Déployer une application Python avec Kubernetes

Donner du sens à Kubernetes avec une application Python simple qui se déploie dans le service IBM Cloud Kubernetes

Par Kunal Malhotra

Date de publication : 25 septembre 2018

TOUT PUBLIC

Durée : 45mn

Ce tutoriel vous présente les différentes étapes pour déployer une application programmée en Python, avec Kubernetes sur IBM Cloud.

Un espace de discussion est disponible sur le forum pour recevoir vos avis sur ce tutoriel. **Commentez**

I - Introduction

Kubernetes est sorti depuis quelques années (sa version initiale a été lancée en juin 2014) et, bien que la communauté du codage soit au courant et ait eu vent de ses plus grandes capacités, de nombreuses personnes ne l'ont pas encore utilisé. Si vous n'utilisez pas encore Kubernetes, vous n'êtes pas seul.

D'après ce que j'ai entendu dire par quelques développeurs, une des raisons expliquant cette hésitation est le fait qu'il n'existe pas de guide approprié disponible (ce qui signifie qu'il y a une énorme quantité d'articles répartis sur plusieurs sites Web plutôt que dans un seul endroit) ou ils ont peur de commencer quelque chose de nouveau.

Dans ce tutoriel, mon objectif est de vous simplifier les choses en utilisant l'idée de base de créer une application Python avec Docker et de la déployer sur un service Kubernetes.

I-A - Objectifs d'apprentissage

Après avoir terminé ce tutoriel, vous pourrez conteneuriser une application Flask à l'aide de Docker et la déployer sur le service IBM Cloud Kubernetes.

I-B - Prérequis

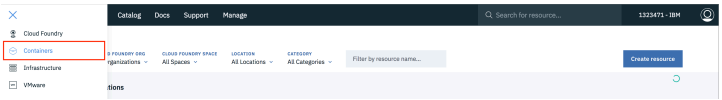
Afin de compléter ce tutoriel, vous devez disposer des prérequis suivants :

- Un compte IBM Cloud™ – (**inscription gratuite**)
- **CLI IBM Cloud**
- **CLI Docker**
- **CLI Kubernetes**

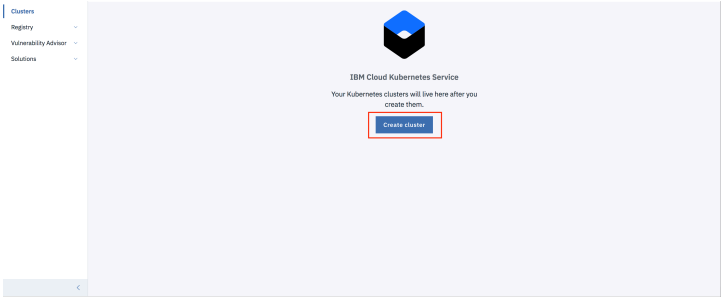
II - Différentes étapes du déploiement

II-A - Créer un cluster Kubernetes

- Connectez-vous à votre **Tableau de bord IBM Cloud**
- Ouvrez le service **« IBM Kubernetes »**

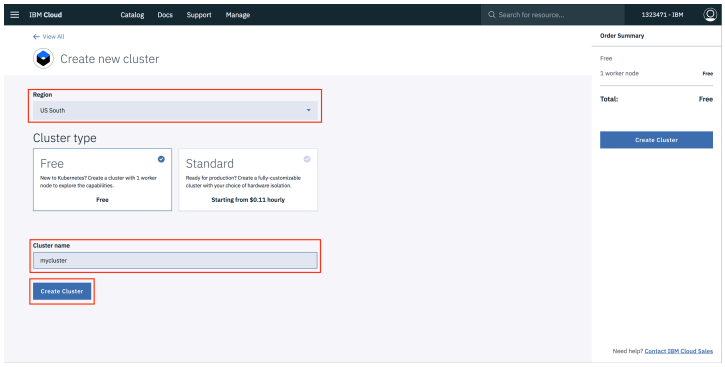


- Cliquez sur « **Create Cluster** »

