Institut Spécialisé de Technologie
Appliquée de Taourirt

Direction Régionale de l'Oriental

Créer une application cloud native

Réalisé par : Hajar ZAIDI

Année de formation : 2022 - 2023

Définition du cloud :

Le terme « Cloud » désigne les serveurs accessibles sur Internet ainsi que les logiciels et bases de données qui fonctionnement sur ces serveurs.

Les avantages du cloud :

- Faible coût et disponibilité continue
- Maintenance allégée et automatisée
- Les employés peuvent travailler de n'importe où.
- Optimisation des ressources
- Hébergement d'applications et de services
- La flexibilité

Les caractéristiques du cloud :

- Le service doit être en libre-service à la demande
- Le service doit être mesurable
- Il doit y avoir une mutualisation des ressources
- Il doit être rapidement élastique

Exemples des fournisseurs cloud :

Microsoft, Amazon, IBM, Google cloud, Oracle …

Définition du serveur informatique :

Un serveur informatique relie un post jouant le rôle de serveur à différents postes utilisateurs (postes clients) et met ces derniers en réseau

Les services d'un serveur informatique :

- Le courrier électronique
- L'accès à Internet
- Le partage de fichier
- Le partage d'imprimantes
- Le stockage en base de données
- La mise à disposition d'applications

Les limites du serveur informatique :

- Pannes matérielles
- Infestation ou piratage des données
- Une capacité de stockage limitée
- Des coûts élevés pour l'entreprise

Cloud public :

Les Cloud publics sont généralement des environnements cloud crées à partir d'une infrastructure informatique qui n'appartient pas à l'utilisateur final.

Exemples : Alibaba cloud, Microsoft Azure, Google cloud, AWS, IBM …

Cloud privé :

Les cloud privés sont généralement définis comme des environnements cloud spécifiques à un utilisateur final ou à un groupe, et sont habituellement exécutés derrière le pare-feu de l'utilisateur ou du groupe.

Types du cloud privé :

- Cloud privés gérés : ce type de cloud est crée et utilisé par les clients, tandis qu'il est déployé, configuré et géré par un fournisseur tier.
- Cloud dédiées : il s'agit d'un cloud au sein d'un autre cloud

Cloud hybride :

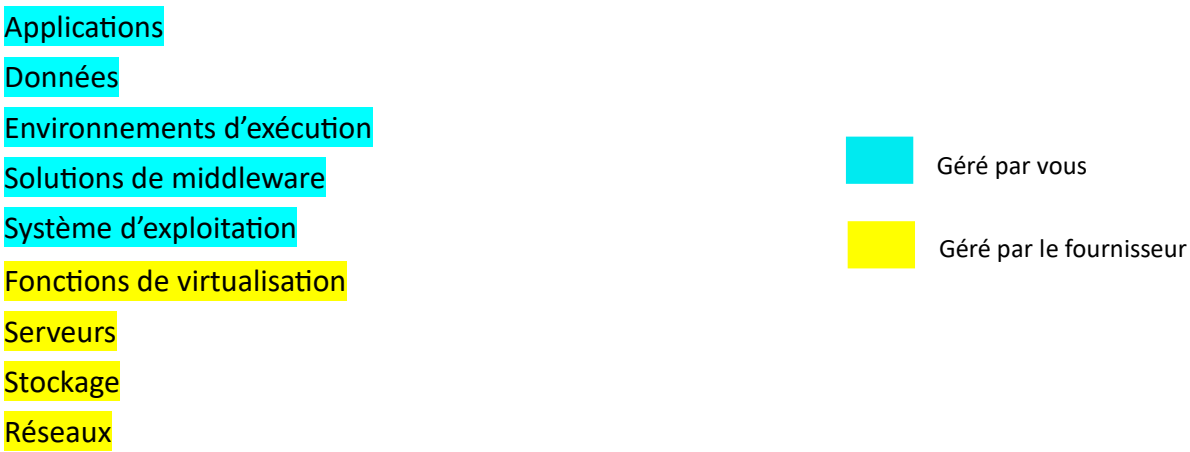
Un cloud hybride fonctionne comme un environnement informatique unique crée à partir de plusieurs environnements connectés via des réseaux locaux (LAN), des réseaux étendus (WAN), des réseaux privés virtuels (VPN) et / ou des APIs.

