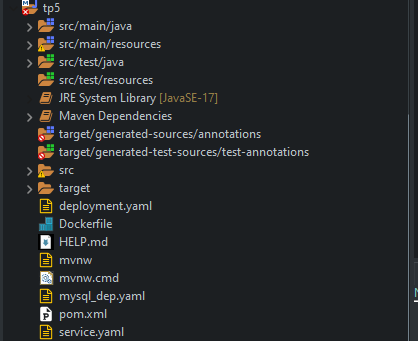
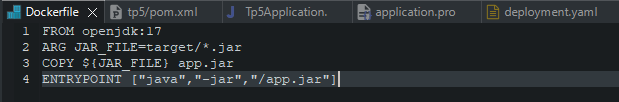
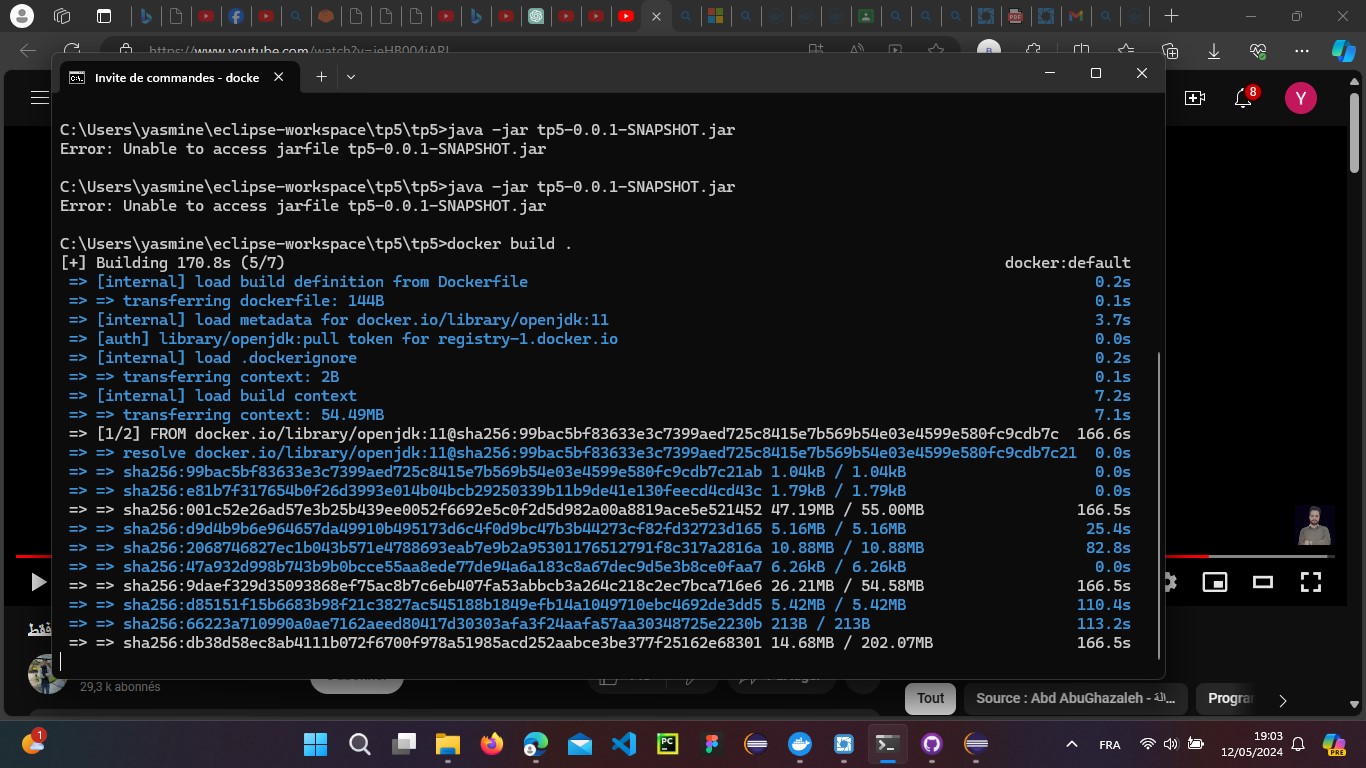
Kubernetes

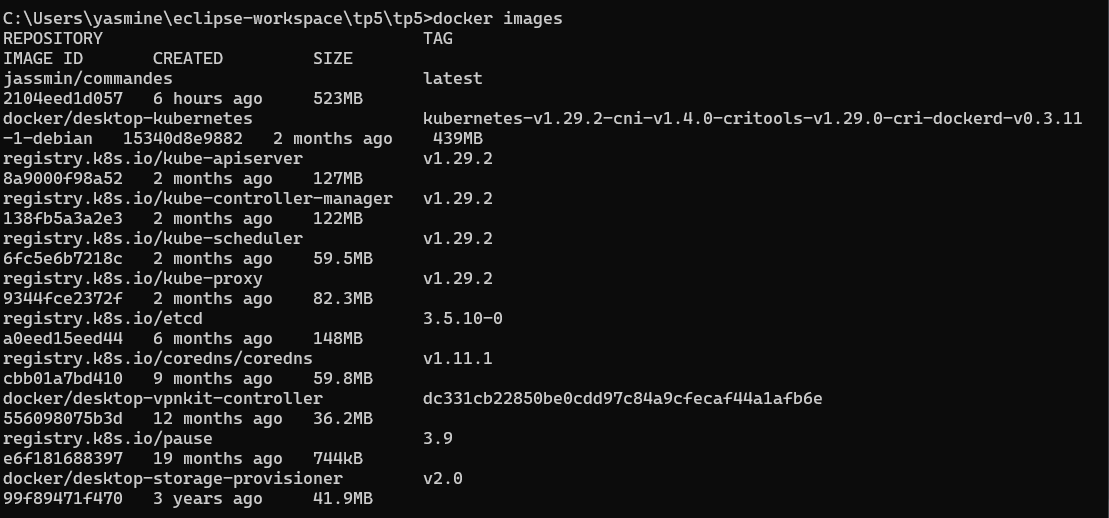
Premièrement nous devons écrire un Docker file qui définit les étapes pour créer une image Docker permettant d'exécuter une application Java. Tout d'abord, il spécifie l'utilisation de l'image de base **openjdk:17**, qui contient Java JDK 17. Ensuite, il copie le fichier JAR de l'application Java dans le conteneur Docker. Enfin, il configure le point d'entrée du conteneur pour exécuter l'application Java lors du démarrage en utilisant la commande **java -jar /app.jar**. Ainsi, ce Docker file encapsule l'application Java dans un conteneur Docker, fournissant un environnement isolé et portable pour son exécution.



