




# Implémentation d'une Pipeline CI/CD Scalable avec Jenkins et Azure Kubernetes Service

## Rapport de Mini-Projet

Pour 5ème année Génie Télécommunication et Réseaux

### Réalisé Par :

 BAOUSSOUS Reda  
 ELKADDIRI Mohamed  
 EL HOUMAM SEMLALI Abdelkarim

### Encadré par :

 Pr. DALLI Anouar

---

Soutenu le 05/01/2024

## REMERCIEMENTS

Nous tenons à exprimer notre sincère gratitude envers toutes les personnes qui ont joué un rôle, direct ou indirect, dans la concrétisation de ce projet. Leur contribution a été une source d'inspiration et a grandement enrichi notre parcours.

En particulier, nous adressons nos remerciements les plus chaleureux à Monsieur DALLI Anouar, dont le soutien indéfectible, la disponibilité et les conseils éclairés ont guidé nos efforts. Son expertise a été une boussole précieuse tout au long de cette aventure.

Nous ne pouvons oublier de reconnaître tous ceux qui ont partagé leurs idées, expertise et soutien, créant ainsi un environnement propice à l'innovation et à l'apprentissage. Chaque interaction, qu'elle soit technique ou humaine, a contribué de manière significative à la qualité de notre projet.

C'est avec une profonde gratitude que nous réfléchissons à cette collaboration collective. Le succès de ce projet ne serait pas possible sans la générosité, l'enthousiasme et l'expertise de chacun d'entre vous. Merci infiniment pour votre précieuse contribution à notre réussite.

## RESUME

Ce rapport expose la conception et la réalisation d'un pipeline d'intégration continue et de déploiement continu (CI/CD) visant à répondre aux besoins de solutions évolutives intégrant l'orchestration de conteneurs. Le projet s'appuie sur les technologies Jenkins et Azure Kubernetes Service (AKS) pour automatiser le cycle de vie du logiciel et garantir des déploiements efficaces.

L'importance croissante du CI/CD dans le développement logiciel moderne est soulignée, mettant en avant la nécessité de livraisons rapides et fiables. Le choix stratégique de Jenkins, outil d'automatisation bien établi, conjugué à l'utilisation d'Azure Kubernetes Service pour l'orchestration de conteneurs, offre une solution robuste et évolutive.

La structure du rapport comprend une introduction détaillée, une vue d'ensemble de l'architecture du projet, des instructions de démarrage rapide avec déploiement manuel et Terraform, ainsi que la mise en œuvre complète du pipeline. La configuration de Jenkins, l'exemple de Jenkinsfile, la vérification des déploiements, et les perspectives d'avenir sont également abordées.

Ce travail permet d'appréhender les avantages de l'automatisation du CI/CD, en offrant une solution pratique pour le déploiement d'applications sur des environnements basés sur des conteneurs. Les résultats obtenus soulignent l'efficacité du pipeline mis en place, ouvrant la voie à des améliorations futures et à l'intégration de mesures de sécurité supplémentaires.

# TABLE DES MATIERES

<b>REMERCIEMENTS .....</b>	<b>1</b>
<b>RESUME .....</b>	<b>2</b>
<b>TABLE DES MATIERES .....</b>	<b>3</b>
<b>INTRODUCTION GENERALE.....</b>	<b>4</b>
<b>CHAPITRE 1 : CONTEXTE ET PROBLÉMATIQUE.....</b>	<b>5</b>
1.    INTRODUCTION :.....	5
2.    CONTEXTE DU PROJET : .....	5
3.    PROBLÉMATIQUE : .....	5
4.    OBJECTIFS : .....	5
<b>CHAPITRE 2 : ARCHITECTURE GLOBALE .....</b>	<b>6</b>
1.    INTRODUCTION : .....	6
2.    VUE D'ENSEMBLE DE L'ARCHITECTURE .....	6
3.    DIAGRAMME D'INTERACTION ENTRE LES COMPOSANTS .....	7
<b>CHAPITRE 3 : MISE EN PLACE DE PROJET .....</b>	<b>9</b>
1.    DESCRIPTION DE L'APPLICATION : .....	9
2.    CREATION D'UNE INSTANCE EC2 ET L'INSTALLATION DE JENKINS, DOCKER ET AZURE CLI .....	10
3.    AUTOMATISATION DE LA CREATION D'UNE MACHINE VIRTUELLE AZURE AVEC TERRAFORM ET INSTALLATION DE JENKINS AVEC ANSIBLE .....	12
4.    LA MISE EN PLACE DU PROJECT ON GITHUB : .....	15
5.    DEPLOIEMENT MANUEL DE L'APPLICATION DANS AKS AVEC DOCKER COMPOSE : .....	17
a. <i>Introduction</i> :.....	17
b. <i>Étapes de déploiement</i> :.....	17
6.    AUTOMATISATION DU DEPLOIEMENT CI/CD DE L'APPLICATION AVEC JENKINS, DOCKER ET KUBERNETES/HELM : .....	23
7.    AUTOMATISATION DU DEPLOIEMENT D'UNE APPLICATION SUR AZURE KUBERNETES SERVICES (AKS) VIA JENKINS:.....	25
CONCLUSION.....	27

# INTRODUCTION GENERALE

L'évolution rapide du paysage technologique a transformé la manière dont les applications sont développées, testées et déployées. Dans ce contexte, l'intégration continue (CI) et le déploiement continu (CD) se sont affirmés comme des piliers essentiels pour les équipes de développement logiciel. Ce rapport explore la conception et la mise en œuvre d'un pipeline CI/CD spécifique, mettant en lumière l'utilisation conjointe de deux technologies phares : Jenkins, un outil d'automatisation, et Azure Kubernetes Service (AKS), une solution d'orchestration de conteneurs. L'objectif est de fournir une solution robuste et évolutive, alignée sur les besoins contemporains du développement logiciel.

Dans le contexte dynamique de la technologie moderne, les exigences accrues en termes de fréquence et de fiabilité des déploiements logiciels ont conduit à une refonte des pratiques de développement. L'intégration continue, en établissant un processus automatisé de compilation, d'emballage et de test des applications, ainsi que le déploiement continu, garantissant la propagation automatisée des changements de code, sont devenus essentiels pour maintenir un rythme de développement efficace et assurer une collaboration optimale.

L'accent de ce rapport est mis sur l'utilisation simultanée de Jenkins et d'AKS, offrant une approche intégrée pour répondre aux défis de l'intégration continue et du déploiement continu. Jenkins, en tant qu'outil d'automatisation de renom, facilite la création d'un pipeline robuste, tandis qu'AKS fournit une solution d'orchestration de conteneurs, permettant une gestion efficace des environnements de déploiement.

La suite de ce rapport détaillera le contexte du projet, les objectifs visés, les choix stratégiques de technologie, l'architecture générale du système, ainsi que les étapes de mise en œuvre du pipeline CI/CD. L'analyse des résultats obtenus, les perspectives d'avenir, et les références utilisées compléteront ce document, offrant une vision complète de la réalisation de ce projet ambitieux.

# CHAPITRE 1 : CONTEXTE ET PROBLÉMATIQUE

## 1. INTRODUCTION :

L'évolution constante du domaine technologique a profondément modifié les pratiques de développement logiciel. Afin de rester compétitives et de répondre aux exigences croissantes du marché, les entreprises adoptent des approches innovantes telles que l'intégration continue (CI) et le déploiement continu (CD). Ce chapitre établit le socle sur lequel repose notre projet, en présentant le contexte dynamique du développement logiciel moderne, soulignant l'importance cruciale de l'intégration continue et du déploiement continu.