As-a-Service :

L'expression AaS ou As-a-Service signifie généralement qu'un tier se charge de vous fournir un service de cloud computing.

Types As-a-Service :

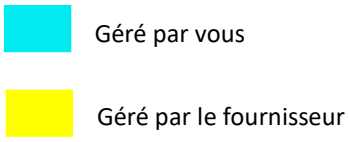
- IaaS : Infrastructure as a Service :



Exemples : Elastic Compute Cloud, Compute Engine, Virtual Machine

- PaaS : Platform as a Service :

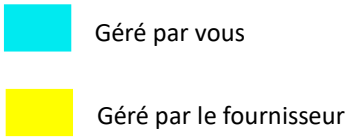
Applications
Données
Environnements d'exécution
Solutions de middleware
Système d'exploitation
Fonctions de virtualisation
Serveurs
Stockage
Réseaux



Exemples : AWS Elastic Beanstalk, Google App Engine, Azure App Service, Azure function App.

- SaaS : Software as a Service :

Applications
Données
Environnements d'exécution
Solutions de middleware
Système d'exploitation
Fonctions de virtualisation
Serveurs
Stockage
Réseaux



Exemples : Zoom, Google Apps, Microsoft Office 365 …

Définition du cloud native :

Le Cloud Native décrit une approche de développement logiciel dans laquelle les applications sont dès le début conçues pour une utilisation sur le Cloud.

Les 4 piliers du cloud native :

- Du côté technique, on trouve les micro-services et les conteneurs. Les différents micro-services remplissent une fonction précise et peuvent être rassemblés dans un conteneur avec tout ce qui est nécessaire à leur exécution.
- Du côté de la stratégie, les processus de développement et la livraison continue sont bien établis. L'équipe de développeurs ajoute à un micro-service certaines fonctionnalités livrées automatiquement par des processus de la livraison continue.

Les avantages du cloud native :

- La flexibilité : les modifications apportées au code n'ont pas d'impact sur le logiciel dans son ensemble.
- L'évolutivité : éviter la mise à niveau coûteuse du matériel en cas d'augmentation des exigences pour un service.
- L'automatisation : qui permet d'éliminer les erreurs.
- La vitesse et l'agilité : répondre rapidement aux conditions du marché.

Définition de l'architecture monolithique :

Dans une architecture monolithique, l'application est développée en un seul bloc, avec une même technologie et déployée dans un serveur d'application.

Les inconvénients de l'architecture monolithique :

- La complication du déploiement
- La scalabilité non optimisée

Définition de l'architecture micro-services :

L'architecture micro-services consiste à décomposer l'application en un ensemble de petites services indépendantes appelés micro-services. Chaque micro-service possède son propre logique et sa propre base de données.

Les avantages de l'architecture micro-services :

- L'agilité technologique : le choix technologique dans une architecture micro-services peut être adapté aux besoins spécifiques de chaque micro-service
- Un déploiement continue et rapide : grâce à l'utilisation des conteneurs
- La scalabilité : selon le trafic du système, on peut ajouter des instances d'un micro-service ou en détruire.

Les inconvénients de l'architecture micro-services :

- La communication entre les services est complexe
- Plus de services équivaut à plus de ressources
- Les tests globaux sont difficiles
- Les problèmes de débogage peuvent être plus difficiles

Architecture micro-services ou monolithique :

- Déploiement fréquent : architecture micro-service
- Déploiement une seule fois : architecture monolithique

Types de communication entre services :

- Appel synchrone point à point : un service appelle un autre service et attend la réponse pour continuer son processus.
- Appel asynchrone point à point : un service appelle un autre service et continue son processus sans attendre la réponse. Ce type de communication est utilisé lorsqu'un service désire envoyer un message à un autre service, et peut être implémenté en utilisant un protocole de transport de messages (AMQP).
- Diffusion d'événements : un service envoie une notification aux autres services sans avoir aucune idée des services écoutant cet événement, et il n'attend pas de réponse.

Définition des passerelles API :

Les passerelles API constituent la couche intermédiaire entre les micro-services et les applications clientes qui ont besoin des micro-services pour fonctionner.

