Update README.md
macta authored 2 minutes ago

Name	Last commit	Last update
scripts	Check for undetected errors on test run	3 months ago
src	updated baseline	3 months ago
.gitignore	Fix merge error	3 months ago
.gitlab-ci.yml	Use standard pharo docker image for builds	3 months ago
.project	Sample application	3 months ago
.properties	Fix merge error	3 months ago
README.md	Update README.md	2 minutes ago

README.md

Pharo GitLab Example

Introduction

This is a simple Pharo Smalltalk pipeline example, demonstrating how easy it is to get up and running with Pharo development using GitLab, and additionally how to build and publish a Pharo web application in a Docker container that can be easily deployed to a 3rd party service (e.g. Render.com).

If you create a new Pharo project in GitLab (using this template), you can load and customize it in Pharo with everything setup to commit, build, test and deploy an application.

To do this, either use the Iceberg Repository browser or copy and Evalute the following in a Pharo playground (replacing and SampleProject with your details):

```
Metacello new
  repository: 'gitlab://<user>/SampleProject:main';
  baseline: 'SampleProject';
  load.
```

Reference links

If you're new to Pharo and Gitlab the following starter documentation will help:

- GitLab CI Documentation
- Pharo Documentation

But if you're already a seasoned developer then this should look familiar.

What's contained in this project

The src directory contains standard tonel files, used to store Pharo projects, and the `.project` file is already configured to look for a baseline in this folder.

In addition to tonel content, there is a ready-to-go `.gitignore` file which helps keep your repository clean of build files and other configuration.

The `build/test/deploy` scripts contain basic templates to automatically run Pharo headless in a Docker image to build, test and deploy in a GitLab CI. These are all called from similarly named pipeline jobs specified in the standard .gitlab-ci.yml file.

The get a successful build to deploy on Render.com you will also need to define some additional variables in the CI/CD settings panel:

- `$DEPLOYED_TRIGGER_URL` - this is the private end-point URL from your render.com dashboard, which if defined will cause render to retrieve your docker image from your project docker container registry (use instructions on render for configuring your account to deploy a Docker image).

Pipeline Waltkhrough

If we take a look at the `.gitlab-ci.yml`, section by section.

First, we note that we see a supported Pharo Docker image used to build the project.

```
image: ghcr.io/ba-st/pharo:latest
```

We're defining three stages: `build`, `test` and `deploy`. If your project grows in complexity you can add more or modify these.

```
stages:
    - build
    - test
    - deploy
```

Next, we define a build job which simply runs a `build.sh` command and identifies the `build` and `deploy` folders as the output. Anything in these folders will automatically be handed off to future pipeline stages, and is also available to download through the web UI for diagnosing issues. Note the cache key setting to speed up builds by caching metacello package files. The cache can potentially give invalide results and may need the prefix changing in extreme circumstances.

```
build:
  stage: build
  cache:
    key: "${CI_COMMIT_REF_SLUG}_${CACHE_KEY_SUFFIX}" # or see: https://docs.gitlab.com/ee/ci/caching/
    paths:
      - $PHARO_CACHE_DIR
  script:
    - source ./scripts/build.sh
  artifacts:
    paths:
      - build   # included to diagnose any issues
      - deploy
```

Similar to the build step, we get test output by simply running `test.sh`. Noticde the `GIT_STRATEGY` is set to none to avoid an additional src checkout, as we can just use the artifacts already pushed from the previous build.

```
test:
  stage: test
  variables:
    GIT_STRATEGY: none # stop checkout, as using assets in /build
  script:
    - source ./build/test.sh
  artifacts:
    paths:
      - build/${PROJECT_NAME}*.xml
    reports:
      junit: build/${PROJECT_NAME}*.xml
```

This should be enough to get you started. There are many, many powerful options for your `.gitlab-ci.yml`. You can read about them in the documentation.