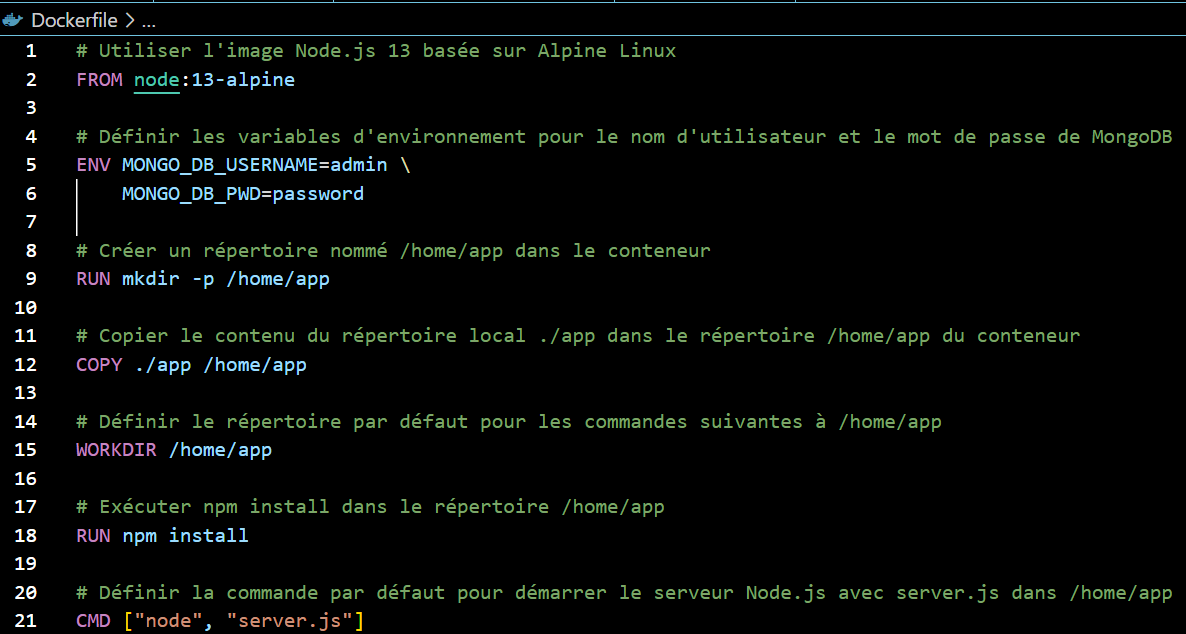
**Rapport TP Examen**

Wafae hamdaoui

1. **Docker et Dockerfile**

**Exercice 1** :

Créez un Dockerfile pour une application web simple (par exemple, une application Node.js) et construisez l'image Docker correspondante.



Construire une image Docker en utilisant les instructions du Dockerfile actuel, étiquetée comme *docker:latest* avec la commande suivante :

***docker build --pull --rm -f "Dockerfile" -t docker:latest "."***

***docker build:*** Commande pour construire une image Docker.

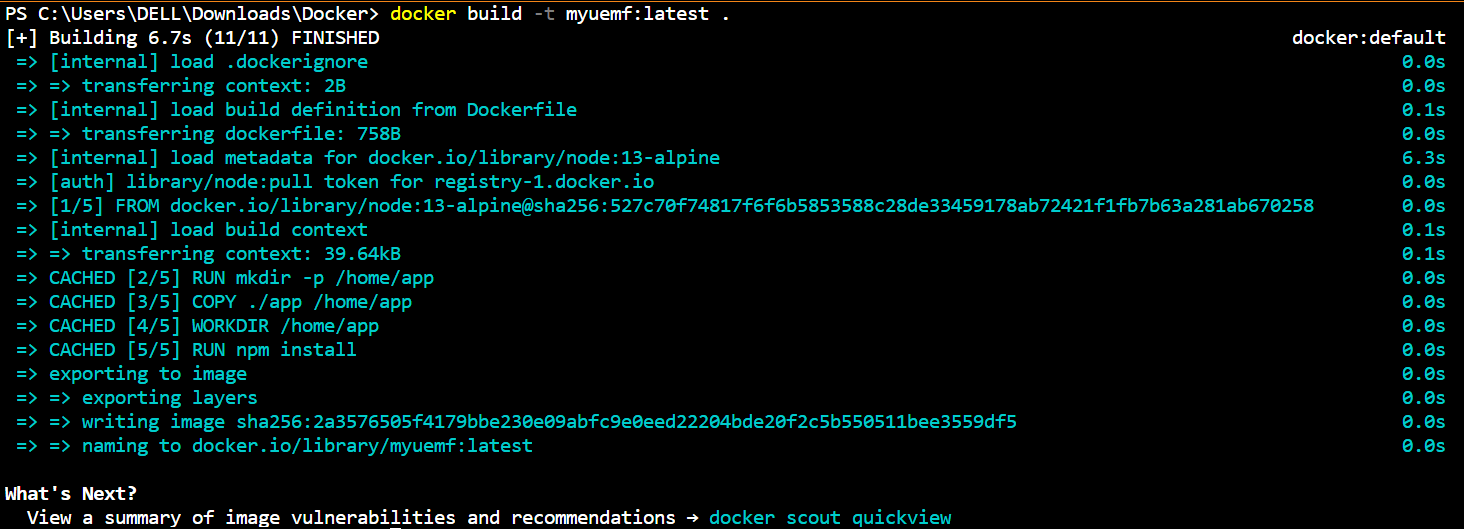
***--pull:*** Cette option indique à Docker de récupérer les versions les plus récentes des images de base (node:13-alpine dans ce cas) lors de la construction de l'image.

***--rm:*** Cette option supprime le conteneur intermédiaire une fois que la construction de l'image est terminée, ce qui économise de l'espace.

***-f "Dockerfile":*** Spécifie le nom du fichier Dockerfile à utiliser pour la construction de l'image.

***-t docker:latest:*** Utilise l'option -t pour tagger (étiqueter) l'image créée avec le nom docker et la version latest.

