

**Conteneurisation des Applications**

**Encadré par :**

Driss ALLAKI

**Réalisé par :**

Abir HSAINI

Contents

[***Partie 1 :*** 3](#_Toc119760311)

[1. Créer un conteneur (nommé mongodb-service) pour la base de données MongoDB en instanciant l’image officielle mongo : 3](#_Toc119760312)

[2. Créer un réseau Docker (tout en lui précisant un driver convenable). Puis Connecter le conteneur mongodb avec ce réseau : 4](#_Toc119760313)

[3. Créer un fichier Dockerfile pour le projet Backend : 5](#_Toc119760314)

[4. Enumérer et expliquer "en détails" les bonnes pratiques utilisées pour l’écriture de ce fichier Dockerfile : 5](#_Toc119760315)

[5. Taper les commandes docker nécessaires pour : 6](#_Toc119760316)

Considérons une application web 3-tiers (composée de frontend, backend et base de données) développée avec le MERN stack (MongoDB, Express, React, Node.js). Le code source de cette application est disponible sur ces repos : frontend et backend.

# ***Partie 1 :***

Dans cette partie, il est demandé de réaliser les manipulations suivantes en utilisant un outil de conteneurisation comme Docker.

## 1. Créer un conteneur (nommé mongodb-service) pour la base de données MongoDB en instanciant l’image officielle mongo :

Tout d’abord j’ai essayé de créer un volume avec le nom backendVolume. J’ai utilisé les volumes afin de rendre le stockage de cette base de données persistant, car ces derniers sont stockés dans une partie du système de fichiers hôte géré par Docker (/var/lib/docker/volumes/ sous Linux). Les processus non Docker ne doivent pas modifier cette partie du système de fichiers. Les volumes sont le meilleur moyen de conserver les données dans Docker.

Apres on crée le conteneur en lui associons ce volume par la méthode ci-dessus :

On ouvre docker desktop on voit que notre conteneur existe

## 2. Créer un réseau Docker (tout en lui précisant un driver convenable). Puis Connecter le conteneur mongodb avec ce réseau :

On crée un réseau bridge car nous serons besoin de communiquer les conteneurs puisqu’ils sont autonomes.

On utilise cette commande :

docker network create --driver bridge --subnet 192.168.137.218/24 mongoNetwork

Après on le connecte a mongodb

Pour s’assurer que la connexion est faite on lance un inspect sur notre conteneur

On voit que la connexion est faite :

## 3. Créer un fichier Dockerfile pour le projet Backend :

Voici le dockerfile

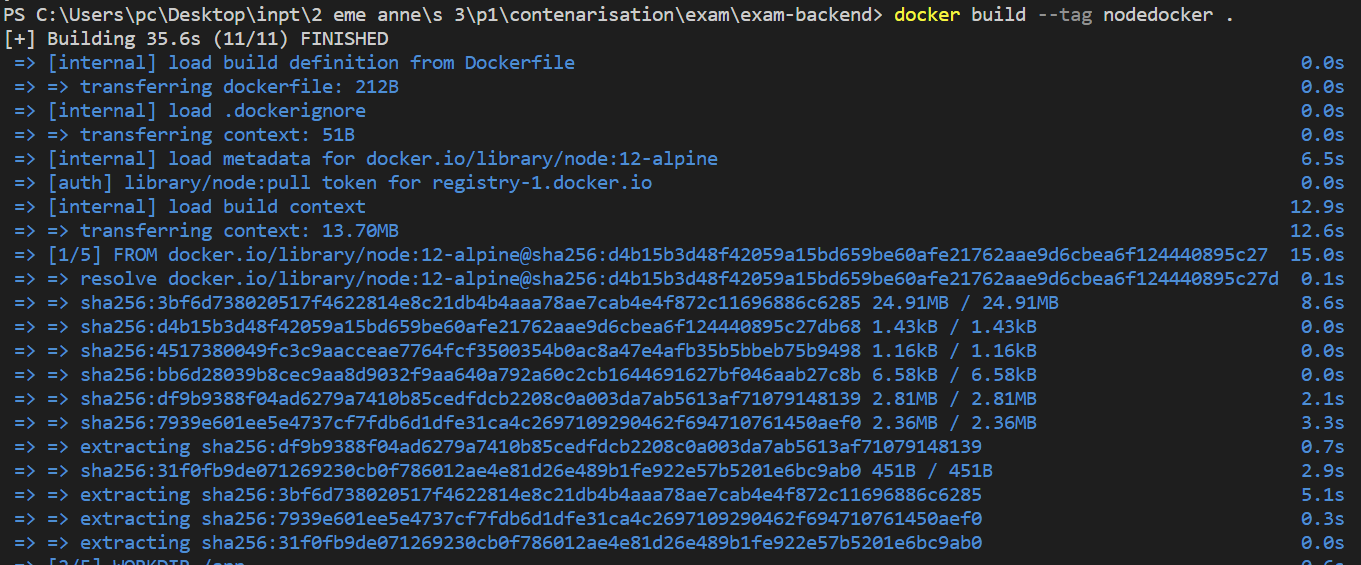
## 4. Enumérer et expliquer "en détails" les bonnes pratiques utilisées pour l’écriture de ce fichier Dockerfile :

