Rapport du projet Big Data

Le but du projet est de déployer un micro-service (application Node.js + MongoDB) à l'aide de *minikube* (*kubernetes*) et d'un provider *helm* à l'aide de *terraform*.

Nous expliquerons comment nous avons réalisé le déploiement de l'application puisque nous n'avons pas réussi à terminer le projet. Pour réaliser précisément ce que nous vous faire, nous avons cherché beaucoup d'informations sur Internet mais en vain. De plus, nous avons été confrontés à beaucoup d'erreurs. Le déploiement est une nouvelle notion pour nous, nous avons peu de compétences dans ce domaine et cela a été un frein pour nous. Nous espérons que vous prendrez cela en compte lors de la notation et nous vous remercions pour votre indulgence.

1) Initialisation de *minikube*

a) Problème d'installation

Nous avons tout d'abord initialisé notre minikube de cette façon :

```
namane@namane-VirtualBox:~/Bureau$ sudo -E minikube start --driver=none
😄  minikube v1.24.0 sur Ubuntu 21.10 (vbox/amd64)
✨  Utilisation du pilote none basé sur la configuration de l'utilisateur

🤷  Fermeture en raison de PROVIDER_NONE_NOT_FOUND : Le fournisseur 'none' n'a pas été trouvé : exec: "docker":
 executable file not found in $PATH
💡  Suggestion : Install docker
📘  Documentation: https://minikube.sigs.k8s.io/docs/reference/drivers/none/
```

Figure 01 : initialisation de minikube (1)

Comme nous pouvons le voir, nous avons un problème de fournisseur et de *Docker* (celui-ci n'est pas présent). Pour régler ce problème, nous avons effectués des recherches et nous avons lu de la documentation. La première des pistes a été d'installer les librairies suivantes :

- ca-certificates
- curl
- gnupg
- lsb-release

```
namane@namane-VirtualBox:~/Bureau$ sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

Figure 02 : installation des librairies

Puis nous avons installés Docker :

```
namane@namane-VirtualBox:~/Bureau$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Figure 03 : installation de Docker

b) Problème de cœur

Nous avons ensuite été confrontés à un second problème. Le nombre de cœur était inférieur à ce qui était demandé :

```
namane@namane-VirtualBox:~/Bureau$ sudo -E minikube start --driver=none
😄  minikube v1.24.0 sur Ubuntu 21.10 (vbox/amd64)
✨  Utilisation du pilote none basé sur la configuration de l'utilisateur

⛔  Fermeture en raison de RSRC_INSUFFICIENT_CORES : Le nombre de processeurs demandés 2 est supérieur au nombr
e de processeurs disponibles de 1
```

Figure 04 : initialisation de minikube (2)

Après cela, nous avons utilisés une commande retrouvée sur internet afin de modifier la configuration de notre minikube :

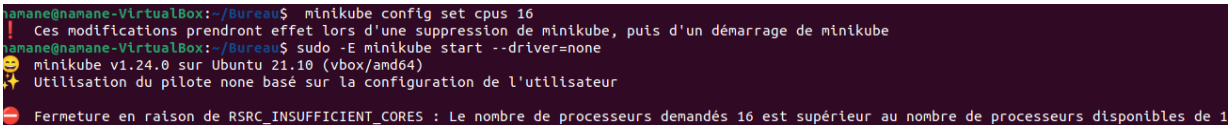

Figure 05 : modification de la configuration minikube

Nous avons réglé cela en modifiant les paramètres de notre VM et en passant le nombre de processeur à 2.

c) Problème utilisateur

Ensuite nous avons eus une autre erreur :

Figure 06 : initialisation de minikube (3)

Cette fois, c'est un problème de permission ! Nous avons donc exécuté la commande suggérée :

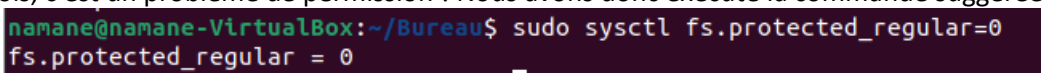

Figure 07 : changement des privilèges

d) Utilisation du driver *Docker*

Les erreurs n'ont fait que de s'ajouter, nous avons donc pris la décision de lancer le minikube dans un docker, de cette façon :

