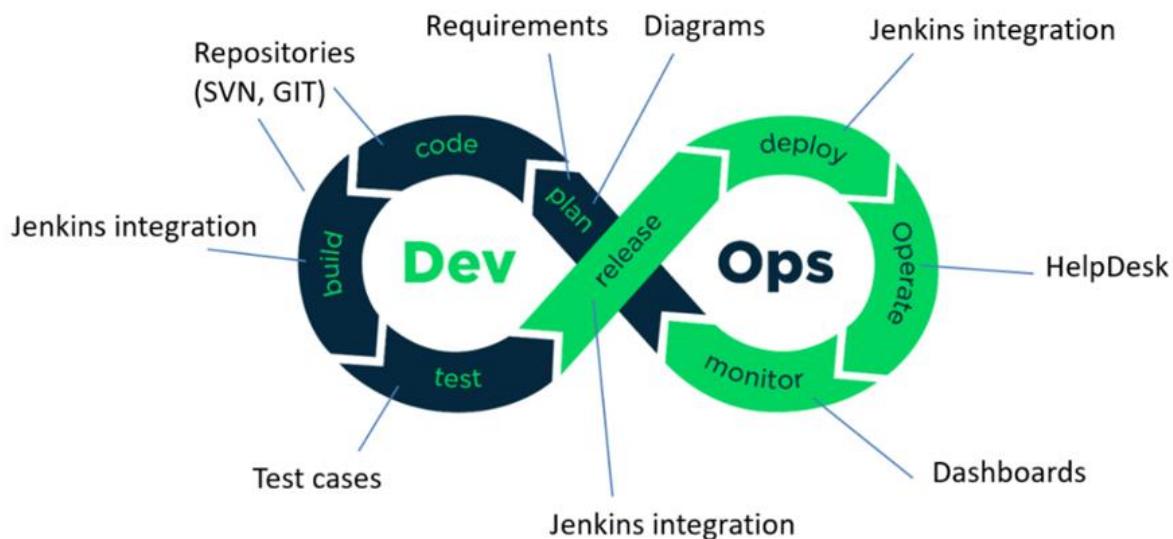


Projet final de la formation « DevOps »

Abdelkader RAHMANI
Aurélien DAIX
Oussama ZAID
Renaud SAUTOUR

Mise en place d'une chaîne complète CICD pour une application Node EJS

Projet proposé et encadré par Mr. Frazer SADO



I- Introduction

A- La Philosophie DevOps

Le DevOps offre un framework conçu pour dynamiser et améliorer le développement d'applications et accélérer la mise à disposition de nouvelles fonctionnalités, de mises à jour logicielles ou de produits.

Il favorise la communication, la collaboration, l'intégration, la visibilité et la transparence continues entre les équipes chargées du développement d'applications (Dev) et celles responsables des opérations IT (Ops).

Cette relation plus étroite entre Dev et Ops se reflète dans chaque phase du cycle de vie DevOps : planification logicielle initiale, codage, développement, test, publication, déploiement, opérations et surveillance continue. Elle génère de façon constante des retours clients, ce qui renforce le potentiel d'amélioration lors du développement, des tests et du déploiement. La publication accélérée et permanente des modifications ou ajouts de fonctionnalités en est un exemple.

Les objectifs du DevOps s'articulent autour de quatre catégories : culture, automatisation, mesure et partage. Dans chacun de ces domaines, les outils DevOps améliorent la rationalisation et la collaboration des workflows de développement et d'opérations en automatisant les tâches chronophages, manuelles ou statiques des phases d'intégration, de développement, de test, de déploiement ou de surveillance.

B- Notre Pratique du DevOps

Notre pratique DevOps nous permet d'améliorer en continu en automatisant les processus sur plusieurs phases du cycle de développement :

- Développement continu. Cette pratique couvre les phases de planification et de codage dans le cycle de vie DevOps et peut inclure des mécanismes de contrôle des versions.
- Tests continus. Cette pratique prévoit des tests automatisés, planifiés et continus lors de l'écriture ou de la mise à jour du code de l'application qui accélèrent la livraison du code en production.
- Intégration continue. Cette pratique rassemble des outils de gestion de la configuration, de test et de développement pour assurer le suivi de la mise en production des différentes portions du code. Elle implique une collaboration étroite entre les équipes responsables des tests et du développement pour identifier et résoudre rapidement les problèmes de code.
- Livraison continue. Cette pratique automatise la publication des modifications du code après la phase de test, dans un environnement intermédiaire ou de préproduction. Un membre de l'équipe peut décider de publier ces modifications dans l'environnement de production.
- Déploiement continu. À l'instar de la livraison continue, cette pratique automatise la publication d'un code nouveau ou modifié dans l'environnement de production. Les entreprises peuvent être amenées à publier plusieurs fois par jour des modifications du code ou des fonctionnalités. Dans un contexte de déploiement continu, les technologies de conteneur comme Docker et Kubernetes assurent la cohérence du code entre plusieurs plateformes et environnements.

C- Nos outils DevOps

- Nous allons gérer le code source à l'aide de l'outil du système de contrôle de version Git.
- Automatisez le processus de création de code à l'aide d'outil CI Jenkins.
- Scannez le code pour détecter les failles avec Snyk.
- Créez une image et la déployez avec une technologie conteneurisée Docker.
- Faire évoluer l'application à l'aide de l'outil d'orchestration de conteneurs Kubernetes.

D- Infrastructure en tant que code

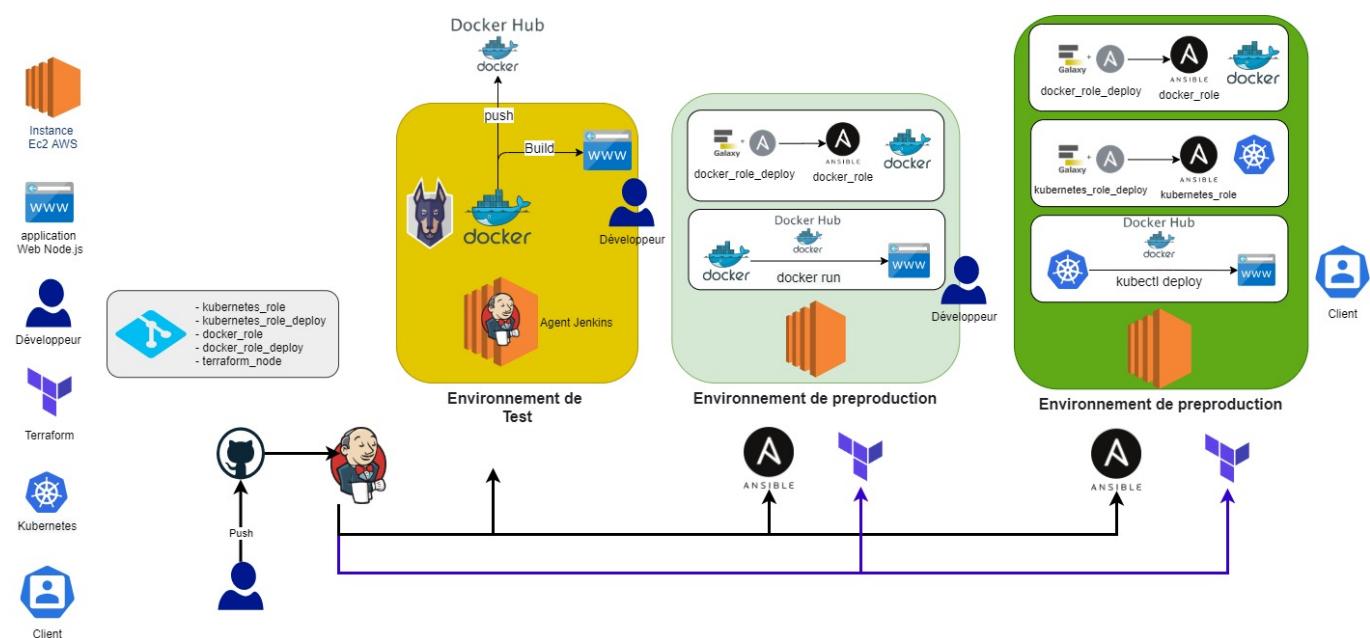
Il s'agit du processus de gestion de l'infrastructure dans un ou plusieurs fichiers plutôt que de configurer manuellement les ressources dans une interface utilisateur. Une ressource dans cette instance est toute pièce d'infrastructure dans un environnement donné, telle qu'une machine virtuelle, un groupe de sécurité, une interface réseau, etc.

À un niveau élevé, Terraform permet aux opérateurs d'utiliser HCL pour créer des fichiers contenant les définitions de leurs ressources souhaitées sur presque tous les fournisseurs (AWS, GCP, GitHub, Docker, etc.) et automatise la création de ces ressources au moment de la candidature.

Dans cette piste, nous couvrirons les fonctions de Terraform pour créer une infrastructure sur AWS.

II- Préparation du projet

A- Infrastructure cible du projet



B- Prérequis

- Compte GitHub (projetajc-group3)
- Compte DockerHub (projetajcgroup3)
- Compte Slack (projetajc.group3)
- Compte utilisateur AWS

C- Création des serveurs pour Jenkins sur AWS

Nous utilisons ici Jenkins qui un serveur d'automatisation autonome et open source utilisé pour automatiser les tâches associées à la création, aux tests et à la livraison/déploiement de logiciels. Jenkins Pipeline implémente des pipelines de livraison continue dans Jenkins grâce à l'utilisation de plugins et d'un fichier Jenkins.

Dans ce projet, nous allons créer 2 instances AWS pour Jenkins:

- Jenkins master: serveur qui pilotera l'ensemble du pipeline
- Jenkins agent: agent dédié au build/scan/test du projet

1- Création d'une instance ec2 pour Jenkins master

- t2.large
- SSD 20 Go
- Sécurité Group : 22 / 8080
- Key pair: key-projetgrp3.pem

Vous avez été invité à essayer une itération anticipée de l'assistant de lancement d'instance. Nous allons continuer à améliorer l'expérience au cours des prochains mois. Nous demandons à un petit groupe de clients de nous faire part de leurs commentaires sur cette version anticipée. Pour quitter l'assistant de lancement d'instance à tout moment, cliquez sur le bouton Annuler.

L'essayer maintenant

Étape 1 : Sélection d'une Amazon Machine Image (AMI)

Annuler et quitter

Une AMI est un template qui contient la configuration logicielle (par ex., un système d'exploitation, un serveur d'applications et des applications) nécessaire pour lancer votre instance. Vous pouvez sélectionner une AMI fournie par AWS, notre communauté d'utilisateurs ou AWS Marketplace ; vous pouvez également sélectionner une de vos propres AMI.

Ubuntu

Rechercher par paramètre Systems Manager

Quick Start (7)

<input type="checkbox"/>	Mes AMI (0)		Ubuntu Server 20.04 LTS (HVM), SSD Volume Type - ami-04505e74c0741db8d (64 bits x86) / ami-0b49a4a6e8e22fa16 (64 bits Arm)	<input type="button" value="Sélectionner"/>	<input checked="" type="radio"/> 64 bits (x86)	<input type="radio"/> 64 bits (Arm)	
<input type="checkbox"/>	AWS Marketplace (1069)		Ubuntu Server 20.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).				
<input type="checkbox"/>	AMI de la communauté (39225)		Type de périphérique racine: ebs Type de virtualisation: hvm ENA activée: Oui				
<input type="checkbox"/>	Offre gratuite		Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-0e472ba40eb589f49 (64 bits x86) / ami-0a940cb939351ccca (64 bits Arm)	<input type="button" value="Sélectionner"/>	<input checked="" type="radio"/> 64 bits (x86)	<input type="radio"/> 64 bits (Arm)	

Étape 2 : Choisir un type d'instance

Amazon EC2 fournit un vaste éventail de types d'instances optimisés pour différents cas d'utilisation. Les instances sont des serveurs virtuels qui peuvent exécuter des applications. Les types d'instances se composent de différentes combinaisons de processeur, de mémoire, de stockage et de capacité réseau, et vous offrent une flexibilité dans le choix de l'association de ressources adaptées à vos applications. [En savoir plus](#) à propos des types d'instances et de la manière dont ils peuvent répondre à vos besoins informatiques.

Filtrer par:

Actuellement sélectionné : t2.large (- ECU, 2 vCPU, 2.3 GHz, -, 8 Gio mémoire, EBS uniquement)

	Famille	Type	vCPU	Mémoire (Go)	Stockage d'instance (Go)	Disponible en version optimisée pour EBS	Performances réseau	Prise en charge IPv6
<input type="checkbox"/>	t2	t2.nano	1	0.5	EBS uniquement	-	Faibles à modérées	Oui
<input type="checkbox"/>	t2	t2.micro <small>Eligible à l'offre gratuite</small>	1	1	EBS uniquement	-	Faibles à modérées	Oui
<input type="checkbox"/>	t2	t2.small	1	2	EBS uniquement	-	Faibles à modérées	Oui
<input type="checkbox"/>	t2	t2.medium	2	4	EBS uniquement	-	Faibles à modérées	Oui
<input checked="" type="checkbox"/>	t2	t2.large	2	8	EBS uniquement	-	Faibles à modérées	Oui
<input type="checkbox"/>	t2	t2.xlarge	4	16	EBS uniquement	-	Modérées	Oui

Étape 3 : Configurer les détails de l'instance

Configurez l'instance en fonction de vos besoins. Vous pouvez lancer plusieurs instances à partir de la même AMI, demander des instances Spot pour bénéficier d'un tarif inférieur, attribuer un rôle de gestion d'accès à l'instance et bien d'autres choses encore.

Nombre d'instances **Lancer dans le groupe Auto Scaling**

Option d'achat Demander des instances Spot

Réseau

Sous-réseau

Attribuer automatiquement l'adresse IP publique

Type de nom d'hôte

DNS Hostname Enable IP name IPv4 (A record) DNS requests
 Activer les demandes DNS IPv4 (enregistrement A) basées sur les ressources
 Activer les demandes DNS IPv6 (enregistrement AAAA) basées sur les ressources

Annuler **Précédent** **Vérifier et lancer** **Suivant : Ajouter le stockage**

Étape 4 : Ajouter le stockage

Votre instance sera lancée avec les paramètres suivants du périphérique de stockage. Vous pouvez attacher des volumes EBS supplémentaires et des volumes de stockage d'instance à votre instance ou modifier les réglages du volume racine. Vous pouvez également attacher des volumes EBS supplémentaires après le lancement d'une instance, mais pas des volumes de stockage d'instance. [En savoir plus](#) sur les options de stockage dans Amazon EC2.

Type de volume	Dispositif	Instantané	Taille (Go)	Type de volume	IOPS	Débit (Mo/s)	Supprimer à la résiliation	Chiffrement
Racine	/dev/sda1	snap-0f7a6eae6d90437c4	<input type="text" value="20"/>	Volume à usage général SSD (gp2)	100 / 3000	N/A	<input checked="" type="checkbox"/>	Non chiffré

Ajouter un nouveau volume

Les clients éligibles à l'offre gratuite peuvent obtenir jusqu'à 30 Go de stockage EBS à usage général (SSD) ou magnétique. [En savoir plus](#) à propos de l'éligibilité pour le niveau d'offre gratuite et des restrictions d'utilisation.

▼ Shared file systems (i)

Annuler **Précédent** **Vérifier et lancer** **Suivant : Ajouter des balises**

Étape 5 : Ajouter des balises

Une balise est composée d'une paire de clé-valeur sensible à la casse. Par exemple, vous pourriez définir une balise avec clé = Nom et valeur = Serveur. Une copie de balise peut être appliquée aux volumes, aux instances ou aux deux. Les balises seront appliquées à toutes les instances et à tous les volumes. [En savoir plus](#) à propos du balisage de vos ressources Amazon EC2.

Clé (128 caractères maximum)	Valeur (256 caractères maximum)	Instances	Volumes	Interfaces réseau
Name	ec2-jenkins-projetgrp3	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
formation	Frazer	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Ajouter une autre balise (Maximum 50 balises)

Annuler **Précédent** **Vérifier et lancer** **Suivant : Configurer le groupe de sécurité**

Étape 6 : Configurer le groupe de sécurité

Un groupe de sécurité est un ensemble de règles de pare-feu qui contrôlent le trafic de votre instance. Sur cette page, vous pouvez ajouter des règles pour permettre qu'un trafic spécifique atteigne votre instance. Par exemple, si vous voulez configurer un serveur Web et permettre au trafic Internet d'atteindre votre instance, ajoutez des règles qui autorisent un accès restreint aux ports HTTP et HTTPS. Vous pouvez créer un nouveau groupe de sécurité ou en sélectionner un parmi les groupes existants ci-dessous. [En savoir plus](#) à propos des groupes de sécurité Amazon EC2.

