



Projet Infrastructure

Infrastructure pour le Cloud et Big Data

Groupe I

Philippe Lopes
Jade DORCIER
Juliana OJO
Vincent Fuhrmann
Alexandre Fougères

Table des matières

Table des matières	2
Introduction	3
I. Automatisation du déploiement de Kubernetes sur un cluster de VM sur AWS	3
Déploiement des instances EC2	3
Installation de Kubernetes	5
II. Déploiement d'un outil de monitoring d'application dans Kubernetes (Kuber-opex-analytics)	5
III. Déploiement de Hadoop Spark et exécution de WordCount	7
Déploiement de Hadoop Spark	7
Exécution de WordCount	7
Problèmes rencontrés et Solutions proposées	9
Installation de K8 sur des VM AWS	9
Problème	10
Solution	10
CrashLoopBackOff	12
Problème	12
Solution	13
Utilisation des montages et des volumes avec helm charts	13
Problèmes	14
Solutions	14
Conclusion	15
Bibliography	15

Introduction

L'objectif de ce projet est de consolider nos connaissances acquises durant les cours d'infrastructure pour le Cloud et le Big Data et d'approfondir les sujets abordés pendant les travaux pratiques.

Il y a 3 étapes majeur à suivre pour réaliser ce projet :

1. Automatiser le déploiement de Kubernetes sur un cluster de VM sur AWS
2. Déployer un outil de monitoring d'application dans Kubernetes en utilisant kube-opex-analytics
3. Déployer Hadoop Spark sur le cluster Kubernetes et exécution de WordCount

I. Automatisation du déploiement de Kubernetes sur un cluster de VM sur AWS

Philippe - Jade

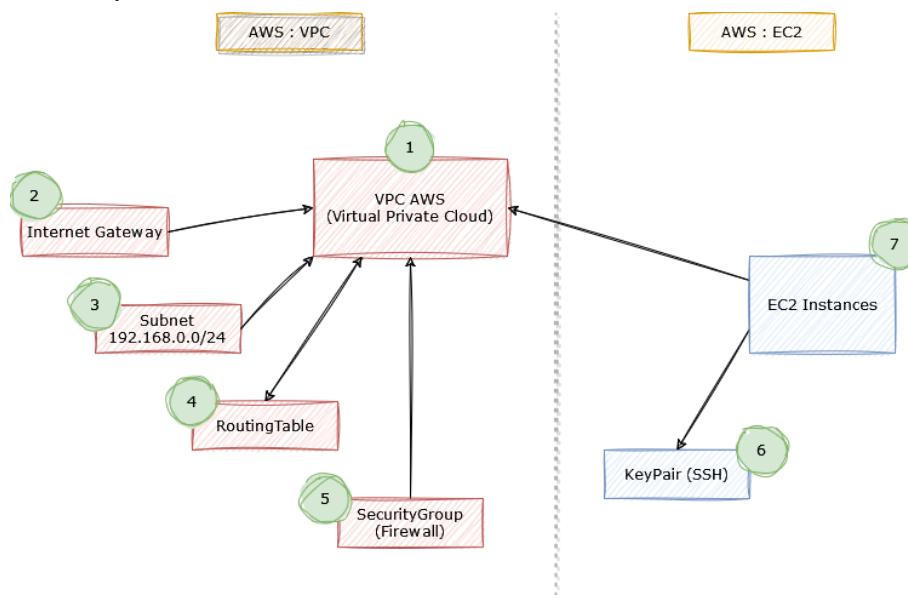
Le déploiement du cluster Kubernetes se fait en deux étapes :

- La création d'un ensemble d'instances EC2 AWS, qui feront office de serveurs. Ceux-ci seront contenus dans un VPC spécifique, afin de répondre à nos besoins.
- L'installation de Docker et Kubernetes sur chacune de ces instances, en attribuant l'une des instance, le rôle de master au sein du cluster K8s

Déploiement des instances EC2

Le déploiement des instances et du VPC associé se fait par l'intermédiaire d'un [script Python](#), qui s'occupera de configurer le tout et de récupérer en retour, les ID et adresses IP des objets créés.

Le script suit les étapes suivantes :



1. Il crée un objet VPC, sous le nom de “ProjetCloud-VPC”. Par défaut, cet objet est vide et ne possède aucune utilité. Ce VPC aura un attribut spécifique activé *EnableDnsHostnames*, qui permettra de fournir un nom de domaine aux futures instances.
2. Il crée un objet *InternetGateway* : “ProjetCloud-InternetGateway”, qu'il va ensuite le lier au VPC précédemment créé. Celui-ci sera utilisé dans le but de fournir un accès à Internet aux futures instances.
3. Il crée un objet *Subnet* : “ProjetCloud-Subnet”, qui va spécifier le sous-réseau utilisé au sein du VPC auquel on le lie. Sur cet objet, on active l'attribut *MapPublicIpOnLaunch*, qui permet de fournir une adresse IP publique à chaque nouvelle instance.
4. Il récupère l'objet *RoutingTable*, créé automatiquement par le VPC, il le renomme “ProjetCloud-RoutingTable, et ensuite créé une route statique par défaut vers l'adresse IP de l'objet *InternetGateway* précédemment créé. De ce fait, la redirection des flux vers Internet est possible.
5. Il crée ensuite un objet *SecurityGroup* : “ProjetCloud-SecurityGroup”, et le lie au VPC. Par défaut les flux sortant sont tous autorisés. En plus de ça, on crée une nouvelle règle qui autorise tous les flux **entrants** vers toutes les futures instances.
6. Il crée ensuite une paire de clé de sécurité “ProjetCloud-KeyPair” (de type RSA 2048, et en format .pem) qui sera utilisé pour accéder aux instances par SSH. Cette clé sera téléchargée localement.
7. Enfin, il crée un nombre d'instances EC2 (par défaut 3, pouvant être modifié en paramètre du script avec l'option -n). Avant de passer à la suite, il s'assure qu'elles sont créées et qu'elles sont en mode “running”.
8. Enfin il stocke toutes les informations dans un fichier *inventory.json*, qui sera utilisé pour les scripts suivants (notamment les adresses IP publiques). Le format ci-dessous :

```
{
  "VpcId": "vpc-xxx",
  "InternetGatewayId": "igw-xxx",
  "SubnetId": "subnet-xxx",
  "SecurityGroupId": "sg-xxx",
  "KeyPairPath": ".../ProjetCloud-KeyPair.pem",
  "Instances": [
    {
      "InstanceId": "i-xxx",
      "InstanceIp": "x.x.x.x"
    },
    {
      "InstanceId": "i-xxx",
      "InstanceIp": "x.x.x.x"
    }
  ]
}
```

