

Projet DevOps

**Conception d'un pipeline CI/CD : intégration de Jenkins
pour l'automatisation et Kubernetes pour l'orchestration
des conteneurs**

Groupe :

Imane AABOUCHE - Mahamat Taha ABATCHA - Rihab BENABDELAZIZ

Formation : Master 2 Sécurité des Systèmes Informatiques

Année Universitaire : 2023 - 2024

Objectif du projet

Le travail présenté dans ce rapport a été effectué dans le cadre du projet d'Administration Réseaux, ayant pour but de démontrer l'application pratique et efficace des principes et techniques de l'administration réseau moderne, notamment dans le contexte du DevOps. L'objectif principal est de développer, déployer et gérer une application dédiée à l'affichage des résultats des participants au concours M2SSI, en utilisant des technologies de pointe telles que les conteneurs Docker, l'orchestration avec Kubernetes, et l'intégration et le déploiement continu via Jenkins.

Ce projet vise à offrir une expérience concrète des processus et outils d'administration réseau avancée, mettant l'accent sur l'automatisation, la scalabilité et la haute disponibilité.

Table des matières

Objectif du projet	1
Table des matières	2
Liste des figures	3
Introduction générale	4
I - Contexte du projet	5
II - Réalisations	8
1 - Pré-requis	8
2 - Mise en place du cluster Kubernetes	10
2.1 - Configuration du cluster Minikube	10
a - Conditions préalables	10
b - Installer les dépendances	11
c - Installer Docker	11
d - Installer Minikube	11
e - Installer Kubectl	11
f - Démarrer Minikube	11
2.2 - Configuration du cluster Kubeadm	12
3 - Configuration de Jenkins	14
3.1 Définitions	14
3.2 Installation de Jenkins	14
Conditions préalables	15
3.3 Administration de Jenkins	15
3.4 Pipeline Push Image	16
3.5 Pipeline Déploiement de l'application sur le cluster	17
4 - Test de fonctionnement	19
Annexes	25
Conclusion	27

Liste des figures

Figure 1. Architecture du projet	5
Figure 2. Dépôt Git	8
Figure 3. Docker Hub	8
Figure 4. Environnement Kubernetes	8
Figure 6. Architecture minikube	9
Figure 7. Démarrer minikube	10
Figure 8. Informations du cluster	11
Figure 9. Les noeuds du cluster minikube	11
Figure 10. La version de kubeadm	12
Figure 11. La version de kubectl	12
Figure 12. L'image sur Docker Hub	16
Figure 13. Le pipeline du déploiement de l'application	16
Figure 14. Statut du déploiement sous Jenkins	18
Figure 15. Les déploiements sous Kubeadm	18
Figure 16. Les services sous Kubeadm	18
Figure 17. Lien du Github sous Jenkins	19
Figure 18. Spécifier le fichier Jenkinsfile sous Jenkins	19
Figure 19. Pipeline Push sous Jenkins	19
Figure 20. Pipeline Push sous Jenkins	19
Figure 21. Statut du Push d'image sous Jenkins	20
Figure 22. Pipeline du déploiement sous Jenkins	20
Figure 23. Script du déploiement sous Jenkins	21
Figure 24. Le statut du déploiement de l'application	21
Figure 25. L'adresse IP du manager 1	22
Figure 26. L'application depuis manager 1	22
Figure 27. L'adresse IP du worker 1	22
Figure 28. L'application depuis worker 1	23

Introduction générale

Le DevOps, combinant les pratiques de développement (Dev) et d'opérations (Ops), est devenu un élément essentiel dans le monde de l'ingénierie logicielle moderne. Cette méthodologie vise à rapprocher le développement, les tests et le déploiement dans un cycle continu, permettant ainsi une livraison plus rapide, une plus grande réactivité aux changements et une amélioration significative de la qualité des logiciels.

Dans ce contexte, le Projet "**Déploiement d'une Application dans un Cluster Kubernetes**", s'inscrit comme une mise en pratique de ces principes. Il montre l'intégration des concepts DevOps dans un projet réel d'administration réseau, focalisé sur le déploiement et la gestion efficaces d'une application au sein d'un environnement Kubernetes.

Dans sa première phase, le projet commence par le développement de l'application, posant ainsi les bases pour l'application des pratiques DevOps. L'accent est mis sur la création d'une interface utilisateur intuitive, conçue pour afficher de manière efficace les résultats des participants à un concours proposé par l'organisation M2SSI.

Une fois développée, l'application est intégrée dans un workflow GitHub, où elle subit un contrôle de version et de gestion du code. Cette étape permet la gestion des versions et des branches du code, facilitant un contrôle détaillé des commits et des mises à jour, essentiels pour une collaboration technique efficace et une gestion rigoureuse du code source.

