Le génie pour l'industrie

Patrick Bertin TALLA TAMADJE(TALP19077806)
Nadine Carelle MASSUDOM MGNIE(MASN29548301)
MOISE MAHARA MOUKENE (MAHM22089208)

Introduction à l'approche DevOps

Cours: Log680.
Equipe : 07
Laboratoire 2

Travail présenté à
Vincent Boivin

Remis le 28 Juin 2021
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

1. INTRODUCTION

Le présent document a pour but de décrire ce que nous avions réalisé dans le cadre du laboratoire 2 qui consistait à mettre en place un pipeline d'intégration continue qui contient l'automatisation des tests et la conteneurisation d'une application qui va permettre de supporter le développement de l'application Oxygène CS.

Le but principal de notre travail était de faciliter la phase d'intégration du logiciel.

Pour atteindre l'objectif de ce laboratoire , nous avons structuré notre travail en trois étapes:

Premièrement, nous avons créé un répertoire GitHub, par la suite nous avons créé et mis en ligne une image avec Docker pour faire la conteneurisation de l'application et paramétré Travis CI qui est la technologie que nous avons décidé d'utiliser pour notre intégration continue .

Enfin nous avons apporté des modifications au code source qui nous a été fourni au préalable afin de vérifier le bon fonctionnement de notre pipeline.

2. Explication du fonctionnement de notre intégration continue

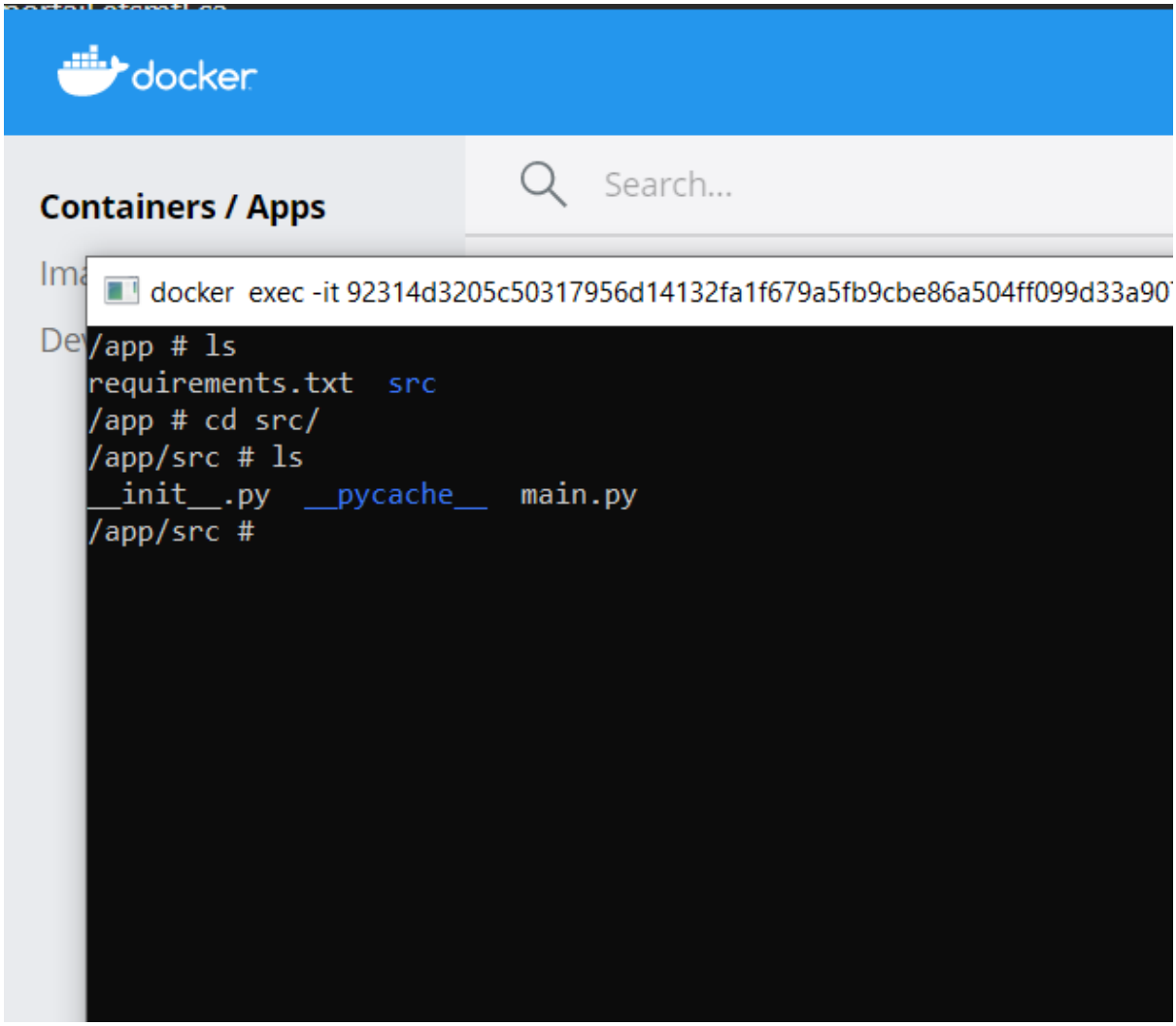
Pour expliquer comment fonctionne notre intégration continue, nous allons le faire en plusieurs points suivants:

