RÉPUBLIQUE DU BÉNIN
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ D'ABOMEY-CALAVI

INSTITUT DE FORMATION ET DE RECHERCHE EN INFORMATIQUE

BP 526 Cotonou Tel : +229 21 14 19 88
http://www.ifri-uac.bj Courriel : contact@ifri.uac.bj

MÉMOIRE

pour l'obtention du

Diplôme de Licence en Informatique

Option : Sécurité Informatique

Présenté par :
Koutikomou Stanislas **BAYORD**

Sécurité des infrastructures informatiques avec la Politique en tant que Code en DevOps

Sous la supervision :
Ing. Francky **ADEOSSI**

Maurice COMLAN	Docteur(MA)	ENEAM	Président
Honorat BIBI	Ingénieur	IFRI	Examinateur
Francky ADEOSSI	Ingénieur	IFRI	Rapporteur

Année Académique : **2022-2023**

Sommaire

Dédicace ii

Remerciements iii

Résumé iv

Abstract v

Table des figures vi

Liste des tableaux vii

Liste des acronymes viii

Introduction 1

1 État de l'Art 3

2 Pipeline et Solution Politique en tant que Code (PaC) 13

3 Tests et résultats 29

Conclusion 38

Bibliographie 39

Webographie 40

Table des matières 42

Dédicace

À ma mère **Y. Pauline N'KOUEI NATA** ;

À mon père **B. Mathieu BAYORD** ;

Remerciements

J'adresse mes plus sincères remerciements :

- À mon **Dieu**, Le tout Puissant ;
- À mon maître mémoire, monsieur **Francky ADEOSSI** qui malgré son emploi du temps chargé a pris la responsabilité de m'encadrer ;
- Au Professeur **Eugène C. EZIN**, à tout le personnel administratif et au corps professoral de l'**Institut de Formation et de Recherche en Informatique** ;
- À la **fondation VALLET**, au **CAEB** et à tous leurs collaborateurs spécialement à monsieur **Odon VALLET** pour son soutien voire son aide depuis mes classes du collège ;
- À toute personne qui, de près ou de loin, m'a soutenu ne serait-ce que par un petit mot d'encouragement.

... À vous tous, je dis un grand merci !!!

Résumé

Au cœur de l'évolution rapide du développement logiciel, l'équilibre entre agilité opérationnelle et sécurité devient une préoccupation cruciale. L'émergence des pratiques DevOps, axées sur la rapidité et l'efficacité, accentue la nécessité d'adapter les approches de sécurité. Dans ce contexte dynamique, plusieurs approches ont vu le jour, notamment celle de la Politique en tant que Code. La question fondamentale qui se pose est celle de savoir comment intégrer de manière effective la Politique en tant que Code au sein d'un environnement DevOps pour renforcer la sécurité tout au long du cycle de développement.
Le but de ce projet est d'étudier les fondements de la Politique en tant que Code, d'explorer sa mise en œuvre pratique dans un contexte DevOps plus précisément un pipeline CI/CD, et s'assurer de l'automatisation des vérifications de conformité.

Mots clés : Politique, Code, DevOps, Infrastructure, CI/CD.

Abstract

At the heart of the rapidly evolving software development landscape, the balance between operational agility and security becomes a crucial concern. The emergence of DevOps practices, focused on speed and efficiency, emphasizes the need to adapt security approaches. In this dynamic context, several approaches have emerged, notably that of Policy as Code. The fundamental question is how to effectively integrate Policy as Code within a DevOps environment to enhance security throughout the development cycle.
The goal of this project is to study the foundations of Policy as Code, explore its practical implementation in a DevOps context, specifically in a CI/CD pipeline, and ensure the automation of compliance checks.

Key words: Policy, Code, DevOps, Infrastructure, CI/CD.

Table des figures

1.1 Schéma de la boucle DevOps [20]. . . . 4
1.2 Schéma résumant le fonctionnement. . . . 8

2.1 Logo de Jenkins . . . 15
2.2 Logo de Git . . . 15
2.3 Logo de Apache Maven . . . 15
2.4 Logo de Checkstyle . . . 16
2.5 Logo de Slack . . . 16
2.6 Logo de Nexus Repository . . . 16
2.7 Logo de SonarQube . . . 17
2.8 Logo de Ansible . . . 17
2.9 Logo de Selenium . . . 17
2.10 Logo de Apache Tomcat . . . 18
2.11 Schéma de l'Architecture du pipeline . . . 19
2.12 Logo de Open Policy Agent . . . 20

3.1 Tableau de bord Jenkins . . . 29
3.2 Tableau de bord SonarQube . . . 29
3.3 Interface Nexus . . . 30
3.4 Présentation du CI du pipeline . . . 30
3.5 Création du job de test de politiques . . . 31
3.6 Ajout de l'URL et la branche du projet . . . 31
3.7 Script dans le job . . . 32
3.8 Dernière étape du job . . . 33
3.9 Script pour l'exécution du job . . . 33
3.10 Partie CD du pipeline . . . 34
3.11 Test du job isolé . . . 34
3.12 Modification du code . . . 35
3.13 Résultat d'échec après la modification du code . . . 35
3.14 Données des logs du job . . . 36
3.15 Espace Slack avant le début des tests . . . 36
3.16 Pipeline déclenché par un utilisateur manuellement . . . 36
3.17 Résultat pipeline déclenché par un utilisateur manuellement . . . 36
3.18 Pipeline déclenché automatiquement par un commit . . . 37
3.19 Résultat pipeline déclenché automatiquement par un commit . . . 37
3.20 Pipeline déclenché automatiquement par un commit après modification . . . 37
3.21 Résultat pipeline déclenché automatiquement par un commit après modification . . . 37

Liste des tableaux

1.1 Tableau des outils de PaC (moteurs et langages de politique) 10
1.2 Exemple de tableau de note pour deux solutions. 12

Liste des acronymes

API :

Interface de Programmation d'Application 5, 20

AWS :

Amazon Web Services 10, 20

CI/CD :

Intégration Continue/Livraison Continue 1, 2, 4, 6, 8–10, 12–20, 24, 25, 28, 29, 37, 42, 43

IaC :

Infrastructure en tant que Code 4, 9, 42

JSON :

JavaScript Object Notation 20, 32

OPA :

Open Policy Agent 20–23, 28, 30, 32, 35, 43

PaC :

Politique en tant que Code i, vii, 1, 2, 8–30, 42, 43

SSH :

Secure Shell 25, 35, 37

YAML :

YAML Ain't Markup Language 7, 10

Introduction Générale

1. Contexte et Justification

Avec l'évolution rapide des technologies et l'adoption généralisée des pratiques de développement et de déploiement continu, les environnements DevOps sont devenus incontournables dans le domaine de l'informatique. C'est une approche qui favorise la collaboration étroite entre les équipes de développement et d'exploitation, accélérant ainsi le cycle de vie des applications et permettant des déploiements plus fréquents. Cependant, cette agilité accrue expose également les infrastructures informatiques à des défis de sécurité complexes. Les cycles de déploiement rapides et la multiplication des points de contrôle dans un environnement DevOps exigent une adaptation des pratiques de sécurité traditionnelles.
Le recours à la Politique en tant que Code (PaC) émerge comme une solution prometteuse pour relever ces défis. Cette solution offre la possibilité de définir, de gérer et d'assurer la conformité aux politiques de sécurité de manière automatisée, tout en s'intégrant harmonieusement au flux de travail DevOps, de s'adapter aux nouvelles pratiques apportées par le DevOps, de garantir une réactivité accrue face à paysage de menaces en constante évolution. Le présent projet vise à étudier l'approche de PaC ainsi que sa mise en oeuvre en DevOps plus précisément dans un pipeline Intégration Continue/Livraison Continue (CI/CD)

2. Probématique

Garantir une sécurité efficace des infrastructures informatiques dans un environnement DevOps en constante évolution, tout en assurant une conformité rigoureuse et une intégration fluide des politiques de sécurité au sein du cycle de vie de développement, d'intégration et de déploiement continu.

3. Objectifs

L'objectif de ce travail est d'étudier et d'explorer comment la mise en œuvre de la Politique en tant que Code peut renforcer la sécurité des infrastructures informatiques dans un environnement DevOps, en mettant l'accent sur l'automatisation, la conformité et l'intégration transparente des politiques de sécurité. De façon spécifique, il s'agit de :

- Identifier et définir des politiques spécifiques répondant aux exigences particulières d'un environnement DevOps.
- Convertir ces politiques en code informatique utilisable, en utilisant des outils et des langages adaptés.
- Intégrer le code de politique résultant dans un pipeline CI/CD DevOps.

4. Organisation du mémoire

Le présent travail est structuré en trois (03) chapitres. Le premier est l'état de l'art qui traite des concepts fondamentaux liées à la Politique en tant que Code et la compréhension de cette approche. Le deuxième chapitre est intitulé Pipeline et Solution PaC et il aborde l'outil de PaC et le pipeline exhaustif qui seront utilisés pour cette étude. Le troisième chapitre, quant à lui, est intitulé Tests et résultats ; il traite du processus d'intégration et des résultats obtenus.

Chapitre 1

État de l'Art

Introduction

L'évolution constante des pratiques informatiques a conduit à l'émergence de l'environnement DevOps, où le développement logiciel et les opérations convergent pour accélérer la livraison des applications. Cette transition vers DevOps ne s'est pas effectuée sans susciter des défis significatifs en matière de sécurité informatique. Et pour faire face à ces défis de nouvelles pratiques comme la Politique en tant que Code ont aussi vu le jour. Dans le présent chapitre, nous ferons une étude des principaux axes de notre sujet. Cela nous permettra de donner une meilleure orientation à la suite de nos travaux.

1.1 Concept de DevOps

1.1.1 Qu'est-ce que DevOps ?

Le terme « DevOps » est la contraction des mots anglais « Development » (développement) et « Operations » (exploitation). Il reflète toutefois un ensemble d'idées et de pratiques bien plus large que ces deux seuls mots pris ensemble ou séparément. Le DevOps décrit des approches d'accélération des processus selon lesquels une idée (par exemple, une nouvelle fonction logicielle, une demande d'amélioration ou la correction d'un bogue) passe de la phase de développement à celle du déploiement dans un environnement de production où elle peut apporter de la valeur à l'utilisateur. Ces approches nécessitent que les équipes de développement et d'exploitation communiquent fréquemment entre elles et abordent leur travail en gardant à l'esprit celui de leurs collaborateurs[16].
En plus simple, ce terme est utilisé pour décrire un ensemble de philosophies culturelles, de pratiques et d'outils qui unifie le développement logiciel et les opérations informatiques, ce qui augmente la capacité des organisations à distribuer des applications et des services très rapidement [8]. L'objectif principal du DevOps est d'automatiser et d'accélérer le processus de livraison des logiciels, tout en garantissant la stabilité et la fiabilité des systèmes.

Un peu d'historique : le terme DevOps a été utilisé pour la première fois par Patrick Debois et Andrew Shafer dans leur conférence Agile infrastructure, lors de la 2008 Agile Toronto conference.