* La première chose nous avons pris la version minimale de l’image afin d’éviter les erreurs de compatibilité ainsi que cette image est oficielle .
* J’ai utilisé dockerignore afin d’eviter de copier nodemodules
* Afin d’optimiser le temps nous essayons tout d’abord de copier les packages necessaires , s’il n’y a pas de changement on aura pas besoin de copier le tous.
* Optimisation de la cache de l’image

## 5. Taper les commandes docker nécessaires pour :

1. **Créer une image docker en local à partir de ce Dockerfile**:

On lance la commande suivante :



Pour –tag : il nous permet de donner un tag à notre image

Si nous ouvrons docker desktop nous allons trouver l’image :



1. **Scanner l’image des vulnérabilités qu’elle peut contenir.**

On lance la commande ci-dessus :

1. **Publier cette image dans Docker Hub.**

On push notre image dans dockerhub, on lance d’abord la commande qui va nous permettre de mettre un tag à notre image

docker tag nodedocker abirhsaini/exam:nodedocker

après on lance la commande de push

1. **Instancier cette image en créant un conteneur (nommé backend) s’exécutant en local et partageant le même réseau avec le conteneur de la base de données mongodb.**

On lance la commande ci-dessus :

-itd : demande à Docker d'allouer un pseudo-TTY connecté au stdin du conteneur ; créer un Shell Bash interactif dans le conteneur. Ainsi que d nous donne l’accès au Bash mode détachée

-p : pour spécifier le port.

--name : pour nommer notre conteneur.

--network : pour le connecter à un network spécifique.

Apres on voit que notre conteneur s’exécute correctement dans docker desktop

1. **Inspecter ce conteneur backend.**
2. **Afficher les logs liés à ce conteneur backend**

## **6. Refaire le travail demandé dans les questions 3), 4) et 5) pour le projet Frontend :**

On crée le dockefile :

Pour les bons pratiques , ce sont les mêmes pour le backend , maintenant on essaye de Builder notre image

Maintenant on essaye de la scanner

On la push dans dockerhub

Maintenant on cherche a créer notre conteneur

On voit que le conteneur est là

On lance docker logs frontend

## **7. S’assurer que l’application a été bien conteneurisée et qu’elle fonctionne correctement. (Prendre des prises d’écran du navigateur) :**

on ouvre notre navigateur est on tape url : <http://localhost:5000/>

on aura la page suivante :

On tappe maintenant : <http://localhost:3000/> on aura la page suivante

On tappe maintenant : <http://localhost:27017/> on aura la page suivante

Ce qui veut dire que notre application fonctionne correctement

## **8. Supprimer les 3 conteneurs en exécution :**

On lance les deux commande l’une pour le stopper et l’autre pour le supprimer

## 9. Redéployer l’application (frontend, backend et base de données) en utilisant docker-compose.

1. **Créer le fichier docker-compose :**

On cree le fichier deocker compose :

1. **Taper la commande docker permettant de l’exécuter.**

**c. S’assurer que l’application fonctionne correctement (prises d’écran du**

**navigateur)**

# 10.Supprimer les conteneurs qui s’exécutent ainsi que les images se trouvant dans le docker host.

Les images :