Attribuer un groupe de sécurité: Créez un nouveau groupe de sécurité Sélectionnez un groupe de sécurité existant

Nom du groupe de sécurité: projetgrp3-sg

Description: Security Group for project group 3

Type	Protocole	Plage de ports	Source	Description
SSH	TCP	22	N'importe où 0.0.0.0/0, ::/0	par exemple SSH for Admin Desktop
Règle TCP per	TCP	80	N'importe où 0.0.0.0/0, ::/0	par exemple SSH for Admin Desktop
Règle TCP per	TCP	8080	N'importe où 0.0.0.0/0, ::/0	par exemple SSH for Admin Desktop

[Ajouter une règle](#)

Avertissement
Les règles avec une source de 0.0.0.0/0 permettent à toutes les adresses IP d'accéder à votre instance. Nous recommandons de paramétriser les règles du groupe de sécurité afin de permettre l'accès.

Annuler Précédent Vérifier et lancer

paire de clés

Une paire de clés se compose d'une **clé publique** détenue par AWS et d'une **clé privée, incluse dans un fichier**, que vous conservez. Ensemble, elles vous permettent de vous connecter à votre instance en toute sécurité. Avec les AMI Windows, une clé privée est requise afin d'obtenir le mot de passe utilisé pour se connecter à votre instance. Avec les AMI Linux, la clé privée vous permet d'accéder en toute sécurité à votre instance via SSH. Amazon EC2 prend en charge les types de paire de clés ED25519 et RSA.

Remarque : La paire de clés sélectionnée sera ajoutée à l'ensemble de clés autorisé pour cette instance. En savoir plus sur [la suppression de paires de clés existantes d'une AMI publique](#).

Créer une nouvelle paire de clés

Type de paire de clés
 RSA ED25519

Nom de la paire de clés
key-projetgrp3

Télécharger une paire de clés

Vous devez télécharger le fichier de clé privée (fichier *.pem) avant de pouvoir continuer. Stockez-le dans un endroit sûr et accessible. Vous ne pourrez pas le télécharger à nouveau après sa création.

Annuler Lancer des instances

Résultat:

- Nom de l'instance ec2 : **projetgrp3-ec2-jenkins**
- ID de mon instance ec2 : **i-04ce15266a590e510**
- IPv4 publique de mon instance ec2 : **54.174.144.82**

2- Création d'une instance ec2 pour Jenkins agent

De la même manière nous créons une instance ec2 avec les caractéristiques suivantes:

- t2.large
- SSD 20 Go
- Sécurité Group : 22 / 8080
- Key pair: key-projetgrp3.pem

Résultat:

- Nom de l'instance ec2 : **projetgrp3-ec2-jenkins-agent**
- ID de mon instance ec2 : **i-01e49b48c289b3992**
- IPv4 publique de mon instance ec2 : **54.147.236.68**

D- Installation de Jenkins

1- Commande de l'installation de java

```
sudo apt-get -y update
sudo apt-get -y install openjdk-11-jre-headless
sudo apt-get -y install default-jre
java -version
```

Résultat:

```
openjdk version "11.0.13" 2021-10-19
OpenJDK Runtime Environment (build 11.0.13+8-Ubuntu-0ubuntu1.20.04)
OpenJDK 64-Bit Server VM (build 11.0.13+8-Ubuntu-0ubuntu1.20.04, mixed mode)
```

2- Commande de l'installation de jenkins

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key
add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
sudo apt-get -y update
sudo apt-get -y install jenkins
sudo systemctl start jenkins
sudo systemctl status jenkins
```

E- Premier démarrage de Jenkins

On se connecte sur le serveur Jenkins: <http://54.174.144.82:8080/>

Démarrage

Débloquer Jenkins

Pour être sûr que que Jenkins soit configuré de façon sécurisée par un administrateur, un mot de passe a été généré dans le fichier de logs ([où le trouver](#)) ainsi que dans ce fichier sur le serveur :

`/var/lib/jenkins/secrets/initialAdminPassword`

Veuillez copier le mot de passe depuis un des 2 endroits et le coller ci-dessous.

Mot de passe administrateur

.....

Continuer



L'assistant de configuration post-installation démarre. Lorsque vous accédez pour la première fois à une nouvelle instance Jenkins, vous êtes invité à la déverrouiller à l'aide d'un mot de passe généré automatiquement.

- La commande suivante imprimera le mot de passe sur la console sur serveur Jenkins master:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
6384c868fe614bd3ac72b3495daf7ae3
```

- On installe les plugins suggérés et on crée un utilisateur

Démarrage

Personnaliser Jenkins

Les plugins étendent Jenkins avec des fonctionnalités additionnelles pour satisfaire différents besoins.

Installer les plugins suggérés

Installer les plugins que la communauté Jenkins trouve les plus utiles.

Sélectionner les plugins à installer

Sélectionner et installer les plugins les plus utiles à vos besoins.

Jenkins 2.319.1

- On renseigne les informations du 1er utilisateur:

Démarrage

Créer le 1er utilisateur Administrateur

Nom d'utilisateur:	Administrator
Mot de passe:
Confirmation du mot de passe:
Nom complet:	Administrator
Adresse courriel:	admin@umanis.com

Jenkins 2.319.1

[Continuer en tant qu'Administrateur](#)

[Sauver et continuer](#)

Démarrage

Configuration de l'instance

URL de Jenkins :

L'URL de Jenkins est utilisée pour fournir l'URL de base pour les liens absous vers les diverses ressources Jenkins. Cela signifie que cette valeur est nécessaire pour le bon fonctionnement de nombreuses fonctionnalités de Jenkins, notamment les notifications par mail, les mises à jour des statuts des pull requests, et la variable d'environnement `BUILD_URL` fournie pour les étapes de build.

La valeur par défaut affichée **n'est pas encore sauvegardée** et est générée à partir de la requête actuelle, lorsque c'est possible. Il est fortement recommandé d'utiliser comme valeur l'URL qui est censée être utilisée par les utilisateurs. Cela évitera des confusions lors du partage ou de la visualisation de liens.

Jenkins 2.319.1

[Passer cette étape et terminer](#)

[Sauver et terminer](#)

Démarrage

Jenkins est prêt !

L'installation de votre Jenkins est terminée.

[Commencer à utiliser Jenkins](#)

Jenkins 2.319.1

 Jenkins

Tableau de bord

-  Nouveau Item
-  Utilisateurs
-  Historique des constructions
-  Administrer Jenkins
-  Mes vues
-  Ressources Verrouillables
-  Créer une Vue

Bienvenue sur Jenkins !

Vos jobs Jenkins seront affichés sur cette page. Pour commencer, vous pouvez mettre en place un build distribué ou commencer à créer un projet.

Commencer à créer votre projet

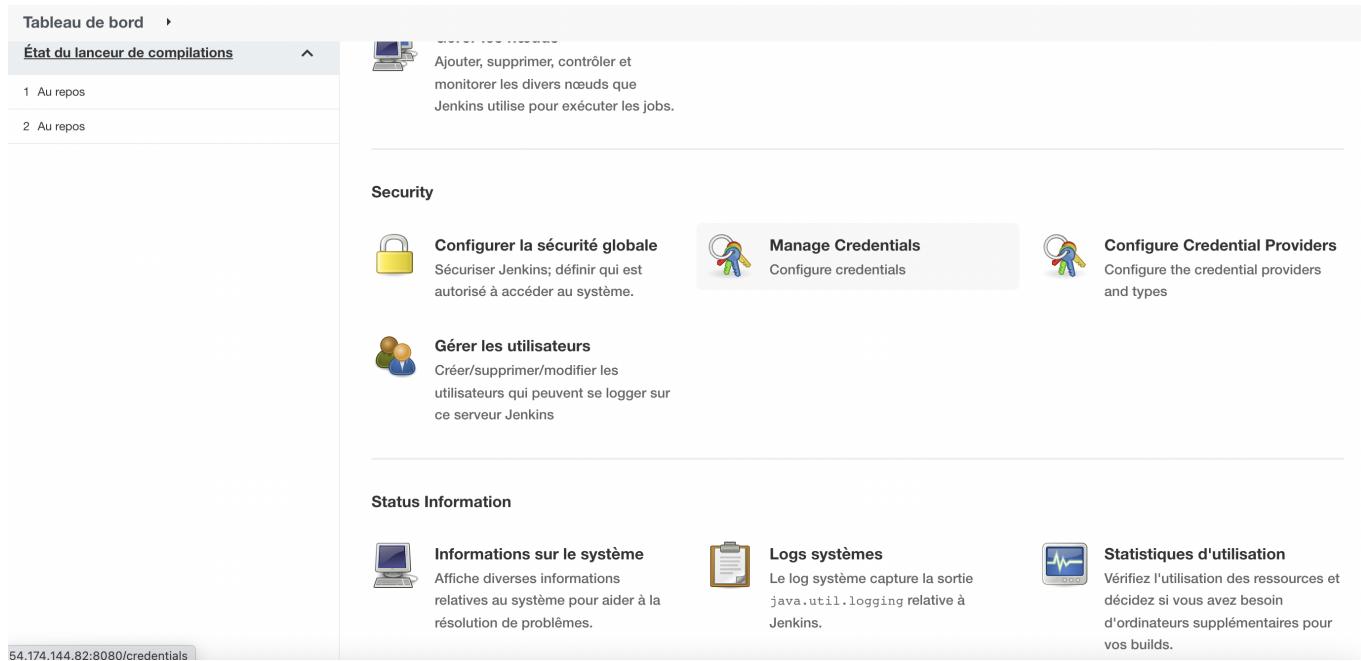
[Créer un job](#) →

Configurer un build distribué

[Mettre en place un agent](#) →

[Configurer un cloud](#) →

[En apprendre plus sur les builds distribués](#) ↗



The screenshot shows the Jenkins dashboard with the following sections and links:

- Tableau de bord** (Dashboard)
- État du lanceur de compilations** (Build Executor Status)
 - 1 Au repos
 - 2 Au repos
- Security**
 - Configurer la sécurité globale** (Configure global security): Sécuriser Jenkins; définir qui est autorisé à accéder au système.
 - Gérer les utilisateurs** (Manage users): Créer/supprimer/modifier les utilisateurs qui peuvent se logger sur ce serveur Jenkins.
 - Manage Credentials** (Configure credentials): Configure credentials
 - Configure Credential Providers** (Configure the credential providers and types): Configure the credential providers and types
- Status Information**
 - Informations sur le système** (System information): Affiche diverses informations relatives au système pour aider à la résolution de problèmes.
 - Logs systèmes** (System logs): Le log système capture la sortie java.util.logging relative à Jenkins.
 - Statistiques d'utilisation** (Usage statistics): Vérifiez l'utilisation des ressources et décidez si vous avez besoin d'ordinateurs supplémentaires pour vos builds.

Address bar: 54.174.144.82:8080/credentials

F- Paramétrage de l'agent Jenkins

Ce nouvel agent s'occupera exclusivement de tout la partie consistant à construire l'image docker, la lancer, la scanner, la tester et la pousser sur DockerHub. Une fois ces tâches effectuées l'agent effacera toutes les données qui lui ont été passées pour être de nouveau fraîchement disponible pour un futur build.

1- Commande de l'installation de java

```
sudo apt-get -y update
sudo apt-get -y install openjdk-11-jre-headless
sudo apt-get -y install default-jre
java -version
```

Résultat:

```
openjdk version "11.0.13" 2021-10-19
OpenJDK Runtime Environment (build 11.0.13+8-Ubuntu-0ubuntu1.20.04)
OpenJDK 64-Bit Server VM (build 11.0.13+8-Ubuntu-0ubuntu1.20.04, mixed mode)
```

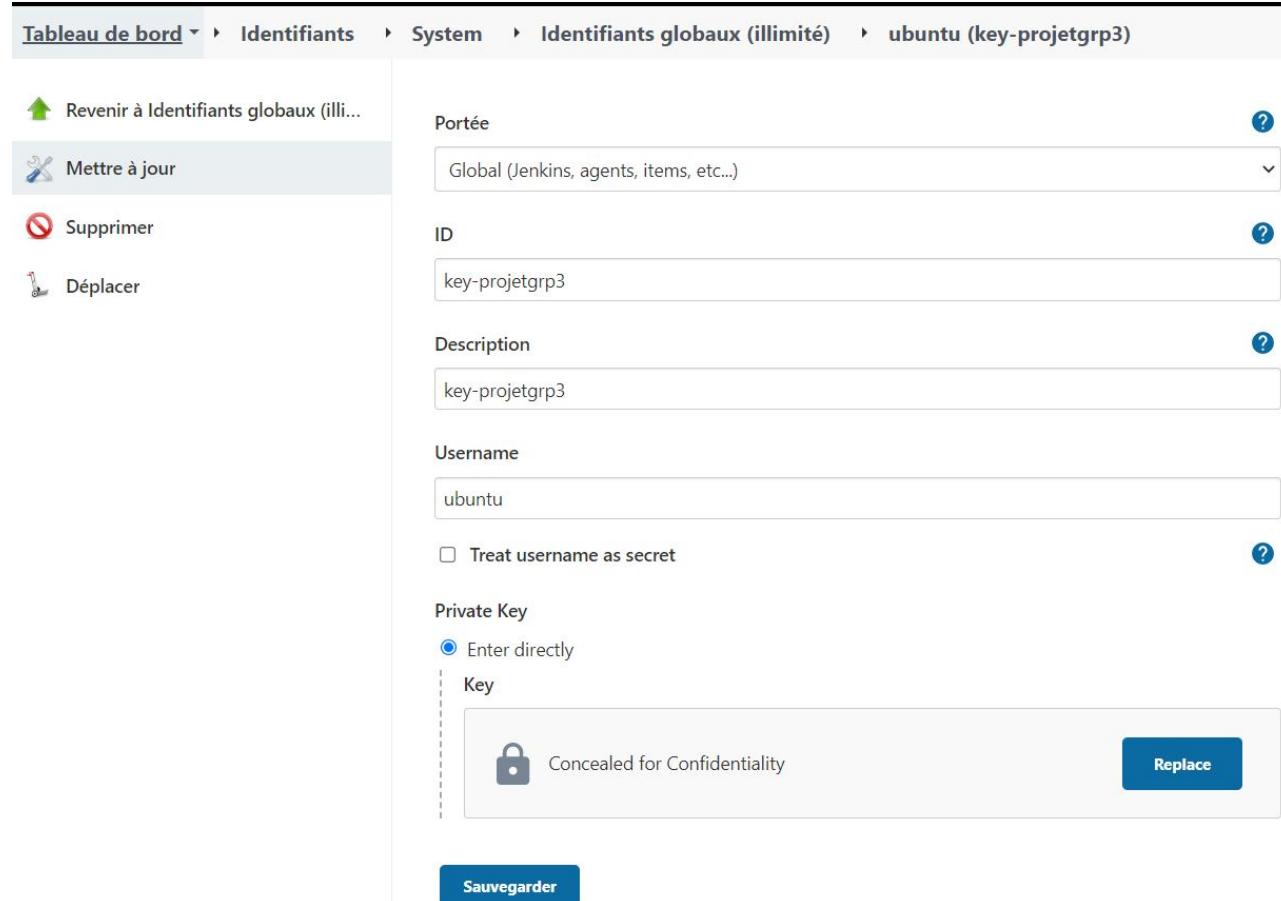
2- Ajout credential dans Jenkins

Sur jenkins, nous ajoutons la clé privée qui permet au serveur Jenkins de se connecter en ssh à l'instance créée pour l'agent Jenkins.

- Etapes:

Administrer Jenkins -> Manage Credentials -> Global -> Ajouter des credentials **->**Type "SSH Username with private key" et on colle dans "Private Key" la clé privée

- Résultat:



The screenshot shows the Jenkins Global Identifiers configuration page. The URL in the browser is `Tableau de bord > Identifiants > System > Identifiants globaux (illimité) > ubuntu (key-projetgrp3)`. The page displays the following fields:

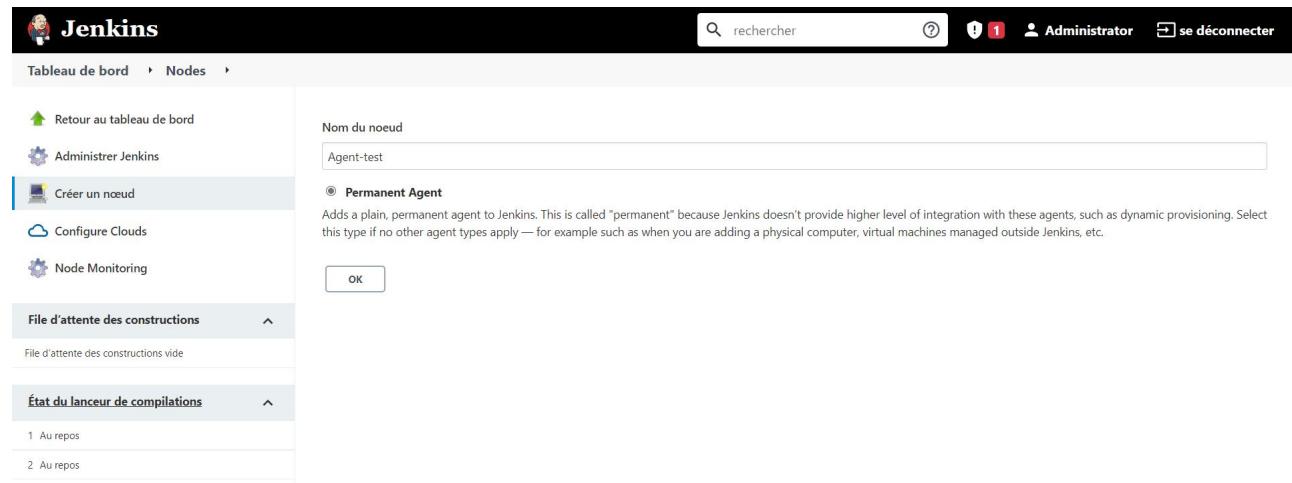
- Portée:** Global (Jenkins, agents, items, etc...)
- ID:** key-projetgrp3
- Description:** key-projetgrp3
- Username:** ubuntu
- Treat username as secret:** (checkbox)
- Private Key:**
 - Enter directly (radio button selected)
 - Key: Concealed for Confidentiality (with a **Replace** button)

At the bottom is a **Sauvegarder** (Save) button.