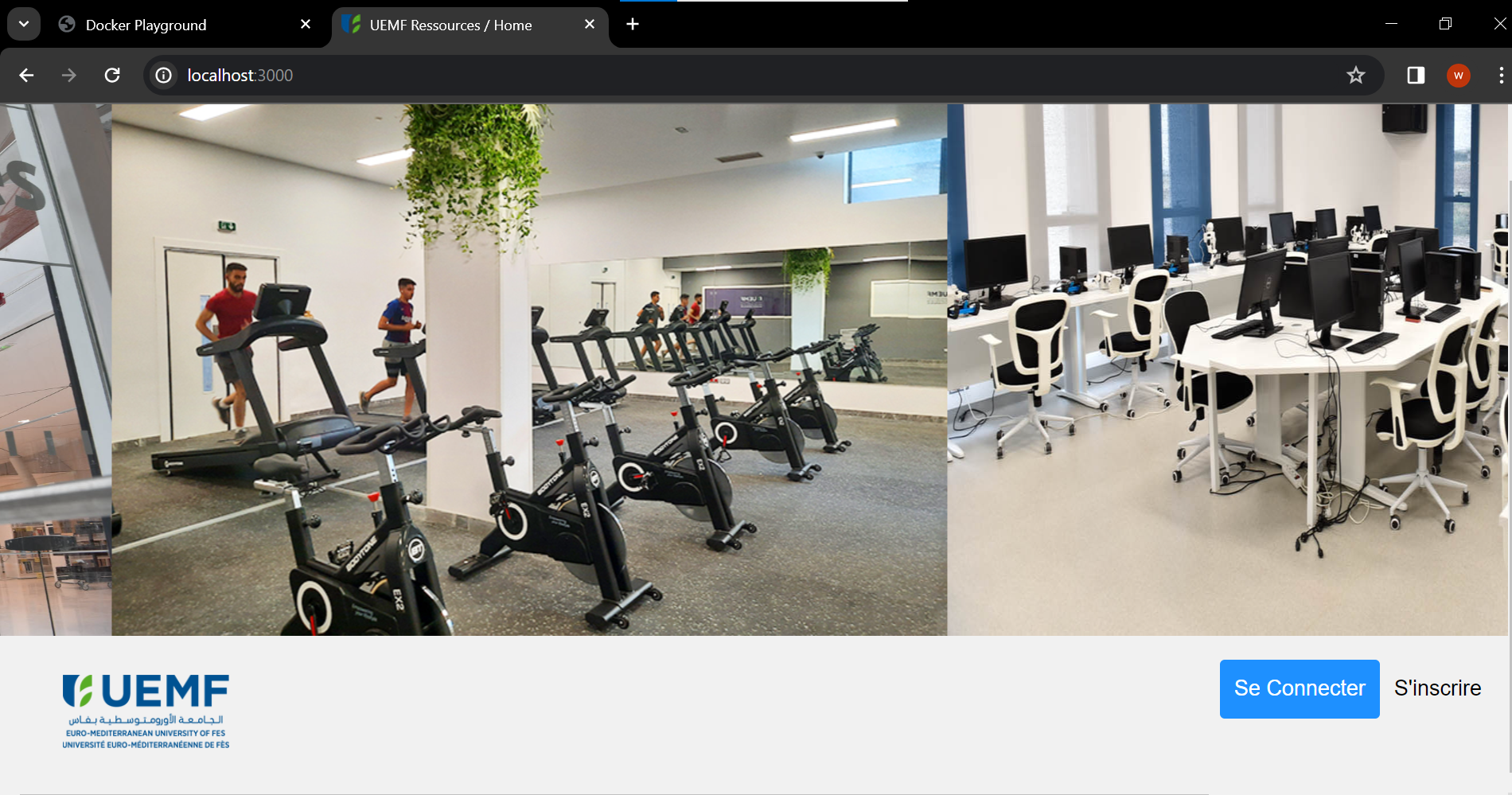
**". ":** Indique à Docker de chercher le Dockerfile dans le répertoire courant (.), où se trouve le Dockerfile.



**Exercice 2 :**

Lancez le conteneur à partir de l'image construite et testez son fonctionnement.

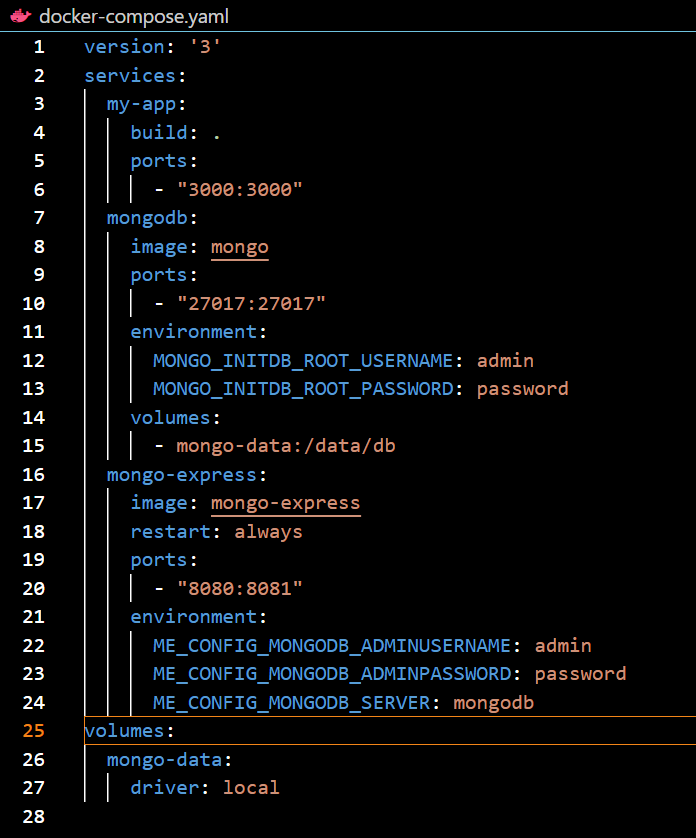




1. **Docker Compose**

**Exercice 3** :

Écrivez un fichier docker-compose.yaml pour l'application web précédente, en y ajoutant une base de données (par exemple, MySQL ou MongoDB) comme service distinct.



Expliquons chaque partie :

* **Version Docker Compose** : Le fichier est écrit en syntaxe de version 3, comme indiqué par version: '3'.
* **Services :**

**Service my-app** :

* Utilise l'instruction build: . pour construire l'image Docker du service à partir du contexte local (actuel) du répertoire où se trouve le fichier Docker Compose.
* Mappe le port 3000 sur l'hôte vers le port 3000 dans le conteneur.

**Service mongodb :**

* Utilise l'image Docker officielle MongoDB.
* Mappe le port 27017 sur l'hôte vers le port 27017 dans le conteneur.
* Définit des variables d'environnement MongoDB pour le nom d'utilisateur et le mot de passe root.
* Définit un volume nommé mongo-data pour persister les données de MongoDB.

