جامعة الحسن الثاني بالدار البيضاء
UNIVERSITÉ HASSAN II DE CASABLANCA

FSBM
FACULTE DES SCIENCES BEN M'SICK
UNIVERSITE HASSAN II DE CASABLANCA

MODULE :

Le Cloud Computing & DevOps

Pr. F. Benabbou
Master DSBD
2024-2025
Faculté des Sciences Ben M'Sik Casablanca

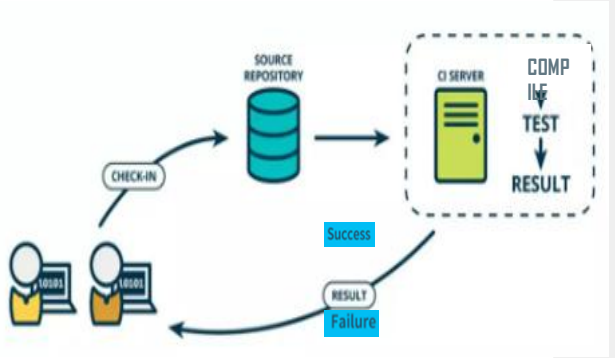
TABLE OF CONTENTS

02 CI/CD/CT

Fonctionnalités DevOps

Continue Integration (CI)

- L'intégration continue est une pratique de développement qui exige des développeurs qu'ils intègrent chaque modification du code dans un référentiel partagé (Repository) assez fréquemment et à exécuter automatiquement des tests pour valider ces modifications.
- La CI permet :
  - De détecter un stade précoce les problèmes logiciels et d'alerter les développeurs si leurs modifications ont été accepté ou ont entraîné un échec afin d'améliorer la qualité du logiciel.
  - De s'assurer qu'un code qui fonctionne bien sur la machine locale de Dev. fonctionnera bien sur d'autres
  - De faire en sorte que le code le plus récent soit toujours disponible pour tous les développeurs et les ingénieurs QA le plus rapidement possible



Etapes du processus CI

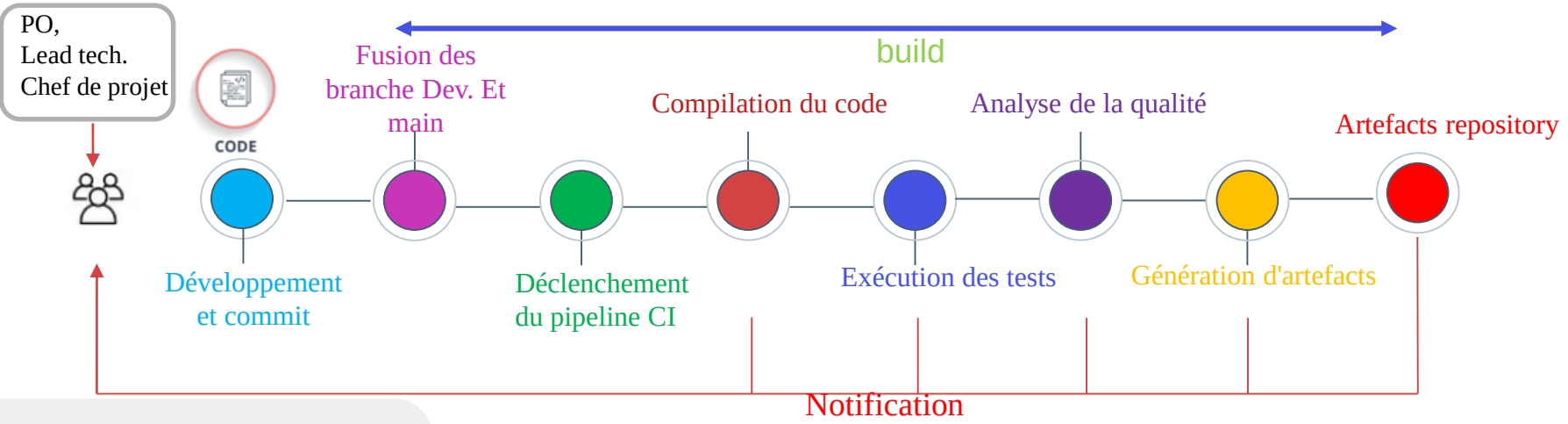
- **Développement et commit** : Les développeurs travaillent sur leurs fonctionnalités respectives; une fois une tâche est terminée, le développeur effectue un commit (enregistrement) de ses modifications dans le dépôt de code source (Git, SVN, etc.).
- **Déclenchement automatique du pipeline CI** : Chaque commit déclenche automatiquement l'exécution du pipeline CI qui consiste en une série d'étapes automatisées qui vont vérifier et valider les modifications.
- **Compilation du code** : Le code source est compilé pour générer le logiciel exécutable.
- **Exécution des tests** : Une suite de tests automatisés est exécutée pour vérifier que les nouvelles modifications n'introduisent pas de régressions (bugs) et que le logiciel fonctionne toujours comme prévu. Ces tests peuvent être :
    - *Unitaires* : Test des unités de code isolées (fonctions, classes).
    - *Intégration* : Test l'interaction entre différentes parties du système.
    - *Fonctionnels* : Test le comportement global du logiciel.

Etapes du processus CI

- **Analyse de la qualité du code** : Rechercher les vulnérabilités à l'aide de tests statiques, dynamiques, ou fuzz (aléatoire) et d'analyse des dépendances. Le but est de détecter :
  - les potentielles erreurs,
  - les problèmes de style de codage
  - et les vulnérabilités.
- **Génération d'artefacts** :
  - Si tous les tests réussissent et que la qualité du code est satisfaisante, cette étape aura pour objectifs de créer des **livrables** concrets qui seront déployés dans les différents environnements: **artefact**
  - Un artefact regroupe tous les éléments nécessaires (les applications et les dépendances) pour exécuter l'application dans un environnement donné.
  - Un artefact peut prendre différentes formes selon le type d'application et les technologies utilisées : 1) un fichier exécutable: Un fichier JAR, WAR, EXE, etc.; 2) une image de conteneur; 3) un package d'installation: Un RPM, DEB, etc.; 4) un bundle de fichiers: un ensemble de fichiers compressés (zip, tar.gz) contenant les ressources de l'application.
  - Ces artefacts vont être utilisés dans les étapes suivantes du processus de **livraison** continue.

Continue Integration (CI)

- **Notification :** Les développeurs sont notifiés du résultat du pipeline CI, en cas d'échec, ils sont informés des problèmes à résoudre.
- **Signature numérique**: L'artefact peut être signé numériquement pour garantir son intégrité et son authenticité.
- **Stockage**: L'artefact est stocké dans un référentiel d'artefacts (Frog Artifactory, Sonatype Nexus) pour être utilisé ultérieurement dans le processus de déploiement.



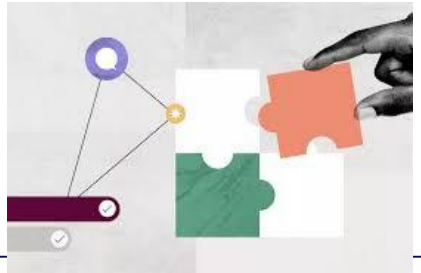
Avantages du CI

Augmentation de la fréquence des déploiements permet de livrer de nouvelles fonctionnalités plus rapidement aux utilisateurs

Détection rapide des problèmes grâce aux tests automatisés

Les développeurs intègrent leur code plus souvent, ce qui réduit les risques de conflits.

