Gitlab

een CI / CD pipeline

Gitlab is meer dan een code repository

Ook:

- Issues bijhouden
- Documentatie (wiki’s)
- Hosten van packages, containers
- Continuous Integration (CI) en Continues Deployment (CD) pipelines

Deze laatste gaan we nu onderzoeken.

Voor nu gebruiken we de hosted gitlab-omgeving.

Het mooie van gitlab is dat je deze ook zelf kan hosten, wellicht iets voor een andere keer.

Een CI/CD pipeline

We gaan een CI/CD pipeline maken voor een statische web-pagina wat het volgende inhoudt:

- De pipeline start bij elke nieuwe commit (na de git push)
- Een docker image (nginx) wordt gemaakt (build) en geupload (push) naar de Gitlab container registry (op de gitlab-server zelf)
- Het image wordt gedeployed op een server (doblo.osbus.nl)
- Resultaat: een eenvoudige statische web-pagina (te zien via ff.osbus.nl)

Nodig:

- Gitlab omgeving
- Linux server (in dit geval ubuntu 22.04) met een webserver (bv. nginx), docker en gitlab-runners

Een eigen gitlab runner

CI/CD jobs worden gerund door “gitlab runners”:

- Deze zijn op de gitlab-server beschikbaar.
- Maar je kunt ook met “eigen runners” werken.

Als je ook automatisch wilt deployen (“CD”) naar jouw servers dan **beter een eigen runner maken**:

- Meer controle en daardoor ook veiliger te maken.
- De connectie wordt opgezet van runner naar gitlab server (dus geen open poorten)

gitlab server

GitLab

produktie server

2) build en publish

3) deploy

1) add, commit en push

git project met: index.html, Dockerfile, .gitlab-ci.yml

Automatisch deployen ("CD")

Er zijn (minimaal) 2 scenario's om automatisch te deployen:

1) maak een private key beschikbaar zodat je met jouw runner op je server kan inloggen om te deployen.
   a) **Nadeel**: je breekt het principe dat je **private key private moet blijven waar die gegenereerd** is. Run je je eigen gitlab-server dan iets "aanvaardbaarder"...
   b) **[Nadeel]**: jouw servers moeten port 22 open hebben voor een lokale docker

Voorbeeld deploy (fout)

```
deploy:                                     # job name
  image: alpine:latest
  stage: deploy                             # stage
  script:
    - apk update && apk add openssh-client
    - ssh -i  $ID_RSA -o StrictHostKeyChecking=no $SRV_USER@$SRV_IP "docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY"
    - ssh -i $ID_RSA -o StrictHostKeyChecking=no $SERVER_USER@$SERVER_IP "docker pull $TAG_COMMIT"
    - ssh -i $ID_RSA -o StrictHostKeyChecking=no $SERVER_USER@$SERVER_IP "docker container rm -f my-app || true"
    - ssh -i $ID_RSA -o StrictHostKeyChecking=no $SERVER_USER@$SERVER_IP "docker run -d -p 5007:80 --name my-app $TAG_COMMIT"
```

Automatisch deployen (“CD”)

2) gebruik verschillende runners (een “docker” runner om je images te maken, **een “shell” runner** voor de deployment (om deze images op je servers te pullen en starten (**mbv je eigen docker daemon**)))
   a) **Voordeel**: geen ssh login en dus ook geen private key nodig

Ik heb voor 2 gekozen.

We hebben dus een docker runner en een shell runner nodig.

De stappen

1. Maak een gitlab project aan, met een statische html pagina (index.html)
2. Maak een Dockerfile die een nginx image maakt en de index.html serveert:

```
FROM nginx:latest

COPY index.html /usr/share/nginx/html
```

3. Maak een “eigen gitlab runner” (in dit geval 2)
4. Maak je CI/CD configuratie: hierin beschrijf je de pipeline(s) met bijbehorende gitlab runners en commando’s

Maken en activeren eigen gitlab runner(s)

Maak eerst de runners aan in je gitlab omgeving:

1. een docker runner. Deze start een docker die images kan maken en pusht naar de docker registry van de gitlab server. Deze wordt gebruikt in de build en publish stage ("CI")
2. een shell runner. Deze gebruikt de docker daemon **op jouw server** om docker commando's te runnen (pull en run). Deze wordt gebruikt in de deploy stage ("CD")

Je krijgt per runner een "**authentication token**" waarmee de runner een verbinding kan maken met de gitlab server.