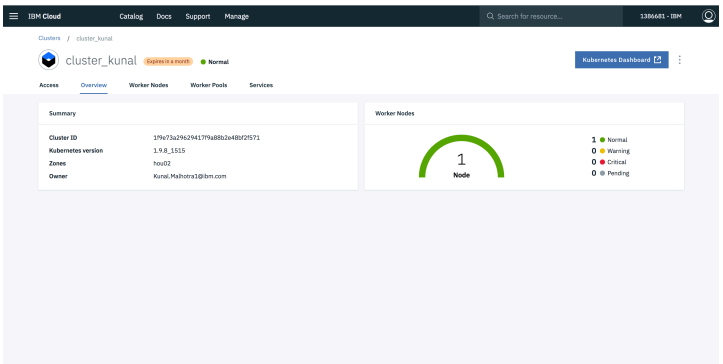


- Sélectionnez la Région dans laquelle vous souhaitez déployer le cluster, saisissez un nom pour votre cluster, et cliquez sur Créer Cluster.

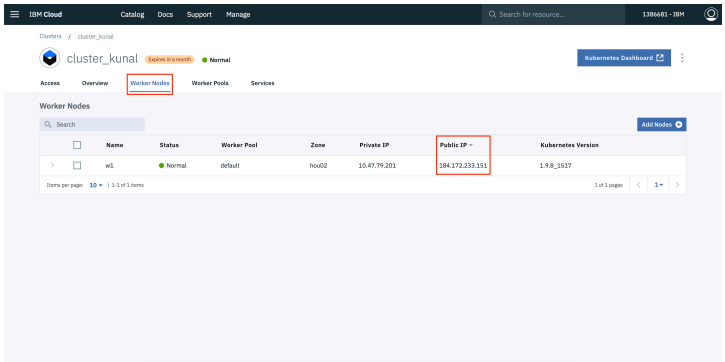
- En fonction de votre compte (**payant ou gratuit**), sélectionnez le type de cluster approprié.
- Le cluster prend du temps pour être prêt (environ 30 minutes).



- Dès que le cluster est prêt, cliquez sur le nom de votre cluster pour être redirigé vers une nouvelle page contenant toutes les informations concernant votre cluster et votre « worker node ».

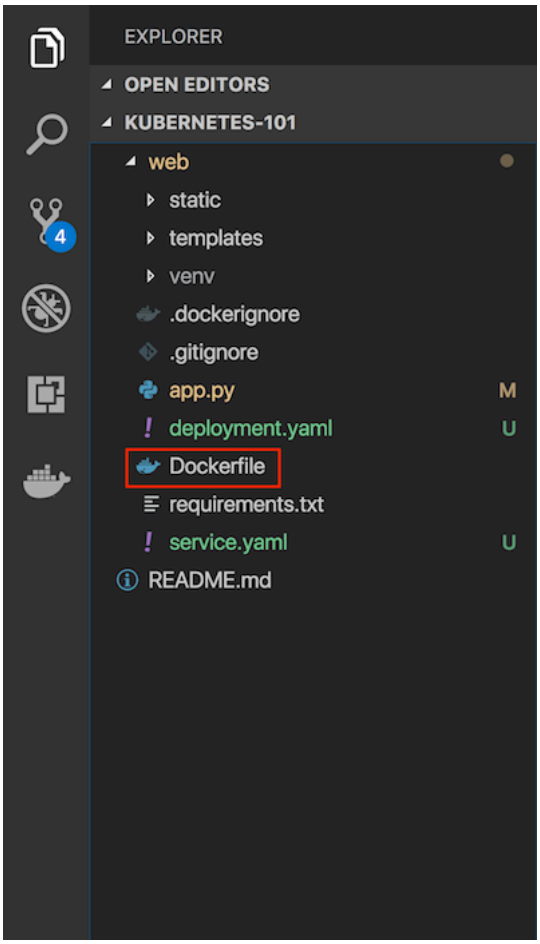


- Cliquez sur l'onglet « **Worker nodes** » pour noter l'adresse IP publique du cluster.