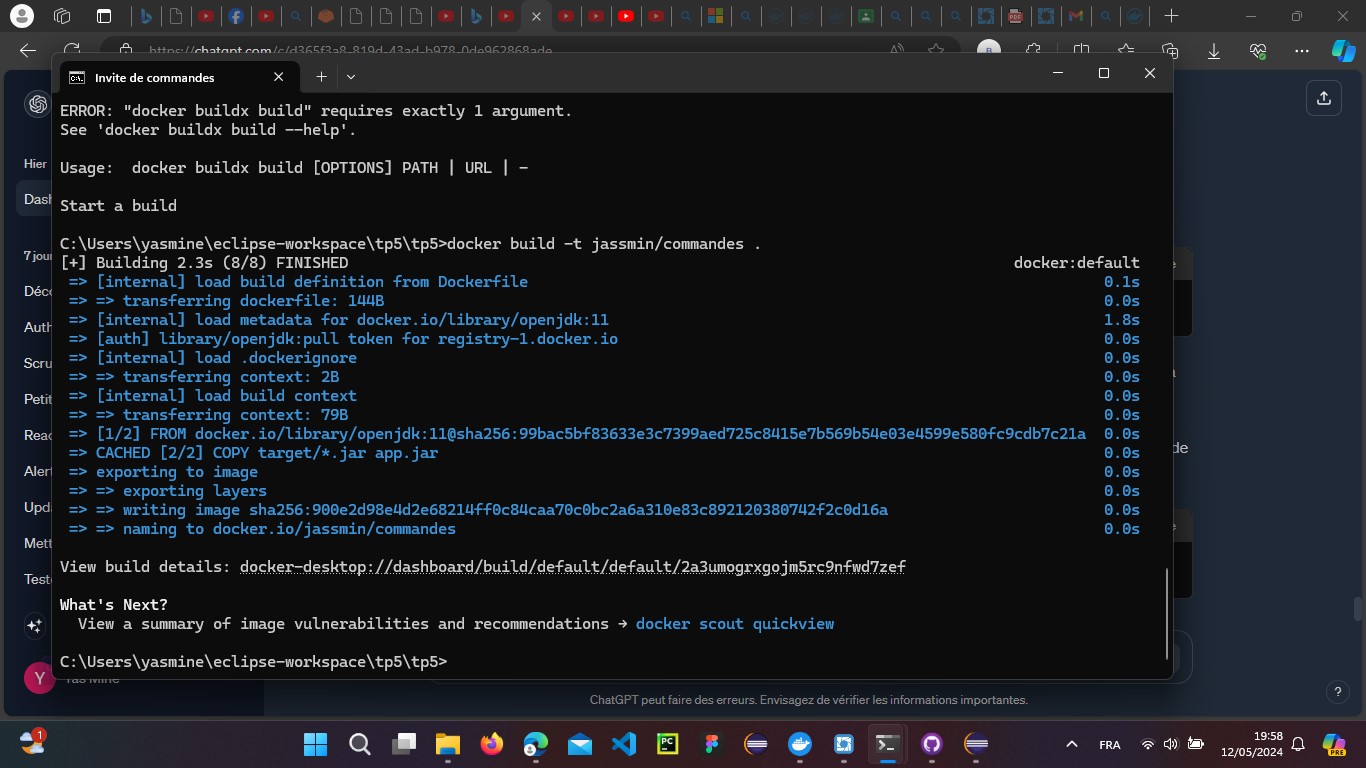
  
Après cela, nous allons construire notre image Docker.