Les principaux aspects de DevOps incluent :

- **Collaboration** : Le DevOps encourage la collaboration étroite entre les équipes de développement, d'exploitation et d'autres parties prenantes impliquées dans le cycle de vie du logiciel.
- **Automatisation** : L'automatisation est au cœur de DevOps. Elle vise à éliminer les tâches manuelles répétitives, accélérant ainsi le déploiement des logiciels et réduisant les risques d'erreurs humaines.
- **Livraison continue** : Le DevOps favorise la livraison continue, permettant aux équipes de livrer fréquemment des versions de logiciels fonctionnelles et stables.
- **Infrastructure en tant que Code (Infrastructure en tant que Code (IaC))** : Cette approche consiste à gérer et à provisionner l'infrastructure informatique de manière automatique, en utilisant des scripts et des configurations définies en tant que code.
- **Surveillance** : Le DevOps met l'accent sur la surveillance continue des performances et des incidents, ainsi que sur la collecte de données pour améliorer en permanence les processus[7].

En adoptant le DevOps, les organisations cherchent à réduire les silos entre les équipes, à accélérer le développement de logiciels, à améliorer la qualité des produits, et à répondre de manière plus agile aux exigences changeantes du marché.

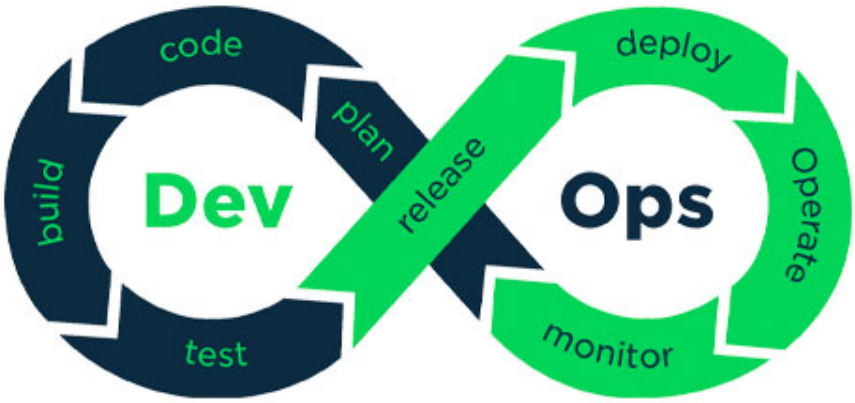


FIGURE 1.1 : Schéma de la boucle DevOps [20].

1.1.2 L'Approche CI/CD

L'approche CI/CD est une pratique clé dans le contexte de DevOps. Elle vise à automatiser et à accélérer le processus de développement, de test et de déploiement des logiciels. En d'autres termes, elle permet d'augmenter la fréquence de distribution des applications grâce à l'introduction de l'automatisation au niveau des étapes de développement des applications. Plus précisément, il s'agit de garantir une automatisation et une surveillance continues tout au long du cycle de vie des applications, des phases d'intégration et de test jusqu'à la distribution et au déploiement [18]. En s'appuyant sur l'intégration continue (CI), les développeurs fusionnent les changements de code au sein d'un référentiel plusieurs fois par jour, et les changements sont automatiquement intégrés aux versions.

Grâce à la livraison continue, le code se trouve toujours dans un état « déployable » permettant ainsi de le déployer dans un environnement de production à tout moment.

1.2 Sécurité des infrastructures informatiques en environnement DevOps

1.2.1 Qu'est-ce qu'une infrastructure informatique ?

L'infrastructure informatique est l'ensemble des logiciels, du matériel, des réseaux et des services connectés qui constituent l'environnement informatique d'une organisation. Elle comprend tout ce dont vous avez besoin pour créer et exécuter des applications logicielles dans une organisation. Un large éventail de composants constitue l'ensemble du système d'infrastructure informatique [3]. L'approche DevOps vise à automatiser et à orchestrer ces composants de manière à favoriser le déploiement continu et la livraison agile de logiciels.

1.2.2 Défis relatifs à la sécurité des infrastructures informatiques en environnement DevOps

Malgré les nombreux avantages qu'il offre, le DevOps présente de nouveaux risques et des changements culturels qui posent des problèmes de sécurité difficiles à résoudre en utilisant des solutions de gestion de la sécurité et des pratiques traditionnelles. Souvent, ces approches traditionnelles se révèlent trop lentes, trop coûteuses ou trop complexes pour accompagner l'automatisation de la distribution et du déploiement de logiciels. Parmi ces défis, on peut relever :

- **Identifiants et accès à privilèges :** Les attaquants ciblent fréquemment les identifiants à privilèges utilisés en DevOps. De ce fait, l'un des plus grands défis de sécurité dans les environnements DevOps est la gestion des accès à privilèges. Les processus DevOps nécessitent l'utilisation d'identifiants à privilèges humains et non-humains qui sont très puissants, et donc plus vulnérables aux cyberattaques. Une fois que les attaquants ont réussi à obtenir des identifiants à privilèges, ils peuvent compromettre l'ensemble des pipelines DevOps, accéder à des bases de données sensibles, voire infiltrer l'ensemble de l'infrastructure de l'organisation cible. Les attaquants en sont conscients et intensifient leurs efforts pour dénicher des identifiants à privilèges tels que mots de passe, clés d'accès, clés SSH ou tokens, ainsi que d'autres types de secrets tels que certificats, clés de chiffrement ou clés Interface de Programmation d'Application (API). L'utilisation d'identifiants non sécurisés dans les environnements DevOps expose à des risques tels que des fuites de données et la destruction de propriété intellectuelle[8].

- **Priorité à la rapidité plutôt qu'à la sécurité :** Les équipes mettent l'accent sur la rapidité plutôt que sur la sécurité. Les équipes DevOps se focalisent sur la rapidité de production du code, adoptant souvent des pratiques mal sécurisées qui échappent à la vigilance des équipes de sécurité. Parmi ces pratiques, on observe parfois l'inclusion de secrets et d'identifiants dans les applications et les fichiers de configuration, la réutilisation de code tiers sans précautions suffisantes, l'adoption de nouveaux outils sans évaluation approfondie de leurs problèmes potentiels en matière de sécurité, ou encore une protection insuffisante des outils et de l'infrastructure DevOps[8].

- **Gestion des secrets centrée sur les outils :** Les outils DevOps intègrent fréquemment des fonctionnalités destinées à la protection des secrets. Cependant, ces fonctionnalités ne favorisent pas toujours l'interopérabilité ni le partage sécurisé des secrets entre les outils et les plateformes. Souvent, les équipes DevOps se reposent sur les fonctionnalités intégrées de leurs outils pour la gestion des secrets. Cette approche peut compliquer la tâche de protection adéquate des secrets, car il devient difficile de les surveiller et de les gérer de manière homogène[8].

- **Erreurs de Configuration :** Les changements fréquents dans le code et l'utilisation d'outils d'automatisation peuvent conduire à des erreurs de configuration. Ces erreurs peuvent créer des failles de sécurité non intentionnelles, mettant en danger la confidentialité et l'intégrité des données[8].

- etc.

1.2.3 Évolution des pratiques de sécurité dans DevOps

Dans ce contexte dynamique de DevOps et face aux différents défis, l'évolution des pratiques de sécurité est nécessaire pour garantir une protection continue tout au long du cycle de vie du développement. Les acteurs concernés implémentent donc des mesures pour développer la sécurité du DevOps à grande échelle tout en gérant les risques liés à la sécurité et en s'alignant sur la culture et les méthodes DevOps :

- **Automatisation des Contrôles de Sécurité :** L'automatisation devient un pilier essentiel pour garantir la rapidité et la cohérence des contrôles de sécurité. Des outils d'automatisation sont utilisés pour vérifier en continu la conformité aux politiques de sécurité prédéfinies, permettant des corrections rapides en cas de détection d'anomalies.

- **Collaboration Étroite entre les Équipes :** La sécurité n'est plus la seule responsabilité d'une équipe distincte. La collaboration étroite entre les équipes de développement, d'exploitation et de sécurité est cruciale. Cela se traduit par des cycles de développement sécurisés, où les implications de la sécurité sont prises en compte dès la phase de conception.

- **Mentalité de "Shift Left"** : L'adoption d'une mentalité "Shift Left" signifie que la sécurité est déplacée vers les phases les plus précoces du cycle de développement. Elle permet d'effectuer des tests et de corriger les bugs à un stade plus précoce du processus de développement grâce au pipeline CI/CD[7].

- **Politique en tant que Code :** La transition vers la Politique en tant que Code permet d'exprimer les politiques sous forme de code informatique. Cela favorise une gestion plus souple, versionnée et automatisée des politiques, alignée avec les principes de DevOps.

- etc.

Ces pratiques de sécurité visent à intégrer la sécurité de manière transparente dans l'environnement DevOps, permettant des déploiements plus rapides sans compromettre la robustesse des systèmes et infrastructures.

Notre étude est axée sur l'une de ces pratiques, celle de la politique en tant que code.

1.3 Politique en tant que Code

1.3.1 Explication du concept de Politique en tant que Code

Pour mieux comprendre ce que signifie la politique en tant que code, il est nécessaire de comprendre d'abord la définition d'une « politique ».

Dans ce contexte, une politique désigne tout type de règle, condition ou instruction qui régit les opérations ou processus informatiques. Une politique peut être une règle définissant les conditions à remplir pour qu'un code réussisse un contrôle et soit déployé, par exemple. Ou bien, il peut s'agir d'un ensemble de procédures exécutées automatiquement en réponse à un événement [15].

La politique en tant que code est une approche de gestion des politiques dans laquelle les politiques sont définies, mises à jour, partagées et appliquées à l'aide de code [17]. Cela signifie que les règles qui régissent le comportement des systèmes, les paramètres de sécurité, les configurations système, les instructions de nommage et d'autres aspects de l'infrastructure informatique sont définies de manière formelle et automatisée, souvent en utilisant des langages de programmation spécifiques tels que Python, YAML Ain't Markup Language (YAML), etc. Le langage spécifique dépend généralement des outils de gestion et d'application de la politique en tant que code que vous utilisez.

1.3.2 En quoi consiste l'approche de la politique en tant que code ?

L'approche de la Politique en tant que Code repose sur la formalisation et l'automatisation des politiques de sécurité à travers l'utilisation de langages de programmation ou de formats déclaratifs. Elle implique trois composantes principales : **Politique, Donnée, et Requête.**

- **Politique :** La politique représente les règles, directives ou spécifications qui définissent le comportement souhaité ou autorisé dans un système. Ces politiques peuvent couvrir divers aspects de la sécurité informatique, tels que le contrôle d'accès, la gestion des configurations, la conformité aux normes, etc.
- **Données :** Les données correspondent aux informations sur lesquelles les politiques sont appliquées. Cela peut inclure des données utilisateur, des configurations système, des paramètres de sécurité, des données sur un environnement ou un service.
- **Requête :** Une requête déclenche le processus de prise de décision sur la base des données disponibles et de la politique saisie vers le moteur de politique.

En résumé, un moteur de politique prend en compte la politique à utiliser dans l'évaluation et les données à évaluer, suite à une requête. Ensuite, le moteur de politique évalue les données fournies par rapport à la politique fournie et produit une réponse utilisée par le système auquel le moteur de politique est intégré ou auquel il sert. Les politiques sont rédigées pour correspondre aux données qu'elles évaluent.