II-B - Conteneuriser votre application « Flask »

- Dans votre répertoire de projet, créez un fichier nommé « Dockerfile ».
  Suggestion : nommez exactement votre fichier « Dockerfile » rien d'autre.



Un « Dockerfile » est utilisé pour indiquer à Docker une image de base, les paramètres Docker dont vous avez besoin et une liste des commandes que vous souhaiteriez exécuter pour préparer et démarrer votre nouveau conteneur.

- Dans le fichier, collez ce code :

```
1. FROM python:2.7
2. LABEL maintainer="Kunal Malhotra, kunal.malhotra1@ibm.com"
3. RUN apt-get update
4. RUN mkdir /app
5. WORKDIR /app
6. COPY . /app
7. RUN pip install -r requirements.txt
8. EXPOSE 5000
9. ENTRYPOINT [ "python" ]
10. CMD [ "app.py" ]
```

II-B-1 - Explication et décomposition du code Dockerfile ci-dessus

1 La première partie du code ci-dessus est :

FROM python:2.7

Comme cette application Flask utilise Python 2.7, nous voulons un environnement qui le supporte et l'a déjà installé. Heureusement, DockerHub a une image officielle installée sur Ubuntu. Dans une ligne, nous aurons une image Ubuntu de base avec Python 2.7, virtualenv et pip. Il y a des tonnes d'images sur DockerHub, mais si vous souhaitez commencer avec une nouvelle image d'Ubuntu et la développer, vous pouvez le faire.

2 Regardons la partie suivante du code :

```
LABEL maintainer="Kunal Malhotra, kunal.malhotra1@ibm.com"
RUN apt-get update
```

3 Notez le maintainer et mettez à jour l'index du package Ubuntu. La commande est RUN, qui est une fonction qui exécute la commande suivante.

```
RUN mkdir /app
WORKDIR /app
COPY . /app
```

4 Il est maintenant temps d'ajouter l'application Flask à l'image. Pour que ça soit plus simple, copiez l'application dans le répertoire /app sur notre image Docker.

WORKDIRest essentiellement un équivalent de la commande cden bash, et COPY copie un répertoire donné dans le répertoire fourni dans une image. ADDest une autre commande qui fait la même chose que COPY, mais cela vous permet également d'ajouter un référentiel depuis une URL. Ainsi, si vous souhaitez cloner votre référentiel git au lieu de le copier depuis votre référentiel local (à des fins de transfert et de production), vous pouvez l'utiliser. COPY, cependant, devrait être utilisé la plupart du temps sauf si vous avez une URL.

5 Maintenant que notre référentiel est copié dans l'image, nous allons installer toutes nos dépendances qui sont définies dans la partie requirements.txt du code.

RUN pip install --no-cache-dir -r requirements.txt

6 Nous voulons exposer le port (5000) sur lequel s'exécute l'application Flask. Nous utilisons donc EXPOSE.

EXPOSE 5000

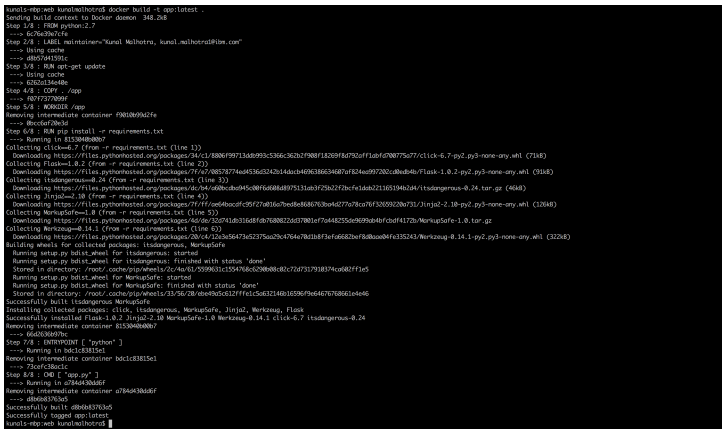
7 ENTRYPOINT spécifie le point d'entrée de votre application.

```
ENTRYPOINT [ "python" ]
      CMD [ "app.py" ]
```

