{EPITECH}

TRINITY - DEV OPS

BOOTSTRAP

TRINITY - DEV OPS

Hi folks! If you feel lost, do not worry, this bootstrap is here to help you get started.

Initializing the application

Clone your [TRIGRAM-BOOTSTRAP-701] repository locally.

In your local repository, create a simple application. For instance, a Node.js application with an `app.js` file and a `package.json` file.

Push this code to your GitLab repository.

Congratulations!
You now have a GitLab repository containing the source code of your application.

Dockerizing the application

In your repository, create two Dockerfile files:

- ✓ `Dockerfile.dev`:
  configured for development (e.g., includes watch mode) ;
- ✓ `Dockerfile.prod`:
  configured for production (e.g., optimized for performance).

Then, update the `.gitignore` file to exclude any unnecessary files.

Finally, test building the Docker images locally.

Congratulations!
You now have Dockerfiles and commands to build and run containers in various environments.

Configuring the pipeline

At the root of your project, create a `.gitlab-ci.yml` file.

1. **Define Jobs**:
   Add jobs for each pipeline step:
   - ✓ A job to install dependencies and run tests (add a test file if needed).
   - ✓ A job to build Docker images using Dockerfiles.
   - ✓ A job to deploy the application in a target environment (for this initial setup, you can simulate deployment with a simple message).
2. **Set Conditions**:
   Specify conditions for each job. For example, run tests on every commit to the dev branch, but only build on the main branch.

Congratulations!
You now have a basic `.gitlab-ci.yml` pipeline with configured jobs and trigger conditions.

Installing and configuring the runners

On a virtual machine or dedicated server, install a GitLab Runner.

1. **Connect the Runner**:
   Link the runner to your GitLab repository.
2. **Configure the Runner**:
   Ensure the runner is available for pipeline jobs, specifying tags if needed.
3. **Run a Test**:
   Perform a test to ensure that the runner is functioning correctly and is properly configured.

Congratulations!
You now have a functional GitLab runner linked to your project.

Validating and improving the pipeline

Let's ensure your pipeline is working properly, and make some improvements if necessary.

1. **Push Code**:
   Begin by pushing some code to your repository to trigger the pipeline.
   Verify that it runs as expected.
2. **Check Logs**:
   Review the logs of each job to confirm that each step executes correctly.
3. **Fix Errors**:
   If any errors occur, resolve them and adjust the .gitlab-ci.yml configuration or Dockerfiles as necessary.
4. **Add Notifications** (Optional):
   If you feel comfortable, add a notification step to alert you if the pipeline fails.
   For instance, you could integrate with Slack, or set up some email alerts.

Congratulations!
You now have a GitLab working pipeline, with automated tests, build and deployment steps.

v 0.1.1