Meilleure qualité du code.



Meilleure collaboration : L'intégration continue favorise une collaboration étroite entre les Dévs., Ops, QAs.

Exemple d’outils pour la CI

- **Jenkins**: est un outil open-source qui automatiser les différentes étapes du cycle de vie d'une application, de la compilation à la livraison en passant par les tests (CI/CD).
- **GitLab** : Intégré à GitLab, il offre des fonctionnalités CI/CD.
- **CircleCI**: est une plateforme CI/CD utilisée pour mettre en œuvre des pratiques Devops.
- **Azure DevOps** : offre le service complet CI/CD

Livraison Continue (CD)

- La livraison continue est l'étape qui suit l'intégration continue.
- Elle est responsable du déploiement des modifications du code dans l'environnement de **production**.
- Dans la plupart des cas, le pipeline CD n'est déclenché que lorsque des modifications sont apportées à certaines branches (main/develop/feature), mais cela peut varier en fonction des normes de l'entreprise.

Etapes de la Livraison Continue

- **Pré-production (ou Staging)**
  - *Déploiement* : déployer l'artefact produit lors de l'étape CI dans un environnement similaire à la production.
  - *Tests supplémentaires*: effectuer des tests plus approfondis notamment des tests d'acceptation (**UAT**) par les utilisateurs finaux ou les clients; ces tests permettent de valider que le logiciel fonctionne comme prévu dans un contexte proche de la réalité.
  - *Configuration spécifique*: Simuler des scénarios spécifiques tels que des charges élevées, des interruptions réseau ou des erreurs système.

Etapes de la Livraison Continue

- **Validation manuelle (optionnelle)**
  - *Tests manuels*: Des testeurs peuvent effectuer des tests manuels pour vérifier des scénarios complexes ou des fonctionnalités qui sont difficiles à automatiser.
  - *Approbation*: Les responsables peuvent donner leur approbation pour le déploiement en production après avoir vérifié que le logiciel est prêt.
  - *Retour en arrière*: Si des problèmes sont détectés lors de cette étape, il est possible de revenir à une version précédente du logiciel.

Livraison Continue (CD) ou Déploiement Continue(CD)

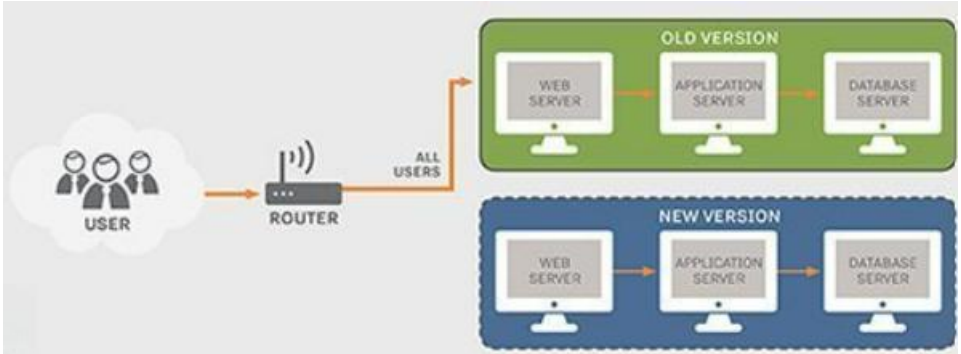
- **Déploiement en production**
  - *Déploiement automatisé ou manuel*:
    - ✓ Livraison continue: Le logiciel est prêt à être déployé, mais la décision finale appartient à une équipe humaine.
    - ✓ Déploiement continu: Chaque changement qui passe avec succès tous les tests est automatiquement déployé en production.

Livraison Continue (CD) & Déploiement Continue(CD)

- **Déploiement en production**
  - *Stratégies de déploiement*: Il existe différentes stratégies de déploiement comme exemple on a :
    - ✓ le déploiement bleu-vert,
    - ✓ le déploiement canari,
    - ✓ ou le déploiement progressif.
  - *Surveillance*: Une fois déployé en production, le logiciel est surveillé de près pour détecter d'éventuels problèmes et assurer sa disponibilité.

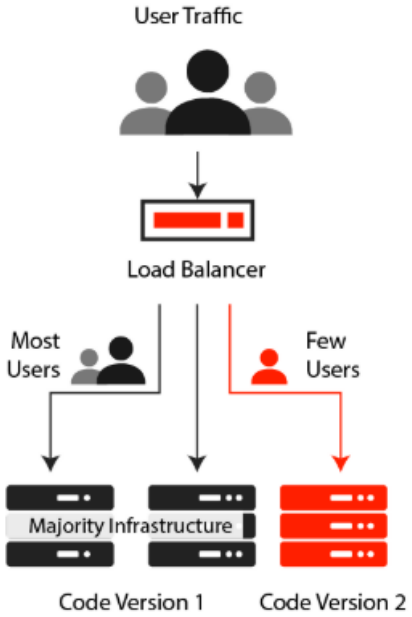
Stratégies de déploiement

- Le déploiement **bleu-vert**
  - C’est une technique qui consiste à maintenir deux environnements de production identiques "bleu" (actif) et "vert" (inactif), mais un seul est actif à un moment donné.
  - Déploiement de la nouvelle version: La nouvelle version est déployée dans l'environnement vert.
  - Basculement rapide du trafic: Une fois la nouvelle version validée, le trafic est basculé de l'environnement bleu vers l'environnement vert de manière instantanée.
  - Retour en arrière facile: Si un problème survient, il suffit de rebasculer le trafic vers l'environnement bleu.



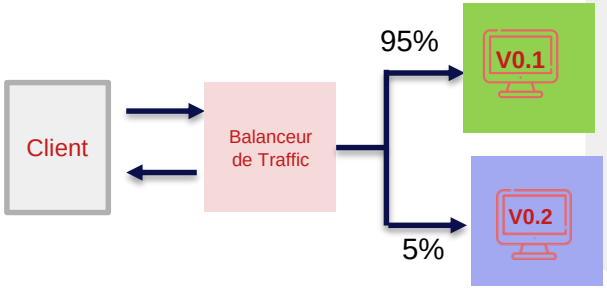
Stratégies de déploiement

- Le déploiement **canari**
  - Le déploiement canari consiste à déployer la nouvelle version de l'application auprès d'un **sous-ensemble d'utilisateurs** (les "canaris") avant de la déployer à l'ensemble des utilisateurs.
  - Surveillance: Le comportement de la nouvelle version est étroitement surveillé pour détecter d'éventuels problèmes.
  - Élargissement progressif: Si aucun problème n'est détecté, la nouvelle version est déployée à un groupe d'utilisateurs plus large, etc.
  - Le déploiement canari est une approche plus spécifique qui se concentre sur la **détection rapide des problèmes**