Doe dan op jouw server:

```
# curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" | bash

# apt update; apt install gitlab-runner

# gitlab-runner register -n --url https://your_gitlab.com --token runner1_token --executor  docker \

                --description "Build Docker Runner" --docker-image "docker:stable"   --docker-privileged
```

En voor de tweede runner:

```
# gitlab-runner register -n --url https://gitlab.com --token runner2_token --executor  shell \

                --description "Deployment Shell Runner"

# systemctl enable --now gitlab-runner;  # enable en start de gitlab runners.
```

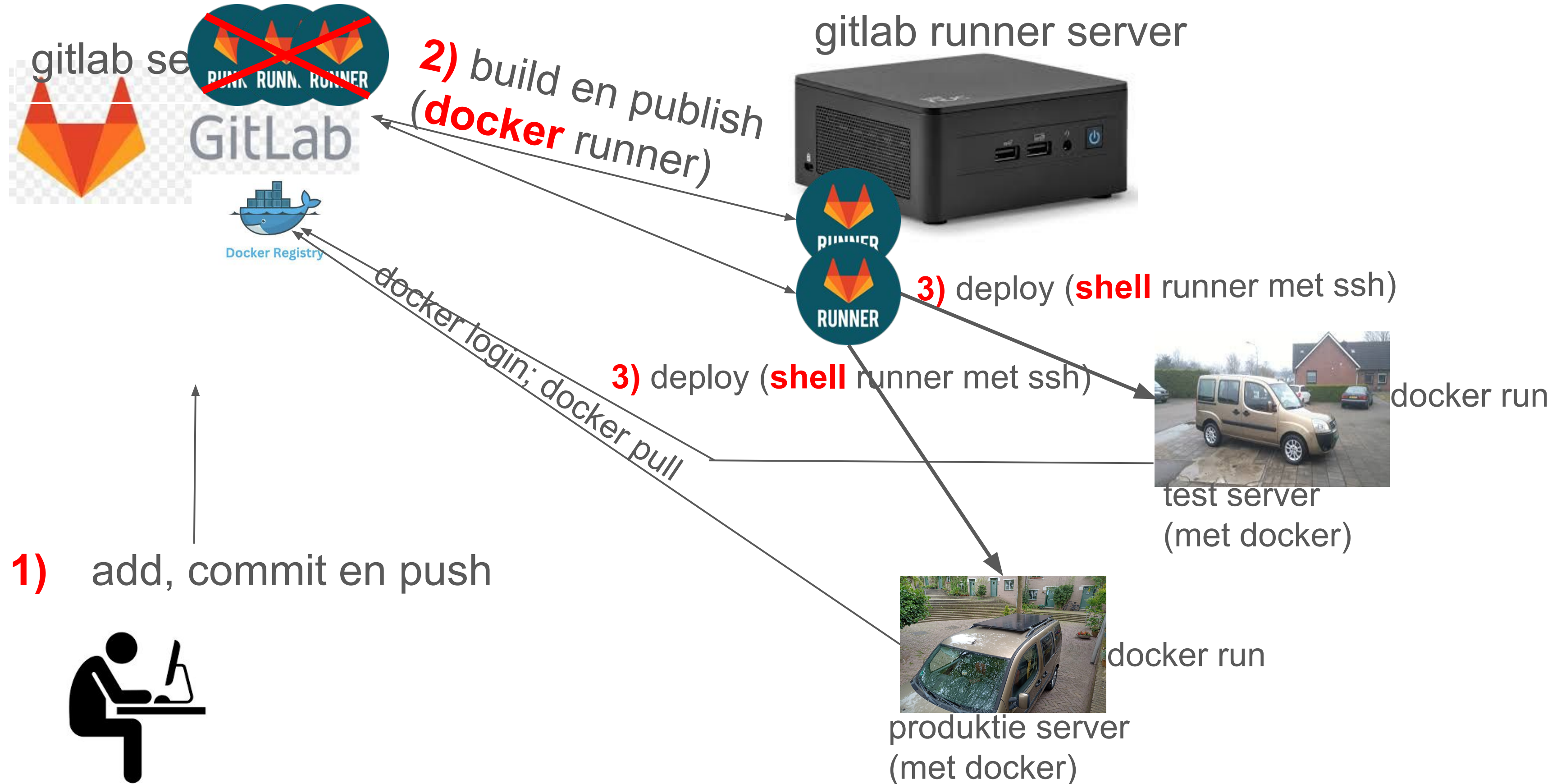
privileged mode vanwege “dind”