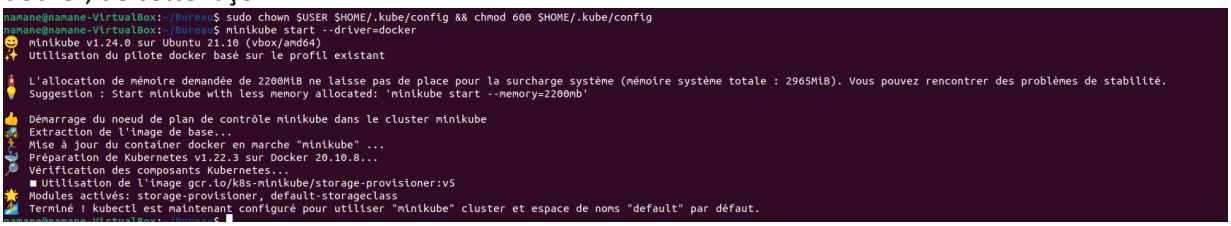

Figure 08 : initialisation de minikube avec Docker

Et cela a fonctionné :

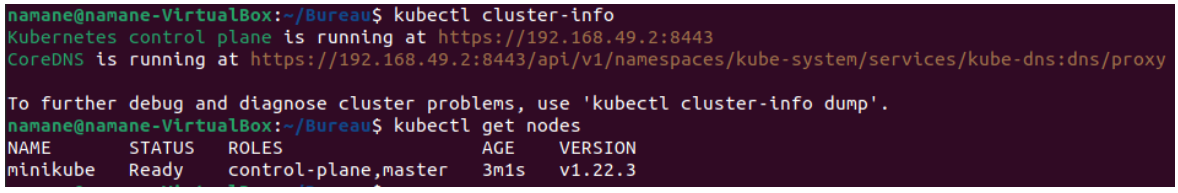

Figure 09 : test du bon fonctionnement

2) Initialisation de *terraform*

a) Installation de *tap* puis de *terraform*

Nous avons ensuite installé *tap*, nous permettant ensuite d'installer *terraform* :

```
namane@namane-VirtualBox:~/Bureau$ brew tap hashicorp/tap
==> Tapping hashicorp/tap
Cloning into '/home/namane/Bureau/homebrew/Library/Taps/hashicorp/homebrew-tap'.
remote: Enumerating objects: 1777, done.
remote: Counting objects: 100% (531/531), done.
remote: Compressing objects: 100% (233/233), done.
remote: Total 1777 (delta 363), reused 450 (delta 298), pack-reused 1246
Receiving objects: 100% (1777/1777), 319.57 KiB | 1.97 MiB/s, done.
Resolving deltas: 100% (1059/1059), done.
Tapped 1 cask and 15 formulae (47 files, 540.3KB).
```

Figure 10 : installation de tap (1)

```
namane@namane-VirtualBox:~/Bureau$ brew install hashicorp/tap/terraform
==> Downloading https://releases.hashicorp.com/terraform/1.1.2/terraform_1.1.2_linux_amd64.zip
######################################################################## 100.0%
==> Installing terraform from hashicorp/tap
🍺  /home/namane/Bureau/homebrew/Cellar/terraform/1.1.2: 3 files, 59MB, built in 4 seconds
==> Running `brew cleanup terraform`...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
namane@namane-VirtualBox:~/Bureau$ brew update
Already up-to-date.
namane@namane-VirtualBox:~/Bureau$ brew upgrade hashicorp/tap/terraform
Warning: hashicorp/tap/terraform 1.1.2 already installed
```

Figure 11 : installation de tap (2)

Pour prouver la bonne installation de terraform, nous avons exécuté la commande suivante :

```
namane@namane-VirtualBox:~/Bureau$ terraform -help
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
  plan          Show changes required by the current configuration
  apply         Create or update infrastructure
  destroy       Destroy previously-created infrastructure

All other commands:
  console       Try Terraform expressions at an interactive command prompt
  fmt           Reformat your configuration in the standard style
  force-unlock  Release a stuck lock on the current workspace
```

Figure 12 : test de la bonne installation de terraform

Il est bien installé !

b) Initialisation de *terraform*

Nous avons créés un docker-compose pour déployer notre micro-service :

- Déclaration des ports
- Appel de l'image
- Définition du network
- Définition du volume