3- Paramétrage de l'agent dans Jenkins

Commençons par créer **Agent-test** sur l'interface de Jenkins :

- Administrer Jenkins -> Gérer les noeuds -> Créer un noeud:



The screenshot shows the Jenkins 'Create Node' configuration page. The URL in the browser is `Tableau de bord > Nodes > Créer un noeud`. The page displays the following fields:

- Nom du noeud:** Agent-test
- Type:** Permanent Agent (radio button selected)
- OK** button

On the left sidebar, the 'Nodes' section is selected. Other sections visible include 'Tableau de bord', 'Administrer Jenkins', 'Créer un noeud', 'Configure Clouds', and 'Node Monitoring'.

- Nous lui donnons le nom de **Agent-test** et nous le réservons pour les jobs qui lui sont associés. On se connecte sur celui-ci en SSH en lui précisant l'hôte et les credentials.

4- Utilisation de l'agent dans le pipeline

Il faudra maintenant faire appel à cet agent dans le pipeline là où on le souhaite de la manière suivante:

```
stage ('<Nom du stage>') {
    agent {label 'Agent-test'}
    steps{
        ...
    }
}
```

G- Crédit du Backend

C'est une bonne pratique de traiter notre fichier d'état comme un secret et de le stocker à distance.

Nous voulons nous assurer que notre option backend verrouille notre état lorsque quelqu'un met en service des ressources. Si deux membres de l'équipe tentent d'apporter des modifications en même temps, le fichier d'état peut être corrompu. Nous disons cela parce que bien que la plupart des options backend le fassent automatiquement, S3 ne le fait pas. Pour ce projet, nous ne l'avons pas activé le verrouillage d'état car nous travaillons sur le projet en équipe et il n'y avait aucun risque que mes modifications entrent en collision avec quelqu'un d'autre.

- on creer un backend de compartiment S3 sur aws



Amazon S3

Compartiments (94) [Info](#)

Les compartiments sont des conteneurs pour les données stockées dans S3. [En savoir plus](#)

[Créer un compartiment](#)

Rechercher des compartiments par nom

Nom	Région AWS	Accéder	Date de création
Compartment 1	Region 1	Access 1	Creation Date 1
Compartment 2	Region 2	Access 2	Creation Date 2
Compartment 3	Region 3	Access 3	Creation Date 3

The screenshot shows the AWS S3 'Create a compartment' configuration page. At the top, the AWS logo is on the left, followed by a 'Services' button and a search bar with the placeholder 'Rechercher des services, des fonctions, des blogs, des documents et [Option+S]'. To the right of the search bar are icons for notifications, help, and user 'renaud @ 5320-0049-9530'. Below the search bar, a breadcrumb navigation shows 'Amazon S3 > Créer un compartiment'. The main title 'Créer un compartiment' has an 'Info' link. A sub-instruction 'Les compartiments sont des conteneurs pour les données stockées dans S3. En savoir plus' is shown with a link icon. The configuration section is titled 'Configuration générale'. It contains a 'Nom du compartiment' field with the value 'projetgrp3-terraform-backend' and a note that the name must be unique and cannot contain spaces or uppercase letters, with a link to 'l'attribution de noms de compartiment'. A 'Région AWS' dropdown is set to 'USA Est (Virginie du Nord) us-east-1'. At the bottom, there is a note about copying parameters from an existing compartment (marked as optional) and a 'Sélectionner un compartiment' button.

aws Services Rechercher des services, des fonctions, des blogs, des documents et [Option+S] Global ▾ renaud @ 5320-0049-9530 ▾

Copier les paramètres depuis un compartiment existant - **facultatif**
Seuls les paramètres de compartiment dans la configuration suivante sont copiés.

Sélectionner un compartiment

Propriété d'objets Info

Contrôlez la propriété des objets écrits dans ce compartiment à partir d'autres comptes AWS et accordés à l'aide de listes de contrôle d'accès (ACL). La propriété des objets détermine qui peut spécifier l'accès aux objets.

Listes ACL désactivées (recommandé)
Tous les objets de ce compartiment sont gérés par ce compte. L'accès à ce compartiment et à ses objets est spécifié en utilisant uniquement des politiques.

Listes ACL activées
Les objets de ce compartiment peuvent être gérés par d'autres comptes AWS. L'accès à ce compartiment et à ses objets peut être spécifié à l'aide des listes ACL.

Propriété d'objets

Propriétaire du compartiment appliquée

Paramètres de blocage de l'accès public pour ce compartiment

L'accès public aux compartiments et aux objets est accordé via des listes de contrôle d'accès (ACL), des stratégies de compartiment, de point d'accès ou tous ces éléments à la fois. Pour bloquer l'accès public à votre compartiment et aux objets qu'il contient, activez le paramètre Bloquer tous les accès publics. Il s'applique uniquement à ce compartiment et à ses points d'accès. AWS recommande de bloquer tous les accès publics, mais avant d'appliquer ces paramètres, vérifiez que vos applications fonctionneront correctement sans accès public. Si vous souhaitez autoriser un certain niveau d'accès public pour votre compartiment ou ses objets, vous pouvez personnaliser les paramètres individuels ci-dessous en fonction de vos besoins en stockage. [En savoir plus](#)

Amazon S3

Compartiments

- Points d'accès
- Points d'accès de l'objet Lambda
- Points d'accès multi-région
- Opérations par lot
- Analyseur d'accès pour S3

Paramètres de blocage de l'accès public pour ce compte

Storage Lens

- Tableaux de bord
- Paramètres AWS Organizations

Fonctionnalité spot (3)

AWS Marketplace pour S3

Paramètres de blocage de l'accès public pour ce compartiment

L'accès public aux compartiments et aux objets est accordé via des listes de contrôle d'accès (ACL), des stratégies de compartiment, de point d'accès ou tous ces éléments à la fois. Pour bloquer l'accès public à votre compartiment et aux objets qu'il contient, activez le paramètre Bloquer tous les accès publics. Il s'applique uniquement à ce compartiment et à ses points d'accès. AWS recommande de bloquer tous les accès publics, mais avant d'appliquer ces paramètres, vérifiez que vos applications fonctionneront correctement sans accès public. Si vous souhaitez autoriser un certain niveau d'accès public pour votre compartiment ou ses objets, vous pouvez personnaliser les paramètres individuels ci-dessous en fonction de vos besoins en stockage. [En savoir plus](#)

Bloquer tous les accès publics

L'activation de ce paramètre revient à activer les quatre paramètres ci-dessous. Chacun des paramètres suivants est indépendant l'un de l'autre.

- Bloquer l'accès public aux compartiments et aux objets, accordé via de nouvelles listes de contrôle d'accès (ACL)**
- Bloquer l'accès public aux compartiments et aux objets, accordé via n'importe quelles listes de contrôle d'accès (ACL)**
- Bloquer l'accès public aux compartiments et aux objets, accordé via de nouvelles stratégies de compartiment ou de point d'accès public**
- Bloquer l'accès public et entre comptes aux compartiments et objets via n'importe quelles stratégies de compartiment ou de point d'accès public**

Amazon S3

Compartiments

- Points d'accès
- Points d'accès de l'objet Lambda
- Points d'accès multi-région
- Opérations par lot
- Analyseur d'accès pour S3

Paramètres de blocage de l'accès public pour ce compte

Storage Lens

- Tableaux de bord
- Paramètres AWS Organizations

Fonctionnalité spot (3)

AWS Marketplace pour S3

Gestion des versions de compartiment

La gestion des versions est un moyen de conserver plusieurs variantes d'un objet dans le même compartiment. Vous pouvez utiliser la gestion des versions pour conserver, récupérer et restaurer chaque version de chaque objet stocké dans votre compartiment Amazon S3. Grâce à la gestion des versions, vous pouvez aisément récupérer en cas d'actions involontaires des utilisateurs et de défaillances des applications. [En savoir plus](#)

Gestion des versions de compartiment

- Désactiver
- Activer

Balises (2) - facultatif

Suivez les coûts de stockage ou d'autres critères en balisant votre compartiment. [En savoir plus](#)

Clé	Valeur - facultatif	Supprimer
Name	projetgrp4-terraform-backend	Supprimer
formation	Frazr	Supprimer

Ajouter une balise

Amazon S3

Compartiments

- Points d'accès
- Points d'accès de l'objet Lambda
- Points d'accès multi-région
- Opérations par lot
- Analyseur d'accès pour S3

Paramètres de blocage de l'accès public pour ce compte

Storage Lens

- Tableaux de bord
- Paramètres AWS Organizations

Fonctionnalité spot (3)

AWS Marketplace pour S3

Compartiment «projetgrp3-terraform-backend » créé avec succès

Pour charger des fichiers et dossiers ou configurer des paramètres de compartiment supplémentaires, sélectionnez **Afficher les détails**.

Instantané de compte

Dernière mise à jour : 11 Jan y par Storage Lens. Les métriques sont générées toutes les 24 heures. [En savoir plus](#)

Compartiments (95) [Info](#)

Les compartiments sont des conteneurs pour les données stockées dans S3. [En savoir plus](#)

		Vider	Supprimer	Créer un compartiment
<input type="text" value="grp3"/>				
1 correspondance < 1 >				
Nom	Région AWS	Accéder	Date de création	
projetgrp3-terraform-backend	USA Est (Virginie du Nord) us-east-1	Compartiment et objets non publics	12 Jan 2022 02:54:21 PM CET	

H- Création du Pipeline

1- Création du pipeline dans Jenkins

Tableau de bord > Tous >

Saisissez un nom

projetajc-group3
» Champ obligatoire

Construire un projet free-style
Ceci est la fonction principale de Jenkins qui sert à builder (construire) votre projet. Vous pouvez intégrer tous les outils de gestion de version avec tous les systèmes de build. Il est même possible d'utiliser Jenkins pour tout autre chose qu'un build logiciel.

Construire un projet maven
Construit un projet avec maven. Jenkins utilise directement vos fichiers POM et diminue radicalement l'effort de configuration. Cette fonctionnalité est encore en bêta mais elle est disponible afin d'obtenir vos retours.

Pipeline
Organise des activités de longue durée qui peuvent s'étendre sur plusieurs agents de construction. Adapté pour la création des pipelines (anciennement connues comme workflows) et/ou pour organiser des activités complexes qui ne s'adaptent pas facilement à des tâches de type libre.

Construire un projet multi-configuration
Adapté aux projets qui nécessitent un grand nombre de configurations différentes, comme des environnements de test multiples, des binaires spécifiques à une plateforme, etc.

OK

Tableau de bord > projetajc-group3 >

General Build Triggers Advanced Project Options Pipeline

Description

[Plain text] **Prévisualisation**

Ce build a des paramètres **?**
 Do not allow concurrent builds **?**
 Do not allow the pipeline to resume if the controller restarts **?**
 GitHub project **?**

Project url

https://github.com/projetajc-group3/projetajc_node.git/ **Avancé...**

Pipeline speed/durability override **?**
 Preserve stashes from completed builds **?**
 Supprimer les anciens builds **?**
 Throttle builds **?**

Sauver **Apply**

Pipeline

Definition

Pipeline script from SCM

SCM

Git

Repositories

Repository URL

https://github.com/projetajc-group3/projetajc_node.git

Credentials

- aucun -

Avancé...

Add Repository

Branches to build

Branch Specifier (blank for 'any')

*/master

2- Création du Jenkinsfile

```

def EC2_PRODUCTION_HOST = "@IP"
def EC2_STAGING_HOST = "@IP"

pipeline {
    environment {
        IMAGE_NAME = "node"
        IMAGE_TAG = "1.0"
        USERNAME = "projetajcgroup3"
        CONTAINER_NAME = "webapp"
        CONTAINER_PORT = "3000"
        TEST_EXTERNAL_PORT = "30001"
        STAGING_EXTERNAL_PORT = "30001"
        PROD_EXTERNAL_PORT = "30000"
        URL_GIT_NODE ="https://github.com/projetajc-group3/projetajc_node.git"
        URL_GIT_TERRAFORM ="https://github.com/projetajc-
group3/terraform_node.git"
        URL_GIT_DEPLOY_DOCKER = "https://github.com/projetajc-
group3/docker_role_deploy.git"
        URL_GIT_DEPLOY_KUBERNETES = "https://github.com/projetajc-
group3/kubernetes_role_deploy.git"
    }
    agent none
    stages {

```

```
    }  
}
```

III- Implémentation du Pipeline Jenkins

A- Build de l'image

1- Installation de Docker sur le serveur Jenkins

Voici les commandes a exécuter:

```
#Install docker  
curl -fsSL https://get.docker.com -o get-docker.sh  
sh get-docker.sh  
sudo usermod -aG docker ubuntu  
sudo usermod -aG docker jenkins
```

2- Crédation du Dockerfile

- Nous créons un repository projetjc_node sur lequel nous stockons l'application à déployer (fork):
https://github.com/projetjc-group3/projetjc_node.git
- Nous ajoutons à la racine de ce repository le Dockerfile, qui va builder l'image, que nous avons écrit comme suit:

```
FROM node:14.16  
WORKDIR /usr/src/app  
# Installation des dépendance  
# On s'assure que les packages package.json,package-lock.json sont copiés  
COPY package*.json ./  
RUN npm install  
COPY . .  
EXPOSE 3000  
CMD [ "node", "./bin/www" ]
```

3- Build de l'image

Maintenant que nous avons créé notre Dockerfile, construisons notre image. Pour ce faire, nous utilisons la commande docker build qui crée des images Docker à partir d'un Dockerfile. Ce build s'execute sur l'Agent-test. Nous prenons quand même soin de supprimer les conteneurs et images qui auraient pu rester depuis le dernier build avant de lancer le build.

Voici le stage a rajouter dans le Jenkinsfile:

```

stage ('Image Build (TEST)') {
    agent {label 'Agent-test'}
    steps{
        script{
            sh'''
            rm -rf projetajc_node/ || true
            docker stop $CONTAINER_NAME || true
            docker rm $CONTAINER_NAME || true
            docker rmi $USERNAME/$IMAGE_NAME:$IMAGE_TAG || true
            git clone $URL_GIT_NODE || true
            cd projetajc_node
            docker build -t $USERNAME/$IMAGE_NAME:$IMAGE_TAG .
            '''
        }
    }
}

```

B- Test de l'application

1- Run de l'image

Nous lançons ensuite le conteneur à l'aide des commandes docker afin de pourvoir tester notre image fraîchement créée.

Voici le stage qui effectue cette étape pour le Jenkinsfile:

```

stage ('Run build (TEST)') {
    agent {label 'Agent-test'}
    steps{
        script{
            sh'''
            docker run -d -p $TEST_EXTERNAL_PORT:$CONTAINER_PORT --name
$CONTAINER_NAME $USERNAME/$IMAGE_NAME:$IMAGE_TAG
            '''
        }
    }
}

```

2- Test de l'image

Nous avons conteneurisé notre application, nous avons construit l'image, et nous l'avons lancé. L'étape suivante est simplement de tester si l'application est bien déployée sur l'Agent-test.

Nous savons que notre application est un site internet écrit en Javascript. A l'intérieur du fichier `/views/index.ejs`, nous avons le titre du site qui est **Devops Foundation**.

Le test le plus simple est de faire un stage dans le pipeline qui fera un `curl` sur le titre du site. Stage qui effectue cette étape pour le Jenkinsfile:

```
stage ('Test curl (TEST)') {
    agent {label 'Agent-test'}
    steps{
        script{
            sh '''
                curl http://localhost:$TEST_EXTERNAL_PORT/ | tac | tac |
                grep -iq "DevOps Foundation"
            '''
        }
    }
}
```

Si le curl réussi alors c'est que le site internet a bien été déployé.

Note: nous faisons ici deux `tac` en commandes pour laisser le temps au curl de se finir.

C- Scan du code

1- Scan de sécurité et de vulnérabilité de l'application

Avant de pousser notre application en pré-prod, nous lançons un scan du code applicatif à la recherche d'éventuelles vulnérabilités. Cela dans le but de renforcer la sécurité. Pour ça, nous utilisons **Snyk**, qui à l'énorme avantage d'avoir un module sur Jenkins pour une meilleure intégration de l'outil directement au pipeline.

2- Installation de Snyk

Pour l'installation rien de plus simple. Snyk possède un module directement sur Jenkins avec toute la documentation à suivre : <https://plugins.jenkins.io/snyk-security-scanner/>.