**Service mongo-express :**

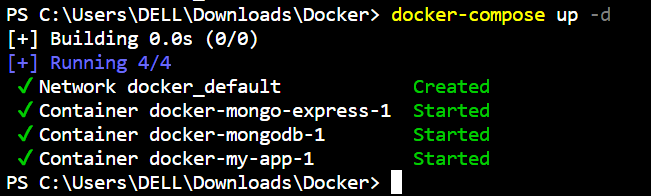
* Utilise l'image Docker officielle Mongo Express.
* Redémarre toujours le conteneur.
* Mappe le port 8080 sur l'hôte vers le port 8081 dans le conteneur.
* Définit des variables d'environnement Mongo Express pour le nom d'utilisateur admin, le mot de passe et l'adresse du serveur.

**Volumes :** Définit un volume nommé mongo-data avec le pilote local.

* Ce fichier Docker Compose décrit une configuration avec une application Node.js (my-app) utilisant une base de données MongoDB (mongodb). Mongo Express (mongo-express) est inclus pour la gestion de la base de données. Les services sont interconnectés au sein d'un réseau Docker, ce qui leur permet de communiquer.

**Exercice 4** :

Démarrez l'ensemble de l'application avec Docker Compose et validez le fonctionnement intégral (application web + base de données).

**

La commande **docker-compose up -d** est utilisée pour démarrer les services définis dans un fichier Docker Compose en mode détaché.

**docker-compose** : C'est l'outil en ligne de commande permettant de définir et d'exécuter des applications Docker multi-conteneurs à l'aide d'un fichier Compose.

**up** : Cette commande est utilisée pour démarrer les services définis dans le fichier Compose. Si les services n'existent pas, ils sont créés.

**-d :** Ce drapeau signifie "détaché", ce qui signifie que les services s'exécuteront en arrière-plan.

Vous avez un fichier docker-compose.yml dans votre répertoire qui définit vos services, réseaux et volumes.

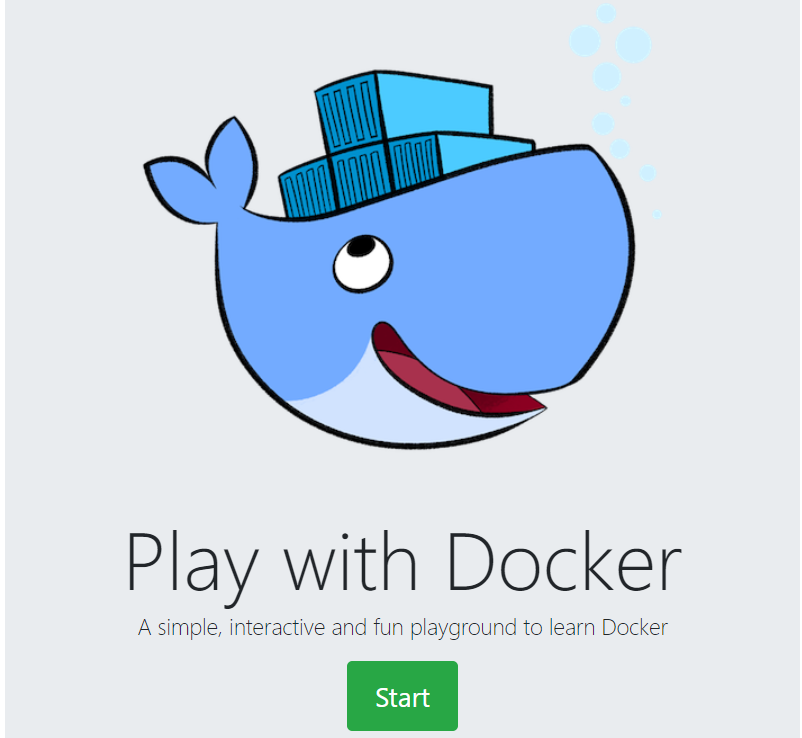
* Cela démarrera les services définis dans votre fichier docker-compose.yml, et ils s'exécuteront en arrière-plan.

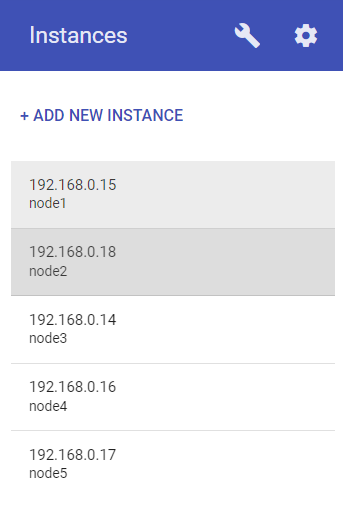
1. **Orchestration avec Docker Swarm**

**Exercice 5** :

Utilisez la plateforme play-with-docker pour initier un cluster Docker Swarm.

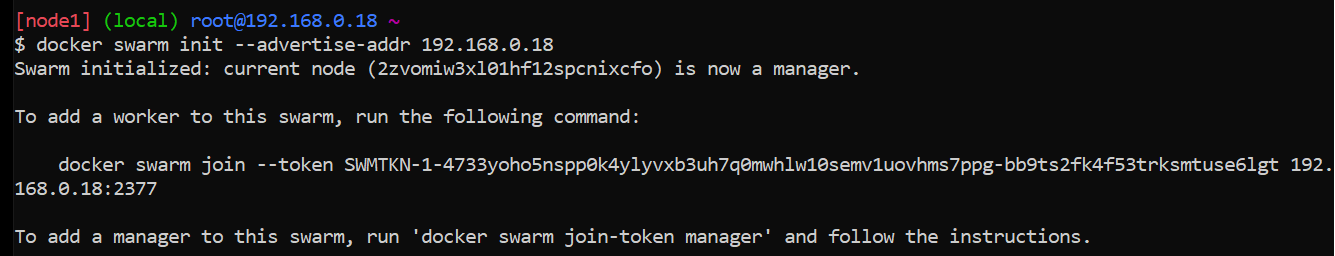
Démarrons une nouvelle session en cliquant sur "Start" pour lancer une nouvelle session.



Puis créons 5 nœuds. 