Le rôle des passerelles API :

- Servent de point d'entrée unique au système en exposant que les points de terminaison requis.
- Équilibrent la charge des demandes des clients en les répartissant sur plusieurs instances d'un service.

Les raisons de créer des micro-services avec NodeJS :

- Améliorer le temps d'exécution
- Se bénéficier d'une architecture événementielle.
- Se bénéficier des appels non bloquants.
- L'évolutivité et la mise à l'échelle.

Définition de RabbitMQ :

RabbitMQ est un système de messagerie open-source basé sur le modèle de message broker. Il agit comme un intermédiaire entre les applications distribuées, leur permettant d'échanger des messages de manière fiable et asynchrone.

Les mots clés dans RabbitMQ :

- Message broker : est un modèle de message, qui permet l'envoi, la réception, et la mise en file d'attente de messages entre différentes applications.
- Protocole AMQP (Advanced Message Queuing Protocol) : est un protocole de messagerie qui permet la communication entre les applications clientes et le serveur RabbitMQ.
- Producteurs (Publishers) : sont des applications ou des composants qui envoient des messages à RabbitMQ pour qu'ils soient traités et distribués.

- Echanges (Exchanges) : sont des unités dans RabbitMQ qui reçoivent les messages des producteurs et les routent vers les files d'attente appropriées en fonction des règles de routage définies.
- Liaisons (binding) : relient les échanges aux files d'attente et définissent les règles de routage pour acheminer les messages.
- Files d'attente (Queues) : sont des structures de données FIFO (premier entré, premier sorti), utilisé dans RabbitMQ pour stocker les messages envoyés par les producteurs, en attendant qu'ils soient consommés par les consommateurs.
- Consommateurs (consumers) : sont des applications ou des composants qui se connectent à RabbitMQ et récupèrent les messages à partir des files d'attentes pour les traiter.
- Routage des messages : après la réception des messages des producteurs par les échanges, ces derniers les routent vers les files d'attente, qui permettent le stockage de ces messages en attendant qu'ils soient récupérés par les consommateurs. Les échanges et les files d'attente sont reliées par les liaisons, qui spécifient quelles files d'attentes recevront les messages en fonction des critères de routage spécifiques tels que les clés de routage, les types d'échanges, et les liaisons avec les files d'attente. Le routage peut être un routage direct (envoie le message à la queue avec la clé de routage exacte du message), un routage par sujet (utilise des expressions de routage pour acheminer les messages à une ou plusieurs queues correspondantes), un routage par fanout (envoie le message à toutes les queues liées à l'échange, sans utiliser de clé de routage), ou un routage par header (utilise des en-têtes de message pour acheminer les messages à une ou plusieurs queues correspondantes).

Les avantages de RabbitMQ :

- La durabilité : les données ne seront pas perdues en cas de pannes du serveur ou de redémarrage.
- Haut disponibilité et mise en cluster : plusieurs nœuds RabbitMQ travaillent ensemble pour assurer la fiabilité et la disponibilité des messages.

Exemple pratique RabbitMQ :

1) Initialisation du projet Node.js et installation la bibliothèque amqplib (un client RabbitMQ pour Node.js) :

```
mkdir tp1-rabbitmq
cd tp1-rabbitmq
npm init -y
npm install amqplib
```

2) Configuration de RabbitMQ. Par défaut, RabbitMQ écoute sur le port 5672, vous pouvez utiliser les paramètres par défaut ou modifier les informations de connexion dans le code ci-dessous.

3) Création de l'émetteur (Publisher) :
Dans un fichier publisher.js :

```
const amqp = require('amqplib');
async function run() {
try {
// Se connecter au serveur RabbitMQ
const connection = await amqp.connect('amqp://localhost');
 // Créer un canal
const channel = await connection.createChannel();
// Déclarer une file
const queueName = 'test_queue';
await channel.assertQueue(queueName, { durable: false });
// Envoyer un message
const message = 'Hello, from ofppt RabbitMQ!';
channel.sendToQueue (queueName, Buffer.from (message));
console.log("Message envoyé:", message);
// Fermer la connexion
await channel.close();
await connection.close();
} catch (error) {
console.error(error);
}
}
run();
```

Ce code se connecte à RabbitMQ, crée un canal, déclare une file (si elle n'existe pas déjà), puis envoie un message à la file.

