



28 janvier 2021

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Automatisation du déploiement de Kubernetes sur EC2 de AWS</b>	<b>3</b>
2.1	KeyPairToFile() . . . . .	4
2.2	CreateSecurityGroup() . . . . .	4
2.3	createInstances() . . . . .	4
2.4	getPublicIpOfRunningInstances() . . . . .	4
2.5	execute_command_with_ssh() . . . . .	5
2.6	Implantation finale . . . . .	5
2.7	Difficultes . . . . .	5
<b>3</b>	<b>Automatisation du déploiement de WordCount avec Spark dans le cluster Kubernetes</b>	<b>6</b>
<b>4</b>	<b>Difficultés</b>	<b>7</b>
<b>5</b>	<b>Déploiement d'un outil de monitoring : Kube-Opex-Analytics</b>	<b>7</b>
<b>6</b>	<b>Participation de chaque membre du groupe :</b>	<b>9</b>

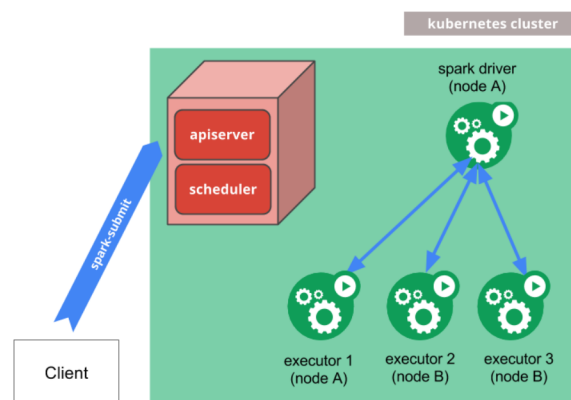
# 1 Introduction

Ce projet avait pour but d'automatiser le déploiement de Spark sur Kubernetes, sur un cluster de VM AWS, et d'utiliser un outil de monitoring sur Kubernetes.

Pour cela, nous nous sommes dans un premier temps occupés d'automatiser le déploiement d'un Kubernetes sur les machines EC2 de AWS.

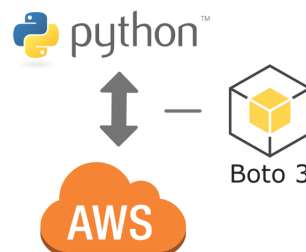
Nous nous sommes ensuite intéressés à l'automatisation du déploiement d'une application de Spark dans ce cluster Kubernetes.

Et nous avons finalement travaillé sur le déploiement d'un outil de monitoring afin d'évaluer l'utilisation du cluster Kubernetes.

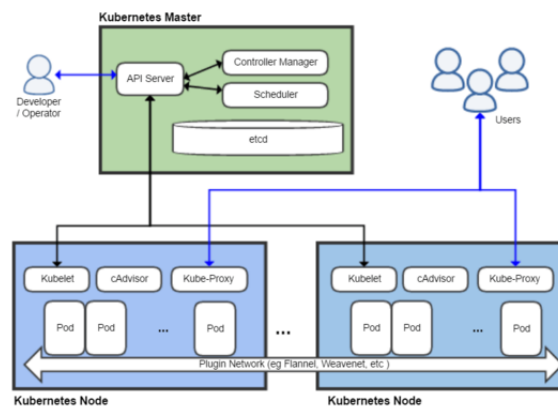


## 2 Automatisation du déploiement de Kubernetes sur EC2 de AWS

Nous avons utilisé le SDK de python Boto3 pour le développement des différentes parties. Pour réaliser le Cluster Kubernetes les 5 fonctions suivantes ont été implantées :



Architecture du Kubernetes :



## 2.1 KeyPairToFile()

Le but de cette fonction est de créer une clé dans AWS et de la stocker localement dans un fichier.

## 2.2 CreateSecurityGroup()

Cette fonction crée un groupe de sécurité dans AWS avec la permission des règles entrantes sur ports 80(HTTP), 22(SSH), 6443(Pour la commande kubectl join) et 5483(Pour Kube-Opex-Analytics).

