4DVOP - DevOps Engineer

Noémie PAÏS (304084) - William DA CONCEIÇÃO (223689) - Cyril BREURE (294899) - Théo WROBLEWSKI (304222)

POZOS

Sommaire

1- Context

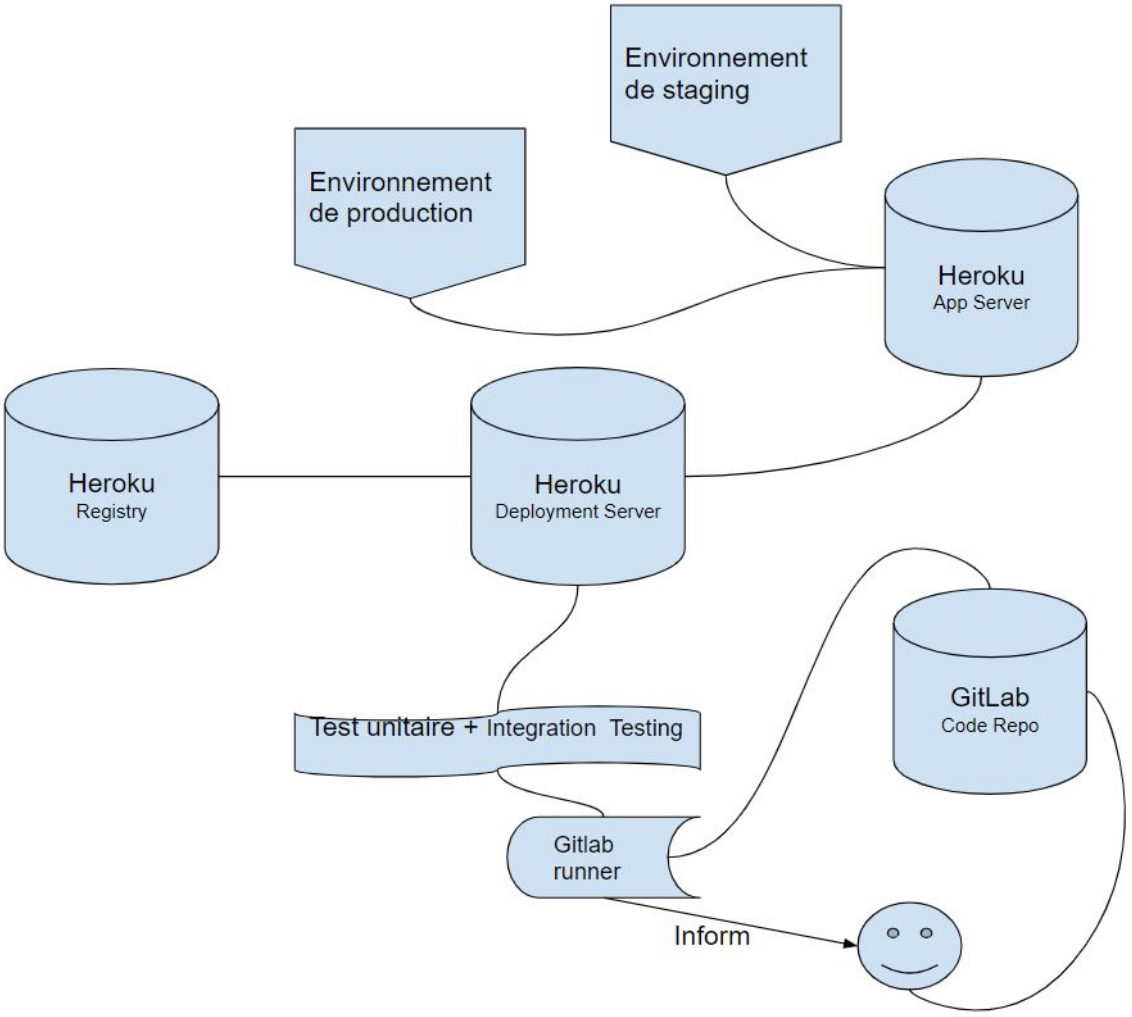
POZOS est une société informatique située en France qui développe des logiciels à destination d'établissement scolaire.

Après avoir déjà fait appel à nous lors d'un POC, pour montrer comment la technologie Docker pouvait les aider et montrer son efficacité, La DSI de POZOS souhaite que nous gérions l'évolution du processus de déploiement.