**Initialisatons de mode Swarm:**

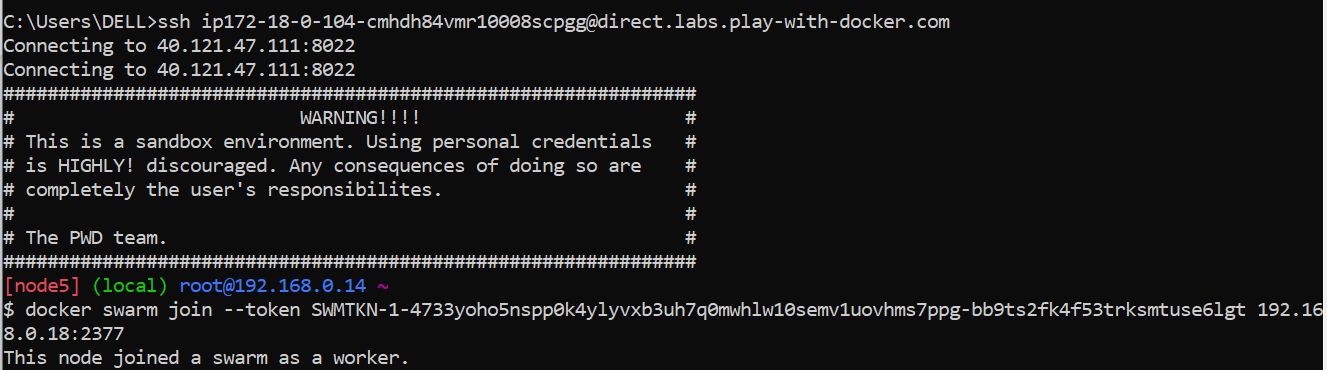
Ouvrons le terminal sur l'une des instances puis exécutons la commande suivante pour initialiser Docker Swarm sur le premier nœud :

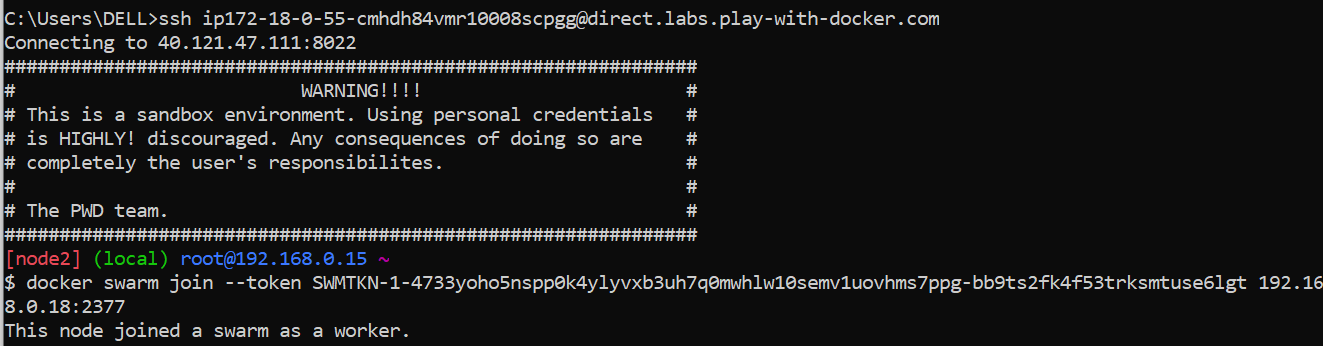
****

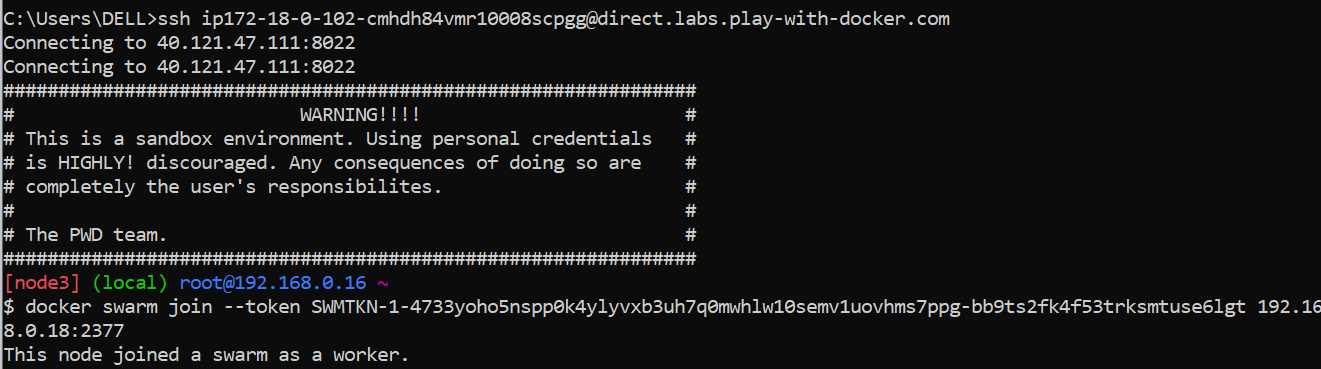
La commande nous fournis un jeton que nous utilisons pour joindre d'autres nœuds au Swarm.

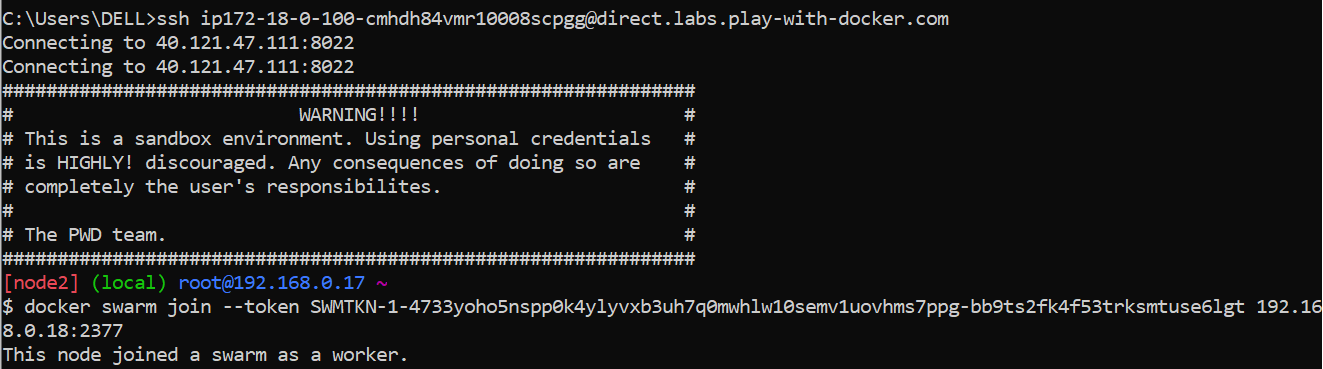
**Joindre des nœuds supplémentaires:**

Ouvrons des terminaux sur d'autres instances et exécutons la commande fournie par docker swarm init sur le premier nœud.



