# TP2 devops

## Théophile Avenel

# 1 Intégration continue fournie par Gitlab

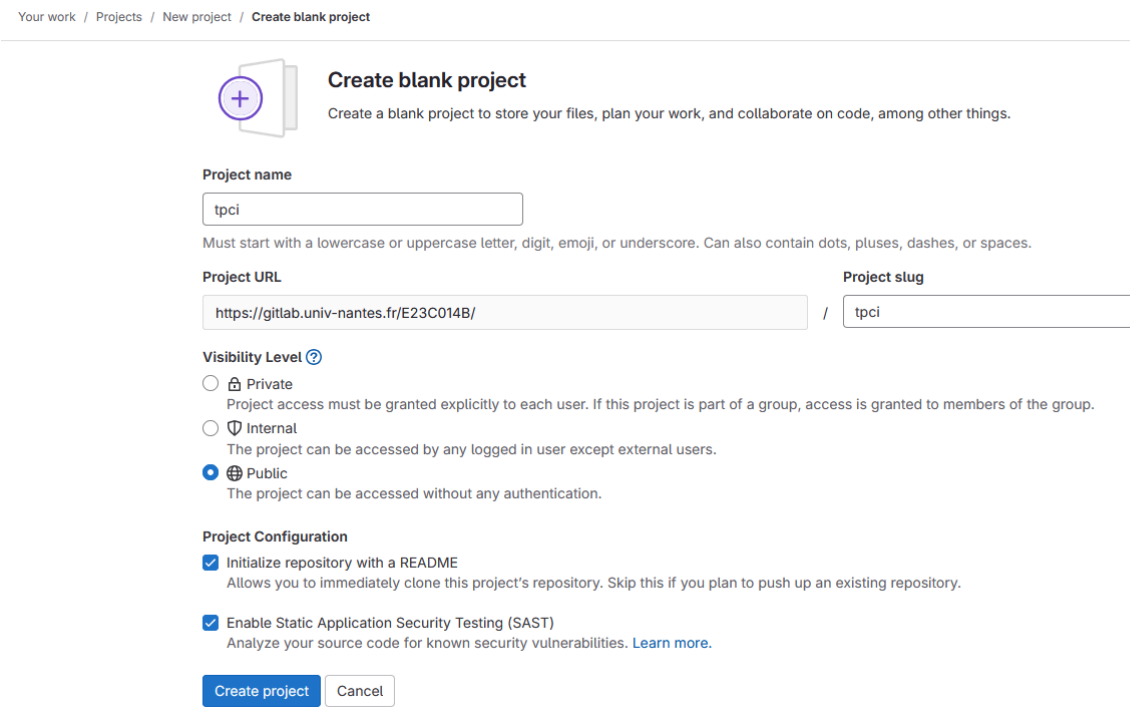


Figure 1: Création d'un blank project sur Gitlab

Nous commençons par créer un nouveau dépôt dans GitLab puis nous commençons par gérer la configuration pour permettre à l'autre membre du binôme du TP de collaborer.

Pour effectuer cette étape, il faut se rendre dans l'onglet manage puis cliquer sur members et ajouter un user

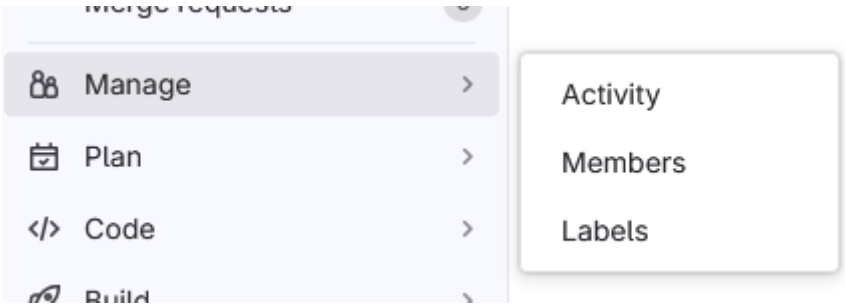


Figure 2: Clic sur l'onglet Manage de Gitlab pour manager les "membres" du repository

Il faut ensuite cliquer sur le bouton invite members

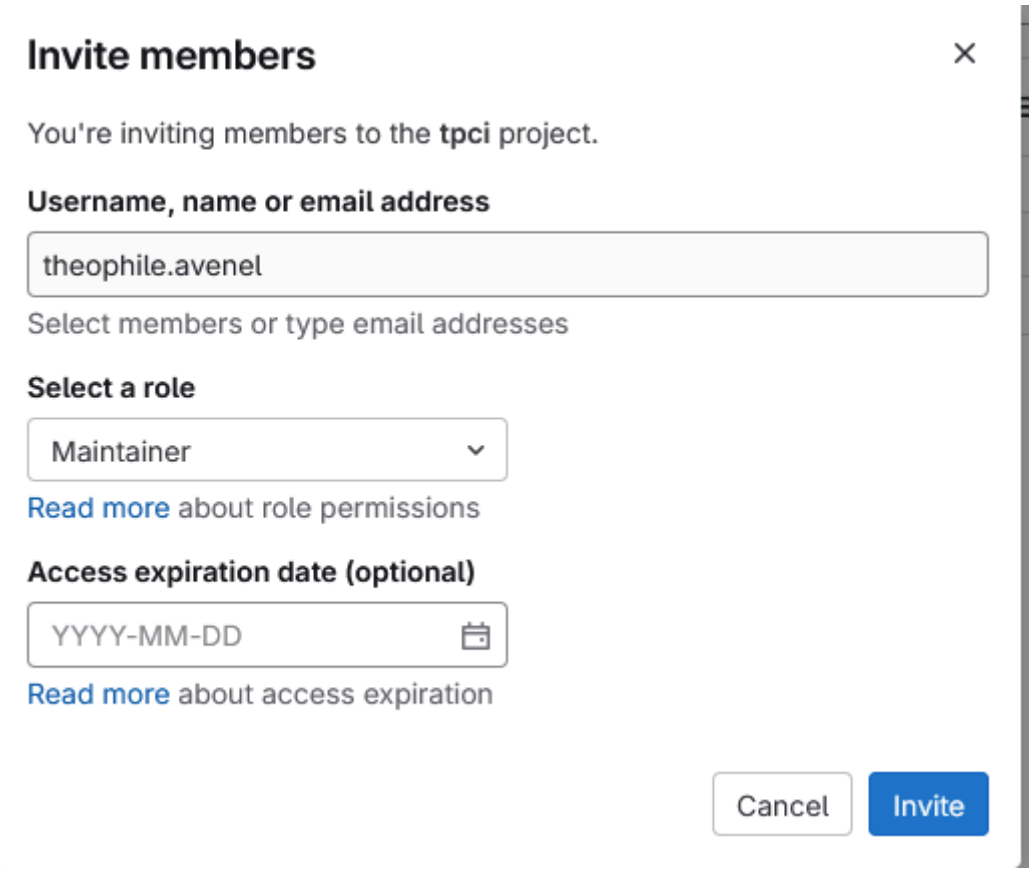


Figure 3: Fenêtre d'invitation des membres au repository gitlab

L'utilisateur apparait ensuite comme invité via le pending invitations.

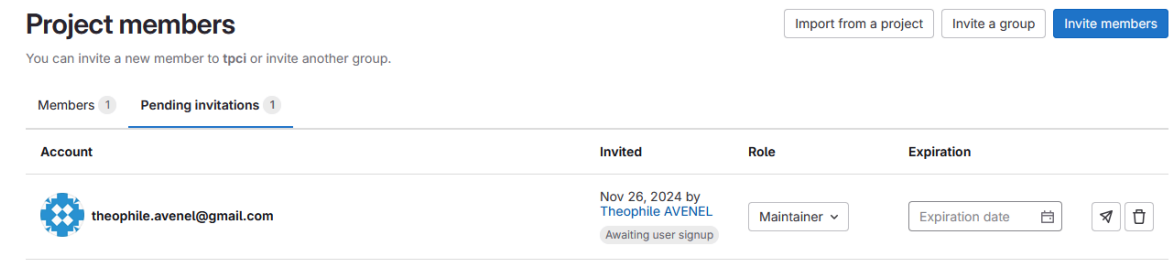


Figure 4: Onglet des invitations en attente

Pour faciliter la collaboration, j'ai créé une branche develop ce qui permet de garder la branche main comme branche de production avec livraison du code final.

Deux autres branches ont été créé pour que chaque members puissent avoir une branche de feature (feature-th1 et feature th-2).

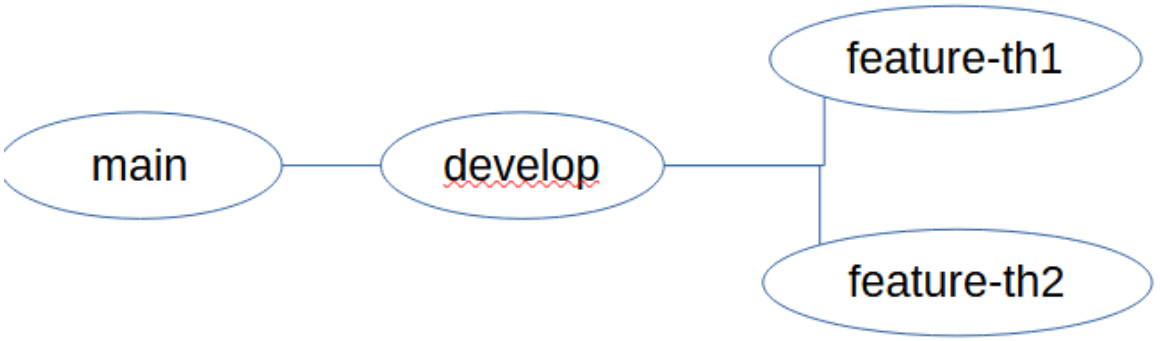


Figure 5: Répartition des branches sur le repository Gitlab

Nous allons commencer par rédiger un code main.c dans la branche feature-th1. Pendant ce temps l'autre membre ajoute main.cpp dans le code feature-th2

Chaque développeur a pu coder sur sa branche son fichier main lors d'une peer review ils ont décidé de merge leur travail sur la branche develop.

Voici un exemple de merge request d'une branche feature sur develop.

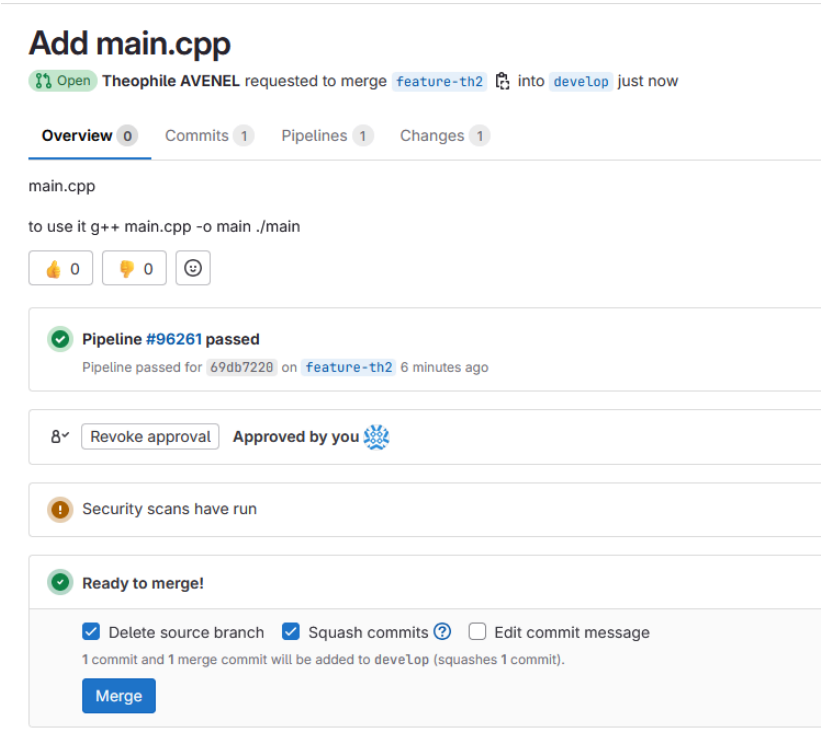


Figure 6: Merge request de la branche feature-th2 vers develop

Voici ci dessous un exemple de résultat de merge request

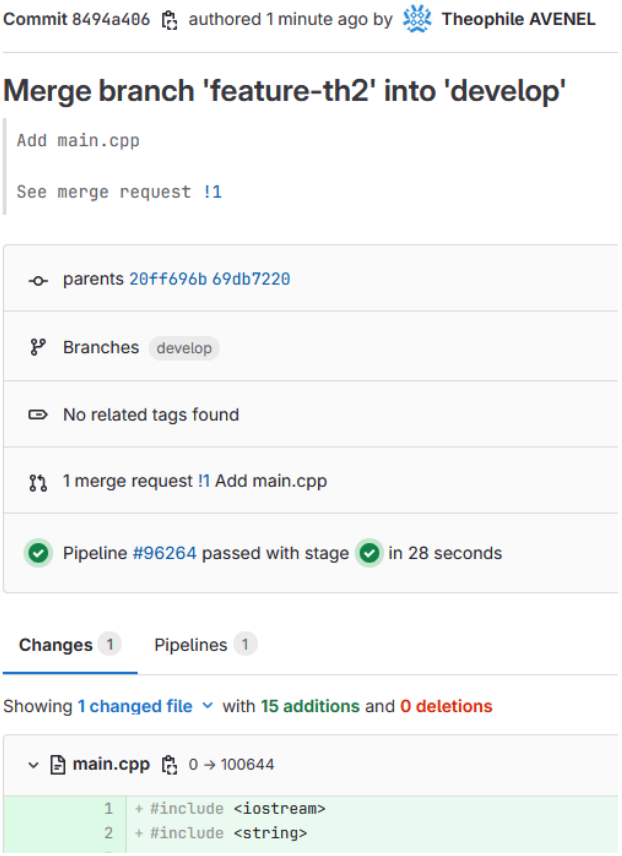


Figure 7: La merge de la branche feature-th2 vers develop a focntionné avec succès