- Administrer Jenkins -> Gestion des plugins -> Disponibles -> Recherche de "Snyk" -> Installer
- Administrer Jenkins -> Configuration globale des outils -> Installations Snyk

Tableau de bord > Configuration globale des outils

Liste des installations SonarScanner pour MSBuild sur ce système

SonarQube Scanner

Installations SonarQube Scanner

[Ajouter SonarQube Scanner](#)

Liste des installations SonarQube Scanner sur ce système

Ant

Installations Ant

[Ajouter Ant](#)

Liste des installations Ant sur ce système

Maven

Installations Maven

[Ajouter Maven](#)

Liste des installations Maven sur ce système

Snyk

Installations Snyk...

[Enregistrer](#) [Appliquer](#)

- On ajoute le nom de notre installation, nous l'appelons **snyk@latest**, on laisse les autres options par défaut.

Tableau de bord > Configuration globale des outils

Snyk

Installations Snyk

[Ajouter Snyk](#)

Snyk

Nom

snyk@latest

Install automatically

Install from snyk.io

Version

latest

Update policy interval (hours)

24

[Supprimer un installateur](#)

[Ajouter un installateur](#)

[Supprimer Snyk](#)

Ajouter Snyk

Liste des installations Snyk sur ce système

[Enregistrer](#) [Appliquer](#)

* Création d'un compte utilisateur sur Snyk, nous prenons soin de copier la clé.

General

Auth Token

Use this token to authenticate the Snyk CLI and in CI/CD pipelines. Learn more about authenticating CLI in our docs.

KEY CREATED

click to show 13 January 2022, 10:37:55

Revoke & Regenerate

Authorized Applications

List of applications you have authorized

No applications

Preferred Organization

Choose which organization you are taken to when logging into the site.

- Administrer Jenkins -> Manage Credentials -> Global -> Ajouter des identifiants -> Type : Snyk API token
- On copie ici notre clé et on lui donne un nom : **snyk-token**

rechercher

Tableau de bord Identifiants System Identifiants globaux (illimité)

Type

Snyk API token

Portée

Global (Jenkins, agents, items, etc..)

Token

.....

ID

snyktoken

Description

Snyk API token

OK

REST API Jenkins 2.319.1

L'installation et la configuration sont terminées.

3- Utilisation de Snyk dans le pipeline

Pour lancer un test sur le code du projet, nous avons un stage dans le pipeline, que nous lançons sur l'Agent-test. Nous devons fournir à Snyk le nom de notre installation Snyk et le nom du token Snyk. Nous précisons aussi le **targetFile** qui pointe sur le manifest du projet. Dans notre cas, c'est le fichier `package.json`.

```

stage('Scan code (TEST)') {
    agent { label 'Agent-test'}
    steps {
        echo 'Testing...'
        snykSecurity(
            snykInstallation: 'snyk@latest',
            snykTokenId: 'snyk-token',
            targetFile: 'projetajc_node/package.json',
        )
    }
}

```

D- Push de l'image vers DockerHub

1- Créditration credential pour Dockerhub

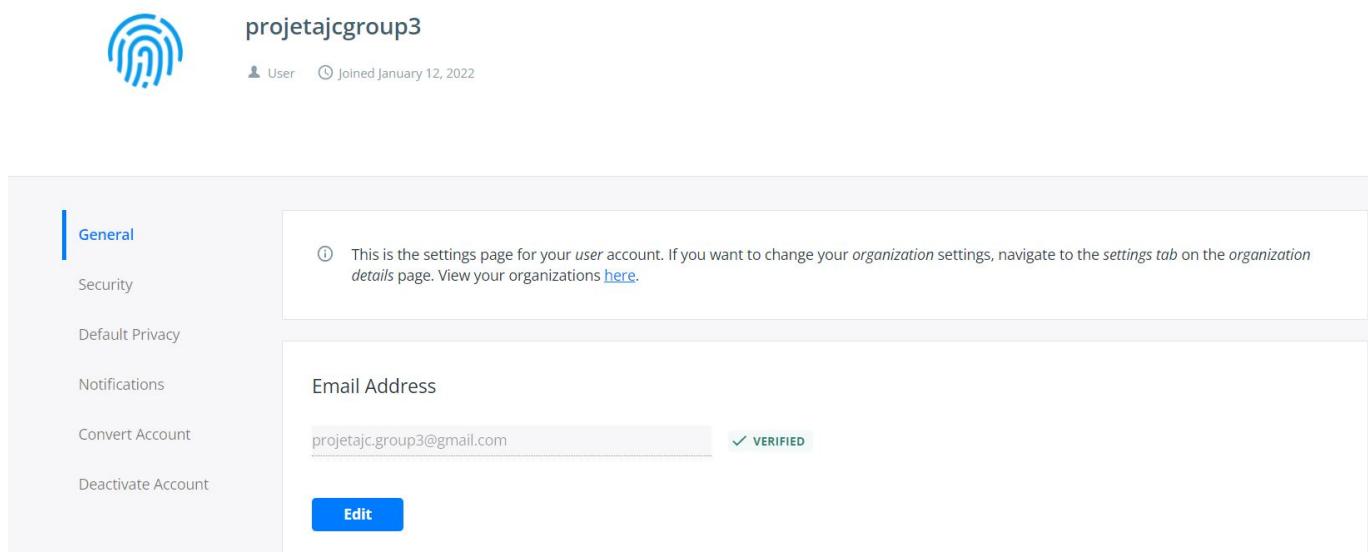
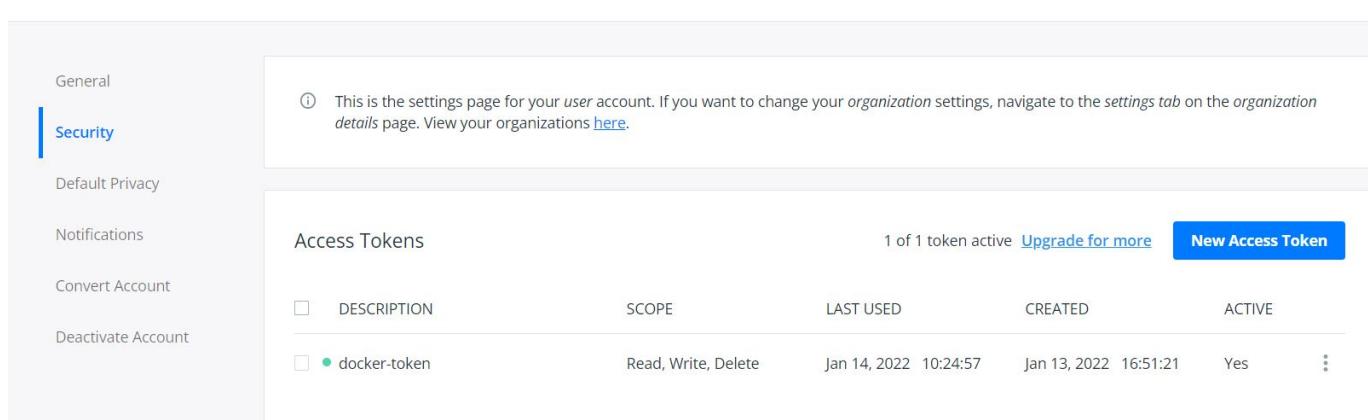
Après le build, nous comptons sauvegarder notre artefact sur le repository Dockerhub. Nous devons alors communiquer au serveur Jenkins les credentials nécessaires pour s'y connecter et pousser l'artefact créé sur l'Agent-test.

- Sur Dockerhub:

Account Settings -> Security -> New Access Token.

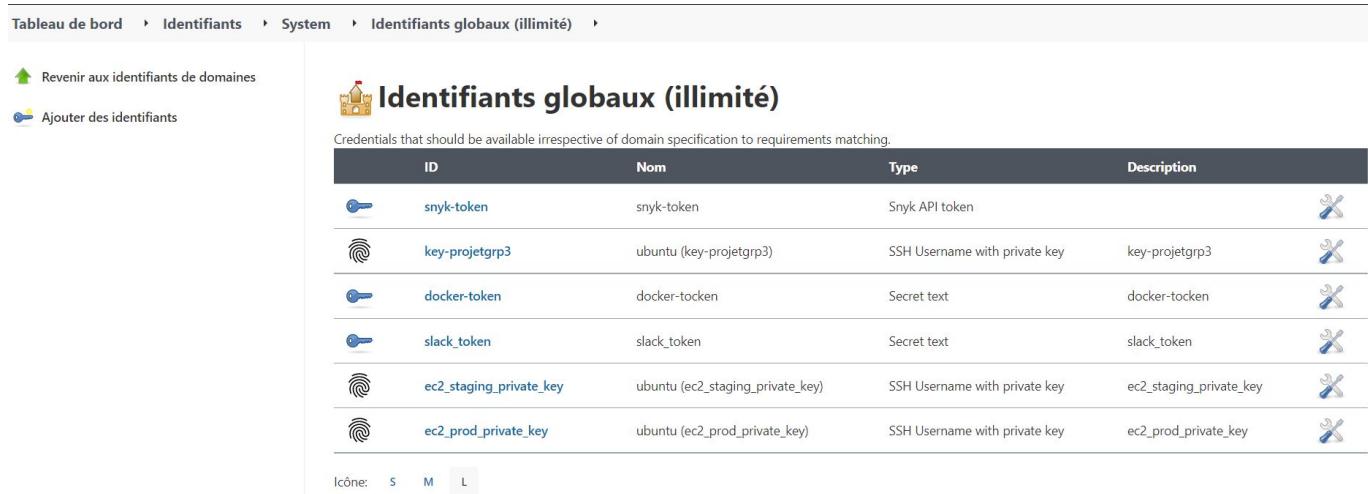
Nous prenons soin de copier la clé.

The screenshot shows the DockerHub website. At the top, there is a search bar with 'Search for great content (e.g., mysql)' and a dropdown menu. Below the search bar, there are links for 'Explore', 'Repositories', 'Organizations', and 'Help'. An 'Upgrade' button is also visible. On the right, there is a user profile for 'projetajcgroup3' with a dropdown menu showing options like 'What's New', 'My Profile', 'Content Subscriptions', 'Account Settings', 'Billing', and 'Log out'. The main content area shows a repository named 'projetajcgroup3 / node'. The repository details are: 'Not Scanned', 0 stars, 13 downloads, and 'Public'. A tip message at the bottom of the repository card says: 'Tip: Not finding your repository? Try switching namespace via the top left dropdown.'

- Sur Jenkins:

Administrer Jenkins -> Manage Credentials -> Global -> Ajouter des credentials -> Type "Secret text" et on colle dans "secret" le token Docker que nous avons créé sur Docker-hub.



ID	Nom	Type	Description
 snyk-token	snyk-token	Snyk API token	
 key-projetgrp3	ubuntu (key-projetgrp3)	SSH Username with private key	key-projetgrp3 
 docker-token	docker-tocken	Secret text	docker-tocken 
 slack_token	slack_token	Secret text	slack_token 
 ec2_staging_private_key	ubuntu (ec2_staging_private_key)	SSH Username with private key	ec2_staging_private_key 
 ec2_prod_private_key	ubuntu (ec2_prod_private_key)	SSH Username with private key	ec2_prod_private_key 

Tableau de bord > Identifiants > System > Identifiants globaux (illimité) >

Revenir aux identifiants de domaines

Ajouter des identifiants

Type: Secret text

Portée: Global (Jenkins, agents, items, etc...)

Secret:

ID: docker-token

This ID is already in use

Description: docker-token

OK

2- Docker Push

Sur le pipeline Jenkins nous allons créer un stage qui s'occupe de pousser sur Dockerhub l'image que nous venons de construire et de nettoyer notre **Agent-test**.

Il s'agit tout d'abord de se connecter sur le Dockerhub à l'aide de notre username et du credential tout juste créé et de pousser ensuite l'image.

Enfin on stop et supprime le conteneurs et l'image utilisée.

```
stage ('Clean test environment and save artifact (TEST)') {
    agent {label 'Agent-test'}
    environment{
        PASSWORD = credentials('docker-token')
    }
    steps {
        script{
            sh '''
                docker login -u $USERNAME -p $PASSWORD
                docker push $USERNAME/$IMAGE_NAME:$IMAGE_TAG
                docker stop $CONTAINER_NAME || true
                docker rm $CONTAINER_NAME || true
                docker rmi $USERNAME/$IMAGE_NAME:$IMAGE_TAG
            '''
        }
    }
}
```

E- Création des machines Preproduction et Production avec Terraform

1- Installation de Terraform sur Jenkin master

Terraform est l'offre d'infrastructure en tant que code de HashiCorp. C'est un outil pour construire, modifier et gérer l'infrastructure de manière sûre et reproductible.

Nous allons donc nous en servir pour déployer de manière automatique sur AWS nos serveurs de préproduction et de production ainsi que pour la gestion de notre backend sur S3

Voilà les commandes que nous exécutons sur le serveur Jenkins master:

```
#Install TERRAFORM
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com
$(lsb_release -cs) main"
```

2- Repository Terraform

Voici l'arborescence de notre repository terraform (https://github.com/projetajc-group3/terraform_node.git):

```
.
├── README.md
└── modules
    ├── ec2
    │   ├── main.tf
    │   ├── output.tf
    │   └── variables.tf
    ├── sg_preprod
    │   ├── main.tf
    │   ├── output.tf
    │   └── variables.tf
    └── sg_prod
        ├── main.tf
        ├── output.tf
        └── variables.tf
└── preprod
    ├── backend.tf
    ├── data.tf
    ├── main.tf
    ├── provider.tf
    ├── terraform.tfvars
    └── variable.tf
└── prod
    ├── backend.tf
    ├── data.tf
    ├── main.tf
    └── provider.tf
```

```

└── terraform.tfvars
└── variable.tf

```

3- Les modules

- sg_preprod

Ce module va permettre la création du security group sur AWS afin de gérer l'ouverture des ports nécessaire au bon fonctionnement de notre application en préproduction.

variable.tf

Ici nous avons les variables en entrée afin de gérer les tag de la SG

```

variable "sg_author" {
  type = string
  default = "projetgrp3"
}

variable "sg_env" {
  type = string
  default = "dev"
}

```

output.tf

Le fichier de sortie nous permet de renvoyer l'id de la SG afin de l'associer à l'ec2 lors de sa création

```

output "sg_id" {
  value = aws_security_group.mysg.id
}

```

main.tf

La création de la SG se fait donc ici avec ingress qui permet l'ouverture du port 30001 (port d'accès à l'application) et du port 22 (permettant les connexions ssh). La partie egress ici ouvre l'accès au réseau extérieur pour l'ec2.

```

resource "aws_security_group" "mysg" {
  name = "${var.sg_author}-sg-${var.sg_env}"
}

```

```

ingress {
  description      = "NODE_PORT"
  from_port       = 30001
  to_port         = 30001
  protocol        = "tcp"
  cidr_blocks     = ["0.0.0.0/0"]
  ipv6_cidr_blocks = [":/:0"]
}

ingress {
  description      = "SSH"
  from_port       = 22
  to_port         = 22
  protocol        = "tcp"
  cidr_blocks     = ["0.0.0.0/0"]
  ipv6_cidr_blocks = [":/:0"]
}

egress {
  from_port = 0
  to_port   = 0
  protocol  = "-1"
  cidr_blocks = ["0.0.0.0/0"]
  ipv6_cidr_blocks = [":/:0"]
}

tags = {
  Name = "${var.sg_author}-sg-${var.sg_env}"
  formation = "Frazer"
}
}

```

- sg_prod

Ce module comme le précédent va permettre la création du security group sur AWS afin de gérer l'ouverture des ports nécessaire au bon fonctionnement de notre application mais cette fois ci pour la production.

variable.tf

Ici nous avons les variables en entrée afin de gérer les tag de la SG

```

variable "sg_author" {
  type = string
  default = "projetgrp3"
}

variable "sg_env" {
  type = string
}

```

```
    default = "dev"
}
```

output.tf

Le fichier de sortie nous permet de renvoyer l'id de la SG afin de l'associer à l'ec2 lors de sa création

```
output "sg_id" {
  value = aws_security_group.mysg.id
}
```

main.tf

La création de la SG pour la production avec cette fois ci l'ouverture du port 30000 (port d'accès à l'application) et du port 22 (permettant les connexions ssh). La partie egress ici ouvre aussi l'accès au réseau extérieur pour l'ec2.

```
resource "aws_security_group" "mysg" {
  name = "${var.sg_author}-sg-${var.sg_env}"

  ingress {
    description      = "NODE_PORT"
    from_port        = 30000
    to_port          = 30000
    protocol         = "tcp"
    cidr_blocks     = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }

  ingress {
    description      = "SSH"
    from_port        = 22
    to_port          = 22
    protocol         = "tcp"
    cidr_blocks     = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }

  egress {
    from_port = 0
    to_port = 0
    protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }

  tags = {
    Name = "${var.sg_author}-sg-${var.sg_env}"
    formation = "Frazer"
  }
}
```

```
    }
}
```