II-C - Construire une image depuis le Dockerfile

Ouvrez le terminal et tapez cette commande pour créer une image à partir de votre fichier Dockerfile :

docker build -t <image_name>:<tag>

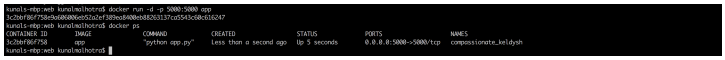


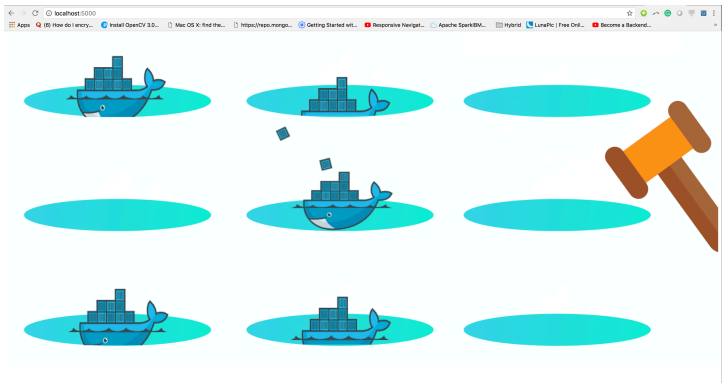
II-D - Exécutez votre conteneur localement et testez

Après avoir construit votre image avec succès, tapez :

docker run -d -p 5000:5000 app

>Cette commande va créer un conteneur contenant tout le code de l'application et les dépendances de l'image et exécute le code localement.