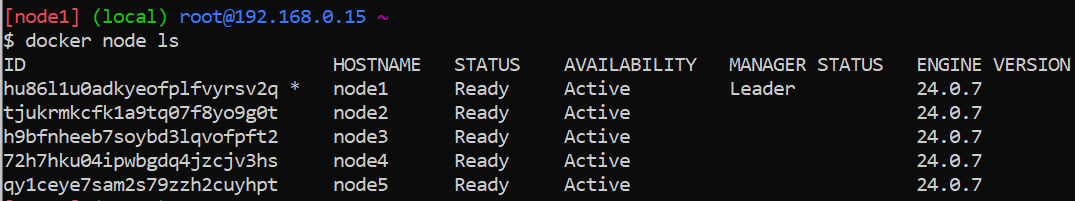






**Vérifier l'état du Swarm:**

Sur le nœud manager, exécutons la commande **docker node ls** pour voir tous les nœuds.



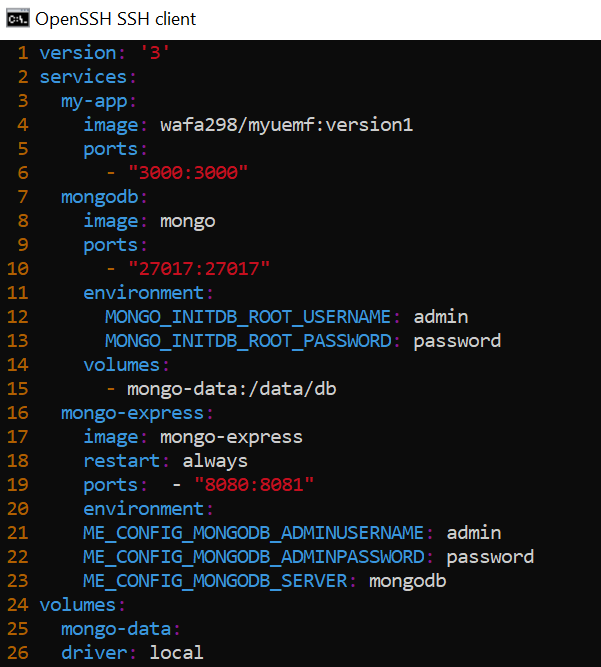
**Exercice 6 :**

Déployez l'application web et la base de données sur le cluster Docker Swarm en utilisant les fichiers Docker Compose comme référence.

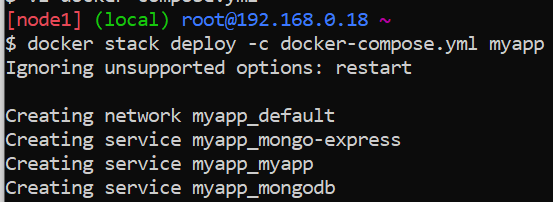
**Création de fichier Docker Compose:**

****

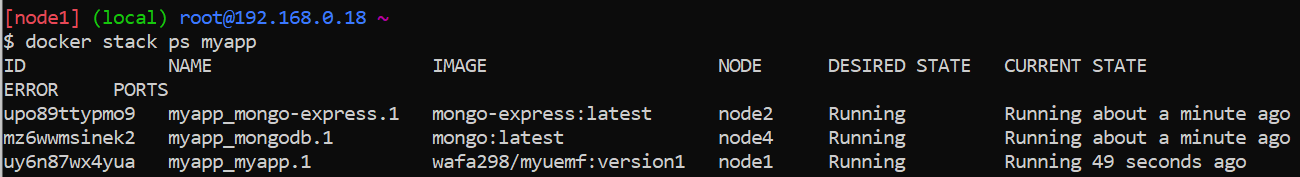
Puis copions le contenu de fichier précédant dans ce fichier

****

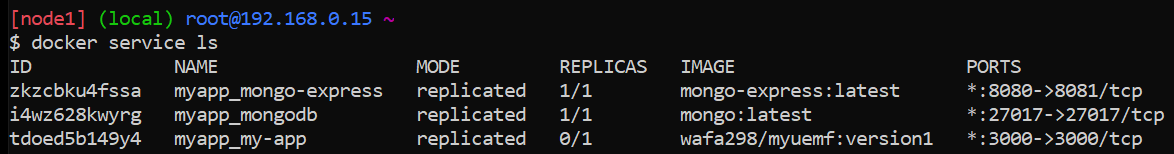
**Déployer sur Swarm:**

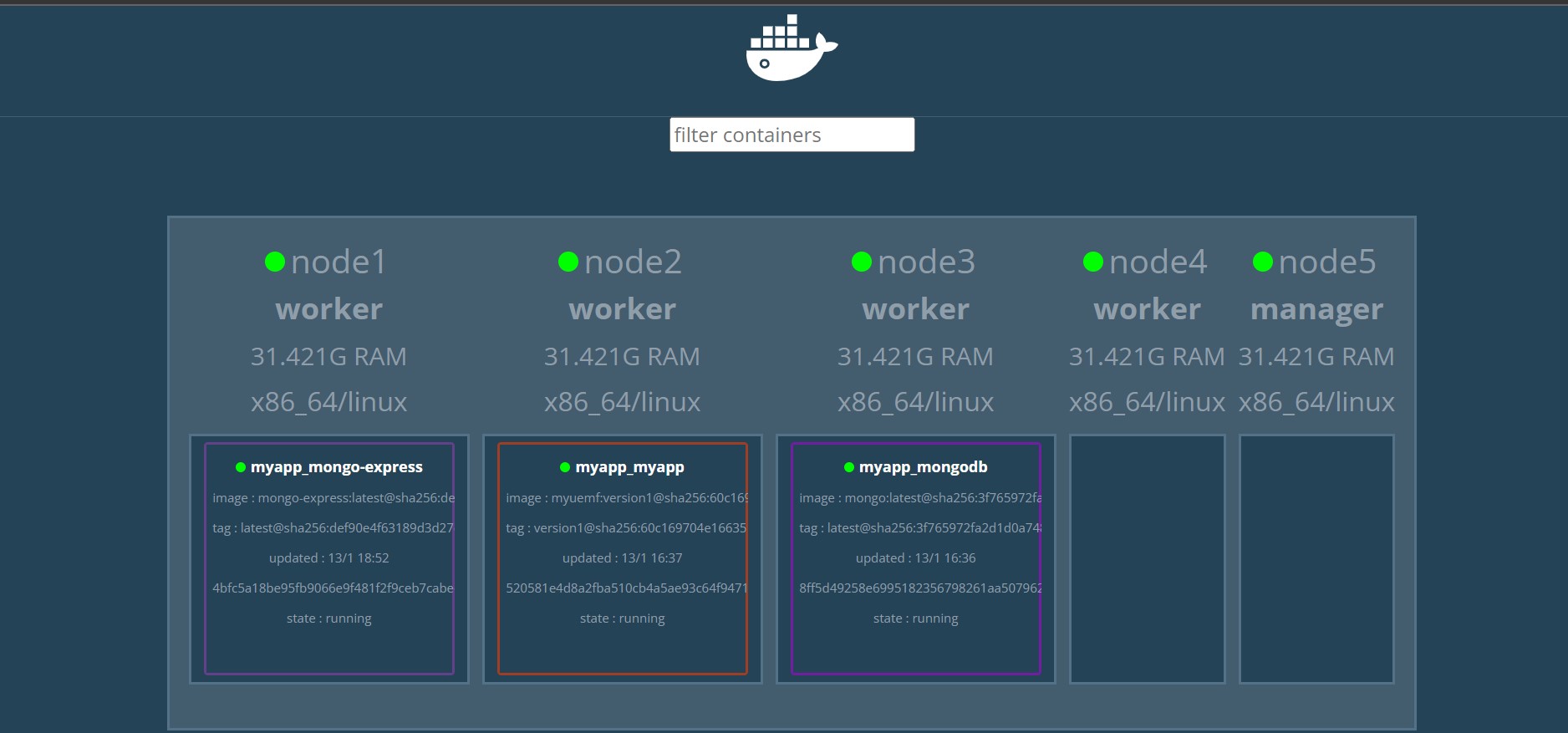
Utilisons la commande **docker stack deploy** pour déployer les services sur le cluster Swarm. 