Par la suite, le projet adopte pleinement les pratiques du DevOps en mettant en place un pipeline d'intégration continue (CI) et de déploiement continu (CD) via Jenkins. Cette étape clé automatise le processus de test et de déploiement du logiciel, assurant que chaque mise à jour ou ajout de fonctionnalité passe par une série de vérifications automatisées pour garantir son efficacité et sa fiabilité avant le déploiement.

En conclusion, une fois l'application prête, elle est déployée via Kubernetes, qui s'occupe de son orchestration dans un environnement basé sur des conteneurs. Cette phase finale met en évidence la compétence de Kubernetes dans la gestion dynamique et l'évolutivité de l'application, illustrant ainsi le rôle crucial de l'orchestration de conteneurs.

I - Contexte du projet

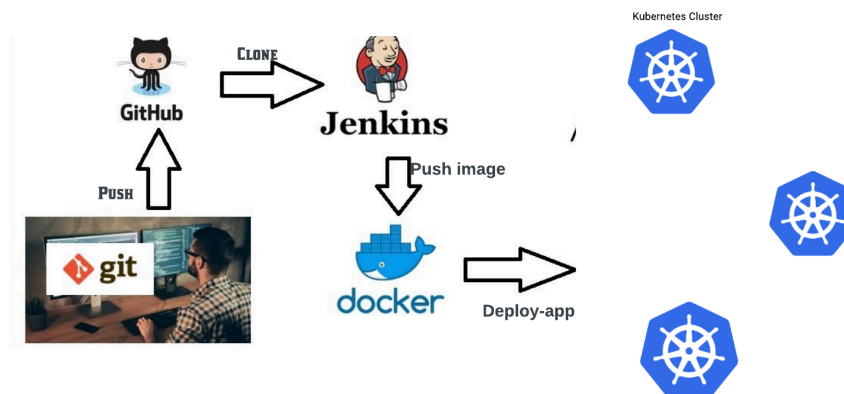


Figure 1. Architecture du projet

L'idée de ce projet DevOps est d'établir un pipeline d'intégration et de déploiement continu qui prend en charge le cycle de vie complet du développement jusqu'au déploiement d'une application web conçue pour afficher les résultats des participants au concours M2SSI.

Développement de l'Application

Les développeurs créent une application qui est conçue pour afficher les noms, prénoms et notes des participants du concours M2SSI. Cette application est développée localement sur leurs machines.

Gestion de Code Source sur GitHub

Une fois que l'application est prête pour le déploiement ou une fois qu'une nouvelle fonctionnalité a été ajoutée, les changements sont envoyés (push) vers un dépôt GitHub (git-repository). GitHub sert de plateforme de contrôle de version et de collaboration, permettant aux développeurs de suivre les changements et de travailler ensemble de manière organisée.

Intégration Continue avec Jenkins

Jenkins, configuré en tant que serveur d'intégration et de déploiement continu (CI/CD), joue un rôle crucial dans notre pipeline de développement. À chaque fois que du nouveau code est "pushé" sur GitHub, Jenkins est automatiquement notifié. Réagissant à cette notification, Jenkins commence par cloner le dernier état du dépôt GitHub. Il lance ensuite un pipeline CI/CD, qui comprend la compilation du code, l'exécution de tests automatisés pour assurer la qualité et la fiabilité, et la construction de l'application. Ce processus garantit que chaque modification apportée au code source maintient l'application dans un état stable et prêt pour le déploiement.

Conteneurisation avec Docker

Suite à un build réussi par Jenkins dans le cadre du pipeline CI/CD, l'étape suivante consiste en la création d'une image Docker pour l'application. Ce processus implique l'encapsulation de l'application ainsi que de toutes ses dépendances dans une image Docker standardisée. Jenkins utilise un fichier Dockerfile, qui définit les instructions nécessaires pour assembler cette image, y compris la sélection de l'image de base, l'installation des dépendances, la configuration de l'environnement d'exécution et la copie du code de l'application dans l'image.

Déploiement dans Kubernetes

L'image Docker de l'application est ensuite déployée dans un cluster Kubernetes, qui prend en charge le déploiement, la mise à l'échelle, et les mises à jour.

Pour la mise en place d'un cluster Kubernetes, deux approches principales sont utilisées : Minikube et Kubectl. Minikube est souvent privilégié pour les environnements de développement ou de test, car il permet de créer un cluster Kubernetes simplifié sur une machine locale. Cela facilite les tests rapides et le développement sans nécessiter une infrastructure complexe. En revanche, Kubectl est utilisé pour initialiser et configurer des clusters Kubernetes adaptés à un environnement de production. Ce dernier offre une méthode plus robuste pour initialiser et gérer des clusters Kubernetes adaptés aux déploiements à grande échelle et plus complexes.