A partir du docker-compose on génère une charte *helm* permettant de déployer notre application sur un cluster *minikube* :

```
namane@namane-VirtualBox:~/charts$ kompose convert -c
WARN Volume mount on the host "/home/namane/charts/services/api" isn't supported - ignoring path on the host
WARN Volume mount on the host "/home/namane/charts/services/mongo/data" isn't supported - ignoring path on the host
INFO Kubernetes file "docker-compose/templates/api-service.yaml" created
INFO Kubernetes file "docker-compose/templates/mongo-service.yaml" created
INFO Kubernetes file "docker-compose/templates/api-deployment.yaml" created
INFO Kubernetes file "docker-compose/templates/dev-env-configmap.yaml" created
INFO Kubernetes file "docker-compose/templates/api-claim0-persistentvolumeclaim.yaml" created
INFO Kubernetes file "docker-compose/templates/mongo-deployment.yaml" created
INFO Kubernetes file "docker-compose/templates/mongo-claim0-persistentvolumeclaim.yaml" created
INFO chart created in "docker-compose/"
```

Figure 13 : génération des fichier yaml

Nous pouvons voir que différents fichiers sont créés comme :

- api-service.yaml
- mongo-service.yaml
- api-deployment.yaml...
-

Une fois notre kompose de fait, un fichier chart.yaml permettant de déployer l'application a été créé. Nous avons ensuite créé notre provider Helm à partir de terraform en renseignant la charte Helm précédemment créée :

```
provider "helm" {
  kubernetes {
    config_path = "~/.kube/config"
  }

}

resource "helm_release" "application"{
      name  = "docker-compose"
      chart ="./charts/docker-compose"

}
```

Figure 14 : création du provider

Nous avons ensuite initialisé notre terraform :

```
namane@namane-VirtualBox:~$ terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/helm from the dependency lock file
- Using previously-installed hashicorp/helm v2.4.1

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Figure 15 : initialisation de terraform

Une fois cela fait, nous avons créés un plan d'exécution, qui nous permet de prévisualiser les changements que terraform prévoit d'apporter à notre infrastructure, que voici :

```
namane@namane-VirtualBox:~$ terraform plan
helm_release.application: Refreshing state... [id=docker-compose]

Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since the last "terraform apply":

  # helm_release.application has changed
  ~ resource "helm_release" "application" {
      id                         = "docker-compose"
    ~ metadata                   = [
        ~ {
            ~ values       = jsonencode(
                ~ {} -> null
              )
              # (6 unchanged elements hidden)
          },
      ]
      name                       = "docker-compose"
      # (24 unchanged attributes hidden)
  }


Unless you have made equivalent changes to your configuration, or ignored the relevant attributes using ignore_changes, the following plan may
include actions to undo or respond to these changes.
```

Figure 16 : planification de terraform

Puis nous avons exécuté les actions proposées dans un plan Terraform à l'aide de la commande suivante :

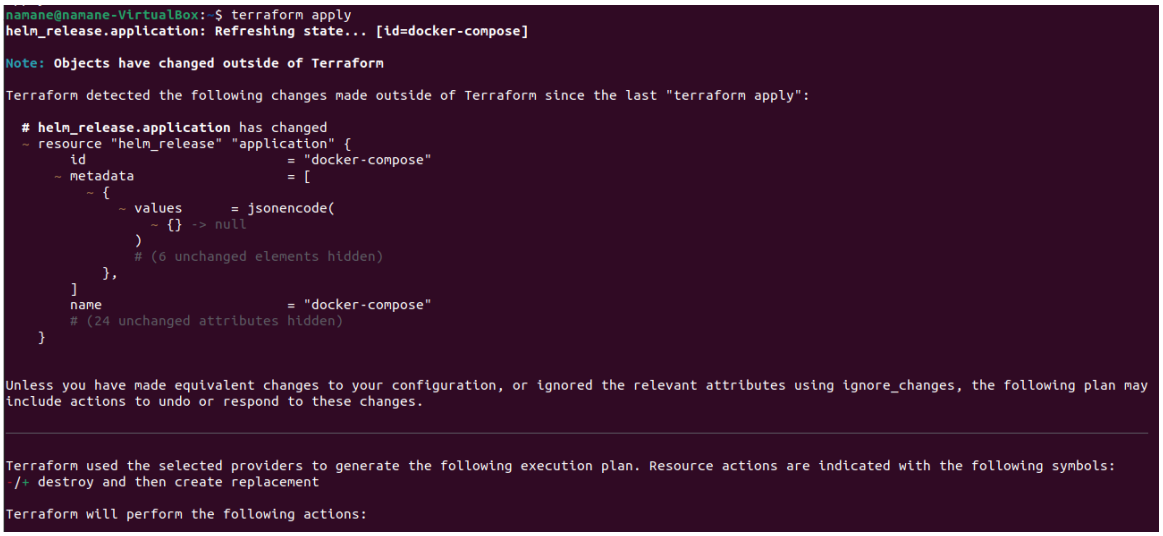

Figure 17 : lancement du terraform