## 11.Créer un registre privé d’images docker en local qui vous permettra de :

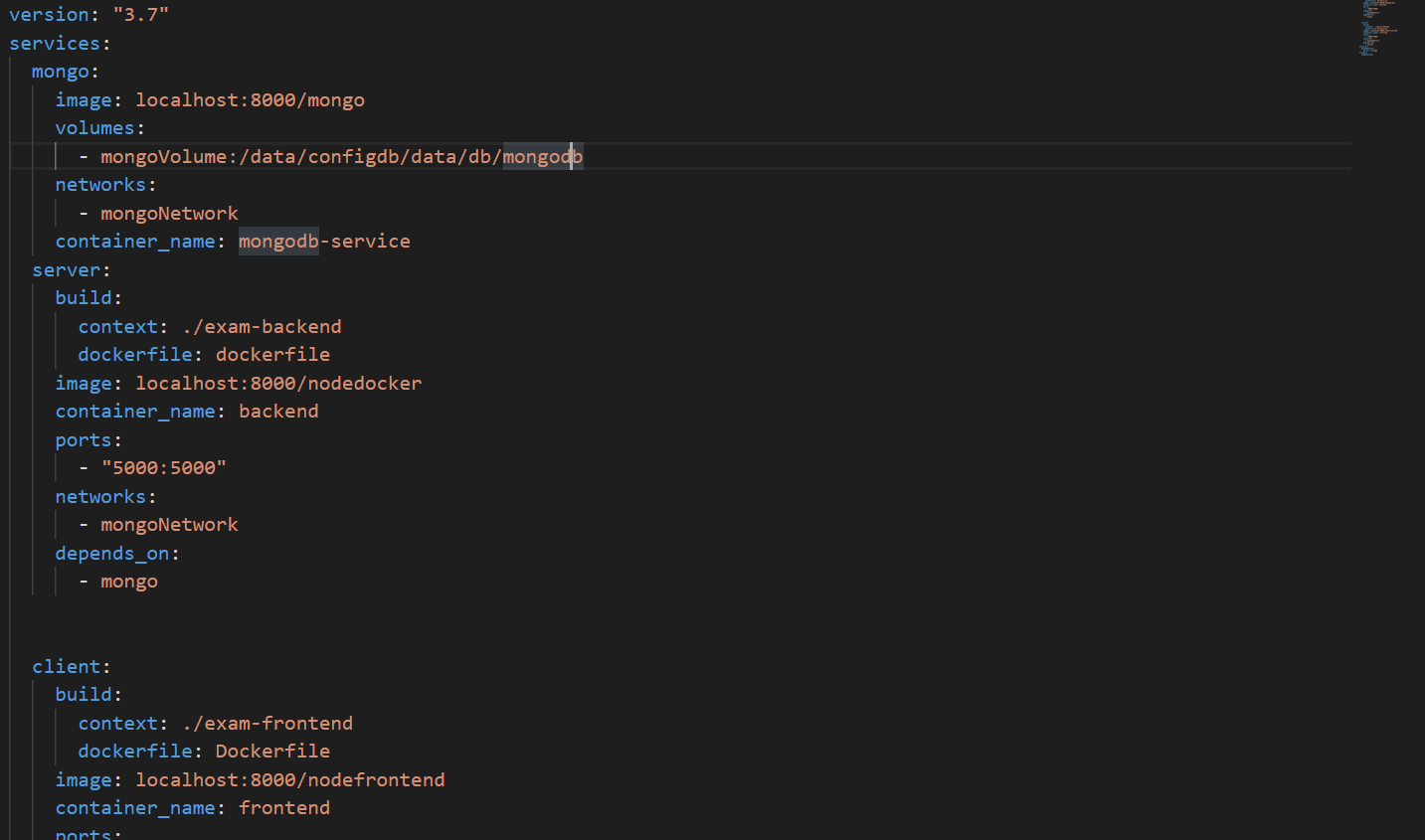
Voici la méthode pour créer un register privee

1. **Stocker les images que vous avez construites :**
2. **Visualiser via une UI les images qu’il contient :**