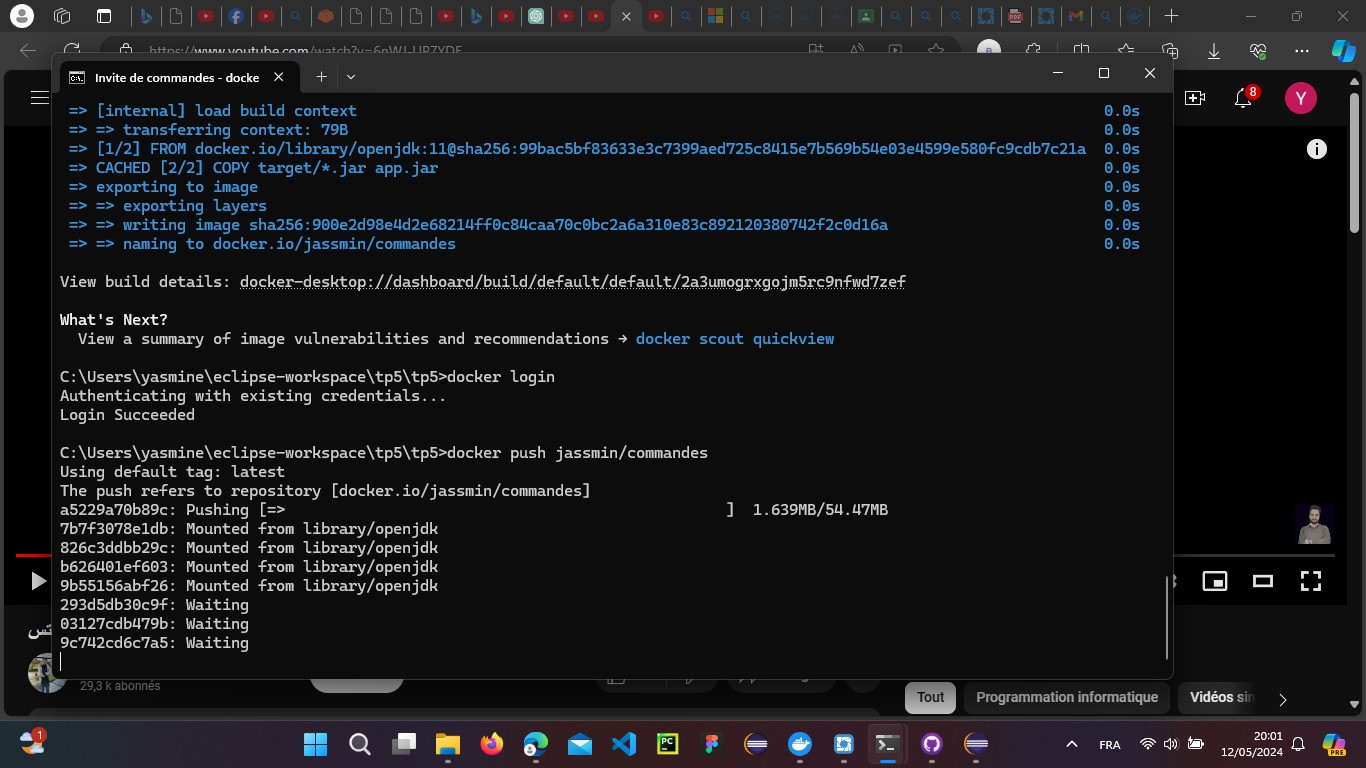


On liste les images disponibles dans notre Docker à l'aide de la commande "docker images".Haut du formulaire

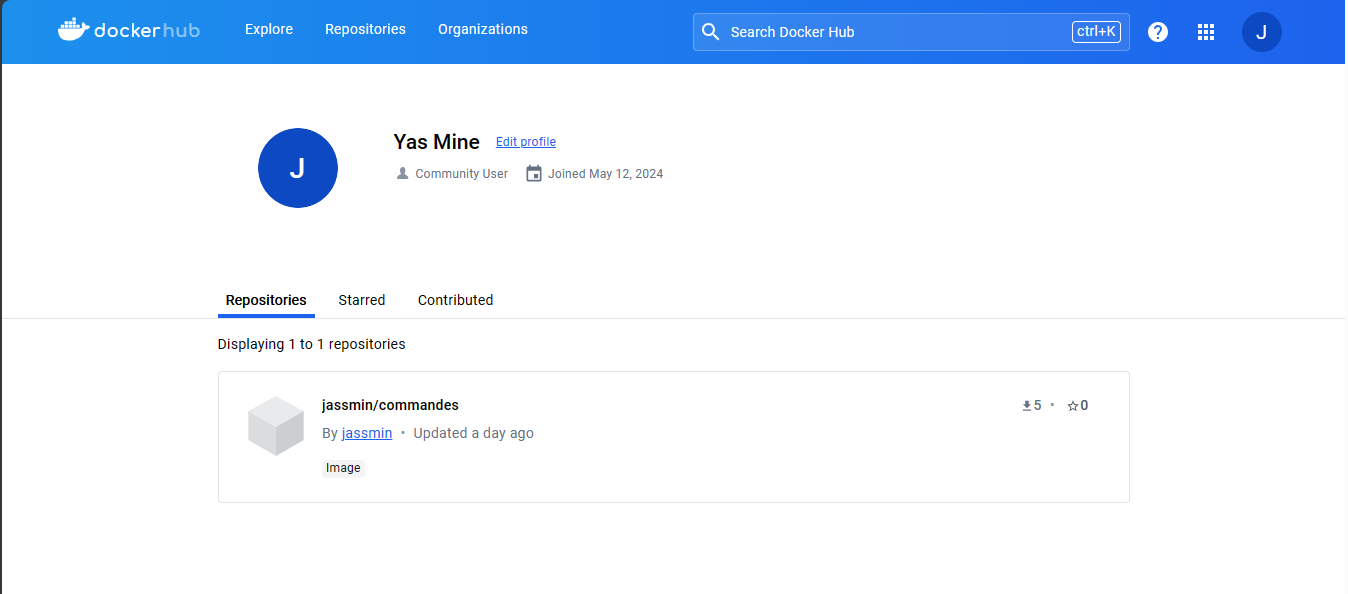


Lorsque nous procédons au déploiement, il est nécessaire de créer les images en effectuant le build, puis de les pousser dans notre Docker Hub.





Après cela, nous vérifions la disponibilité de cette image dans le référentiel de mon compte

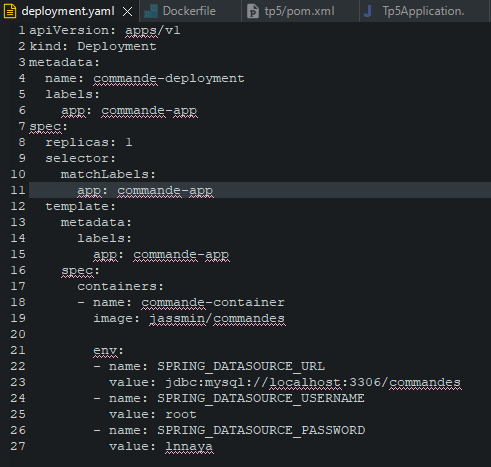


Maintenant, nous pouvons procéder au déploiement de deux manières : impérative et déclarative. La méthode impérative implique l'utilisation de lignes de commande, par exemple :

C:\Users\yasmine\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\Capture d'écran 2024-05-12 194141.jpeg

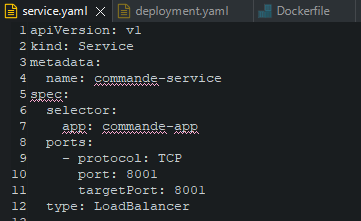
Kubernetes utilise kubectl (Kubernetes Control) pour interagir avec le cluster. Donc, dans l'exemple ci-dessus, lorsque nous parlons de déploiement de manière impérative, nous faisons référence à l'utilisation de kubectl pour envoyer des commandes directes au cluster Kubernetes.