## 2. CONTEXTE DU PROJET :

Le contexte du projet s'inscrit dans la nécessité de rationaliser les processus de développement, de test et de déploiement des applications. Les équipes de développement font face à la pression croissante de livrer des fonctionnalités de manière rapide et fiable. L'utilisation de pratiques traditionnelles peut entraîner des retards, des erreurs et des coûts supplémentaires. C'est dans ce contexte que notre projet trouve sa pertinence, en visant à instaurer un pipeline CI/CD performant grâce à l'utilisation conjointe de Jenkins et d'Azure Kubernetes Service (AKS).

## 3. PROBLÉMATIQUE :

La problématique centrale réside dans la nécessité d'optimiser les processus de développement pour répondre aux impératifs de rapidité et de fiabilité. Les méthodes traditionnelles de déploiement peuvent entraver la capacité des équipes à s'adapter aux changements fréquents des exigences du marché. La question clé est de savoir comment instaurer un environnement d'intégration continue et de déploiement continu capable d'améliorer l'efficacité du cycle de développement, de favoriser la collaboration entre les équipes et de garantir une qualité logicielle optimale.

## 4. OBJECTIFS :

Les objectifs de ce projet se déclinent en plusieurs axes. Tout d'abord, il vise à établir un pipeline CI/CD robuste en utilisant Jenkins et AKS. Ensuite, il aspire à accélérer le processus de livraison des applications tout en garantissant la stabilité du système. De plus, le projet s'attache à réduire les coûts associés aux retards de livraison et aux erreurs de déploiement. Ces objectifs seront détaillés et évalués tout au long du rapport, avec des illustrations visuelles pour faciliter la compréhension.

## CHAPITRE 2 : ARCHITECTURE GLOBALE

### 1. Introduction :

La conception d'une architecture solide et évolutive est un élément central de tout projet axé sur l'intégration continue et le déploiement continu (CI/CD). Ce chapitre se penche sur les fondements de l'architecture globale de notre projet, mettant en lumière les choix clés qui orientent la mise en place du pipeline CI/CD.

### 2. Vue d'Ensemble de l'Architecture

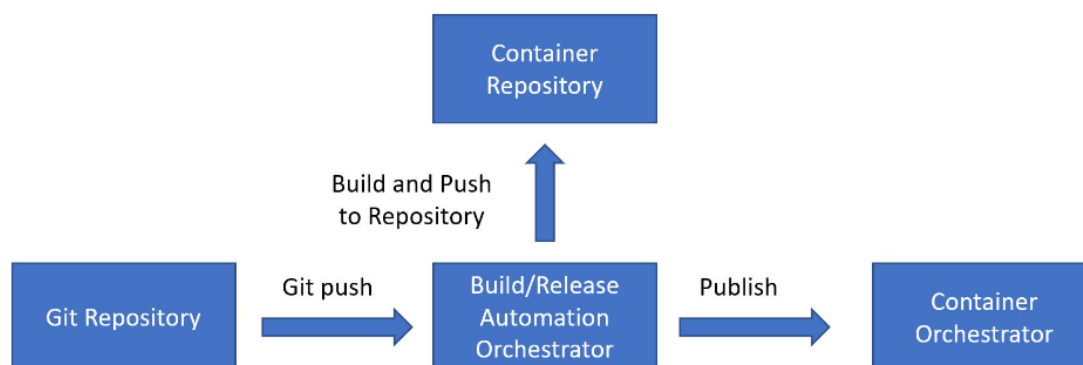
L'architecture globale de notre projet est conçue pour créer un écosystème fluide où les différentes phases du processus CI/CD sont orchestrées de manière efficace. Voici un aperçu détaillé de la façon dont ces éléments interagissent pour soutenir le processus continu d'intégration et de déploiement :

- **GitHub** : Notre référentiel central pour le code source. Les développeurs effectuent des commits sur GitHub, déclenchant ainsi le processus d'intégration continue.
- **Jenkins** : L'outil d'automatisation qui coordonne le pipeline CI/CD. Jenkins surveille le référentiel GitHub et déclenche automatiquement des builds à chaque modification du code source.
- **Azure Container Registry (ACR)** : La registry qui stocke les images Docker générées lors du processus de build. Jenkins pousse les images Docker réussies vers ACR pour les rendre disponibles pour le déploiement ultérieur.
- **Azure Kubernetes Service (AKS)** : La plateforme d'orchestration de conteneurs qui gère le déploiement et la mise à l'échelle des conteneurs. Une fois les images disponibles dans ACR, AKS orchestre le déploiement sur les nœuds du cluster.

Cette interaction harmonieuse entre GitHub, Jenkins, ACR, et AKS forme la colonne vertébrale de notre architecture globale, permettant une livraison continue et fiable des applications. Le diagramme ci-dessous offre une représentation visuelle de cette architecture, mettant en évidence les flux de données entre les différents composants.

### 3. Diagramme d'Interaction entre les Composants

L'architecture globale de notre projet repose sur une conception robuste et évolutive, mettant en œuvre une synergie entre plusieurs composants clés. Le diagramme d'interaction ci-dessous offre une vue d'ensemble de la manière dont ces composants interagissent pour assurer le bon fonctionnement du pipeline CI/CD.



*Figure 1: diagramme d'interaction entre les composants*

#### Description du Diagramme :

Le diagramme d'interaction entre les composants dévoile les rôles essentiels de chacun des éléments majeurs dans notre architecture CI/CD. GitHub, en tant que point central de gestion du code source, représente le cœur de notre projet en hébergeant l'ensemble du code et en assurant la traçabilité des versions. Les développeurs interagissent avec GitHub pour effectuer des modifications et proposer des améliorations.

Jenkins, en tant qu'outil d'automatisation clé, joue un rôle central dans la coordination du processus CI/CD. Il surveille en permanence les modifications apportées à notre code source sur GitHub. Dès qu'une modification est détectée, Jenkins prend l'initiative de déclencher le pipeline CI/CD. Cela marque le début d'une série d'étapes automatisées, allant de la compilation du code à la création d'images Docker, en passant par le déploiement sur Azure Kubernetes Service (AKS).

Azure Container Registry (ACR) sert de référentiel centralisé pour stocker les images Docker générées lors du processus d'intégration continue. Ces images, contenant des versions spécifiques de l'application, sont préservées dans ACR, prêtes à être déployées lorsque nécessaire. Ainsi, ACR offre une gestion efficace des images Docker, facilitant le déploiement cohérent de l'application.



Azure Kubernetes Service (AKS) intervient dans la phase de déploiement et de gestion des conteneurs dans l'environnement Azure. En plus de fournir une orchestration efficace des conteneurs, AKS assure une mise à l'échelle automatique pour répondre à la demande fluctuante de l'application. Cette solution offre également une gestion intelligente des ressources, garantissant une utilisation optimale des capacités disponibles dans l'environnement cloud. En somme, AKS constitue l'infrastructure robuste qui permet l'exécution fluide de l'application dans un environnement de conteneurs élastique et dynamique.