La direction de leur secteur Innovation veut en effet, procéder à quelques modifications sur l'infrastructure existante pour assurer que le processus d'intégration et de déploiement soit totalement automatisé, en intégrant la culture DevOps.

POZOS veut que nous présentions un POC pour montrer comment le pipeline CI / CD peut aider et, surtout, montrer à quel point cette technologie est efficace.

2- Infrastructure



3- Before starting

Il faut avant toute chose s'assurer d'avoir lancé le playbook Ansible. Ce playbook permet d'installer la dernière version de docker, d'installer la commande "docker compose" et d'installer les différentes dépendances requises.

Les commandes se trouvant dans le fichier 4dvops-setup.md permettent de mettre en place des testes pour s'assurer du fonctionnement de notre solution. Ces manipulations permettront de créer un runner, en plus d'initialiser l'environnement Gitlab.

On commence par renseigner les valeurs dans les variables suivantes à l'adresse :

localhost:10080/{username}/{project-name}/-/settings/ci_cd

Et lesdites variables sont :

- *HEROKU_APP_STAGING* : est le nom de l'application heroku destinée à l'environnement staging.
- *HEROKU_APP_PRODUCTION* : est le nom de l'application heroku destinée à l'environnement de production.
- *HEROKU_API_KEY* : est la clef permettant de se lier à HEROKU.
- *DOCKER_USERNAME* : est le nom d'utilisateur de DockerHub.
- *DOCKER_PASSWORD* : est le mot de passe qui correspond à l'utilisateur DockerHub.

Après cela, les tests de notre pipeline seront prêts.

4- Pipeline component integration and build

Note : Vous pouvez vous munir du code source de notre projet et l'ajouter à votre gitlab local. Une fois, cela fait, vous devriez avoir un pipeline s'exécutant dans la partie correspondante de votre projet.

On commence par créer une nouvelle branche nommée "testing", puis une branche nommée "staging" (où se trouvera la prochaine version qui sera disponible au client). En réalisant cela, deux nouveaux pipelines devraient s'exécuter, les élevants au nombre de 3, dont l'un appliqué à la branche "testing" qui contient seulement une étape.

Nous allons maintenant plonger dans le code pour vous expliquer son fonctionnement.

```yaml
image: docker:latest

services:
  - docker:dind

variables:
    DOCKER_HOST: tcp://docker:2375
    DOCKER_TLS_CERTDIR: ""
    CONTAINER_IMAGE: "supinfo/simple_api"
```

Ici, nous avons 3 parties :

- La partie image : elle nous permet de spécifier le nom de l'image que va utiliser notre CI (en l'occurrence : docker:latest).
- La partie service : elle nous permet de spécifier les services à activer pour le CI (ici docker in docker : DIND).
- La partie variable : elle nous permet de spécifier des variables utilisées par notre CI ou par ses services. Ici DOCKER_HOST et DOCKER_TLS_CERTDIR, nous permettent de créer un serveur HTTP, quant au CONTAINER_IMAGE, il sera utilisé plus tard dans le CI.

```
########################################
########          JOBS          ########
########################################

stages:
    - test
    - release

before_script:
    - echo "Connecting to docker and heroku registry"
    - echo $DOCKER_PASSWORD | docker login -u $DOCKER_USERNAME --password-stdin
    - docker login -u _ -p $HEROKU_API_KEY registry.heroku.com
```

La partie stages est décomposée en deux parties : test et release, qui seront explicitées après.

La partie intitulée “before_script” sera lancée à chaque fois et permettra de mettre en place les connexions vers notre docker et notre registry Heroku. Ainsi, nous pouvons y retrouver nos identifiants (voir capture ci-dessus), ou plutôt les variables qui les contiennent.

```
47  test:
48    stage: test
49    script:
50      - docker build -f simple_api/Dockerfile.test .
51
```

L'étape ”test” décrite ci-dessus est exécutée qu’importe le nom de la branche. Elle comprend une seule partie éponyme qui va exécuter une commande afin de build notre Dockerfile de test.