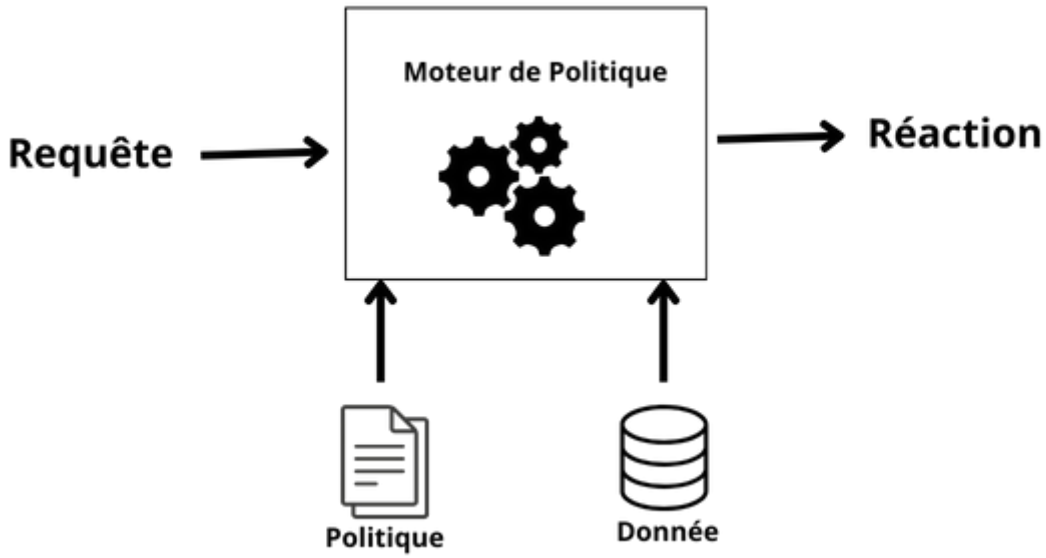


FIGURE 1.2 : Schéma résumant le fonctionnement.

1.3.3 Avantages de la politique en tant que code

L'approche de la Politique en tant que Code présente plusieurs avantages significatifs dans la gestion des politiques et des configurations opérationnelles. Voici quelques-uns des principaux avantages de cette approche :

- **Codification et reproductibilité :** La codification des politiques permet aux entreprises d'adopter les meilleures pratiques établies en matière de développement logiciel pour créer et mettre à jour leurs politiques. Ces bonnes pratiques sont cruciales pour garantir la réflexion approfondie des politiques, ainsi que leur facilité de test et de gestion. Cette codification des politiques permet aussi de reproduire exactement les mêmes règles sur différents environnements, assurant ainsi une cohérence et une uniformité au sein de l'infrastructure[15].

- **Automatisation :** L'adoption de la PaC améliore l'efficacité de la création et de la gestion des politiques, encourage l'automatisation des tests et du déploiement, tout en réduisant la dépendance à l'égard des interventions manuelles. Aussi, les politiques exprimées en tant que code peuvent être intégrées dans des pipelines CI/CD pour automatiser les contrôles de conformité. Cela permet de garantir la conformité des déploiements tout au long du processus de développement[15].

- **Réactivité aux changements :** La PaC offre une agilité accrue, permettant une adaptation rapide des politiques en fonction des évolutions des exigences de sécurité, des nouvelles menaces, ou des changements dans l'infrastructure[15].

- **Traçabilité et historique :** Comme le code source, les politiques en tant que code peuvent être versionnées, fournissant une traçabilité complète des changements. Cela facilite la gestion de l'historique des politiques et la compréhension des évolutions[15].

- **Validation avant déploiement :** Le problème lorsque les politiques sont testées seulement après le déploiement est désormais du passé. Avec la bonne solution PaC, vous pouvez tester son fonctionnement lors de la session de prévisualisation. Les politiques commencent à fonctionner avant même que les ressources ne soient enregistrées et que tous les paramètres soient

soigneusement validés. Les ressources qui enfreignent la politique sont supprimées et modifiées. Dans les cas où vous devez vérifier une ressource particulière séparément des autres, cela peut également être fait avec un outil PaC approprié. D'une manière ou d'une autre, l'ensemble de l'infrastructure est testé et validé avant le déploiement, ce qui permet d'économiser considérablement du temps et de l'argent[15].

- **Collaboration :** En fournissant un moyen uniforme et systématique de gérer les politiques, la politique en tant que code simplifie la collaboration. Cela inclut la collaboration non seulement au sein de la même équipe, mais aussi entre différents types d'équipes, en particulier entre les développeurs (qui ont l'habitude de penser et de travailler en termes de code) et les spécialistes d'autres domaines, tels que la sécurité ou les opérations informatiques[15].

1.3.4 Politique en tant que Code VS Infrastructure en tant que Code (IaC)

L'IaC, représente une méthode d'ingénierie logicielle qui implique la définition et la gestion de la configuration de l'infrastructure au moyen d'un code. Ce code peut être automatiquement déployé à l'aide d'outils tels qu'Ansible ou Terraform.
L'IaC offre une automatisation, une gestion de version et une cohérence dans le provisionnement de l'infrastructure. En parallèle, la PaC permet d'écrire les politiques dans le code afin d'assurer une application uniforme de ces politiques à travers tous les environnements.
L'IaC et la PaC, qui sont des concepts complémentaires, visent essentiellement à automatiser et uniformiser la mise en œuvre pour développer des flux de travail efficaces et cohérents. Cependant, elles diffèrent en matière d'objectifs :

- **L'IaC définit et déploie les ressources de l'infrastructure :** L'IaC se concentre sur l'automatisation de la gestion de l'infrastructure informatique dans son ensemble, y compris les serveurs, les réseaux, les bases de données, etc.

- **La PaC définit et met en œuvre des politiques :** La PaC se concentre sur l'expression formelle et automatisée des politiques, des règles opérationnelles et des configurations spécifiques au niveau des applications, des services, et de l'ensemble de l'infrastructure[6].

1.4 Solutions pour la Mise en œuvre de la Politique en tant que Code

La mise en œuvre de la Politique en tant que Code nécessite l'utilisation d'outils et de technologies pour exprimer, automatiser et gérer les politiques de sécurité de manière efficace. En réalité, plusieurs outils populaires d'automatisation, de gestion de configuration et de CI/CD offrent des fonctionnalités pouvant contribuer à la mise en place d'une approche PaC. Quelques-uns des outils utilisés pour la mise en œuvre de la PaC :

TABLE 1.1 : Tableau des outils de PaC (moteurs et langages de politique)

	Moteur de politique	Langage de politique	Applications
Cloud Custodian	Cloud Custodian (c7n)	YAML	Cloud (Amazon Web Services (AWS), Azure, , etc.)
jsPolicy	jsPolicy	JavaScript - TypeScript	Kubernetes
k-rail	k-rail	Golang	Kubernetes
KubeWarden	KubeWarden	Rust	Kubernetes
Kyverno	Kyverno	YAML	Kubernetes, CI/CD pipeline
MagTape	MagTape	Rego	Kubernetes
Open Policy Agent	Open Policy Agent	Rego	Kubernetes, Docker, Envoy, GKE, CI/CD pipeline, Terraform, AWS, etc.
Gatekeeper	Gatekeeper	Rego & YAML	Kubernetes
Sentinel	Sentinel	HCL	Terraform, Cloud, CI/CD pipeline, etc.
Pulumi CrossGuard		JavaScript - TypeScript - Python	AWS, Azure, , Kubernetes, etc.

1.5 Choisir la solution PaC adéquate

1.5.1 Facteurs de sélection à prendre en compte

Le choix de la solution Politique en tant que Code dépend de divers facteurs et aspects, notamment les besoins spécifiques de l'organisation, les technologies utilisées, et les objectifs en matière de sécurité. Ces facteurs peuvent se résumer en :

- **Nature des Politiques :** Il faut examiner les types de politiques qui seront automatisées. Certains outils PaC sont spécialisés dans des domaines spécifiques tels que le contrôle d'accès, la gestion des configurations, ou la conformité aux normes.
- **Environnement Technologique :** Prendre en considération les technologies et les plateformes que l'organisation utilise. Certains outils PaC sont spécifiquement conçus pour fonctionner avec des environnements tels que Kubernetes, Terraform, ou des services cloud spécifiques.
- **Intégration avec les Outils Existant :** Évaluer la capacité de l'outil PaC à s'intégrer harmonieusement avec les outils et les processus existants, tels que les pipelines CI/CD, les systèmes de gestion de versions, ou les outils de surveillance.
- **Flexibilité et Adaptabilité** : Il faut penser à une solution PaC qui offre une flexibilité suffisante pour s'adapter aux besoins changeants de l'organisation. Une solution capable de couvrir un large éventail de politiques est souvent préférable.
- **Support de la Communauté :** Tenir compte de la communauté d'utilisateurs et de contributeurs autour de l'outil. Une communauté active peut garantir des mises à jour régulières, des corrections de bugs, et une documentation étendue.

- **Sécurité de l'outil lui-même :** L'outil PaC choisi doit respecter les normes de sécurité et suivre les meilleures pratiques en matière de protection des données sensibles, surtout s'il est utilisé pour gérer des politiques de sécurité.
- **Facilité d'Utilisation :** Il faut opter pour une solution PaC qui offre une interface conviviale et des mécanismes de configuration clairs. Une adoption plus facile au sein de l'équipe facilite l'intégration de la PaC dans les processus de développement.
- **Évolutivité :** L'outil PaC choisi doit être capable de s'adapter à l'évolution des besoins de l'organisation en termes de nombre de politiques, de complexité et de diversité des configurations.
- **Conformité aux Normes :** Si l'organisation doit respecter des normes ou des réglementations spécifiques, il faut s'assurer que l'outil PaC permet de mettre en œuvre et de valider ces politiques de conformité. etc.

1.5.2 Méthode pour évaluer différentes solutions PaC

Une fois que les facteurs de sélection sont définis et compris, il est possible de déduire les critères de sélection et créer un tableau de notation. Les critères de sélection sont composés des facteurs de sélection relatifs à la situation, annotés d'un poids. Le poids du facteur indique à quel point le facteur individuel est important pour l'organisation. Pour les facteurs d'importance nulle, il est possible soit pondérer le facteur en conséquence (avec un zéro), soit le laisser complètement de côté dans le tableau de notation. Le score d'adéquation, est explicite ; il indique dans quelle mesure le critère de sélection de la solution PaC correspond aux besoins.

Dans le tableau de notation (voir table 1.2), la colonne Poids contient des valeurs de 1 à 3, indiquant laquelle des trois classifications (devrait-avoir, plaisant-à-avoir, doit-avoir) est appliquée aux facteurs de sélection, pour former les critères de sélection choisis pour l'évaluation. Le tableau de notation inclut des évaluations pour deux solutions, côte à côte. Vous multipliez les colonnes Adéquation dans chaque section d'évaluation par la colonne Poids commune pour déterminer le score individuel de chaque critère de sélection. La colonne Adéquation utilise des valeurs de 1 à 5. Pour indiquer une plus grande différence dans les évaluations, vous pourriez élargir les plages dans les colonnes Poids et Adéquation, ou même remplacer les nombres en base 10 par des nombres de Fibonacci [1].

TABLE 1.2 : Exemple de tableau de note pour deux solutions.

