunk-based development practices by analyzing its commit history from Bitbucket. It consists of two main components:

* **Frontend Service**: A ReactJS-based application that allows users to input a Git repository and view the score.
* **Backend Service**: A Python-based REST API that fetches commit history from Bitbucket and calculates a score based on trunk-based development metrics.

Both services are deployed on OpenShift, ensuring scalability and ease of deployment.

# Architecture

## **High-Level Overview**

The system consists of two primary components:

1. **Frontend Service (ReactJS)**: Accepts a Git repository as input, sends the request to the backend, and displays the adherence score.
2. **Backend Service (Python REST API)**: Fetches commit history from Bitbucket using Bitbucket’s API and calculates a score based on adherence to trunk-based development principles.

## **Architecture Diagram**

# Frontend

## **Technology Stack**

* **ReactJS**: Frontend framework.
* **Axios or Fetch API**: For making HTTP requests to the backend service.
* **OpenShift**: For deployment.

## **Functionality**

The frontend service provides the user interface to interact with the system:

* **User Input**: Accepts the Git repository URL for the project under evaluation.
* **API Interaction**: Sends the repository details to the backend for score calculation.
* **Score Display**: Receives the calculated score from the backend and displays it to the user.

## **Components**

* **Repository Input Form**: A form where the user inputs the Git repository URL.
* **Submit Button**: Triggers the API request to the backend.
* **Score Display Component**: Displays the calculated score once received.

## **API Integration**

* **HTTP Request**: The frontend uses Axios (or Fetch API) to make a POST request to the backend service. The request body contains the Git repository URL.

## **Error Handling**

If the backend service fails to calculate the score, the frontend displays an error message to the user. This can happen due to invalid repository URLs or issues with the Bitbucket API.

# Backend Service

## **Technology Stack**

* **Python**: Programming language used to build the service.
* **Flask or FastAPI**: REST framework for handling API requests.
* **Bitbucket API**: Used to fetch commit history.
* **OpenShift**: For deployment.

## **Functionality**

The backend service is responsible for:

* Fetching the commit history from Bitbucket using Bitbucket’s API.
* Calculating a score based on whether the commit history adheres to trunk-based development practices.
* Returning the score to the frontend service.

## **API Endpoints**

* **Endpoint**: <ADD ENDPOINT HERE>
  + **Method**: POST
  + **Request Body**: <ADD IF ANY>
  + **Response Body**: <ADD RESPONSE JSON HERE>
  + **Description**: This endpoint receives a repository URL, fetches the commit history from Bitbucket, and returns a trunk-based development adherence score.

## **Bitbucket API Integration**

The backend uses Bitbucket’s API to fetch the commit history for a given repository.

Example: <ADD CODE SNIPPET HERE>

### **Score Calculation <INCLUDE THIS ONLY IF WE ARE PROVIDING SCORE>**

The score is calculated based on trunk-based development metrics such as:

* Frequency of merges into the main branch.
* Small, frequent commits.
* Minimal feature branching.

Example: <ADD CODE SNIPPET>

## **Error Handling**

The backend handles various errors, including:

* Invalid repository URLs.
* Issues with fetching data from Bitbucket.
* Returns appropriate HTTP status codes and error messages to the frontend.

# Environment & Deployment

## **OpenShift Overview**

OpenShift is used for deploying both the frontend and backend services. The containerized deployment ensures scalability and robustness.

## **Deployment Pipeline**

* **CI/CD Pipeline**: The deployment pipeline is automated using OpenShift's CI/CD capabilities.
  + **Build**: Build the ReactJS frontend and Python backend.
  + **Deploy**: Push the containers to OpenShift and deploy them as pods.

## **Configuration**

* **Dockerfile**: Both services are containerized using Docker.
* **Kubernetes Config**: YAML configuration files define the deployment, services, and pods in OpenShift.