II - Réalisations

1 - Pré-requis

Pour la réalisation de ce projet "Déploiement d'une Application dans un Cluster Kubernetes", plusieurs prérequis sont essentiels :

Ressources Matérielles :

- Quatre machines virtuelles, chacune dotée d'au moins 2 CPU et 4 GB de RAM. Ces VMs serviront à différentes fonctions dans le cadre du projet, notamment pour les environnements de développement, de test et de production.
- Parmi ces VMs, trois seront configurées pour former un cluster Kubernetes. Chaque VM dans ce cluster agit comme un nœud, contribuant ainsi à l'orchestration et la gestion de l'application conteneurisée.
- La quatrième VM, affectée exclusivement à Jenkins, gèrera le pipeline CI/CD, ce qui va englober la compilation, les tests, et le déploiement de l'application.

Infrastructure et Outils :

- Dépôt Git : Un dépôt GitHub sera utilisé pour le stockage et la gestion du code de l'application, facilitant le suivi des modifications et la collaboration, tout en s'intégrant au processus de CI/CD.

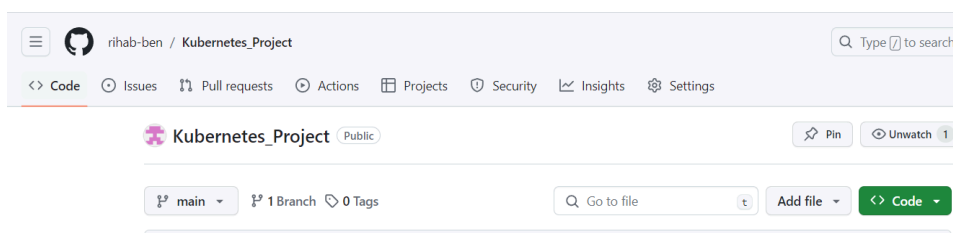


Figure 2. Dépôt Git

- Docker Hub : Un compte Docker Hub ou un registre d'images Docker similaire pour stocker les images Docker construites. Ceci est essentiel pour gérer et distribuer les versions de l'application sous forme de conteneurs.



Figure 3. Docker Hub

- Accès à un environnement Kubernetes : Que ce soit via Minikube pour un environnement de test ou un cluster Kubernetes (kubeadm) pour un environnement de production.



Figure 4. Environnement Kubernetes

- Serveur Jenkins : Pour la mise en place des pipelines CI/CD.



Figure 5. Serveur Jenkins

2 - Mise en place du cluster Kubernetes

2.1 - Configuration du cluster Minikube

Minikube est un outil open source qui permet de configurer un cluster Kubernetes à nœud unique sur la machine local.



Figure 6. Architecture minikube

Il facilite l'exécution d'un cluster Kubernetes à nœud unique sur l'ordinateur personnel pour le travail de développement quotidien. Il est multiplateforme et peut être installé sur macOS, Linux et Windows.

L'installation et la configuration suivante est pour une machine Ubuntu 20.04 / 22.04.

a - Conditions préalables

- Le bureau Ubuntu 20.04 est installé sur notre système.
- Au moins 4 Go de RAM et 2 cœurs de processeur ou plus.
- La virtualisation matérielle doit être activée sur notre système local.
- Un mot de passe root est configuré sur le serveur.

Script pour l'installation ...

b - Installer les dépendances**c - Installer Docker****d - Installer Minikube****e - Installer Kubectl****f - Démarrer Minikube**

Tous les packages requis sont installés, on peut maintenant démarrer Minikube avec la commande suivante :

```
pkumar@linuxtech1:~$ minikube start --driver=docker
* minikube v1.30.1 on Ubuntu 22.04 (vbox/amd64)
* Using the docker driver based on user configuration
* Using Docker driver with root privileges
* Starting control plane node minikube in cluster minikube
* Pulling base image ...
* Downloading Kubernetes v1.26.3 preload ...
  > gcr.io/k8s-minikube/kicbase... : 373.53 MiB / 373.53 MiB 100.00% 3.75 Mi
  > preloaded-images-k8s-v18-v1... : 397.02 MiB / 397.02 MiB 100.00% 3.67 Mi
* Creating docker container (CPUs=2, Memory=2200MB) ...
* Preparing Kubernetes v1.26.3 on Docker 23.0.2 ...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring bridge CNI (Container Networking Interface) ...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Verifying Kubernetes components...
* Enabled addons: default-storageclass, storage-provisioner
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Figure 7. Démarrer minikube

Vérifier les informations du cluster avec la commande suivante :

- **kubectl cluster-info**