Installation de Kubernetes

Une fois les instances EC2 créées, un deuxième [script Python](#) entre en jeu pour les configurer. Les étapes de configurations sont les suivantes :

1. A partir du fichier *inventory.json* précédemment créé, il considère la première instance de la liste comme étant le futur *master* du cluster Kubernetes. Les autres seront des *workers*. Il récupère ensuite le chemin vers la clé de sécurité téléchargée, qui servira à se connecter par SSH à toutes les instances, grâce à la librairie *paramiko*.
2. La première réelle étape consiste à installer Kubernetes sur toutes les instances par une série de commandes en shell (Installation de Docker, ajouts/modifications des fichiers de configurations nécessaires, installation de kubeadm et kubectl, redémarrage des services).
3. Il va ensuite configurer seulement l'instance *master*. Il initialise un cluster Kubernetes, crée un lien (que le script récupère pour la suite) afin que les *workers* puissent le rejoindre et installer une série de services nécessaire au bon fonctionnement et déploiement de pods (kube-flannel, Helm).
4. Il configure ensuite les *workers*, à savoir leur faire exécuter le lien précédemment récupéré, dans le but qu'ils rejoignent le cluster créé.
5. Il patiente une trentaine de secondes afin de laisser le temps aux *workers* de bien s'intégrer au cluster et affiche ensuite les *pods* et *nodes* présents au sein du cluster.

A noter, il est possible d'avoir un retour verbeux des commandes entrées en précisant -v (pour récupérer le flux de sortie des commandes) ou -vv (pour récupérer le flux de sortie ET d'erreurs).

II. Déploiement d'un outil de monitoring d'application dans Kubernetes **(Kuber-opex-analytics)** Philippe Vincent Jade

Dans cette partie, nous avons pour objectif de déployer Kuber-opex-analytics sur notre cluster Kubernetes. Nous avons choisi de l'implémenter via un chart helm. Nous avons choisi cette solution pour sa simplicité d'implémentation et sa scalabilité.

Cependant nous avons rencontré plusieurs problèmes dans cette mise en place, le helm chart fourni en documentation s'est avéré incomplet et donc difficile à implémenter sur notre cluster. En effet, nous avons dans un premier temps rencontré des problèmes liés au Persistent Volume de notre cluster, le pod ne se lançait pas car un persistent volume

claim était créé mais aucun persistent volume n'y était rattaché, nous avons ensuite remédié à cette solution en travaillant directement dans le pod et en éditant le statefulset (fichier de déploiement du pod) puis d'autres problème sont alors survenu, notamment le fait que le secret nécessaire au déploiement était manquant.

Ainsi suite à ces différents problèmes rencontrés qui se suivent en cascade nous n'avons pas réussi à déployer correctement l'application, et surtout à automatiser ce déploiement. En effet le fait de devoir modifier en direct le pod ou alors devoir reconstruire le déploiement en ajoutant des persistent volumes ou des secrets par exemple, sortait un peu du contexte de ce projet et représente une tâche relativement complexe. De plus cela rendait également la tâche de l'automatisation plus compliquée avec cette partie modification de pod.

Enfin nous avons également essayé de passer par le déploiement via kustomize également présent dans le répertoire GitHub, cependant lors du déploiement des erreurs au sein du code même de celui-ci était présente, nous ne nous sommes donc pas aventurés plus loin dans cette partie, pour les mêmes raisons abordées dans le déploiement via Helm.

III. Déploiement de Hadoop Spark et exécution de WordCount

Juliana · Jade · Vincent · Alexandre ·

Dans cette partie, nous avons pour objectif de déployer Spark sur notre cluster Kubernetes en mode cluster. Ce qui signifie que nous aurons un namenode et plusieurs datanodes (workers).

Ensuite, nous devons s'assurer du bon fonctionnement de notre déploiement en exécutant le programme WordCount qui compte chaque mot dans un fichier de texte quelconque.

Déploiement de Hadoop Spark

Au début, nous avons essayé d'écrire notre propre yaml de déploiement pour le maître Spark et le travailleur Spark, mais nous avons rapidement constaté que c'était une tâche fastidieuse. Nous avons décidé de faire quelque chose de plus simple : utiliser **spark helm charts**. Voici les étapes que nous avons suivies pour déployer les pods Spark dans notre cluster K8. Le code entier a été lancé sur le noeud maître.

1. Nous avons commencé par installer helm (sur K8)
2. Nous avons ajouté 'bitnami spark chart' à la liste des helm charts dans notre dépôt. Cela signifie que nous pouvons créer des clusters spark sans avoir à écrire des fichiers yaml de déploiement. L'image docker pour les conteneurs spark (exécuteurs) n'a pas besoin d'être construite ou poussée. Tout est décrit dans la charte spark.
3. L'étape suivante a été de créer notre cluster spark. Le cluster spark comprend un pod maître et deux pods travailleurs.
4. Pour nous assurer que tout est bien installé et fonctionne correctement, nous avons lancé une application de test appelée '**'spark-examples_2.12-3.3.1.jar'**'. Il peut être trouvé dans les pods par défaut, avec tous les jars nécessaires dans le paquetage spark.

L'application d'exemple a été lancée et s'est terminée avec succès, nous sommes donc passés à l'exécution de l'application WordCount.

Exécution de WordCount

Dans cette partie, notre objectif est de lancer un 'spark-submit' qui exécute l'application WordCount sur les exécuteurs. Un fichier texte spécifique est analysé et le résultat du comptage de mots est stocké de manière appropriée. Pour atteindre cet objectif, les étapes suivantes ont été suivies :

1. Le jar de l'application est stocké sur GitHub, nous devons donc d'abord télécharger le fichier (cette étape devrait déjà avoir été effectuée dans les sections précédentes).