## 2.3 createInstances()

Cette fonction crée des instances sur AWS avec la distribution la distribution qu'on choisi. Nous avons déployé des instances de type linux (gratuite), de type t3.small qui remplit les prérequis du déploiement d'un Cluster Kubernetes. Liée par la clé et le groupe de sécurité créés précédemment.

## 2.4 getPublicIpOfRunningInstances()

Cette fonction va retourner les adresses IP des instances en cours d'exécution sur AWS.

## 2.5 execute\_command\_with\_ssh()

Cette fonction a comme paramètre le nom de l'utilisateur, l'adresse IP de l'instance, ainsi que la commande qu'on veut exécuter sur l'instance. Elle permet de se connecter à nos instances pour exécuter la commande donnée dans les paramètres. elle retourne la dernière ligne du terminal de l'instance à laquelle on est connecté à travers SSH. On verra son utilité dans la partie Difficultés

## 2.6 Implantation finale

On crée la clé d'authentification qu'on stocke dans un fichier, le groupe de sécurité puis notre master et slaves. Ensuite, on envoie les fichiers de configurations master.conf et slave.conf dans l'instance du master et slaves respectivement afin d'y exécuter notre configuration shell.

## 2.7 Difficultés

1- Il fallait attendre la création des instances EC2 du master et du slave avant de continuer l'exécution du code. Pour remédier à cela, nous avons utilisé la fonction sleep de time en arrêtant l'exécution du code pendant 90 secondes. Cependant ce n'est pas la méthode la plus optimale en terme de temps d'exécution. On pouvait par exemple ajouter ces lignes de code suivant au lieu du sleep. On a opté pour le time.sleep parce que le code est plus lisible et permet de résoudre le même problème en une ligne.

```
clientec2 = boto3.client('ec2')
resp = clientec2.run_instances(ImageId = 'ami-0a0ad6b70e61be944',
                               InstanceType = 't2.micro',
                               MinCount = 1, MaxCount = 1,
                               KeyName = 'momo_keypair',
                               SecurityGroupIds=['sg-0a36ff444c898b8d5'],
                               SecurityGroups=['launch-wizard-3'])
instance_ids = []
for instance in resp['Instances']:
    instance_id = instance['InstanceId']
    instance_ids.append(instance_id)
```

```
ec2 = boto3.resource('ec2', region_name='us-east-2')
instance = ec2.Instance(id=instance_id)
instance.wait_until_running()
```

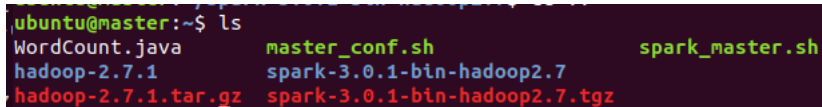
2- Pour réaliser les fonctions SSH et SCP il fallait modifier les droits du fichier contenant la clé d'authentification pour permettre à l'utilisateur root seulement d'accéder à la clé à travers la commande `chmod 400`.

3- Dans la partie de `execute_command_with_ssh()`, on a vu que notre fonction `ssh` retournait un string. Ce string s'agit de la `join command` (`kubeadm join`) qu'on va récupérer dans une variable puis on modifiera la dernière ligne de notre fichier de configuration du slave pour y ajouter cette commande pour que le slave puisse rejoindre le master.

4- Cette difficulté découle de la difficulté précédente, en effet qu'on on enregistre la `join command` on enregistre aussi les 'retailing spaces', on doit donc impérativement les retirer pour que notre commande `kubeadm join` puisse marcher.