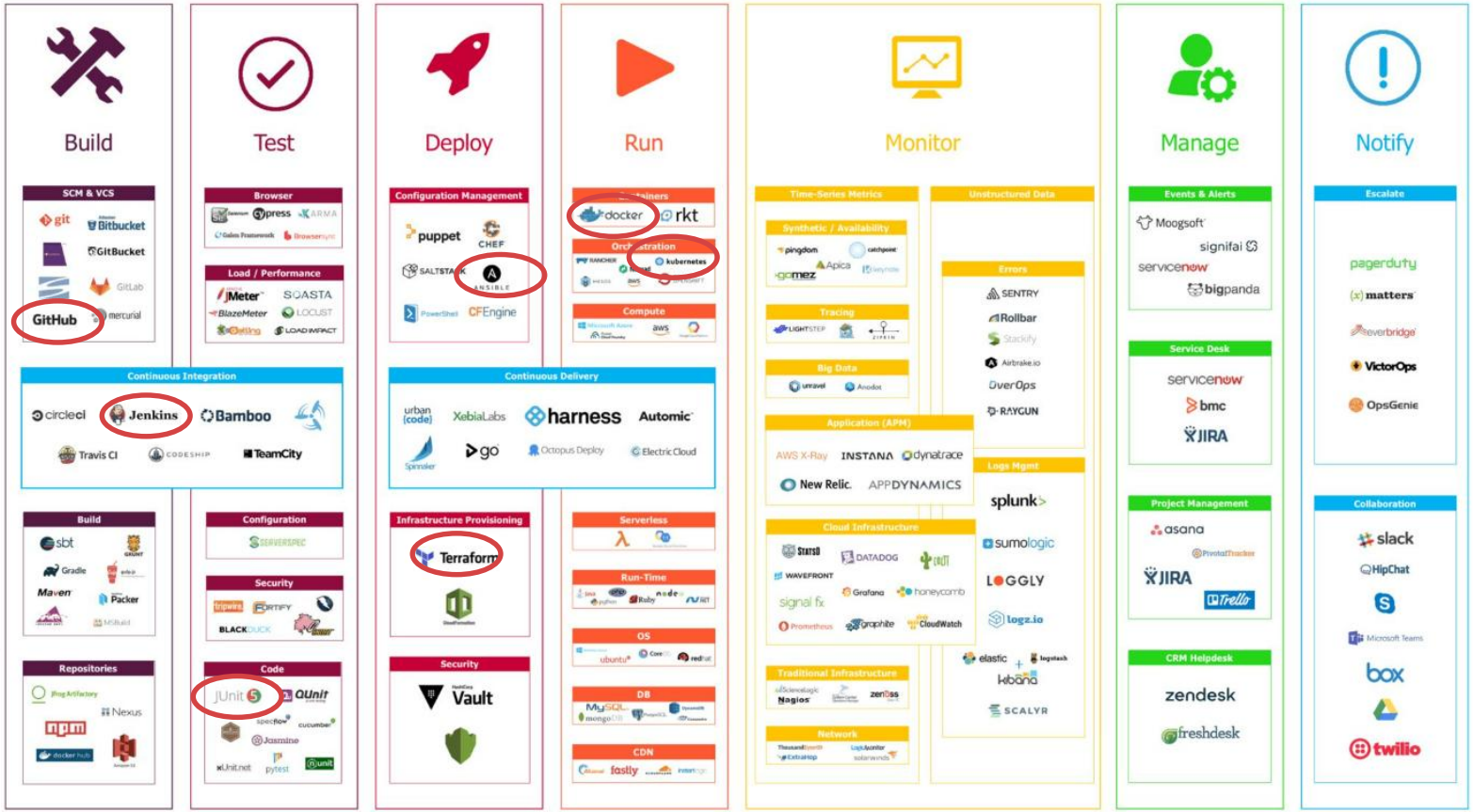


Stratégies de déploiement

- Le déploiement **progressif**
  - Déploiement initial: La nouvelle version de l'application est déployée sur un **sous-ensemble de serveurs**.
  - Répartition du trafic: Une partie du trafic est progressivement redirigée vers les serveurs exécutant la nouvelle version.
  - Surveillance: Le comportement de la nouvelle version est étroitement surveillé pour détecter d'éventuels problèmes.
  - Ajustement du trafic: Si la nouvelle version se comporte comme prévu, la proportion de trafic dirigée vers cette version est augmentée. Sinon, le trafic est redirigé vers l'ancienne version.
  - Déploiement complet: Une fois que la nouvelle version a été validée, tout le trafic est redirigé vers cette version.
  - Le déploiement progressif est une approche qui vise à **minimiser l'impact d'un déploiement problématique**



Outils pour CI/CD



Kubernetes

- Kubernetes, souvent abrégé k8s, est une plateforme d'orchestration de conteneurs open-source créé par Google en 2015 :
  - **Automatise le déploiement**: Il gère la création, la mise à l'échelle et la mise à jour des conteneurs.
  - **Assure la haute disponibilité**: Il répartit les conteneurs sur plusieurs machines pour éviter les pannes.
  - **Gère les ressources**: Il alloue les ressources (CPU, mémoire) de manière optimale.
  - **Facilite la découverte de services**: Il permet aux conteneurs de se trouver et de communiquer entre eux.
  - **Permet une configuration déclarative**: Kubernetes permet une configuration déclarative en utilisant des fichiers de configuration écrits en YAML ou JSON pour décrire l'état souhaité des ressources du cluster.

Concept de base de Kubernetes

1 Cluster

- C'est l'unité fondamentale de déploiement dans Kubernetes.
- C'est un ensemble de machines (physiques ou virtuelles) travaillant ensemble comme une seule entité pour exécuter les applications conteneurisées.

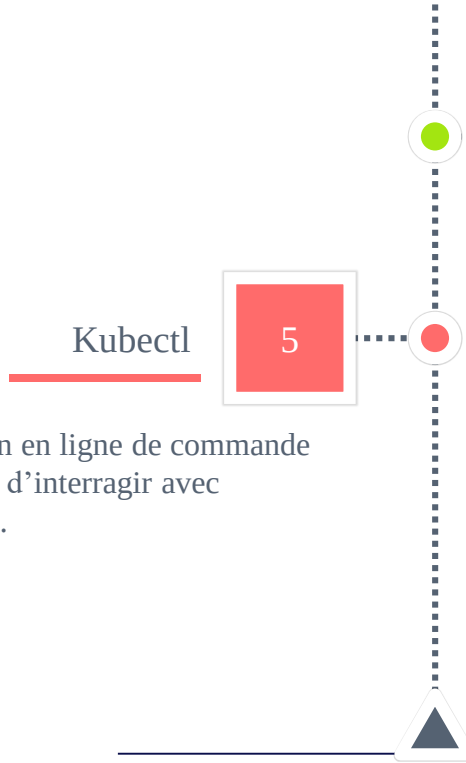
2 Node

- Un node est une machine individuelle au sein d'un cluster Kubernetes.
- C'est là que les conteneurs sont exécutés.
- Chaque node exécute un agent Kubernetes appelé kubelet qui communique avec le maître du cluster pour recevoir les instructions de déploiement et de gestion des conteneurs.
- Chaque Node doit exécuter un Container Engine comme Docker
- 2 types de nodes : master & worker

3 Pod

- C'est la plus petite unité d'exécution dans Kubernetes.
- Il représente un groupe de un ou plusieurs conteneurs étroitement liés qui partagent le même réseau et le même espace de stockage.
- Lorsqu'un déploiement est crée, Kubernetes crée un Pod pour héberger l'instance d'application

Concept de base de Kubernetes



4

Namespace

- Un namespace est un mécanisme d'isolation logique au sein d'un cluster Kubernetes.
- Il permet de diviser un cluster en plusieurs espaces de travail virtuels.
- Chaque namespace a son propre ensemble d'objets Kubernetes (pods, services, etc.) et les objets d'un namespace sont non accessibles depuis un autre namespace.
- Les namespaces peuvent être utilisés pour séparer les équipes ou les applications

5

Kubectl

- Application en ligne de commande permettant d'interragir avec kubernetes.

