CI/CD

Dette er et tomcat projekt eksempel, der bruger jenkins til ci/cd og docker til at køre services

Forudsætninger

Dette projekt forudsætter kørelse på en linux maskine, da projekt test scripts er bash specifike.
Dette projekt påkræver følgende:

- Java 8
- Docker (bruger skal være sat op til at kunne køre sudo less docker commands: `sudo usermod -aG docker $USER`)
- curl (følger med i fleste distros)
- github konto

Konfiguration

Github

Lav et nyt github repository der inkludere alle projktets filer. (Kan godt excludere `job-config.xml, config, runCI.sh` da de kun bruges til starte jenkins) Lav en personal access token med følgende checkmarks:

```
repo
repo:status
repo_deployment
public_repo
repo:invite
write:repo_hook
read:repo_hook
read:user
user:email
```

Lokalt

I `config` filen, udfyld de tome fælter:

```
GITHUB_USER=din-github-bruger
GITHUB_API_KEY=din-api-key
GITHUB_REPO=det-nye-git-repository-navn
```

Kør

Åben terminalen og stå inde i projekt mappen. Kør:

```
bash runCI.sh
```

Vent indtil udførelsen er færdig. En jenkins admin kode vil blive printet i terminalen.

```
Service is ready!
Jenkins admin password: 5b41c1d2e70042668a7f335a002f186f
```

Denne kode kan bruges sammen med admin brugernavnet til at logge ind på ens jenkins instance. I din favorit browser, gå ind på: `localhost:8080` for at interagere med jenkins. Hvis bygget er blevet grøn, kan du gå ind på `localhost:8081/myapp/` for at se tomcat app'en kørende.

Test af flow

Du kan prøve at lave ændringer i det nye repo på github. Jenkins vil hvert minut, se efter ændringer på dit repository. Så snart jenkins ser en ændring, prøver den at bygge projektet og deploy lokalt til din maskine.

Stop alle services

I terminalen, skriv:

```
docker stop jenkinsci
docker rm jenkinsci
docker stop myApp
docker rm myApp
```

Du kan også tvinge en container til at standse og fjerne på samme tid:

```
docker rm jenkinsci -f
docker rm myApp -f
```

Script gennemgang

Scriptet `runCI.sh` bruges til at starte demo'en, men det betyder ikke at man ikke kan lære noget fra det. I denne sektion forklares hvert step og kan bruges som viden til andre automatiserings projekter.

Flexibilitet i ens scripts er vigtigt. Indsamling af projekt specifik variabler gøres simpelt ved at source dem fra en ekstern fil. Dette kunne også gøres via et `case` script og tage imod argumenter.

```
source ./config
```

Vi prøver at fjerne måske eksisterende container.

```
docker stop jenkinsci
docker rm jenkinsci
```

> `Docker stop <container-name>` stopper en container
> `Docker rm <container-name>` fjerner en container

Vi starter en ny container instance.

```
docker run \
  -d \
  --name jenkinsci \
  --group-add $(getent group docker | cut -d: -f3) \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -p 8080:8080 \
  jenkinsci/blueocean:1.4.2
```

> `docker run args*` Hent image hvis den ikke findes lokalt, start ny container hvis kan, og kør den nye container instance
> `-d` Køre container som baggrundsprocess
> `--name` Container navn
> `group-add` Tilføj process til en host gruppe. (Bruges dette tilfælde i forbindelse med at køre docker commands fra en docker container)
> `-v` volume sti. Bruges til at vedvare filer mellem container og host. (Bruges dette tilfælde i forbindelse med at køre docker commands fra en docker container)
> `p` specificere container port (indbound:outbound)

> `jenkinsci/blueocean:1.4.2` image/navn:image_tag som container instance startes fra

De næste steps er Jenkins specifike. Dette kan bruges til at deploy en jenkins instance nemt uden manuelt arbejde.
Indsamling af admin password

```
while ! ADMIN_PASSWORD=$(docker exec jenkinsci cat /var/jenkins_home/secrets/initialA
  sleep 1
done
```

> Vi prøver at hente init passworded
> `docker exec <container-name> <command>` Kør os specifike commands, inde i en container

Vente på at jenkins admin interface startes

```
while ! curl -i -s -u "admin:$ADMIN_PASSWORD" http://localhost:8080/credentials/store
  sleep 1
done
```

Jenkins med blueocean, påkræver CRUMB værdi, for at interagere med REST interfacet.

```
CRUMB=$(curl -u "admin:$ADMIN_PASSWORD" 'http://localhost:8080/crumbIssuer/api/xml?xp
```

> Dette er en sikkerheds feature i Jenkins. Dette er ikke defualt tændt i en Jenkins standard image.

POST credentials.

```
curl -i -X POST -u "admin:$ADMIN_PASSWORD" 'http://localhost:8080/credentials/store/s
--header "Jenkins-Crumb:$CRUMB" \
--data-urlencode "json={
   '':'0',
   'credentials':{
      'scope':'GLOBAL',
      'username':'$GITHUB_USER',
      'password':'$GITHUB_API_KEY',
      'id':'github-api-token',
      'description':'github api token',
      'stapler-class':'com.cloudbees.plugins.credentials.impl.UsernamePasswordCredent
      '\$class':'com.cloudbees.plugins.credentials.impl.UsernamePasswordCredentialsIm
   }
}"
```

> Jenkins api interface er ikke en normalt interface. Dens endpoints bliver auto bygget ud fra de plugins som er installeret på ens jenkins instance.

Forbereder job file

```
cp job-config.xml job-config-tmp.xml
sed -i "s/<repoOwner>.*<\/repoOwner>/<repoOwner>$GITHUB_USER<\/repoOwner>/g" job-conf
sed -i "s/<repository>.*<\/repository>/<repository>$GITHUB_REPO<\/repository>/g" job-
```

Bruger jenkins intended api til at tilføje et nyt job

```
curl -i -X POST -u "admin:$ADMIN_PASSWORD" "http://localhost:8080/createItem?name=$GI
--header "Jenkins-Crumb:$CRUMB" \
--header "Content-Type:text/xml" \
--data-binary @job-config-tmp.xml
```

Man kan bruge ens browser debugger til at finde flere REST interactioner med de auto-generated endpoints.

Referencer

docker run
Jenkinsfile syntax