La méthode déclarative, nous spécifions l'état désiré du système dans un fichier YAML, comme dans les exemples ci-dessous



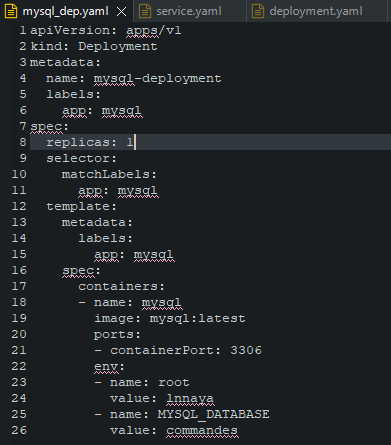
Ce fichier YAML décrit un déploiement dans Kubernetes. Voici la fonctionnalité de chaque section :

* **apiVersion**: Spécifie la version de l'API Kubernetes utilisée.
* **kind**: Définit le type de ressource Kubernetes, dans ce cas, un déploiement.
* **metadata**: Contient des métadonnées sur le déploiement, telles que son nom et ses libellés.
* **spec**: Spécifie les détails du déploiement, y compris le nombre de réplicas, le sélecteur pour identifier les pods cibles et le modèle pour les pods.
* **replicas**: Indique le nombre de répliques souhaité pour ce déploiement.
* **selector**: Définit comment les pods gérés par ce déploiement sont identifiés.
* **matchLabels**: Spécifie les libellés que les pods doivent avoir pour être gérés par ce déploiement.
* **template**: Fournit un modèle pour les pods créés par ce déploiement.
* **metadata**: Métadonnées spécifiques au pod.
* **spec**: Spécifie les spécifications du pod, y compris les conteneurs qu'il doit exécuter.
* **containers**: Liste des conteneurs à exécuter dans le pod.
* **name**: Le nom du conteneur.
* **image**: L'image Docker à utiliser pour ce conteneur.
* **env**: Les variables d'environnement à définir dans le conteneur.



Ce fichier YAML décrit un service dans Kubernetes. Voici la fonctionnalité de chaque section:

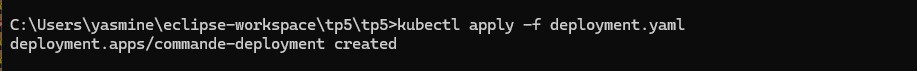
* **apiVersion**: Spécifie la version de l'API Kubernetes utilisée.
* **kind**: Définit le type de ressource Kubernetes, dans ce cas, un service.
* **metadata**: Contient des métadonnées sur le service, telles que son nom.
* **spec**: Spécifie les détails du service.
* **selector**: Définit comment le service trouve les pods à exposer.
* **app: commande-app**: Sélectionne les pods ayant le libellé **app: commande-app**.
* **ports**: Définit les ports sur lesquels le service écoute et redirige le trafic vers les pods.
* **protocol**: Spécifie le protocole utilisé (dans ce cas, TCP).
* **port**: Le port sur lequel le service écoute.
* **targetPort**: Le port sur lequel les pods ciblés reçoivent le trafic.
* **type**: Spécifie le type de service. Dans ce cas, c'est un service de type LoadBalancer, ce qui signifie qu'un équilibreur de charge externe sera provisionné pour exposer le service à l'extérieur du cluster Kubernetes.

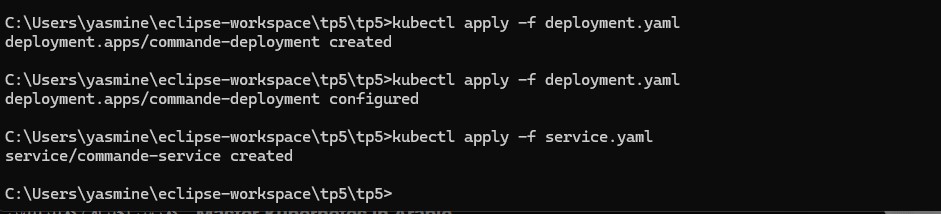


Ce fichier YAML décrit un déploiement dans Kubernetes pour une base de données MySQL. Voici la fonctionnalité de chaque section :

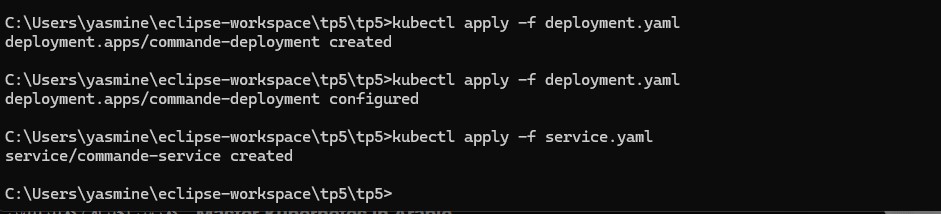
* **apiVersion**: Spécifie la version de l'API Kubernetes utilisée.
* **kind**: Définit le type de ressource Kubernetes, dans ce cas, un déploiement.
* **metadata**: Contient des métadonnées sur le déploiement, telles que son nom.
* **spec**: Spécifie les détails du déploiement.
* **replicas**: Indique le nombre de réplicas (instances) du pod à exécuter.
* **selector**: Définit comment le déploiement sélectionne les pods qu'il gère.
* **matchLabels**: Sélectionne les pods ayant le libellé **app: mysql**.
* **template**: Spécifie le modèle de pod à créer.
* **metadata**: Contient des métadonnées sur le pod, telles que ses libellés.
* **spec**: Spécifie les spécifications du pod.
* **containers**: Liste des conteneurs à exécuter dans le pod.
* **name**: Nom du conteneur.
* **image**: Image Docker à utiliser pour le conteneur, dans ce cas, **mysql:latest**.
* **ports**: Ports exposés par le conteneur.
* **containerPort**: Port sur lequel le conteneur écoute, dans ce cas, **3306** pour MySQL.
* **env**: Variables d'environnement à définir dans le conteneur.
* **name**: Nom de la variable d'environnement.
* **value**: Valeur de la variable d'environnement.
* **root**: Identifiant de l'utilisateur root MySQL.
* **MYSQL\_DATABASE**: Nom de la base de données MySQL à créer, dans ce cas, **commandes**

Pour exécuter ce déploiement dans Kubernetes, vous pouvez utiliser la commande kubectl apply avec le fichier YAML du déploiement. Voici comment faire :