2. Après avoir inspecté le jar pour s'assurer qu'il n'a pas été corrompu (en exécutant la commande '`jar tf wc.jar`'), nous copions le jar dans le worker pod. Cela permet de s'assurer que le fichier peut être lu et exécuté par le pod.

```
(base) raspberry@OJO tmp % git clone https://github.com/out3/projet-cloud.git
Clonage dans 'projet-cloud'...
remote: Enumerating objects: 116, done.
remote: Counting objects: 100% (116/116), done.
remote: Compressing objects: 100% (87/87), done.
remote: Total 116 (delta 40), reused 79 (delta 18), pack-reused 0
Réception d'objets: 100% (116/116), 79.12 Kio | 2.14 Mio/s, fait.
Résolution des deltas: 100% (40/40), fait.
(base) raspberry@OJO tmp % cd projet-cloud
(base) raspberry@OJO projet-cloud % ls
01-deploy-aws-infra 02-install-kubernetes Dockerfile README.md
ents.txt wc.jar
(base) raspberry@OJO projet-cloud % jar tf wc.jar
META-INF/MANIFEST.MF
wc/WordCount.class
.classpath
.project
(base) raspberry@OJO projet-cloud %
```

Figure III.2.1

3. L'étape suivante consiste à créer notre stockage PVC afin de pouvoir stocker le résultat de notre comptage de mots. Normalement, il suffit de créer un PVC et le plan de contrôle crée automatiquement un PV et lie le PV et le PVC ensemble. Afin de ne rien laisser au hasard, nous avons décidé de créer nous-mêmes le PV et le PVC et de leur donner les mêmes valeurs pour être sûrs qu'ils se lient ensemble.

(base) raspberry@OJO ~ % kubectl get pv kubectl get pvc							
NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS	REASON AGE
impvc	2Gi	ROX	Retain	Bound	default/impvc	standard	18m
my-vol	2Gi	ROX	Retain	Bound	default/my-vol	standard	45h
my-volume	2Gi	ROX	Retain	Available		pure-file	42h
myhostp	1Gi	RWO	Retain	Bound	default/myhostp	standard	25h
myvols	2Gi	ROX	Retain	Bound	default/myvols	pure-file	39h
pvc-257a672b-19fc-469a-abbe-6f9b385c4d88	2Gi	ROX	Delete	Bound	default/my-volume	standard	42h
pvc-737b813c-276a-4d10-8c3a-b693c83f1c50	1Gi	RWX	Delete	Bound	default/mypvc	standard	23h
NAME STATUS VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE			
impvc Bound impvc	2Gi	ROX	standard	17m			
my-vol Bound my-vol	2Gi	ROX	standard	45h			
my-volume Bound pvc-257a672b-19fc-469a-abbe-6f9b385c4d88	2Gi	ROX	standard	42h			
myhostp Bound myhostp	1Gi	RWO	standard	25h			
mypvc Bound pvc-737b813c-276a-4d10-8c3a-b693c83f1c50	1Gi	RWX	standard	23h			
myvols Bound myvols	2Gi	ROX	pure-file	39h			

Figure III.2.2

4. Nous avons ensuite lancé notre application Spark jar

```
(base) raspberry@OJO projet-cloud % kubectl exec -ti --namespace default ${SPARK_CLUSTER_NAME}-worker-0 -- spark-submit \
    --master spark://$SPARK_CLUSTER_NAME-master-svc:7077 \
    --class wc.WordCount \
    --conf spark.eventLog.enabled=true \
    --conf spark.eventLog.dir=/opt/bitnami/spark/tmp \
    --conf spark.kubernetes.driver.volumes.persistentVolumeClaim.mypvc.options.claimName=mypvc \
    --conf spark.kubernetes.driver.volumes.persistentVolumeClaim.mypvc.mount.path=/opt/bitnami/spark/tmp \
    --conf spark.kubernetes.executor.volumes.persistentVolumeClaim.mypvc.options.claimName=mypvc \
    --conf spark.kubernetes.executor.volumes.persistentVolumeClaim.mypvc.mount.path=/opt/bitnami/spark/tmp \
    [tmp/$MYAPP /opt/bitnami/spark/licenses/LICENSE-jsp-api.txt] \
23/01/23 11:18:51 INFO SparkContext: Running Spark version 3.3.1
23/01/23 11:18:51 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform...
... using builtin-java classes where applicable
23/01/23 11:18:52 INFO ResourceUtils: =====
=====
23/01/23 11:18:52 INFO ResourceUtils: No custom resources configured for spark.driver.
23/01/23 11:18:52 INFO ResourceUtils: =====
```

Figure III.2.3a

```

23/01/23 11:19:38 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
23/01/23 11:19:38 INFO DAGScheduler: ResultStage 1 (runJob at SparkHadoopWriter.scala:83) finished in 2.707 s
23/01/23 11:19:38 INFO DAGScheduler: Job 0 is finished. Cancelling potential speculative or zombie tasks for this job
23/01/23 11:19:38 INFO TaskSchedulerImpl: Killing all running tasks in stage 1: Stage finished
23/01/23 11:19:38 INFO DAGScheduler: Job 0 finished: runJob at SparkHadoopWriter.scala:83, took 27.332656 s
23/01/23 11:19:38 INFO SparkHadoopWriter: Start to commit write Job job_202301231119106181624181319533161_0005.
23/01/23 11:19:38 INFO SparkHadoopWriter: Write Job job_202301231119106181624181319533161_0005 committed. Elapsed time: 116 ms.
=====
time in ms :34253
23/01/23 11:19:38 INFO SparkUI: Stopped Spark web UI at http://test-spark-worker-0.test-spark-headless.default.svc.cluster.local:4040
23/01/23 11:19:38 INFO StandaloneSchedulerBackend: Shutting down all executors
23/01/23 11:19:38 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shutdown

```

Figure III.2.3b

Check out the WordCount.java file here ([WordCount.java](#))

5. Nous avons ensuite vérifié le résultat final

```
(base) raspberry@OJO projet-cloud % kubectl exec -ti --namespace default $SPARK_CLUSTER_NAME-worker-0 -- tail -n 10 /opt/bitnami/spark/tmp/result/part-00001
(always,1)
(Foundation,3)
(and,117)
(SOFTWARE,,1)
(grantor,.1)
(changing,1)
(employer,1)
(INACCURATE,1)
(being,1)
(compelled,1)
```