En se rendant dans l'onglet merge requests nous pouvons cliquer sur l'onglet merged pour voir les merge requests qui ont été effectuées

https://gitlab.univ-nantes.fr/E23C014B/tpci/-/merge_requests?scope=all&state=merged

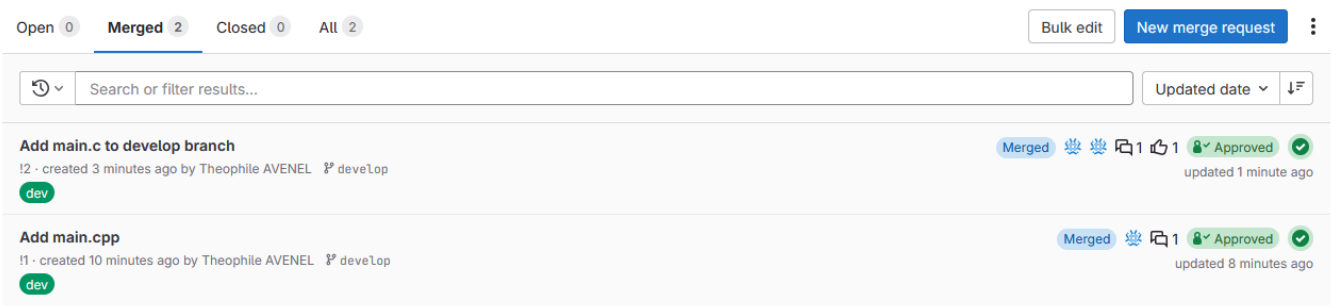


Figure 8: Onglet des merge requests

Voici ci dessous un exemple d'exécution des différents main

```
(base) ┌──(athena㉿DESKTOP-D0DH9M9)-[~/Document/gitwithctp]
└─$ gcc main.c -o main
./main
Veuillez saisir votre nom : user1
Bonjour, user1

(base) ┌──(athena㉿DESKTOP-D0DH9M9)-[~/Document/gitwithctp]
└─$ g++ main.cpp -o main
./main
Veuillez saisir votre nom : test
Bonjour, test!
```

Figure 9: Exemple d'exécution des différents main

Pour éviter les problèmes de "ça marche sur ma machine mais pas moi" l'équide de dev a décidé de faire du pair programming sur la partie de devops.


Figure 10: Figure représentant le pair programming

L'équipe de dev a donc décidé de créer un pipeline de C et un pipeline de C++ pour tester et exécuter le code main.c et le code main.cpp. Ce pipeline CI/CD permettra d'automatiser le dépoiement sur les machines de productions.

Voici ci dessous le résultat de ce premier pipeline.

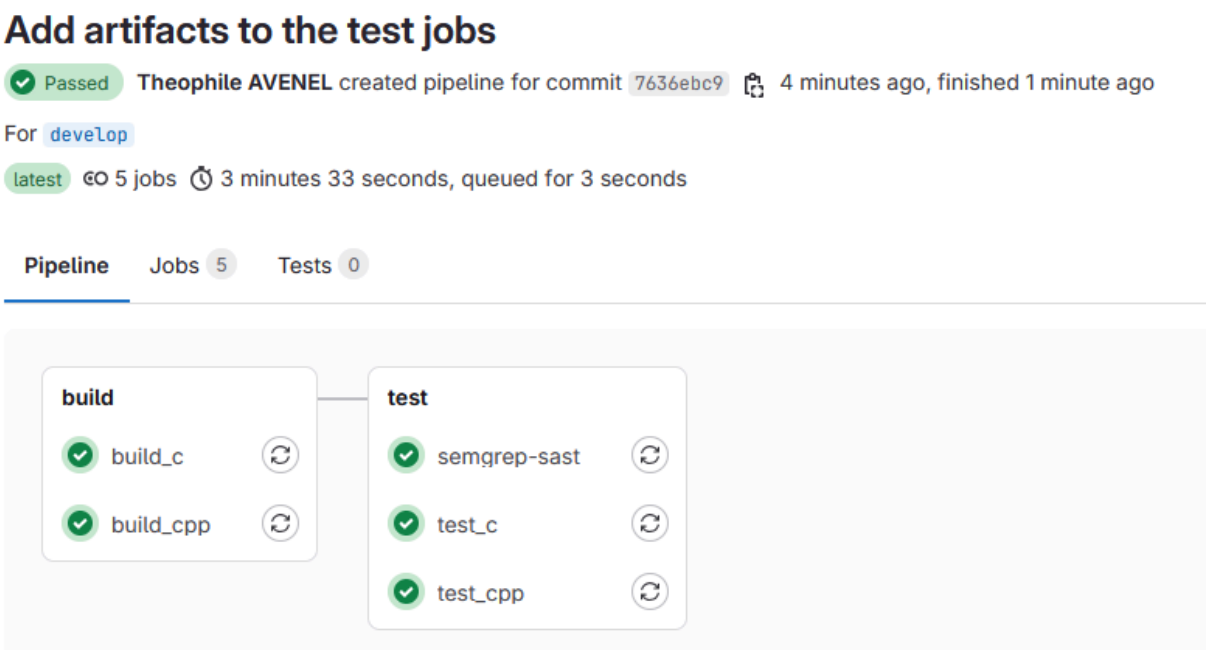


Figure 11: Résultat du premier pipeline sur Gitlab

Le pipeline de build et de test a fonctionné voici le lien ci-dessous

https://gitlab.univ-nantes.fr/E23C014B/tpci/-/pipelines/96290

Une fois que le workflow CI/CD de Gitlab a bien été exécuté, nous pouvons télécharger l'artefact pour essayer de l'exécuter sur notre machine.

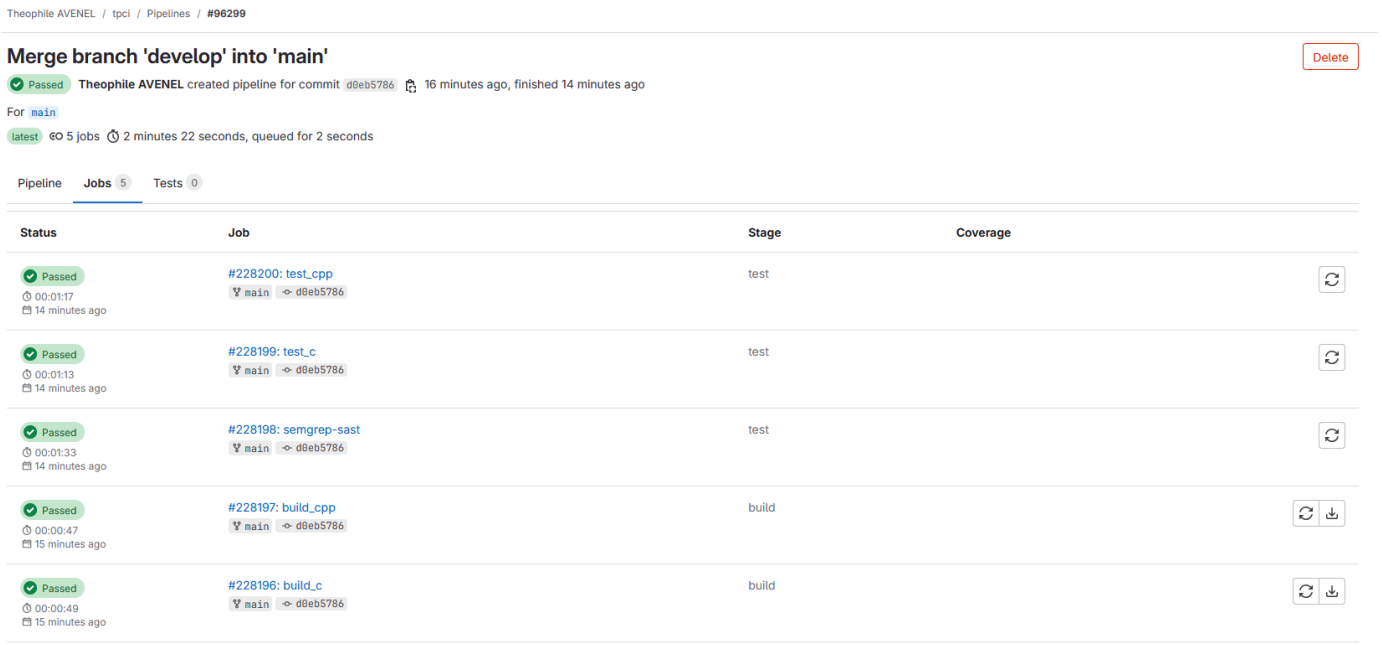


Figure 12: Liste des jobs

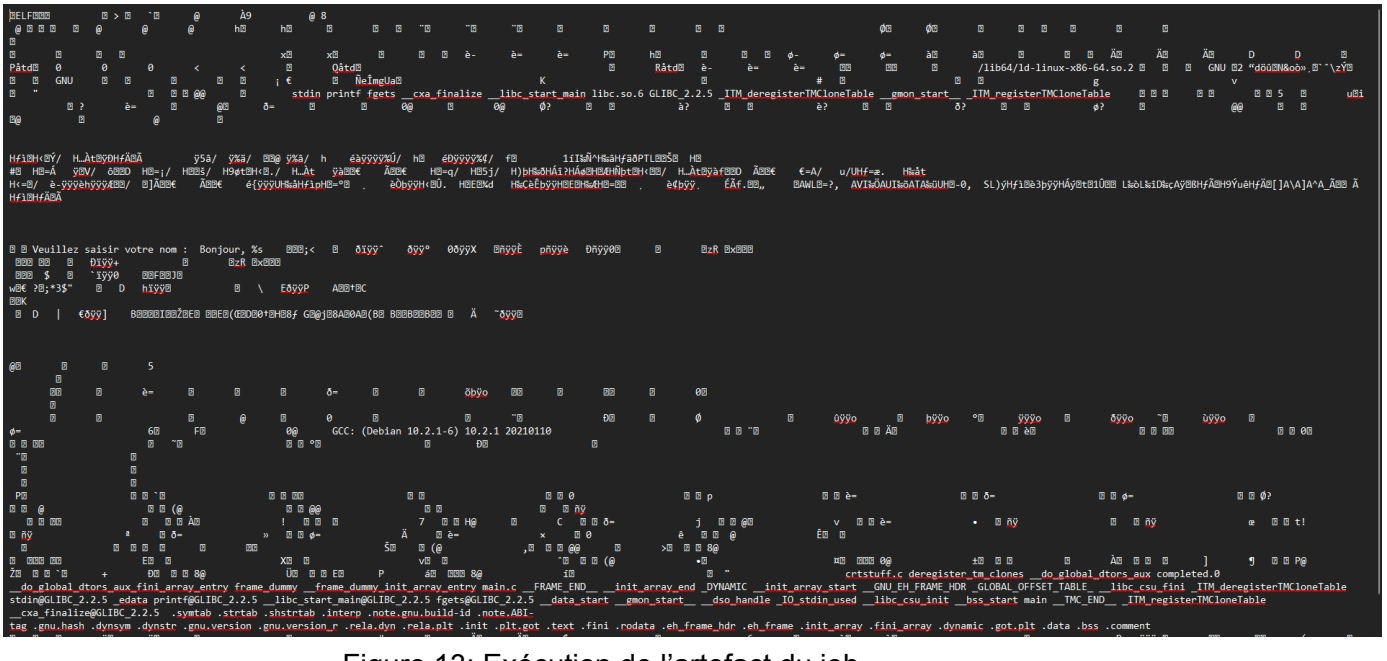


Figure 13: Exécution de l'artefact du job

Comme le montre la figure 13, l'artefact généré par git pour le job est un fichier binaire

Pour vérifier que le pipeline réagit coorectement, j'ai inclu une erreur dans le code main.c pour vérifier que le code ne se compile pas et examiner le message d'erreur du build.