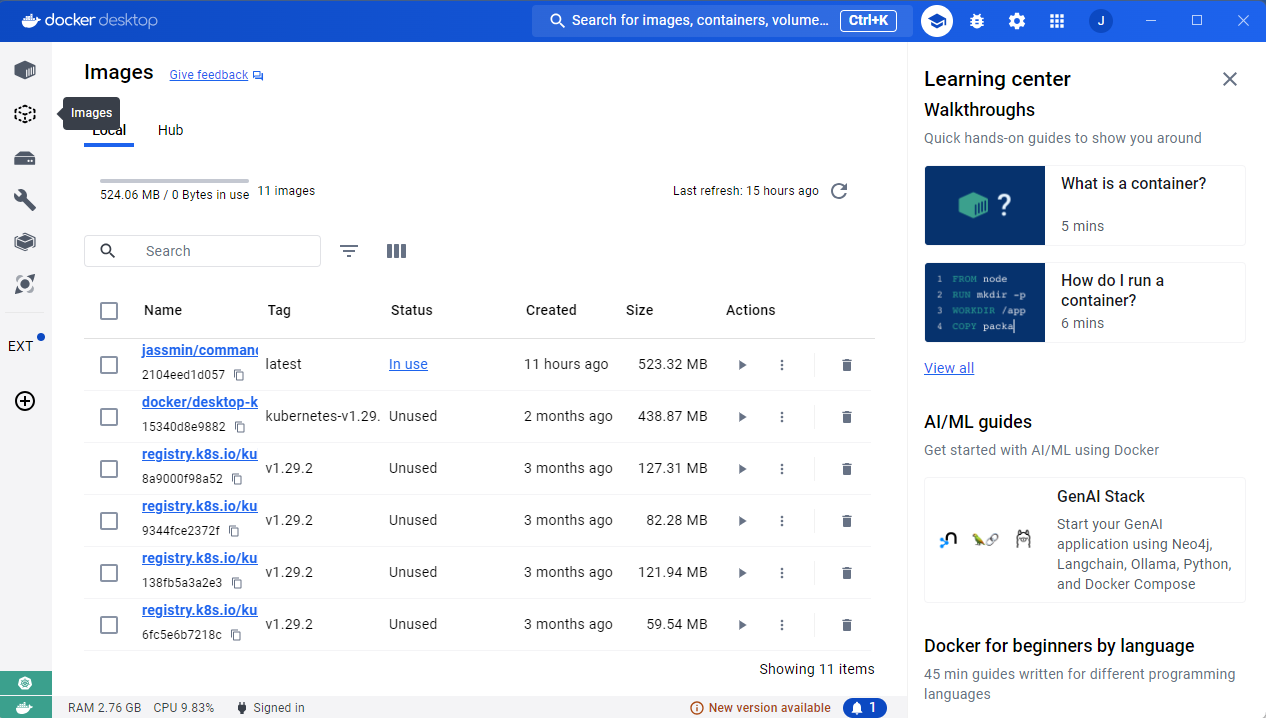




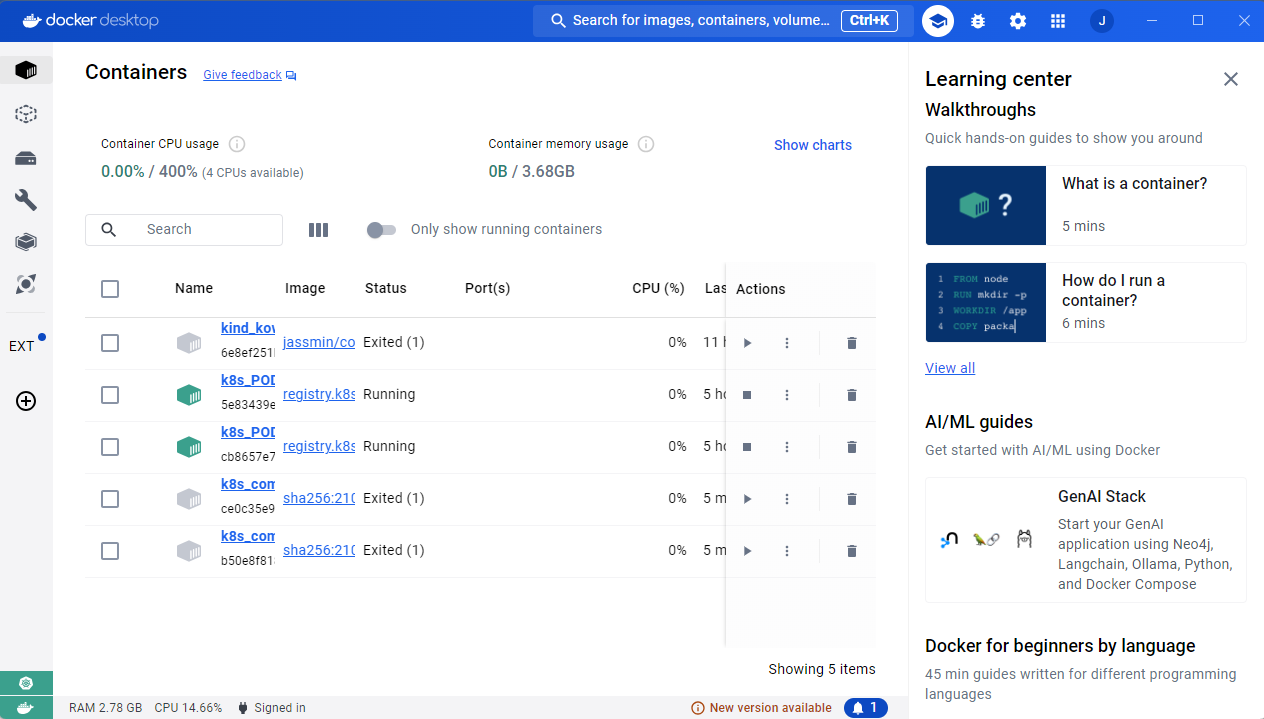
Si vous modifiez le fichier YAML et souhaitez appliquer les modifications à votre déploiement, vous pouvez réexécuter la même commande **kubectl apply -f** avec le fichier modifié.