### 3 Automatisation du déploiement de WordCount avec Spark dans le cluster Kubernetes

Pour réaliser cette partie, il fallait importer le fichier de SPARK, HADOOP ET JAVA en définissant les variables d'environnement de `JAVA_HOME`, `SPARKHOME` ET `HADOOPHOME`. Puis on compile `word-count` pour avoir un jar.



```
ubuntu@master:~$ ls
WordCount.java      master_conf.sh      spark_master.sh
hadoop-2.7.1         spark-3.0.1-bin-hadoop2.7
hadoop-2.7.1.tar.gz  spark-3.0.1-bin-hadoop2.7.tgz
```

On veille bien à mettre le `filesample.txt` dans le répertoire `data` et le `wc.jar` dans les jars. Puis on buildait une docker image à partir de notre répertoire `bin` et on définissait bien le service account et le `clusterrolebinding`. Une fois toutes ces étapes réalisées on exécute le `spark submit` en lui donnant l'adresse du cluster avec le port `6443`, le répertoire de l'image docker ainsi que son tag.

## 4 Difficultés

1) Malgré que toutes ces étapes aient été suivies, le pod exécutant le word count reste dans l'état Running et n'évolue pas à l'état attendu : Completed. L'erreur affichée est liée à une impossibilité d'instancier le SparkContext.

```
21/01/28 05:51:03 ERROR SparkContext: Error initializing SparkContext.  
org.apache.spark.SparkException: External scheduler cannot be instantiated
```

2) Au début on a fait face à une erreur concernant le chemin local au jar, on a pu le repérer grâce aux deux commandes `kubectl get pods` qui listaient les pods sur la VM, puis `kubectl logs IdPod`.

3) Les fichiers `.jar` et `samples.txt` étaient mis en dehors du docker spark cela donnait l'état Pending aux pods, et dès qu'on les a mis respectivement dans le dossier `/jars` et `/data` ils se sont mis dans l'état running.

4] Des erreurs liées au pare-feu du trafic entrant nous ont poussé à laisser rentrer tout le trafic.

## 5 Déploiement d'un outil de monitoring : Kube-Opex-Analytics

Nous devons dans cette partie, déployer un outil de monitoring sur le cluster Kubernetes : Kube-Opex-Analytics.

Bref, Kube-Opex-Analytics est un outil destiné à aider les organisations à suivre les ressources consommées par leurs clusters Kubernetes afin d'éviter les surpaiements. À cette fin, il génère des rapports d'utilisation à court, moyen et long terme montrant des informations pertinentes sur la quantité de ressources consommée par chaque projet au fil du temps. L'objectif final étant de faciliter les décisions d'allocation des coûts et de planification de la capacité grâce à des analyses factuelles.

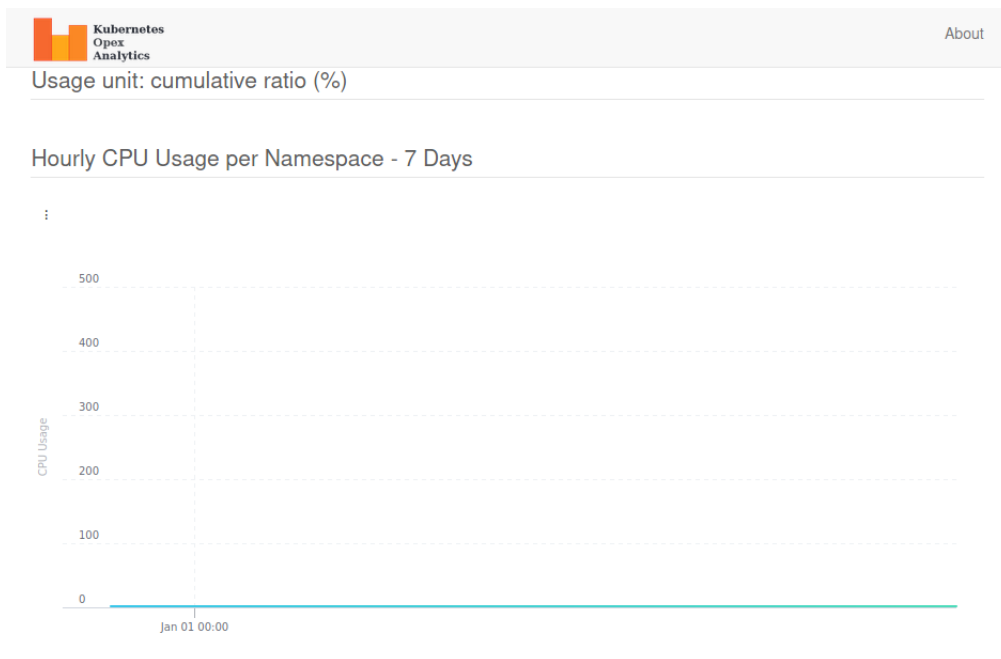
Pour automatiser le déploiement de cet outil, on a créé un script shell contenant deux commandes majeures. La première sert à avoir un accès en lecture seule aux API Kubernetes suivantes en utilisant Kubernetes Metrics Server :