Critère de Sélection		Solution PaC 1		Solution PaC 2	
Facteurs	Poids	Adéquation	Score	Adéquation	Score
Alignement-Capacité de l'organisation	3	3	9	1	3
Alignement-Stratégie de l'organisation	3	2	6	3	9
Alignement- Standards de l'organisation	3	5	15	2	6
Exemples et modèles disponibles	2	4	8	5	10
Support de la communauté	1	5	5	5	5
Complexité	2	3	6	4	8
Documentation	3	5	15	5	15
Récence du projet	2	5	10	5	10
Modes de fonctionnement	3	5	15	4	12
Extensibilité de la solution	3	5	15	2	6
Cas d'utilisation	3	5	15	5	15
Expérience Utilisateur	3	4	12	4	12
Flexibilité et Adaptabilité	2	4	8	4	8
Conformité aux Normes	3	4	12	5	15
Maturité de la Solution	1	2	2	5	5
			135		127

Conclusion

Dans ce chapitre, nous avons exploré le concept de DevOps pour mettre en lumière les défis auxquels les organisations sont confrontées pour leurs infrastructures informatiques ainsi que l'évolution de la sécurité des infrastructures informatiques dans un environnement DevOps. Nous avons examiné l'approche de la Politique en tant que Code comme une réponse innovante aux défis. En mettant l'accent sur la formalisation des politiques, la gestion des données et l'évaluation automatisée des requêtes, la PaC se positionne comme une solution agile et efficace pour renforcer la sécurité tout en s'adaptant aux besoins dynamiques des environnements DevOps.
Dans le prochain chapitre, nous approfondirons notre compréhension de la Politique en tant que Code en réalisant un exemple concret de politiques de sécurité exprimées en code pour un pipeline CI/CD DevOps spécifique. Pour cela nous aurons également à explorer une solution spécifique utilisée pour mettre en œuvre la PaC.

Chapitre 2

Pipeline et Solution PaC

Introduction

Les pipelines CI/CD constituent le socle des pratiques DevOps, permettant une livraison logicielle continue. Dans ce chapitre, nous verrons la structure et le fonctionnement d'un pipeline CI/CD spécifique. Nous explorerons les différentes composantes qui le composent, depuis la gestion du code source jusqu'au déploiement automatisé, en passant par les étapes critiques de tests et de validation. Ensuite, nous choisirons une solution PaC, l'étudierons en profondeur, puis rédigerons des politiques de sécurité documentées pour le pipeline. Enfin, nous utiliserons cet outil pour traduire ces politiques en code.

2.1 Présentation du pipeline CI/CD

2.1.1 Scénario pour le pipeline

Imaginons qu'il y ait une équipe de développement logiciel Agile qui effectue un développement continu de leur produit. Ils apportent des changements continus au code. Ce code doit être empaqueté et déployé sur des serveurs pour les tests. Avec un tel processus continu et des commits de code continus, ils doivent déployer en continu l'artefact sur les serveurs pour les tests. Et lorsque tous les tests sont concluants, alors ils peuvent enfin déployer en production. Donc après un déploiement, les tests logiciels et d'autres types de tests sont effectués et les rapports de test sont évalués. Une approbation est donnée avant de passer en production ou non. C'est un processus de déploiement, de test et de déploiement encore, car il y a différents environnements : Dev, QA, UAT, production, staging. Avec ce déploiement continu, il y a également une dépendance continue envers l'équipe opérationnelle. Différentes équipes ou différents propriétaires de produits de différents projets dépendront de l'équipe opérationnelle pour les déploiements réguliers. Chaque projet aura des environnements Dev, QA, staging, production, plusieurs environnements. Il y a donc des demandes continues de déploiement envoyées à l'équipe opérationnelle.

2.1.2 Problème

Dans ce scénario, un problème subsiste. Si le processus de déploiement du code est manuel, cela prendra beaucoup de temps pour promouvoir les changements en production. Peu importe à quelle vitesse l'équipe effectue le changement de code, le déploiement sera lent, car il y a plusieurs environnements où le déploiement doit être effectué.

Aussi, dans ce processus de déploiement, il doit avoir des systèmes de ticketing, des approbations, des affectations, différents propriétaires pour différents environnements. Le développeur détient l'environnement de développement, les testeurs logiciels détiennent l'environnement de QA, et l'équipe de surveillance des ingénieurs en fiabilité du site (SRE) détient l'environnement de production. Il y aura trop de tickets, d'affectations et d'approbations ; ce qui pose un problème dans un processus Agile. Trop de dépendances sur les équipes opérationnelles, de développement et de mise en production pour chaque déploiement.

2.1.3 Solution

La solution serait d'abord d'établir un processus d'intégration continue et un processus de livraison continue. Ainsi, pour chaque commit de code, développer, tester, déployer, puis tester.
Cela doit se produire pour chaque commit et le processus de déploiement doit être automatisé. Le déploiement sur différents environnements avec différents propriétaires doit être notifié en cas de réussite ou d'échec. Il y aura plusieurs équipes et aussi plusieurs outils. Donc tous les outils d'automatisation doivent être intégrés ensemble : l'outil d'intégration continue intégré avec les dépôts, intégré avec les outils d'automatisation, avec les outils de gestion de configuration, etc. Lorsque ces intégrations sont effectuées, cela élimine l'intervention humaine. En cas de problème, de bogue dans le code ou dans le processus de déploiement, le processus s'arrête automatiquement et le problème est corrigé, puis le processus se répète. C'est tout ce processus automatisé qui s'appelle le processus de livraison continue.
Ce pipeline est un exemple de mise en œuvre de la solution avec l'intégration de différents outils ensemble.

2.1.4 Principaux outils/technologies du pipeline

2.1.4.1 Jenkins

Jenkins est un logiciel open source écrit en Java. Il s'agit d'un outil d'intégration, livraison et déploiement continus (CI/CD) et d'automatisation des différentes étapes du développement logiciel telles que le build, le test et le déploiement. Jenkins est utilisé par de nombreuses entreprises pour automatiser les processus de développement logiciel, en particulier dans le cadre de la méthodologie Agile. Ici, il sera utilisé pour l'intégration continue et il sera connecté à tous les outils, notamment Github, où notre code source est stocké[9].



FIGURE 2.1 : Logo de Jenkins

2.1.4.2 Git

Git est un système de contrôle de version qui a été inventé et développé par Linus Torvalds, également connu pour l'invention du noyau Linux, en 2005. Il s'agit d'un outil de développement qui aide une équipe de développeurs à gérer les changements apportés au code source au fil du temps. Git est le plus connu des , c'est un projet open source très puissant qui est utilisé par l'ensemble de la communauté des développeurs[13].



FIGURE 2.2 : Logo de Git

2.1.4.3 Apache Maven

Apache Maven (couramment appelé Maven) est un outil open source de la communauté Apache entièrement écrit en Java. Il permet d'automatiser la gestion et la construction d'un projet Java : ce que l'on appelle communément un outil de build. Il permet aussi de gérer des dépendances vis-à-vis des bibliothèques nécessaires à un projet, de générer des documentations concernant un projet, d'exécuter des tests unitaires. Maven se présente comme un exécutable en ligne de commande, mais il est également intégré nativement dans les IDE les plus courants dans le monde Java : Eclipse, Intellij IDEA, NetBeans[21].



FIGURE 2.3 : Logo de Apache Maven

2.1.4.4 CheckStyle

Checkstyle est un outil d'analyse statique de code pour Java. À l'origine conçu pour faire respecter un ensemble de normes de codage hautement configurable, Checkstyle permet aussi maintenant de vérifier les mauvaises pratiques de codage, ainsi que le code trop complexe ou dupliqué. C'est un outil open source, polyvalent et flexible qui propose de puissantes fonctionnalités pour appliquer des contrôles sur le respect de règles de codifications. Checkstyle supporte un très grand nombre de règles, incluant celles liées aux normes de nommage, annotations, commentaires javadoc, taille de méthode et de classe, mesures de complexité de code, mauvaises pratiques de codage, et beaucoup d'autres[11].



FIGURE 2.4 : Logo de Checkstyle

2.1.4.5 Slack

Slack est une plateforme de communication collaborative largement utilisée dans les environnements professionnels. Elle permet aux équipes de travailler ensemble de manière efficace en facilitant la communication, la collaboration et le partage d'informations. Slack offre des fonctionnalités telles que la messagerie instantanée, la création de canaux thématiques, le partage de fichiers, l'intégration avec d'autres outils et applications, et la possibilité de rechercher facilement des informations dans les discussions passées. Pour ce pipeline, il nous sera utile pour les notifications[4].



FIGURE 2.5 : Logo de Slack

2.1.4.6 Sonatype Nexus Repository

Nexus est une plateforme de gestion de dépôts développée sous licence GPL par la société Sonatype permettant d'héberger des artéfacts. Ces artéfacts sont des composants, générés par exemple au build d'un projet, et déposés ensuite sur Nexus. L'intérêt de Nexus est de pouvoir y partager des artéfacts avec les autres développeurs d'un projet, voire avec toute une communauté. Il est disponible en version gratuite, dite "communautaire", ou payante selon les besoins[19].



FIGURE 2.6 : Logo de Nexus Repository

2.1.4.7 SonarQube

Autrefois appelé Sonar, SonarQube est un logiciel libre, sous licence GNU LGPL, qui sert à analyser et mesurer continuellement la qualité du code source. Il peut analyser plus de vingt-cinq langages tels que JAVA, Objective-C, PHP et tant d'autres. L'objectif de SonarQube est de donner des indicateurs « simplifié » pour comprendre comment « va un projet » / afin d'en mesurer sa santé. Cette santé est exprimée sous la forme de 4 grandes notes : le nombre de « bugs », le nombre de vulnérabilités, le nombre de points chauds/sensibles à vérifier, la dette technique[12].



FIGURE 2.7 : Logo de SonarQube

2.1.4.8 Ansible

Ansible est un outil open source de gestion de configuration, d'automatisation et de déploiement d'application. Il est conçu pour simplifier et automatiser la gestion de configurations complexes sur des infrastructures distribuées. Devenu rapidement un incontournable dans le monde DevOps ces dernières années, il a été adopté par de nombreuses entreprises, à la fois en raison de sa simplicité d'utilisation, de configuration, mais aussi, car sa personnalisation lui permet d'être autonome quel que soit l'environnement utilisé (Cloud public, On-Premise, etc)[5].



FIGURE 2.8 : Logo de Ansible

2.1.4.9 Selenium

Selenium est un framework web qui permet d'exécuter des tests multi-navigateurs. Cet outil est utilisé pour automatiser les tests d'applications Web pour vérifier qu'ils fonctionnent correctement, fournit une plateforme pour exécuter des tests sur différents navigateurs et systèmes d'exploitation, ce qui le rend très utile pour les équipes de développement logiciel suivant les pratiques Agile et DevOps[2].



FIGURE 2.9 : Logo de Selenium

2.1.4.10 Apache Tomcat

Apache Tomcat est un conteneur web libre (logiciel de serveur d'applications web open source) de servlets et JSP Java EE. Issu du projet Jakarta, c'est un des nombreux projets de l'Apache Software Foundation. Il implémente les spécifications des servlets et des JSP du Java Community Process, est

paramétrable par des fichiers XML et des propriétés, et inclut des outils pour la configuration et la gestion. Il comporte également un serveur HTTP. Il est, lui-même écrit en Java[10].



FIGURE 2.10 : Logo de Apache Tomcat

2.1.5 Architecture du pipeline

Examinons l'architecture du pipeline CI/CD.

Donc, les développeurs effectuent régulièrement des changements de code, comme dans un processus de développement Agile. Ils font le commit de ces changements dans le référentiel Git local, qui est ensuite synchronisé avec un référentiel distant. Nous utiliserons GitHub comme référentiel, où le code source est stocké. Chaque fois qu'un nouveau code est modifié, à chaque commit sur GitHub, Jenkins le détecte automatiquement, récupère tous les changements, puis effectue des tests de code et des tests unitaires. En fonction de la réussite ou de l'échec, des notifications sont envoyées. Si le test réussit, le processus passe à l'analyse de code, utilisant Checkstyle et le scanner SonarQube. Ces outils analysent le code, génèrent un rapport et l'envoient au serveur SonarQube. Le serveur SonarQube le compare avec ses critères de qualité. Si le critère est respecté, le processus continue, sinon il s'arrête avec une notification d'échec.