https://gitlab.univ-nantes.fr/E23C014B/tpci/-/jobs/228245

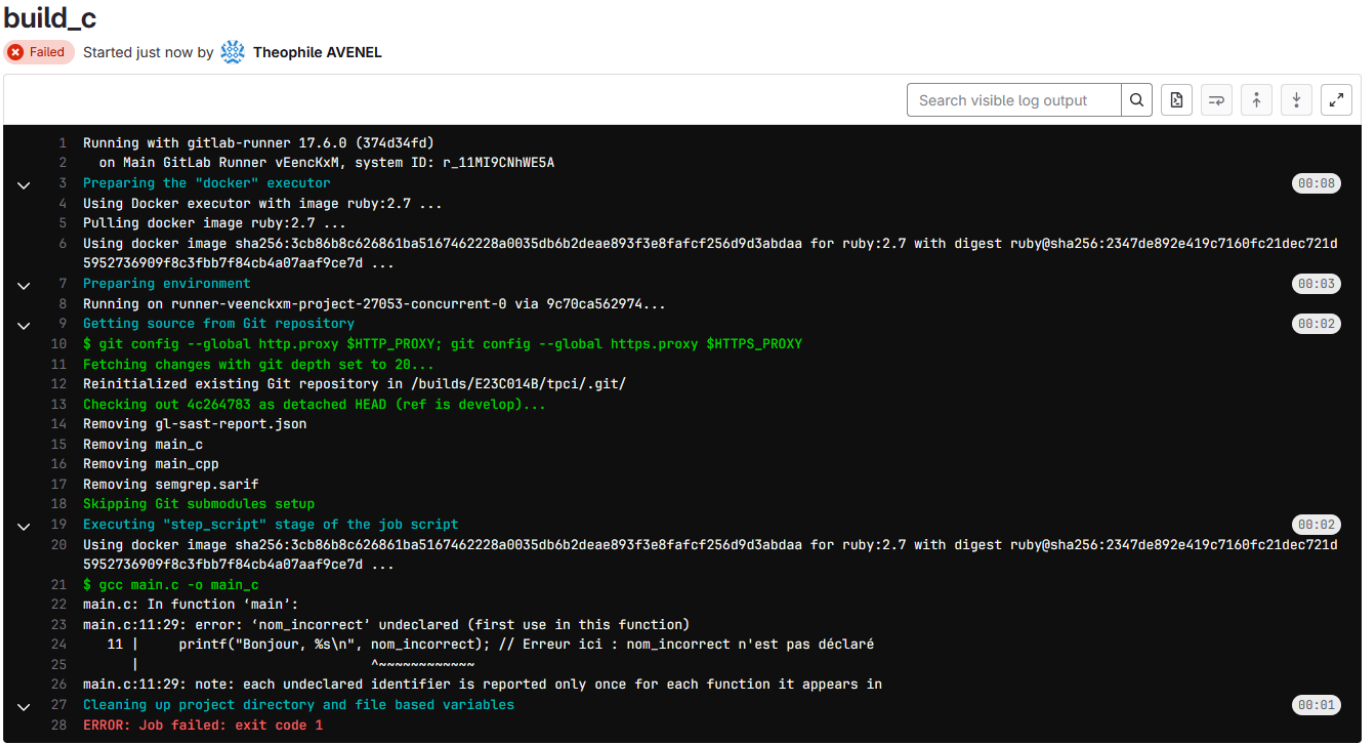


Figure 14: Le pipeline échoue quand on onclu une erreur

L'erreur : ERROR/ Job failed: exit code 1 montre que le pipeline réagit correctement en cas d'erreur.

Nous avons ensuite pu rajouter les fichiers calcul.c et calcul.cpp tout en modifiant main.c et main.cpp puis nous avons pu modifier le gitlab workflow en effectuant une emrge request final sur la branche main avec un nouveau README.md .

Le repository gitlab est en mode d'accès Public est est accessible depuis l'url :

https://gitlab.univ-nantes.fr/E23C014B/tpci

Vous pouvez donc trouver le code des fichiers sur le repository Gitlab.

# 2 Installation du serveur Jenkins

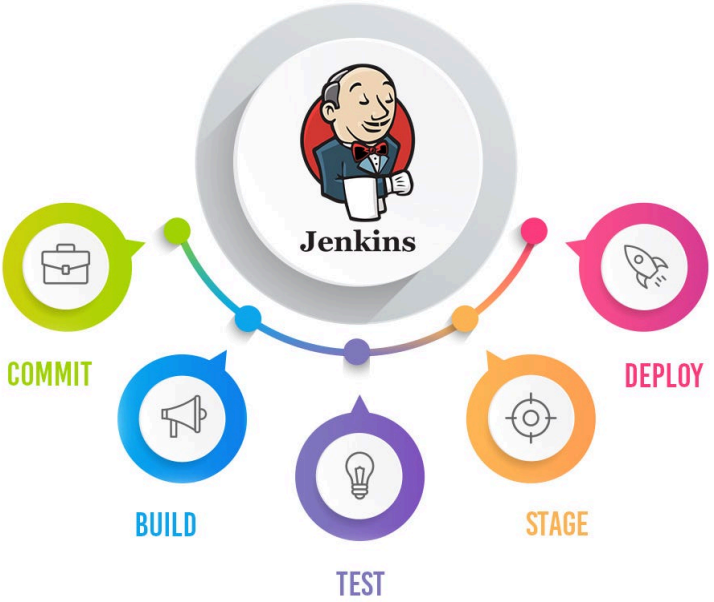


Figure 15: Illustration de jenkins

L'installation dans Kubernetes suppose que nos outils font partie d'un espace de nom dédié pour le développement qui est différent des autres services qui pourront être utilisé dans d'autres aspects. Pour commencer nous pouvons utiliser la commande minilube start pour starter mlinikube.

```
athena@ubuntumate2204:~$ minikube start
😄  minikube v1.34.0 sur Ubuntu 22.04 (vbox/amd64)
```

Figure 16: Lancement de minikube

**Remarque**: des erreurs peuvent apparaitre si on ne fait pas de minikube start

```
athena@ubuntumate2204:~$ kubectl get serviceaccounts -n devops-tools
Unable to connect to the server: dial tcp 192.168.49.2:8443: connect: no route t
o host
athena@ubuntumate2204:~$ nano service-account.yaml
athena@ubuntumate2204:~$ minikube status
minikube
type: Control Plane
host: Stopped
kubelet: Stopped
apiserver: Stopped
kubeconfig: Stopped
```

Figure 17: Des erreurs se produisent si on n'effectue pas la commande minikube start

Nous commençons par créer un espace de nom nommé devops-tools.

L'espace de nom (namespace) permet d'isoler les ressources Kubernetes. Pour créer le namespace `devops-tools`, j'utilise la commande kubectl create namespace devops-tools. Pour lister les namespaces j'utilise la commande kubectl get namespaces

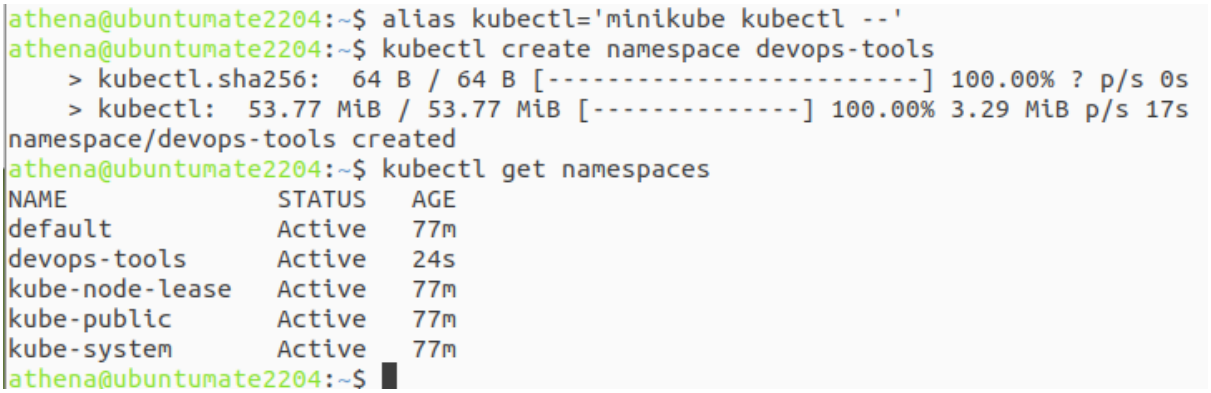


Figure 18: Création d'un alias pour appeler kubectl avec minikube

Nous allons ensuite créer un service administrateur à partir du fichier service-account.yaml.

Pour réaliser cette étape, j'ai pu configurer le fichier `service-account.yaml` en définissant un `ServiceAccount` nommé `admin-service-account` dans le namespace `devops-tools`. J'ai ensuite pu "appliquer" ce fichier avec la commande `kubectl apply -f service-account.yaml -n devops-tools`. Pour vérifier sa création j'ai pu utiliser `kubectl get serviceaccounts -n devops-tools`, où `admin-service-account` devrait apparaître dans la liste.

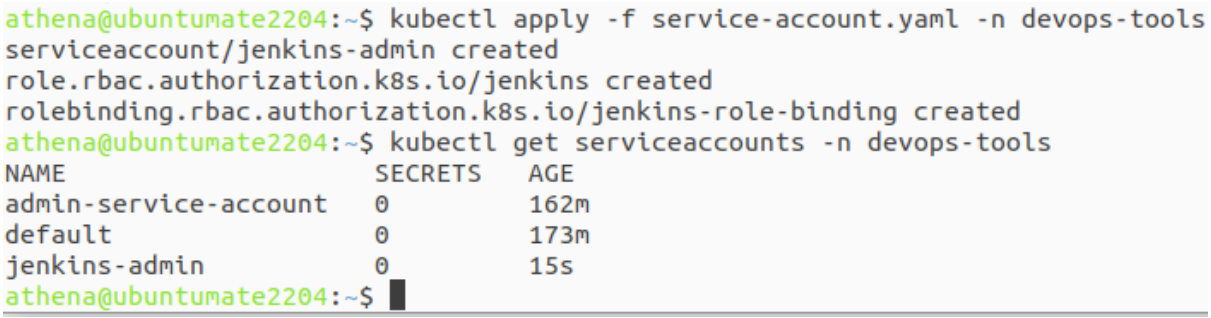


Figure 19: Application du service-account.yaml au kubectl

J'ai ensuite pu créer "un secret" pour accéder au contrôleur en appliquant le fichier secret.yaml

Pour réaliser cette étape, j'ai pu configurer le fichier `secret.yaml` en y incluantl'username et le mot de passe encodées en base64, comme le nom d'utilisateur et le mot de passe. Avec la commande `kubectl apply -f secret.yaml -n devops-tools` j'ai pu "appliquer" le secret au namespace `devops-tools` . Pour vérifier sa création lj'ai pu exécuter la commande `kubectl get secrets -n devops-tools`, où `jenkins-secret` devrait être listé.

```
athena@ubuntumate2204:~$ kubectl apply -f secret.yaml -n devops-tools
secret/sa-token-secret created
athena@ubuntumate2204:~$ kubectl get secrets -n devops-tools
NAME              TYPE                                  DATA   AGE
jenkins-secret    Opaque                                2      159m
sa-token-secret   kubernetes.io/service-account-token   3      14s
athena@ubuntumate2204:~$
```

Figure 20: Application du secret.yaml au kubectl

Une fois que les étapes précédentes ont été réalisé, j'ai pu créer un déploiement à partir du fichier deployment.yaml

J'ai donc pu configurer le fichier `deployment.yaml` pour déployer Jenkins dans le namespace `devops-tools`, en définissant un seul réplica, l'image officielle de Jenkins, les ports nécessaires et le montage d'un volume pour les données persistantes. J'ai pu appliquer ce changement avec la commande `kubectl apply -f deployment.yaml -n devops-tools`. Pour vérifier le déploiement et l'état des pods j'ai pu écécuter `kubectl get deployments -n devops-tools` et `kubectl get pods -n devops-tools`, en vous assurant que le pod Jenkins est en état `Running`.

```
athena@ubuntumate2204:~$ kubectl apply -f deployment.yaml -n devops-tools
persistentvolumeclaim/jenkins-pv-claim created
deployment.apps/jenkins-deployment configured
service/jenkins-service configured
athena@ubuntumate2204:~$ kubectl get deployments -n devops-tools
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
jenkins-deployment   1/1     1            1           157m
athena@ubuntumate2204:~$ kubectl get pods -n devops-tools
NAME                                 READY   STATUS    RESTARTS      AGE
jenkins-deployment-659f779669-62krq   1/1     Running   1 (91m ago)   158m
jenkins-deployment-769696f4b4-hh8s9   0/1     Running   0             30s
athena@ubuntumate2204:~$
```

Figure 21: Application du deployment.yaml au kubectl

Nous pouvons désormais passer à l'étape d'identification de l'adresse du service Jenkins et l'étape d'accès au dashboard.
Pour cela j'ai créé le fichier service.yaml

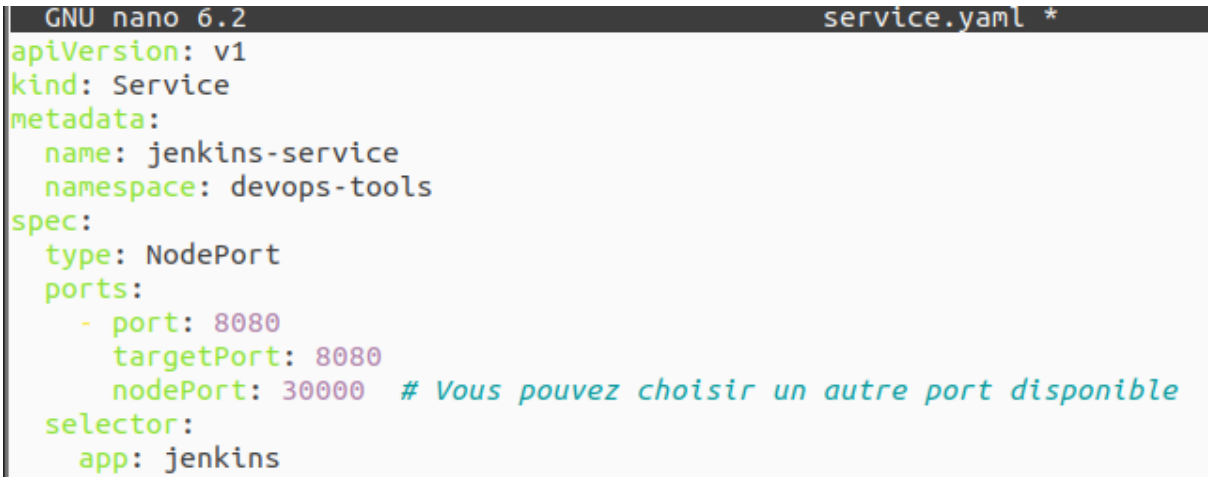


Figure 22: Le fichier service.yaml

Ce fichier `service.yaml` permet de définir un Service Kubernetes nommé `jenkins-service`, configuré pour exposer une application Jenkins. Le champ `apiVersion: v1` indique qu'il utilise la première version de l'API Kubernetes pour cet objet. Le `kind: Service` précise qu'il s'agit d'un service, utilisé pour rendre les pods accessibles. Sous `metadata`, le nom du service est spécifié comme `jenkins-service`, et il est associé au namespace `devops-tools`. Dans la section `spec`, le type de service est défini comme `NodePort`, permettant l'accès à Jenkins depuis l'extérieur du cluster via un port spécifique. La configuration des ports indique que le port interne du service (8080) est mappé au `nodePort: 30000`, qui est un port accessible sur les nœuds du cluster. Enfin, le `selector: app: jenkins` relie ce service aux pods ayant le label `app: jenkins`, garantissant que le trafic est acheminé vers les pods Jenkins correspondants.