**Fonctionnement Global :**

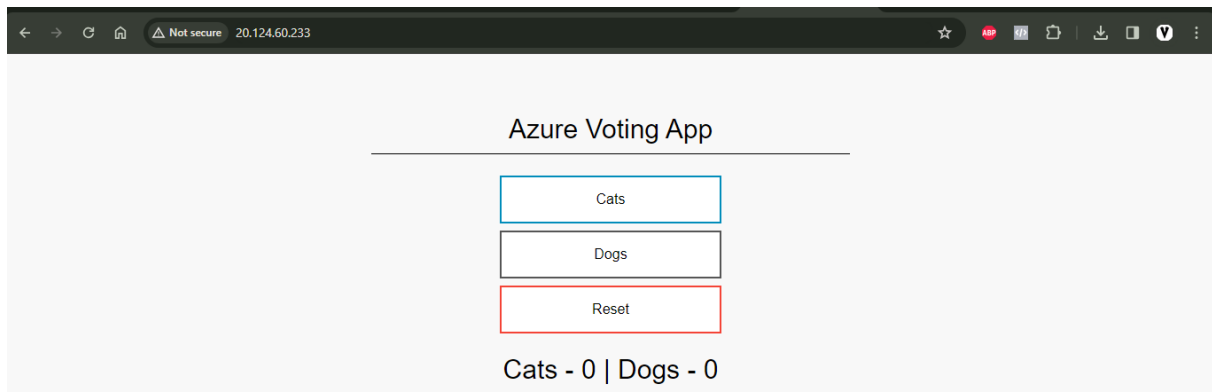
1. Lorsqu'un développeur effectue des modifications sur GitHub, Jenkins détecte ces changements.
2. Jenkins déclenche le pipeline CI/CD, démarrant la compilation, la création d'images Docker et le déploiement sur AKS.
3. Les images Docker sont stockées dans Azure Container Registry pour une référence future.
4. AKS déploie et gère les conteneurs, assurant ainsi la disponibilité de l'application.

Ce diagramme offre une vision détaillée de l'interaction harmonieuse entre les composants clés du projet, démontrant la fluidité du processus de CI/CD et la collaboration efficace entre les différents outils utilisés.

## CHAPITRE 3 : MISE EN PLACE DE PROJET

### 1. Description de l'Application :

L'application est accessible via une interface web simple et ergonomique. Elle présente trois boutons principaux permettant aux utilisateurs de voter pour leur animal de compagnie préféré. Les deux premiers boutons sont associés respectivement au chien et au chat, tandis que le troisième bouton permet de réinitialiser les votes.



#### Fonctionnalités Principales :

- Boutons de Vote :

Deux boutons distincts sont disponibles, l'un pour voter en faveur du chien et l'autre pour voter en faveur du chat.

Chaque bouton déclenche une action de vote lorsque l'utilisateur clique dessus.

- Bouton de Réinitialisation :

Un bouton dédié à la réinitialisation des votes permet aux utilisateurs de remettre à zéro le compteur de votes accumulés.

- Compteur de Votes :

Un affichage numérique sous les boutons indique le nombre total de votes enregistrés.

#### Utilisation de l'Application :

L'utilisateur peut exprimer sa préférence en cliquant sur le bouton associé à l'animal de compagnie de son choix. Chaque vote est pris en compte, et le compteur est mis à jour en temps réel. Si un utilisateur souhaite changer son vote ou réinitialiser complètement les résultats, le bouton de réinitialisation est disponible à cet effet.

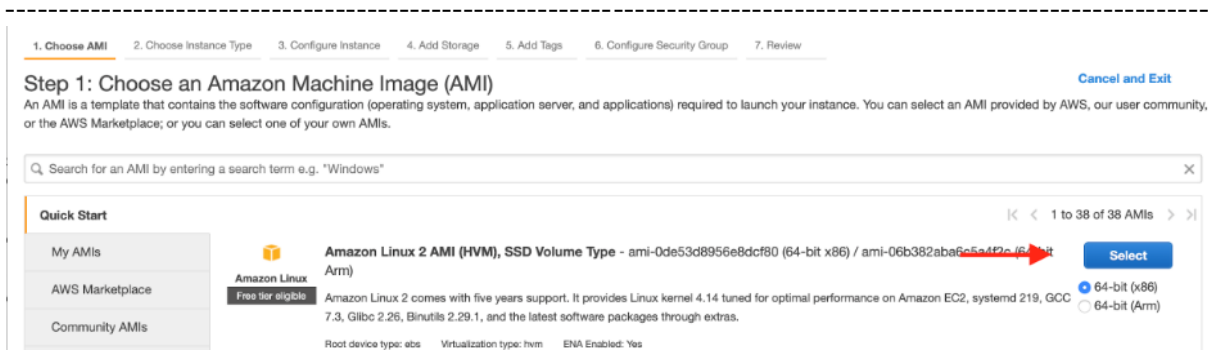
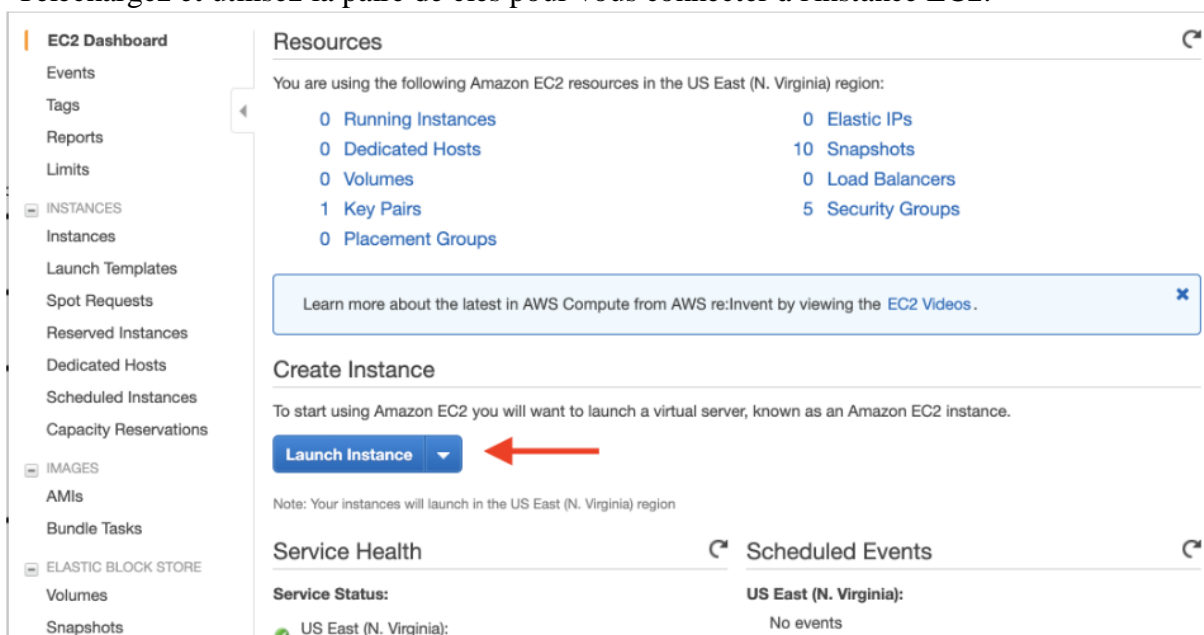
## 2. Création d'une Instance EC2 et l'Installation de Jenkins, Docker et Azure CLI

### Introduction :

L'objectif de cette section est de détailler les étapes nécessaires pour déployer une instance EC2 sur AWS, installer Jenkins, et configurer les outils supplémentaires tels que Docker et Azure CLI. Ces outils sont cruciaux pour la mise en place d'un environnement de développement et de déploiement robuste.