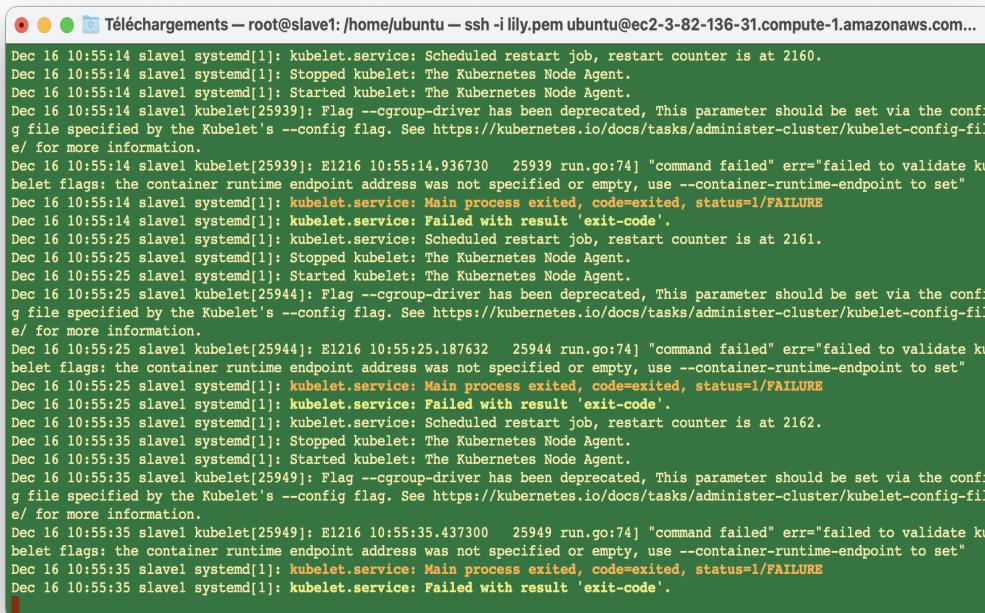
Figure III.2.4

L'étape finale consistait à intégrer le code dans le fichier Docker Compose ([Jade](#)).

Problèmes rencontrés et Solutions proposées

Installation de K8 sur des VM AWS

Avant de pouvoir commencer cette partie de notre projet qui implique le déploiement des pods spark et l'exécution de l'application Wordcount, nous avions besoin d'un cluster K8 entièrement fonctionnel pour tester et valider le code au fur et à mesure que nous le développions. Cela signifie que nous avons dû créer manuellement notre cluster de test et installer K8 nous-mêmes puisque le projet était divisé entre les membres. La première étape a été de créer 3 VMs AWS, de mettre à jour et d'installer les utilitaires



The screenshot shows a terminal window with the title "Téléchargements – root@slave1:/home/ubuntu – ssh -i lily.pem ubuntu@ec2-3-82-136-31.compute-1.amazonaws.com...". The window contains a log of errors from the kubelet service. The errors are repeated multiple times, indicating a continuous loop of restarts. Each error message is as follows:

```
Dec 16 10:55:14 slave1 systemd[1]: kubelet.service: Scheduled restart job, restart counter is at 2160.  
Dec 16 10:55:14 slave1 systemd[1]: Stopped kubelet: The Kubernetes Node Agent.  
Dec 16 10:55:14 slave1 systemd[1]: Started kubelet: The Kubernetes Node Agent.  
Dec 16 10:55:14 slave1 kubelet[25939]: Flag --cgroup-driver has been deprecated, This parameter should be set via the config file specified by the Kubelet's --config flag. See https://kubernetes.io/docs/tasks/administer-cluster/kubelet-config-file/ for more information.  
Dec 16 10:55:14 slave1 kubelet[25939]: E1216 10:55:14.936730 25939 run.go:74] "command failed" err="failed to validate kubelet flags: the container runtime endpoint address was not specified or empty, use --container-runtime-endpoint to set"  
Dec 16 10:55:14 slave1 systemd[1]: kubelet.service: Main process exited, code=exited, status=1/FAILURE  
Dec 16 10:55:14 slave1 kubelet[25939]: kubelet.service: Failed with result 'exit-code'.  
Dec 16 10:55:25 slave1 systemd[1]: kubelet.service: Scheduled restart job, restart counter is at 2161.  
Dec 16 10:55:25 slave1 systemd[1]: Stopped kubelet: The Kubernetes Node Agent.  
Dec 16 10:55:25 slave1 kubelet[25939]: Started kubelet: The Kubernetes Node Agent.  
Dec 16 10:55:25 slave1 kubelet[25944]: Flag --cgroup-driver has been deprecated, This parameter should be set via the config file specified by the Kubelet's --config flag. See https://kubernetes.io/docs/tasks/administer-cluster/kubelet-config-file/ for more information.  
Dec 16 10:55:25 slave1 kubelet[25944]: E1216 10:55:25.187632 25944 run.go:74] "command failed" err="failed to validate kubelet flags: the container runtime endpoint address was not specified or empty, use --container-runtime-endpoint to set"  
Dec 16 10:55:25 slave1 systemd[1]: kubelet.service: Main process exited, code=exited, status=1/FAILURE  
Dec 16 10:55:25 slave1 kubelet[25944]: kubelet.service: Failed with result 'exit-code'.  
Dec 16 10:55:35 slave1 systemd[1]: kubelet.service: Scheduled restart job, restart counter is at 2162.  
Dec 16 10:55:35 slave1 systemd[1]: Stopped kubelet: The Kubernetes Node Agent.  
Dec 16 10:55:35 slave1 kubelet[25944]: Started kubelet: The Kubernetes Node Agent.  
Dec 16 10:55:35 slave1 kubelet[25949]: Flag --cgroup-driver has been deprecated, This parameter should be set via the config file specified by the Kubelet's --config flag. See https://kubernetes.io/docs/tasks/administer-cluster/kubelet-config-file/ for more information.  
Dec 16 10:55:35 slave1 kubelet[25949]: E1216 10:55:35.437300 25949 run.go:74] "command failed" err="failed to validate kubelet flags: the container runtime endpoint address was not specified or empty, use --container-runtime-endpoint to set"  
Dec 16 10:55:35 slave1 systemd[1]: kubelet.service: Main process exited, code=exited, status=1/FAILURE  
Dec 16 10:55:35 slave1 kubelet[25949]: kubelet.service: Failed with result 'exit-code'.
```