J'ai ensuite pu appliquer le contenu du service.yaml au kubectl via la commande kubectl apply -f service.yaml -n devops-tools.

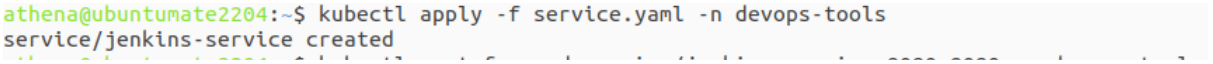


Figure 23: Application du service.yaml au kubectl

Une fois que j'ai pu appliquer toutes les configurations précédentes, j'ai pu me connecter au Jenkins via la commande minikube service jenkins-service -n devops-tools –url

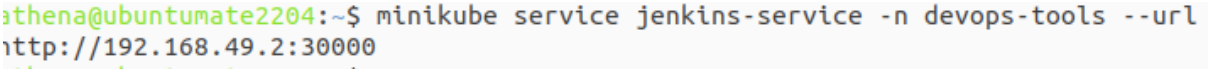


Figure 24: Lancement de Jenkins et de l'attribution de l'url

Quand je me rends sur l'url http://192.168.49.2:30000 j'arrive sur une page m'invitant à "Débloquer Jenkins"


Figure 25: Fenêtre pour débloquer Jenkins

J'ai pu chercher le  mot de passe en cherchant le pod associé à devops-tools avec la commande kubectl get pods -n devops-tools. Une fois le nom obtenu j'ai pu exécuter la commande kubectl logs <nom_pod> -n devops-tools et j'ai pu trouver le mot de passe.

```
athena@ubuntumate2204:~$ kubectl get pods -n devops-tools
NAME                                 READY   STATUS    RESTARTS   AGE
jenkins-deployment-659f779669-62krq   1/1     Running   0          36m
athena@ubuntumate2204:~$ kubectl logs jenkins-deployment-659f779669-62krq -n dev
ops-tools
```

Figure 26: Commande pour obtenir le mot de passe administrateur de Jenkins

```
*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password
 generated.
Please use the following password to proceed to installation:

b88290157c45449ca6f8cf49b2230ba9

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************
```

Figure 27: Obtention du mot de passe administrateur de Jenkins

J'ai donc pu rentrer le mot de passe "b88290157c45449ca6f8cf49b2230ba9" pour le mot de passe administrateur et je peus désormais passer à l'étape de configuration.

# 3 Configurer

J’ai pu passer à la partie configuration du Jenkins et j’ai pu choisir l’installation des plugins suggérés automatiquement.

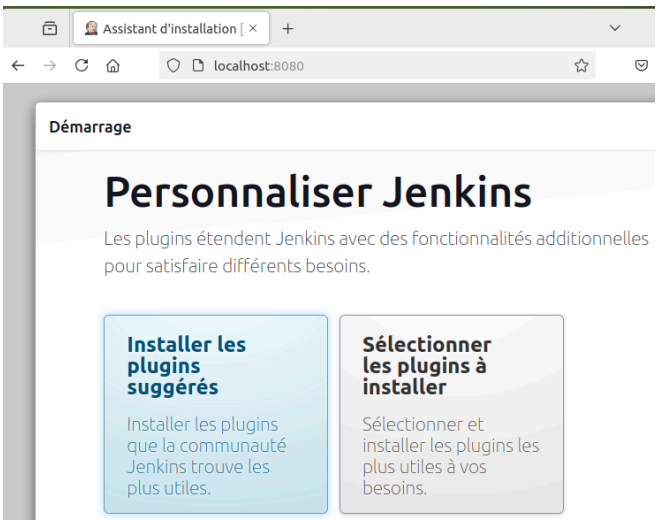


Figure 28: Sélection de l’onglet Installer les plugins suggérés

Cependant de nombreuses erreurs sont apparus en raison de problème de connexion internet

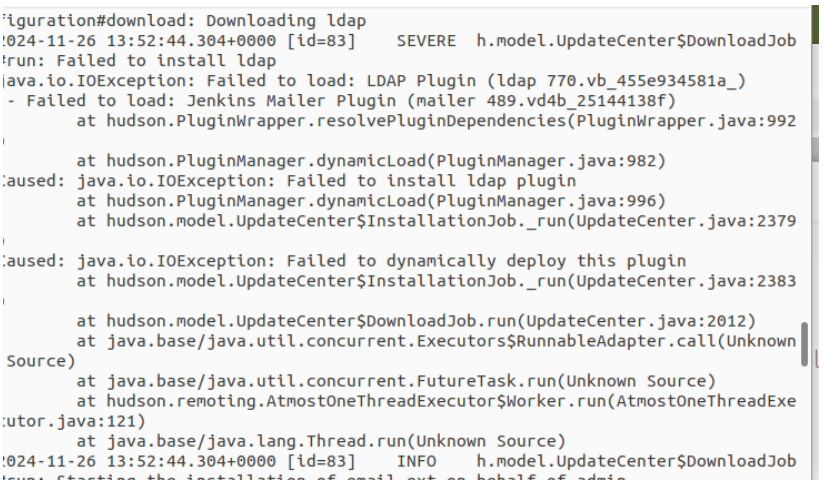


Figure 29: Erreur de connexion internet dans les logs

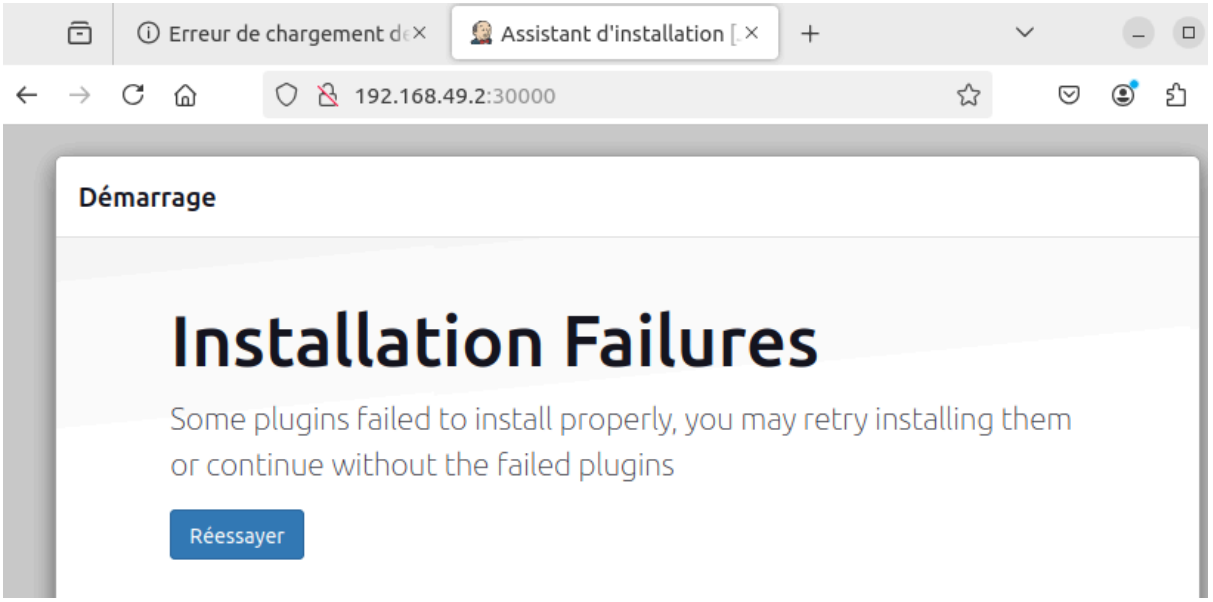


Figure 30: Erreur de connexion internet lors de l'installation des plugins

Après de nombreux clics sur le bouton Réessayer les plugins se sont installés et j'ai pu passer à la création du premier utilisateur Administrateur

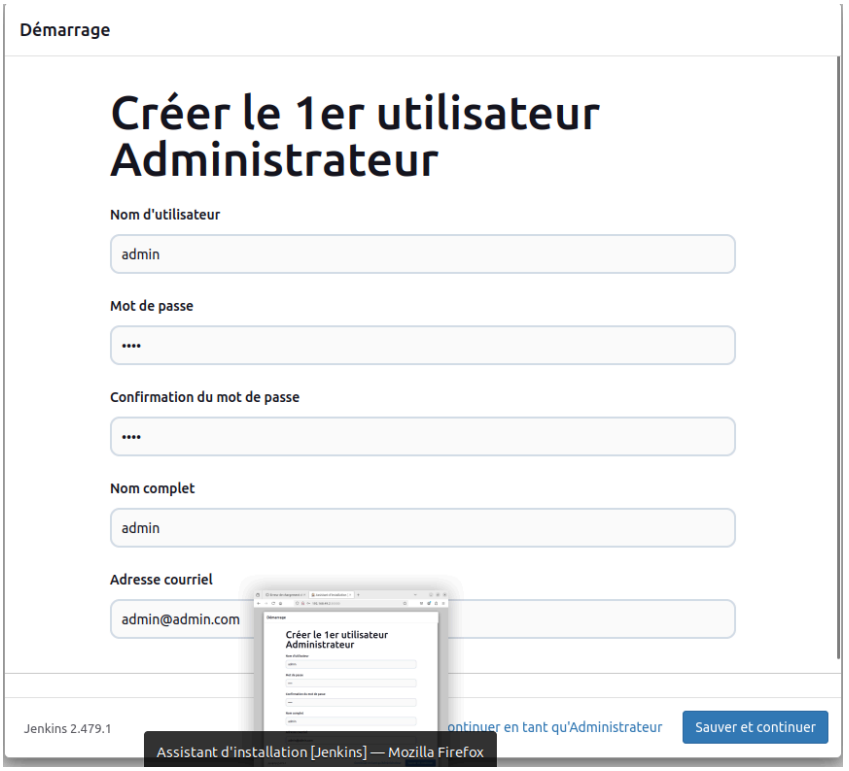


Figure 31: Création du premier utilisateur Administrateur

Une fois cette étape de création de compte administrateur réalisé je suis arrivé sur le tableau de bord de Jenkins.

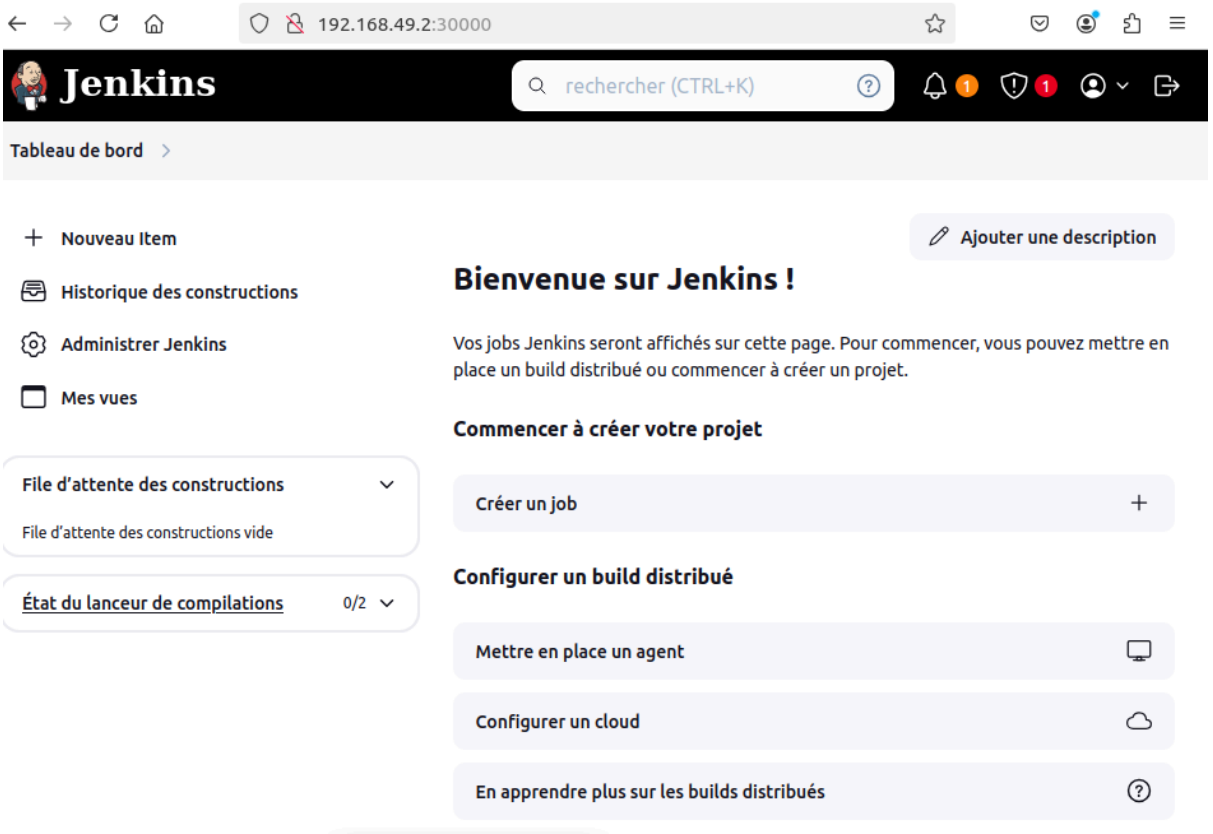


Figure 32: Page d'accueil de Jenkins

# 3.1 Cloud

J'ai ensuite pu installer le plugin cloud pour gérer les builds dans un cluster Kubernetes.