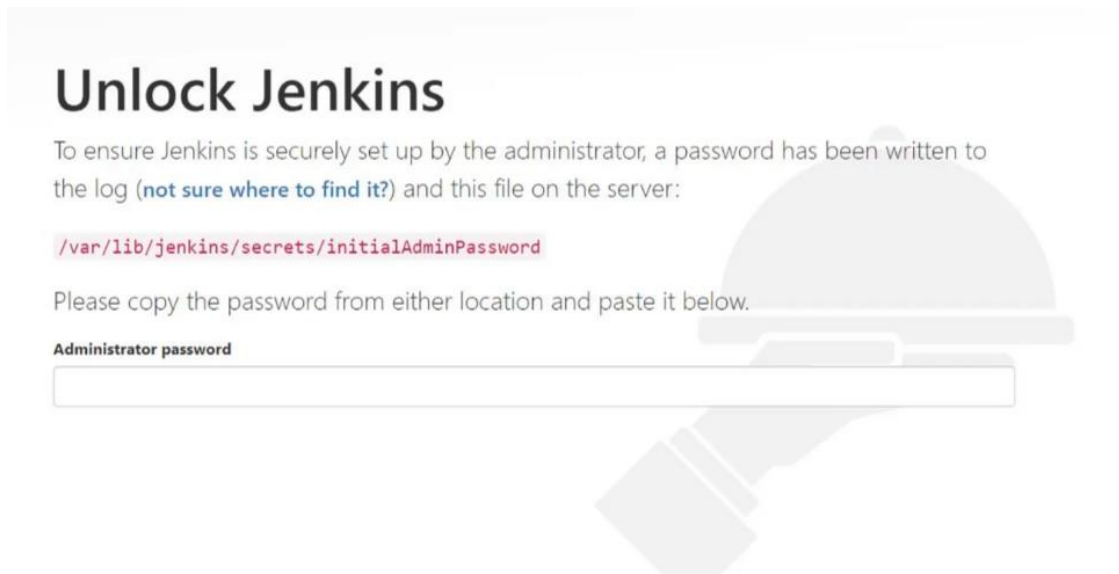
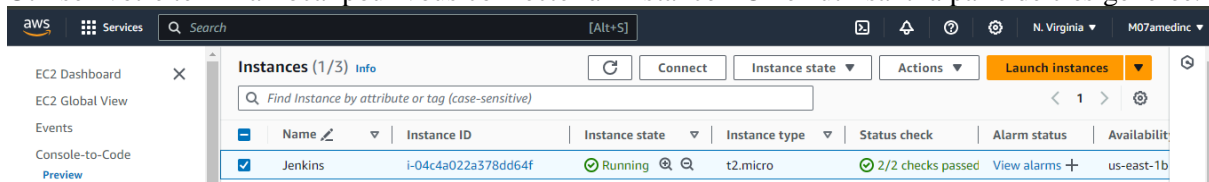
### Étapes détaillées :

- Création de l'Instance EC2 :
  - Connectez-vous à la console AWS.
  - Accédez au service EC2 et cliquez sur "Lancer une instance".
  - Sélectionnez une image Amazon Machine Image (AMI) compatible avec vos besoins.
  - Choisissez le type d'instance, configurez les détails, ajoutez du stockage, et configurez les règles de sécurité (groupes de sécurité).
  - Téléchargez et utilisez la paire de clés pour vous connecter à l'instance EC2.



### Connexion à l'Instance EC2 :

Utilisez votre terminal local pour vous connecter à l'instance EC2 en utilisant la paire de clés générée.



Après on vas installer Docker, Java et Azure CLI dans le serveur avec les commandes suivants :

```
sudo yum install -y docker
```

```
sudo yum install -y java-1.8.0
```

```
wget -q https://aka.ms/InstallAzureCLIDeb  
sudo bash InstallAzureCLIDeb
```

Ce processus complet de création d'une instance EC2, d'installation de Jenkins, de Docker et de Azure CLI offre une base solide pour la mise en place d'un environnement de développement et de déploiement. Ces étapes peuvent être adaptées en fonction des besoins spécifiques du projet et des exigences de sécurité. N'oubliez pas de documenter les détails spécifiques à votre environnement.

### 3. Automatisation de la Création d'une Machine Virtuelle Azure avec Terraform et Installation de Jenkins avec Ansible

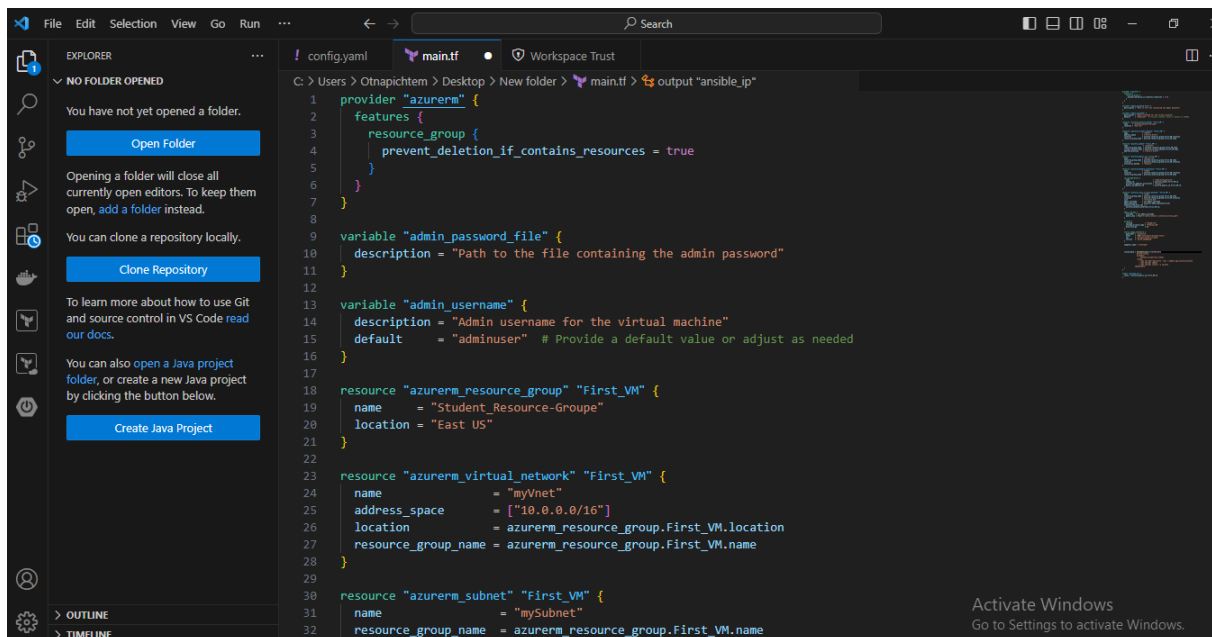
Dans cette étape du projet, nous avons entrepris de réinstaller Jenkins, mais cette fois-ci, nous avons opté pour une approche entièrement automatisée. En utilisant des outils d'automatisation tels qu'Ansible, le processus de réinstallation de Jenkins a été orchestré de bout en bout. Les tâches manuelles antérieures associées à la configuration et à l'installation de Jenkins ont été éliminées, et à la place, un ensemble de playbooks Ansible a été créé pour gérer de manière cohérente l'ensemble du processus. Cette automatisation a non seulement accéléré la réinstallation de Jenkins, mais elle a également réduit les risques d'erreurs humaines, assurant ainsi une configuration homogène et fiable du serveur Jenkins. Cette approche alignée sur les bonnes pratiques d'automatisation a démontré son efficacité en optimisant le déploiement de Jenkins, contribuant ainsi à la fluidité et à la robustesse de notre environnement de développement.

#### Introduction :

Ce rapport documente la mise en œuvre réussie d'une automatisation complète de déploiement, depuis la création d'une machine virtuelle (VM) sur Microsoft Azure jusqu'à l'installation et la configuration de Jenkins à l'aide de Terraform et Ansible respectivement.