Architectures de Kubernetes

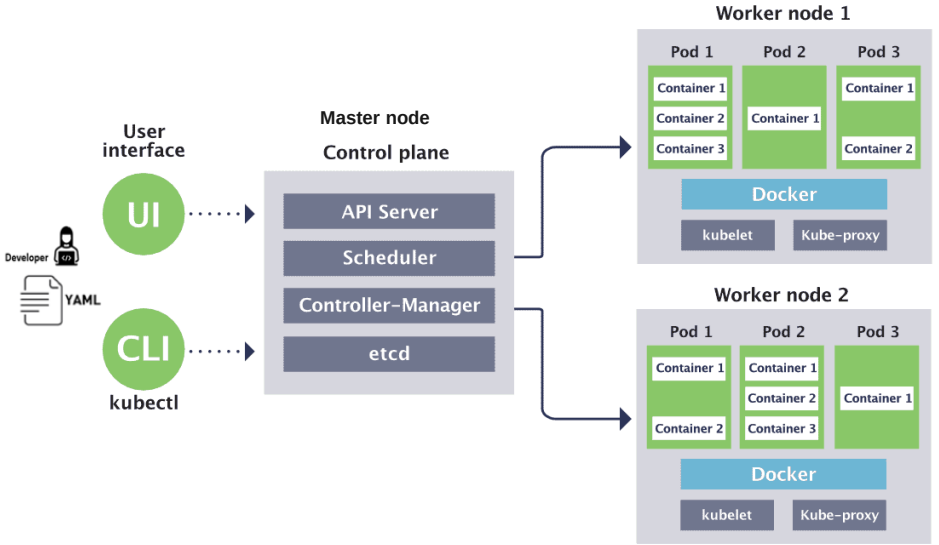
Etcd : Unité de stockage distribuée où on peut stocker et met à jour l'état désiré du cluster.

API server : Il est responsable de gérer l'état du cluster, de communiquer avec les autres composants et de fournir une interface pour que les utilisateurs puissent interagir avec le cluster.

Scheduler : Il permet de choisir le node qui va héberger un pod.

Controller Manager est le maintien de l'état désiré du cluster: compare l'état actuel avec l'état désiré spécifié dans le manifeste, s'assure que les Services sont correctement configurés, maintient le nombre de replicas, supprime les pods terminés,etc.

Iil existe deux rôles de machines : **les nœuds maîtres (ou masters)** et **les nœuds de travail (ou workers)**.

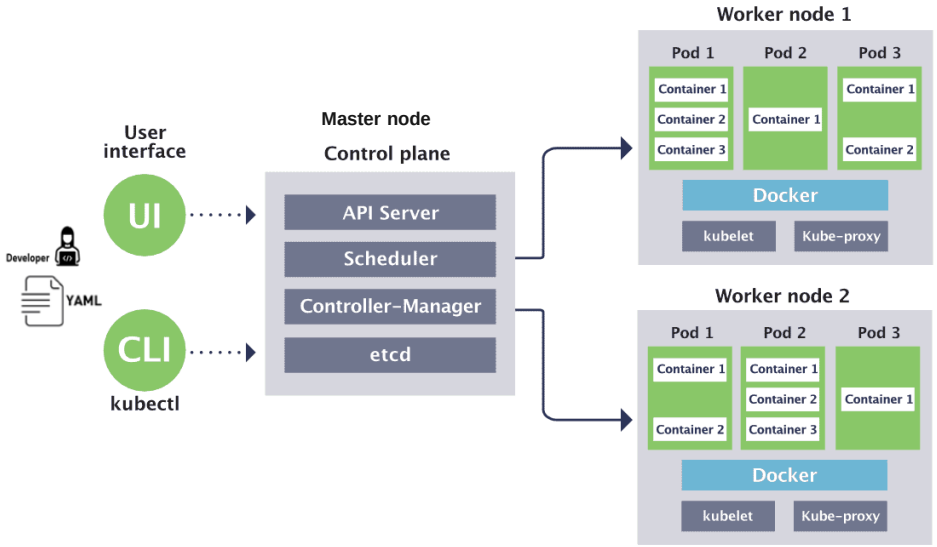


Architectures de Kubernetes

Kubelet : est un agent qui s'exécute sur chaque node d'un cluster Kubernetes. Il est responsable de la gestion des pods et de leurs conteneurs au niveau du node, interagit avec le master et docker engine pour la gestion des conteneurs.

Kube proxy : gère le routage réseau au sein du cluster, expose les services à l'extérieur du cluster, assure la communication entre les services et les pods, réparti les charge de service entre les différends nodes, etc.

cAdvisor : Agent du node qui surveille et récupère les données de consommation des ressources et des performances comme le processeur, mémoire, ainsi que l'utilisation disque et réseau des conteneurs du Node.



Avantages de Kubernetes

- **Isolation**: Les **namespaces** permettent d'isoler les applications et les équipes.
- **Scalabilité**: Les pods peuvent être créés et détruits dynamiquement pour s'adapter à la charge.
- **Disponibilité**: Kubernetes assure la haute disponibilité des applications en redémarrant automatiquement les pods défaillants.
- **Flexibilité**: Kubernetes permet de déployer des applications complexes et hétérogènes.

Gestion des ressources dans kubernetes

- Kubernetes offre une gestion avancée des ressources qui permet aux développeurs de définir des **limites** et des **demandes, Quotas** de ressources pour chaque conteneur et **QOS** .
- Cela garantit que les applications ont accès aux ressources dont elles ont besoin pour fonctionner efficacement, tout en évitant les gaspillages de ressources inutiles.

Gestion des ressources dans Kubernetes

- Gestion des **limites** et des **demandes** de ressources
  - *Demandes* (Requests) : Représentent les ressources minimales (CPU et mémoire) qu'un Pod ou un conteneur requiert pour fonctionner; Cette valeur permet à kubernetes de planifier (scheduler) le Pod sur un nœud capable de fournir ces ressources.
  - *Limites* (Limits) : Définissent les ressources maximales que le conteneur peut consommer; Kubernetes empêche le conteneur de dépasser cette limite.
- Gestion des **quotas** de ressources
  - Permettent de restreindre la consommation de ressources à l'échelle d'un namespace; cela permet d'éviter qu'une seule application ne monopolise les ressources du cluster.
- Gestion des classes de **QoS** (Quality of Service)
  - Kubernetes classe les Pods en trois catégories de QoS basées sur leurs demandes et limites
    - ✓**Guaranteed** :Si les demandes et les limites sont égales et définies pour chaque conteneur du Pod.
    - ✓**Burstable** :Si les demandes sont inférieures aux limites ou partiellement définies.
    - ✓**BestEffort** :Si ni les demandes ni les limites ne sont définies.