Ce Dockerfile va exécuter notre fichier de test unitaire nommé student_age_test.py, dans un environnement vierge. Il se trouve dans nos fichiers de codes et s’intitule Dockerfile.test (comme indiqué dans la capture ci-dessus).

```
319  Step 8/8 : RUN python ./student_age_test.py
320   ---> Running in 7c5fb2dd9e28
321  .
322  ----------------------------------------------------------------------
323  Ran 1 test in 0.000s
324  OK
325  Removing intermediate container 7c5fb2dd9e28
326   ---> eb380c30d79e
327  Successfully built eb380c30d79e
328  Job succeeded
```

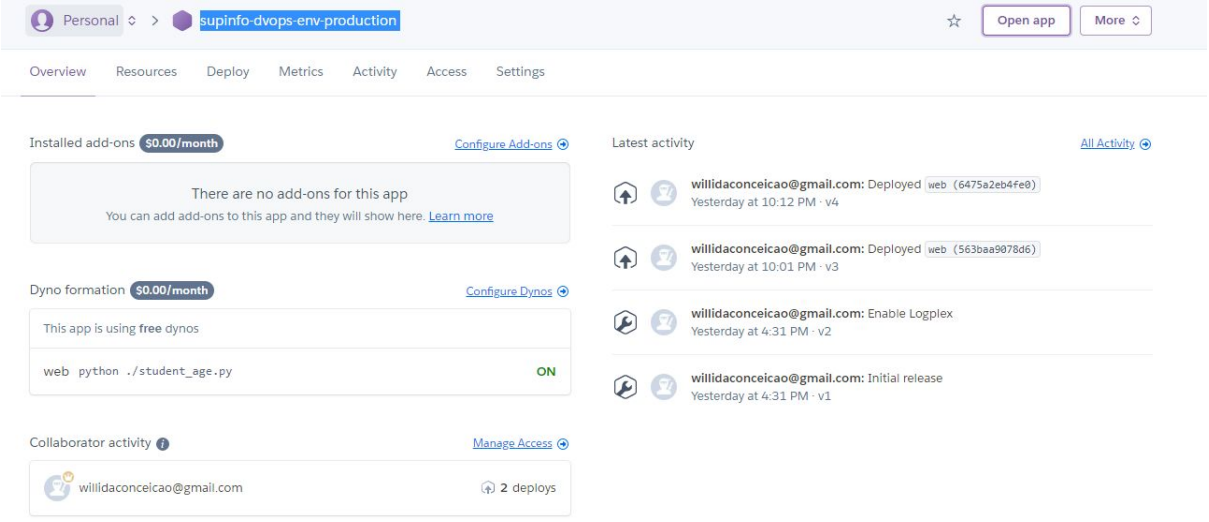
Voici le résultat du pipeline “test” que l’on obtient.

Par exemple, si nous “pushons” sur la branche “master” ou “staging” les parties du CI respectives seront exécutées.

```
release-prod:
    stage: release
    script:
    - echo "Start releasing for production environnement"
    - docker build -f simple_api/Dockerfile -t $CONTAINER_IMAGE:stable .
    - docker tag $CONTAINER_IMAGE:stable registry.heroku.com/$HEROKU_APP_PRODUCTION/web
    - docker push registry.heroku.com/$HEROKU_APP_PRODUCTION/web
    - docker run --rm -e HEROKU_API_KEY=$HEROKU_API_KEY wingrunr21/alpine-heroku-cli container:release web --app $HEROKU_APP_PRODUCTION
    only:
    - master
```

Cette étape du CI va récupérer le Dockerfile qui va alors être construit, ou “build”, c’est-à-dire exécuter et créer dans un conteneur docker. Ensuite, il se verra attribuer le tag “stable” et poussé dans le registry Heroku puis release dans notre application Heroku.