Si tout est en ordre, le processus passe à l'étape suivante, où Maven construit l'application, effectue des tests unitaires, et télécharge les dépendances depuis le référentiel Nexus. Le processus s'arrête en cas d'échec, mais si tout passe, l'artefact est empaqueté et envoyé dans le référentiel Nexus. À ce stade, l'artefact a passé les tests unitaires et l'analyse de code. Terraform provisionne les serveurs de staging et production. Si une modification doit être effectuée au niveau d'un serveur ou si un serveur doit être supprimé et récrée, il le fait.

Ensuite, Ansible entre en jeu pour déployer l'artefact sur le serveur de staging. Ansible s'assure que les bons paquets sont installés, que les services fonctionnent correctement, et que les configurations sont correctes. Ansible peut également effectuer un rollback en cas d'échec lors du déploiement, offrant ainsi la possibilité de revenir à l'état précédent.

Une fois le déploiement sur le serveur de staging terminé, la suite de tests logiciels peut être exécutée pour tester l'application dans un environnement similaire à la production. Si les tests réussissent et que l'approbation est obtenue, Ansible peut ensuite déployer l'artefact sur le serveur de production.

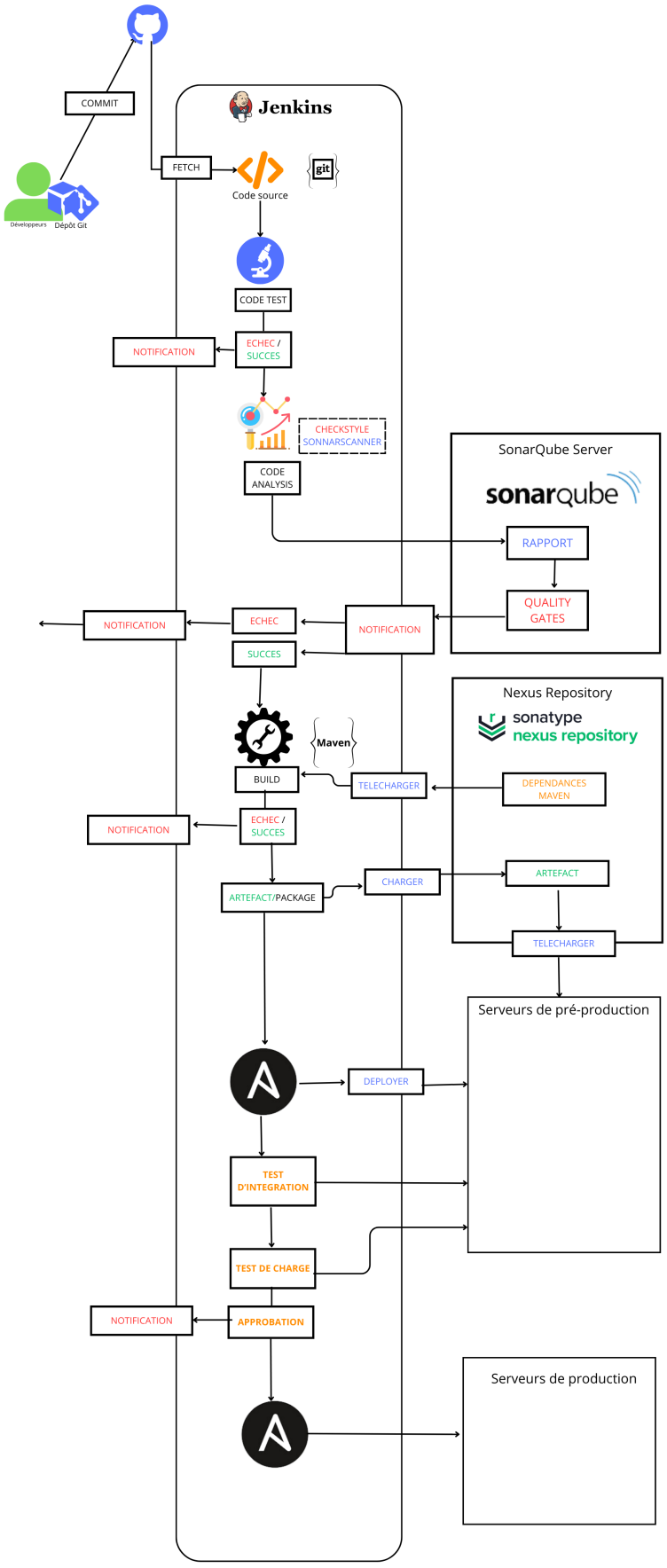


FIGURE 2.11 : Schéma de l'Architecture du pipeline

2.2 Présentation de la solution PaC choisie

La solution PaC qui sera utilisée pour la mise en œuvre de la politique en tant que code pour le pipeline présenté ci-dessus est : **Open Policy Agent**.

2.2.1 Qu'est-ce que Open Policy Agent ?

Open Policy Agent (OPA) (prononcé « oh-pa ») est un moteur de politique open source polyvalent qui unifie l'application des politiques à travers l'ensemble de la pile technologique[14]. OPA propose un langage déclaratif de haut niveau (Rego) vous permettant de spécifier la politique en tant que code, ainsi que des API simples pour décharger la prise de décision politique de votre logiciel. Vous pouvez utiliser OPA pour appliquer des politiques dans des microservices, Kubernetes, des pipelines CI/CD, des passerelles API, Terraform, AWS, Docker, Envoy, GKE et bien plus encore.

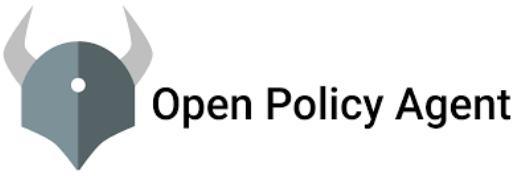


FIGURE 2.12 : Logo de Open Policy Agent

2.2.2 Pourquoi utiliser OPA ?

Voici quelques raisons pour lesquelles choisir d'utiliser Open Policy Agent :

- **Flexibilité des politiques** : OPA permet de définir des politiques de manière déclarative à l'aide du langage de règles de politique appelé Rego. Cela offre une grande flexibilité pour exprimer des politiques complexes et spécifiques à l'application ou l'environnement.
- **Support multi-environnement** : OPA peut être utilisé pour appliquer des politiques de sécurité dans divers environnements, tels que les conteneurs, les microservices, les API, les infrastructures en tant que code et bien d'autres. Cela en fait un outil polyvalent pour la gestion des politiques de sécurité dans un large éventail de scénarios.
- **Intégration facile** : OPA peut être intégré dans des systèmes existants en tant que composant externe, ce qui facilite son adoption. Il propose des API RESTful et des bibliothèques client dans plusieurs langages de programmation, ce qui permet une intégration transparente avec différents types d'applications.
- **Réduction de la complexité** : En utilisant OPA , il devient possible de centraliser la logique de politique de sécurité, réduisant ainsi la complexité de la gestion des politiques dans différents services ou composants logiciels. Cela simplifie également la maintenance et l'évolution des politiques de sécurité[14].

2.2.3 Comment OPA fonctionne ?

OPA sépare la prise de décision en matière de politique d'un service. Lorsque le système doit prendre des décisions de politique, il interroge OPA et fournit des données structurées (par exemple, JavaS-

cript Object Notation (JSON)) en tant qu'entrée. OPA accepte ces données structurées arbitraires en tant qu'entrée. Ensuite, il génère des décisions de politique en évaluant l'entrée de la requête par rapport aux politiques et aux données. OPA et son langage de politique Rego ne sont pas restreints à un contexte particulier ou à des types de problèmes spécifiques. Ils sont agnostiques par rapport au domaine, permettant ainsi de décrire presque n'importe quel type d'invariant dans les politiques. Par exemple :

- Quels utilisateurs peuvent accéder à quelles ressources.
- À quelles sous-réseaux le trafic de sortie est autorisé.
- Dans quels clusters une charge de travail doit être déployée.
- À partir de quels registres les binaires peuvent être téléchargés.
- Avec quelles capacités du système d'exploitation un conteneur peut être exécuté.
- À quelles heures du jour le système peut être accessible.

Par ailleurs, les décisions de politique ne se limitent pas à des réponses simples de type oui/non ou autoriser/refuser. Tout comme les entrées de requête, les politiques peuvent générer des données structurées arbitraires en tant que sortie[14].

2.2.4 Rego

Rego est un langage de politique déclaratif conçu pour spécifier des politiques complexes utilisées dans le domaine de la prise de décision automatisée. Il a été développé en tant que partie intégrante du projet Open Policy Agent. Le principal objectif de Rego est de permettre la définition claire et expressive de règles qui peuvent être utilisées pour prendre des décisions en matière d'autorisation, de validation et d'autres aspects liés à la sécurité logicielle. Les autres objectifs incluent la simplicité, l'expressivité et la déclarativité. En utilisant Rego, les développeurs peuvent décrire des politiques de manière concise et compréhensible, facilitant ainsi la maintenance et la compréhension du code de politique. Certains avantages de l'utilisation de Rego comprennent la flexibilité dans la définition de règles complexes, la facilité d'intégration avec différentes applications grâce à OPA , et la possibilité de séparer les logiques de politique du code d'application, favorisant ainsi la modularité et la réutilisation du code[1].

2.2.4.1 Modèle de Document d'OPA

Avant de commencer avec Rego, il faut d'abord comprendre le modèle de document OPA . Lorsque l'on travaille avec les données, OPA modélise les données en deux documents : les documents de base et les documents virtuels. Les documents de base sont des données chargées à partir de sources externes dans OPA, et les documents virtuels sont des données créées par OPA , comme les décisions de politique.

2.2.4.2 Règles

Dans le modèle de document d'OPA , les règles définissent le "contenu des documents virtuels" grâce à l'évaluation des règles. Les politiques sont constituées de règles, et les règles contiennent des expressions. La politique présentée ci-dessous a une règle nommée hello. Cette règle a une valeur

par défaut de 'au revoir', qui est renvoyée en cas de condition fausse. Lorsque OPA évalue la règle, le document virtuel produit sera soit "hello" : "world" soit "hello" : "au revoir".

```
1
2   default hello := "au revoir" (1)
3   hello = "world" {           (2)
4     msg := input.message (3)
5     msg == "world"       (4)
6   }
```

(1) Valeur par défaut de la règle
(2) Tête de la règle
(3) Expression du corps de la règle - charger la valeur depuis l'entrée
(4) Expression du corps de la règle - comparer la valeur d'entrée à la valeur souhaitée

Comme on peut le voir ci-dessus, notre règle a une Valeur par défaut, une Tête et un Corps. L'élément Tête spécifie le nom de la règle, et dans notre cas, la valeur (facultative) de la règle à renvoyer en cas de condition vraie. Sans la valeur explicite "world", la règle renverra un booléen vrai lorsque toutes les déclarations de la règle sont vraies, même si le retour par défaut est une chaîne de caractères. L'élément Corps contient les expressions de la règle. Un "ET logique" (appliqué par un retour à ligne ou un point virgule) des expressions de règle est appliqué, de sorte que chaque expression qui est une comparaison doit être vraie pour que la règle soit vraie. La valeur par défaut de la règle hello, qui est "au revoir", est renvoyée si :

- Le document d'entrée n'existe pas
- Le champ input.message n'existe pas
- Le champ input.message ne correspond pas à la valeur de "world"