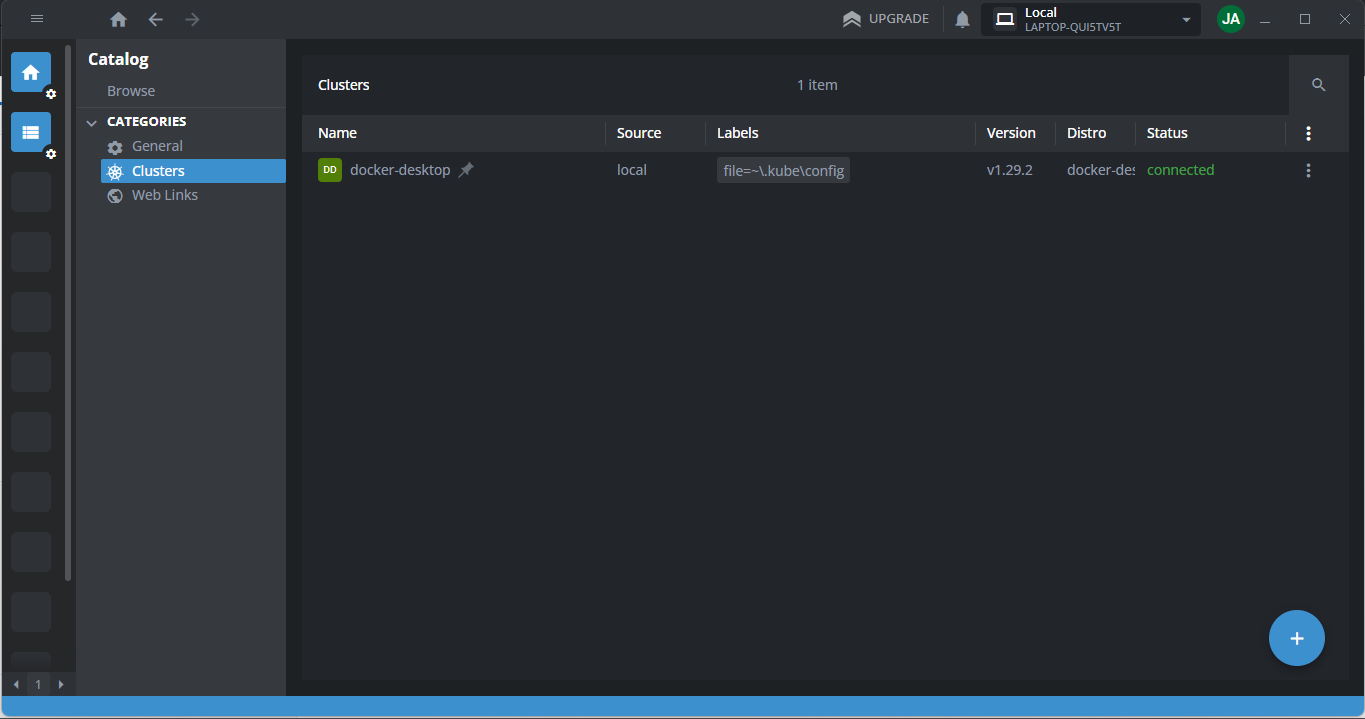
Voici notre interface Docker, plus précisément dans la section des images. C'est là que nous pouvons voir toutes les images disponibles dans notre environnement Docker. Nous utilisons cette interface pour gérer et interagir avec les images Docker, y compris les construire, les exécuter et les partager avec d'autres utilisateurs. Les images Docker sont essentielles pour le déploiement et la gestion des conteneurs, car elles contiennent toutes les dépendances nécessaires pour exécuter une application ou un service de manière isolée et portable.

