Gitlab CI/CD

Gitlab bietet eine automatische Continous Build/Continous Deployment Pipeline. Damit wird in jedem, entsprend vorbereiteten, Projekt nach einer Änderung des Git-Repositories automatisch der Code ausgechecked, gebaut, evtl. Tests durchgeführt und in ein ausführbares Format gebracht. Wenn diese Schritte erfolgreich sind, kann das fertige Programm auch in der ICC automatisch ausgeführt werden und Messdaten direkt in Gitlab beobachtet werden.

Im Wurzelverzeichnis eines Projektes muss die Datei `.gitlab-ci.yml` existieren. Diese wird dann mit im gemeinsamen GitLab-Runner über Kubernetes innerhalb der ICC ausgeführt. Für Spezialfälle können auch eigene Runner angebunden werden. Siehe ensprechenden Abschnitt weiter unten.

Minimales .gitlab-ci.yml

Um ein Projekte für die Ausführung in der ICC vorzubereiten muss der Code als Docker-Image zur Verfügung stehen. Die folgende Vorlage illustriert dies anhand eine Golang-Projektes. Für andere Sprachen müssen die Anweisungen im Abschnitt `job_build.script` angepasst werden. Eine vollständige Dokumentation für die Build-Spezifikation findet sich hier: https://docs.gitlab.com/ee/ci/yaml/README.html

Vorlage:

```
stages:
  - build

before_script:
  - git config --global http.sslVerify true

job_build:
  stage: build
  image: nexus.informatik.haw-hamburg.de/golang:1.8.3
  variables:
    CGO_ENABLED: "0"
    GOOS: "linux"
  script:
    - glide install
   # no tests for now - go test gitlab.informatik.haw-
hamburg.de/timadorus/timadorus-monitor/$(glide novendor)
    - go build -a -installsuffix cgo gitlab.informatik.haw-
hamburg.de/timadorus/k8s-job-monitor
    - mkdir -p /$CI_PROJECT_NAMESPACE/$CI_PROJECT_NAME/
    - mv ./k8s-job-monitor /$CI_PROJECT_NAMESPACE/$CI_PROJECT_NAME/
```

Während des Build-Prozesses Zugriff auf anderes Gitlab-

Repo gewähren

In der Regel werden für das Bauen von Software noch zusätzliche Komponenten wie Bibliotheken, Werkzeuge oder Daten benötigt. Diese werden zusammenfassend als Abhängigkeiten bezeichnet. Wenn diese Abhängigkeiten in zugriffsbeschränkten Repositories auf dem Gitlab zu Verfügung stehen, so können für einen Build-Prozess die notwendigen Zugangsinformationen mit sog. ***Deploy Keys*** (https://docs.gitlab.com/ce/ssh/README.html#deploy-keys) zur Verfügung gestellt werden.

Hierzu müssen Sie

1. Ein SSH Schlüsselpaar erzeugen. Dies sollte auf keinen Fall ihr persönliches Paar sein.
2. Den öffentlichen Schlüssel legen Sie in dem Projekt **auf das zugegriffen werden soll** als Deploy Key ab. Öffnen Sie dazu im Sidebar die Seite „*Settings | Repository*“ und erweitern den Abschnitt „*Deploy Keys*“. Legen Sie nun einen neuen Deploy Key an. Hier finden Sie auch einen Link auf detailierte Informationen zur Erzeugung und Formatierung der benötigten Schlüssel.
3. Den private Schlüssel legen Sie in das Projekt **aus dem heraus zugegriffen werden soll** als geheime Variable ab. Öffnen Sie dazu im Sidebar die Seite „*Settings | CI/CD*“ und erweitern den Abschnit „*Secret variables*“. Legen Sie eine Variable mit dem Namen „SSH_PRIVATE_KEY“ an und füllen Sie deren Wert auf den private Key.
4. In dem Projekt von dem aus zugegriffen werden soll fügen, Sie folgenden Abschnitt zur `.gitlab-ci.yml` hinzu (oder erweitern ihn, wenn Sie schon einen solchen Abschnitt haben:

```
before_script:
  # setup SSH if on ubuntu
  - eval $(bash ./scripts/setup-ssh.sh "$SSH_PRIVATE_KEY")
```

1. Folgendes Script dem zugreifenden Projekt unter `scripts/setup-ssh.sh` hinzufügen:

```
#!/bin/bash
#
# set up the SSH agent to access the repositories of the dependencies of
# this project. Currently only works on Ubuntu or Debian machines.
#
# use: setup-ssh #
# set the private key as project secret
#
SSH_PRIVATE_KEY="$1"
# check if this is actually ubuntu
ID="something else"
test -f /etc/os-release && source /etc/os-release
case "$ID" in
  debian|ubuntu)
    # Install ssh-agent if not already installed, it is required by Docker.
    which ssh-agent>/dev/null || ( apt-get update -y && apt-get install
openssh-client -y )

    # Run ssh-agent (inside the build environment)
    AGENT_SETUP=$(ssh-agent -s) ; echo "$AGENT_SETUP"

    # need info here too
```

```
    eval $AGENT_SETUP>/dev/null

    # Add the SSH key stored in SSH_PRIVATE_KEY variable to the agent store
    echo -n "#"
    ssh-add <(echo "$SSH_PRIVATE_KEY")
    echo -n "# Key loaded:  "
    ssh-add -l

    # For Docker builds disable host key checking. Be aware that by adding
that
    # you are susceptible to man-in-the-middle attacks.
    # WARNING: Use this only with the Docker executor, if you use it with
shell
    # you will overwrite your user's SSH config.
    mkdir -p ~/.ssh

    [[ -f /.dockerenv |]] && echo -e "Host *\n\tStrictHostKeyChecking
no\n\n"> ~/.ssh/config
  ;;
  *)
    echo "echo OS is $ID, which is unknown to mee, no setup done"
    exit 0
esac
```

Docker Images in Gitlab Build bauen

Es ist natürlich auch möglich Docker Images in einer Gitlab Pipeline zu bauen. Dazu ist die DOCKER_HOST Variable in der Gitlab Job Beschreibung zu setzen und das stable-dind images als „service“ im Job zu setzen. Beispiel für eine .gitlab-ci.yml aus https://gitlab.informatik.haw-hamburg.de/ail/haw-world/blob/master/.gitlab-ci.yml:

```
stages:
    - dockerize

variables:
 DOCKER_HOST: "tcp://localhost:2375"
 DOCKER_REGISTRY: "nexus.informatik.haw-hamburg.de"
 SERVICE_NAME: "haw-world"

createImage:
 stage: dockerize
 image: nexus.informatik.haw-hamburg.de/docker:stable-dind
 services:
   - nexus.informatik.haw-hamburg.de/docker:stable-dind
 script:
   - docker login -u $NEXUS_USER -p $NEXUS_PW $DOCKER_REGISTRY
   - docker build -t $DOCKER_REGISTRY/$SERVICE_NAME:$CI_PIPELINE_ID .
   - docker push $DOCKER_REGISTRY/$SERVICE_NAME:$CI_PIPELINE_ID
```

Deployment einer Applikation aus Gitlab in die ICC

den Artikel dazu finden Sie im entsprechenden Abschnitt der Dokumentation für die Informatik Computer Cloud.

Eigene Build Runner

Siehe https://docs.gitlab.com/ce/ci/runners/README.html für weitere Details.