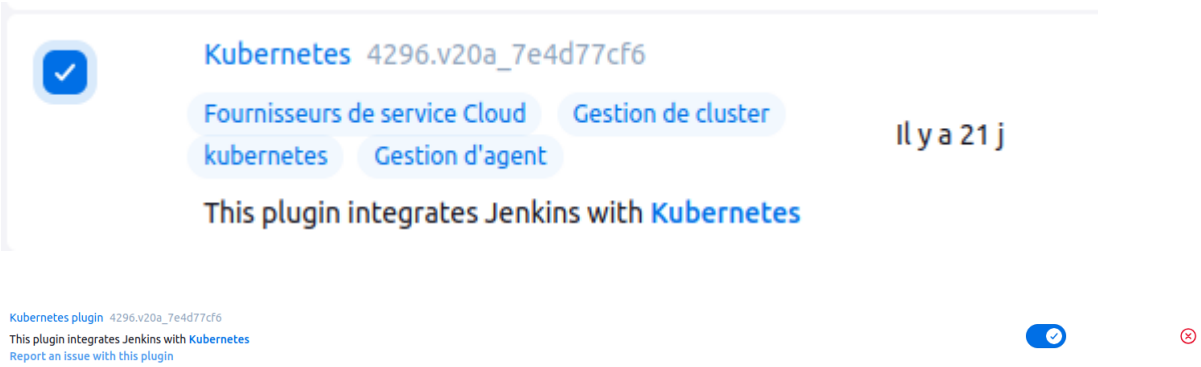


Figure 33: Installation du plugin cloud Kubernetes

J'ai ensuite pu configurer le cloud en ajoutant l'adresse IP du nœuds Jenkins dans le champ jenkins URL. J'ai pu appeler ce nouveau cloud could-kubernetes et j'ai pu lui donner le type Kubernetes.

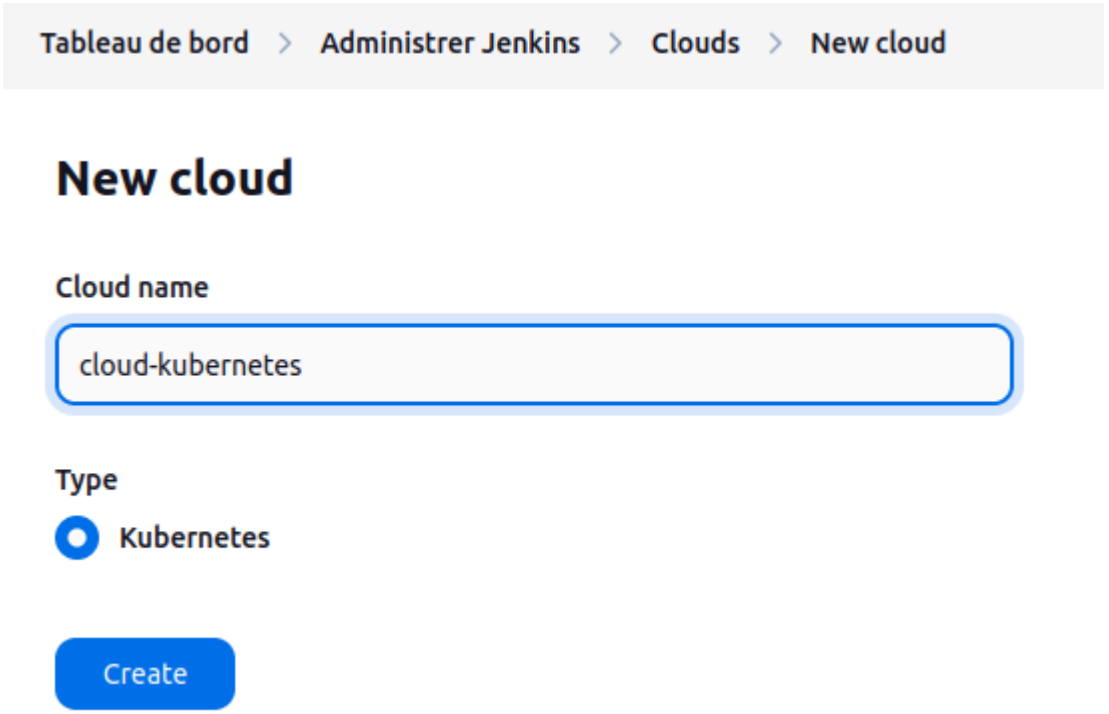


Figure 34: Création du cloud “cloud-kubernetes” de type Kubernetes

Une fois que le cloud a été créé, j’ai pu m’attaquer à la partie configuration de ce cloud.

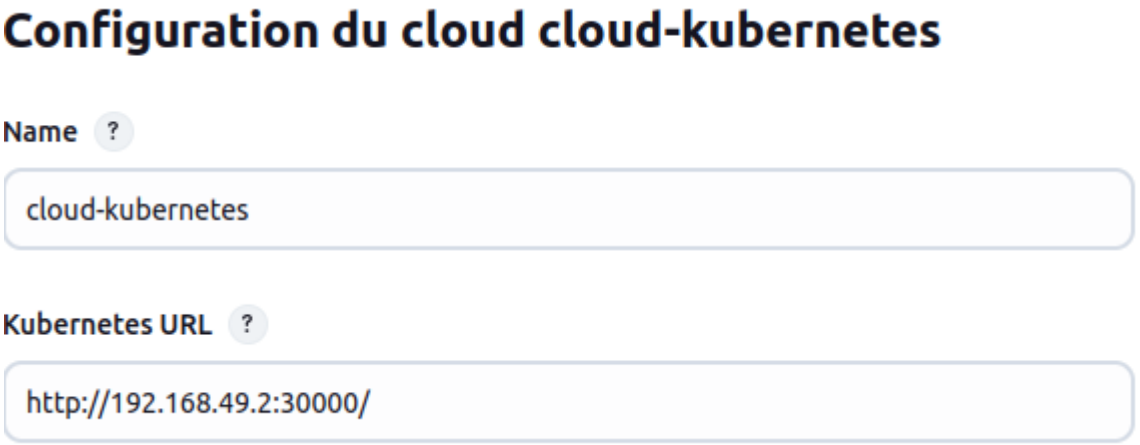


Figure 35: Configuration du cloud “cloud-kubernetes” de type Kubernetes

Pour ajouter le certificat j’ai exécuter la commande minikube ssh “sudo cat /var/lib/minikube/certs/ca.crt” > ca.crt et afficher son contenu

```
athena@ubuntumate2204:~$ minikube ssh "sudo cat /var/lib/minikube/certs/ca.crt"
> ca.crt
athena@ubuntumate2204:~$ more ca.crt
-----BEGIN CERTIFICATE-----
MIIDBjCCAe6gAwIBAgIBATANBgkqhkiG9w0BAQsFADAVMRMwEQYDVQQDEwptaW5p
a3ViZUNBMB4XDTI0MTEyNTEwNTYwMVoXDTM0MTEyNDEwNTYwMVowFTETMBEGA1UE
AxMKbWluaWt1YmVDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBANpy
5CZR14gKyji9NPskVYXJKuRXDwnLE10frSGDxTxNNWPopL8Xo6SauT0zVn3dADry
z0yRPK67cYoig+bI/xT/ZbjlCDHwKFgmTxBp/SMpHWnlK9rDiOXmaZEz6KqTnPwk
Oi0T0PYvDiR7jB0QemexkYNKOmmYfa8geO56LeUkkRG9BXL3GeU87n5LkcQablgF
ciFqxbtg+LzSwsvCiFmKAqlbzmiKNDDGomxn7lWfCIZ4R+cbv3Izqm4FMt27nyuU
bW6tktRs4k4v79QSaQzJ4EaeicCgz1A1CvjryUTtXOa9RNq5bGNKHOHCTWPg+kBf
O6bANTgz2LZhBIaVgNECAwEAAaNhMF8wDgYDVR0PAQH/BAQDAgKkMB0GA1UdJQQW
MBQGCCsGAQUFBwMCBggrBgEFBQcDATAPBgNVHRMBAf8EBTADAQH/MB0GA1UdDgQW
BBS6pFjwezCzx/WL08psOXOp85DynTANBgkqhkiG9w0BAQsFAAOCAQEAR35owaMu
gzXIJ3JrtLaCjuosjo6Ii+4TkWDUxmb/brmSE6bTHTdR0+PUs0ADprEgNHSXcxmj
NXMjlAYA8h8NhDn0BoTKwWOUInpq+renpJSbCYWj/64ugKi2hNl5ikCOMb5kIMbm
eQGJ5M88HBkX9RvHKzBh7DfF4lCfBkONiRNgaiUzTzVpLQ25whjY3r1T6Lg5fjqV
zUek9jwJtkE4ONBVQYD7DcclqK+9bxteeTExQkE9Mq+0S1vR05oXrsldeD146HSU
7wwNuOwf91F3+/hpEJlMrJ64nSXXtTfWV3nKWx2ohMZbxVS5vwKO4SqaK4skRMDm
0VEbchO7F36kEQ==
-----END CERTIFICATE-----
```

Figure 36: Certificat attaché à minikube

J’ai pu ensuitel cliquer sur le bouton Test Conenction pour vérifier que la connexion fonctionnait correctement mais j’ai eu uen erreur..

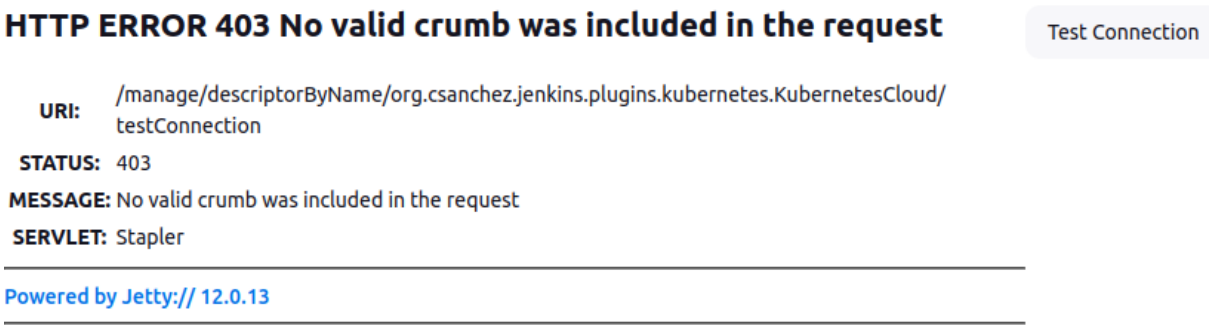


Figure 37: Erreur 403

J’ai donc pu effectuer un refresh pour corriger l’erreur

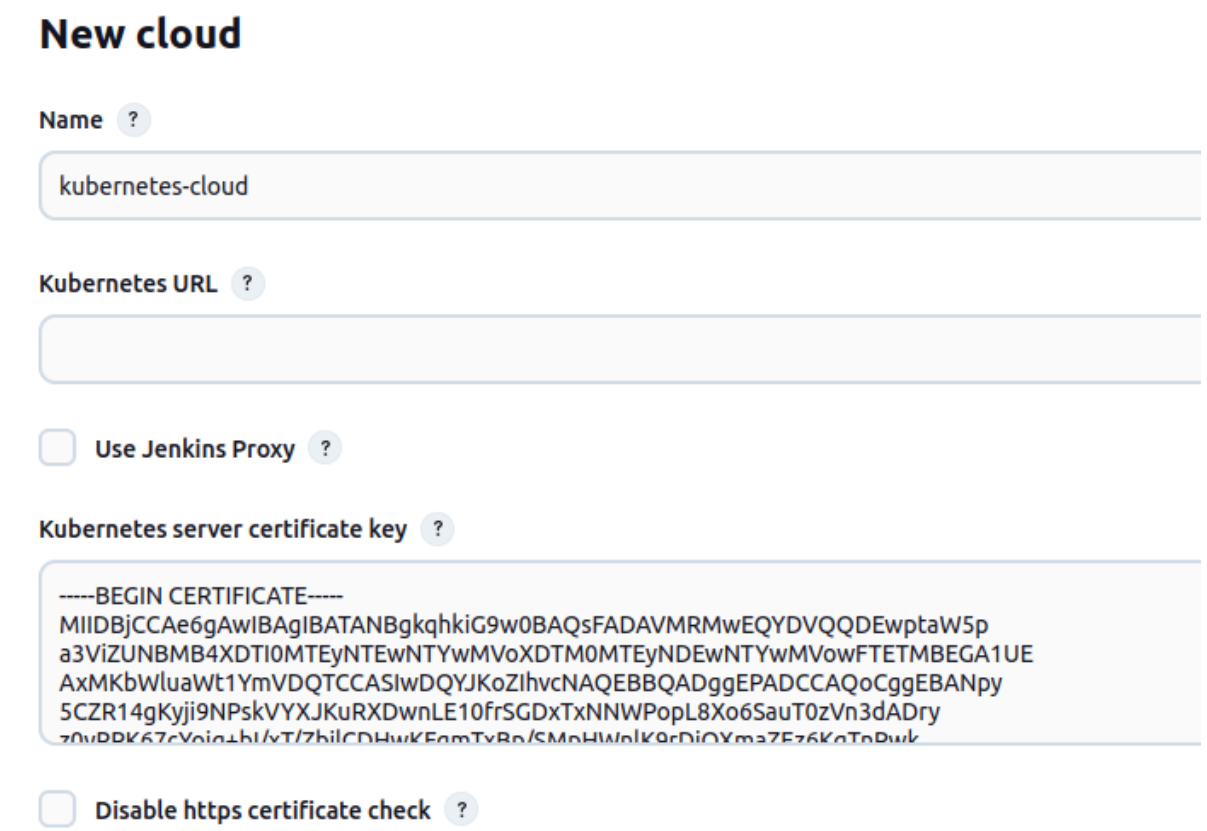


Figure 38: Ajout du certificat de minikube au cloud “kubernetes-cloud’

Le test de connexion c’est révélé concluant.