## **12.Redéployer l’application (frontend, backend et base de données) en utilisant docker-compose tout en récupérant l’image des conteneurs depuis ce nouveau registre privé. NB : S’assurer que l’application fonctionne correctement (prises d’écran dunavigateur) :**

D’abord on tag nos conteneurs qui se trouve dans dockerhub

on modifie maintenant le fichier docker-compose

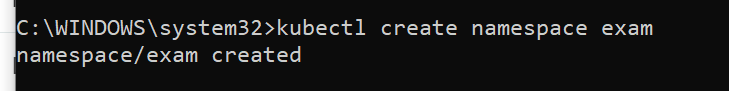


# ***Partie 2 :***

Dans cette partie, il est permis d’utiliser n’importe quel outil permettant de créer un

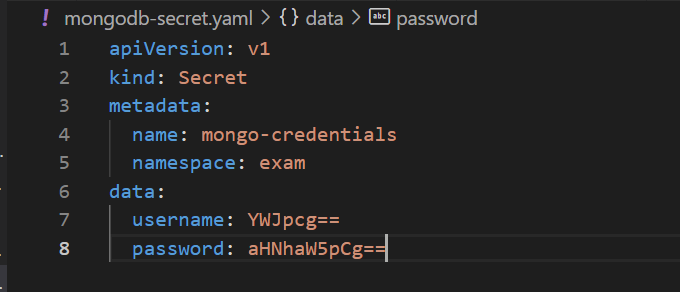
cluster Kubernetes en local (ex : Minikube, MicroK8s, kind, k3d, k3s, etc.)

## **Créer les Kubernetes manifestes (fichiers YAML) nécessaires pour déployer cette application dans un cluster Kubernetes (sous un namespace nommé "exam") :**

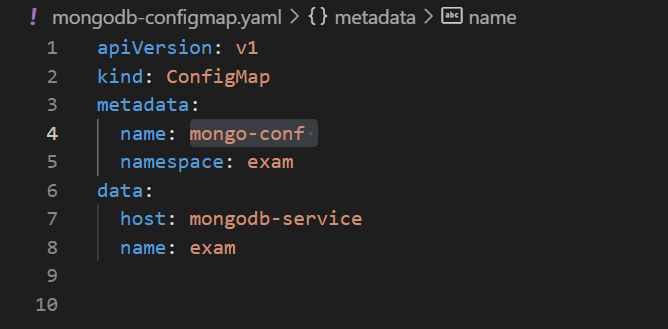
On doit d’abord créer un namespace nomme exam.

On configure notre current namespace en exam . 

Maintenant tout d’abord on cree le fichier mongodb-secret.yaml



Maintenant on cree le fichier mongodb-configmap.yaml



Maintenant on cree mongodb deployment and service

apiVersion: v1

kind: Service

metadata:

  name: mongodb-service

  namespace: exam

  labels:

    app: mongodb-service

    tier: exam

spec:

  ports:

    - port: 27017

      targetPort: 27017

  selector:

    app: mongodb

    tier: database

  clusterIP: None

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

  name: mongodbvolume

  namespace: exam

  labels:

    app: mongodb

    tier: database

spec:

  accessModes:

    - ReadWriteOnce

  resources:

    requests:

      storage: 1Gi

---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: mongodb

  namespace: exam

  labels:

    app: mongodb

    tier: database

spec:

  selector:

    matchLabels:

      app: mongodb

      tier: database

  strategy:

    type: Recreate

  replicas: 2

  template:

    metadata:

      labels:

        app: mongodb

        tier: database

    spec:

      containers:

      - image: mongo # image from docker-hub

        name: mongodb

        env:

        - name: mongo\_USER # Setting USER username on mysql From a 'Secret'

          valueFrom:

            secretKeyRef:

              name: mongo-credentials

              key: username

        - name: mongo\_PASSWORD # Setting USER Password on mysql From a 'Secret'

          valueFrom:

            secretKeyRef:

              name: mongo-credentials

              key: password

        - name: mongodb\_DATABASE # Setting Database Name from a 'ConfigMap'

          valueFrom:

            configMapKeyRef:

              name: mongo-conf

              key: name

        ports:

        - containerPort: 27017

          name: mongodb

        volumeMounts:        # Mounting voulume obtained from Persistent Volume Claim

        - name: mongodbvolume-storage

          mountPath: /var/lib/mongodb

      volumes:

      - name: mongodbvolume-storage # Obtaining 'vloume' from PVC

        persistentVolumeClaim:

          claimName: mongodbvolume

maintenant backend-deployment.yaml :

apiVersion: v1

kind: Service

metadata:

  name: app-backend

spec:

  selector:  # backend application pod lables should match these

    app: app-backend

    tier: backend

  ports:

  - protocol: "TCP"

    port: 5000

    targetPort: 5000

  type: NodePort   # use NodePort, if you are not running Kubernetes on cloud

---

# Configure 'Deployment' of backend application

apiVersion: apps/v1

kind: Deployment

metadata:

  name: app-backend

  labels:

    app: app-backend

    tier: backend

spec:

  replicas: 2    # Number of replicas of back-end application to be deployed

  selector:

    matchLabels: # backend application pod labels should match these

      app: app-backend

      tier: backend

  template:

    metadata:

      labels: # Must macth 'Service' and 'Deployment' labels

        app: app-backend

        tier: backend

    spec:

      containers:

      - name: backend

        image: abirhsaini/exam:nodedocker

        env:

          - name: DB\_HOST   # Setting Database host address from configMap

            valueFrom:

              configMapKeyRef:

                name: mongo-conf   # name of configMap

                key: host

          - name: mongodb\_DATABASE  # Setting Database name from configMap

            valueFrom:

              configMapKeyRef:

                name: mongo-conf

                key: name

          - name: mongo\_USER  # Setting Database username from Secret

            valueFrom:

              secretKeyRef:

                name: mongo-credentials # Secret Name

                key: username

          - name: mongo\_PASSWORD # Setting Database password from Secret

            valueFrom:

              secretKeyRef:

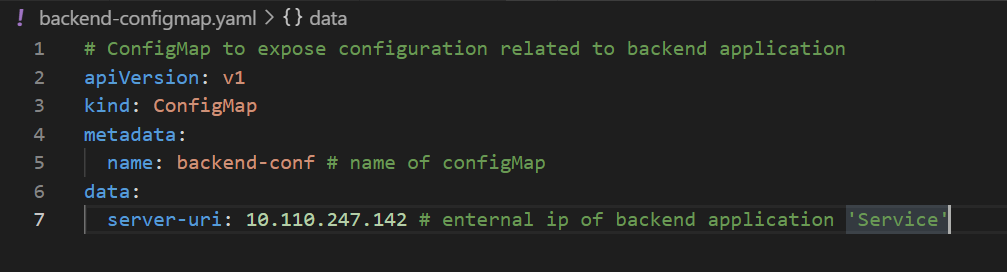
                name: mongo-credentials

                key: password

        ports:

        - containerPort: 5000

Maintenant bakend-configmap:



Finalement frontend-deployment.yaml

# Define 'Service' to expose FrontEnd Application

apiVersion: v1

kind: Service

metadata:

  name: app-frontend

spec:

  selector: # pod labels should match these

    app: app-frontend

    tier: frontend

  ports:

  - protocol: "TCP"

    port: 3000

    targetPort: 3000

  type: NodePort # use NodePort if you are not running Kubernetes on Cloud

---

# 'Deployment' to manage of configuration of frontEnd Deployment

apiVersion: apps/v1

kind: Deployment

metadata:

  name: app-frontend

  labels: # pod labels should match these

    app: app-frontend

    tier: frontend

spec:

  replicas: 2  # number of replicas of frontEnd application

  selector:

    matchLabels:

      app: app-frontend

      tier: frontend

  template:

    metadata:

      labels: # Must match 'Service' and 'Deployment' labels

        app: app-frontend

        tier: frontend

    spec:

      containers:

      - name: frontend

        image: abirhsaini/exam:nodefrontend # docker image of frontend application

        env: # Setting Environmental Variables

          -  name: SERVER\_URI # Setting Backend URI from configMap

             valueFrom:

               configMapKeyRef:

                 name: backend-conf # Name of configMap

                 key: server-uri

        ports:

        - containerPort: 3000

Pour appliquer ces changement :