- ec2

variables.tf

Liste des variables permettant la création de l'ec2 sur AWS, nous avons fait en sorte que l'ensemble de l'ec2 soit paramétrable.

```
#Name of the creator
variable "ec2_author" {
    type = string
    default = "projetgrp3"
}

#Environment of the instance
variable "ec2_env" {
    type = string
    default = "dev"
}

#Ami for the instance
variable "ec2_ami" {
    type = string
    default = "ami-04505e74c0741db8d"
}

#Type of the instance
variable "ec2_instance_type" {
    type = string
    default = "t2.micro"
}

#Name of the ssh key
variable "ec2_key_name" {
    type = string
    default = "key-projetgrp3"
}

#Size of the root volume
variable "ec2_vol_size" {
    type = number
    default = 8
}

#Id of the security group to associate
variable "sg_id" {
    type = string
}
```

```
    default = """
}
```

output.tf

Nous avons deux sorties suite à la création de l'ec2, son id et surtout son adresse ip public qui nous permettra de pouvoir y accéder en ssh.

```
#Return is of the instance
output "ec2_id" {
    value = aws_instance.myec2.id
}

#Return Public IP
output "ec2_public_ip" {
    value = aws_instance.myec2.public_ip
}
```

main.tf

Création de l'ec2 avec tous les paramètres en entrée afin de la rendre dynamique en fonction de l'environement.

```
resource "aws_instance" "myec2" {
    ami           = "${var.ec2_ami}"
    instance_type = "${var.ec2_instance_type}"
    key_name      = "${var.ec2_key_name}"
    tags = {
        Name      = "${var.ec2_author}-ec2-${var.ec2_env}"
        formation = "Frazer"
        iac       = "terraform"
    }

    root_block_device {
        delete_on_termination = true
        volume_size = var.ec2_vol_size
    }

    vpc_security_group_ids = [var.sg_id]
}
```

4- Les environnement

- preprod

provider.tf

Ici nous avons les informations permettant la connexion à environnement AWS. Nous utilisons un fichier credentials afin d'éviter d'avoir les access key visibles.

```
provider "aws" {
  region           = "us-east-1"
  shared_credentials_file = "/var/lib/jenkins/workspace/credentials"
}
```

variables.tf

Les variables nécessaires à la création de la SG et de l'ec2 sont ici définies.

```
variable "author" {
  default = "projetgrp3"
}

variable "env" {
  default = "dev"
}

variable "key_name" {
  default = "key-projetgrp3"
}

variable "instance_type" {
  default = "t2.nano"
}

variable "vol_size" {
  default = 8
}
```

data.tf

Nous effectuons la récupération de l'id de l'AMI afin de la fournir à l'ec2 dans cette partie. Ici nous recherchons l'image la plus récente de Ubuntu focal.

```
data "aws_ami" "myami" {
  most_recent = true
  owners      = ["099720109477"]
  filter {
    name    = "name"
    values  = ["ubuntu/images/hvm-ssd/ubuntu-focal*"]
  }
}
```

backend.tf

Cette partie nous permet de stocker l'état sous la forme d'une clé donnée dans un compartiment Amazon S3. Ce backend prend également en charge le verrouillage d'état et la vérification de la cohérence.

```
terraform {
  backend "s3" {
    bucket = "projetgrp3-terraform-backend"
    key    = "projetgrp3-backend-preprod.tfstate"
    region          = "us-east-1"
    shared_credentials_file = "/var/lib/jenkins/workspace/credentials"
  }
}
```

main.tf

Lancement de la création de la SG(module sg_preprod) puis de l'ec2 (module ec2) de preproduction avec tous les paramètres adéquates. Nous publions aussi ici en sortie l'ip public de l'ec2 afin de pouvoir la récupérer dans le pipeline de Jenkins.

```
module "sg_creation" {
  source = "../modules/sg_preprod"
  sg_author = var.author
  sg_env = var.env
}

module "ec2_creation" {
  source = "../modules/ec2"
  ec2_instance_type = var.instance_type
  ec2_ami = data.aws_ami.myami.id
  ec2_key_name = var.key_name
  ec2_author = var.author
  ec2_env = var.env
  ec2_vol_size = var.vol_size
  sg_id = module.sg_creation.sg_id
}
```

```

}

output "ec2_ip" {
    value = module.ec2_creation.ec2_public_ip
}

```

terraform.tfvars

Simple fichier regroupant les principaux paramètres que l'on peut maintenant surcharger pour cet environement de preproduction

```

author = "projetgrp3"
env = "preprod"
key_name = "key-projetgrp3"
instance_type = "t2.medium"
vol_size = 15

```

- prod

provider.tf

Ici nous avons les informations permettant la connexion à environement AWS. Nous utilisons un fichier credentials afin d'éviter d'avoir les access key visibles.

```

provider "aws" {
    region           = "us-east-1"
    shared_credentials_file = "/var/lib/jenkins/workspace/credentials"
}

```

variables.tf

Les variables nécessaires à la creation de la SG et de l'ec2 sont ici définit.

```

variable "author" {
    default = "projetgrp3"
}

variable "env" {
    default = "dev"
}

```

```

variable "key_name" {
  default = "key-projetgrp3"
}

variable "instance_type" {
  default = "t2.nano"
}

variable "vol_size" {
  default = 8
}

```

data.tf

Nous effectuons la récupération de l'id de l'AMI afin de la fournir à l'ec2 dans cette partie. Ici nous recherchons l'image la plus récente de Ubuntu focal.

```

data "aws_ami" "myami" {
  most_recent = true
  owners       = ["099720109477"]
  filter {
    name    = "name"
    values  = ["ubuntu/images/hvm-ssd/ubuntu-focal*"]
  }
}

```

backend.tf

Cette partie nous permet de stocker l'état sous la forme d'une clé donnée dans un compartiment Amazon S3. Ce backend prend également en charge le verrouillage d'état et la vérification de la cohérence.

```

terraform {
  backend "s3" {
    bucket = "projetgrp3-terraform-backend"
    key    = "projetgrp3-backend-prod.tfstate"
    region           = "us-east-1"
    shared_credentials_file = "/var/lib/jenkins/workspace/credentials"
  }
}

```

main.tf

Lancement de la création de la SG (module sg_prod) puis de l'ec2 (module ec2) de production avec tous les paramètres adéquates. Nous publions aussi ici en sortie l'ip public de l'ec2 afin de pouvoir la récupérer dans le pipeline de Jenkins.

```
module "sg_creation" {
  source = "../modules/sg_prod"
  sg_author = var.author
  sg_env = var.env
}

module "ec2_creation" {
  source = "../modules/ec2"
  ec2_instance_type = var.instance_type
  ec2_ami = data.aws_ami.myami.id
  ec2_key_name = var.key_name
  ec2_author = var.author
  ec2_env = var.env
  ec2_vol_size = var.vol_size
  sg_id = module.sg_creation.sg_id
}

output "ec2_ip" {
  value = module.ec2_creation.ec2_public_ip
}
```

terraform.tfvars

Simple fichier regroupant les principaux paramètres que l'on peut maintenant surcharger pour cet environement de production

```
author = "projetgrp3"
env = "prod"
key_name = "key-projetgrp3"
instance_type = "t2.large"
vol_size = 20
```

5- Création du fichier credential pour AWS

Nous créons sur le serveur un fichier qui contient les credentials pour se connecter à AWS au niveau /var/lib/jenkins/workspace/ (path renseigné sur le repository Terraform au niveau provider)
Le fichier contient les informations suivantes:

```
[default]
aws_access_key_id = "XXXXXX"
aws_secret_access_key = "XXXXXXXXXXXXXX"
```

6- Crédation ec2 de Preproduction

Nous allons donc récupérer les repository Terraform que nous avons créer précédemment et nous placer dans le repertoire preprod qui contient l'ensemble des paramétrages pour cet environnement.

Après le lancement des commandes init et apply pour appliquer notre manifest, nous récupèrerons l'ip public de l'instance pour la placer dans un fichier afin de l'extraire en supprimant les " que génère la commande output de Terraform pour la placer en variable d'environnement globale de Jenkins

Voici le script a ajouter dans le Jenkins file:

```
stage ('Creation ec2 instance if necessary (STAGING)') {
    agent any
    steps {
        script{
            sh '''
                rm -rf src_terraform
                mkdir src_terraform
                cd src_terraform
                git clone $URL_GIT_TERRAFORM
                cd terraform_node/preprod
                terraform init -reconfigure
                terraform apply --auto-approve
                terraform output ec2_ip > ec2_ip.txt
                '''

                env.EC2_STAGING_HOST = sh( script: "sed -e 's/\"//g' ${src_terraform}/terraform_node/preprod/ec2_ip.txt",returnStdout: true).trim()
                sleep 20
            }
        }
    }
}
```

7- Crédation ec2 de Production

Nous effectuons les mêmes opération que pour la preproduction à la différence que nous nous plaçons dans le répertoire prod.

Nous ajoutons aussi un timeout afin d'attendre une validation manuelle pour le lancement en production. En effet cette étape viendra après le déploiement en preproduction que nous allons voir ensuite et ne pourra

être lancée uniquement après validation de l'environnement de preproduction.

Voici le script à ajouter dans le Jenkins file:

```

stage ('Creation ec2 instance if necessary (PRODUCTION)') {
    agent any
    steps {
        script{
            timeout(time: 30, unit: "MINUTES") {
                input message: 'Do you want to approve the deploy in
production?', ok: 'Yes'
            }

            sh '''
            rm -rf src_terraform
            mkdir src_terraform
            cd src_terraform
            git clone $URL_GIT_TERRAFORM
            cd terraform_node/prod
            terraform init -reconfigure
            terraform apply --auto-approve
            terraform output ec2_ip > ec2_ip.txt
            '''

            env.EC2_STAGING_HOST = sh( script: "sed -e 's/\"//g'
src_terraform/terraform_node/preprod/ec2_ip.txt",returnStdout: true).trim()
            sleep 20
        }
    }
}

```

F- Déploiement en Preproduction

1- Crédit d'un rôle Ansible "docker_role"

Ce rôle "docker_role" a simplement pour but d'installer Docker sur le serveur, nous créons donc un repository docker_role (https://github.com/projetajc-group3/docker_role.git)

Voici son arborescence:

```

.
├── README.md
├── defaults
│   └── main.yml
├── galaxy.yml
└── meta
    └── main.yml
└── tasks
    └── main.yml
└── tests

```

```

└── inventory
    └── test.yml

```

- tasks/main.yml

Ce manifest s'assure de l'installation de Docker et lance son service.

```

---
# tasks file for install_docker
- name: "Generate install docker sh"
  command:
    cmd: "curl -fsSL https://get.docker.com -o get-docker.sh"

- name: "Execute install docker"
  command:
    cmd: "sh get-docker.sh"

- name: "Add Docker Privilege"
  command:
    cmd: "sudo usermod -aG docker ubuntu"

- name: "start docker"
  service:
    name: docker
    state: started
    enabled: yes

```

- meta/main.yml

```

---
galaxy_info:
  role_name: docker_role
  author: Group3
  description: Ansible docker role
  company: UmanisAjc

```

2- Crédit d'un déploiement pour "docker_role"

Nous créons un répository "docker_role_deploy" afin de déployer rapidement le rôle Docker (https://github.com/projetajc-group3/docker_role_deploy.git)

Voici son arborescence:

```
•
├── docker.yml
├── hosts.yml
└── roles
    └── requirements.yml
```

- roles/requirements.yml

Ce manifest permet de récupérer et d'installer le "docker_role":

```
- src: https://github.com/projetajc-group3/docker\_role.git
```

- host.yml

Fichier d'inventaire pour lancer le role sur l'hôte:

```
all:
  hosts:
    localhost:
      ansible_connection: local
```

- docker.yml

Playbook qui lance le "docker_role"

```
- name: deploy docker using a role
  hosts: localhost
  become: true
  roles:
    - docker_role
```

3- Déploiement de l'application en préproduction

- Création de credentials pour se connecter en ssh à l'environnement de preproduction

Tableau de bord ➔ Identifiants ➔ System ➔ Identifiants globaux (illimité) ➔

[Revenir aux identifiants de domaines](#)

[Ajouter des identifiants](#)

Type: SSH Username with private key

Portée: Global (Jenkins, agents, items, etc...)

ID: ec2_staging_private_key

Description: ec2_staging_private_key

Username: ubuntu

Treat username as secret

Private Key: Enter directly

Key:

OK

[Revenir aux identifiants de domaines](#)[Ajouter des identifiants](#)

Identifiants globaux (illimité)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Nom	Type	Description	Actions
 snyk-token	snyk-token	Snyk API token		
 key-projetgrp3	ubuntu (key-projetgrp3)	SSH Username with private key	key-projetgrp3	
 docker-token	docker-tocken	Secret text	docker-tocken	
 slack_token	slack_token	Secret text	slack_token	
 ec2_staging_private_key	ubuntu (ec2_staging_private_key)	SSH Username with private key	ec2_staging_private_key	
 ec2_prod_private_key	ubuntu (ec2_prod_private_key)	SSH Username with private key	ec2_prod_private_key	

Icône:  S  M  L

- Ajout du stage au Jenkinsfile

Nous connectons donc en ssh à l'ec2 à l'aides des credentials prédemment créés ainsi qu'avec l'ip public que nous avions stockés dans une variable d'environnement global à Jenkins.

La première étape consiste à installer Ansible sur la machine afin de pouvoir utiliser notre "docker_role". Nous récupérons donc notre repository de déploiement du "docker_role" afin de l'exécuter sur l'hôte. Une fois Docker installé nous vérifions qu'une ancienne version de notre image n'est pas présente et qu'une ancienne version de notre conteneur n'est pas en cours non plus. Enfin nous lançons la nouvelle version à partir de notre nouvelle image de DockerHub.

Voici le stage que nous rajoutons au Jenkinsfile:

```
stage ('Deploy application (STAGING)') {
    agent any
    steps{
        withCredentials([sshUserPrivateKey(credentialsId:
"ec2_staging_private_key", keyFileVariable: 'keyfile', usernameVariable:
'NUSER')]) {
            catchError(buildResult: 'SUCCESS', stageResult: 'FAILURE') {
                script{
                    sh'''
                        ssh -o StrictHostKeyChecking=no -i ${keyfile}
${NUSER}@${EC2_STAGING_HOST} sudo apt-get update -y
                        ssh -o StrictHostKeyChecking=no -i ${keyfile}
${NUSER}@${EC2_STAGING_HOST} sudo apt-get -y install python3-pip
                        ssh -o StrictHostKeyChecking=no -i ${keyfile}
${NUSER}@${EC2_STAGING_HOST} sudo pip3 install ansible
                        ssh -o StrictHostKeyChecking=no -i ${keyfile}
${NUSER}@${EC2_STAGING_HOST} git clone ${URL_GIT_DEPLOY_DOCKER}
                        ssh -o StrictHostKeyChecking=no -i ${keyfile}
${NUSER}@${EC2_STAGING_HOST} ansible-galaxy install -r
docker_role_deploy/roles/requirements.yml
                        ssh -o StrictHostKeyChecking=no -i ${keyfile}
${NUSER}@${EC2_STAGING_HOST} ansible-playbook -i docker_role_deploy/hosts.yml
docker_role_deploy/docker.yml
                        ssh -o StrictHostKeyChecking=no -i ${keyfile}
${NUSER}@${EC2_STAGING_HOST} sudo rm -rf docker_role_deploy || true
                        ssh -o StrictHostKeyChecking=no -i ${keyfile}
${NUSER}@${EC2_STAGING_HOST} docker stop ${CONTAINER_NAME} || true
                        ssh -o StrictHostKeyChecking=no -i ${keyfile}
${NUSER}@${EC2_STAGING_HOST} docker rm ${CONTAINER_NAME} || true
                        ssh -o StrictHostKeyChecking=no -i ${keyfile}
${NUSER}@${EC2_STAGING_HOST} docker rmi ${USERNAME}/${IMAGE_NAME}:${IMAGE_TAG} || true
                        ssh -o StrictHostKeyChecking=no -i ${keyfile}
${NUSER}@${EC2_STAGING_HOST} docker run -d -p
$STAGING_EXTERNAL_PORT:${CONTAINER_PORT} --name ${CONTAINER_NAME}
${USERNAME}/${IMAGE_NAME}:${IMAGE_TAG}
                        ...
                    }
                }
            }
        }
    }
}
```

G- Déploiement en Production

Nous allons utiliser Kubernetes pour disposer d'une plateforme de déploiement résiliente et à haute disponibilité avec une configuration et une gestion simple à l'aide de nos manifestes.

Kubernetes nous permet d'orchester nos conteneurs sur notre environnement de production et de faire appel à l'image que nous avons conteneurisé et déployé dans le registry Docker Hub.