Figure 39: Le Test Connection a fonctionné

Comme le montre la figure 39 ci-dessus, la connexion fonctionné car nous obtenons le résutlat “Connected to Kubernetes v1.31.0”

## 3.2 Agents de builds

Les tâches des builds sont des tâches lourdes d’habitude, il est donc préférable de consacrer des pods dédiés pour les exécuter.

Le noeud correspondant au contrôleur ne doit pas épuiser ces ressources pour effectuer les tâches libres comme les builds. Dans la section correspondante à la gestion de Jenkins, modifier les paramètres pour que le contrôleur n’exécute pas les tâches des builds.

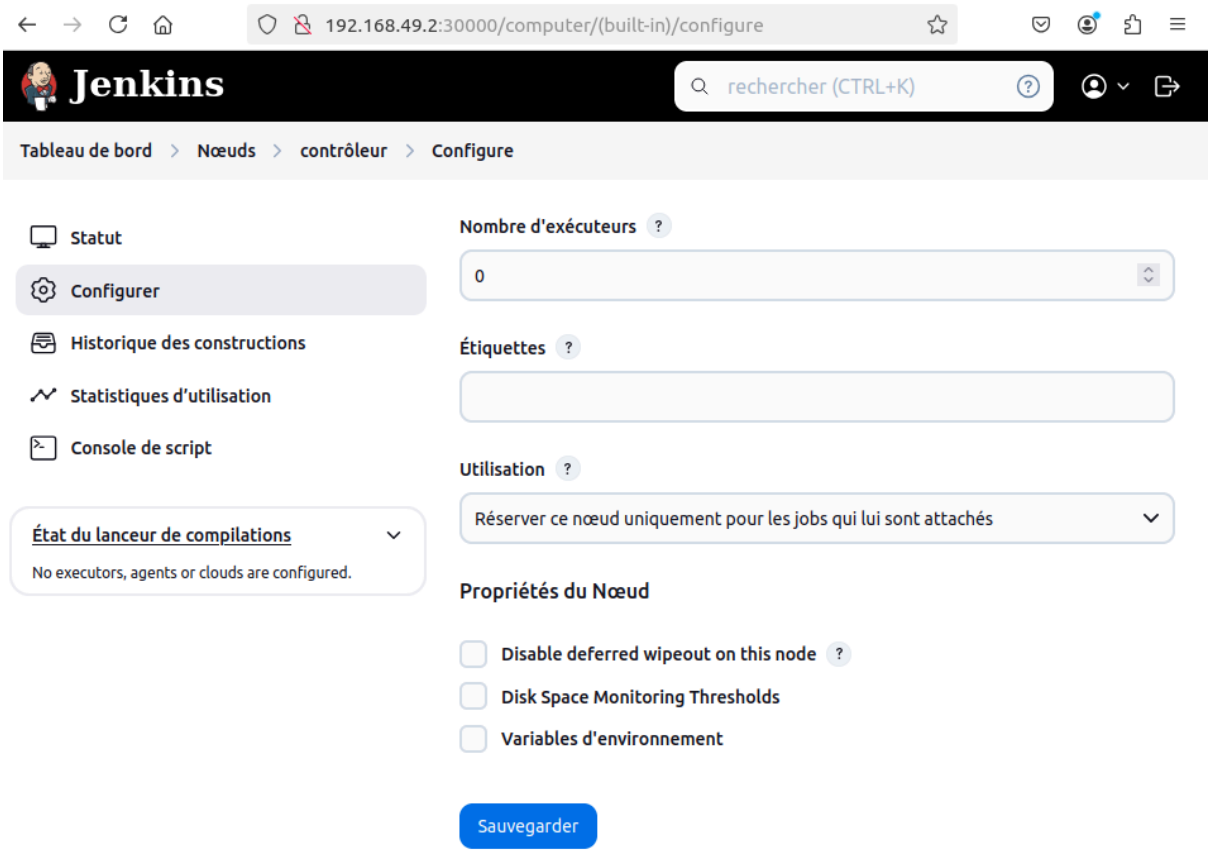


Figure 40: Modification de la configuration du contrôleur

Ensuite, on va s'intéresse à configurer un agent pour les builds. J'ai donc pu créer un nouvel agent dans l'espace de noms jenkins

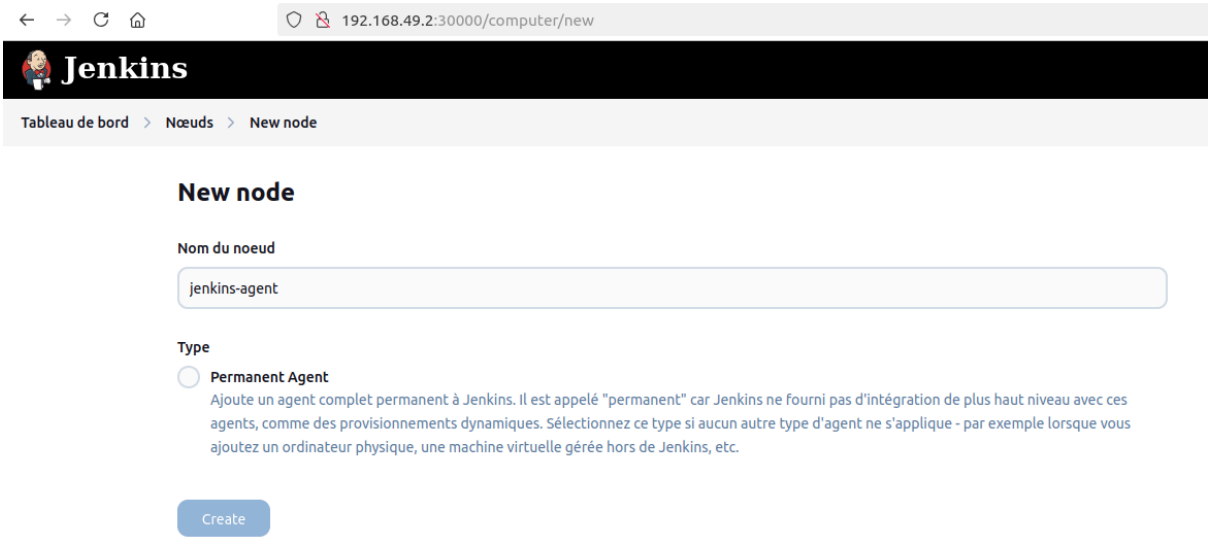


Figure 41: Ajout du nouveau "node" de type Permanent Agent

Il faut cocher la checkbox permanent agent pour pouvoir cliquer sur le bouton Create . Pour pouvoir le pour effectuer les builds j'ai pu associer un label kubeagent à l'agent.

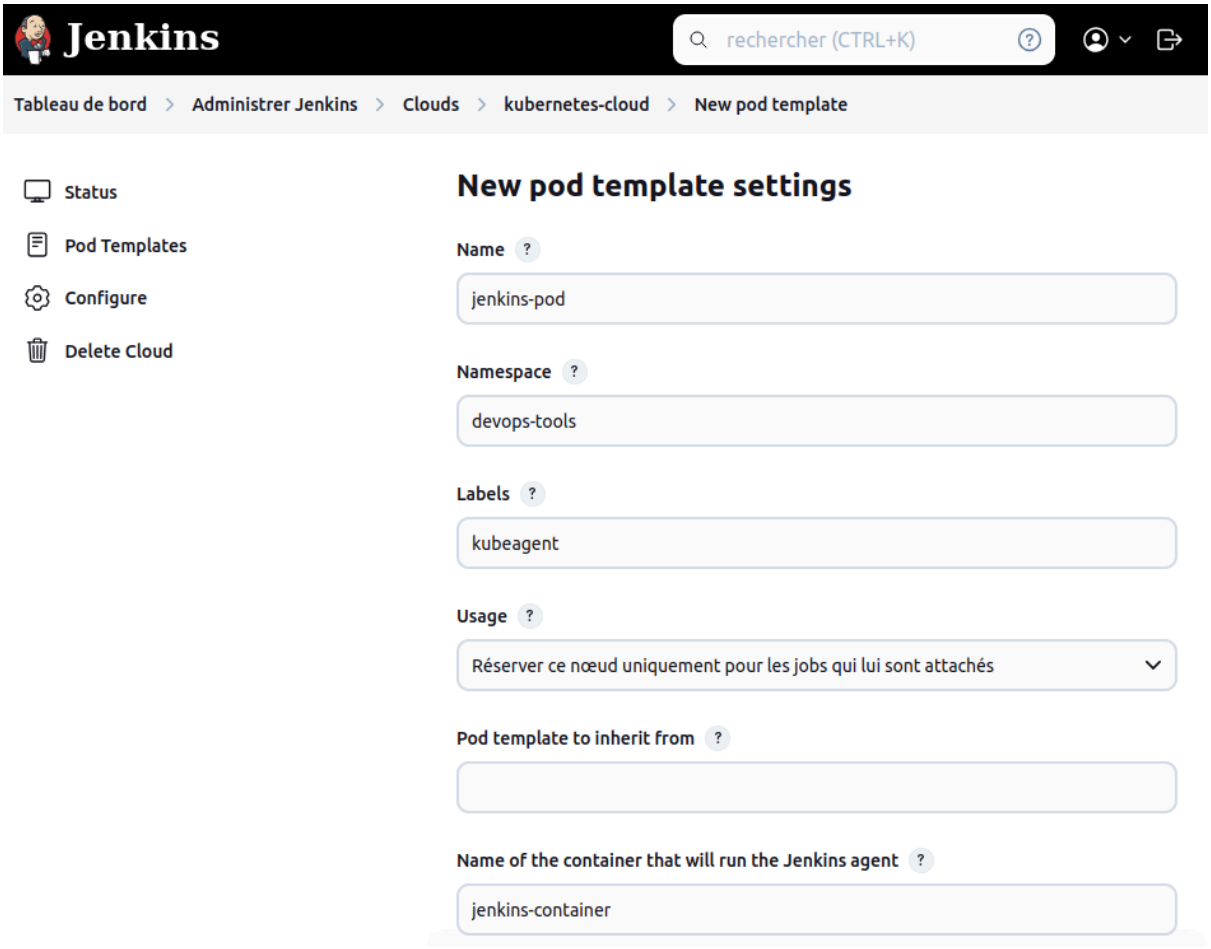


Figure 42: Création du nouveau pod template de nom jenkins-pod

J'ai ensuite pu configurer le template avec le nom jnlp avec l'image jenkins/inbound-agent:latest et le working directory /home/jenkins/agent