4) Création du récepteur (Consumer) :
   Dans un fichier consumer.js :

```
const amqp = require('amqplib');
async function run() {
try {
// Se connecter au serveur RabbitMQ
const connection = await amqp.connect('amqp://localhost');
 // Créer un canal
const channel = await connection.createChannel();
// Déclarer une file
const queueName = 'test_queue';
await channel.assertQueue(queueName, { durable: false });
// Définir une fonction de rappel pour traiter les messages reçus
channel.consume (queueName, (message) => {
console.log("Message reçus :", message.content.toString());
}, {noAck : true}) ;
Console.log('En attente de messages …');
} catch (error) {
console.error(error);
}
}
run();
```

   Ce code se connecte également à RabbitMQ, crée un canal, déclare une file (assurez-vous d'utiliser le même nom de file que dans l'émetteur) et définit une fonction de rappel pour traiter les messages reçus.

5) Exécution, dans une invite de commandes, on exécute le fichier publisher.js et dans une autre, on exécute le fichier consumer.js.
   Vous devriez voir le message "Hello, RabbitMQ!" apparaître dans la console du consommateur.

Définition de Azure :

Azure est la plateforme de cloud computing de Microsoft. Il s'agit d'un ensemble de services et de ressources informatiques disponibles à la demande via Internet. Il permet aux entreprises de déployer, gérer et exécuter diverses applications et charges de travail.

Les avantages de Azure :

- Évolutivité : faire face à des besoins variables en termes de ressources informatiques.
- Fiabilité : grâce à la réplication des données et à la redondance des services sur des centres de données multiples.

- Sécurité : des fonctionnalités de sécurité avancées pour protéger les données et les applications.
- Flexibilité : création et déploiement des applications hybrides.
- Tarification flexible : payer uniquement pour les ressources réellement utilisées.
- Intégration avec les outils Microsoft.
- Large gamme de services.

Définition de YAML :

YAML (YAML Ain't Markup Language) est un langage de sérialisation de données lisible par l'homme utilisé pour représenter des données structurées.

Configuration de l'application web :

- La section "database" contient les informations de connexion à la base de données.
- La section "server" contient les paramètres du serveur web.
- La section "logging" définit les paramètres de journalisation.
- La section "features" est une liste des fonctionnalités activées pour l'application.

Exemple :

database :	server :	logging :	features :
driver: mysql host: localhost port: 3306 username: myuser password: mypassword dbname: mydatabase	host: localhost port: 8080 timeout: 30	level: info file: logs/app.log	- authentication - analytics

Définition de Kubernetes :

Kubernetes (K8S) est une plateforme open-source d'orchestration de conteneurs qui permet de déployer, gérer et mettre à l'échelle des applications conteneurisées de manière efficace.

Définition du cluster Kubernetes :

Un cluster Kubernetes est un ensemble de machines (les nœuds) qui permettent d'exécuter des applications conteneurisées.

Définition de Apache Kafka :

Apache Kafka est une plateforme distribuée de diffusion de données en continu, capable de publier, stocker, traiter et souscrire à des flux d'enregistrement en temps réel.

Différence entre Kubertnetes et AKS :

- Kubernetes (K8S) est une plateforme open-source d'orchestration de conteneurs qui permet de déployer, gérer et mettre à l'échelle des applications conteneurisées de manière efficace. Tandis que Azure Kubernetes Service (AKS) est un service managé par Microsoft Azure qui simplifie le déploiement et la gestion de clusters Kubernetes sur Azure.
- Kubernetes offre une plus grande flexibilité et un contrôle direct sur l'infrastructure, tandis que AKS est conçu pour simplifier la gestion de Kubernetes spécifiquement sur Azure.

Avantages de AKS :

- Scalabilité : AKS peut augmenter ou réduire automatiquement le nombre de nœuds de calcul en fonction de la demande.
- Fiabilité : AKS est un service géré par Microsoft Azure, ce qui signifie que Microsoft se charge de la gestion des opérations sous-jacentes.
- Sécurité : AKS fournit des fonctionnalités avancées de sécurité pour les applications conteneurisées.
- Portabilité : les applications déployées sur AKS peuvent être facilement déplacées vers d'autres environnements Kubernetes.
- Intégration avec l'écosystème Azure.

