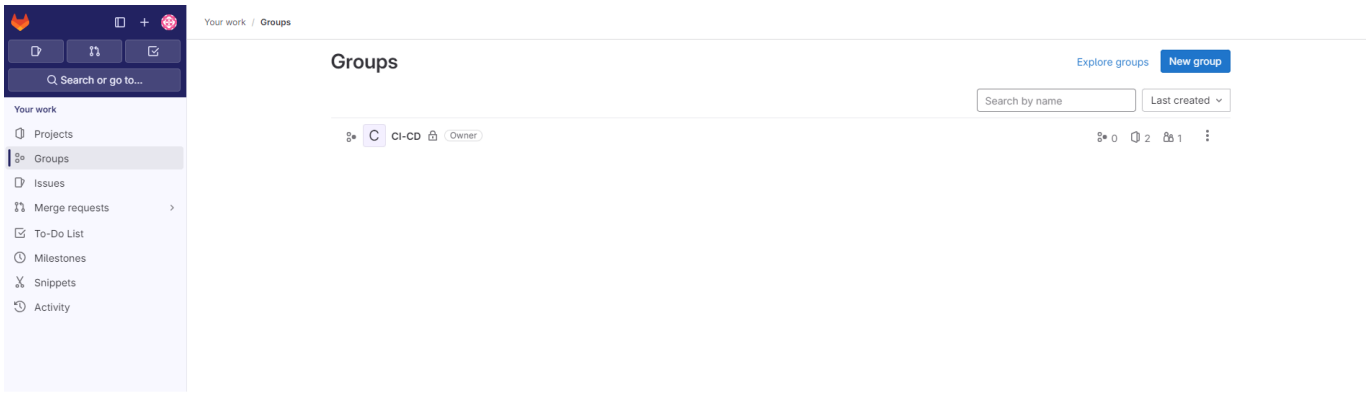
CI/CD Basislehrgang

Continuous Integration (CI) und **Continuous Delivery (CD)** sind Praktiken in der Softwareentwicklung, die darauf abzielen, Softwareänderungen häufiger und zuverlässiger zu veröffentlichen. CI ist der Prozess, bei dem Entwickler ihre Änderungen in einen Hauptzweig integrieren, wobei jeder Check-in automatisch gebaut und getestet wird. Dies hilft dabei, Integrationsprobleme frühzeitig zu erkennen. CD erweitert CI, indem es sicherstellt, dass nach dem Passieren der Tests die Software in einer standardisierten Weise in Produktionsumgebungen ausgerollt werden kann. Die Kombination von CI und CD ermöglicht es Teams, Softwareentwicklungszyklen zu beschleunigen und die Qualität der Releases zu verbessern, indem manuelle Fehlerquellen und Verzögerungen reduziert werden.

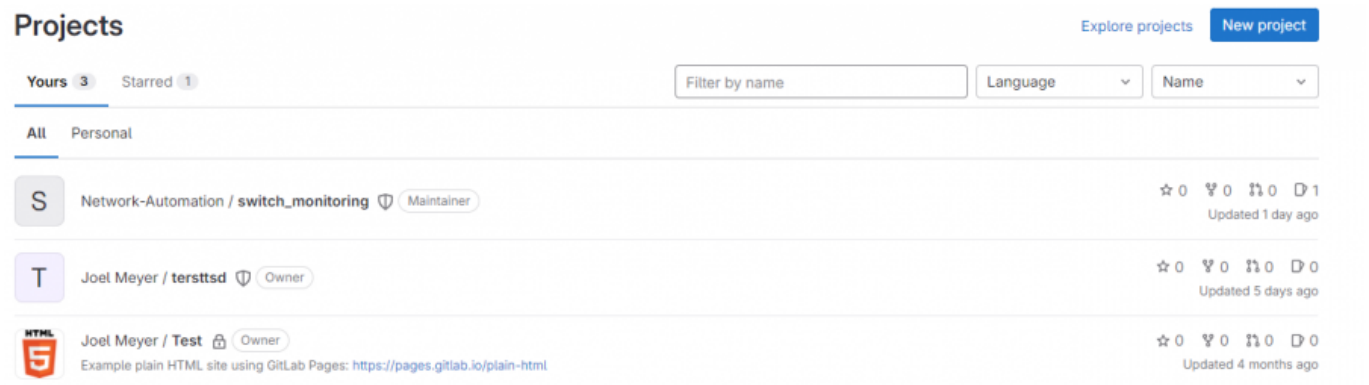
Die GitLab-Umgebung

Die GitLab-Test-Plattform der Rafisa findest du unter der URL https://gitlab.rafisa.test.

Um ein neues Projekt zu starten, erstelle zuerst eine Gruppe, da dein Name Grossbuchstaben im Namespace enthält, und dies mit Docker nicht funktioniert.