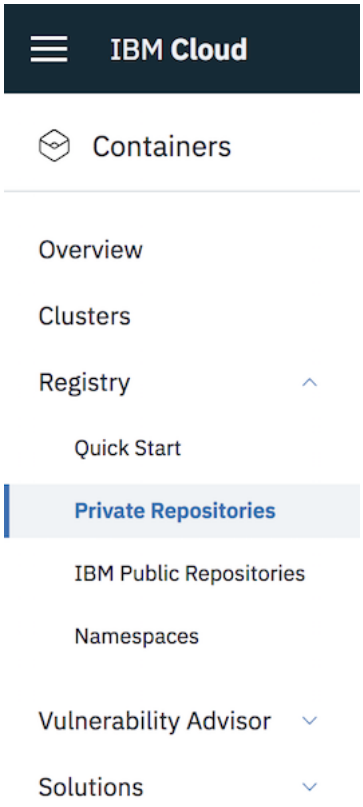




II-E - Poussez l'image vers le registre IBM Cloud

1 Depuis le tableau de bord de votre compte, accédez à « **IBM Cloud Kubernetes Service** ».

2 Dans le menu de navigation de gauche, sélectionnez « **Private Repositories** ».



3 Installez le plug-in Registre de conteneur

```
ibmcloud plugin install container-registry -r Bluemix
```

4 Connectez-vous à votre compte IBM Cloud.

```
ibmcloud login -a <cloud_foundary_end_point_for_the_region>
```

5 Nommez et créez votre namespace. Utilisez ce namespace pour le reste du Quick Start.

ibmcloud cr namespace-add <namespace>

6 Connectez votre daemon Docker local sur l'IBM Cloud Container Registry

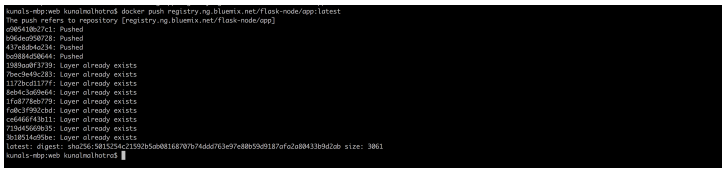
ibmcloud cr login

7 Choisissez un référentiel et une balise permettant d'identifier l'image.

docker tag <image_name> <region_url>/<namespace>/<image_name>:<tag>

8 Poussez l'image.

docker push <region_url>/<namespace>/<image_name>:<tag>

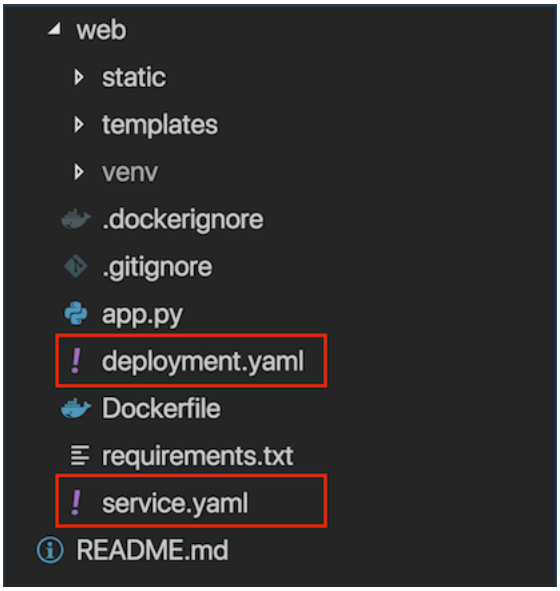


9 Vérifiez que votre image est dans votre registre privé.

ibmcloud cr image-list

II-F - Créer des fichiers de configuration pour Kubernetes

Une fois l'image téléchargée avec succès dans le registre privé, accédez au répertoire de votre projet et créez deux fichiers : **deployment.yaml** et **service.yaml**.



1 Dans le fichier **deployment.yaml**, collez ce code :

```
apiVersion: extensions/v1beta1
 kind: Deployment
 metadata:
   name: flask-node-deployment
```

```
  spec:
    replicas: 1
    selector:
      matchLabels:
        app: flasknode
    template:
      metadata:
        labels:
          app: flasknode
      spec:
        containers:
        - name: flasknode
          image: registry.ng.bluemix.net/flask-node/app
          imagePullPolicy: Always
          ports:
          - containerPort: 5000
```

2 Dans le fichier **service.yaml**, collez ce code :

```
apiVersion: v1
 kind: Service
 metadata:
   name: flask-node-deployment
 spec:
   ports:
   - port: 5000
     targetPort: 5000
   selector:
     app: flasknode
```

II-F-1 - Explication et décomposition du code deployment.yaml

1 Un déploiement nommé flask-node-deployment est créé, indiqué par le champ .metadata.name.
2 Le déploiement crée un pod répliqué, indiqué par le champ replicas.
3 Le champ sélecteur définit comment le déploiement détecte les modules à gérer. Dans le cas présent, nous sélectionnons simplement sur une étiquette définie dans le modèle Pod (app:flasknode). Cependant, des règles de sélection plus sophistiquées sont possibles, à condition que le modèle Pod lui-même satisfasse la règle.
4 La spécification du modèle de module, .template.spec, indique que les modules exécutent un conteneur, flasknode, qui exécute l'image de registre privée de l'application.
5 Le déploiement ouvre le port 5000 pour une utilisation par les modules.

II-F-2 - Explication et décomposition du code service.yaml

1 La spécification du service.yaml créera un nouvel objet de service nommé flask-node-deployment qui cible le port TCP 5000 sur tout pod avec le libellé « app = flasknode ». Ce service se verra également attribuer une adresse IP (parfois appelée le cluster IP), utilisée par les mandataires de service (voir ci-dessous). Le sélecteur du service sera évalué en continu et les résultats seront affichés sur un objet Endpoints également appelé flask-node-deployment.
2 Notez qu'un service peut mapper un port entrant sur un targetPod quelconque. Par défaut, le targetPort aura la même valeur que le champ du port. Un fait plus intéressant est que ce targetPort peut être une chaîne, en référence au nom d'un port dans les modules backend. Le numéro de port réel attribué à ce nom peut être différent dans chaque pod backend. Cela offre beaucoup de souplesse pour déployer et faire évoluer vos services. Par exemple, vous pouvez modifier le numéro de port que les modules exposent dans la prochaine version de votre logiciel de gestion, sans casser les clients.