Architecture AKS :

- Cluster AKS : un groupe de machines virtuelles (VM) appelées nœuds qui exécutent les conteneurs.
- Nœuds : sont des machines virtuelles qui exécutent les conteneurs dans le cluster AKS. Chaque nœud contient un agent Kubernetes qui communique avec le contrôleur de cluster pour la gestion des applications et des tâches de maintenance.
- API Kubernetes : un moyen permettant d'interagir avec le cluster AKS.
- Contrôleur de cluster : Le contrôleur de cluster est responsable de la gestion de l'état global du cluster AKS.
- Azure Container Registry (ACR) : ACR est un registre de conteneurs dans le cloud qui permet de stocker et de gérer les images de conteneurs.
- Réseau virtuel : permet d'isoler les ressources du cluster AKS du reste de votre réseau Azure. Le VNet permet également d'établir des connexions sécurisées entre les nœuds du cluster et d'autres ressources Azure.
- Services Azure : AKS peut être intégré à d'autres services Azure pour créer des applications plus complexes.

Terminologie AKS :

- Pools : Dans AKS, un pool (ou node pool) est un groupe de nœuds homogènes au sein du cluster.
- Nodes : Les nœuds (nodes) sont des machines virtuelles au sein du cluster AKS qui exécutent les conteneurs.
- Pods : Un pod est l'unité de base dans Kubernetes. Il représente un groupe d'un ou plusieurs conteneurs qui sont déployés ensemble sur un même nœud.
- Conteneur (Container) : Un conteneur est une instance d'une image de conteneur qui contient une application et toutes ses dépendances nécessaires pour s'exécuter de manière isolée.
- Deployment : est un objet Kubernetes qui définit comment une application doit être déployée et mise à l'échelle dans un cluster.
- Manifest : est un fichier YAML ou JSON qui décrit l'état désiré d'un objet Kubernetes.

Définition de Kubectl :

Kubectl est un outil en ligne de commande largement utilisé pour interagir avec les clusters Kubernetes.

Définition d'un pipeline :

Un pipeline (ou pipeline de livraison continue) fait référence à un processus automatisé qui permet de compiler, tester et déployer de manière continue des applications et des infrastructures cloud.

Définition de Azure pipelines :

Azure Pipelines est un service spécifique de la plateforme Azure qui facilite la création, la gestion et l'exécution de pipelines de livraison continue.

Le fichier .gitlab-ci.yml :

Le fichier .gitlab-ci.yml est un fichier de configuration utilisé pour effectuer la configuration d'un pipeline CI (Continuous Integration) dans GitLab CI. Il contient les instructions nécessaires pour définir les étapes du pipeline et les actions à effectuer lors de l'intégration continue.

Exemple : configuration de base pour un pipeline CI dans GitLab CI

```
# Définition des étapes du pipeline
stages:
- build
- test
- deploy
# Définition des jobs (actions) pour chaque étape
build_job:
stage: build
script:
- echo "Building the application..."
test_job:
stage: test
script:
- echo "Running tests..."
deploy_job:
stage: deploy
script:
- echo "Deploying the application..."
```

Les étapes Pour configurer un pipeline CI dans GitLab CI :

1) Créez un fichier .gitlab-ci.yml à la racine de votre dépôt Git.
2) Ajoutez les étapes (stages) que vous souhaitez inclure dans votre pipeline.
3) Définissez les jobs (jobs) pour chaque étape en spécifiant le nom du job, l'étape à laquelle il appartient et les commandes ou scripts à exécuter.
4) Commitez et poussez votre fichier .gitlab-ci.yml dans votre dépôt GitLab.

GitLab CI détectera automatiquement le fichier de configuration et démarrera le pipeline CI lors des pushs suivants.

Définition de la conteneurisation :

La conteneurisation est un type de virtualisation, qui consiste à rassembler le code du logiciel et tous ses composants de manière à les isoler dans leur propre « conteneur ».

Définition de Docker :

Docker est une plateforme qui permet de créer facilement des conteneurs et des applications basées sur les conteneurs.

Définition de la machine virtuelle :

Une machine virtuelle est un environnement entièrement virtualisé qui fonctionne sur une machine physique.

La différence entre la machine virtuelle et le conteneur :

- Les applications conteneurisées utilisent moins de ressources que les machines virtuelles et réduisent la pression sur la mémoire de l'hôte.
- Les conteneurs sont plus compacts que les machines virtuelles et démarrent plus rapidement.