**Vérifier le déploiement :**

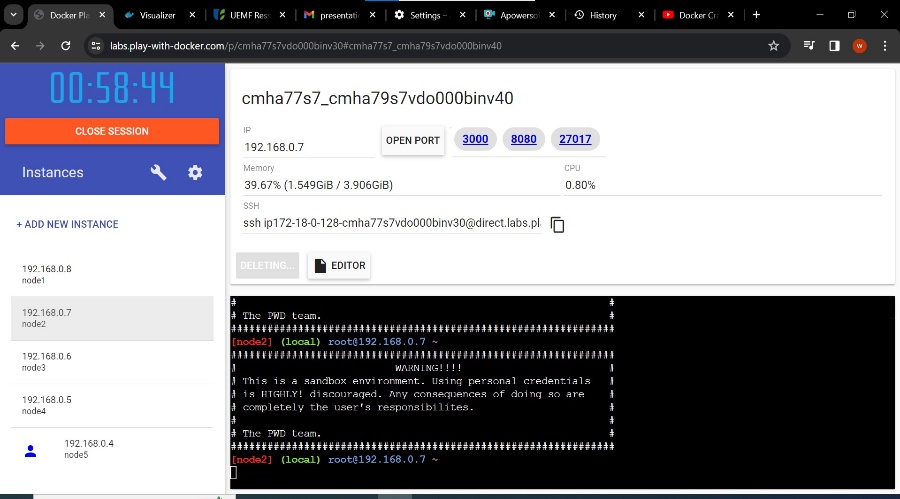
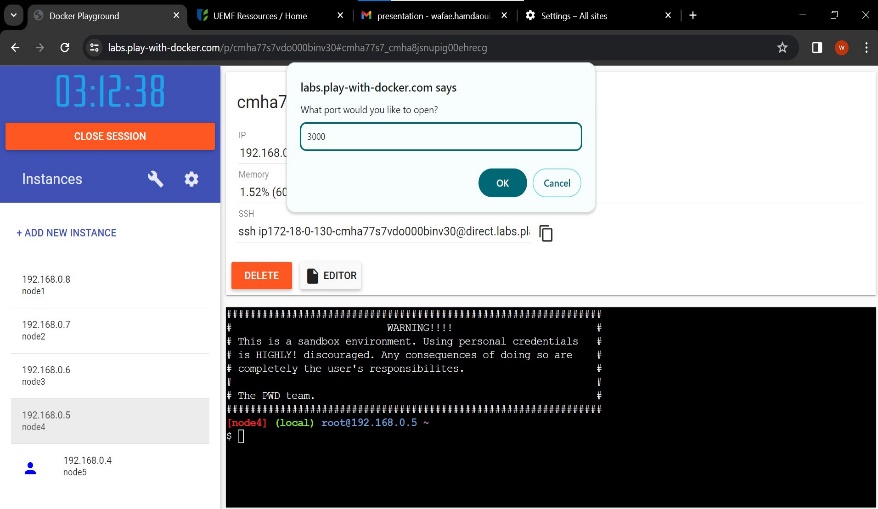
Utilisons la commande **docker stack ls** pour lister les stacks déployés. 

Utilisons la commande **docker service ls** pour lister les services déployés dans le stack.