#### Création de la Machine Virtuelle avec Terraform :

La première phase du processus a consisté à créer une machine virtuelle Azure en utilisant Terraform. Les fichiers de configuration Terraform ont été développés pour définir les ressources nécessaires, y compris la taille de la VM, le réseau, et d'autres paramètres spécifiques au déploiement.



```
1 provider "azurerm" {
2   features {
3     resource_group {
4       prevent_deletion_if_contains_resources = true
5     }
6   }
7 }
8
9 variable "admin_password_file" {
10  description = "Path to the file containing the admin password"
11 }
12
13 variable "admin_username" {
14  description = "Admin username for the virtual machine"
15  default     = "adminuser" # Provide a default value or adjust as needed
16 }
17
18 resource "azurerm_resource_group" "First_VM" {
19   name     = "Student_Resource-Group"
20   location = "East US"
21 }
22
23 resource "azurerm_virtual_network" "First_VM" {
24   name            = "myVnet"
25   address_space   = ["10.0.0.0/16"]
26   location         = azurerm_resource_group.First_VM.location
27   resource_group_name = azurerm_resource_group.First_VM.name
28 }
29
30 resource "azurerm_subnet" "First_VM" {
31   name            = "mySubnet"
32   resource_group_name = azurerm_resource_group.First_VM.name
```

## Installation et Configuration de Jenkins avec Ansible :

Une fois la VM déployée, Ansible a été employé pour automatiser l'installation et la configuration de Jenkins sur le serveur. Des playbooks Ansible ont été développés pour gérer l'ensemble du processus, couvrant l'installation des dépendances, la configuration du serveur Jenkins, et l'activation des plugins requis.

```

1 ---
2 - name: Install jenkins
3   hosts: localhost
4   become: yes
5   become_user: root
6
7   tasks:
8     - name: Update all packages to their latest version
9       apt:
10        name: "*"
11        state: latest
12
13     - name: download jenkins key
14       ansible.builtin.get_url:
15         url: https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
16         dest: /usr/share/keyrings/jenkins-keyring.asc
17
18     - name: Add Jenkins repo
19       ansible.builtin.apt_repository:
20         repo: deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/
21         state: present
22         filename: jenkins.list
23
24     - name: Update all packages to their latest version
25       apt:
26        name: "*"
27        state: latest
28
29     - name: Install fontconfig
30       shell: apt install fontconfig -y
31
32     - name: Install java
  
```

**Microsoft Azure** Search resources, services, and docs (G+)

Home > Virtual machines > ngx-plus-1

**ngx-plus-1**  
Virtual machine

Search (Cmd+/) << Connect Start Restart Stop Capture Delete Refresh

Resource group (change)	Azure Spot
NGINX-Plus-HA	N/A
Status	Public IP address
Running	13.66.134
Location	Private IP address
West US 2	10.0.0.4
Subscription (change)	Public IP address (IPv6)
NGINX-Plus-HA-subscription	-

Overview  
Activity log  
Access control (IAM)  
Tags  
Diagnose and solve problems

## Résultats et Avantages :

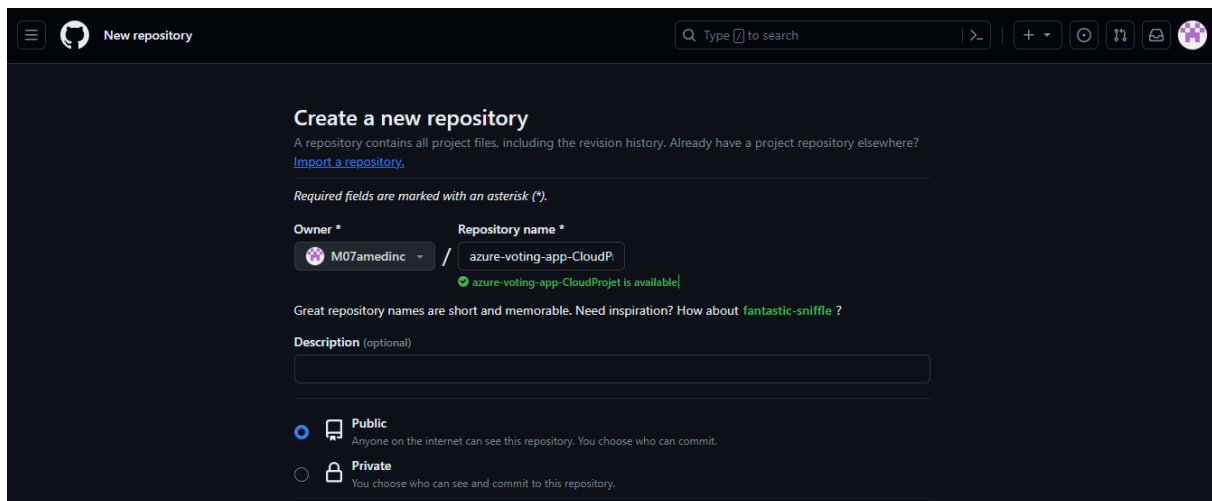
- L'automatisation du déploiement avec Terraform a permis de créer rapidement et de manière reproductible une machine virtuelle Azure, garantissant une cohérence entre les environnements.
- L'utilisation d'Ansible pour installer et configurer Jenkins a facilité le processus, réduisant les erreurs humaines et assurant une configuration uniforme du serveur Jenkins

**Conclusion :**

Cette partie met en lumière la réussite de l'automatisation complète du déploiement d'une machine virtuelle Azure avec Terraform, suivie de l'installation et de la configuration efficaces de Jenkins à l'aide d'Ansible. Ces approches d'automatisation offrent des avantages significatifs en termes de rapidité, de cohérence et de fiabilité dans le déploiement d'infrastructures et d'applications.

## 4. La mise en place du Project on GitHub :

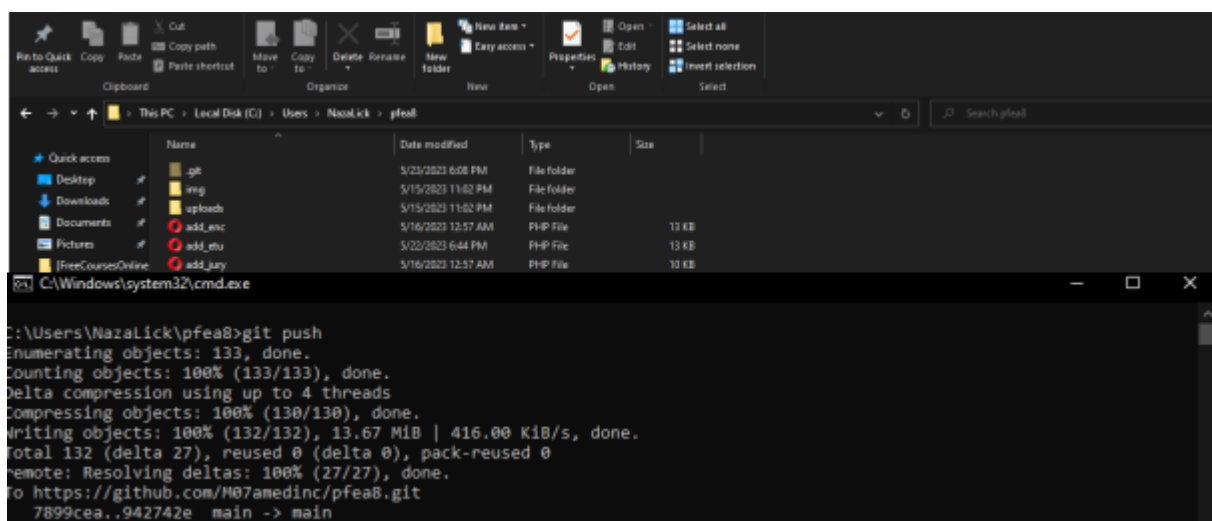
La Création d'un compte GitHub et d'un nouveau dépôt (Repository) :