Avantages des conteneurs :

- La portabilité : les conteneurs peuvent s'exécuter sur n'importe quelle plateforme ou cloud.
- L'isolation : les conteneurs fonctionnent d'une manière indépendante.
- Mise à l'échelle flexible : le nombre des conteneurs démarrés est en fonction du nombre d'utilisateurs.
- La facilité de gestion : grâce à l'utilisation des plateformes d'orchestration des conteneurs.
- La vitesse.

Terminologies Docker :

Les concepts clés :

- Moteur Docker (Engine) : permet de créer, exécuter, et de gérer des conteneurs Docker. En tant que cœur du système Docker, il réunit tous les composants de la plateforme en un seul endroit.
- Docker Daemon : le serveur Docker écoute les requêtes de l'API Docker et gère les objets Docker tels que les images, les conteneurs, les réseaux, et les volumes.
- Client Docker : il s'agit de l'interface utilisateur principale pour communiquer avec le système Docker. Il accepte les commandes via l'interface de ligne de commande et les envoie au démon Docker.
- Register Docker : un système de catalogage pour héberger, pousser, et extraire des images Docker.
- Docker hub : un registre public que n'importe qui peut utiliser pour rechercher des messages.

Les objets Docker :

- Image Docker : un modèle en lecture seule utilisé pour créer des conteneurs Docker.
- Conteneur Docker : une instance d'une image qui permet d'exécuter un micro-service individuel ou une pile d'application complète.

Définition de Dockerfile :

Un Dockerfile est un fichier qui liste les instructions à exécuter pour construire une image.

Les instructions de bases dans un dockerfile :

- FROM NomImage : sert à spécifier l'image de base que vous allez utiliser, image qui est présente sur Docker Hub.
- RUN command : permet d'exécuter des commandes supplémentaires à l'intérieur du build du dockerfile. On peut s'en servir afin de télécharger et d'installer les dépendances nécessaires à l'application ou encore à directement afficher un résultat ou un message.
- COPY ou ADD : permet de copier des fichiers depuis notre machine locale vers le conteneur Docker.
- ENV : variables d'environnements utilisables dans le Dockerfile et dans le conteneur.
- EXPOSE : expose un port.
- ENTRYPOINT : c'est le point d'entrée de votre conteneur, en d'autres termes, c'est la commande qui sera toujours exécutée au démarrage du conteneur.
- CMD : spécifie les arguments qui seront envoyés au ENTRYPOINT.
- WORKDIR : définit le répertoire de travail qui sera utilisé pour le lancement des commandes CMD et/ou ENTRYPOINT et ça sera aussi le dossier courant lors du démarrage du conteneur.

Exemple simple de configuration d'une image à l'aide d'un Dockerfile :

1) Créer un fichier sans extention nommé Dockerfile et ajoutez le code suivant :

```
FROM alpine
CMD ["echo","Ceci est un test"]
```

2) Dans la ligne de commandes accédez au chemin du fichier et exécutez les commandes suivantes :

```
docker build .
```

Cette commande va nous permettre de créer l'image

```
docker build  --tag  NomTag  .
```

Cette commande va nous permettre de créer l'image et va attribuer un tag à cette image.

Définition de docker-compose :

Compose est un outil permettant de définir et d'exécuter des applications Docker multi-conteneurs. Docker-compose fonctionne à partir d'un fichier yml, dans lequel on définit tous les services que l'on souhaite.
Quand on lance la commande d'exécution, le Daemon Docker va lire le docker-compose.yml afin de monter chaque container avec les paramètres que l'on a choisi.