1- Création d'un role Ansible "kubernetes_role"

Ce rôle "kubernetes_role" va nous permettre l'installation de Kubernetes et de déployer notre service et nos pods, nous créons donc un repository kubernetes_role (https://github.com/projetajc-group3/kubernetes_role.git)

Voici son arborescence:

```
•
├── README.md
├── defaults
│   └── main.yml
├── galaxy.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
│   └── nodeapp.yml.j2
└── tests
    ├── inventory
    └── test.yml
```

- tasks/main.yml

Ce manifest s'assure de l'installation de kubernetes et lance son service. Il contient les différentes instructions afin d'installer kubernetes à l'aide de différentes tâches.

```
---
# tasks file for install kubernetes
- name: install tools
  shell: |
    sudo apt-get -y update
    sudo apt install -y qemu qemu-kvm libvirt-daemon libvirt-clients bridge-utils
    virt-manager
    sudo apt-get install -y socat
    sudo apt-get install -y conntrack
    sudo apt-get -y install wget
- name: "Kubernetes bin"
  shell: |
    sudo wget https://storage.googleapis.com/minikube/releases/latest/minikube-
    linux-amd64
```

```

    sudo chmod +x minikube-linux-amd64
    sudo mv minikube-linux-amd64 /usr/bin/minikube
- name: "Kubernetes ctl"
  shell: |
    sudo curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl
-s https://storage.googleapis.com/kubernetes-
release/release/stable.txt`/bin/linux/amd64/kubectl
    sudo chmod +x kubectl
    sudo mv kubectl /usr/bin/
    sudo echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables
    sudo systemctl enable docker.service
- name: "Kubernetes stop"
  shell: |
    minikube stop
    sudo sysctl fs.protected_regular=0
- name: "Kubernetes launch"
  command: "minikube start --driver=none"

- name: "Create Manifests Kubernetes template"
  template:
    src: "nodeapp.yml.j2"
    dest: "/home/ubuntu/nodeapp.yml"

- name: "Deploy Manifests Kubernetes"
  command: "kubectl apply -f nodeapp.yml"
  args:
    chdir: "/home/ubuntu/"

```

- meta/main.yml

```

---
galaxy_info:
  role_name: kubernetes_role
  author: Group3
  description: Ansible kubernetes role
  company: UmanisAjc

```

- templates/nodeapp.yml.j2

Nous avons créé un template de fichier qui contient un manifest détaillant le déploiement de notre application Node, il sera déployé sur l'instance de production à l'aide du module template, qui prend en charge le transfert d'un fichier local du nœud de contrôle vers l'hôte géré. Le template contient les instructions de base qui seront ensuite recopierées. Il contient également des variables qui seront remplacées individuellement sur la machine cible.

- Nous allons créer un deployment avec 02 replicas de notre application node app
- Créez un service de type "nodeport" pour exposer notre déploiement précédemment

créé

- Nodeport

Un service NodePort est le moyen le plus simple d'aiguiller du trafic externe directement vers un Pod.

NodePort, comme son nom l'indique, ouvre un port spécifique sur tous les Nœuds (les VMs), et tout trafic envoyé vers ce port est transféré vers le service.

Nous allons créer un nouvel objet Service nommé "nodeport", qui cible le port TCP 3000 sur n'importe quel pod avec l'étiquette «app=nodeapp».

- port : port du service
- target port : port du pod à utiliser (doit correspondre à l'application)
- nodeport : port utilisable depuis l'extérieur du cluster

```

apiVersion: v1
kind: Service
metadata:
  name: nodeport
spec:
  type: NodePort
  selector:
    app: nodeapp
  ports:
    #port du pod à utiliser
    - targetPort: {{ containers_port }}
      #port du service
      port: {{ containers_port }}
      #port utilisable depuis l'extérieur du cluster
      nodePort: {{ external_port }}

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nodeapp
  labels:
    app: nodeapp
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nodeapp
  template:
    metadata:
      labels:
        app: nodeapp
    spec:
      containers:
        - name: {{ name_containers }}

```

```
image: {{ image_containers }}
```

```
ports:
```

```
  - containerPort: {{ containers_port }}
```

2- Création d'un déploiement pour "kubernetes_role"

Nous créons un répository "kubernetes_role_deploy" afin de déployer rapidement le rôle Kubernetes (https://github.com/projetajc-group3/kubernetes_role_deploy.git)

Voici son arborescence:

```
.
```

```
└── hosts.yml
```

```
└── kubernetes.yml
```

```
└── roles
```

```
    └── requirements.yml
```

- roles/requirements.yml

Ce manifest permet de récupérer et d'installer le "kubernetes_role":

```
- src: https://github.com/projetajc-group3/kubernetes_role.git
```

- host.yml

Fichier d'inventaire pour lancer le rôle sur l'hôte:

```
all:
```

```
  hosts:
```

```
    localhost:
```

```
      ansible_connection: local
```

- kubernetes.yml

Playbook qui lance le "kubernetes_role"

```
- name: deploy kubernetes using a role
```

```
hosts: localhost
```

```
become: true
```

```
roles:
```

```
  - kubernetes_role
```

3- Déploiement de l'application en production

- Création de credentials pour se connecter en ssh à l'environnement de production

- Ajout du stage au Jenkinsfile

Nous connectons donc en ssh à l'ec2 à l'aides des credentials prédemment créés ainsi qu'avec l'ip public que nous avions stockés dans une variable d'environnement global à Jenkins. La première étape consiste à installer Ansible sur la machine afin de pouvoir utiliser notre "docker_role", et cela de la même manière que dans l'étape "Déploiement de l'application en préproduction dans le chapitre F"

La seconde étape consiste à installer Ansible sur la machine afin de pouvoir utiliser notre "kubernetes_role". Nous récupérons donc notre repository de déploiement du "kubernetes_role" afin de l'exécuter sur l'hôte. Une fois Kubernetes installé nous vérifions qu'une ancienne version de notre image n'est pas présente et qu'une ancienne version de notre déploiement n'est pas en cours non plus. Enfin nous lançons la nouvelle version à partir de notre nouvelle image de DockerHub. Voici le stage que nous rajoutons au Jenkinsfile:

```
stage ('Deploy application (PRODUCTION)') {
    agent any
    when{
        expression{GIT_BRANCH == 'origin/master'}
    }
    steps{
        withCredentials([sshUserPrivateKey(credentialsId:
"ec2_prod_private_key", keyFileVariable: 'keyfile', usernameVariable: 'NUSER')]) {
            catchError(buildResult: 'SUCCESS', stageResult: 'FAILURE') {

```

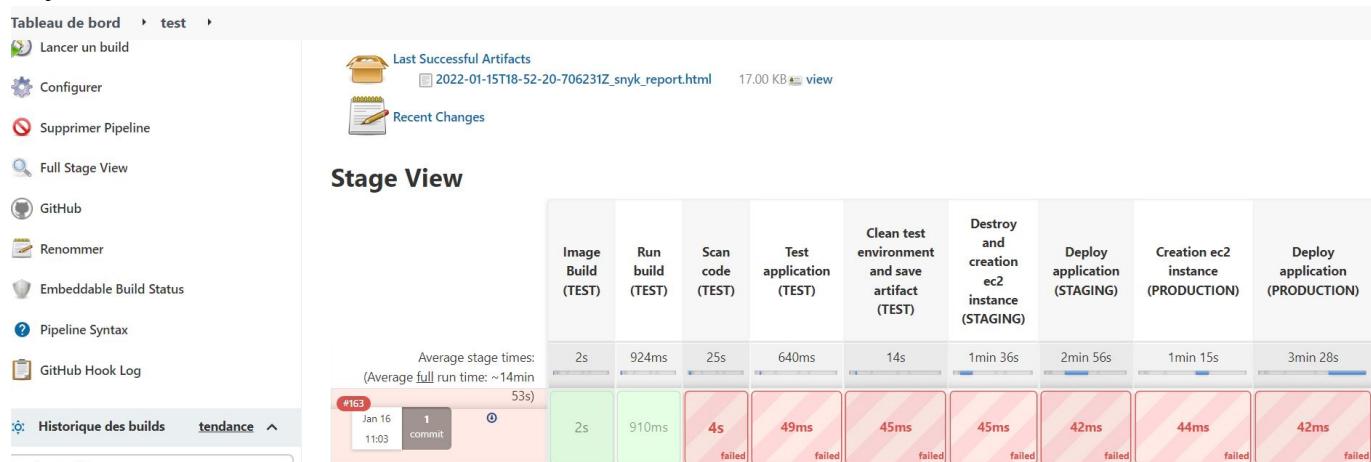
IV- Run du Pipeline Jenkins

Tout est enfin prêt pour le test final du lancement du Pipeline Jenkins. Plusieurs points sont à vérifier.

A- Test

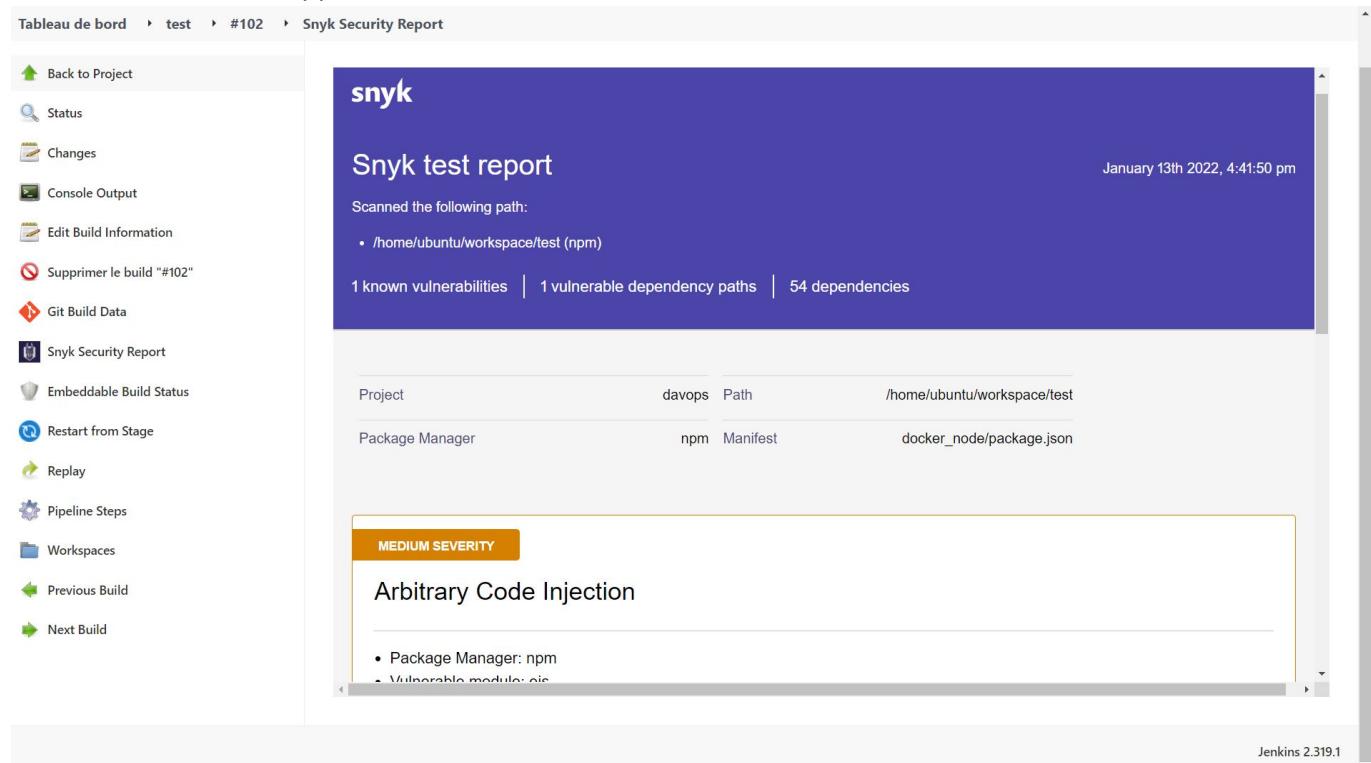
1- Rapport Snyk

En lançant le pipeline sur Jenkins, nous remarquons que le stage "Scan code (TEST)" que nous venons d'ajouter est en échec.



The screenshot shows the Jenkins Pipeline Stage View. On the left, a sidebar lists various Jenkins management options like 'Tableau de bord', 'Lancer un build', 'Configurer', 'Supprimer Pipeline', 'Full Stage View', 'GitHub', 'Renommer', 'Embeddable Build Status', 'Pipeline Syntax', and 'GitHub Hook Log'. The 'Historique des builds' section is currently selected, showing build #103. The main area is titled 'Stage View' and displays a timeline of stages: 'Image Build (TEST)', 'Run build (TEST)', 'Scan code (TEST)', 'Test application (TEST)', 'Clean test environment and save artifact (TEST)', 'Destroy and creation ec2 instance (STAGING)', 'Deploy application (STAGING)', 'Creation ec2 instance (PRODUCTION)', and 'Deploy application (PRODUCTION)'. The 'Scan code (TEST)' stage is highlighted in red, indicating a failure. A tooltip for this stage shows the duration as 4s failed. The overall average stage time is 14min 53s.

Nous allons donc voir le détail du build. Sur la gauche, le menu "Snyk Security Report" est présent. En cliquant dessus nous avons le rapport suivant :



The screenshot shows the Jenkins Pipeline build detail for build #102. The left sidebar includes 'Back to Project', 'Status', 'Changes', 'Console Output', 'Edit Build Information', 'Supprimer le build "#102"', 'Git Build Data', 'Snyk Security Report' (which is currently selected), 'Embeddable Build Status', 'Restart from Stage', 'Replay', 'Pipeline Steps', 'Workspaces', 'Previous Build', and 'Next Build'. The main content area is titled 'snyk' and 'Snyk test report'. It shows the following details: Scanned the following path: '/home/ubuntu/workspace/test (npm)'. Date: January 13th 2022, 4:41:50 pm. Summary: 1 known vulnerabilities | 1 vulnerable dependency paths | 54 dependencies. Below this, it shows the project details: Project: davops, Path: '/home/ubuntu/workspace/test', Package Manager: npm, Manifest: docker_node/package.json. A detailed section for 'Arbitrary Code Injection' is shown, labeled 'MEDIUM SEVERITY'. It lists: 'Package Manager: npm' and 'Vulnerable module: eis'. The Jenkins version is 2.319.1.

- [Back to Project](#)
- [Status](#)
- [Changes](#)
- [Console Output](#)
- [Edit Build Information](#)
- [Supprimer le build "#102"](#)
- [Git Build Data](#)
- [Snyk Security Report](#)
- [Embeddable Build Status](#)
- [Restart from Stage](#)
- [Replay](#)
- [Pipeline Steps](#)
- [Workspaces](#)
- [Previous Build](#)
- [Next Build](#)

Arbitrary Code Injection

- Package Manager: npm
- Vulnerable module: ejs
- Introduced through: davops@0.0.0 and ejs@2.6.2

Detailed paths

- *Introduced through:* davops@0.0.0 > ejs@2.6.2

Overview

[ejs](#) is a popular JavaScript templating engine.

Affected versions of this package are vulnerable to Arbitrary Code Injection via the `render` and `renderFile`. If external input is flowing into the `options` parameter, an attacker is able run arbitrary code. This include the `filename`, `compileDebug`, and `client` option.

POC

```
let ejs = require('ejs')
ejs.render('./views/test.ejs',{
  filename: '/etc/passwd\nfinally { this.global.process.mainModule.require('child_process').execSync('touch EJS_HACKED') }',
  compileDebug: true,
  message: 'test',
  client: true
})
```

- [Back to Project](#)
- [Status](#)
- [Changes](#)
- [Console Output](#)
- [Edit Build Information](#)
- [Supprimer le build "#102"](#)
- [Git Build Data](#)
- [Snyk Security Report](#)
- [Embeddable Build Status](#)
- [Restart from Stage](#)
- [Replay](#)
- [Pipeline Steps](#)
- [Workspaces](#)
- [Previous Build](#)
- [Next Build](#)

Affected versions of this package are vulnerable to Arbitrary Code Injection via the `render` and `renderFile`. If external input is flowing into the `options` parameter, an attacker is able run arbitrary code. This include the `filename`, `compileDebug`, and `client` option.

POC

```
let ejs = require('ejs')
ejs.render('./views/test.ejs',{
  filename: '/etc/passwd\nfinally { this.global.process.mainModule.require('child_process').execSync('touch EJS_HACKED') }',
  compileDebug: true,
  message: 'test',
  client: true
})
```

Remediation

Upgrade `ejs` to version 3.1.6 or higher.