```
pkumar@linuxtech1:~$ kubectl cluster-info
Kubernetes control plane is running at https://192.168.49.2:8443
CoreDNS is running at https://192.168.49.2:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Figure 8. Informations du cluster

Nous pouvons vérifier tous les nœuds en cours d'exécution avec la commande suivante :

- **kubectl get nodes**

```
pkumar@linuxtech1:~$ kubectl get nodes
NAME        STATUS    ROLES    AGE   VERSION
minikube    Ready    control-plane  2m52s  v1.26.3
```

Figure 9. Les noeuds du cluster minikube

2.2 - Configuration du cluster Kubeadm

kubeadm est une interface en ligne de commande qui facilite le déploiement et la gestion de clusters Kubernetes. Il est principalement utilisé pour initialiser un cluster Kubernetes en provisionnant le contrôleur de cluster, en rejoignant des nœuds au cluster, et en configurant les composants principaux nécessaires. Il est souvent utilisé dans des scénarios de déploiement sur des environnements, des machines virtuelles ou des instances cloud.

Nous avons opté pour cette solution pour pouvoir palier au problème du cluster single node (dans le cas de minikube), ainsi nous avons pu mettre en place un cluster multi node avec un manager et deux workers.

Pour déployer le cluster kubernetes en utilisant kubeadm, nous avons utilisé trois machines LXD, une vm manager et deux autres vms considérés comme workers.

Après avoir rencontré d'énormes difficultés pour mettre en place le cluster kubeadm, nous nous sommes appuyé sur le dépôt github suivant:

<https://github.com/justmeandopensource/kubernetes/blob/master/lxd-provisioning/README.md>

Ce lien utilise juste des playbooks pour déployer le cluster kubeadm sur des machines lxd.

Les étapes de la mise en place du cluster sont les suivantes :

01. Création d'un profil pour le cluster k8s

La création d'un profil LXC lors du déploiement de Kubernetes avec kubeadm sur des machines LXC offre une flexibilité et une personnalisation accrues en permettant la configuration fine des ressources, du réseau et des permissions pour chaque nœud du cluster. Par défaut les vms lxd n'ont pas suffisamment des ressources pour pouvoir déployer kubeadm dessus.

➤ lxc profile create k8s

Après la création du profil, nous l'avons personnalisé en ajoutant plus des ressources cpu et RAM aux vms.

02. Création des noeuds du cluster

- `lxc launch ubuntu:20.04 mgr --profile k8s`
- `lxc launch ubuntu:20.04 worker1 --profile k8s`
- `lxc launch ubuntu:20.04 worker2 --profile k8s`

03. Execution du script permettant l'installation de kubeadm

Nous avons un script qui permet l'installation de kubeadm, kubelet et kubectl sur chaque nœud de notre cluster, ce script se trouve sur notre dépôt github, car il n'est pas pratique de l'écrire sur ce rapport .

La commande pour exécuter le script est la suivante, on l'exécute pour chaque en commençant absolument par le manage.

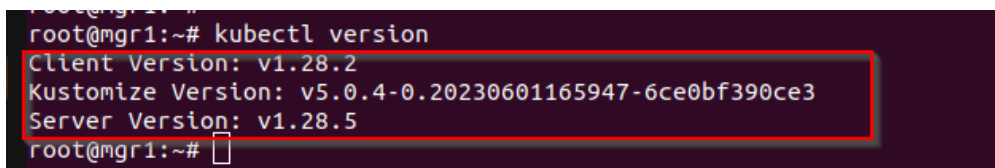
- `cat install-kube.sh | lxc exec mgr bash`

04. Vérification de l'installation



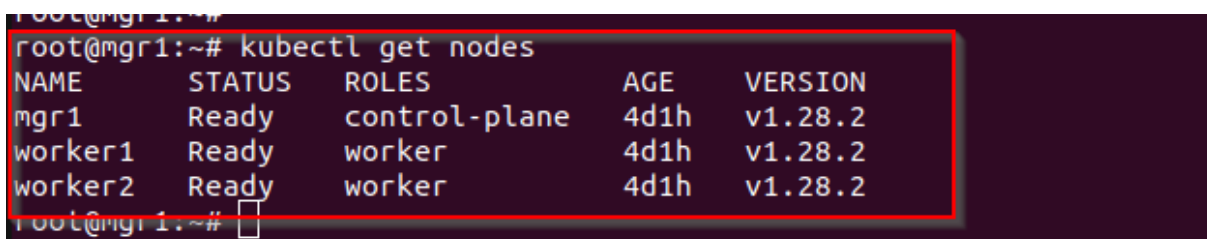
```
root@mgr1:~# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"28",
8", Compiler:"gc", Platform:"linux/amd64"}
root@mgr1:~#
```