Gestion des ressources dans Kubernetes

- **Autoscaling** des ressources
  - Horizontal Pod Autoscaler (HPA) : Ajuste dynamiquement le nombre de réplicas d'un Pod en fonction de l'utilisation des ressources (par exemple, CPU ou mémoire).
  - Cluster Autoscaler : Ajuste dynamiquement le nombre de nœuds dans le cluster en fonction des besoins en ressources.
- **Affinité** et anti-affinité
  - Permet de contrôler où les Pods sont placés sur un nœud particulier ou selon leur affinité avec d'autres Pods.
  - Node Affinity : Contraint les Pods à être placés sur des nœuds correspondant à des étiquettes spécifiques.
  - Pod Affinity/Anti-affinity : Contraint les Pods à être placés proches ou loin d'autres Pods.
- Gestion des **priorités** et des préemptions
  - Kubernetes permet de définir des niveaux de priorité pour les Pods via des PriorityClasses.
  - Un Pod qui a la plus haute priorité aura accès aux ressources en cas de pénurie

Gestion des ressources dans Kubernetes

- Monitoring et optimisation des ressources
  - Kubernetes fournit des métriques via **Metrics Server** ou des outils comme **Prometheus** pour surveiller l'utilisation des ressources et ajuster les configurations si nécessaire.
- Gestion des volumes de stockage
  - Kubernetes permet de gérer les besoins en stockage via des **PersistentVolumes** (PV) et des **PersistentVolumeClaims** (PVC), avec des options pour le provisionnement dynamique.
  - Quand un conteneur a besoin d'un volume, il crée une PersistentVolumeClaim : une demande de volume (persistant). Si un des objets StorageClass est en capacité de le fournir, alors un PersistentVolume est créé et lié à ce conteneur : il devient disponible en tant que volume monté dans le conteneur.
- Gestion des ressources spécialisées
  - Kubernetes peut gérer des ressources matérielles spécifiques comme les GPUs ou d'autres périphériques.

L’écosystème Kubernetes

- Kubernetes est un ensemble de standards.
- Il existe de nombreuses variantes (ou "flavours") de Kubernetes, qui définissent des solutions techniques pour les fonctionnalités que Kubernetes défini comme les solutions réseau, le stockage (distribué ou non), l'équilibrage de charge, les services de reverse proxy (Ingress), l'autoscaling des clusters, le monitoring, et bien d'autres.
- Exemple:
  - **Google Kubernetes Engine** (GKE) (Google Cloud Plateform).
  - **Azure Kubernetes Services** (AKS) (Microsoft Azure).
  - Elastic Kubernetes Services (EKS) (Amazon Web Services).

Fichier de configuration YAML

- YAML (YAML Ain't Markup Language) est un langage de sérialisation de données, souvent utilisé pour écrire des fichiers de configuration en raison de sa lisibilité.
- YAML est largement utilisé dans l'industrie, ce qui en fait un format de configuration standard pour Kubernetes.
- Dans le contexte de Kubernetes, les fichiers YAML servent à décrire l'état souhaité des applications : quels pods déployer, quels services exposer, etc.

Syntaxe YAML

- L’indentation est très importante dans les fichiers YAML puisqu’elle sert à définir la hierarchie des données.
- Les caractères de tabulation ne sont pas autorisés, les espaces blancs sont donc utilisés à la place.
- Utilise le concept « clé: valeur »:
  - Un fichier YAML Kubernetes est composé d'une série de clés-valeurs.
  - Chaque clé représente un objet Kubernetes (comme un pod, un service, un déploiement) et les valeurs définissent les propriétés de cet objet.
- L’inconvénient du YAML, c’est l’absence de correcteur syntaxique automatique, mais on peut utiliser la commande yamllint pour s’assurer que le fichier est valide .

Attribut YAML Kubernetes

- **ApiVersion** indique la version de l'API Kubernetes utilisée pour créer l'objet (ici pod).
- **Kind** spécifie le type d'objet créé, il existe de nombreux objets dans Kubernetes comme Pod, Service, Volume, Namespace, Deployment et d'autres.
- **Metadata** est la section servant à décrire l'objet:
  - Name est le nom de l'objet qu'on veut créer.
  - Labels sont des paires clé-valeur qu'on associe à un objet pour des questions d'organisation, et faciliter la sélection (Supprimer tous les pods ayant un label spécifique).
  - Annotations sont également des paires clé-valeur qui peuvent contenir des informations supplémentaires pour des outils et des bibliothèques.
- **spec** est la section contenant les détails spécifiques de l'objet à créer. Par exemple pour un pod, il faut spécifier les conteneurs qu'il va exécuter ou les volumes à monter.

Création d'un Pod

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-1
  labels:
    app: first-app
    environment: production
  annotations:
    creator: "Benabbou F."
    created-at: "2025-01-15"
spec:
  containers:
  - name: conteneur-1
    image: nginx
```

Attribut YAML

Création d'un déploiement

- **ApiVersion** indique la version de l'API Kubernetes utilisée pour créer un deployment « apps/v1 ».
- **Kind** : Deployment.
- **Replica** 3: Indique que le déploiement doit créer et maintenir 3 répliques (pods) de l'application.
- **Selector** : Cette section définit les critères de sélection pour les pods gérés par ce déploiement.
- **matchLabels**: défini le critère de sélection, ce déploiement gère les pods ayant l'étiquette "app" avec la valeur "first-app".
- **template**: Cette section définit le modèle pour les pods qui seront créés par le déploiement.
- **image**: localhost:5000/firstapp:local: Spécifie l'image Docker à utiliser pour le conteneur.
- **ports**: Définit les ports exposés par le conteneur.
- **containerPort**: 3000: Indique que le conteneur écoute sur le port 3000.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: first-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: first-app
  template:
    metadata:
      labels:
        app: first-app
    spec:
      containers:
      - name: cont-first-app
        image: localhost:5000/firstapp:local
        ports:
        - containerPort: 3000
```

Syntaxe YAML

- **apiVersion**: Indique la version v1 de l'API "core", qui est responsable des pods, des services, etc.
- **name**: first-app-service donne un nom au service.
- **spec**: Contient les spécifications du service.
- **selector** Définit les critères de sélection pour les pods auxquels le service doit être lié.
- **app**: first-app Indique que le service doit être lié aux pods ayant l'étiquette "app: first-app".
- **ports**: Définit les ports du service.
- **protocol**: TCP: Indique que le protocole utilisé est TCP.
- **port**: 8080 Spécifie le port externe d'où le service sera accessible.
- **targetPort**: Spécifie le port interne du conteneur auquel le service doit être mappé.
- type: LoadBalancer: Indique comment le service doit être exposé
  - Nodeport: Via l'IP d'un nœud et un port dédié, pas d 'équilibrage de charge.
  - LoadBalancer: Via une adresse IP publique unique, il y a équilibrage de charge entre les nodes.