Clonez le dépôt sur l'ordinateur : Pour commencer à travailler sur le projet localement, on doit cloner le dépôt sur l'ordinateur. On Utilise la commande git clone suivie de l'URL du dépôt pour le cloner localement.

```
C:\Users\NazaLick>git clone https://github.com/M07amedinc/pfea8.git
Cloning into 'pfea8'...
remote: Enumerating objects: 153, done.
remote: Counting objects: 100% (153/153), done.
remote: Compressing objects: 100% (106/106), done.
remote: Total 153 (delta 44), reused 150 (delta 44), pack-reused 0
10/s
Receiving objects: 100% (153/153), 13.68 MiB | 583.00 KiB/s, done.
Resolving deltas: 100% (44/44), done.
```

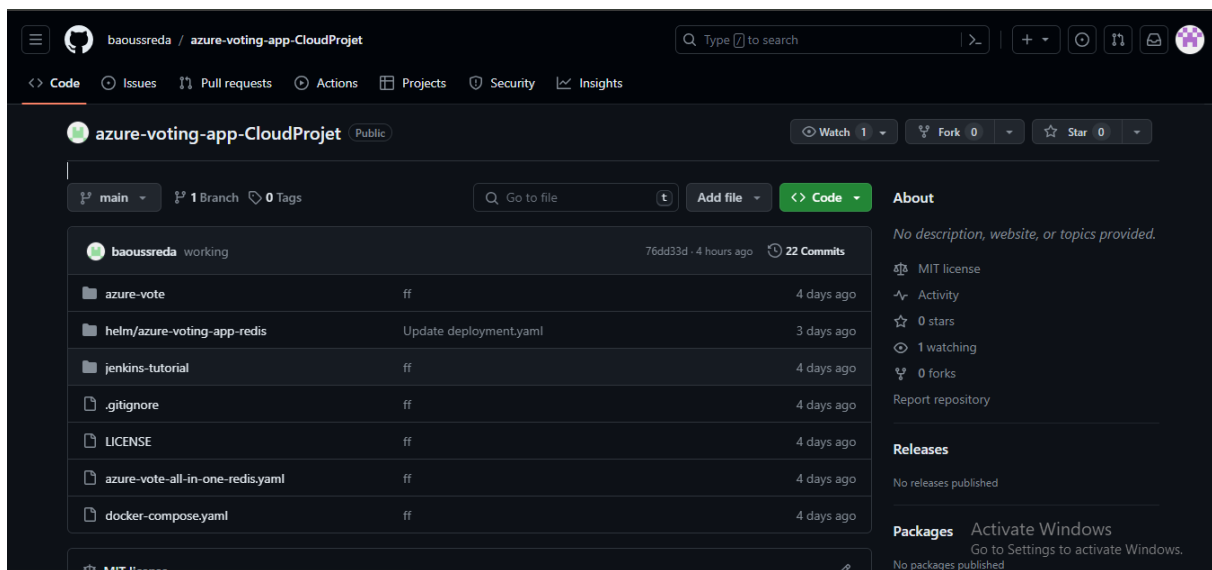
Comme le dossier contient .git dossier alors on peut commencer





Une fois le dépôt cloné, chaque membre du groupe de développement peut apporter des modifications au code, ajouter de nouvelles fonctionnalités ou résoudre des problèmes. Les membres peuvent pousser leurs modifications vers le dépôt distant en utilisant les commandes Git appropriées, en utilisant GIT COMMIT et GIT PUSH

Et voilà l'équipe va travailler efficacement ensemble sur le projet, même s'ils sont géographiquement dispersés.



## 5. Déploiement Manuel de l'Application dans AKS avec Docker Compose :

### a. Introduction :

Le déploiement manuel d'une application sur Azure Kubernetes Service (AKS) avec Docker Compose constitue une étape cruciale dans le processus d'implémentation d'une infrastructure moderne et évolutive. Cette approche permet de comprendre en détail les étapes du déploiement avant d'automatiser le processus avec des outils tels que Jenkins pour la CI/CD.

Dans cette section, nous explorerons de manière détaillée chaque étape du déploiement manuel. De la création du groupe de ressources Azure à la configuration de kubectl pour interagir avec le cluster AKS, nous aborderons les étapes nécessaires à la mise en place d'une application conteneurisée. Le fichier Docker Compose `azure-vote.yaml` servira de guide pour orchestrer les différents services de l'application.

Cette approche manuelle offre une opportunité d'approfondir notre compréhension du déploiement sur AKS, tout en jetant les bases nécessaires pour la mise en œuvre ultérieure de pipelines CI/CD automatisés. En suivant attentivement chaque étape, nous serons en mesure de déployer avec succès l'application multi-conteneurs et de garantir son bon fonctionnement sur le cluster AKS.

### b. Étapes de déploiement :

#### i. Création du groupe de ressources Azure :

Commencez par créer un groupe de ressources dans Azure pour organiser et gérer les ressources liées au déploiement de l'application.

[Home](#) >

## Create a resource ...

Get Started

Recently created

### Categories


AI + Machine Learning

Analytics

Blockchain

Compute

Containers

 Search services and marketplace



Popular Azure services [See more in All services](#)



**Azure Kubernetes Service (AKS)**

[Create](#) | [Docs](#) | [MS Learn](#)



**Web App for Containers**

[Create](#) | [Docs](#) | [MS Learn](#)



**Batch Service**

[Create](#) | [Docs](#) | [MS Learn](#)

## ii. Création du Cluster AKS :

Déployez un cluster AKS pour héberger l'application conteneurisée. Pour cet exemple, nous utilisons un seul nœud de travail, mais cela peut être adapté en fonction des exigences de l'application.

[Home](#) > [Create a resource](#) >

### Create Kubernetes cluster ...

#### Basics

#### Node pools

#### Networking

#### Integrations

#### Monitoring

#### Advanced

#### Tags

#### Review

Subscription \* ⓘ

Azure for Students



Resource group \* ⓘ

Student\_Resource-Groupe

[Create new](#)

#### Cluster details

Cluster preset configuration \*

Production Economy

To quickly customize your Kubernetes cluster, choose one of the preset configurations above. You can modify these configurations at any time.

[Compare presets](#)

Kubernetes cluster name \* ⓘ

ASH\_Cluster

Region \* ⓘ

(US) East US

Availability zones ⓘ

None

AKS pricing tier ⓘ

Free

### Create Kubernetes cluster ...

#### Basics

#### Node pools

#### Networking

#### Integrations

#### Monitoring

#### Advanced

#### Tags

#### Review + create

#### Node pools

In addition to the required primary node pool configured on the Basics tab, you can also add optional node pools to handle a variety of workloads [Learn more](#)

[+](#) Add node pool [Delete](#)

<input type="checkbox"/>	Name	Mode	Node size	OS SKU	Node count	Availability
<input type="checkbox"/>	agentpool	System	Standard_D8ds_v5 ...	Ubuntu	2 - 5	None
<input type="checkbox"/>	userpool	User	Standard_D8as_v4 ...	Ubuntu	0 - 25	None

User node pool is recommended for production economy configuration with maximum of 25 nodes and a default value of 3 nodes.