Erstelle das Projekt auch unter der Gruppe! Klicke dazu auf '**New Project**' und wähle '**Create blank project**' aus. Vergib anschliessend dem Projekt einen passenden Namen. Mit einem Klick auf '**Create project**' gelangst du zur Arbeitsfläche deines Projekts.



Projekt Hello World

In diesem Lehrgang wirst du eine Einführung in die Grundlagen von Continuous Integration (CI) und Continuous Delivery (CD) erhalten. Wir werden gemeinsam durch die Schritte gehen, um eine einfache Webanwendung zu erstellen und sie mithilfe von CI/CD automatisch zu integrieren und bereitzustellen. Du wirst lernen, wie man Docker-Container verwendet, um Anwendungen zu containerisieren, und wie man eine Pipeline erstellt, um automatisierte Tests durchzuführen und die Anwendung in einer Produktionsumgebung zu deployen. Am Ende des Lehrgangs wirst du in der Lage sein, deine eigene CI/CD-Pipeline zu erstellen und deine Softwareänderungen effizienter und zuverlässiger zu veröffentlichen.

Nun erstellst du entweder lokal mit deinem Editor deiner Wahl oder direkt in der Web-Oberfläche eine Datei. Dazu klickst du auf das „+“ und dann auf „New File“.

Jetzt erstellst du eine neue Datei mit dem Namen **hello.py** und schreibst folgenden Code in den Editor:

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"
```

Anschliessend klickst du auf '**Commit Changes**' in der Web-Oberfläche oder führst ein '**git push**' und '**git commit**' für das Projekt durch.

Nun erstellst du eine zweite Datei nach demselben Prinzip und nennst sie nur Dockerfile ohne Endung. Durch die Dockerfile wird unser Projekt „dockerisiert“. Das bedeutet, dass wir es in ein Docker-Image umwandeln. Dieses Image enthält alles, was unsere Anwendung braucht, um zu laufen, und kann auf jedem System mit Docker installiert werden.

Bitte ersetze die Sterne durch einen Port deiner Wahl, den du dir merken solltest, da du ihn später benötigen wirst:

```
# syntax=docker/dockerfile:1
FROM python:3.11

# install app dependencies
RUN pip install flask

# install app
COPY hello.py /
```

```
# final configuration
ENV FLASK_APP=hello
EXPOSE 8000
CMD flask run --host 0.0.0.0 --port ****
```

Die Pipeline

Die Pipeline automatisiert den Prozess der Integration und Bereitstellung unserer Softwareänderungen. Sie beginnt damit, dass der Sourcecode von unserem Repository heruntergeladen wird. Anschliessend durchläuft der Code verschiedene Phasen wie das Kompilieren, Testen und Bauen des Docker-Images. Während dieser Phasen werden automatische Tests ausgeführt, um sicherzustellen, dass die Software korrekt funktioniert. Sobald alle Tests bestanden sind, wird das Docker-Image erstellt und in einem Container bereitgestellt.

Nun gehst du links zu der Zeile in deinem Projekt und suchst nach „Build“. Dort wählst du den „Pipeline-Editor“ aus.

Dort drückst du dann auf **'Configure Pipeline'** und schreibst dort diesen Code ein:

Bitte ersetze alle Port 8000 durch deinen Port in deiner Umgebung. Verwende nicht Port 8000, da dieser wahrscheinlich bereits in Verwendung ist.

```
# General project configuration
#------------------------------
image: docker:24-dind  # Using Docker-in-Docker for building processes

stages:
  - build
  - push
  - test
  - deploy

services:
  - name: "docker:dind"  # Docker-in-Docker service for builds
    alias: dockerservice
    command: ["--tls=false", "--insecure-registry=gitlab.rafisa.test:5050"]


variables:
    DOCKER_DRIVER: overlay2  # Setting the Docker driver
    DOCKER_TLS_CERTDIR: ""   # TLS configuration for Docker
    PROJECT_REGISTRY: "$CI_REGISTRY/$CI_PROJECT_PATH"  # GitLab Registry URL
```

```
# Building Jobs
#------------------------------
build-job:
  stage: build
  tags:
    - docker-build
  resource_group: pipeline_lock  # Avoiding concurrent builds
  script:
    - echo "$CI_REGISTRY_PASSWORD" | docker login -u "$CI_REGISTRY_USER" --
password-stdin $CI_REGISTRY
    - docker pull "$PROJECT_REGISTRY:latest" || true  # Attempting to pull
the latest image for caching
    - |
      docker build --cache-from "$PROJECT_REGISTRY:latest" \
      --label "org.opencontainers.image.title=$CI_PROJECT_TITLE" \
      --label "org.opencontainers.image.url=$CI_PROJECT_URL" \
      --label "org.opencontainers.image.created=$CI_JOB_STARTED_AT" \
      --label "org.opencontainers.image.revision=$CI_COMMIT_SHA" \
      --label "org.opencontainers.image.version=$CI_COMMIT_REF_NAME" \
      --tag "$PROJECT_REGISTRY:$CI_COMMIT_SHA" .
    - docker push "$PROJECT_REGISTRY:$CI_COMMIT_SHA"  # Pushing the built
image to the GitLab Registry