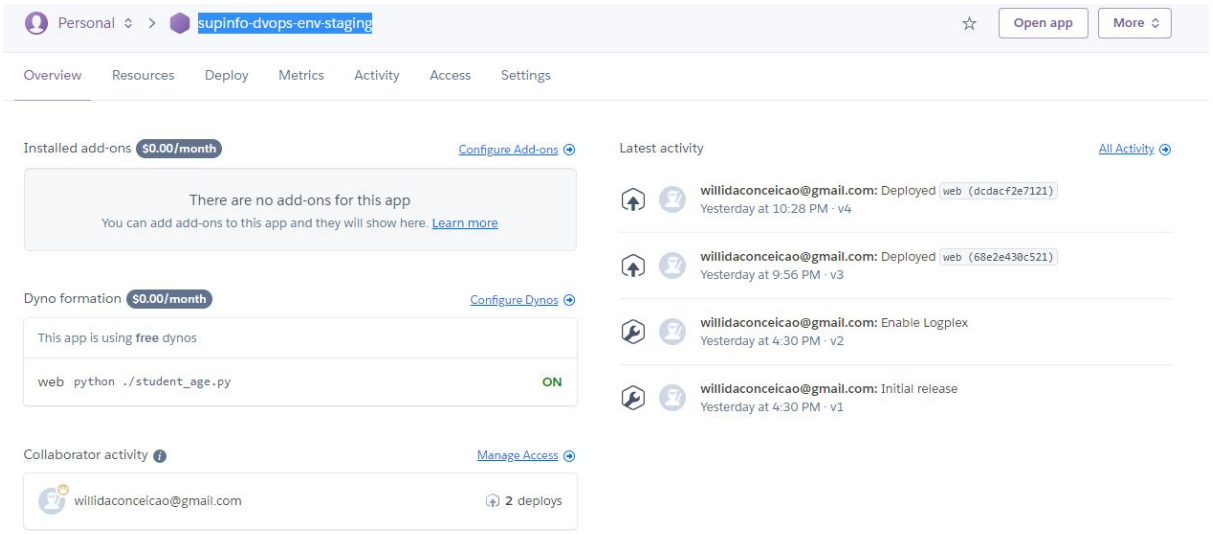
Ensuite, il sera lancé et mis en production. On peut en voir un exemple ci-dessous :



```yaml
release-staging:
    type: deploy
    stage: release
    script:
    - echo "Start releasing for staging environnement"
    - docker build -f simple_api/Dockerfile -t $CONTAINER_IMAGE:staging .
    - docker tag $CONTAINER_IMAGE:staging registry.heroku.com/$HEROKU_APP_STAGING/web
    - docker push registry.heroku.com/$HEROKU_APP_STAGING/web
    - docker run --rm -e HEROKU_API_KEY=$HEROKU_API_KEY wingrunr21/alpine-heroku-cli container:release web --app $HEROKU_APP_STAGING
    only:
    - staging
```

La partie ci-dessus, est l'étape de staging qui va réaliser les mêmes étapes que celle de production, mais déployer vers une application Heroku dédiée, avec des tags de docker en version staging (ce qui explique le :staging après chaque "$CONTAINER_IMAGE").

On aura alors le même résultat que pour l'environnement de production sur notre environnement staging :



Voici le résultat des étapes de la pipeline "release" , ici est représenté celui de "release-staging"

```
371  $ docker run --rm -e HEROKU_API_KEY=$HEROKU_API_KEY wingrunr21/alpine-heroku-cli container:release web --app
     $HEROKU_APP_STAGING
372  Unable to find image 'wingrunr21/alpine-heroku-cli:latest' locally
373  latest: Pulling from wingrunr21/alpine-heroku-cli
374  ff3a5c916c92: Pulling fs layer
375  94f111771efe: Pulling fs layer
376  87fbae1a7501: Pulling fs layer
377  a3e878edcc9d: Pulling fs layer
378  a3e878edcc9d: Waiting
379  87fbae1a7501: Verifying Checksum
380  87fbae1a7501: Download complete
381  ff3a5c916c92: Verifying Checksum
382  ff3a5c916c92: Download complete
383  ff3a5c916c92: Pull complete
384  94f111771efe: Verifying Checksum
385  94f111771efe: Download complete
386  94f111771efe: Pull complete
387  87fbae1a7501: Pull complete
388  a3e878edcc9d: Verifying Checksum
389  a3e878edcc9d: Download complete
390  a3e878edcc9d: Pull complete
391  Digest: sha256:f4a0a7c79656c060261713991391680afba9bf28a76dbf79b24bbf17d909cbb4
392  Status: Downloaded newer image for wingrunr21/alpine-heroku-cli:latest
393  Releasing images web to supinfo-dvops-env-staging... done
395  Job succeeded
```

6- Bonus stage

Comme nous avons deux applications Heroku (supinfo-dvops-env-staging et supinfo-dvops-env-production) , les bonnes pratiques d'un CI est d'avoir des builds différents pour les environnements de production (celle à fournir au client) et staging (la prochaine version qui sera disponible au client).

L'étape "release" du CI est donc divisé en deux : release-prod et release-staging qui se déclenchent en fonction de la branche où on procède à un commit (master ou staging).