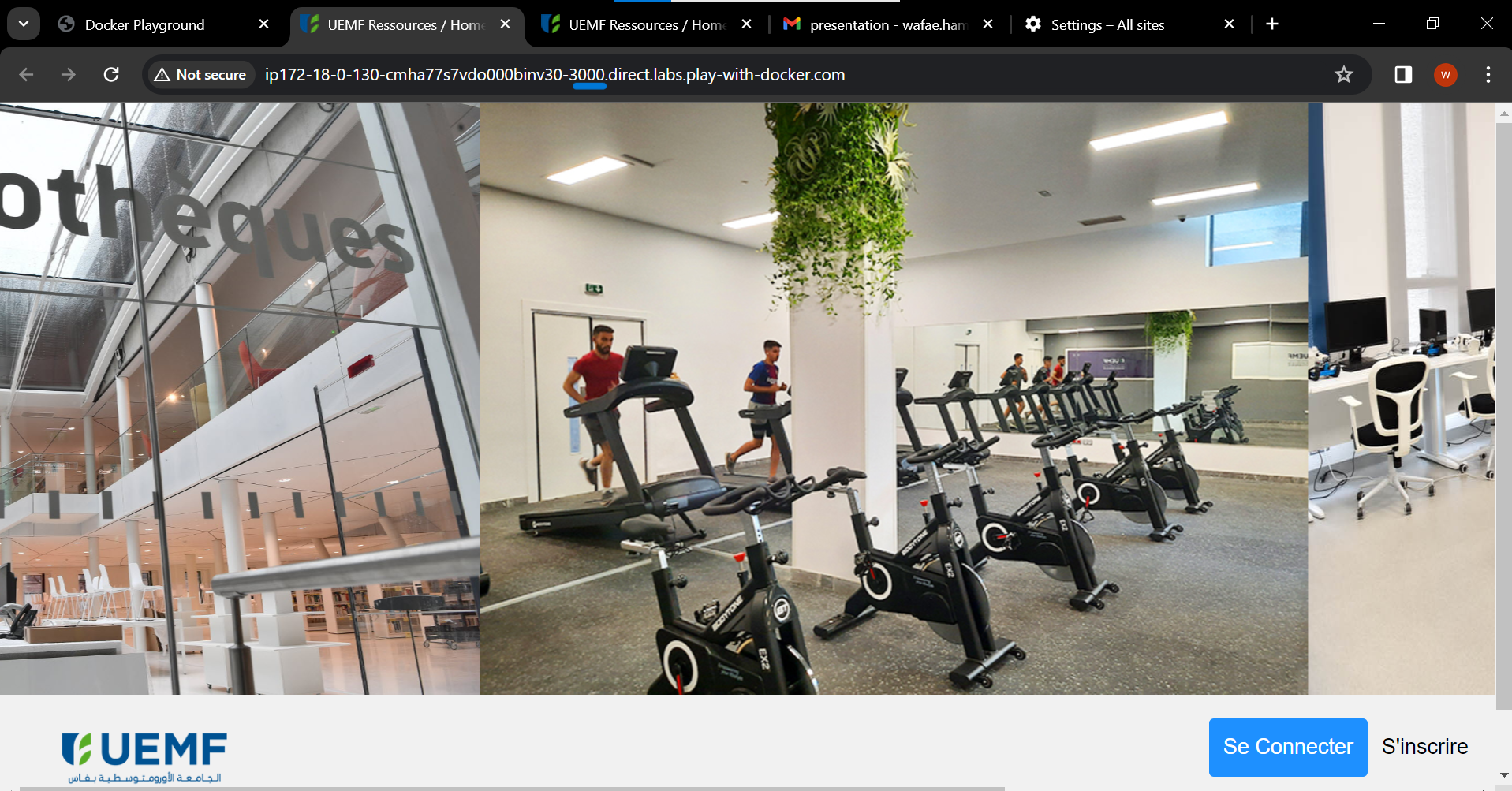


Voici le résultat dans docker swarm visualizer : 

Si nous ouvrons le port 3000 comme suivant

Nous obtenu notre application



**Exercice 7** :

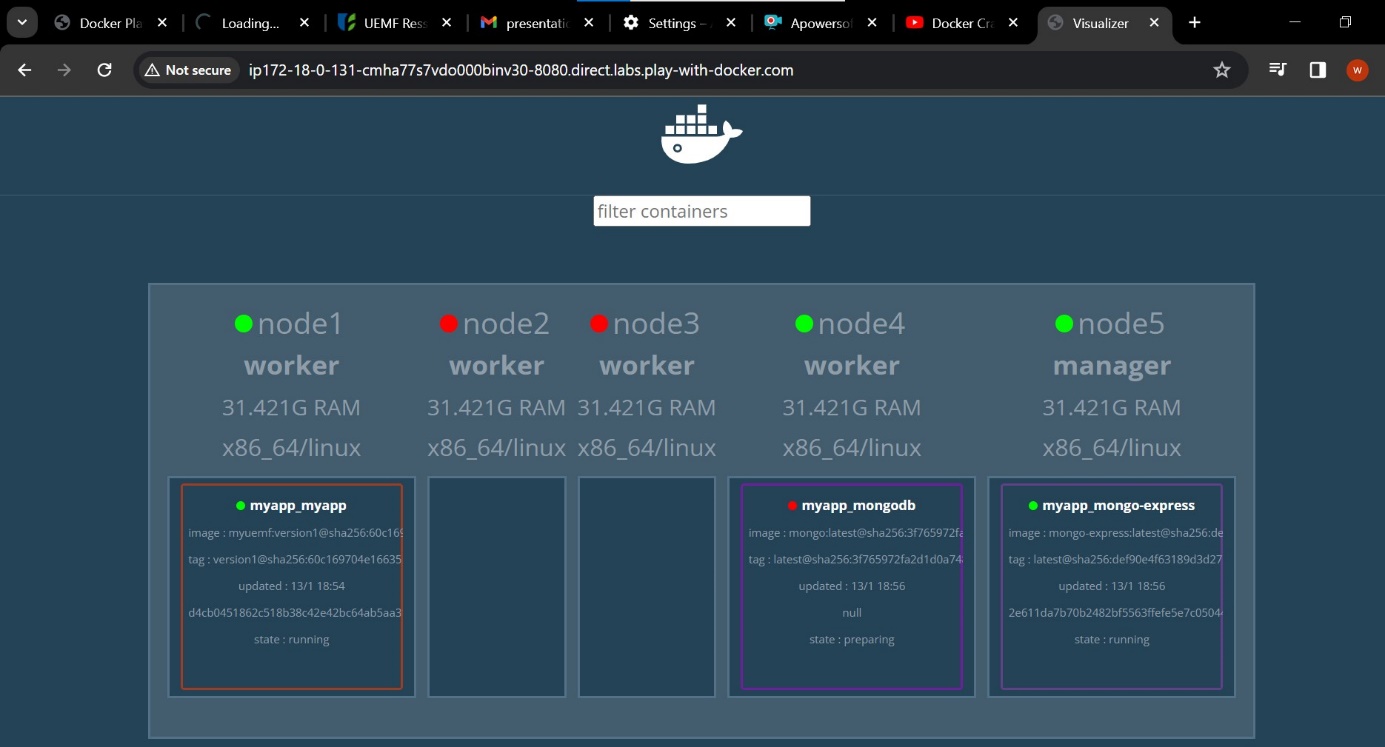
Testez la résilience de l'application en arrêtant manuellement un des nœuds et observez le comportement du cluster.

**Arrêtez Manuellement le Nœud 2 :** ****

**Observez le Comportement du Cluster :**

le manager a détecté que le nœud 2 n'est plus accessible. Docker Swarm donc réorganise automatiquement les services pour maintenir le nombre de réplicas spécifié dans la configuration des services. Les réplicas des services qui étaient initialement déployées sur le nœud 2 sont redéployées sur nœuds 5 actifs du cluster. 