Pour appliquer une condition logique "OU", il faut plusieurs règles avec le même nom dans la Tête, comme indiqué ci-dessous. Ces règles multiples forment un ensemble de règles

```
1
2 hello = "world" {
3   msg := input.message
4   msg == "world"
5   from := input.from
6   from == "john"
7 }
8 hello = "me" {
9   msg := input.message
10  msg == "me"
11  from := input.from
12  from == "john"
13 }
```

Cette nouvelle version de la politique renverra la valeur vraie de 'world' ou 'me' si le champ 'msg' est égal à 'world' et que le champ 'from' est égal à 'john' OU si le champ 'msg' est égal à 'me' et que le champ 'from' est égal à 'john', respectivement. Il faut noter que les règles qui ne sont pas évaluées comme vraies renverront une valeur 'undefined', sauf si la règle ou l'ensemble de règles a une valeur par défaut définie. Une seule valeur par défaut peut être définie par l'ensemble de règles

2.2.4.3 Fonctions

Les fonctions Rego sont un excellent moyen d'appliquer le principe DRY (Don't Repeat Yourself) à Rego et de dédupliquer le code, tout en appliquant la modularité. L'exemple ci-dessous utilise la

fonction "build_return_msg" pour créer un message de retour.

```
1
2 import future.keywords.in
3 default hello := "goodbye"
4 hello = output {
5   input.method == "POST"
6   input.message in {"world", "planet"}
7   output := build_return_msg("Hello",input.from)
8 }
9 build_return_msg (msg, from) := result {
10   result := sprintf("%s, %s", [msg, from])
11 }
```

Il faut noter dans l'exemple ci-dessus, l'importation du package future.keywords.in. Il permet d'utiliser des opérateurs OPA plus récents comme 'in', tout en maintenant la compatibilité ascendante.

Si les fonctions ressemblent aux règles, c'est parce qu'elles le sont ; les fonctions ne sont rien d'autre que des règles qui prennent des arguments. Tout comme les règles, les fonctions ont une tête et un corps, et peuvent être dupliquées (même tête) pour être utilisées dans des conditions logiques de type OU.

2.2.4.4 Fonctions intégrées

En plus de permettre au développeur d'écrire ses propres fonctions, Rego inclut de nombreuses fonctions intégrées (plus de 140). Ces fonctions sont regroupées en catégories fonctionnelles, telles que Agrégats et Strings, et peuvent être facilement identifiées par leur syntaxe : <nom>(<arg-1>, ..., <arg-n>). Par exemple, les fonctions count, sum, product, min et max (Agrégats) ainsi que la fonction sprintf (Chaînes) sont des fonctions intégrées.

2.2.4.5 Objets, collections et compréhensions

Comme d'autres langages, Rego prend en charge les valeurs composées. Par exemple, les objets sont des paires clé/valeur non ordonnées, comme des maps.

```
1
2 #Les clés sont obligatoirement uniques
3 animals := {"a":"deer", "b":"moose", "c":"bear"}
```

Contrairement à certains langages où les tableaux sont strictement typés, les tableaux Rego peuvent contenir plusieurs types.

```
1
2 stuff := [1,"hello",null,{"a":"b"}]
```

Les ensembles contiennent des valeurs uniques et ne sont pas ordonnés.

```
1
2 # les valeurs sont obligatoirement uniques
3 material := {"metal","wood","cloth"}
```

Les ensembles peuvent également être des documents non indexés.

```
1
2 k := {"metal","wood","cloth"} == {"cloth","wood","metal"}
```

Les compréhensions peuvent être utilisées pour créer des valeurs composées. Par exemple, on peut utiliser une compréhension d'ensemble pour convertir un tableau de valeurs dupliquées en un ensemble de valeurs uniques.

```
1
2 m := ["metal","wood","cloth","metal"]
3 u := {z | z = m[_]}
```

Résultat :

```
1 "u": ["cloth","metal","wood"]
```

Avec les compréhensions d'ensemble ci-dessus, nous avons construit des collections d'ensembles en utilisant des sous-requêtes, sous la forme <terme> | <corps> . La syntaxe "m[_]" est utilisée pour parcourir chaque élément du tableau.

Comme dans n'importe quel langage, l'utilisation de types de données tels que les objets et les collections permet de regrouper, d'agréger et de traiter des données connexes. Les compréhensions Rego sont utiles pour effectuer des conversions entre ces types de données.

2.2.4.6 Unification, affectation et comparaison

Rego dispose de plusieurs opérateurs qui peuvent être utilisés à différentes fins. Par exemple, en Rego, l'opérateur == est utilisé pour les comparaisons d'égalité. Les opérateurs de comparaison incluent également !=, >, >=, <, et <=. L'opérateur d'affectation est := et doit être utilisé lors d'une affectation de variable avant que la variable ne soit référencée. Les affectations aux variables avec l'opérateur d'affectation ne peuvent se produire qu'une fois pour cette variable. En d'autres termes, les variables sont immuables, elles ne peuvent pas être modifiées après leur affectation initiale.

En plus des opérateurs de comparaison et d'affectation, Rego nous surprend avec l'opérateur d'unification, =. L'unification combine la comparaison et l'affectation en attribuant des valeurs aux variables qui rendent la comparaison vraie. En d'autres termes, en utilisant l'opérateur d'unification dans les politiques, l'on demande des valeurs de variable qui rendent l'expression d'unification vraie. Si ces valeurs sont présentes, l'expression sera vraie.

2.3 Définition des politiques pour le pipeline CI/CD

Voici quelques politiques qui ont été définies pour le pipeline dans le cadre de l'étude. Ces politiques sont centrées sur les serveurs provisionnés avec Terraform.

Politique de Version Terraform

Objectif : S'assurer que la version de Terraform utilisée est conforme à une version spécifique.
Cette politique, cruciale pour la gestion de l'infrastructure en tant que code, veille à ce que le pipeline CI/CD utilise la version précise de Terraform requise pour les déploiements. En spécifiant une version spécifique, elle garantit la cohérence et la prévisibilité des opérations de déploiement. Cette rigueur dans la gestion des versions minimise les risques d'incompatibilité. De plus, elle offre une traçabilité claire des changements et assure que l'équipe travaille toujours avec un outil Terraform stable et bien testé.

Politique de Version d'AMI

Objectif : S'assurer que seules les versions d'AMI approuvées sont utilisées.
Cette politique répond à un impératif de sécurité en restreignant l'utilisation des AMIs à celles préalablement approuvées. Elle renforce la conformité aux normes de sécurité internes et externes. En garantissant que seules les AMIs de confiance sont déployées, elle réduit considérablement le risque d'exploitation de vulnérabilités liées à des versions non validées.

Politique de Clé Secure Shell (SSH)

Objectif : Garantir l'utilisation de clés SSH approuvées.
La sécurité des accès est cruciale dans tout environnement informatique. Cette politique s'assure que seules des clés SSH approuvées sont utilisées lors des déploiements. Elle assure ainsi un contrôle strict sur l'authentification des utilisateurs, limitant les risques d'accès non autorisés.

Politique de Taille d'Instance

Objectif : Limiter l'utilisation de certaines tailles d'instance.
Cette politique permet de contrôler les coûts et d'optimiser les ressources en limitant l'utilisation de tailles d'instance spécifiques définies comme conformes aux standards de l'organisation. Les équipes peuvent ainsi prendre des décisions éclairées lors du dimensionnement des instances, assurant une utilisation efficiente des ressources cloud et garantissant ainsi la disponibilité.

Politique de Groupe de Sécurité

Objectif : S'assurer que les instances utilisent uniquement le groupe de sécurité qui leur est destiné.
Cette politique s'articule autour de la segmentation réseau en veillant à ce que chaque instance utilise le groupe de sécurité approprié. Elle empêche la modification ou la révocation de l'appartenance d'une instance à un groupe de sécurité, renforçant la sécurité et limitant ainsi les accès non autorisé (port, IP, etc.)

Politique de Tags

Objectif : Garantir la présence d'un tag "Name", "Team", et "Product" pour chaque instance.
Cette politique vise à standardiser la gestion des ressources en imposant la présence de tags spécifiques sur chaque instance. Les informations incluses dans chaque tag, clarifient leur utilité pour l'identification, la traçabilité et la facturation des ressources. Elle contribue ainsi à une gestion efficace des ressources et à une organisation transparente au sein de l'infrastructure.

Politique de Gestion des données utilisateur

Objectif : S'assurer que les données utilisateur de l'instance EC2 ne contiennent pas de secrets.
Cette politique répond à des impératifs de sécurité en vérifiant que les données utilisateur des instances EC2 ne contiennent pas de secrets sensibles. Cela permet de prévenir les fuites d'informations sensibles et contribuer ainsi à renforcer la posture de sécurité des instances EC2.

Politique de Validation des instantanés de volumes EBS

Objectif : Assurer que les instantanés de volumes EBS sont correctement configurés et chiffrés. Cette politique se concentre sur la sécurité des données stockées en garantissant que les instantanés de volumes EBS respectent des configurations précises. Elle s'assure de leur configuration adéquate et de leur chiffrement. Le but étant de protéger les données sensibles, corriger les instantanés non conformes et maintenir un niveau élevé de sécurité des données tout au long du cycle de vie des volumes EBS.

2.4 Codification des politiques définies

Voici le code commenté des politiques en Rego :

```
package pipeline.policy
import future.keywords


# Politique : Version Terraform
# Objectif : S'assurer que la version de Terraform utilisée est conforme à une version spécifique.
default allow_terraform_version = false
allow_terraform_version {
   input.format_version >= "1.2"
   input.terraform_version >= "1.5.6"
}

# Politique de Version d'AMI
# Objectif : S'assurer que seules les versions d'AMI approuvées sont utilisées.
default allow_ami = false

is_approved_ami(ami) {
   approved_amis = {"ami-0e001c9271cf7f3b9", "ami-0d94353f7bad10668"}
    ami == approved_amis[_]
}
allow_ami {
     every instance in input.planned_values.root_module.resources
    {
      is_approved_ami(instance.values.ami)
    }
}

# Politique de Clé SSH
# Objectif : Garantir l'utilisation de clés SSH approuvées.
default allow_ssh_key = false
 allow_ssh_key {
    every instance in input.planned_values.root_module.resources
     {instance.values.key_name  == "ci-vprofile-key"}
 }

# Politique de Taille d'Instance
# Objectif : Limiter l'utilisation de certaines tailles d'instance.
default allow_instance_type = false

allow_instance_type {
    every instance in input.planned_values.root_module.resources
    {instance.values.instance_type == "t2.micro"}
 }

# Politique de Groupe de Sécurité
```

```
# Objectif : S'assurer que les instances utilisent uniquement le groupe de sécurité approprié.
default allow_security_group = false

is_approved_sg(sg) {
   approved_sgs = {"sg-069740889d6a5f5f6", "sg-03f43ad628fcd001c"}
    sg == approved_sgs[_]
}
allow_security_group {
     every instance in input.planned_values.root_module.resources
    {
      is_approved_sg(instance.values.vpc_security_group_ids[_])
    }
}

# Politique de Tags
# Objectif : Garantir la présence d'un tag "Name", "Team" et "Product" pour chaque instance.
default allow_tags = false
 allow_tags {
   every input_instance in input.planned_values.root_module.resources  {
       input_instance.values.tags.Name
       input_instance.values.tags.Team
       input_instance.values.tags.Product
    }
 }

#Politique de Gestion des données utilisateur
#Objectif : S'assurer que les données utilisateur de l'instance EC2 ne contiennent pas de secrets
default allow_without_secrets = false
allow_without_secrets {
    not (ec2_user_data_no_secrets)
}

ec2_user_data_no_secrets {
  check_sample_access_secret_key
}

check_sample_access_secret_key{
   every instance in input.planned_values.root_module.resources
  {count(regex.find_n(".*(A3T[A-Z0-9]|AKIA|AGPA|AIDA|AROA|AIPA|ANPA|ANVA|ASIA)[A-Z0-9]{16}.*",

  instance.user_data, -1)) >0}
}

check_sample_access_secret_key{
every instance in input.planned_values.root_module.resources
  {count(regex.find_n("([A-Za-z0-9\\\\\\\\/+\\\\]{40})", instance.user_data,-1))>0}
}

# Politique : Validation des instantanés de volumes EBS
# Objectif : Assurer que les instantanés de volumes EBS sont correctement configurés et chiffrés
default validate_ebs_snapshots = false
validate_ebs_snapshots {
   every snapshot in input.planned_values.root_module.resources {
      snapshot.values.ebs_block_device[_].volume_type == "gp2"
      snapshot.values.ebs_block_device[_].encrypted
   }
}
```