II-G - Déployer votre application avec Kubernetes

1 Ciblez la région de service IBM Cloud Kubernetes où vous souhaitez travailler.

ibmcloud cs region-set us-south

2 Définissez le contexte du cluster dans votre CLI.

a Obtenez la commande pour définir la variable d'environnement et téléchargez les fichiers de configuration de Kubernetes.

ibmcloud cs cluster-config cluster_kunal

b Définissez la variable d'environnement KUBECONFIG. Copiez le résultat de la commande précédente et collez-le dans votre terminal. La sortie de commande doit ressembler à ce qui suit.

> export ***KUBECONFIG***=/Users/***$USER***/.bluemix/plugins/container-service/clusters/< cluster_name >/< cluster_configuration_file.yaml>

c Vérifiez que vous pouvez vous connecter à votre cluster en répertoriant vos noeuds de travail (worker nodes).

kubectl get nodes

d Créez le déploiement.

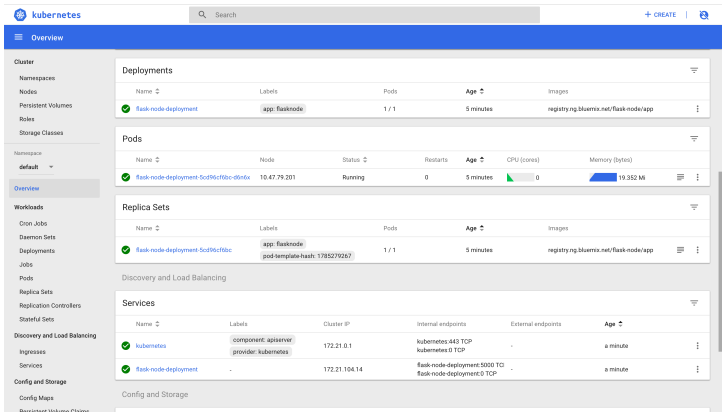
kubectl create -f deployment.yaml

```
Kunals-MacBook-Pro:web kunalmalhotra$ kubectl create -f service.yaml
service/flask-node-deployment created
```

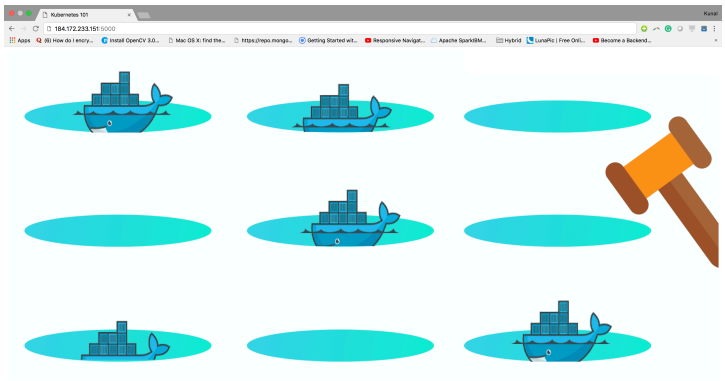
e Créez le service.

kubectl create -f service.yaml

```
Kunals-MacBook-Pro:web kunalmalhotra$ kubectl create -f deployment.yaml
deployment.extensions/flask-node-deployment created
```



f Enfin, accédez à votre navigateur et exécutez une commande ping sur l'adresse IP publique de votre nœud de travail (worker node)



III - Ressources et références

1.**Kubernetes Documentation**

2.**Deploy a microservices app on IBM Cloud by using Kubernetes**

3.**Tutorial: Deploying apps into clusters**

IV - Remerciements Developpez.com

Developpez.com remercie IBM pour la mise à disposition de ce tutoriel. Les remerciements également à **Guillaume Sigui** pour la mise au gabarit et **Bruno Barthel** pour la relecture orthographique.