References

- [GitHub Commit](#)
- [GitHub Issue](#)

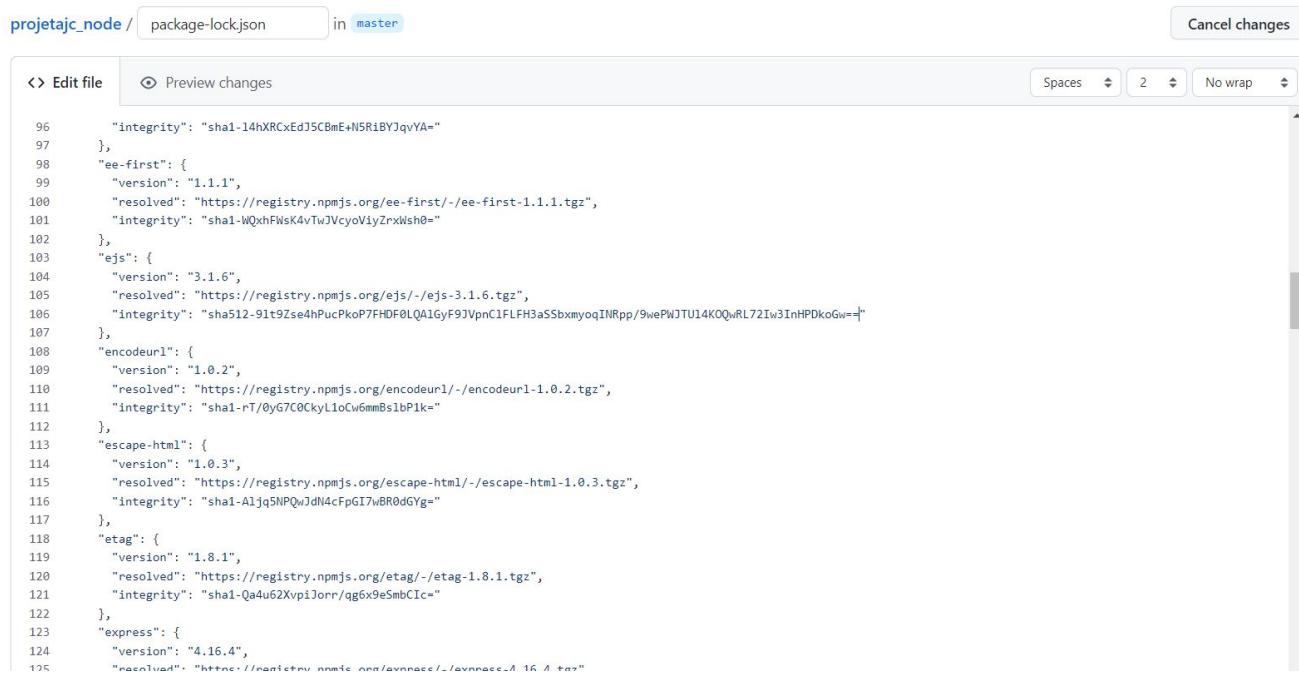
[More about this vulnerability](#)

Il est clair que l'échec est dû à une vulnérabilité qui peut être corrigée simplement en utilisant une version plus à jour de la dépendance [ejs](#).

2- Correction du Scan

Pour ça on met à jour le fichier `package.json` ![package_json](images/Snyk/package_json_after.JPG)

On met également à jour le fichier **package-lock.json** :



```
96      "integrity": "sha1-14hXRCxEdJ5CBmE+NSRiBYJqvYA="
97  },
98  "ee-first": {
99    "version": "1.1.1",
100   "resolved": "https://registry.npmjs.org/ee-first/-/ee-first-1.1.1.tgz",
101   "integrity": "sha1-WQxhFWsK4vTwJVCyoViYzrxWsh0="
102 },
103  "ejs": {
104    "version": "3.1.6",
105   "resolved": "https://registry.npmjs.org/ejs/-/ejs-3.1.6.tgz",
106   "integrity": "sha1-512-91t9Zse4hPucPkoP7FHD8LQAIgF9JVpnC1FLFH3aSSbxmyoqINRpp/9wePWJTU14KOQwRL72Iw3InHPDkoGw="
107 },
108  "encodeurl": {
109    "version": "1.0.2",
110   "resolved": "https://registry.npmjs.org/encodeurl/-/encodeurl-1.0.2.tgz",
111   "integrity": "sha1-r1/0yG7c0kyL1oCw6mmBslbP1k="
112 },
113  "escape-html": {
114    "version": "1.0.3",
115   "resolved": "https://registry.npmjs.org/escape-html/-/escape-html-1.0.3.tgz",
116   "integrity": "sha1-Aljq5MPQwJdN4cFpGI7wBR0dGYg="
117 },
118  "etag": {
119    "version": "1.8.1",
120   "resolved": "https://registry.npmjs.org/etag/-/etag-1.8.1.tgz",
121   "integrity": "sha1-Qa4u62XvpiJorr/qg6x9eSmbC1c="
122 },
123  "express": {
124    "version": "4.16.4",
125   "resolved": "https://registry.npmjs.org/express/-/express-4.16.4.tgz"
```

Enfin, il reste à remplacer les fichiers présents dans **node_modules/ejs** par les fichiers de la version 3.1.6.

Une fois modifié avec la valeur recommandée par Snyk, nous relançons le pipeline. Et comme attendu, cette fois Snyk n'arrête pas le pipeline, et le rapport est vide !

Tableau de bord ➔ test ➔ Pipeline test

Pipeline test

Ajouter une description Désactiver le projet

Last Successful Artifacts 2022-01-16T10-44-05-814976Z_snyk_report.html 17.00 KB view

Recent Changes

Stage View

Image Build (TEST)	Run build (TEST)	Scan code (TEST)	Test application (TEST)	Clean test environment and save artifact (TEST)	Destroy and creation ec2 instance (STAGING)	Deploy application (STAGING)	Creation ec2 instance (PRODUCTION)	Deploy application (PRODUCTION)
10s	855ms	3s	232ms	4s	19s	24s	12s	35s
Jan 16 14:01	1 commit							
5s	938ms	3s	684ms	16s				
11min 27s								

Historique des builds tendance ▾

Filter builds...

Jenkins

Tableau de bord ➔ test ➔ #172 ➔ Snyk Security Report

Back to Project Status Changes Console Output Edit Build Information Log de scrutation Git Build Data Snyk Security Report Embeddable Build Status Thread Dump Pause/resume Replay Pipeline Steps Workspaces

snyk

Snyk test report January 16th 2022, 1:01:40 pm

Scanned the following path:

- /home/ubuntu/workspace/test (npm)

0 known vulnerabilities | 0 vulnerable dependency paths | 54 dependencies

Project	davops	Path	/home/ubuntu/workspace/test
Package Manager	npm	Manifest	projetojo_node/package.json

No known vulnerabilities detected.

54.174.144.82:8080/job/test/172/snykReport/

Note: Nous nous sommes permis de corriger cette erreur pour des raisons de démonstrations du module Snyk. En production réelle la bonne méthode serait plutôt de prévenir l'équipe de développement du problème.

B- Staging

Arrivé en **Staging**, nous vérifions que notre machine ec2 a bien été créé :

Instances (1/61) [Informations](#)

Name	ID d'instance	État de l'instance	Type d'insta...	Contrôle des statuts	Statut d'alarm
oussama-Puppet-slave	i-06958858d8115c7d8	Arrête(e)	t3.small	—	Aucune alarme
projetgrp3-ec2-jenkins	i-04ce15266a590e510	En cours d'exécution	t2.large	2/2 vérifications réussies	Aucune alarme
projetgrp3-ec2-jenkins-agent	i-091f71ed022116048	En cours d'exécution	t2.large	2/2 vérifications réussies	Aucune alarme
projetgrp3-ec2-preprod	i-0c787913d9c583046	En cours d'exécution	t2.medium	2/2 vérifications réussies	Aucune alarme

Instance : i-0c787913d9c583046 (projetgrp3-ec2-preprod)

[Détails](#) [Sécurité](#) [Mise en réseau](#) [Stockage](#) [Vérifications de statut](#) [Surveillance](#) [Balises](#)

Résumé de l'instance [Informations](#)

ID d'instance	Adresse IPv4 publique	Adresses IPv4 privées
i-0c787913d9c583046 (projetgrp3-ec2-preprod)	107.21.172.98 adresse ouverte	172.31.31.22
Adresse IPv6	État de l'instance	DNS IPv4 public
—	En cours d'exécution	ec2-107-21-172-98.compute-1.amazonaws.com adresse ouverte
Type de nom d'hôte	Nom DNS d'IP privé (IPv4 uniquement)	Réponse à un nom DNS de ressource privée
Nom de l'adresse IP: ip-172-31-31-22.ec2.internal	ip-172-31-31-22.ec2.internal	—

Rappelons que le port que nous avons variabilisé dans ce pipeline à une valeur affecté de 30001 lors de ce test. Donc en visitant <http://107.21.172.98:30001> nous devrions voir notre application. Et effectivement :

← → ⏪ ⏪ Non sécurisé | 107.21.172.98:30001

ACCUEIL M'INSCRIRE AVEC CPF CONTACT

Cette offre de formation est éligible à

MON COMPTE FORMATION

moncompteformation.gouv.fr

Présentation

La formation DevOps est un ensemble de pratiques mettant l'accent sur la collaboration et la communication entre les développeurs de logiciels et d'autres professionnels du système d'information, tout en automatisant le processus de livraison des logiciels et les changements d'infrastructure. L'objectif est de créer

C- Production

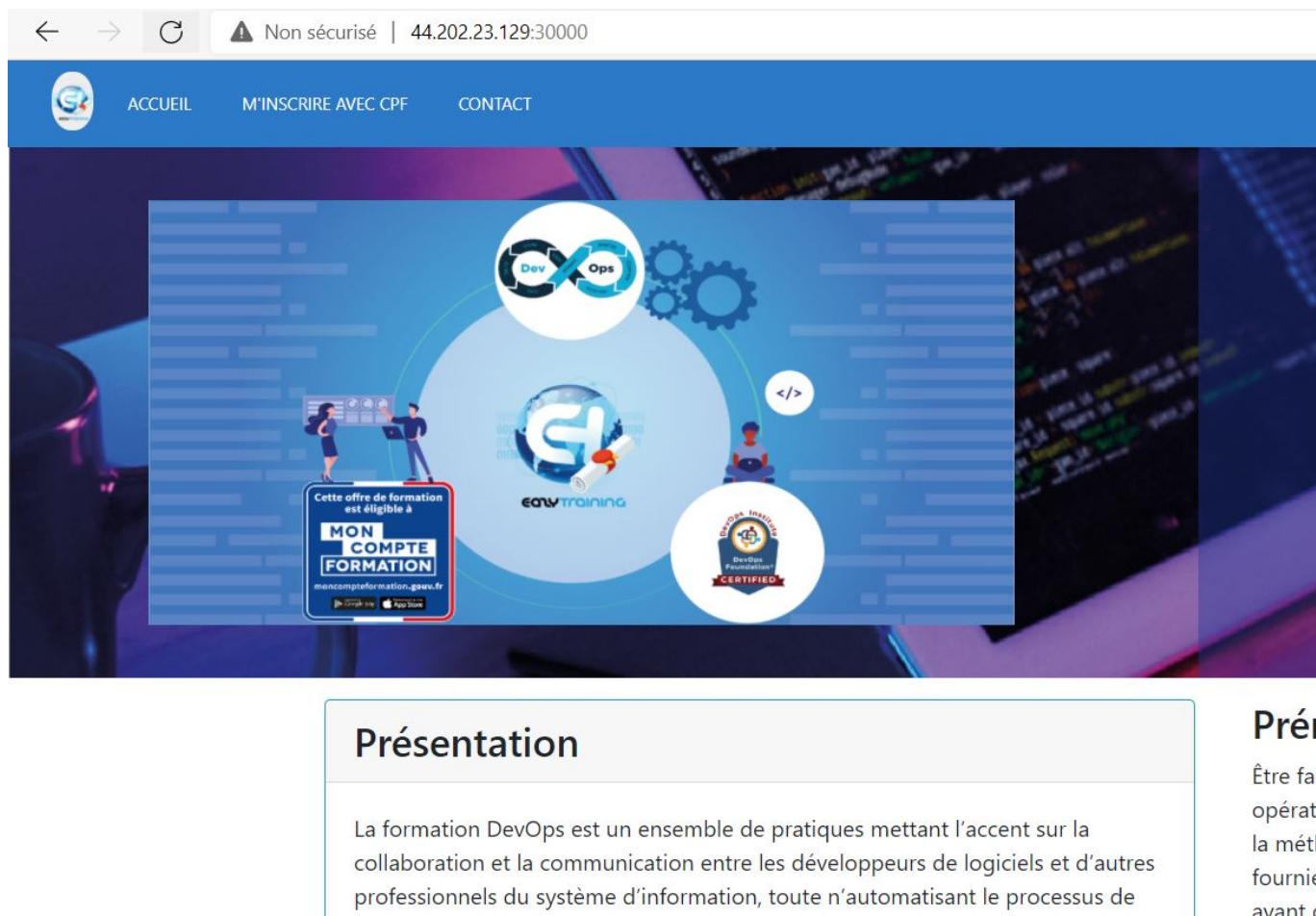
Passons à la production. Ici nous devons d'abord approuver le déploiement :

The screenshot shows the Jenkins Pipeline interface for 'projetajc-groups'. On the left, a sidebar lists various pipeline management options. The main area is titled 'Pipeline projetajc-groups' and shows a 'Stage View' grid. The grid has columns for 'Image Build (TEST)', 'Run build (TEST)', 'Test application (TEST)', 'Scan code (TEST)', 'Clean test environment and save artifact (TEST)', 'Creation ec2 instance if necessary (STAGING)', 'Deploy application (STAGING)', 'Creation ec2 instance if necessary (PRODUCTION)', and 'Deploy application (PRODUCTION)'. The 'Creation ec2 instance if necessary (PRODUCTION)' column contains a yellow box with the text 'Do you want to approve the deploy in production?' and two buttons: 'Yes' and 'Abort'. Below the grid, a message indicates '(paused for 1min 11s)'. The sidebar also shows a 'Historique des builds' section with two entries: '#3 17 janv. 2022 17:59' and '#2 17 janv. 2022 16:24'.

Verifions ensuite si notre machine a été créée :

The screenshot shows the AWS CloudWatch Instances page. The table lists four instances: 'projetgrp3-ec2-prod' (status: En cours d'exécution, type: t2.large), 'pw_app' (status: Arrêté(e), type: t2.micro), 'pw_serv' (status: Arrêté(e), type: t2.micro), and 'Serveur_web_python_arthur' (status: En cours d'exécution, type: t2.micro). The 'projetgrp3-ec2-prod' instance is selected. The detailed view for this instance shows its public IP (44.202.23.129), private IP (172.31.89.56), and DNS name (ec2-44-202-23-129.compute-1.amazonaws.com). The 'Type d'instance' section shows it is an 'Adresses IP Elastic' instance.

Enfin de la même manière qu'en Staging, le port variabilisé à été affecté avec la valeur 30000 pour ce test. L'application devrait être déployé sur <http://44.202.23.129:30000> ce qui est bien le cas :



Non sécurisé | 44.202.23.129:30000

ACCUEIL M'INSCRIRE AVEC CPF CONTACT

Cette offre de formation est éligible à **MON COMPTE FORMATION** moncompteformation.gouv.fr

DevOps

DevOps Institute DevOps Foundation CERTIFIED

Présentation

La formation DevOps est un ensemble de pratiques mettant l'accent sur la collaboration et la communication entre les développeurs de logiciels et d'autres professionnels du système d'information, tout en automatisant le processus de

Pré

Être fait opérat la métal fournie avant c

D- Eléments supplémentaires

1- Github Webhook

Nous souhaitons que le pipeline CICD se lance automatiquement après chaque commit sur le projet [projetajc-node](#) sur notre serveur de gestion de version Github. Pour ce faire, nous configurons un webhook comme trigger pour déclencher notre pipeline CICD.

Pour ce faire on installe le plugin "Github Integration" sur Jenkins :

- Administrer Jenkins -> Gestion des plugins -> Disponibles -> Recherche de "Github Integration" -> Install without restart

Tableau de bord > Gestion des plugins

Retour au tableau de bord

Administrer Jenkins

Update Center

GitHub Integration

emallex Build Triggers

GitHub Integration Plugin for Jenkins

Skip Notifications Trait

DotCi

Install without restart Download now and install after restart

Version Released

0.4.0 4j 11 h ago

1.0.5 2 an. 1 mo. ago

2.40.0 3 an. 11 mo. ago

Update information obtained: 1 j 22 h ago

Vérifier maintenant

Ensute, dans la configuration de notre projet sur Jenkins nous cochons la case "GitHub hook trigger for GITScm polling".

Tableau de bord > test >

General Build Triggers Advanced Project Options Pipeline

Avancé...

Pipeline speed/durability override

Preserve stashes from completed builds

Supprimer les anciens builds

Throttle builds

Build Triggers

Construire après le build sur d'autres projets

Construire périodiquement

GitHub hook trigger for GITScm polling

Scrutation de l'outil de gestion de version

Désactiver le projet

Période d'attente

Déclencher les builds à distance (Par exemple, à partir de scripts)

Advanced Project Options

Avancé...

Pipeline

Definition

Pipeline script from SCM

Sauver Apply

Enfin, on prend le soin de relier Github à notre serveur Jenkins. Pour ce faire, sur Github:

- **projetajc-node** -> setting -> add webhook

Options
Collaborators
Security & analysis
Branches
Webhooks
Notifications
Integrations
Deploy keys
Actions
Environments
Secrets
Pages
Moderation settings

Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

Payload URL *
https://example.com/postreceive

Content type
application/x-www-form-urlencoded

Secret

Which events would you like to trigger this webhook?

Just the push event.
 Send me everything.
 Let me select individual events.

Active
We will deliver event details when this hook is triggered.