- /api/v1
- /apis/metrics.k8s.io/v1beta1

La deuxième commande installe une instance de kube-opex-analytics déjà lancée par une image Docker.

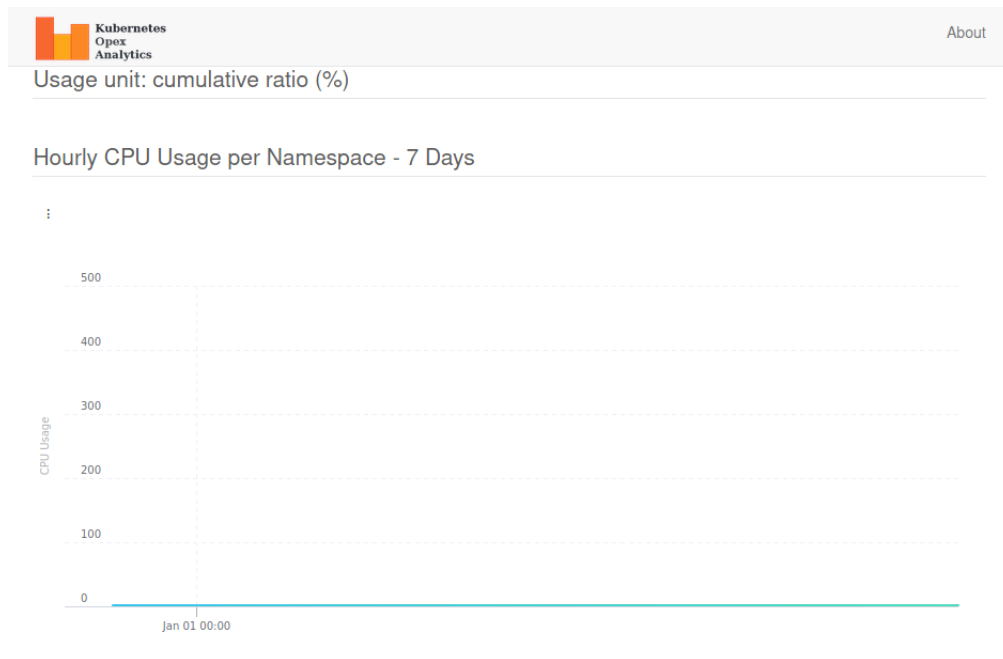
Afin de visualiser les résultats. On accède à l'interface graphique de l'outil via le port 5483.

Avant de lancer l'application WordCount sur spark dans le cluster Kubernetes. On voit que la consommation des ressources est nulle.



Après l'exécution de l'application WordCount :





Difficultés rencontrées : Pour cette section, On n'a pas pu accéder à l'interface graphique due à l'interdiction par le protocole HTTPS. Mais, On a pu résoudre ce problème en ajoutant une section dans la création du security group pour tout type de trafic.

## 6 Participation de chaque membre du groupe :

Pour l'automatisation du Kubernetes : Tout le monde a participé dessus.

Younes OSMAN et Mina ALLA : Déploiement du Spark et l'application WordCount.

Aymen BEN ABDELLAH et Gwendhal Borremans : Déploiement du Kube-Opex-Analytics.