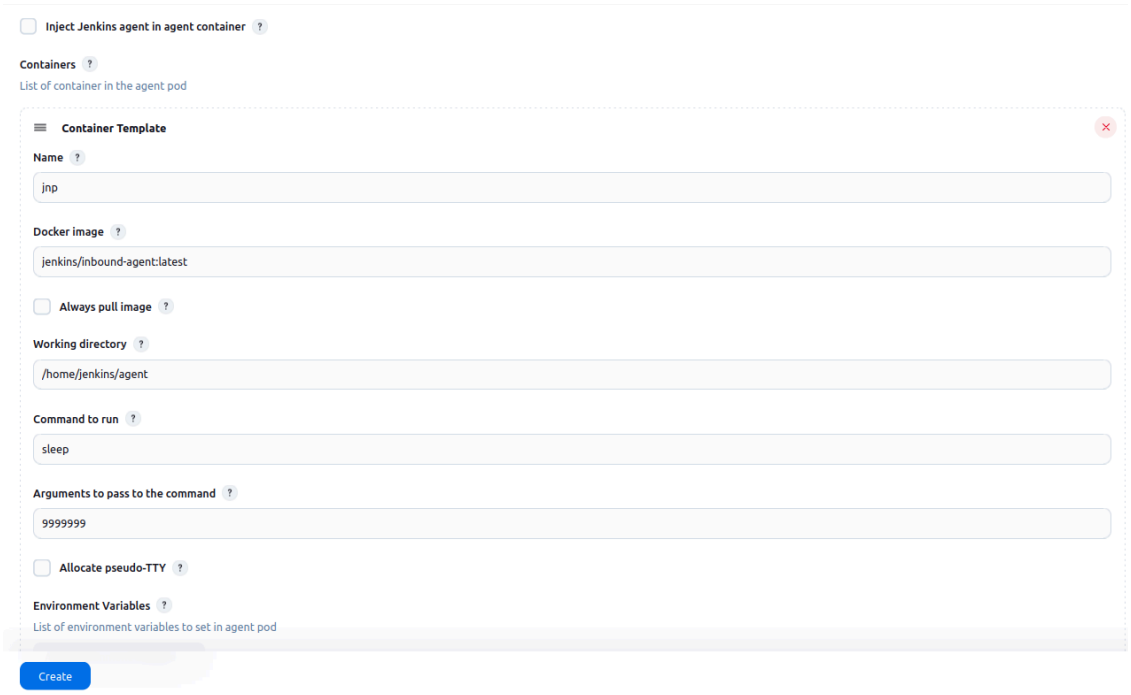


Figure 43: Création et configuration du contianer template

Le pod templates est désormais visible dans la liste des pods tempaltes

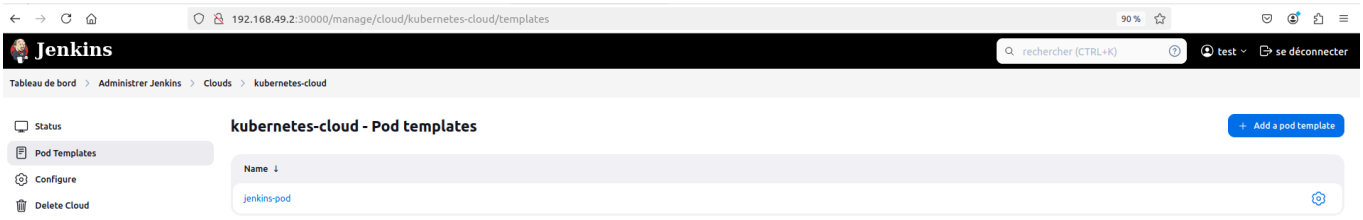


Figure 44: Le pod jenkins-pod apaprait dans les pods templates

# 4 Programmer des tâches

## 4.1 Tâche de test

Nous allons désormais créer une tâche pour tester si les agents de builds peuvent être lancés correctement en affichant simplement le message ”succès” comme étape de build unique. Pour cela lors de la création du nouvel item, j’ai cliqué sur Pipeline

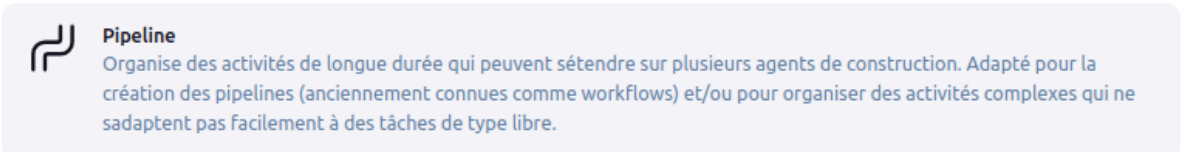


Figure 45: Sélection d’un nouvel item de type Pipeline

J’ai ensuite pu rentrer le Pipeline Script


Figure 46: Création du Pipeline Script

Cela a pu générer un jenkins pipeline comme ci dessous


Figure 47: Création du Jenkins Pipeline

Le pipeline attend que l'agent se réveille

Figure 48: Etat déconnecté de l'agent jenkings-agent

L'agnet jenkins-agent est actuellement dans un état déconnecté, il faut donc exécuter quelques commandes sur  VM qui exécute le jenkins

```
athena@ubuntumate2204:~$ curl -sO http://192.168.49.2:30000/jnlpJars/agent.jar
athena@ubuntumate2204:~$ echo 159aa1ef08510bbccaefc9afd53a2da257d00f87fc0c69cf3601debee47369ae > secret-file
athena@ubuntumate2204:~$ java -jar agent.jar -url http://192.168.49.2:30000/ -secret @secret-file -name "jenkins-agent" -webSocket -workDir "/home/athena/jenkins-agent-workdir"
nov. 26, 2024 7:47:52 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFOS: Using /home/athena/jenkins-agent-workdir/remoting as a remoting work directory
nov. 26, 2024 7:47:52 PM org.jenkinsci.remoting.engine.WorkDirManager setupLogging
INFOS: Both error and output logs will be printed to /home/athena/jenkins-agent-workdir/remoting
nov. 26, 2024 7:47:52 PM hudson.remoting.Launcher createEngine
INFOS: Setting up agent: jenkins-agent
nov. 26, 2024 7:47:52 PM hudson.remoting.Engine startEngine
INFOS: Using Remoting version: 3261.v9c670a_4748a_9
nov. 26, 2024 7:47:52 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFOS: Using /home/athena/jenkins-agent-workdir/remoting as a remoting work directory
nov. 26, 2024 7:47:52 PM hudson.remoting.Launcher$CuiListener status
INFOS: WebSocket connection open
nov. 26, 2024 7:47:52 PM hudson.remoting.Launcher$CuiListener status
INFOS: Connected
```

Figure 49: Exécution des commandes pour réveiller l'agent jenkins-agent

Une fois que ces commandes ont été exécuté l'agent est désormais actif et peut répondre aux tâches de build en attente d'être traité.

Il faut vérifier que l'agent dispose d'un répertoire de travail comme sur la figure ci-dessous.

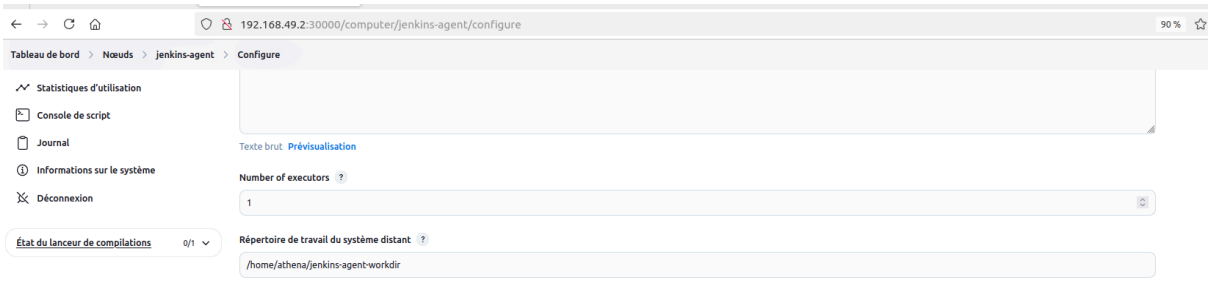


Figure 50: Ajout du répertoire de travail du système distant pour l'agent jenkins-agent

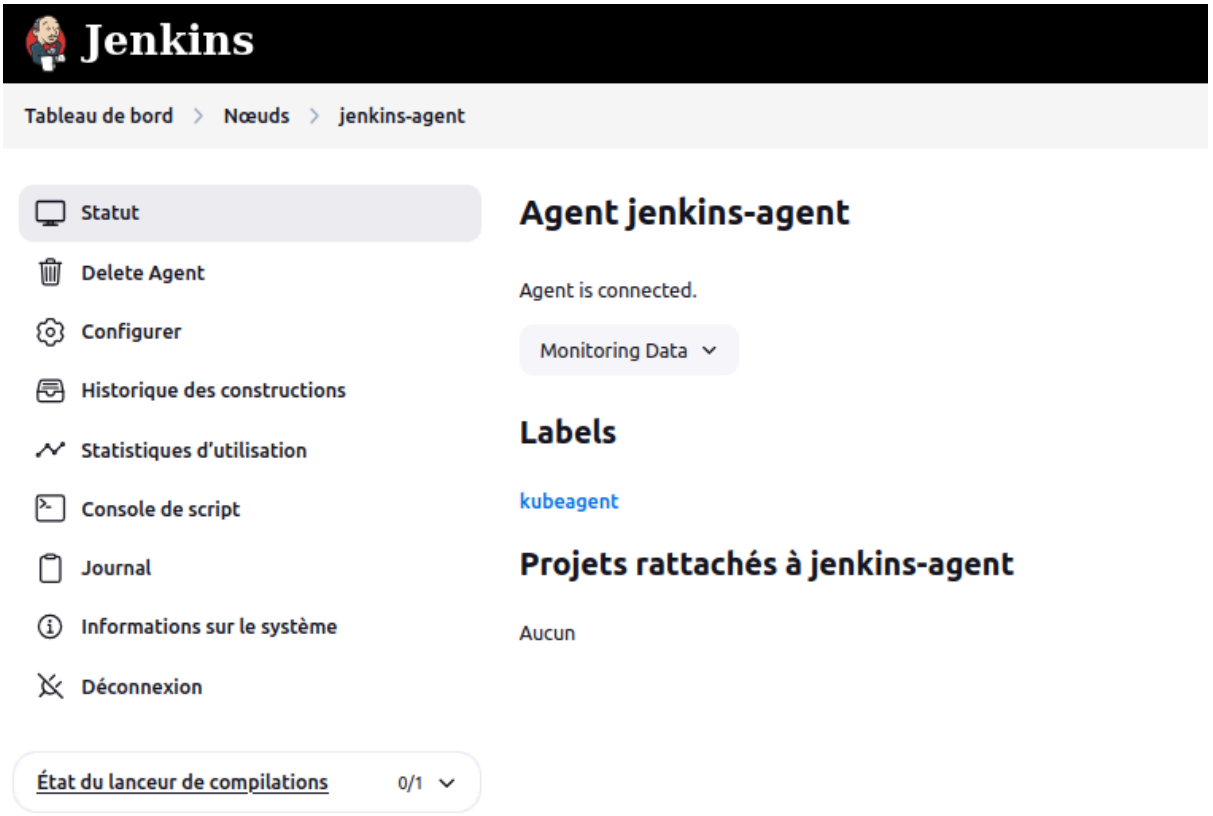


Figure 51: Affichage de l'agent jenkins-agent qui est désormais en état connecté


Figure 52: Lancement du pipeline Jenkins qui peut désormais être exécuter par un agent

## 4.2 Tâche à partir d'un Git

J'ai ensuite pu créer une tâche de build qui doit être exécuté lors d'un commit sur mon repository gitlab. Comme j'ai la branche develop et la branche main, j'ai choisi de créer un item Pipeline Multibranches.


Figure 53: Création d'un nouvel item de type Pipeline Multibranches

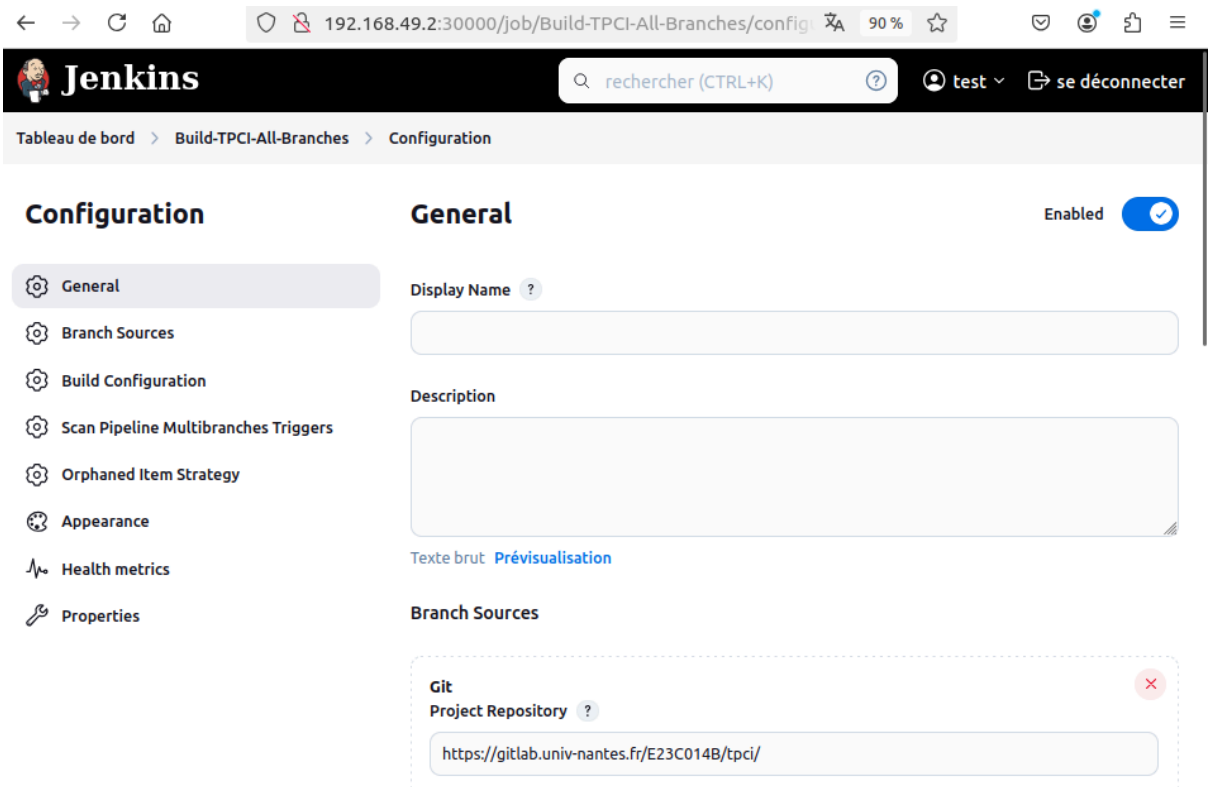


Figure 54: Création d'un pipeline multibranche tapant dans le repository gitlab du projet