Figure 10. La version de kubeadm



```
root@mgr1:~# kubectl version
Client Version: v1.28.2
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Server Version: v1.28.5
root@mgr1:~#
```

Figure 11. La version de kubectl



```
root@mgr1:~# kubectl get nodes
NAME      STATUS   ROLES          AGE    VERSION
mgr1      Ready    control-plane   4d1h   v1.28.2
worker1   Ready    worker          4d1h   v1.28.2
worker2   Ready    worker          4d1h   v1.28.2
root@mgr1:~#
```

Figure 20. Les noeuds du cluster Kubeadm

3 - Configuration de Jenkins

3.1 Définitions

- A. **Jenkins** est un serveur d'automatisation open source qui peut être utilisé pour automatiser toutes sortes de tâches liées à la création, aux tests et à la livraison ou au déploiement d'applications. Il offre une interface conviviale et prend en charge l'intégration continue et le déploiement continu (CI/CD).
- B. **Job (Tâche)** : Un job dans Jenkins représente une tâche spécifique à effectuer, telle que la compilation d'un code source, l'exécution de tests, ou le déploiement d'une application.
- C. **Build (Construction)** : Une build dans Jenkins se réfère au processus de compilation du code source d'une application en un exécutable ou en d'autres artefacts. Jenkins peut automatiser ce processus, déclenché soit par un événement (comme un changement dans le dépôt de code), soit de manière planifiée.
- D. **Pipeline** : Un pipeline est un ensemble de jobs connectés qui représentent les étapes d'un processus d'intégration continue (CI) ou de déploiement continu (CD). Les pipelines aident à orchestrer les différentes phases du développement logiciel, de la construction à la livraison.
- E. L'intégration continue (**CI**) et le déploiement continu (**CD**) dans Jenkins représentent une approche automatisée du développement logiciel. La CI consiste à intégrer fréquemment les modifications de code dans un référentiel partagé, déclenchant automatiquement la compilation et l'exécution de tests. Le CD étend la CI en automatisant le déploiement des applications après des tests réussis, assurant une livraison rapide et fiable des nouvelles fonctionnalités.

3.2 Installation de Jenkins

Jenkins peut être installé via des packages système natifs, Docker, ou même exécuté de manière autonome par n'importe quelle machine sur laquelle un environnement d'exécution Java (JRE) est installé. Dans notre cas, nous avons installé Jenkins sur une VM LXD qui se retrouve sur le même réseau que les machines du cluster Kubernetes.

Conditions préalables

Configuration matérielle minimale requise :

- 256 Mo de RAM
- 1 Go d'espace disque (bien que 10 Go soit un minimum recommandé si nous exécutons Jenkins en tant que conteneur Docker)

Configuration logicielle minimale requise :

- Java (Open JDK && Open JRE)
- Un navigateur pour accéder à l'interface d'administration de Jenkins

Étapes d'installation :

Tout d'abord, on ajoute la clé du référentiel au système en utilisant la commande suivante:

- **wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -**

Ensuite, ajoutons l'adresse du référentiel du paquet Debian à celle du serveur sources.list :

- **sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'**

Maintenant nous installerons Jenkins et ses dépendances.

- **sudo apt update && install jenkins**

Ouverture du pare feu et du port d'écoute de jenkins

- **sudo ufw allow 8080**
- **sudo ufw allow OpenSSH**
- **sudo ufw enable**

Enfin, nous pouvons démarrer jenkins

- **sudo systemctl start jenkins**

3.3 Administration de Jenkins

Après avoir installé Jenkins, nous allons accéder à son interface d'administration via un navigateur web en saisissant l'URL suivant **http://ip_jenkins:8080.**

Quelques plugins par défaut vont s'installer et nous pouvons déjà créer nos tâches et nos pipelines.

3.4 Pipeline Push Image

Nous avons créé un pipeline nommé <Push image> qui consiste à construire l'image de notre application et de la pousser dans notre registre qui est un repository docker hub dans notre cas. Notre pipeline utilise un fichier Jenkinsfile qui se trouve dans répertoire github, un Jenkinsfile est un fichier de configuration écrit en tant que code, généralement en utilisant une syntaxe déclarative ou scriptée, qui définit les étapes d'un pipeline dans Jenkins.