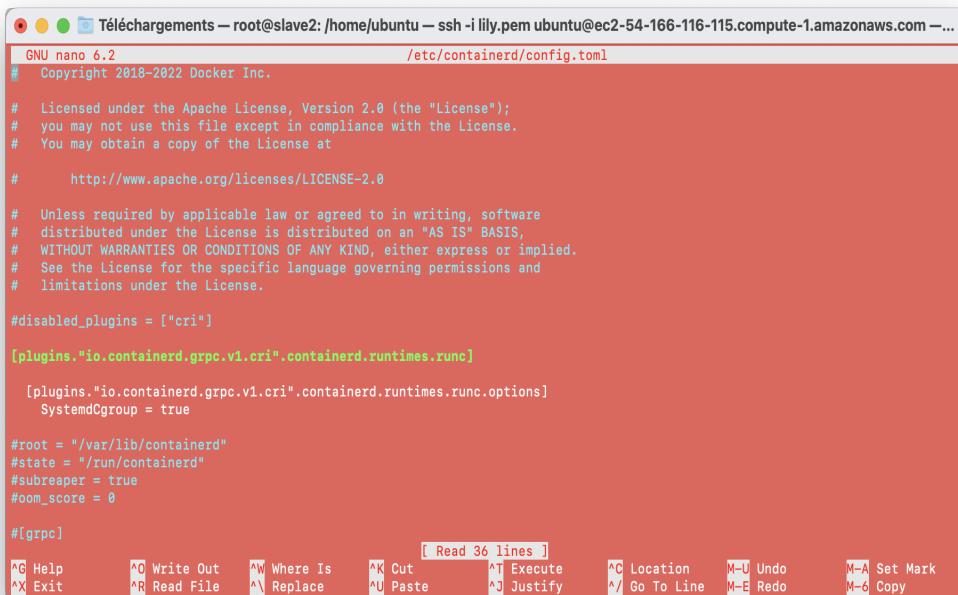
Figure III.3.1

Problème

Un problème est survenu lors de l'installation de K8 sur les 3 VMs. Lorsque nous avons essayé de joindre les nœuds de travail au nœud maître, nous avons obtenu des messages d'erreur (voir fig.III.3.1).

Solution

Après avoir cherché pendant des jours, nous avons découvert que le problème est lié au "cgroup". Lors de notre TP K8, nous avons pu spécifier le cgroup en utilisant le flag cgroup (--cgroup-driver=cgroupfs) lorsque nous modifions le fichier /etc/systemd/system/kubelet.service.d/10-kubeadm.conf. Il s'avère que cela ne fonctionne que sur l'ancienne version de K8 mais pas sur les dernières versions (elle



```

GNU nano 6.2                               /etc/containerd/config.toml
# Copyright 2018-2022 Docker Inc.

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at

#     http://www.apache.org/licenses/LICENSE-2.0

# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

#disabled_plugins = ["cri"]

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
SystemdCgroup = true

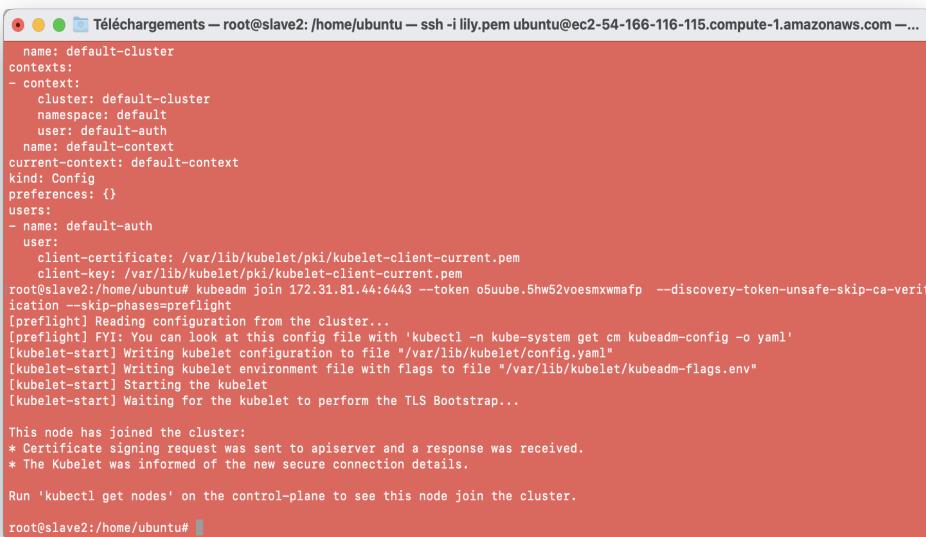
#root = "/var/lib/containerd"
#state = "/run/containerd"
#subreaper = true
#oom_score = 0

#[grpc]
[G Help      ^O Write Out   ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo   M-A Set Mark
[X Exit      ^R Read File   ^A Replace    ^U Paste      ^J Justify    ^L Go To Line M-E Redo   M-C Copy

```

Figure III.3.2

s'est dépréciée). La solution a été de modifier le fichier `/etc/containerd/config.toml` en ajoutant quelques lignes (comme indiqué dans la figure ci-dessus)



```

name: default-cluster
contexts:
- context:
  cluster: default-cluster
  namespace: default
  user: default-auth
  name: default-context
current-context: default-context
kind: Config
preferences: {}
users:
- name: default-auth
  user:
    client-certificate: /var/lib/kubelet/pki/kubelet-client-current.pem
    client-key: /var/lib/kubelet/pki/kubelet-client-current.pem
root@slave2:/home/ubuntu# kubeadm join 172.31.81.44:6443 --token o5uube.5hw52voesmxwmafp --discovery-token-unsafe-skip-ca-verification --skip-phases=preflight
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
root@slave2:/home/ubuntu#

```

Figure III.3.3

Après avoir résolu ce problème, la commande 'join' a fonctionné comme elle aurait dû le faire.