[Home](#) > [Create a resource](#) > [Create Kubernetes cluster](#) >

## Update node pool ...

ASH\_Cluster

Node pool name \* ⓘ

agentpool

Mode \* ⓘ

☐ User☒ System

**i** The primary node pool must be a system node pool to support system pods.

OS SKU \* ⓘ

☐ Azure Linux☒ Ubuntu Linux☐ Windows

**i** Linux is required for system node pools.

Availability zones ⓘ

None

Enable Azure Spot instances ⓘ



**i** Azure Spot instances cannot be used with system node pools.

Node size \* ⓘ

**Standard B2s**

2 vcpus, 4 GiB memory

[Choose a size](#)

Scale method ⓘ

☐ Manual☒ Autoscale - **Recommended**

**✓** This option is recommended so that the cluster is automatically sized correctly for the current running workloads.

Minimum node count \* ⓘ

1



Maximum node count \* ⓘ

1



The maximum node count allowed for an AKS cluster is 1000 per node pool and 5000 nodes across all node pools in this cluster.

### Optional settings

Max pods per node \* ⓘ

110



30 - 250

**Update**

Cancel

## Update node pool ...

ASH\_Cluster

Node pool name * ⓘ	<input type="text" value="userpool"/>
Mode * ⓘ	<input checked="" type="radio"/> User <input type="radio"/> System
OS SKU * ⓘ	<input type="radio"/> Azure Linux <input checked="" type="radio"/> Ubuntu Linux <input type="radio"/> Windows
Availability zones ⓘ	<input type="text" value="None"/>
Enable Azure Spot instances ⓘ	<input checked="" type="checkbox"/>
Azure Spot configuration * ⓘ	<b>Standard D2s v3</b> - 2 vcpus, 8 GiB memory (\$0.0096/hour) <b>Eviction type:</b> Capacity <b>Eviction policy:</b> Delete <a href="#">Configure</a>
Scale method ⓘ	<input type="radio"/> Manual <input checked="" type="radio"/> Autoscale - <b>Recommended</b> <input checked="" type="checkbox"/> This option is recommended so that the cluster is automatically sized correctly for the current running workloads.
Minimum node count * ⓘ	<input type="text" value="0"/> ✓
Maximum node count * ⓘ	<input type="text" value="2"/> ✓ The maximum node count allowed for an AKS cluster is 1000 per node pool and 5000 nodes across all node pools in this cluster.
Optional settings	
Max pods per node * ⓘ	<input type="text" value="110"/> ✓ 10 - 250

### iii. Configuration de kubectl :

Configurez kubectl pour interagir avec le cluster AKS nouvellement créé.

```
PowerShell | ? | ⚙ | ↕ | 📄 | {} | 📄
MOTD: SqlServer has been updated to Version 22!

VERBOSE: Authenticating to Azure ...
VERBOSE: Building your Azure drive ...
PS /home/abdelkarim> az aks get-credentials --resource-group Student_Resource-Groupe --name ASH_Cluster
Merged "ASH_Cluster" as current context in /home/abdelkarim/.kube/config
```

*iv. Déploiement Manuel avec Kubernetes YAML file :*

Déployez l'application manuellement en utilisant le fichier Docker Compose azure-vote.yaml. Assurez-vous d'avoir le fichier sur votre machine locale.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote
  template:
    metadata:
      labels:
        app: azure-vote
    spec:
      containers:
        - name: azure-vote-front
          image: azure-vote-front
          ports:
            - containerPort: 80
        - name: azure-vote-back
          image: redis
          ports:
            - containerPort: 6379
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote
spec:
  selector:
    app: azure-vote
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

Déployez l'application avec la commande suivante :

```
PS /home/abdelkarim> vi azure_app.yml
PS /home/abdelkarim> kubectl apply -f azure_app.yml
deployment.apps/azure-vote-back created
service/azure-vote-back created
deployment.apps/azure-vote-front created
service/azure-vote-front created
PS /home/abdelkarim> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
azure-vote-back-8fd8d8db4-dm5z8     0/1     Pending   0           11s
azure-vote-front-7b9885b5d9-4trmn   0/1     Pending   0           10s
PS /home/abdelkarim> kubectl get pods
```

#### v. Vérification du Déploiement :

Vérifiez le statut du déploiement en listant les PODS et les services. Assurez-vous que tous les conteneurs sont en cours d'exécution.

```
PS /home/abdelkarim> kubectl get nodes
To sign in, use a web browser to open the page https://microsoft.com/devicelogin?code=E0104 14:24:02.522819 574 memcache.go:265] couldn't get current server status: http://localhost:8080/healthz: dial tcp 127.0.0.1:8080: connect: connection refused
DeadlineExceeded (Client.Timeout exceeded while awaiting headers)
NAME                                STATUS    ROLES    AGE   VERSION
aks-agentpool-17189871-vmss000000 Ready     agent    9m8s  v1.27.7
PS /home/abdelkarim> kubectl get nodes
NAME                                STATUS    ROLES    AGE   VERSION
aks-agentpool-17189871-vmss000000 Ready     agent    9m15s  v1.27.7
```

L'application devrait être accessible via l'adresse IP externe fournie par le service LoadBalancer.

```
PS /home/abdelkarim> kubectl get svc
NAME                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
azure-vote-back     ClusterIP   10.0.165.122  <none>         6379/TCP         28s
azure-vote-front    LoadBalancer 10.0.148.70   4.157.219.89   80:30303/TCP     27s
kubernetes          ClusterIP   10.0.0.1      <none>         443/TCP          5h14m
PS /home/abdelkarim>
```

Ces étapes détaillent le déploiement manuel de l'application multi-conteneurs sur AKS en utilisant Kubernetes YAML file. Cette approche permet une compréhension approfondie du processus de déploiement avant d'automatiser davantage avec des outils tels que Jenkins pour la CI/CD.

## 6. Automatisation du Déploiement CI/CD de l'Application avec Jenkins, Docker et Kubernetes/Helm :

Le présent rapport décrit en détail la mise en place d'un pipeline d'intégration continue et de déploiement continu (CI/CD) pour l'application. Ce processus est automatisé à l'aide de Jenkins, Docker et Helm, permettant ainsi une gestion efficace du cycle de vie de développement et de déploiement de l'application.

**Les principaux objectifs de cette automatisation sont les suivants :**

- Récupération du Code Source : Clonage du code source depuis le référentiel Git <https://github.com/baoussreda/azure-voting-app-CloudProj/tree/main>.
- Build, Lint, & Test Unitaires : Exécution des étapes de build, de linter et de test unitaire pour garantir la qualité du code.
- Construction Docker et Poussée vers ACR : Construction d'une image Docker à partir du code source, ajout d'une étiquette avec le numéro de build Jenkins, puis poussée vers le Registre de Conteneurs Azure (ACR).
- Déploiement Helm vers K8s : Utilisation de Helm pour déployer l'application sur un cluster Kubernetes, en configurant dynamiquement le tag de l'image Docker à déployer.

```

1 pipeline {
2   agent any
3   environment {
4     ACR_LOGINSERVER = credentials('ACR_LOGINSERVER')
5     ACR_ID = credentials('ACR_ID')
6     ACR_PASSWORD = credentials('ACR_PASSWORD')
7   }
8   stages {
9     stage('Récupération du Code Source') {
10      steps {
11        checkout([class: 'GitSCM', branches: [[name: '**/master']], doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [], userRemoteConfigs: [[url: 'https://gi
12      ]
13    }
14    stage('Build, Lint, & Test Unitaires') {
15      steps {
16        script {
17          echo "Exécutez ici le build, le linter et le test unitaire"
18        }
19      }
20    }
21    stage('Construction Docker et Poussée vers ACR') {
22      steps {
23        script {
24          def REPO_NAME = "azure-voting-app-redis"
25          def ACR_LOGINSERVER = "myrepo.azurecr.io"
26          def ACR_ID = "myACRid"
27          def ACR_PASSWORD = "myACRpassword"
28          def IMAGE_NAME = "${ACR_LOGINSERVER}/${REPO_NAME}:jenkins${BUILD_NUMBER}"
29          dir('azure-vote') {
30            sh "docker build -t ${IMAGE_NAME} ."
31          }
32          withCredentials([usernamePassword(credentialsId: 'ACR_CREDENTIALS_ID', usernameVariable: 'ACR_ID', passwordVariable: 'ACR_PASSWORD')]) {
33            sh "docker login ${ACR_LOGINSERVER} -u ${ACR_ID} -p ${ACR_PASSWORD}"
34            sh "docker push ${IMAGE_NAME}"
35          }
36        }
37      }
38    }
39    stage('Déploiement Helm vers K8s') {
40      steps {
41        script {
42          def REPO_NAME = "azure-voting-app-redis"
43          def ACR_LOGINSERVER = "myrepo.azurecr.io"
44          def NAME = "azure-voting-app-redis"
45          def HELM_CHART = "./helm/azure-voting-app-redis"
46          def KUBE_CONTEXT = "jenkins-k8s-azure"
47          sh "kubectl config --kubeconfig=/var/lib/jenkins/.kube/config view"
48          sh "kubectl config set-context ${KUBE_CONTEXT}"
49          sh "helm --kube-context ${KUBE_CONTEXT} upgrade --install --force ${NAME} ${HELM_CHART} --set image.repository=${ACR_LOGINSERVER}/${REPO_NAME} --set image.tag-jenkins${B
50        }
51      }
52    }
53  }
54  post {
55    always {
56      echo 'Étapes de Build Terminées'
57    }
58  }
59 }
60 }
61