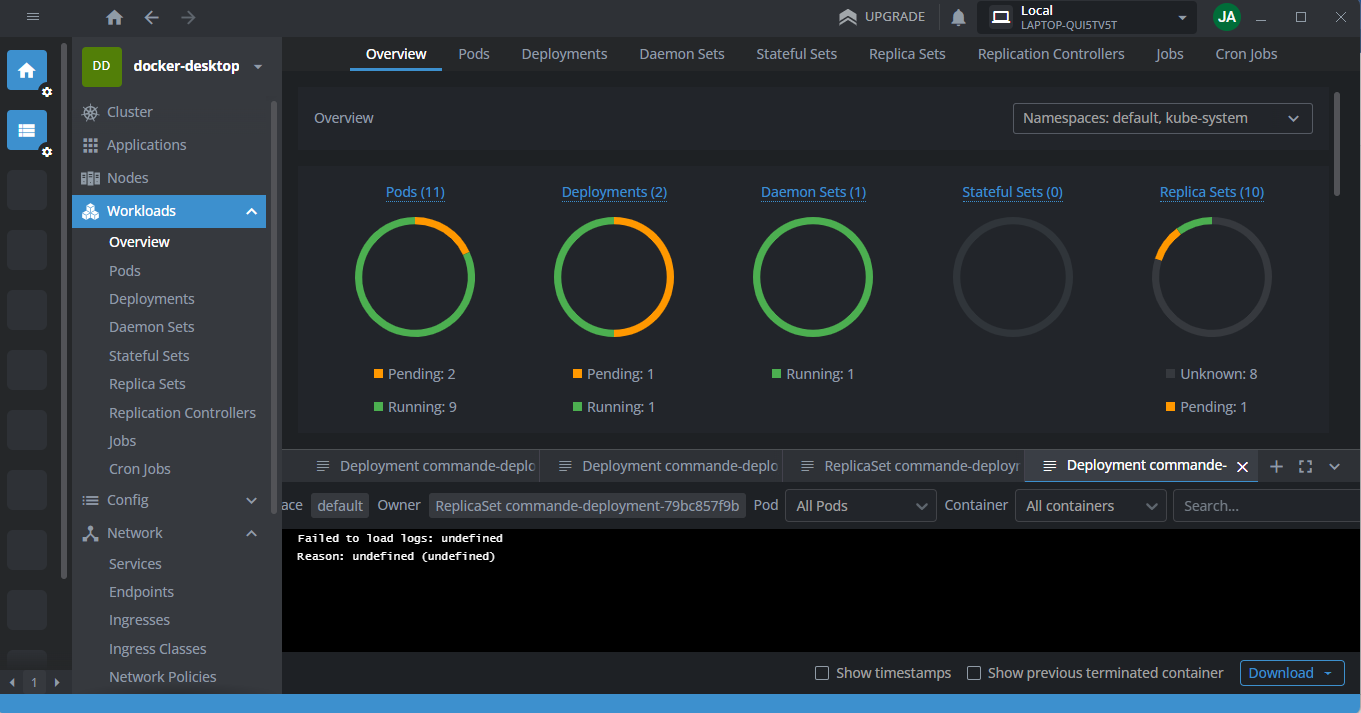


Voici l'interface des conteneurs dans Docker. Les conteneurs sont des instances en cours d'exécution d'images Docker. Cette interface nous permet de visualiser tous les conteneurs actuellement en cours d'exécution sur notre système. À partir d'ici, nous pouvons surveiller l'état et les performances des conteneurs, ainsi que gérer leur cycle de vie, en les démarrant, en les arrêtant ou en les supprimant au besoin. Les conteneurs Docker offrent une isolation légère et portable, ce qui les rend idéaux pour déployer des applications de manière cohérente et efficace.

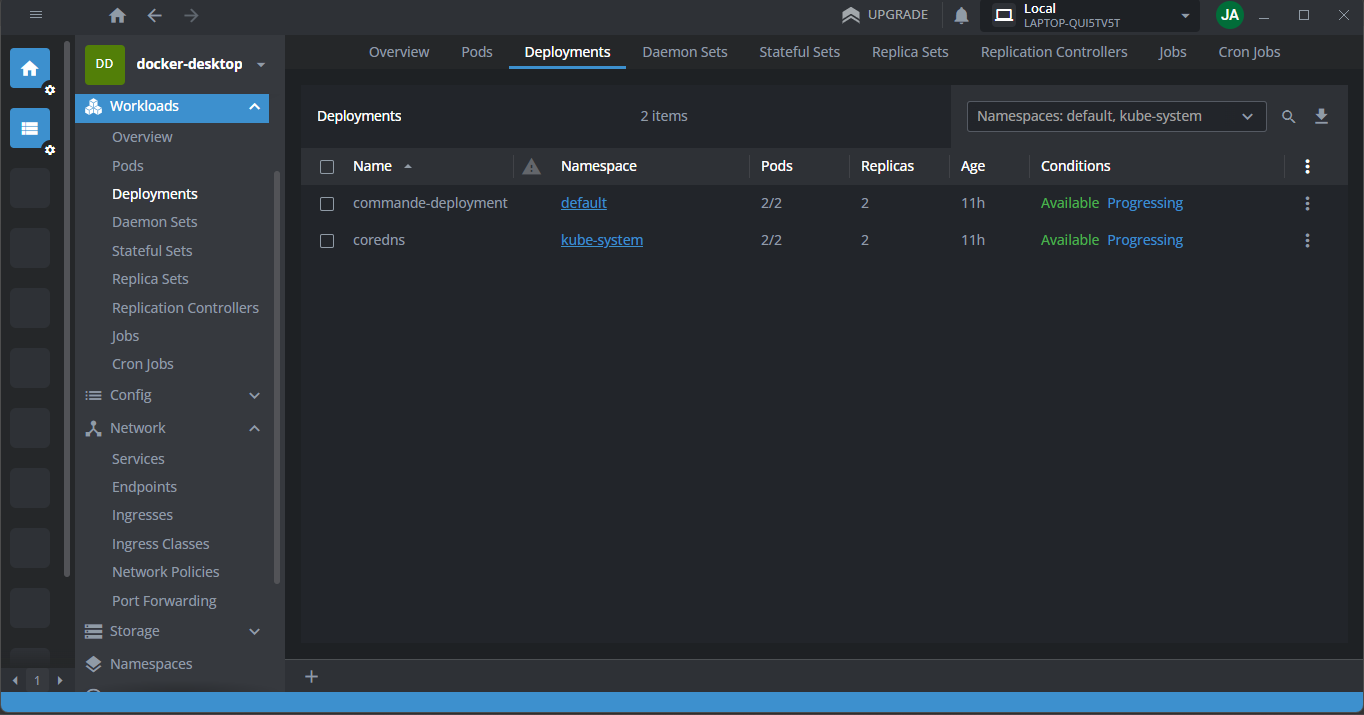


Voici notre interface Lens Kubernetes, un outil essentiel pour la gestion de nos déploiements. Il est connecté avec notre docker-desktop :