Création d'un service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: first-app-service
spec:
  selector:
    app: first-app
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 3000
  type: LoadBalancer
```

MiniKube

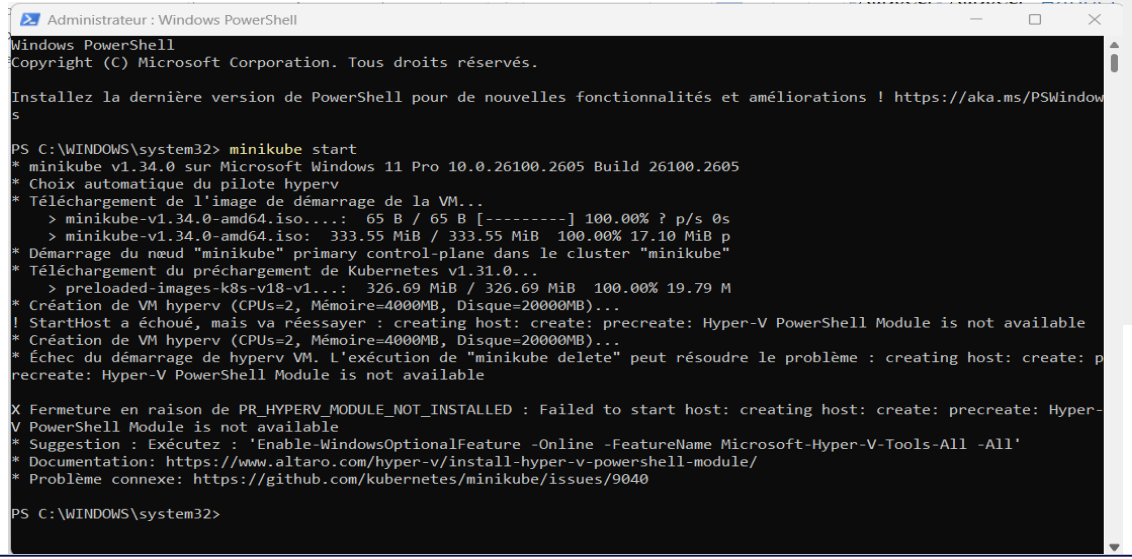
- Minikube est un outil qui permet de créer et de gérer un cluster Kubernetes local sur une seule machine.
- Solution Pratique pour expérimenter et tester des applications Kubernetes sans nécessiter un cluster distant.
- Fournit les principales fonctionnalités de Kubernetes, comme le déploiement de Pods, Services, ConfigMaps, Volumes, etc.
- L’atelier fournit plus de détails de son installation et utilisation,

Commandes Minikube

Démarrer un cluster :

```
minikube start
```

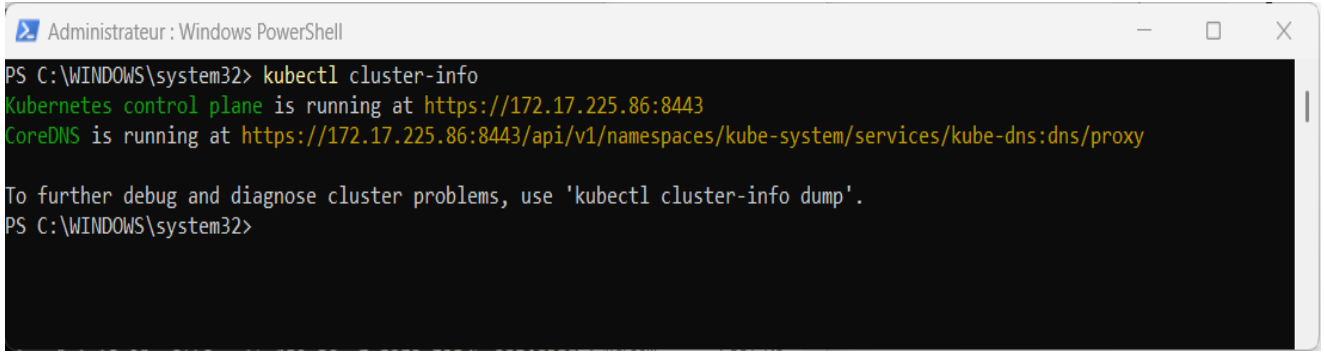
> Cette commande permet de créer un cluster avec un seul node



Commandes Minikube

Utiliser kubectl pour interagir avec le cluster:

```
kubectl cluster.info
```

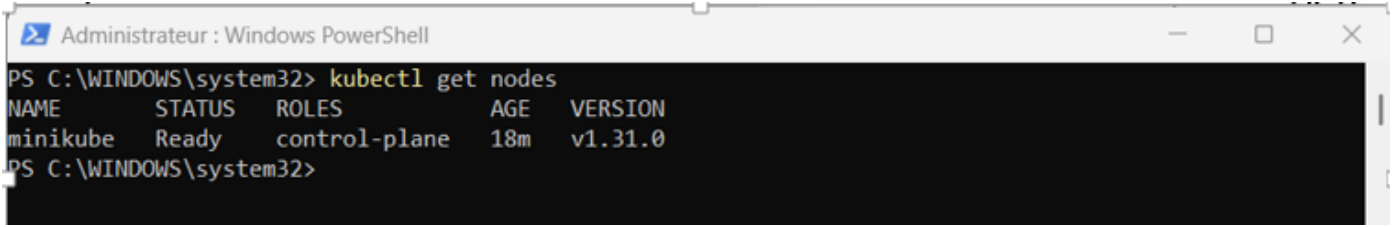


> Cette commande fournit les détails des services principaux du cluster.

Les Commandes Kubectl

Récupération des noeuds :

Kubectl get nodes



Cette commande permet de d'afficher tous les noeuds du cluster masters ou bien des workers et leurs informations .

Déploiement avec 3 replicas

Kubectl create deployment first-app
--image: localhost:5000/firstapp:local --replicas=3

kubectl expose deployment first-app --type=NodePort
--port=8080 --target-port=3000

Ou bien utiliser un fichier de configuration yaml.

kubectl apply -f deployment.yaml

```
PS C:\Users\PC\Desktop\docker\node-bulletin-board-master\bulletin-board-app> kubectl apply -f deployment.yaml
deployment.apps/first-app created
service/first-app-service created
```

Cette commande permet d'appliquer la configuration d'un fichier yaml afin de créer un déploiement, selon des paramètres spécifiques .

Déploiement avec 3

Liste et description des deploiements :

Kubectl get deployments

Cette commande permet de récupérer les déploiements créés et leurs informations (les noms , l'âge , la disponibilité de leurs réplica sets) .

```
PS C:\WINDOWS\system32> kubectl describe deployments
Name:                   first-app
Namespace:              default
CreationTimestamp:      Sun, 29 Dec 2024 20:50:41 +0100
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=first-app
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=first-app
  Containers:
   first-app:
    Image:         localhost:5000/firstapp:local
    Port:          3000/TCP
    Host Port:     0/TCP
    Environment:   <none>
    Mounts:        <none>
  Volumes:         <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   first-app-65dcd57f6b (3/3 replicas created)
Events:          <none>
```

Recuperation des pods

Kubectl get pods

```
PS C:\WINDOWS\system32> kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
first-app-65dcd57f6b-82g4n   1/1     Running   0          3h3m
first-app-65dcd57f6b-qn2wt   1/1     Running   0          3h3m
first-app-65dcd57f6b-tn7kv   1/1     Running   0          3h3m
```

Cette commande permet de recuperer les pods d’une application et leurs informations .

Lister les Replica-set

Kubectl get rs

```
PS C:\WINDOWS\system32> kubectl get rs
NAME                   DESIRED   CURRENT   READY   AGE
first-app-65dcd57f6b   3         3         3       3h11m
PS C:\WINDOWS\system32>
```

Cette commande permet de donner le nombre de replicas de l’application .

Modifier le nombre de Replica-set

kubectl scale deployments/first-app --replicas=5

```
PS C:\WINDOWS\system32> kubectl scale deployments/first-app --replicas=5
deployment.apps/first-app scaled
PS C:\WINDOWS\system32> kubectl get deployments
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
first-app   5/5     5            5           3h17m
```

Cette commande permet de de modifier le nombre de replicas à 5.

Modifier le nombre de Replica-set

kubectl scale deployments/first-app --replicas=2

```
PS C:\WINDOWS\system32> kubectl scale deployments/first-app --replicas=2
deployment.apps/first-app scaled
PS C:\WINDOWS\system32> kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
first-app-65dcd57f6b-82g4n   1/1     Running   0          3h21m
first-app-65dcd57f6b-tn7kv   1/1     Running   0          3h21m
PS C:\WINDOWS\system32>
```

Cette commande permet de de modifier le nombre de replicas à 2.

Description des services

Kubectl describe services/first-app-service

Récupération des informations du service nommé first-app-service, comme l'ip , le namespace qui permet l'accès au service .