Fichier Jenkinsfile

```
pipeline {
  agent any
  environment {
    DOCKERHUB_CREDENTIALS = credentials('Imane-dock')
  }
  stages {
    stage('Build') {
      steps {
        sh 'docker build -t imaneaabouche/myapp:v1 .'
      }
    }
    stage('Login') {
      steps {
        sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin'
      }
    }
    stage('Push') {
      steps {
        sh 'docker push imaneaabouche/myapp:v1'
      }
    }
  }
  post {
    always {
      sh 'docker logout'
    }
  }
}
```

En résumé, pour cette partie nous avons d'abord créé un token(credentials) qui permet la connexion de Jenkins au répertoire docker hub, ensuite nous avons cloné les fichiers de l'application qui se trouvent sur le dépôt github.

Le pipeline est construit en étapes, la première consiste à construire l'image de l'application en utilisant un Dockerfile qui se trouve sur le même dépôt que le fichier Jenkinsfile, la deuxième étape nous permet de se connecter à notre répertoire docker hub, et la dernière étape consiste à pusher l'image construite sur le répertoire en spécifiant un tag (version) de l'application que l'on souhaite.

Sur la capture suivante, nous pouvons voir l'image qui vient d'être pushé.

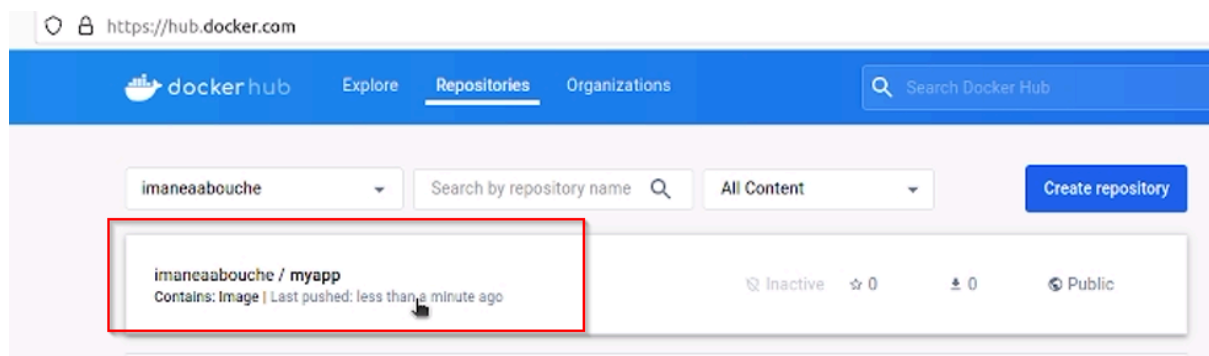


Figure 12. L'image sur Docker Hub

3.5 Pipeline Déploiement de l'application sur le cluster

Dans cette partie, nous avons créé un pipeline qui sert à déployer l'application sur le cluster Kubernetes.

```
1 pipeline {
2   agent any
3   stages {
4     stage('Checkout Source') {
5       steps {
6         git url: 'https://github.com/rihab-ben/Kubernetes_Project.git', branch: 'main'
7       }
8     }
9     stage('Deploy App') {
10      steps {
11        script {
12          kubernetesDeploy(configs: "nginx.yaml", kubeconfigId: "mykubeconfig1")
13        }
14      }
15    }
16  }
17 }
18
```

Figure 13. Le pipeline du déploiement de l'application

Notre pipeline pour le déploiement représente un pipeline déclaratif basique en Jenkins qui effectue deux étapes principales : le checkout du code source depuis un référentiel Git et le déploiement de l'application sur le cluster Kubernetes à l'aide d'un fichier de configuration YAML (**nginx.yaml**).

1. **Agent :**

- agent any spécifie que le pipeline peut être exécuté sur n'importe quel agent disponible dans cluster.

2. **Stages :**

- stages déclare les différentes étapes du pipeline. Dans notre cas, il y a deux étapes :
 - **Checkout Source** : Cette étape utilise le plugin Git pour récupérer le code source depuis le référentiel GitHub spécifié (https://github.com/rihab-ben/Kubernetes_Project.git) et la branche main.
 - **Deploy App** : Cette étape déploie l'application sur le cluster Kubernetes à l'aide du plugin Kubernetes et Kubernetes Continuous Deploy . La configuration de déploiement est spécifiée dans le fichier **nginx.yaml**. Le paramètre **kubeconfigId** fait référence à l'identifiant de la configuration Kubernetes préalablement définie dans Jenkins pour faire la connexion sécurisée entre Jenkins et Kubernetes via des certificats.