Utilisation de l'outil CI

Nous avons porté notre choix sur TravisCI comme outil parce qu'il est gratuit, il s'intègre facilement avec un projet Github et la configuration du projet est très rapide car il ne requiert aucune installation . Puisque Travis CI fournit un service en ligne, alors il nous a permis de faire la compilation, de faire les tests et de déployer le code source développé qui est lié avec le service d'hébergement du code source GitHub. Avec Travis CI, le pipeline se déclenche dès lors qu'on charge le code source sur la branche dev ou main et il permet de suivre clairement les étapes(faire les tests et compiler, construire l'image Docker) et de conserver la nomenclature de chaque pipeline. Nous avons également décidé de travailler avec Travis CI parce qu'il prend en charge Python qui est le langage de programmation que nous avons préféré, en plus TravisCI a des plugins intégrés.

Utilisation du Docker et la structure de l'image

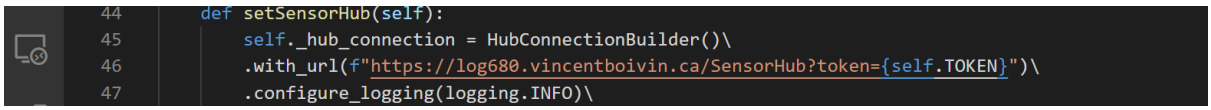
Nous avons utilisé docker pour contenaliser notre image créée à partir de l'image alpine dans docker hub. Notre image est nommée etslog680/image_hvac de 58.5MB que nous avons publié dans docker hub. C'est dans cette image que notre application est déployée lorsque les tests sont réussis lors du processus d'intégration continue



Modifications apportées au code

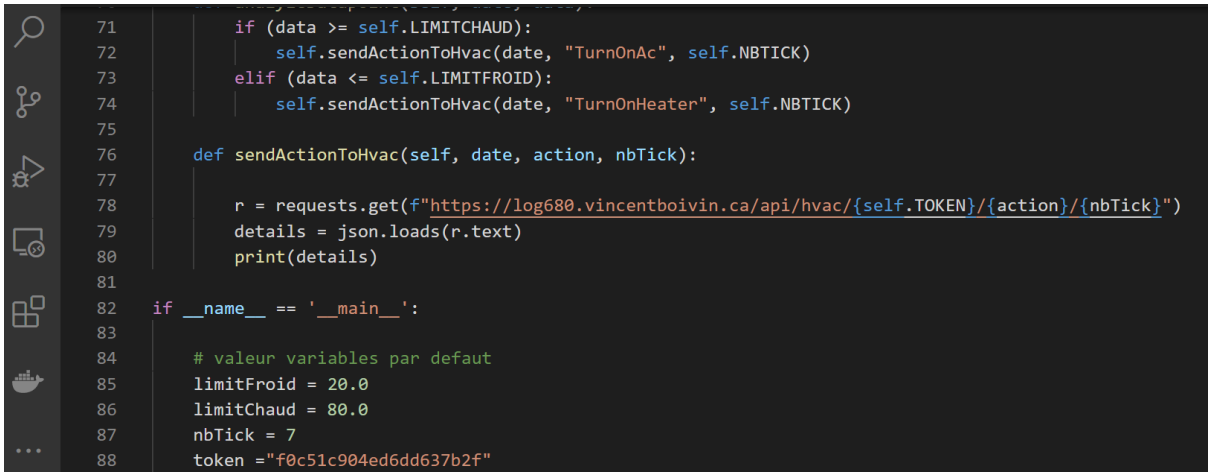
De la ligne 11 à la ligne 23 nous avons apporté des modifications pour définir les différents paramètres et leurs validations.