```
PS C:\WINDOWS\system32> kubectl describe services/first-app-service
Name:                     first-app-service
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=first-app
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.99.191.123
IPs:                      10.99.191.123
Port:                     <unset>  8080/TCP
TargetPort:               3000/TCP
NodePort:                 <unset>  30532/TCP
Endpoints:                10.244.0.20:3000,10.244.0.21:3000
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
PS C:\WINDOWS\system32>
```

Tableau de bord minikube

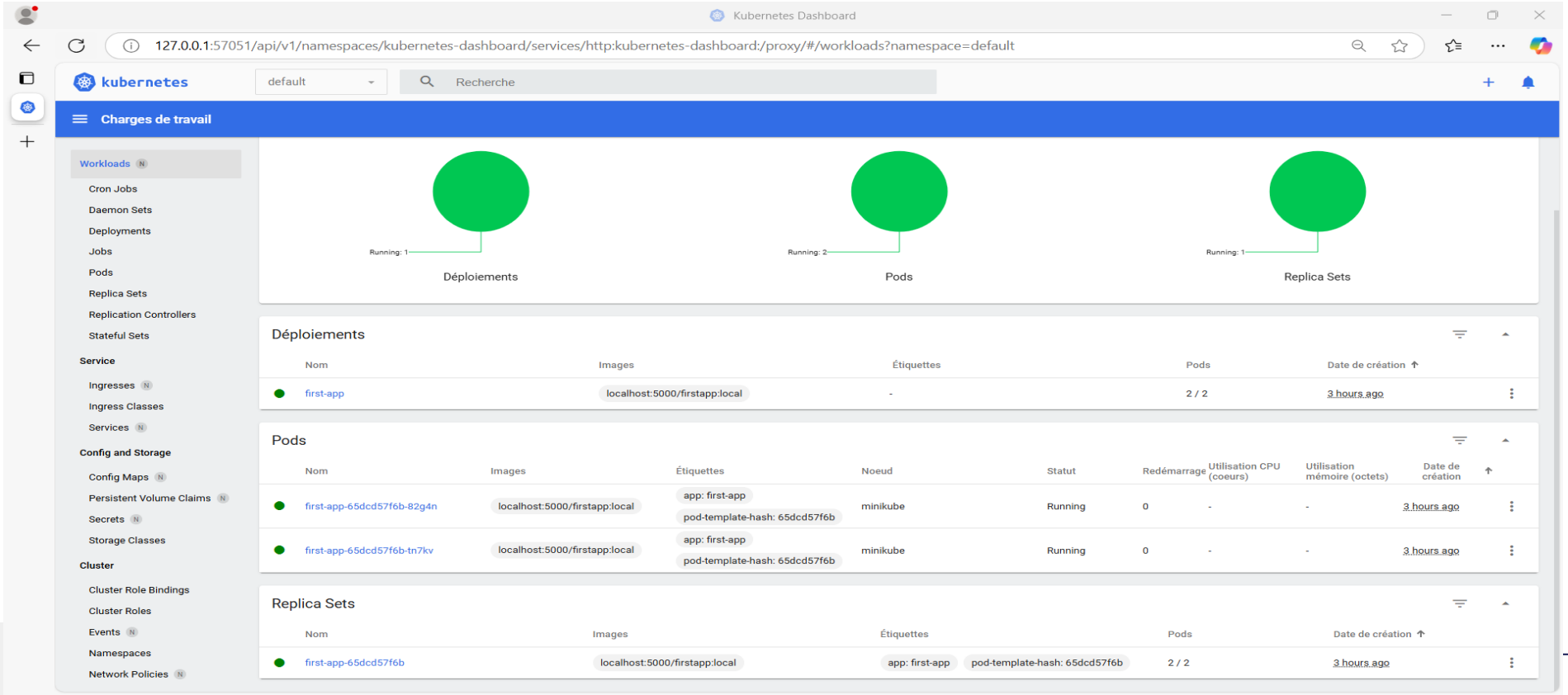
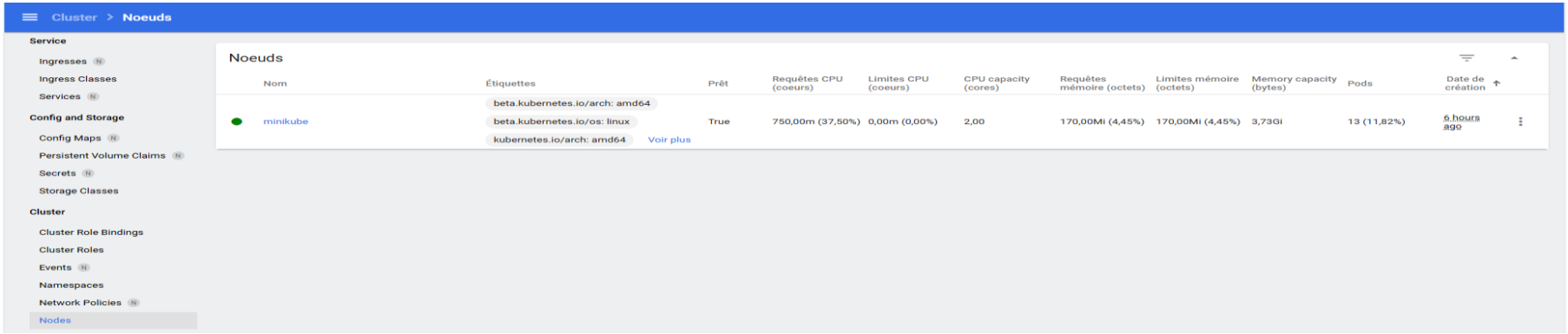
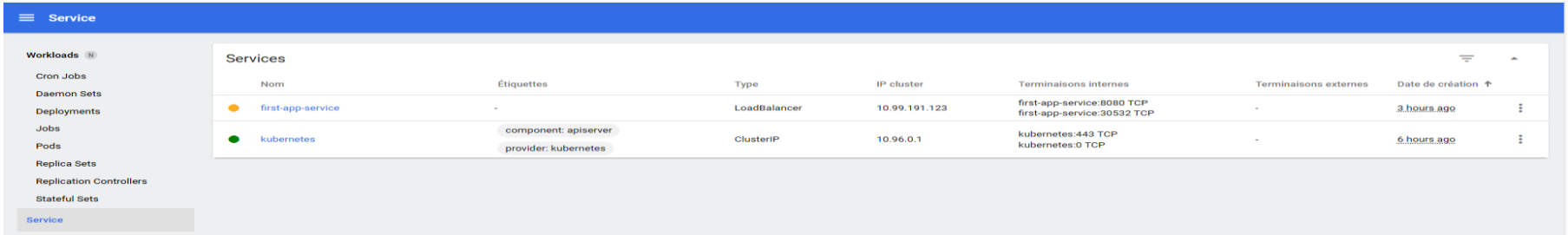


Tableau de bord minikube

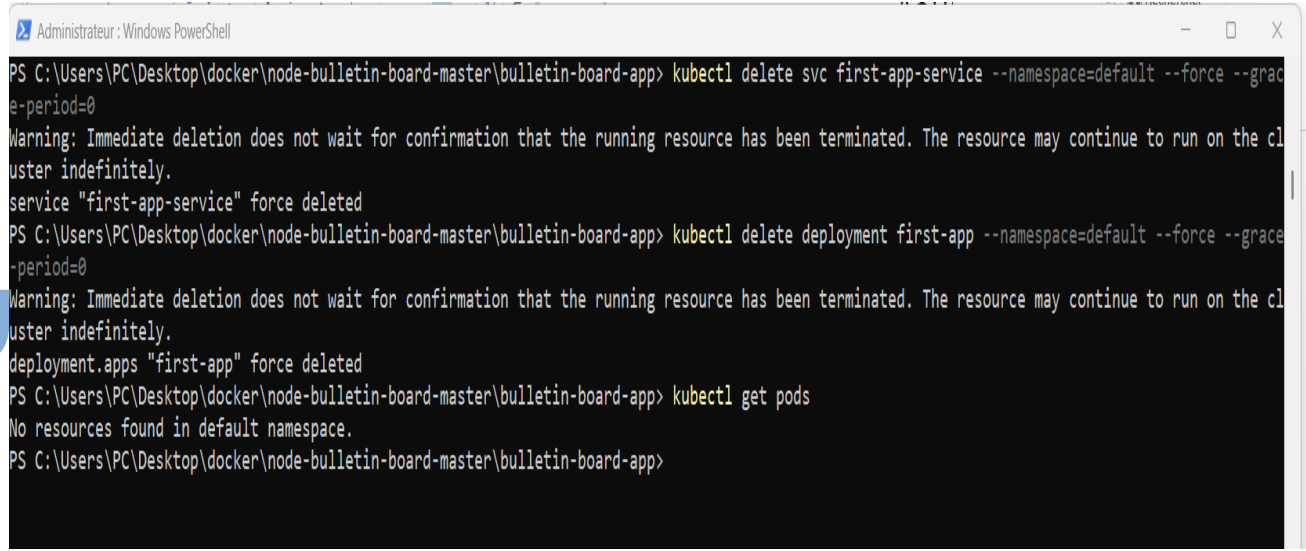




Arrêter les services et déploiements