Après avoir lancé le build de cette tâche, l'application a été bien déployée sur le cluster

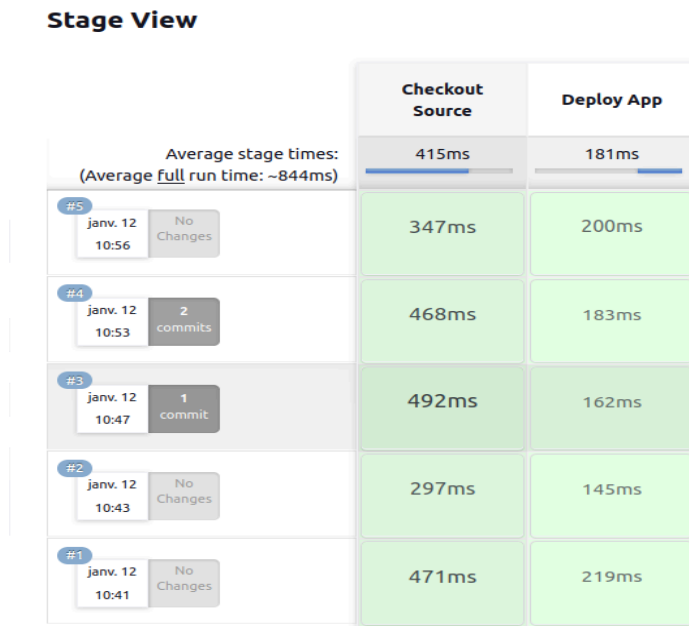


Figure 14. Statut du déploiement sous Jenkins

Nous pouvons aussi vérifier, dans le noeud manager du cluster en exécutant les commandes suivantes:

```
root@mgr1:~#
root@mgr1:~# kubectl get deployments
NAME          READY    UP-TO-DATE    AVAILABLE    AGE
app-m2ssi     1/1      1              1            8h
root@mgr1:~#
```

Figure 15. Les déploiements sous Kubeadm

```
root@mgr1:~#
root@mgr1:~# kubectl get svc
NAME          TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
app-m2ssi     NodePort    10.104.227.234 <none>         80:32226/TCP     8h
kubernetes    ClusterIP   10.96.0.1     <none>         443/TCP          4d1h
root@mgr1:~#
```

Figure 16. Les services sous Kubeadm

4 - Test de fonctionnement

- Push image

Pour créer le pipeline, on a ajouté le lien du repository sur Github

Branch Sources



Git
Project Repository ?

Credentials ?

Figure 17. Lien du Github sous Jenkins

Spécifier le fichier Jenkinsfile sur le Github :

Build Configuration

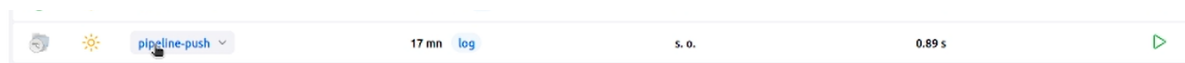
Mode



by Jenkinsfile
Script Path ?

Figure 18. Spécifier le fichier Jenkinsfile sous Jenkins

Lancer le pipeline-push :



pipeline-push 17 min log S. O. 0.89 s

Figure 19. Pipeline Push sous Jenkins

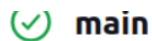
Suivre l'état du push d'image :



État du lanceur de compilations
1 Au repos
2 pipeline-push » main #3 (Push)

Figure 20. Pipeline Push sous Jenkins

Le push est exécuté avec succès :



Full project name: pipeline-push/main

Stage View

	Declarative: Checkout SCM	Build	Login	Push	Declarative: Post Actions
Average stage times: (Average <u>full</u> run time: ~12s)	404ms	2s	1s	6s	335ms
#3 janv. 12 10:56 No Changes	296ms	836ms	1s	2s	345ms
#2 janv. 12 10:52 3 commits	431ms	1s	1s	10s	303ms
#1 janv. 12 10:38 No Changes	486ms	5s	1s	7s	358ms

Figure 21. Statut du Push d'image sous Jenkins

- **Déploiement de l'application**

Lancer le déploiement via le pipeline :



Figure 22. Pipeline du déploiement sous Jenkins

Ci-dessous le script du déploiement :

Pipeline

Definition

Pipeline script

Script ?

```

1 pipeline {
2   agent any
3   stages {
4     stage('Checkout Source') {
5       steps {
6         git url: 'https://github.com/rihab-ben/Kubernetes_Project.git', branch: 'main'
7       }
8     }
9     stage('Deploy App') {
10      steps {
11        script {
12          kubernetesDeploy(configs: "nginx.yaml", kubeconfigId: "mykubeconfig1")
13        }
14      }
15    }
16  }
17 }

```

Figure 23. Script du déploiement sous Jenkins

Suivre l'état de l'exécution :