La ligne 46 a été modifié pour référer au endpoint qui nous permettra de lire les données

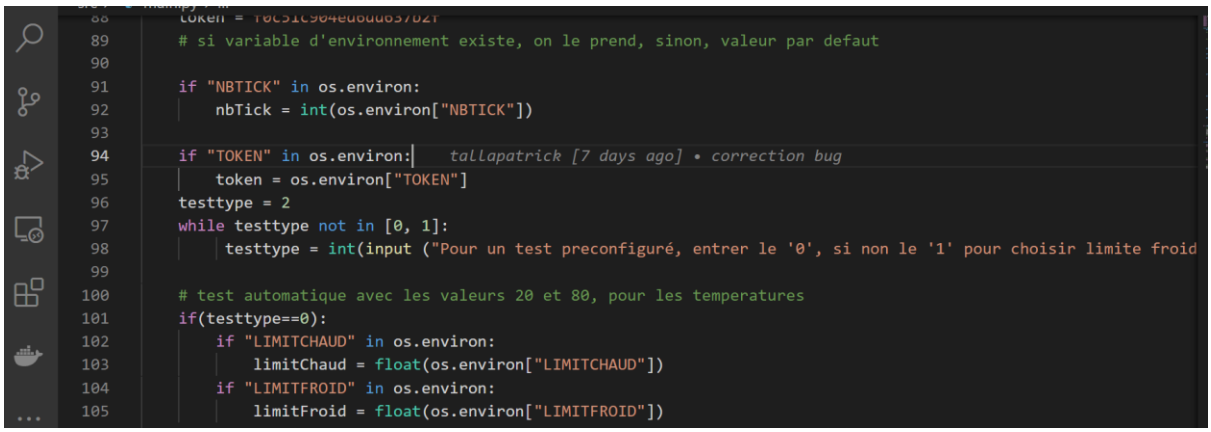


```
44      def setSensorHub(self):
45          self._hub_connection = HubConnectionBuilder()\
46          .with_url(f"https://log680.vincentboivin.ca/SensorHub?token={self.TOKEN}")\
47          .configure_logging(logging.INFO)\
```

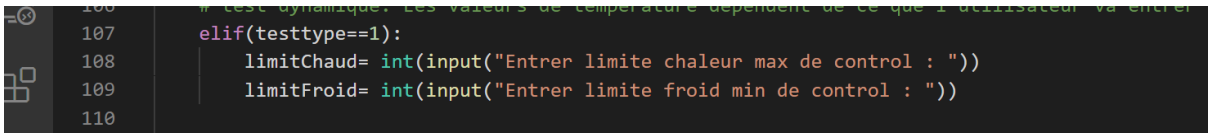
À partir de la ligne 71 jusqu'à la ligne 106 nous avons fait des modifications pour rendre le code dynamique en remplaçant les valeurs limites chaudes et froides par les variables pour les différents modes de fonctionnement



```
71          if (data >= self.LIMITCHAUD):
72              self.sendActionToHvac(date, "TurnOnAc", self.NBTICK)
73          elif (data <= self.LIMITFROID):
74              self.sendActionToHvac(date, "TurnOnHeater", self.NBTICK)
75
76      def sendActionToHvac(self, date, action, nbTick):
77
78          r = requests.get(f"https://log680.vincentboivin.ca/api/hvac/{self.TOKEN}/{action}/{nbTick}")
79          details = json.loads(r.text)
80          print(details)
81
82  if __name__ == '__main__':
83
84      # valeur variables par defaut
85      limitFroid = 20.0
86      limitChaud = 80.0
87      nbTick = 7
88      token ="f0c51c904ed6dd637b2f"
```