CrashLoopBackOff

Problème

Par défaut, la politique de redémarrage d'un pod est Always, ce qui signifie qu'il doit toujours redémarrer en cas d'échec. Selon la politique de redémarrage définie dans le modèle de pod, Kubernetes peut essayer de redémarrer le pod plusieurs fois.

À chaque fois que le pod est redémarré, Kubernetes attend un temps de plus en plus long, appelé "backoff delay". Pendant ce processus, Kubernetes affiche l'erreur CrashLoopBackOff.

Sur le nœud maître (VM), nous avons essayé de soumettre un exemple d'application Spark et le pod n'a cessé de se planter au moment où l'ordonnanceur devait partager les tâches entre les exécuteurs.

```
...@home:ubuntu ~ ssh -i lily.pem ubuntu@ec2-3-88-249-112.compute-1.amazonaws.com ...@home:ubuntu ~ ssh -i lily.pem ubuntu@ec2-62-23-208-112.compute-1.amazonaws.com ...@home:ubuntu ~ ssh -i lily.pem ubuntu@ec2-44-201-196-283.compute-1.amazonaws.com +
```

```
23/01/18 16:56:36 INFO BlockManager: Deleting directory /tmp/spark-d7a8139-5274-4608-aef8-56c0797ca79
command terminated normally, exit code: 10
root@master:/home/ubuntu# kubectl exec -ti --namespace default spark-cluster-worker-0 -- find examples/jars/ -name 'spark-example*.jar' | tr -d '\r'
root@master:/home/ubuntu# kubectl exec -ti --namespace default spark-cluster-worker-0 -- spark-submit --master spark://spark-cluster-master-svc:7077 --class org.apache.spark.examples.SparkPi $EXAMPLE_JAR 2
23/01/18 16:56:47 INFO SparkContext: Starting Spark Version 2.3.1
23/01/18 16:56:47 INFO SparkContext: Using Scala version 2.11.8
23/01/18 16:56:47 INFO SparkContext: Using Hadoop version 2.7.1
23/01/18 16:56:47 INFO ResourceUtils: *-----*
23/01/18 16:56:47 INFO ResourceUtils: No custom resources configured for spark.driver.
23/01/18 16:56:47 INFO ResourceUtils: *-----*
23/01/18 16:56:47 INFO ApplicationMaster: Submitted application: Spark Pi
23/01/18 16:56:47 INFO ResourceProfile: DefaultResourceProfile created, executor resources: Map[cores -> name: cores, amount: 1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offheap, amount: 0, script: , vendor: ], task resources: Map[cpus -> name: cpus, amount: 1.0)
23/01/18 16:56:47 INFO ResourceProfile: Limiting resource is cpus
23/01/18 16:56:47 INFO ResourceProfile: Registered ResourceProfile id: 0
23/01/18 16:56:48 INFO SecurityManager: Changing max acls to spark
23/01/18 16:56:48 INFO SecurityManager: Changing modify acls to: spark
23/01/18 16:56:48 INFO SecurityManager: Changing view acls groups to:
23/01/18 16:56:48 INFO SecurityManager: Changing modify acls groups to:
23/01/18 16:56:48 INFO SecurityManager: Authentication disabled; ui acls disabled; users with view permissions: Set(spark); groups with view permissions: Set(); users with modify permissions: Set(spark); groups with modify permissions: Set()
23/01/18 16:56:48 INFO Utils: Successfully started service 'SparkDriver' on port 46387.
23/01/18 16:56:48 INFO SparkEnv: Registering BlockManagerMaster
23/01/18 16:56:49 INFO SparkEnv: Registering BlockManagerMaster
23/01/18 16:56:49 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
23/01/18 16:56:49 INFO BlockManagerMasterEndpoint: up
23/01/18 16:56:49 INFO SparkEnv: Registering local disk provider at /tmp/blockmgr-e2b0afe-7a50-4c4f-8c99-ed18819a589c
23/01/18 16:56:49 INFO MemoryStore: Block store started with capacity 413.9 MiB
23/01/18 16:56:49 INFO SparkEnv: Registering OutputCommitCoordinator
23/01/18 16:56:50 INFO Utils: Successfully started service 'SparkUI' on port 4040.
23/01/18 16:56:50 INFO SparkContext: Added JAR file:/opt/binnam/spark/examples/jars/spark-examples_2.12-3.3.1.jar at spark://spark-cluster-worker-0.spark-cluster-headless.default.svc.cluster.local:46387/jars/spark-examples_2.12-3.3.1.jar
23/01/18 16:56:50 INFO StandaloneClientEndpoint: Connecting to master spark://spark-cluster-master-svc:7077...
23/01/18 16:56:50 INFO TransportClientFactory: Successfully created connection to spark-cluster-master-svc/19.106.32.73:7077 after 92 ms (0 ms spent in bootstraps)
23/01/18 16:56:51 INFO StandaloneAppClient: Connected to Spark cluster with app ID app-20230118165651-0000
23/01/18 16:56:51 INFO BlockManager: Successfully started service 'org.apache.spark.storage.BlockManagerService' on port 34615.
23/01/18 16:56:51 INFO NettyBlockTransferService: Service created on spark-cluster-worker-0.spark-cluster-headless.default.svc.cluster.local:34615
23/01/18 16:56:51 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
23/01/18 16:56:51 INFO StandaloneAppClientEndpoint: Executor added: app-20230118165651-0000/0 on worker-20230118165651-0000/0 with 1 core(s)
23/01/18 16:56:51 INFO BlockManagerMasterEndpoint: Registered block manager BlockManagerId(driver, spark-cluster-worker-0.spark-cluster-headless.default.svc.cluster.local, 34615, None)
23/01/18 16:56:51 INFO BlockManagerMasterEndpoint: Registering block manager BlockManagerId(driver, spark-cluster-worker-0.spark-cluster-headless.default.svc.cluster.local:34615 with 413.9 MiB RAW, BlockManagerId(driver, spark-cluster-worker-0.spark-cluster-headless.default.svc.cluster.local, 34615, None))
23/01/18 16:56:51 INFO BlockManager: Initiating BlockManager: BlockManagerId(driver, spark-cluster-worker-0.spark-cluster-headless.default.svc.cluster.local, 34615, None)
23/01/18 16:56:52 INFO StandaloneAppClientEndpoint: Executor updated: app-20230118165651-0000/0 is now RUNNING
23/01/18 16:56:52 INFO StandaloneSchedulerBackend: SchedulerBackend is ready for scheduling beginning after reached minRegisteredResourcesRatio: 0.0
23/01/18 16:56:54 INFO SparkContext: Starting job: reduce at SparkPi.scala:38
23/01/18 16:56:54 INFO DAGScheduler: Job 0 is ready to run (reduce at SparkPi.scala:38)
23/01/18 16:56:54 INFO DAGScheduler: Final stage: ResultStage 0 (reduce at SparkPi.scala:38)
23/01/18 16:56:54 INFO DAGScheduler: Parents of final stage: List()
23/01/18 16:56:54 INFO DAGScheduler: Missing parents: List()
23/01/18 16:56:54 INFO DAGScheduler: ResultStage 0 (reduce at SparkPi.scala:34) has no missing parents
23/01/18 16:56:54 INFO MemoryStore: Block broadcast_0 stored as values in memory (estimated size 4.0 KiB, free 413.9 MiB)
23/01/18 16:56:54 INFO MemoryStore: Block broadcast_0_piece0 stored as bytes in memory (estimated size 2.3 KiB, free 413.9 MiB)
23/01/18 16:56:54 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on spark-cluster-worker-0.spark-cluster-headless.default.svc.cluster.local:34615 (size: 2.3 KiB, free: 413.9 MiB)
23/01/18 16:56:54 INFO SparkContext: Created broadcast 0 from broadcast at DAGScheduler.scala:1513
23/01/18 16:56:54 INFO DAGScheduler: Submitting 2 missing tasks from ResultStage 0 (MapPartitionsRDD[1] at map at SparkPi.scala:34) (first 15 tasks are for partitions Vector(0, 1))
23/01/18 16:56:54 INFO TaskSchedulerImpl: Adding task set 0.0 with 2 tasks resource profile
```

