CI/CD Report

Introduction

This document provides an overview of the Jenkins pipeline implemented for the MOP (Melbourne Open Playground) project. The pipeline automates various stages of the application development lifecycle, from code checkout to deployment. It leverages a set of powerful tools and technologies to ensure efficient and reliable CI/CD processes.

Pipeline Workflow

The pipeline is structured into several stages, each responsible for a specific part of the build and deployment process. Below is a detailed explanation of each stage:

1. Checkout

- **Description**: The pipeline begins by checking out the latest code from the project's GitHub repository.
- **Technologies Used**: Git
- **Command**: git 'https://github.com/Chameleon-company/MOP-Code.git'
- **Purpose**: This stage ensures that the latest version of the source code is retrieved for further processing.

2. Setup

- **Description**: The setup stage verifies that the environment is correctly configured by checking the installed versions of Node.js and npm.
- **Technologies Used**: Node.js, npm
- **Commands**:
  - sh 'node -v'
  - sh 'npm -v'
- **Purpose**: This step confirms that the Node.js runtime and npm package manager are available and correctly installed, which are essential for building the Next.js application.

3. Install Dependencies

- **Description**: This stage installs all necessary dependencies for the Next.js application.
- **Technologies Used**: npm
- **Command**: sh 'npm install'
- **Purpose**: Installing dependencies ensures that all required packages are available for the application to build and run properly.

4. Build

- **Description**: The build stage compiles the Next.js application into an optimized production-ready version.
- **Technologies Used**: Next.js, npm

- **Command**: sh 'npm run build'
- **Purpose**: This stage compiles the application and prepares it for deployment by optimizing code and assets.

5. Code Quality Check

- **Description**: This stage checks the code quality using linting tools.
- **Technologies Used**: ESLint, npm
- **Command**: sh 'npm run lint'
- **Purpose**: Linting ensures that the code adheres to defined coding standards, helping to maintain code quality and consistency.

6. Test

- **Description**: This optional stage runs the application's test suite to ensure that the code is functioning as expected.
- **Technologies Used**: Testing frameworks (e.g., Jest), npm
- **Command**: sh 'npm run test'
- **Purpose**: Running tests helps catch bugs early in the development process, ensuring that changes do not break existing functionality.

7. Deploy

- **Description**: The deployment stage is responsible for containerizing the application using Docker and preparing it for deployment.
- **Technologies Used**: Docker
- **Commands**:
  - sh 'docker build -t mop-react-app .'
  - (Optional) sh 'docker push your-docker-repo/mop-react-app:latest'
  - (Optional) sh 'kubectl apply -f deployment.yaml'
- **Purpose**: This stage creates a Docker image of the application, which can then be deployed to various environments (e.g., staging, production).

8. Post Actions

- **Success**: Displays a success message when the pipeline completes without errors.
- **Failure**: Displays a failure message and encourages checking the logs for more information.
- **Cleanup**: Cleans the workspace to ensure that no residual files affect subsequent pipeline runs.
- **Technologies Used**: Jenkins pipeline syntax
- **Commands**:
  - echo 'Pipeline succeeded!'
  - echo 'Pipeline failed. Please check the logs for more details.'

- cleanWs()

9. Technologies Used

The following technologies are employed in the pipeline:

- **Jenkins**: The CI/CD tool orchestrating the entire pipeline.
- **Git**: Version control system used for source code management.
- **Node.js**: JavaScript runtime environment used to run the application.
- **npm**: Node.js package manager used to install dependencies and run scripts.
- **Next.js**: Framework used to build the application.
- **ESLint**: Tool for identifying and reporting on patterns found in ECMAScript/JavaScript code.
- **Docker**: Containerization platform used to create, deploy, and run applications.
- **Kubernetes (Optional)**: Container orchestration system used to manage deployment.

Conclusion

This pipeline automates the entire process of building, testing, and deploying the Next.js application, significantly improving efficiency and reducing the likelihood of human error. By using a combination of powerful tools like Jenkins, Docker, and Next.js, the pipeline ensures that the application can be reliably built and deployed across different environments.

How to Setup a Pipeline:

To set up a Jenkins pipeline, we need to follow several key steps. Here's a detailed guide on how to set up each component of the pipeline, including prerequisites, environment setup, and configurations.

1. Prerequisites

- Jenkins Installation: Ensure Jenkins is installed on the server. It can be downloaded from the official Jenkins website.
- Required Plugins: Install necessary Jenkins plugins such as Git, Pipeline, Docker, NodeJS, and any other plugins relevant to the setup.
- Docker: Install Docker on the Jenkins host to build and push Docker images.
- Node.js and npm: Make sure Node.js and npm are installed on the Jenkins server for building the Next.js application. We can use the NodeJS plugin to manage Node.js installations.
- Google Cloud SDK: Install and configure the Google Cloud SDK on the Jenkins server for deployments to Google Cloud Platform.

2. Jenkins Configuration

- Configure System: Go to Jenkins > Manage Jenkins > Configure System.
    1. NodeJS Plugin: Configure Node.js installations specifying the version needed.
    2. Environment Variables: Define environment variables such as GCP_SERVICE_ACCOUNT_KEY, ensuring sensitive data is stored securely using credentials.

- Credentials: Store credentials like GitHub access tokens and Google Cloud service account keys securely in Jenkins.

    1. Go to Jenkins > Credentials > System > Global credentials > Add Credentials.

    2. Add credentials for the GitHub repository and Google Cloud.

3. Create Pipeline Job

- Create a New Item: In Jenkins, select New Item, name the pipeline, and choose Pipeline as the type.

- Pipeline Configuration:

    1. In the pipeline configuration, under Pipeline, choose Pipeline script from SCM to fetch the pipeline script from the Git repository.

    2. Set the SCM to Git and provide the repository URL and credentials.

4. Pipeline Script

- Jenkinsfile: Ensure the Jenkinsfile is in the root of the repository or specified in the SCM configuration.

- The Jenkinsfile should contain the stages as described in the initial query, with scripts for each task like linting, testing, building, and deploying.

5. Docker and Kubernetes

- Dockerfile: Make sure the Dockerfile is present in the next_webapp directory for building the Docker image.

- Kubernetes Configuration: If deploying to Kubernetes, ensure a Kubernetes cluster is set up and configured to receive deployments from Jenkins.

6. Testing and Security

- Cypress: Install and configure Cypress for running end-to-end tests.

- Trivy: Set up Trivy for scanning Docker images for vulnerabilities as part of the pipeline.

7. Monitoring and Logging

- Set Up Monitoring Tools: If using GCP, configure Google Cloud Monitoring and Logging or set up Prometheus and Grafana for monitoring application metrics.

8. Execution

- Run the Pipeline: Trigger the pipeline manually from Jenkins or set up webhooks in the GitHub repository for automatic triggering on commits.

9. Maintenance and Updates

Regularly update the Jenkins plugins, Docker images, and Node.js versions to keep up with new features and security updates.

By following these steps, a pipeline can be set up a robust CI/CD pipeline in Jenkins that automates the process of building, testing, and deploying a Next.js application. Each step ensures that the deployment process is efficient, secure, and scalable.