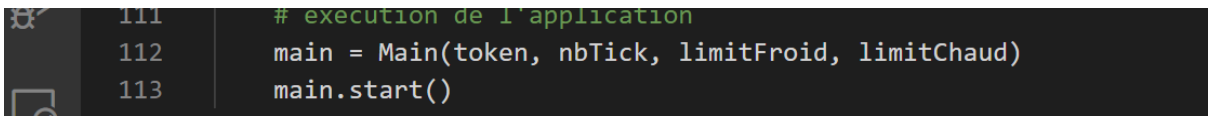


```
89      # si variable d'environnement existe, on le prend, sinon, valeur par defaut
90
91      if "NBTICK" in os.environ:
92          nbTick = int(os.environ["NBTICK"])
93
94      if "TOKEN" in os.environ:    tallapatrick [7 days ago] • correction bug
95          token = os.environ["TOKEN"]
96      testtype = 2
97      while testtype not in [0, 1]:
98          testtype = int(input ("Pour un test preconfiguré, entrer le '0', si non le '1' pour choisir limite froid
99
100     # test automatique avec les valeurs 20 et 80, pour les temperatures
101     if(testtype==0):
102         if "LIMITCHAUD" in os.environ:
103             limitChaud = float(os.environ["LIMITCHAUD"])
104         if "LIMITFROID" in os.environ:
105             limitFroid = float(os.environ["LIMITFROID"])
```



```
107     elif(testtype==1):
108         limitChaud= int(input("Entrer limite chaleur max de control : "))
109         limitFroid= int(input("Entrer limite froid min de control : "))
110
```

Nous avons fait des modifications à la ligne 108 pour passer le token, nbTick, limitFroid et limitChaud en paramètre .



```
111         # execution de l'application
112         main = Main(token, nbTick, limitFroid, limitChaud)
113         main.start()
```

Nous avons apporté des modifications au fichier test_main.py pour tester les differentes fonctionalités developpées dans le main.py

```python
# nous testons une valeur invalide de temperature, car la temperature chaude doit être suppérieure
def test_temperatureChaude_Invalide(self):
    with self.assertRaises(Exception):
        Main("f0c51c904ed6dd637b2f", 5, 33, 25)

# nous testons des valeurs invalides de temperature. chaud=froid
def test_egaliteTemperature(self):
    with self.assertRaises(Exception):
        Main("f0c51c904ed6dd637b2f", 5, 0, 0)
    moisecode [16 hours ago] • mise à jour des tests unitaires
# nous testons le nombre de tick invalide, il ne doit pas être inferieur à 0
def test_nbtick_Invalide(self):
    with self.assertRaises(Exception):
        Main("f0c51c904ed6dd637b2f", -4, 22, 56)
```

Déploiement et fonctionnement avec la structure de branches

Nous avons trois principales branches: features, develop et main. Chaque fonctionnalité du système est développée sur une branche develop. Chaque commit sur branche develop déclenche un pipeline de build jusqu'au déploiement dans docker hub. Une fois le déploiement réussi, nous pouvons faire un merge de la branche dev à la branche main qui a son tour déclenche un nouveau pipeline de build.

3. CONCLUSION

En somme , pour ce deuxième laboratoire nous avions eu pour mandat d'implémenter un pipeline build ce qui nous a permis de comprendre que l'implémentation du pipeline rend plus rapide l'exécution des tâches.

En outre, le laboratoire nous a permis de comprendre comment utiliser le Docker pour contenaliser une image à partir d'une image Alpine.