```
kubectl delete svc first-app-service --namespace=default --force --grace-period=0
kubectl delete deployment firstapp --namespace=default --force --grace-period=0
minikube stop
```

La 1ère arrête les services
La 2ème arrête les déploiements
La 3ème arrête minikube



Cluster avec 3 nœuds/3 replicas

minikube start --nodes 3 -p multinode-node-test --driver=docker --cpus=2 --memory=1500

minikube profile list
Kubectl get nodes

```
PS C:\WINDOWS\system32> minikube profile list
|---------------------|-----------|---------|--------------|------|---------|---------|-------|---------------|--------------------|
|       Profile       | VM Driver | Runtime |      IP      | Port | Version | Status  | Nodes | Active Profile | Active Kubecontext |
|---------------------|-----------|---------|--------------|------|---------|---------|-------|---------------|--------------------|
| multinode-node-test | docker    | docker  | 192.168.49.2 | 8443 | v1.31.0 | Running |     3 |               | *                  |
|---------------------|-----------|---------|--------------|------|---------|---------|-------|---------------|--------------------|
PS C:\WINDOWS\system32>  kubectl get nodes
NAME                    STATUS   ROLES           AGE    VERSION
multinode-node-test     Ready    control-plane   106s   v1.31.0
multinode-node-test-m02   Ready    <none>          66s    v1.31.0
multinode-node-test-m03   Ready    <none>          27s    v1.31.0
```

La 1ère crée un cluster avec 3 nodes
La 2ème affiche les infos du cluster
La 3ème liste les nœuds

Informations sur les nœuds

```
PS C:\Users\PC> kubectl get nodes
NAME                 STATUS   ROLES           AGE   VERSION
multinode-node       Ready    control-plane   9h    v1.31.0
multinode-node-m02   Ready    <none>          9h    v1.31.0
multinode-node-m03   Ready    <none>          9h    v1.31.0
PS C:\Users\PC>
```

Récuperation des noeuds ‘kubectl get nodes’

```
PS C:\Users\PC\Desktop\docker\node-bulletin-board-master\bulletin-board-app> minikube status -p multinode-node-test
multinode-node-test
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

multinode-node-test-m02
type: Worker
host: Running
kubelet: Running

multinode-node-test-m03
type: Worker
host: Running
kubelet: Running
```

Vérification de l’état des noeuds , ‘minikube status -p multinode-node-test’

Afficher les nodes

Kubectl get pods –o wide

```
PS C:\Users\PC\Desktop\docker\node-bulletin-board-master\bulletin-board-app> kubectl get pods  -o wide
NAME                     READY   STATUS    RESTARTS   AGE   IP           NODE                    NOMINATED NODE   READINESS GATES
first-app-65dcd57f6b-247ww   1/1     Running   0          89m   10.244.1.2   multinode-node-test-m02   <none>           <none>
first-app-65dcd57f6b-4qv5r   1/1     Running   0          89m   10.244.0.3   multinode-node-test       <none>           <none>
first-app-65dcd57f6b-xlr5l   1/1     Running   0          89m   10.244.2.2   multinode-node-test-m03   <none>           <none>
PS C:\Users\PC\Desktop\docker\node-bulletin-board-master\bulletin-board-app>
```

“Affiche chaque pod dans quel node il s’exécute”

Supprimer un node

kubectl delete nodes multinode-node-test-m03 --context=multinode-node-test

```
PS C:\Users\PC\Desktop\docker\node-bulletin-board-master\bulletin-board-app> kubectl delete nodes  multinode-node-test-m03 --context=multinode-node-test
node "multinode-node-test-m03" deleted
PS C:\Users\PC\Desktop\docker\node-bulletin-board-master\bulletin-board-app> kubectl --context=multinode-node-test get nodes
NAME                    STATUS   ROLES           AGE    VERSION
multinode-node-test     Ready    control-plane   3h6m   v1.31.0
multinode-node-test-m02   Ready    <none>          3h5m   v1.31.0
PS C:\Users\PC\Desktop\docker\node-bulletin-board-master\bulletin-board-app>
```

“Supprime un noeud”

Ajouter un node

minikube node add -p multinode-node-test

```
PS C:\Users\PC> kubectl get nodes
NAME                STATUS   ROLES           AGE   VERSION
multinode-node      Ready    control-plane   9h    v1.31.0
multinode-node-m02  Ready    <none>          9h    v1.31.0
multinode-node-m03  Ready    <none>          9h    v1.31.0
PS C:\Users\PC>
```

> On peut ajouter des nœuds workers si on veut avec cette commande (en fonction de la performance de la machine hôte)

TABLE OF CONTENTS