Conclusion

Dans ce chapitre, nous avons présenté un pipeline CI/CD exhaustif, sur lequel sera menée notre étude de l'approche de politique en tant que code. Nous avons exploré un scénario concret, identifié un problème spécifique, et défini une solution intégrant des outils et technologies clés tels que Jenkins, Git, Maven, et bien d'autres. Par la suite, notre attention s'est portée sur l'implémentation de la PaC en choisissant OPA comme solution privilégiée. Nous avons décortiqué les fondements d'OPA, de sa logique de fonctionnement à l'utilisation du langage Rego pour exprimer des politiques de manière déclarative. Nous avons ensuite défini quelques politiques pour le pipeline et codifier celles-ci en Rego qui est un langage de politique. Le prochain chapitre approfondira cette mise en pratique, en se concentrant sur l'intégration des politiques et les résultats.

Chapitre 3

Tests et résultats

Introduction

Ce chapitre se focalise sur l'intégration pratique des politiques définies précédemment au sein du pipeline CI/CD d'étude. Nous plongerons d'abord dans le processus d'intégration, examinant comment la solution de PaC peut être ajoutée au pipeline. Ensuite, nous allons tester l'évaluation de la conformité des politiques ainsi que le côté automatique directement lié au pipeline.

3.1 Présentation des étapes et du processus d'intégration des politiques

3.1.1 Les serveurs principaux et la partie CI

Pour parvenir à intégrer les politiques dans le pipeline, la première étape consiste à mettre en place les serveurs Jenkins, Nexus Repository et SonarQube. Cet aspect n'étant pas le sujet principal de cette étude, nous n'allons pas approfondir le processus derrière la mise en place de ces serveurs. Voici un aperçu des tableaux de bord que nous avons obtenus après la mise en place (voir figures 3.1-3.2-3.3).

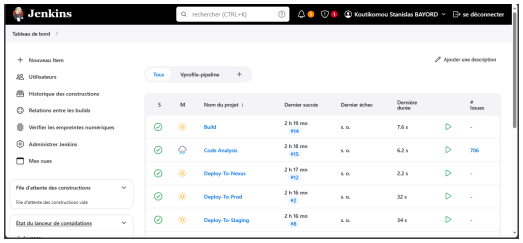


FIGURE 3.1 : Tableau de bord Jenkins

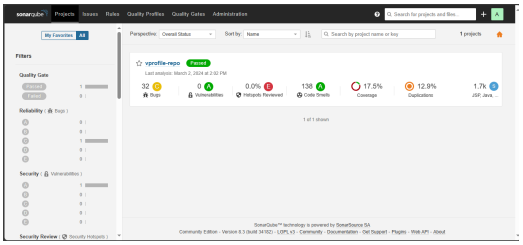


FIGURE 3.2 : Tableau de bord SonarQube

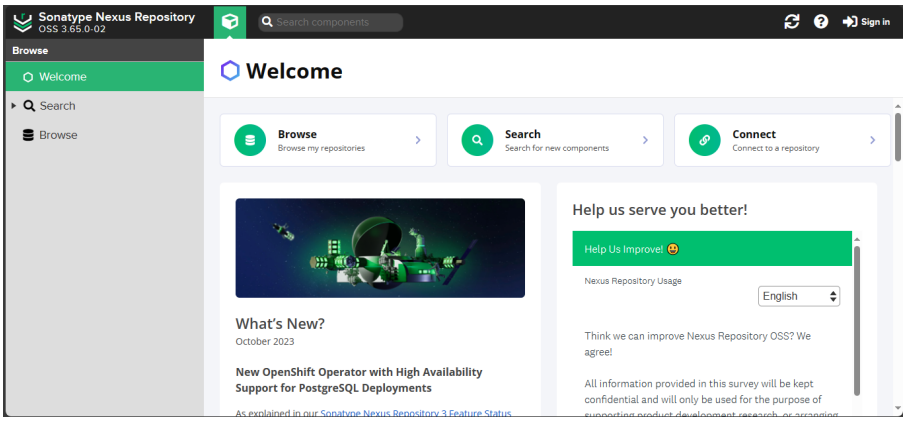


FIGURE 3.3 : Interface Nexus

Une fois les serveurs en marche, nous avons commencé par mettre en place la partie CI du pipeline. Le processus de mise en place ne sera pas également présenté, mais voici le résultat obtenu à la fin de cette étape (voir figure 3.4)

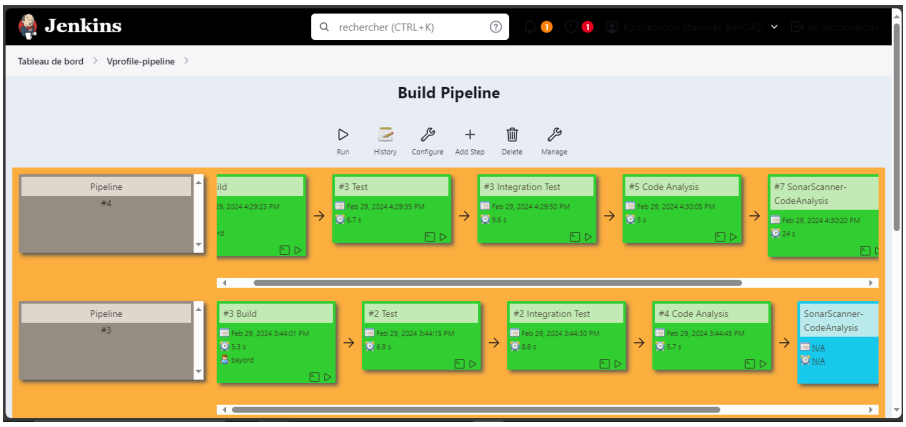


FIGURE 3.4 : Présentation du CI du pipeline

3.1.2 Mise en place du job de test de politiques

Après la mise en place des jobs du CI, l'étape suivante est de mettre en place le job Jenkins pour le test des politiques. Le fonctionnement doit être tel qu'à chaque nouveau commit et nouveau lancement du pipeline, qu'il s'exécute, récupère les politiques et les données sur le repository et évalue la conformité de ces politiques. Comme présenté au chapitre 2, la solution PaC qui est derrière tout ce processus n'est rien d'autre qu'OPA.

3.1.2.1 Création du Job des politiques

Ainsi pour commencer, nous créons un nouveau job Jenkins (dans cette étude, il sera nommé Policy-Test) et nous choisissons "construire un projet free-style" avant de valider (voir figure 3.5)

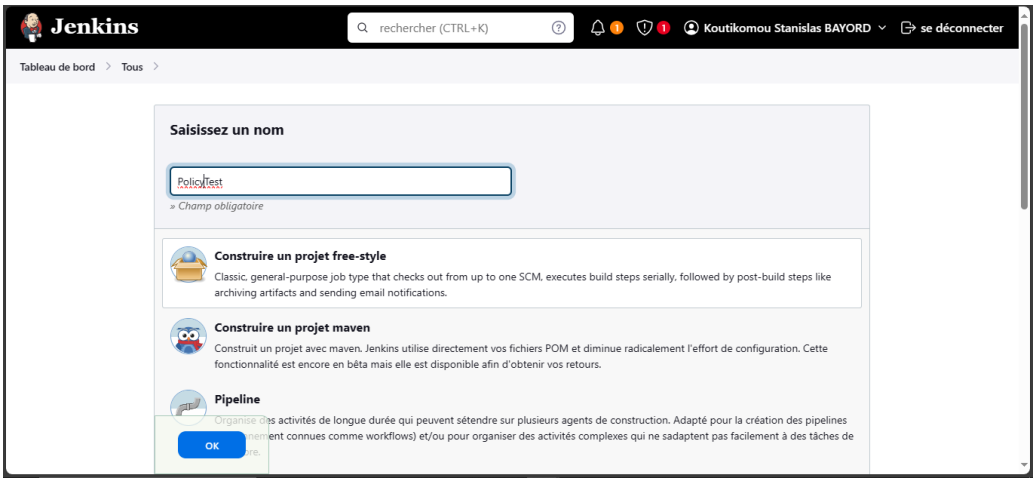


FIGURE 3.5 : Création du job de test de politiques

3.1.2.2 Configuration du job

Après la création du job, nous passons à sa configuration. Nous commençons par fournir l'URL de notre repository github ainsi que la branche qui contient nos politiques et le fichier de provisionning de l'infrastructure concernée. (voir figure 3.6)

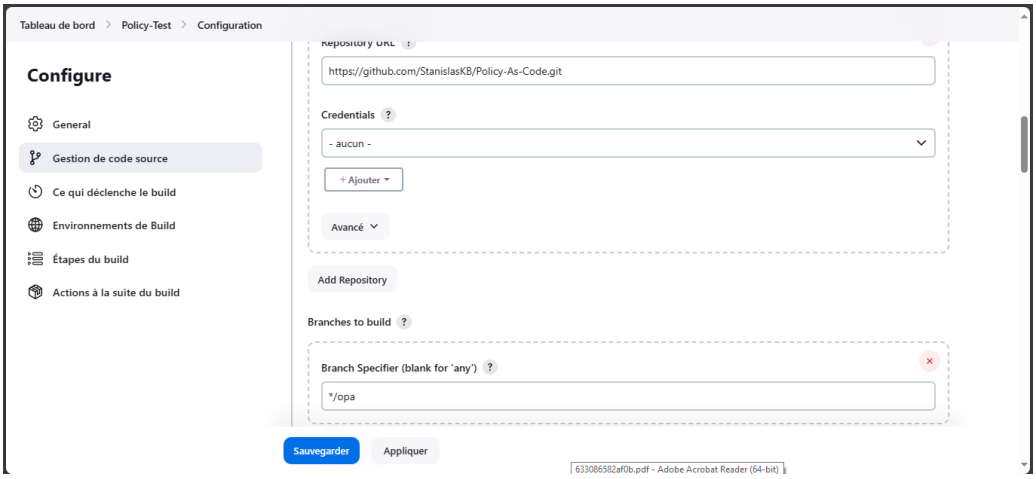


FIGURE 3.6 : Ajout de l'URL et la branche du projet

Ensuite, il faut fournir à Jenkins le script qui sera exécuté à chaque fois. Voici le code complet du script que nous avons écrit pour ce job.

```
cd infra/
terraform init
terraform plan --out input.binary
terraform show -json input.binary > input.json
cd ..
opa eval -d policy/ -i infra/input.json --format pretty "data.pipeline.policy"> policy_result.json
cat policy_result.json

if [ -f "policy_result.json" ]; then
    # Fonction pour convertir une chaine en booléen
    bool() {
        [ "$1" = "true" ]
    }

    allow_ami=$(jq -r '.allow_ami' "policy_result.json")
    allow_instance_type=$(jq -r '.allow_instance_type' "policy_result.json")
```

```
    allow_security_group=$(jq -r '.allow_security_group' "policy_result.json")
    allow_ssh_key=$(jq -r '.allow_ssh_key' "policy_result.json")
    allow_tags=$(jq -r '.allow_tags' "policy_result.json")
    allow_terraform_version=$(jq -r '.allow_terraform_version' "policy_result.json")
    allow_without_secrets=$(jq -r '.allow_without_secrets' "policy_result.json")
    validate_ebs_snapshots=$(jq -r '.validate_ebs_snapshots' "policy_result.json")

    # Vérifier si toutes les valeurs sont a true
    if bool "$allow_ami" && bool "$allow_instance_type" && bool "$allow_security_group" && bool
    "$allow_ssh_key" && bool "$allow_tags" && bool "$allow_terraform_version" && bool
    "$allow_without_secrets" && bool "$validate_ebs_snapshots"; then
        echo "Toutes les politiques sont vérifiées"
        exit 0
    else
        echo "Au moins une politique est non vérifiée"
        exit 1
    fi
fi
```