J'ai ensuite pu réaliser différents commits dans mes branches

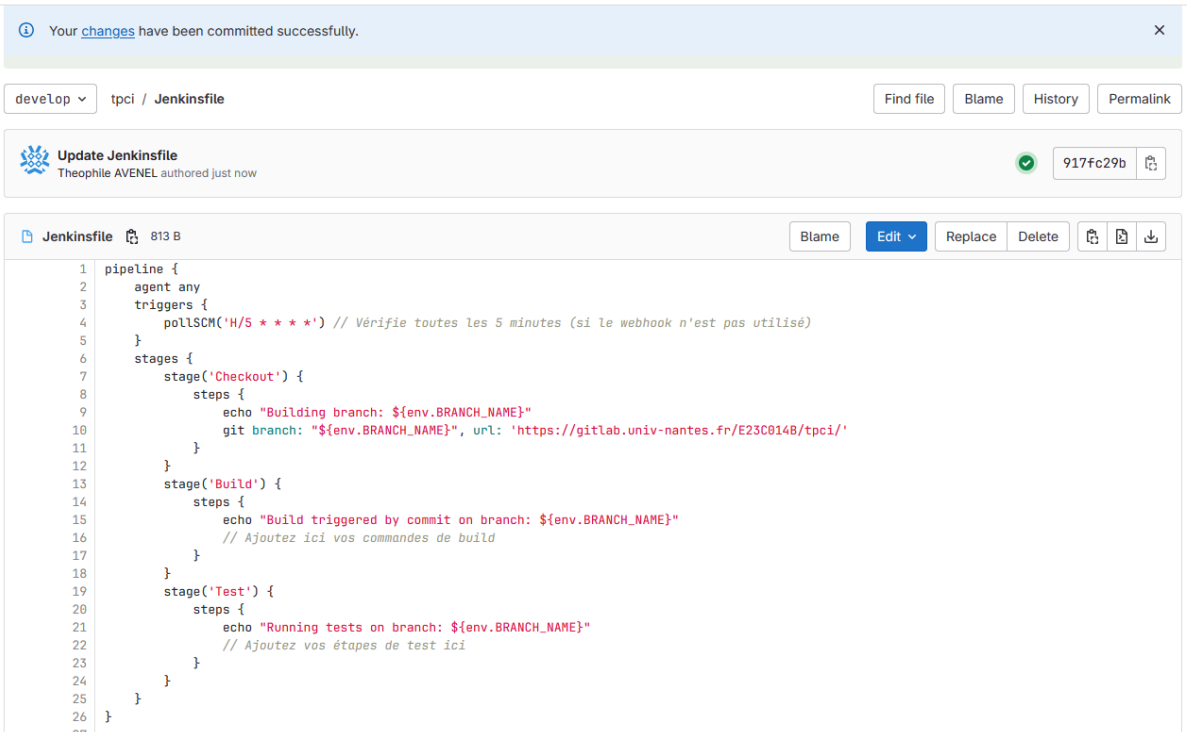


Figure 55: Création du Jenkinsfile mis dans le projet gitlab

Ce Jenkinsfile configure une pipeline CI/CD avec trois étapes principales : Checkout, Build, et Test. Il utilise un agent générique (`agent any`) pour s'exécuter sur n'importe quel nœud disponible. La pipeline est déclenchée automatiquement toutes les 5 minutes via `pollSCM`,

sauf si un webhook est configuré. Dans l'étape Checkout, le code source est extrait de la branche spécifiée (`env.BRANCH_NAME`) depuis un dépôt GitLab. L'étape Build inclut les commandes nécessaires pour compiler ou préparer l'application. Enfin, l'étape Test exécute les tests pour valider le code. Les messages `echo` fournissent des logs pour chaque étape. Voici ci-dessous  le résutlat de l'exécution de ce Jenkinsfile

```
Started by user test
[Tue Nov 26 19:00:52 UTC 2024] Starting branch indexing...
 > git --version # timeout=10
 > git --version # 'git version 2.39.5'
 > git ls-remote --symref -- https://gitlab.univ-nantes.fr/E23C014B/tpci/ #
timeout=10
Creating git repository in /var/jenkins_home/caches/
git-9d3dc4aa44280a46bfc84359c3a57f0b
 > git init /var/jenkins_home/caches/git-9d3dc4aa44280a46bfc84359c3a57f0b #
timeout=10
Setting origin to https://gitlab.univ-nantes.fr/E23C014B/tpci/
 > git config remote.origin.url https://gitlab.univ-nantes.fr/E23C014B/tpci/
# timeout=10
Fetching & pruning origin...
Listing remote references...
 > git config --get remote.origin.url # timeout=10
 > git --version # timeout=10
 > git --version # 'git version 2.39.5'
 > git ls-remote -h -- https://gitlab.univ-nantes.fr/E23C014B/tpci/ #
timeout=10
Fetching upstream changes from origin
 > git config --get remote.origin.url # timeout=10
 > git fetch --tags --force --progress --prune -- origin +refs/heads/*:refs/
remotes/origin/* # timeout=10
Checking branches...
  Checking branch develop
      'Jenkinsfile' found
    Met criteria
Scheduled build for branch: develop
  Checking branch main
      'Jenkinsfile' not found
    Does not meet criteria
Processed 2 branches
[Tue Nov 26 19:01:01 UTC 2024] Finished branch indexing. Indexing took 8.9
sec
Finished: SUCCESS
```

Figure 56: résultat d'exécution du Jenkinsfile

Le jinkins a pu tester un changement au lancement dans la base de code

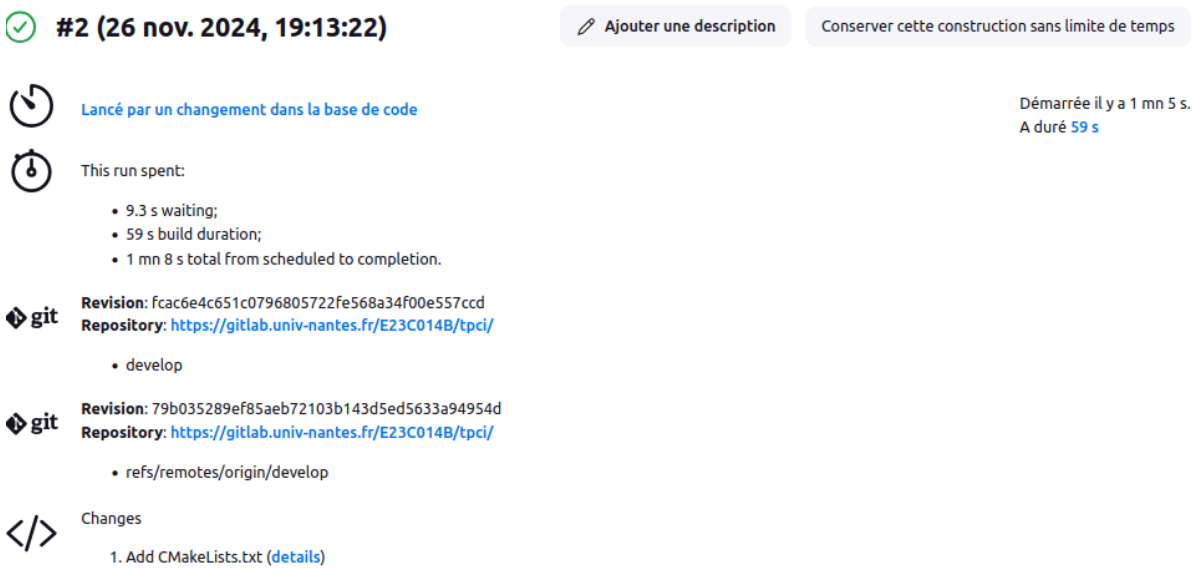


Figure 57: Pipeline de test Jenkins effectué à la suite d'un changement dans la base de code du repository Gitlab

Nous avons ensuite pu automatiser la compilation à l'aide de l'outil cmake et et effectuer le build par Jenkins à partir de cmake.

J'ai donc commencé par modifier le code du JenkinsFile

https://gitlab.univ-nantes.fr/E23C014B/tpci/-/blob/main/Jenkinsfile?ref_type=heads

Ce Jenkinsfile automatise une pipeline CI/CD pour un projet C/C++ utilisant CMake. La pipeline s'exécute sur n'importe quel agent disponible et vérifie les changements toutes les 5 minutes grâce à `pollSCM`. L'étape Checkout clone la branche spécifiée depuis le dépôt Git. L'étape Build utilise CMake pour configurer le répertoire de build et génère les fichiers makefiles, puis compile le projet avec `make`. Les fichiers générés sont archivés si la compilation réussit. Les étapes Test C et Test C++ exécutent respectivement les fichiers binaires `main_c` et `main_cpp` en simulant une entrée utilisateur (`echo "Alice"`). Enfin, les blocs post enregistrent le statut global de la pipeline, affichant des messages en cas de succès ou d'échec. Ce workflow garantit une intégration continue efficace pour un projet multi-langage.

J'ai aussi pu créer un fichier CMakeLists.txt

https://gitlab.univ-nantes.fr/E23C014B/tpci/-/blob/main/CMakeLists.txt?ref_type=heads

Ce fichier CMakeLists.txt configure le projet CMake. Il définit la version minimale de CMake (3.10) et nomme le projet `MultiLangProject`. Deux exécutables sont configurés : `main_c`, qui compile les fichiers C (`main.c` et `calcul.c`), et `main_cpp`, qui compile les fichiers C++ (`main.cpp` et `calcul.cpp`). Ce fichier permet une gestion simple et centralisée de la compilation pour les codes C et C++. Cependant lors de l 'exécution je suis tombé sur une erreur

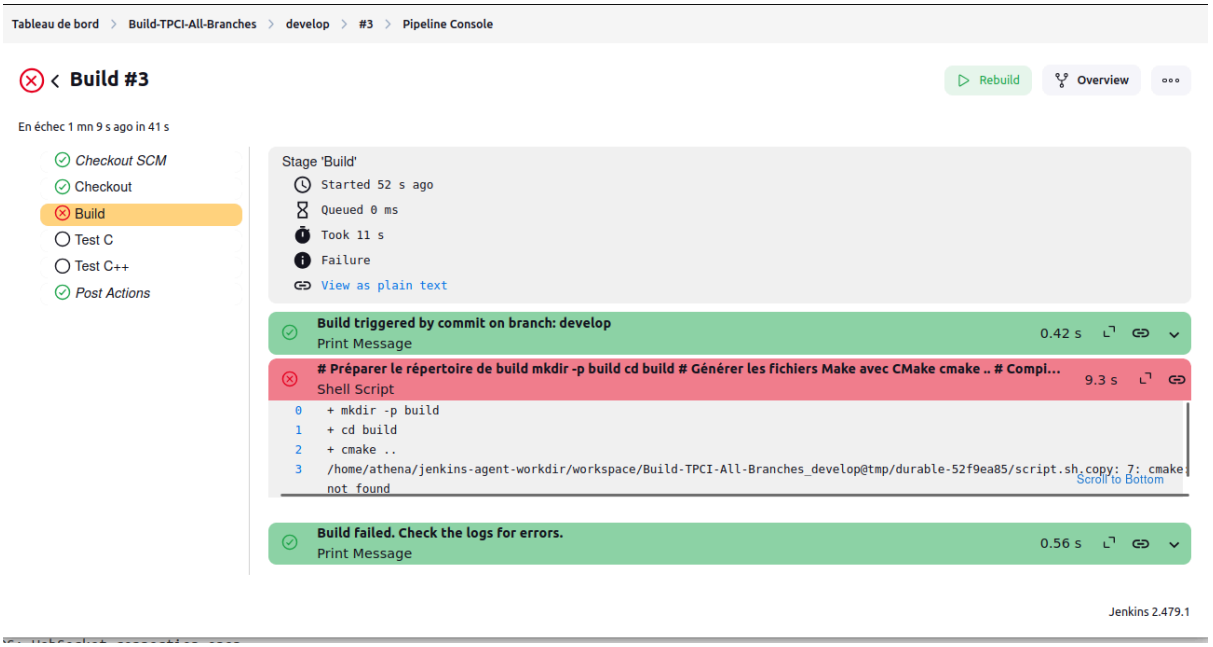


Figure 58: Erreur à l'étape du cmake sur le pipeline Jenkins

L'erreur cmake not found signifie que cmake doit être installé sur la VM qu'a lancé l'agent. J'ai donc utilisé la commande sudo apt update suivi de la commande sudo apt install cmake -y pour isnitaller cmake pusi j'ai vérifié l'istalaltion de cgmake avec la commande cmake –version.

```
athena@ubuntumate2204:~$ sudo apt update
sudo apt install cmake -y
```

Figure 59: Installation de cmake

```
athena@ubuntumate2204:~$ cmake --version
cmake version 3.22.1

CMake suite maintained and supported by Kitware (kitware.com/cmake).
athena@ubuntumate2204:~$
```

Figure 60: Vérification de l'installation de cmake

Après cette vérification, j'ai pu rebuilder er le pipeline est passé dans l'état success


Figure 61: Le pipeline de test Jenkins s'est correctement exécuté

Pour voir une vidéo finale de l'exécution des pipelines de gitlab lors d'un commit dans develop, du pipeline CI/CD , d'une merge request de la branche develop de la branche main et les différentes exécutions de builds sur Jinkins :
https://www.youtube.com/watch?v=gqHrv-xkqx0