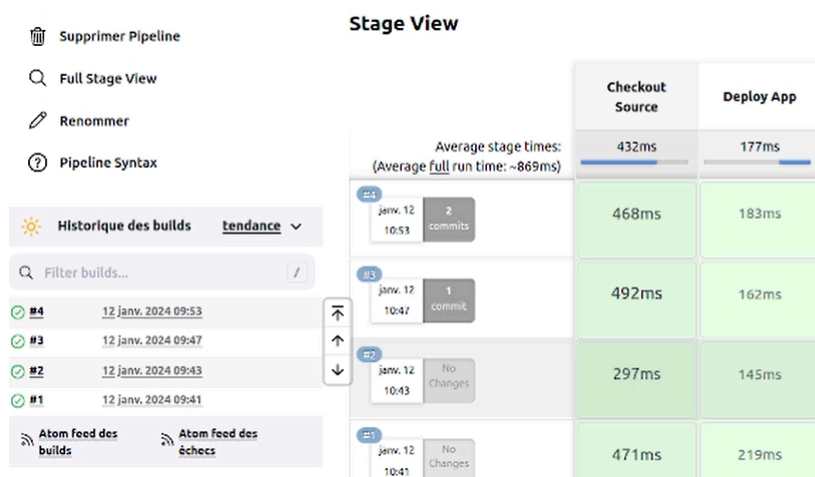


Figure 24. Le statut du déploiement de l'application

L'adresse IP du mgr1 : 10.73.228.6

```

root@mgr1:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp5s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:16:3e:f8:4c:2e brd ff:ff:ff:ff:ff:ff
    inet 10.73.228.6/24 brd 10.73.228.255 scope global dynamic enp5s0
        valid_lft 2156sec preferred_lft 2156sec
    inet6 fd42:5c0a:e504:a347:216:3eff:fef8:4c2e/64 scope global mngtmpaddr noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::216:3eff:fef8:4c2e/64 scope link
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default

```

Figure 25. L'adresse IP du manager 1

Accéder à l'application depuis manager :

← → ↻ 10.73.228.6:32226 ☆ ⌵ ⌵ ⌵ ⌵ ⌵

Résultats du Concours M2SSI 2023-2024

Nom	Prénom	Résultat
Dupont	Jean	85%
Martin	Marie	90%
Petit	Luc	78%
Leroy	Julie	82%
Moreau	François	88%
Lefebvre	Clara	91%

Félicitations à tous les gagnants pour leurs excellentes performances !

Figure 26. L'application depuis manager 1

L'adresse IP du worker1 : 10.73.228.163

```

root@worker1:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp5s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:16:3e:cb:18:3f brd ff:ff:ff:ff:ff:ff
    inet 10.73.228.163/24 brd 10.73.228.255 scope global dynamic enp5s0
        valid_lft 2103sec preferred_lft 2103sec
    inet6 fd42:5c0a:e504:a347:216:3eff:feeb:183f/64 scope global mngtmpaddr noprefixroute

```

Figure 27. L'adresse IP du worker 1

Accéder à l'application depuis worker 1 :

← → ↻ 10.73.228.163:32226 ☆ ⌵ ⌵ ⌵ ⌵ ⌵

Résultats du Concours M2SSI 2023-2024

Nom	Prénom	Résultat
Dupont	Jean	85%
Martin	Marie	90%
Petit	Luc	78%
Leroy	Julie	82%
Moreau	François	88%
Lefebvre	Clara	91%

Félicitations à tous les gagnants pour leurs excellentes performances !

Figure 28. L'application depuis worker 1

Annexes

- Dépôt github: https://github.com/rihab-ben/Kubernetes_Project
- Docker Hub : <https://hub.docker.com/repositories/imaneaabouche>

Webographie

- <https://github.com/justmeandopensource/kubernetes/tree/master/lxd-provisioning>
- <https://www.jenkins.io/doc/book/installing/kubernetes/>
- <https://www.red-gate.com/simple-talk/devops/containers-and-virtualization/deploying-a-dockerized-application-to-the-kubernetes-cluster-using-jenkins/>
- <https://medium.com/codex/jenkins-on-kubernetes-part-1-2fb37c8adb39>
- <https://kubernetes.io/docs/tutorials/hello-minikube/>
- <https://phoenixnap.com/kb/install-kubernetes-on-ubuntu>

Conclusion

En résumé, dans ce projet nous avons appris à adopter une approche Devops visant à déployer une simple application web sur un cluster Kubernetes en utilisant des pipelines Jenkins, cette approche est moderne et efficace pour automatiser le processus de développement, d'intégration continue (CI) et de déploiement continu (CD).

Ce projet DevOps réussi avec Jenkins et Kubernetes nous a donné une idée plus globale de l'automatisation complète du développement logiciel, de l'intégration à la production, en favorisant la collaboration entre les équipes de développement et d'exploitation. Cela conduit à une livraison plus rapide, plus fiable et plus sécurisée des applications web.