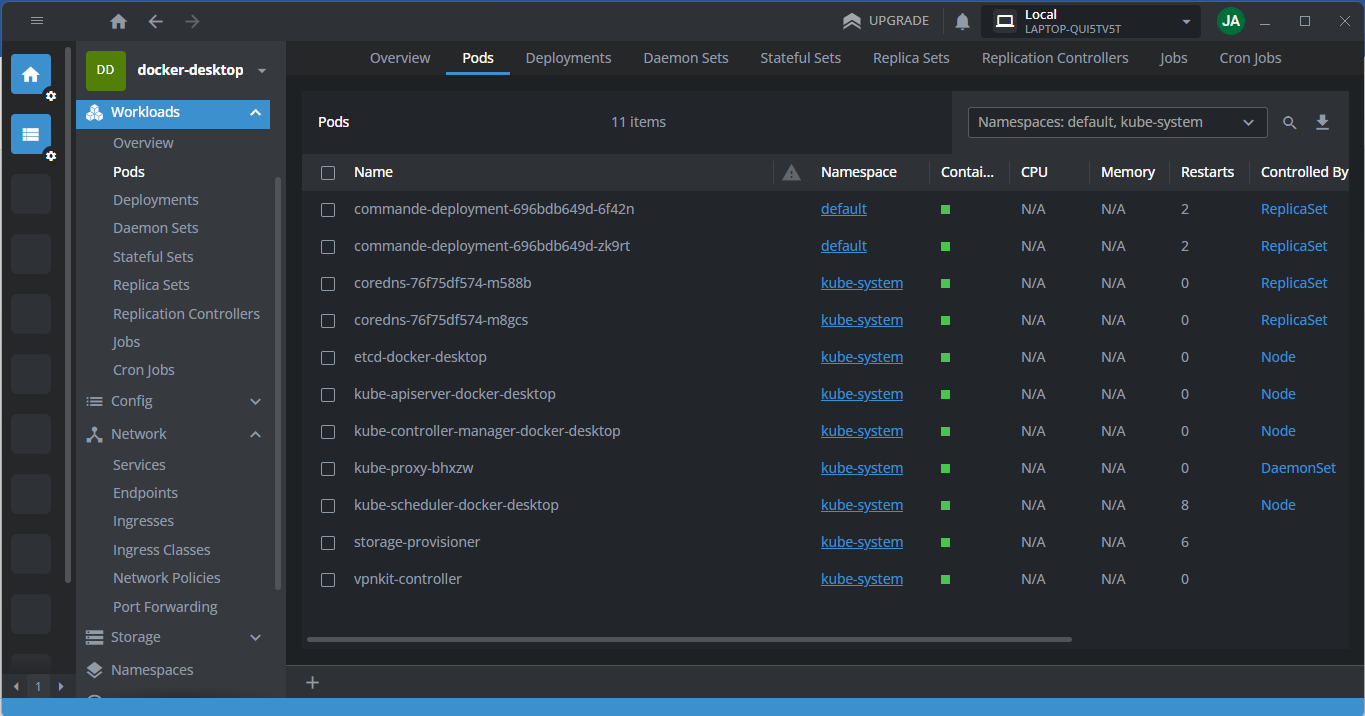
Grâce à cette interface, nous avons un aperçu détaillé des statistiques de nos pods en cours d'exécution, de nos déploiements, de nos ensembles de démons, de nos ensembles étatiques et de nos ensembles de réplicas. Cela nous permet de surveiller de manière efficace l'état et les performances de notre infrastructure Kubernetes, ce qui est crucial pour maintenir un environnement de production stable et fiable.



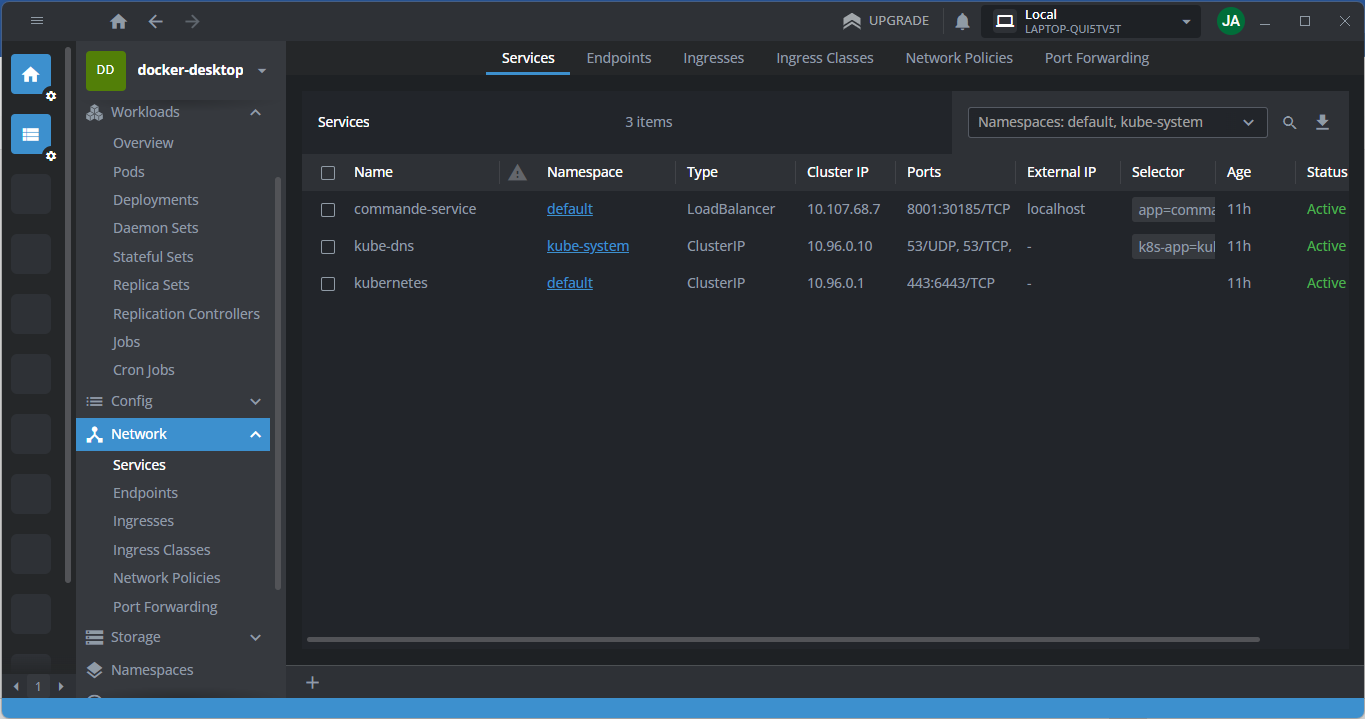
L'interface des deployments nous permet de gérer le cycle de vie de nos applications Kubernetes. Nous pouvons surveiller l'état de déploiement de nos applications, effectuer des mises à jour, ajuster le nombre de répliques et effectuer d'autres opérations de gestion. C'est un élément clé pour assurer le bon fonctionnement de nos services dans le cluster Kubernetes.



L'interface des pods nous permet de surveiller l'état et les performances de chaque pod dans notre cluster Kubernetes. C'est un outil précieux pour diagnostiquer les problèmes et maintenir nos applications en bonne santé.



L'interface des services dans Kubernetes nous permet de gérer l'exposition de nos applications au sein du cluster et à l'extérieur. Nous pouvons configurer les règles de routage du trafic vers les pods, définir les types de services (ClusterIP, NodePort, LoadBalancer, etc.), et gérer les équilibreurs de charge si nécessaire. Cela nous permet de fournir un accès contrôlé et sécurisé à nos applications déployées.Haut du formulaire



Pour vérifier l'exécution de notre application, nous pouvons consulter les journaux (logs) des pods correspondants. Cela nous permet de voir les messages de journalisation produits par notre application en cours d'exécution.