```



**Déroulement du Pipeline :**

## 1. Étape "Récupération du Code Source" :

Le script Jenkins récupère le code source de l'application depuis le référentiel Git spécifié.

## 2. Étape "Build, Lint, &amp; Test Unitaires" :

Les étapes de build, de linter et de test unitaire sont effectuées pour assurer la robustesse et la qualité du code.

## 3. Étape "Construction Docker et Poussée vers ACR" :

L'application est construite en tant qu'image Docker, étiquetée avec le numéro de build Jenkins, puis poussée vers le Registre de Conteneurs Azure.

## 4. Étape "Déploiement Helm vers AKS" :

Utilisation de Helm pour déployer l'application sur un cluster Kubernetes, en ajustant dynamiquement le tag de l'image Docker.

**Conclusion :**

La mise en œuvre réussie de ce pipeline CI/CD automatisé démontre l'efficacité de l'intégration continue et du déploiement continu dans le contexte du développement d'applications conteneurisées. Cette approche permet d'assurer une livraison rapide, cohérente et fiable de l'application tout au long de son cycle de vie.

## 7. Automatisation du Déploiement d'une Application sur Azure Kubernetes Services (AKS) via Jenkins:

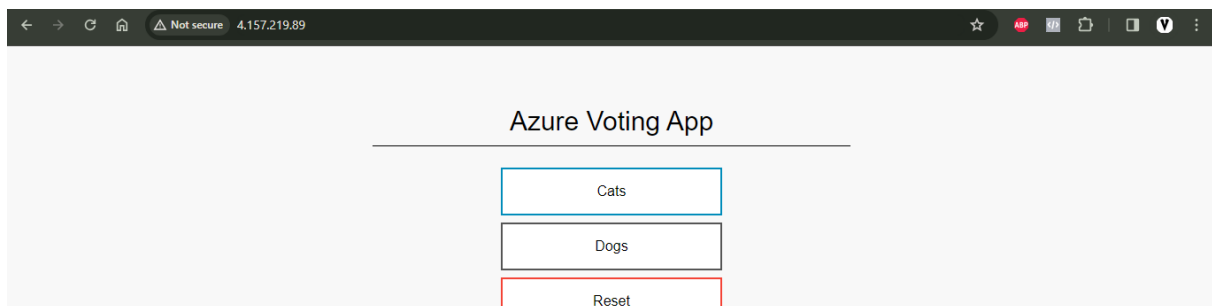
Le présent rapport rend compte d'une expérience fructueuse d'automatisation du déploiement d'une application sur Azure Kubernetes Services (AKS) à l'aide de Jenkins. L'objectif de cette automatisation était de simplifier le processus de déploiement, d'assurer la cohérence et d'améliorer l'efficacité du cycle de vie du développement.

### Déroulement de l'Automatisation :

Le lancement du build a été initié manuellement par un membre de l'équipe en cliquant sur "Build Now" dans l'interface Jenkins. Ensuite, Jenkins a automatiquement récupéré le code source de l'application depuis le référentiel Git spécifié. Les étapes suivantes, comprenant le build, le linter et les tests unitaires, ont été exécutées de manière automatisée pour garantir la qualité du code. Enfin, l'application a été construite en tant qu'image Docker, étiquetée avec un numéro de build unique, puis elle a été poussée avec succès vers le Registre de Conteneurs Azure (ACR)

### Déploiement Helm vers AKS :

Utilisation de Helm pour déployer l'application sur Azure Kubernetes Services (AKS). Le pipeline a pris en charge la configuration dynamique du tag de l'image Docker à déployer.



### Résultats et Avantages :

- Le processus d'automatisation a été accompli avec succès, démontrant la puissance de l'intégration continue et du déploiement continu (CI/CD) dans la livraison rapide et cohérente d'applications.
- L'équipe de développement a gagné en productivité grâce à l'élimination des tâches manuelles et à l'automatisation complète du déploiement.

**Conclusion :**

Cette expérience d'automatisation réussie illustre la valeur ajoutée de Jenkins dans le contexte du développement d'applications conteneurisées. Le simple clic sur "Build Now" a initié un processus automatisé, aboutissant au déploiement réussi de l'application sur Azure Kubernetes Services. Ce rapport souligne l'efficacité de l'automatisation pour accélérer le cycle de vie du développement et garantir des déploiements cohérents et fiables.

## Conclusion :

Ce projet a été une entreprise stimulante et enrichissante, guidée par une vision claire de moderniser et d'optimiser le processus de développement et de déploiement d'applications. Tout d'abord, le contexte et la problématique ont été clairement définis, mettant en évidence les défis à relever. Les objectifs ont été fixés avec précision, orientant le travail vers des résultats tangibles.

L'architecture globale conçue offre une vue d'ensemble structurée de l'environnement de déploiement, avec un accent particulier sur les interactions entre les différents composants. Cette approche architecturale a fourni un cadre solide pour la mise en œuvre pratique.

La mise en place du projet a impliqué plusieurs étapes cruciales. De la création d'une instance EC2 avec l'installation manuelle de Jenkins, Docker et Azure CLI, jusqu'à l'automatisation de la création d'une machine virtuelle Azure avec Terraform et l'installation de Jenkins avec Ansible, chaque étape a contribué à simplifier et à rationaliser le processus de déploiement.

Le déploiement manuel initial de l'application dans Azure Kubernetes Services (AKS) avec Docker Compose a servi de base pour la phase suivante d'automatisation. L'introduction progressive de méthodologies CI/CD avec Jenkins, Docker et Kubernetes/Helm a permis d'atteindre un niveau supérieur d'efficacité et de cohérence dans le déploiement des applications.

Enfin, l'automatisation du déploiement CI/CD de l'application avec Jenkins, Docker et Kubernetes/Helm a représenté un jalon majeur. Cette automatisation garantit une livraison rapide, fiable et cohérente de l'application tout au long de son cycle de vie.

En conclusion, ce projet a démontré avec succès la valeur de l'automatisation dans le domaine du développement et du déploiement d'applications, ouvrant la voie à des processus plus agiles, efficaces et robustes. Les apprentissages tirés de ce projet serviront de fondement pour des initiatives futures visant à optimiser davantage les pratiques de développement et de déploiement.