Figure III.3.4

```

root@master:/home/ubuntu# kubectl get pods --all-namespaces
NAMESPACE     NAME           READY   STATUS        RESTARTS   AGE
default       spark-cluster-master-0   1/1    Running      0          5m28s
default       spark-cluster-worker-0   0/1    CrashLoopBackOff   68 (38s ago)   44h
default       spark-cluster-worker-1   0/1    CrashLoopBackOff   12 (58s ago)   29h
kube-flannel  kube-flannel-ds-89m5v  0/1    CrashLoopBackOff   139 (5s ago)   24d
kube-flannel  kube-flannel-ds-lbgjv  1/1    Running      0          33d
kube-flannel  kube-flannel-ds-n8ztn  1/1    Running      6 (6m30s ago)  24d
kube-system   coredns-787d4945fb-b79m4 1/1    Running      109 (87s ago)  11d
kube-system   coredns-787d4945fb-tbbw2 1/1    Terminating  0          33d
kube-system   coredns-787d4945fb-v55xv 1/1    Terminating  0          33d
kube-system   coredns-787d4945fb-vsmkj 0/1    CrashLoopBackOff  54 (17s ago)  44h
kube-system   etcd-ip-172-31-81-44   1/1    Running      0          33d
kube-system   kube-apiserver-ip-172-31-81-44 1/1    Running      0          33d
kube-system   kube-controller-manager-ip-172-31-81-44 1/1    Running      7 (24d ago)   33d
kube-system   kube-proxy-5vgqr   1/1    Running      6 (6m30s ago)  24d
kube-system   kube-proxy-94gt6   1/1    Running      141 (104s ago) 24d
kube-system   kube-proxy-phw5x   1/1    Running      0          33d
kube-system   kube-scheduler-ip-172-31-81-44 1/1    Running      9 (24d ago)   33d
root@master:/home/ubuntu# kubectl exec -ti --namespace default spark-cluster-worker-1 -- spark-submit --master spark://spark-cluster-master-svc:7077 --class org.apache.spark.examples.SparkPi $EXAMPLE_JAR 2>

```

Figure III.3.5

Solution

Nous n'avons jamais trouvé de solution à ce problème. Au début, nous avons pensé qu'il y avait un problème avec la commande '`spark-submit`' que nous avons exécutée mais ce n'était pas le cas car l'exemple d'application est utilisé pour tester si tout fonctionne comme il se doit. Nous avons soupçonné qu'il s'agissait d'un problème de '`dns`' car nous avons observé (voir la figure ci-dessus) que le pod DNS a également un problème de `CrashLoopBackOff`. Lorsque nous avons effectué le test pour savoir si le dns fonctionnait correctement, il a échoué à plusieurs reprises. Nous avons également recherché d'autres erreurs qui pourraient empêcher le déploiement de nos pods, cependant les events indiqués lors du déploiement du pod par kubernetes (event visible par la commande `kubectl describe pod <pod_name> -n <namespace>`) ne proposaient pas de réponse concrète connue comme par exemple une image docker qui ne se télécharge pas pourrait l'être, ici nous avions une erreur lié à la création du pod par Sandbox. A la fin de la journée, nous n'avons pas trouvé comment réparer ce problème et nous avons changé d'environnement de développement. Nous avons décidé de continuer à développer sur un ordinateur MacOS en utilisant **Minikube** comme un cluster K8.

Notre objectif est de nous assurer que notre code final est correct et fonctionne efficacement avant de réessayer en utilisant un autre cluster K8 sur AWS.

Utilisation des montages et des volumes avec helm charts

Nous avons souhaité monter un volume persistant sur les exécuteurs. Nous voulions que le volume soit accessible depuis la machine hôte ou le pod spark. Le volume serait utilisé comme un espace de stockage pour les résultats des applications spark exécutées et aussi un endroit où les exécuteurs peuvent obtenir le fichier texte sur lequel nous souhaitons effectuer un comptage de mots.

Problèmes

Le premier problème était de comprendre les volumes et les montages, ce qui était extrêmement difficile.