Voor de docker runner: deze heeft “privileged mode” nodig omdat je “docker in docker (dind)” wilt runnen: je maakt je docker images (“docker build ..”) binnen je door je runner gemaakte docker.

Voor de shell runnner: de user (gitlab-runner) moet in de group “docker” zitten om docker commando’s op je server uit te voeren.

```
# usermod -aG docker gitlab-runner
```

Scenario met 2 eigen runners



git project met: index.html, Dockerfile, .gitlab-ci.yml

Tijd voor de CI / CD configuratie

Dit gaat in gitlab middels het file **.gitlab-ci.yml** in de root van je project directory.

Eerst definieer je je stages, bv.:

```
stages:
  - publish
  - deploy
```

Maak gebruik van enkele bekende variabelen:

CI_JOB_TOKEN: authentication token voor de Gitlab registry (alleen geldig voor de running job)

CI_REGISTRY: url van de container registry bij dit project

CI_REGISTRY_IMAGE: url van de container registry bij dit project voor images

CI_COMMIT_REF_NAME: branch name van it project (bv. “main”)

CI_COMMIT_SHORT_SHA: commit revision number (eerste 8 karakters)

En definieer zelf variabelen om je images per commit te kunnen onderscheiden (in .gitlab-ci.yml):

```
variabelen:
    TAG_LATEST: $CI_REGISTRY_IMAGE/$CI_COMMIT_REF_NAME:latest
    TAG_COMMIT: $CI_REGISTRY_IMAGE/$CI_COMMIT_REF_NAME:$CI_COMMIT_SHORT_SHA
```

publish stage in .gitlab-ci.yml

```
build_publish:                          # job name
  #image: docker:latest                 #  niet nodig (default is het geregistreerde docker image voor de runner (docker:stable))
  stage: publish
  services:
    - name: docker:dind
      alias: mydockerhost
  variables:
    DOCKER_HOST: tcp://mydockerhost:2375/   # wordt gestart op de server waar de runner draait
    DOCKER_DRIVER: overlay2                 # betere performance
    DOCKER_TLS_CERTDIR: ""                  # Disable TLS (default gebruik van TLS sinds dind versie 19.03)
  tags:
    - docker                            # my docker runner
  script:
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY   # gitlab heeft een eigen docker registry
    - docker build -t $TAG_COMMIT -t $TAG_LATEST .                    # hier wordt het Dockerfile gebruikt
    - docker push $TAG_COMMIT
    - docker push $TAG_LATEST
```

deploy stage in .gitlab-ci.yml

```
deploy:                  # job name
  image: alpine:latest   # gebruik een "common" image voor deployment
  stage: deploy
 tags:
   - shell               # een shell runner, runt docker commando's en gebruikt de docker daemon op de produktie site
  script:
   - [ssh other-server] docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY
   - [ssh other-server] docker pull $TAG_COMMIT
   - [ssh other-server] docker container rm -f my-app || true
   - [ssh other-server] docker run -d -p 5007:80 --name my-app $TAG_COMMIT
```