Configuration d'un service avec docker-compose :

1) Dans un fichier docker-compose.yml, on indique la version de docker-compose, et les services comme suit :

```
version : '3'
services :
nomService :
image : nomImageUtilisée
container_name : nomConteneur
stdin : true
tty : true
```

2) Pour interagir avec le conteneur lancé :
   docker exec -it idConteneur bash

Les commandes Docker :

- Pour afficher la version de Docker :

  docker –version

- Pour afficher tous les conteneurs :

  docker ps –all

  docker ps -a

  docker container ls -a

- Pour afficher les conteneurs actifs :

  docker container ls

  docker ps

- Pour créer un conteneur à partir d'une image déjà existée et l'exécuter :

  docker container run NomImage

- Pour créer un conteneur à partir d'une image (qui peut exister ou non) et l'exécuter :

  docker run NomImage

- Pour afficher tous les images :

  docker images

- Pour supprimer un conteneur :

  docker rm IdContainer

- Pour supprimer une image :

  docker image rm NomImage

- Pour attribuer un nom au conteneur :

  docker run –name NomAttribue NomImage

- Pour exécuter un conteneur en mode détachée :

  docker run –detach NomImage

  docker run -d NomImage

- Pour publier le port d'un conteneur sur l'hote :

  docker run –publish NomImage

  docker run -p NomImage

- Pour arreter un conteneur en cours d'exécution :

  docker stop IdContainer

- Pour relancer un conteneur :

  docker start NomContainer

- Pour ouvrir un shell dans un conteneur déjà en cours d'exécution :

  docker exec -it NomContainer bash

  docker exec -it IdContainer bash

  docker exec -it NomContainer sh

  docker exec -it IdContainer sh

- Pour afficher la version de nginx :

  service nginx -v

- Pour afficher le statut de nginx :

  service nginx statut

- Pour connecter votre compte docker :

  docker login

  VotreLogin

  VotreMotDePasse

- Pour mettre une image dans votre compte :

  docker push VotreLogin/NomImage:dev

- Pour ajouter un tag à votre image :

  docker tag NomImage VotreLogin/NomImage:dev

- Pour déconnecter votre compte :

  docker logout

Les commandes Docker-compose :

- Pour construire les images définies dans docker-compose.yml :
  docker-compose build
- Pour construire les images si elles ne le sont pas déjà, et démarrer les conteneurs :
  docker-compose up
- Pour démarrer en mode détaché (commande exécutée en arrière-plan du terminal), et construire les images avant le démarrage des conteneurs :
  docker-compose up (-d) (--build)
- Pour arrêter et supprimer l'ensemble des conteneurs qui ont été instanciés par le docker-compose :
  docker-compose down
- Pour arrêter (sans suppression) les conteneurs :
  docker-compose stop
- Pour afficher tous les conteneurs qui ont été lancés par docker-compose (qu'ils tournent actuellement ou non) :
  docker-compose ps
- Pour supprimer tous les conteneurs démarrés avec une commande docker-compose :
  docker-compose rm
- Pour valide la syntaxe du fichier docker-compose.yml :
  docker-compose config

Les commandes Kubectl :

- Pour vérifier la version de Kubectl :
  kubectl version
- Pour vérifier l'état du cluster Kubernetes :
  kubectl cluster-info
- Pour vérifier l'état des nœuds du cluster :
  kubectl get nodes
- Pour vérifier l'état des pods en cours d'exécution dans le cluster :
  kubectl get pods
- Pour afficher des informations détaillées sur un pod spécifique :
  kubectl describe pod <nom-du-pod>
- Pour exécuter une commande à l'intérieur d'un pod :
  kubectl exec -it <nom-du-pod> -- <commande>
- Pour afficher les logs d'un pod :
  kubectl logs <nom-du-pod>
- Pour créer un déploiement à partir d'un fichier de configuration YAML :
  kubectl create -f <fichier-de-configuration.yaml>
- Pour supprimer un déploiement :
  kubectl delete deployment <nom-du-deploiement>
- Pour mettre à l'échelle un déploiement :
  kubectl scale deployment <nom-du-deploiement> --replicas=<nombrede-replicas>
- Pour exposer un déploiement via un service :
  kubectl expose deployment <nom-du-deploiement> -- type=LoadBalancer --port=<port> --target-port=<port-cible>
- Pour appliquer une mise à jour sur un déploiement en remplaçant les pods existants :
  kubectl set image deployment/<nom-du-deploiement> <conteneur>=<image>:<version>
- Pour obtenir des informations sur les services du cluster :
  kubectl get services
- Pour appliquer des règles de mise à l'échelle automatique sur un déploiement :
  kubectl autoscale deployment <nom-du-deploiement> --min=<nombre minimum-replicas> --max=<nombre-maximum-replicas> --cpu percent=<pourcentage-utilisation-CPU>