Le deuxième problème est qu'après avoir créé un volume persistant lié à une réclamation de volume persistant, copié le fichier texte et le jar de l'application dans le dossier correspondant dans le pod spark, exécuté le code suivant :

```
kubectl exec -ti --namespace default test-spark-worker-0 -- spark-submit \
--master spark://test-spark-master-svc:7077 \
--class wc.WordCount \
--conf spark.eventLog.enabled=true \
--conf spark.eventLog.dir=/opt/bitnami/spark/tmp \
--conf spark.kubernetes.driver.volumes.persistentVolumeClaim.mypvc.options.claimName=mypvc \
--conf
spark.kubernetes.driver.volumes.persistentVolumeClaim.mypvc.mount.path=/opt/bitnami/spark/tmp \
--conf spark.kubernetes.executor.volumes.persistentVolumeClaim.mypvc.options.claimName=mypvc \
\
--conf
spark.kubernetes.executor.volumes.persistentVolumeClaim.mypvc.mount.path=/opt/bitnami/spark/tmp \
\
tmp/wc.jar
```

Nous avons eu des erreurs parce que l'exécuteur ne pouvait pas trouver le fichier texte. Le chemin vers le fichier texte était bien indiqué dans le jar d'application mais il ne pouvait toujours pas être trouvé (voir fig.III.3.6).

```
23/01/21 08:43:35 INFO TaskSchedulerImpl: Stage 0 was cancelled
23/01/21 08:43:35 INFO DAGScheduler: ShuffleMapStage 0 (mapToPair at WordCount.java:26) failed in 23.664 s due to Job aborted due to stage failure: Task 0 in stage 0.0 failed 4 times, most recent failure: Lost task 0.3 in stage 0.0 (TID 4) (172.17.0.8 executor 0): java.io.FileNotFoundException: File file:/opt/bitnami/spark/tmp/filesample.txt does not exist
        at org.apache.hadoop.fs.RawLocalFileSystem.deprecatedGetFileStatus(RawLocalFileSystem.java:779)
        at org.apache.hadoop.fs.RawLocalFileSystem.getFileLinkStatusInternal(RawLocalFileSystem.java:1100)
        at org.apache.hadoop.fs.RawLocalFileSystem.getFileStatus(RawLocalFileSystem.java:769)
        at org.apache.hadoop.fs.FilterFileSystem.getStatus(FilterFileSystem.java:462)
        at org.apache.hadoop.fs.ChecksumFileSystem$ChecksumSInputChecker.<init>(ChecksumFileSystem.java:160)
        at org.apache.hadoop.fs.ChecksumFileSystem.open(ChecksumFileSystem.java:372)
        at org.apache.hadoop.fs.ChecksumFileSystem.lambda$openFileWithOptions$0(ChecksumFileSystem.java:896)
        at org.apache.hadoop.util.LambdaUtils.eval(LambdaUtils.java:52)
        at org.apache.hadoop.fs.ChecksumFileSystem.openFileWithOptions(ChecksumFileSystem.java:894)
        at org.apache.hadoop.fs.FileSystem$FSDataInputStreamBuilder.build(FileSystem.java:4768)
```

Figure III.3.6

Solutions

Pour le premier problème, nous l'avons facilement résolu en nous exerçant avec des exemples.

Pour le deuxième problème, nous n'avons pas trouvé de solution idéale. Nous avons essayé plusieurs choses comme l'ajout de l'option '`--files`' avec la commande '`spark-submit`' mais sans succès.

```
23/01/21 08:54:17 INFO SparkContext: Added JAR
file:/opt/bitnami/spark/tmp/word10.jar at
spark://test-spark-worker-0.test-spark-headless.default.svc.cluster.loc
al:42391/jars/word10.jar with timestamp 1674291251160
23/01/21 08:54:17 INFO SparkContext: Added file
file:///opt/bitnami/spark/tmp/filesample.txt at
spark://test-spark-worker-0.test-spark-headless.default.svc.cluster.loc
```

```
al:42391/files/filesample.txt with timestamp 1674291251160
23/01/21 08:54:17 INFO Utils: Copying
/opt/bitnami/spark/tmp/filesample.txt to
/tmp/spark-6a8c092a-a73d-4ae1-b749-7d0eb11e2630/userFiles-889c8a24-ac8c
-4b06-9b9a-32738e12964e/filesample.txt
```

Lorsque nous avons regardé de plus près les journaux d'événements, nous avons vu que le fichier texte a été copié et que son nom a été modifié. Nous avons donc modifié notre fichier d'application en conséquence, mais le fichier ne pouvait toujours pas être trouvé.

Nous n'avons pas trouvé de solution appropriée pour la version de Spark helm charts. Les seules que nous avons trouvées sont celles qui impliquent de créer nos propres fichiers de déploiement et de créer un conteneur spark personnalisé.

Nous avons décidé d'utiliser un fichier dont nous sommes sûrs à 100% qu'il se trouve dans les exécuteurs (conteneurs). Nous avons utilisé le fichier '**'NOTICE'**' du paquet Spark et le code a fonctionné (nous ajoutons le fichier text en tant qu'argument de l'application jar).

L'exécuteur a même été capable de sauvegarder le résultat de l'application Spark dans le volume partagé (mypvc) du pod Spark. Nous avons déduit que le montage a réussi mais nous n'avons pas pu trouver où le fichier texte doit être conservé pour qu'il soit vu et utilisé par les exécuteurs.

Conclusion

....

En fin de compte, nous avons dû coder en dur le chemin d'accès à un fichier qui existe dans les conteneurs d'exécution et nous ne pouvions pas exécuter l'application en mode cluster. Si nous avions eu plus de temps, nous aurions été en mesure de résoudre ces problèmes. Cela mis à part, nous avons quand même pu montrer qu'une application spark peut bien fonctionner sur K8. Nous avons également appris beaucoup de choses sur les graphiques, les volumes persistants et les réclamations.

Bibliography

- projet-cloud - <https://github.com/out3/projet-cloud>
- minikube start - <https://minikube.sigs.k8s.io/docs/start/>
- Running Spark on Kubernetes -
<https://spark.apache.org/docs/latest/running-on-kubernetes.html>
- charts/bitnami/spark/templates/ -
<https://github.com/bitnami/charts/tree/main/bitnami/spark/templates>

- Running spark on Kubernetes with persistent storage -
<https://jboothomas.medium.com/running-spark-on-kubernetes-with-persistent-storage-24b7903bb40a>