Add webhook

- on saisi dans "Payload URL" l'URL pour accéder à notre serveur Jenkins suivi par un github-webhook.
Nous testons la connexion et nous pouvons constater que le lien a été bien mis en place.

projetajc-group3 / **projetajc_node** Public
forked from [sadofrazer/devops-project](#)

Code Pull requests Actions Projects Wiki Security Insights Settings

Webhooks

Add webhook

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).

✓ http://54.174.144.82:8080/github... (push) Edit Delete

Options
Collaborators
Security & analysis
Branches
Webhooks
Notifications

2- Github Build Status

Pour avoir le badge **build passing** sur le README de notre projet sur le repo Github, nous installons le plugin "Embeddable Build Status"

- Administrer Jenkins -> Gestion des plugins -> Disponibles -> Recherche de "Embeddable Build Status" -> Install without restart

Nous prenons le soin de copier le "links/markdown/unprotected"

- Sur Jenkins -> Embeddable Build Status -> links/markdown/unprotected

Links

Plain Link

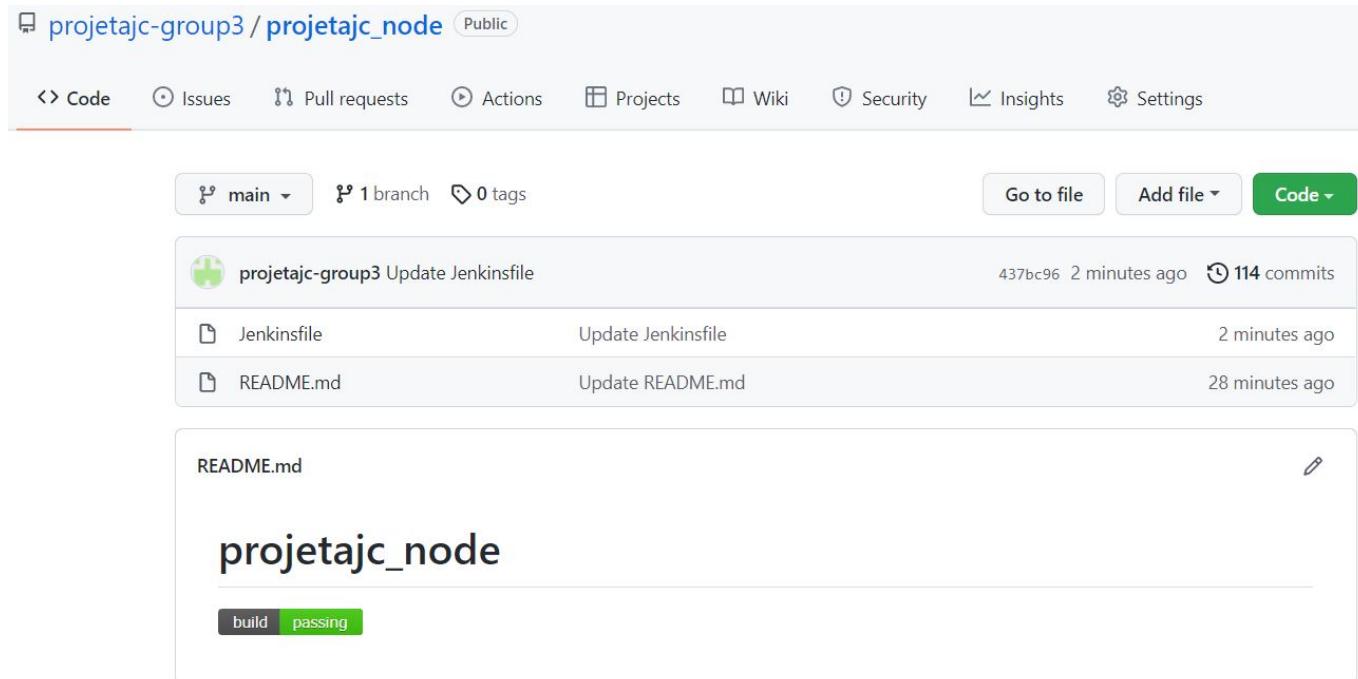
protected
<http://54.174.144.82:8080/job/test/badge/icon>
unprotected
<http://54.174.144.82:8080/buildStatus/icon?job=test>

Markdown

protected
`![Build Status](http://54.174.144.82:8080/job/test/badge/icon)`
unprotected
`![Build Status](http://54.174.144.82:8080/buildStatus/icon?job=test)`

HTML

Nous collons ce lien à la fin de notre fichier **README.md** de projet **projetajc-node** sur Github.



projetajc-group3 / **projetajc_node** Public

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags Go to file Add file Code

projetajc-group3 Update Jenkinsfile 437bc96 2 minutes ago 114 commits

Jenkinsfile Update Jenkinsfile 2 minutes ago

README.md Update README.md 28 minutes ago

README.md

projetajc_node

build passing

3- Notifications Slack

Nous voulons aussi que notre pipeline puisse communiquer avec Slack sur le succès ou l'échec du pipeline. Ainsi que les adresses IP utilisées sur les machines de staging et production. Pour l'occasion nous créons un compte Slack.

Création d'un compte slack:



Tout d'abord, saisissez votre adresse e-mail

Nous vous suggérons d'utiliser votre adresse e-mail professionnelle.

 Continuer avec Google

 Continuer avec Apple

OU

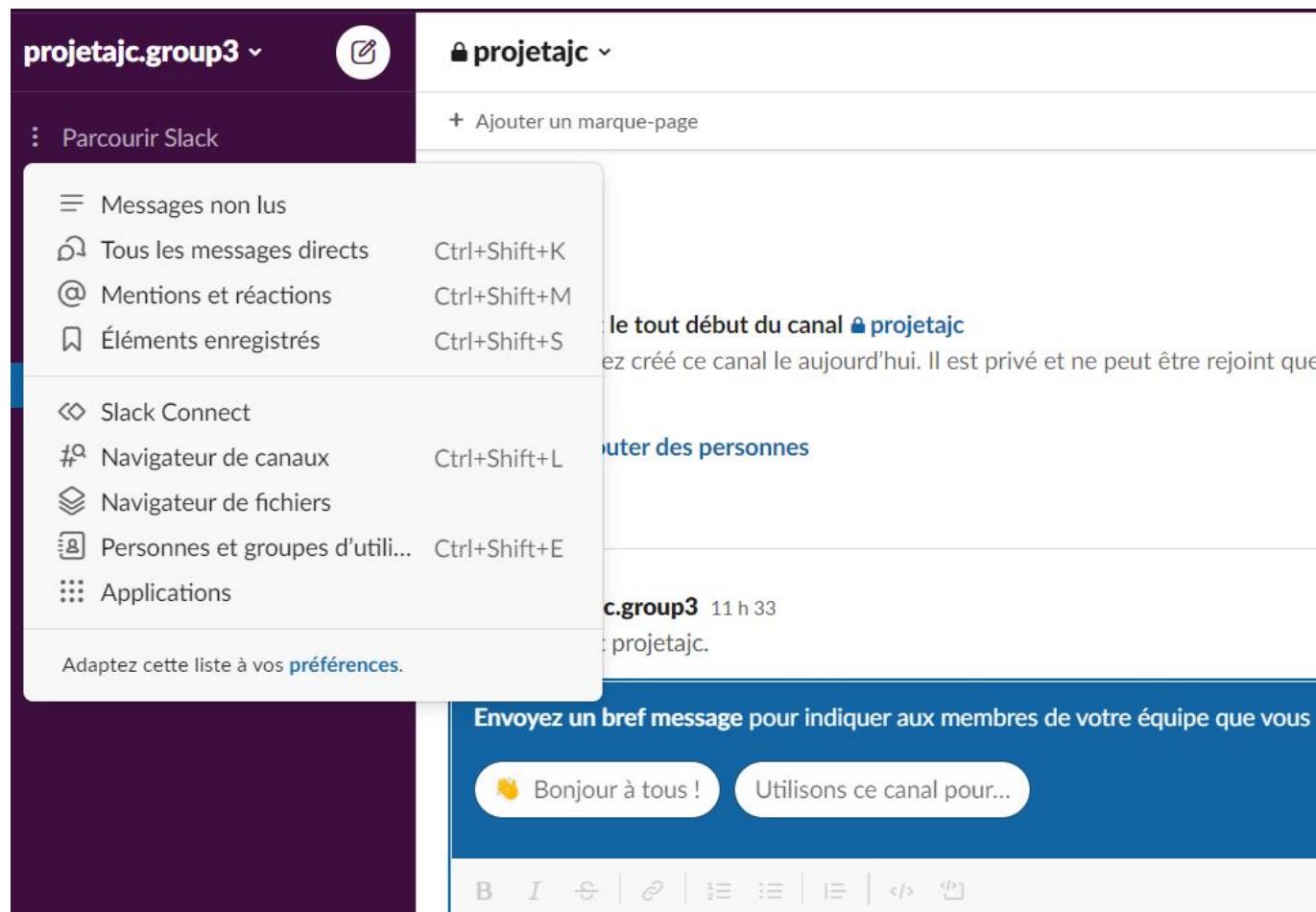
projetajc.group3@gmail.com



Utilisez votre adresse e-mail professionnelle (si vous en avez une) pour inviter plus facilement vos collègues sur Slack. [Changer](#)

Continuer

Après avoir créé un canal privé sur lequel Jenkins communiquera nous procéderons à l'association en ajoutant l'application Jenkins dans Slack.



The screenshot shows the Slack interface with a context menu open over a private channel named "projetajc". The menu includes options like "Messages non lus", "Tous les messages directs", "Mentions et réactions", "Eléments enregistrés", "Slack Connect", "Navigateur de canaux", "Navigateur de fichiers", "Personnes et groupes d'utilisateurs", and "Applications". A note at the bottom of the menu says "Adaptez cette liste à vos préférences." The main Slack window shows a message from "projetajc" at 11 h 33.

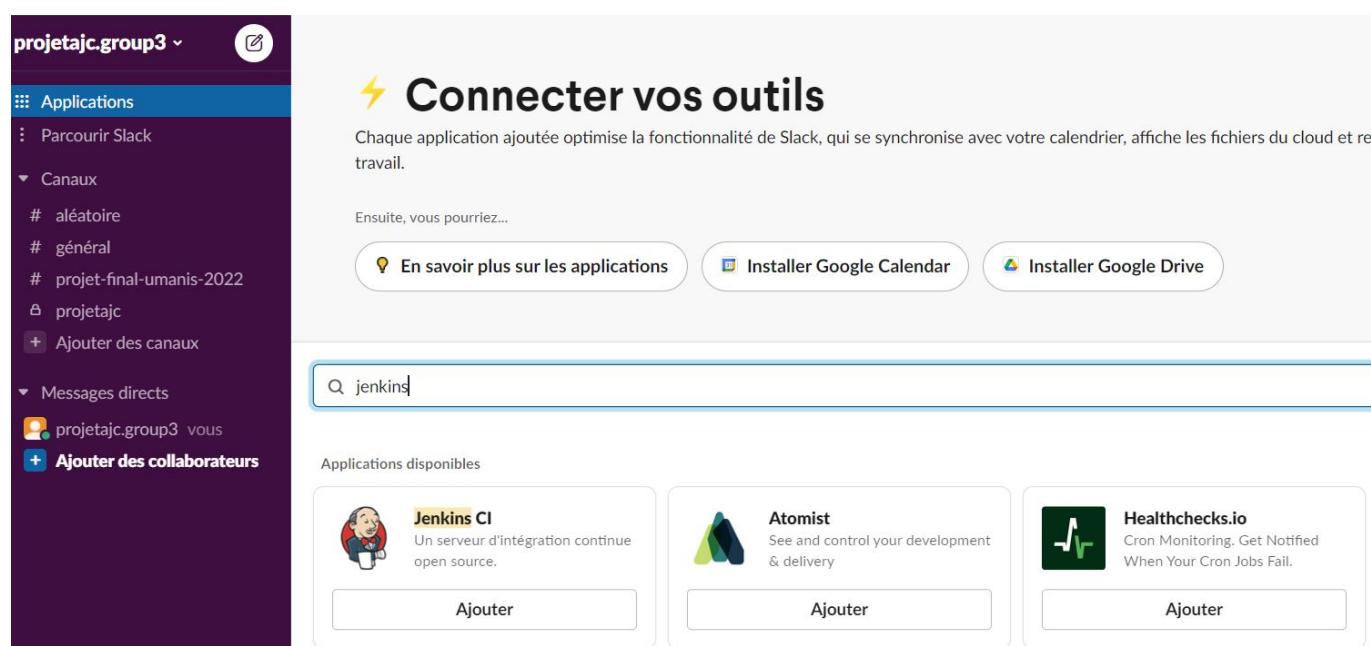
projetajc.group3 11 h 33

Le tout début du canal **projetajc**
ez créé ce canal le aujourd'hui. Il est privé et ne peut être rejoint que par les personnes qui ont été invitées.

projetajc.

Envoyez un bref message pour indiquer aux membres de votre équipe que vous utilisez ce canal.

B I ⌂ | ⌂ | ⌂ | ⌂ | ⌂ | ⌂ | ⌂ | ⌂



The screenshot shows the "Connecter vos outils" (Connect your tools) page in the Slack workspace "projetajc.group3". The "Applications" tab is selected. It shows a search bar with "jenkins" typed in, and a list of available applications: Jenkins CI, Atomist, and Healthchecks.io. Each application has a "Ajouter" (Add) button.

Connecter vos outils

Chaque application ajoutée optimise la fonctionnalité de Slack, qui se synchronise avec votre calendrier, affiche les fichiers du cloud et rend votre travail plus efficace.

Ensuite, vous pourriez...

[En savoir plus sur les applications](#) [Installer Google Calendar](#) [Installer Google Drive](#)

Q jenkins

Applications disponibles

Jenkins CI Un serveur d'intégration continue open source. [Ajouter](#)

Atomist See and control your development & delivery. [Ajouter](#)

Healthchecks.io Cron Monitoring. Get Notified When Your Cron Jobs Fail. [Ajouter](#)

Nous récupérerons les informations (le workspace et le secret) :

Parcourir les applications > Jenkins CI > Nouvelle configuration



Jenkins CI

Un serveur d'intégration continue open source.

Jenkins CI est un serveur d'intégration continue personnalisable qui compte plus de 600 modules d'extension, ce qui vous permet de le configurer selon vos besoins.

Cette intégration publie des notifications de build dans une chaîne Slack.

Publier dans le canal

Commencez par choisir un canal dans lequel les notifications Jenkins seront publiées.

Choisir un canal...

Choisir un canal...

- o projetjc.group3
- # aléatoire
- # général
- # projet-final-umanis-2022
- 🔒 projetjc
- ▼ Slackbot Slackbot

Étape 3

Une fois l'installation effectuée, cliquez de nouveau sur **Manage Jenkins** (Gérer Jenkins) dans le volet de navigation gauche, puis accédez à **Configure System** (Configurer le système).

Recherchez la section **Global Slack Notifier Settings** (Paramètres de notification Slack globaux) et ajoutez les valeurs suivantes :

- **Sous-domaine de l'équipe :** [REDACTED]
- **Identifiant d'authentification du jeton d'intégration :** Créez un identifiant de texte secret ayant pour valeur [REDACTED]

De retour sur Jenkins nous lançons l'installation du plugin Slack et sa configuration

- Administrer Jenkins -> Gestion des plugins -> Disponibles -> Recherche de "slack" -> Install without restart

Tableau de bord > Gestion des plugins

Retour au tableau de bord

Administrer Jenkins

Update Center

slack

Mises à jour Disponibles Installés Avancé

Install	Name	Version	Released
<input checked="" type="checkbox"/>	Slack Notification Build Notifiers slack	2.49	1 mo. 20 j ago
<input type="checkbox"/>	Global Slack Notifier slack	1.5	2 an. 11 mo. ago
<input type="checkbox"/>	Build Notifications Send build notifications through Telegram, Pushover, Botoclo or Slack.	1.5.0	4 an. 4 mo. ago

- Sur Jenkins -> configurer le système -> Slack -> Créer le Credential **slack_token** de type "secret text" qui permettra à Jenkins d'écrire dans le canal créé sur slack et donc nous envoyer des notifications concernant l'évolution de notre pipeline.

Pour que notre pipeline envoie une information à Slack il suffit de faire appel à la commande `slackSend` de cette manière :

```
slackSend (color: '#00FF00', message: "SUCCESSFUL: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]' (${env.BUILD_URL})")
```

Le résultat sur Slack est comme suit :

V- Axes d'amélioration

Plusieurs points de notre projet peuvent être améliorés:

A- Jenkinsfile

Quelques lignes de commandes du Jenkinsfile sont redondantes avec les rôles Ansible que nous avons créés. Les **stages** pourraient aussi mieux gérer les erreurs potentiel avec plus de **catchError**

B- Monitoring

La mise en place d'un outil de monitoring de la production serait un plus pour le maintien de l'application mais aussi pour surveiller et améliorer les performances de déploiement ce qui s'inscrit dans la philosophie DevOps.

C- Ingress

Par défaut, les Services NodePort exposent nos containers sur le port 30001. Afin de rendre l'accès à l'URL plus simple, il est aussi possible d'utiliser un objet Kubernetes Ingress qui gère l'accès externe aux services dans un cluster, dans notre cas du trafic HTTP.