**Répétez le processus pour le nœuds 3 :**



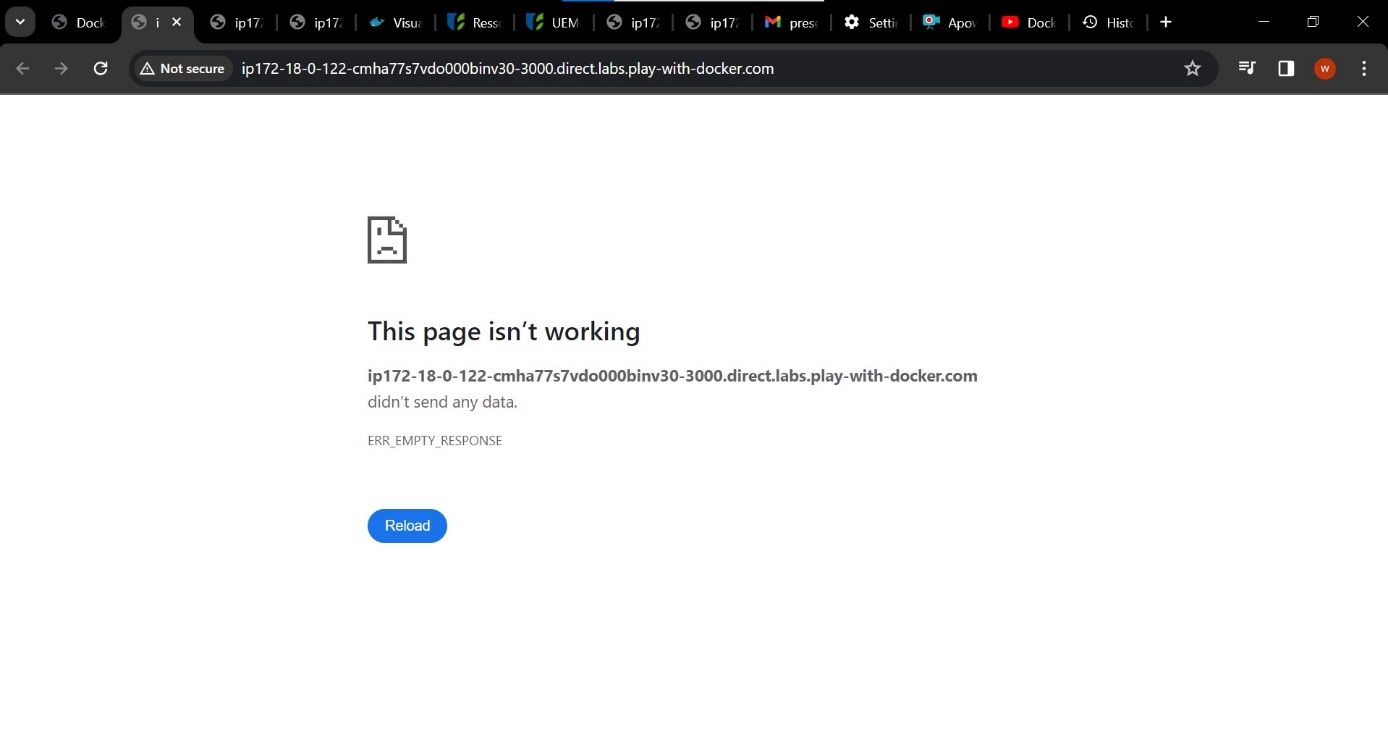
Comme nous pouvons voir Les réplicas des services qui étaient initialement déployées sur le nœud 3 sont redéployées sur nœuds 4 actifs du cluster.

* Docker Swarm vise à maintenir la disponibilité des services malgré la déconnexion d'un nœud. Il utilise la stratégie de redéploiement pour répartir les réplicas sur les nœuds actifs.

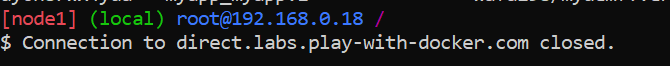
**La suppression de noude manager :**

Nous avons supprimé le noeud manager, le cluster ne disposera plus d'aucun nœud manager pour gérer et coordonner les opérations du cluster. Voici ce qui a produit :

**Indisponibilité des Services :** Les services déployés sur le cluster peuvent devenir indisponibles, car il n'y a plus de nœud pour coordonner leur fonctionnement.



**Perte de Gestion :**

La suppression du nœud manager entraînera une perte totale de la capacité de gestion du cluster. Il n'y aura plus de nœud pour prendre des décisions de déploiement, gérer les services, etc. 

1. **Questions Théoriques**

**Q1 : Expliquez le concept de conteneurisation et comment Docker y contribue.**

La conteneurisation vise à générer des ensembles logiciels autonomes assurant une exécution cohérente, peu importe l'environnement. Les développeurs créent et déploient des images de conteneur, des fichiers renfermant toutes les données essentielles à l'exécution d'une application en mode conteneur. Les conteneurs Docker se présentent comme des packages indépendants renfermant des applications et les fichiers nécessaires, façonnés conformément à la structure standard de Docker.

**Q2 : Décrivez les différences entre une image Docker et un conteneur Docker.**

Les images Docker peuvent être considérées comme des instantanés statiques et figés de conteneurs vivants. Elles représentent un ensemble immuable de fichiers, de configurations, et de dépendances nécessaires pour exécuter une application. En revanche, les conteneurs Docker sont les instances dynamiques qui sont en cours d'exécution à partir de ces images. Ils encapsulent l'image, permettant à une application de fonctionner de manière isolée et reproductible, avec la possibilité de modifier leur état pendant leur exécution.

**Q3 : Discutez de l'utilité de Docker Compose et comment il facilite la gestion des applications multi-conteneurs.**

Docker Compose se présente comme un outil dédié à la définition et à l'exécution d'applications Docker composées de plusieurs conteneurs. Grâce à Compose, la configuration des services de votre application s'effectue à l'aide d'un fichier YAML. Une fois cette configuration définie, le lancement de tous les services se fait aisément avec une seule commande.

Cet outil s'avère particulièrement utile pour orchestrer plusieurs images de conteneurs sur un même ordinateur hôte, offrant ainsi une solution pratique et unifiée pour le déploiement et la gestion d'applications multi-conteneurs.

**Q4 : Décrivez ce qu'est Docker Swarm et son avantage pour l'orchestration de conteneurs**.

Docker Swarm est un outil d'orchestration intégré à Docker spécialement conçu pour faciliter la coordination et l'utilisation collective des ressources sur un grand nombre de machines, dans le but de déployer des conteneurs Docker de manière efficace. Ses fonctionnalités principales incluent la coordination des conteneurs, l'assignation de tâches spécifiques à chaque conteneur, la gestion de l'état des conteneurs au sein du cluster, et la capacité à réaffecter les tâches en cas de besoin. En résumé, Docker Swarm offre une orchestration transparente pour simplifier le déploiement et la gestion des conteneurs sur un ensemble de machines.