# Push master as latest
Push latest:
  stage: push
  tags:
    - docker-build
  resource_group: pipeline_lock
  only:
    - main
  variables:
    GIT_STRATEGY: none
    GIT_CHECKOUT: "false"
  script:
    - echo "$CI_REGISTRY_PASSWORD" | docker login -u "$CI_REGISTRY_USER" --
password-stdin $CI_REGISTRY
    - docker pull "$PROJECT_REGISTRY:$CI_COMMIT_SHA"  # Pull the image
before tagging
    - docker tag "$PROJECT_REGISTRY:$CI_COMMIT_SHA"
"$PROJECT_REGISTRY:latest"
    - docker push "$PROJECT_REGISTRY:latest"

# Push Tags
Push tag:
  stage: push
  tags:
    - docker-build
```

```
  resource_group: pipeline_lock
  only:
    - tags
  variables:
    GIT_STRATEGY: none
    GIT_CHECKOUT: "false"
  script:
    - echo "$CI_REGISTRY_PASSWORD" | docker login -u "$CI_REGISTRY_USER" --
password-stdin $CI_REGISTRY
    - docker tag "$PROJECT_REGISTRY:$CI_COMMIT_SHA"
"$PROJECT_REGISTRY:$CI_COMMIT_REF_NAME"
    - docker push "$PROJECT_REGISTRY:$CI_COMMIT_REF_NAME"

# Test Deploy
test-job1:
  stage: test
  tags:
    - shell-testing
  resource_group: pipeline_lock
  script:
    - echo "$CI_REGISTRY_PASSWORD" | docker login -u "$CI_REGISTRY_USER" --
password-stdin $CI_REGISTRY
    - docker ps -q --filter "ancestor=$PROJECT_REGISTRY:$CI_COMMIT_SHA" --
filter status=running | xargs -r docker stop
    - docker ps -q --filter "expose=8000" | xargs -r docker stop  # Stop any
containers using port 8000
    - docker run -d -p 8000:8000 "$PROJECT_REGISTRY:$CI_COMMIT_SHA"  #
Running the test container

# Production Deploy (Manual Start)
production-job1:
  stage: deploy
  when: manual
  tags:
    - shell-production
  resource_group: pipeline_lock
  script:
    - echo "Production deployment is manual. Starting the container..."
    - echo "$CI_REGISTRY_PASSWORD" | docker login -u "$CI_REGISTRY_USER" --
password-stdin $CI_REGISTRY
    - docker ps -q --filter "ancestor=$PROJECT_REGISTRY:$CI_COMMIT_SHA" --
filter status=running | xargs -r docker stop
    - docker ps -q --filter "expose=8000" | xargs -r docker stop  # Stop any
containers using port 8000
    - docker run -d -p 8000:8000 "$PROJECT_REGISTRY:$CI_COMMIT_SHA"  #
Running the production container
    - echo "Production container started successfully."
```

Nachdem du den Code eingefügt hast, klickst du auf „Commit Changes“.

Nun gehst du links über dem Pipeline-Editor zu den Pipelines und solltest deine Pipeline sehen!

Hurra, deine erste Pipeline läuft! Jetzt gehe zu den Jobs und klicke auf die jeweiligen Jobs. Dort siehst du den Job-Log und kannst versuchen nachzuvollziehen, was genau dort ausgeführt wird.

Du kannst die bereitgestellte Webanwendung unter http://test.rafisa.test:(dein Port) erreichen.

Abschluss und Zugriff

Herzlichen Glückwunsch! Du hast nun erfolgreich den Basislehrgang zu GitLab durchlaufen und bist berechtigt, Continuous Integration und Continuous Deployment (CI/CD) zu nutzen. Dies ermöglicht dir, automatisierte Arbeitsabläufe zu erstellen, um deine Projekte schneller und effizienter zu entwickeln, zu testen und bereitzustellen. Aber das ist noch nicht alles! Mit CI/CD kannst du noch viel mehr erreichen. Du kannst komplexe Pipelines konfigurieren, automatisierte Tests integrieren, Deployments automatisieren und vieles mehr. Diese Tools sollten deine Arbeit nun erheblich erleichtern. Nutze die Gelegenheit, den Pipeline-Code zu optimieren und in dein Projekt zu integrieren. Viel Erfolg dabei!