Les commandes terraforms, permettent d'initialiser un projet terraform, de générer un plan qui est ensuite mis au format JSON. C'est ce fichier JSON qui constitue les données dont OPA a besoin dans l'évaluation des politiques. La commande OPA qui suit ces commandes permet d'évaluer les politiques. Les résultats que nous obtenons sont enregistrés dans un fichier qui nous permet de vérifier si toutes les politiques sont vraies. Si tel est le cas, on notifie au processus du job que tout s'est bien passée et que le pipeline peut continuer. Mais si l'une des politiques retourne faux, on notifie au processus du job qu'il y a un problème. Dans ce cas, le pipeline s'arrête et ne reprend que lorsqu'on aura corrigé le problème de non-conformité et effectuer un nouveau commit.



FIGURE 3.7 : Script dans le job

La dernière étape est de définir les paramètres pour les notifications Slack et le job qui sera exécutée à la suite. Dans notre cas, il s'agit du job pour le provisionning Terraform (Terraform-prov).

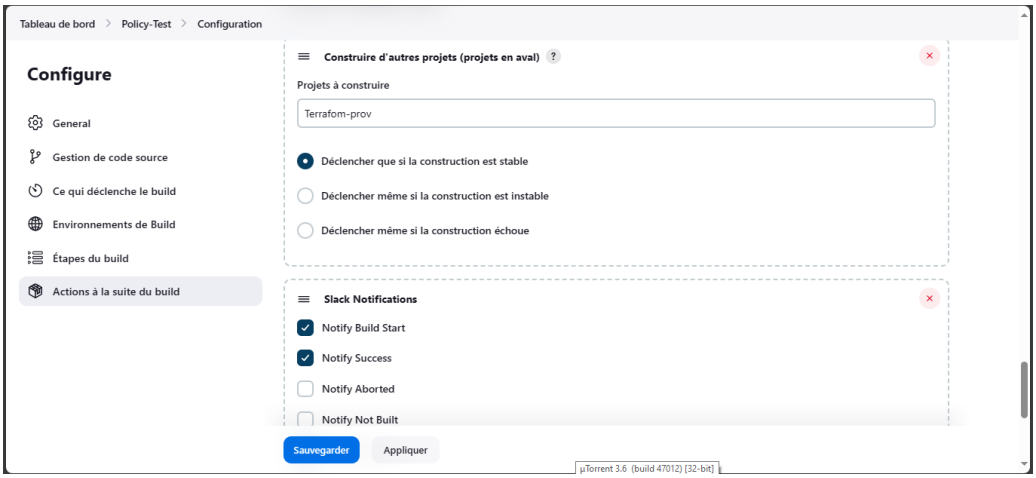


FIGURE 3.8 : Dernière étape du job

3.1.2.3 Le job Terraform-prov

Ce job est configuré pour construire l'infrastructure telle que définie dans le fichier. Donc quand le job de test de politique confirme que l'infrastructure attendue respecte effectivement les politiques, alors il s'exécute grâce à un script (voir figure 3.9)

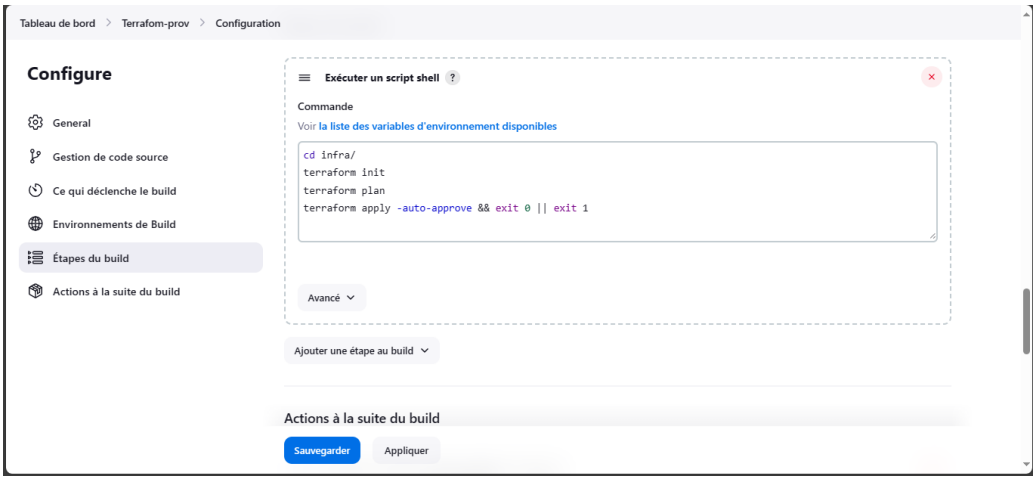


FIGURE 3.9 : Script pour l'exécution du job

Lorsque ce job va s'exécuter et qu'une instance n'existera pas, elle sera créée sans détours pour la première fois. Mais si une instance existe déjà, Terraform va examiner l'état actuel (state) de nos ressources déclarées dans notre configuration Terraform et les comparer à l'état souhaité défini dans nos fichiers de configuration Terraform. Dans différentes situations, il se passera ceci :

- **Aucun changement détecté :** Si l'état actuel correspond déjà à l'état souhaité défini dans nos fichiers de configuration Terraform, Terraform ne fera aucune modification.
- **Modifications détectées :** Si Terraform détecte des modifications à apporter à notre infrastructure (par exemple, des modifications de configuration, des changements de taille d'instance, etc.), il nous demandera confirmation avant de les appliquer. Nous devrons saisir "yes" pour que Terraform applique ces modifications. Mais grâce à l'option "-auto-approve" nous ne le ferons pas lors de l'exécution de ce job.
- **Ressource supprimée dans la configuration :** Si une ressource existante n'est plus déclarée

dans nos fichiers de configuration Terraform, Terraform considérera cela comme une suppression et nous demandera confirmation avant de la supprimer. L'option "-auto-approve" nous épargne de cette confirmation.

3.1.3 La partie CD

Pour cette partie, nous avons 4 jobs, mais comme dit ci-dessus, cet aspect n'étant pas le sujet principal de cette étude, nous n'allons pas présenter en profondeur le processus de création de ces jobs. Voici un aperçu de ce que nous avons obtenu après les configurations (voir figure 3.10)

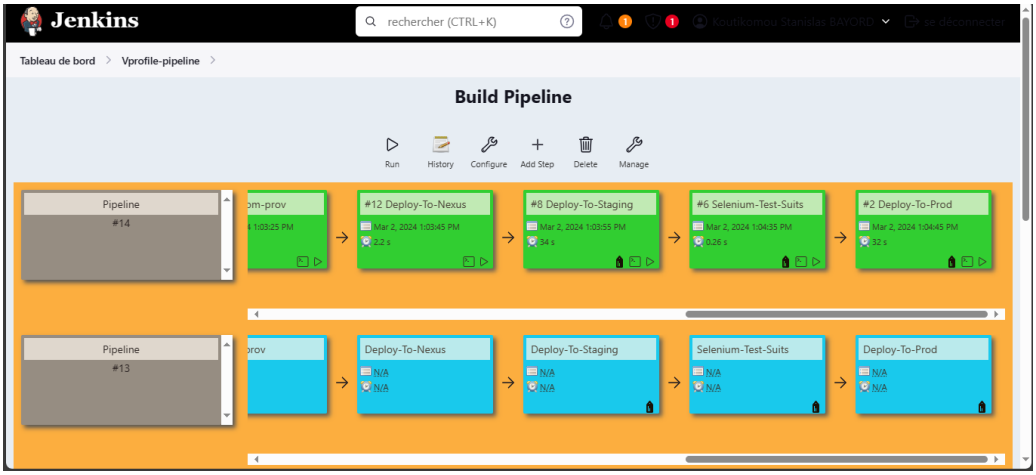


FIGURE 3.10 : Partie CD du pipeline

3.2 Présentation des tests et résultats

3.2.1 Test du job d'évaluation des politiques

Pour ce test, nous allons exécuter notre job d'évaluation de politique de façon isolé. Voici le résultat que nous obtenons (voir figure 3.11)

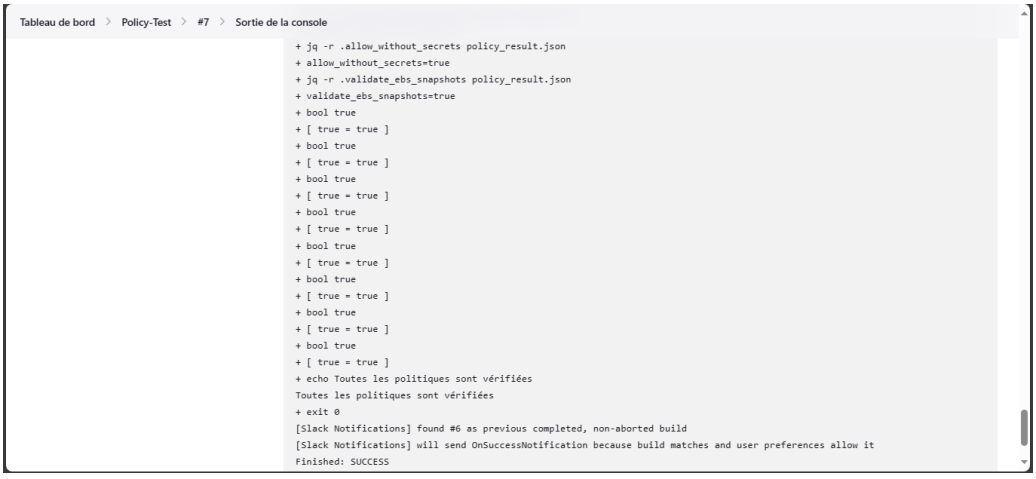


FIGURE 3.11 : Test du job isolé

À la fin du test, nous constatons que toutes les politiques sont vérifiées et par conséquent le pipeline peut continuer.
Pour pousser un peu plus le test, nous irons modifier le nom de la clé SSH dans le fichier de configuration. Ensuite, nous allons effectuer un commit et un push de cette modification sur la branche concernée pour enfin effectuer le test.

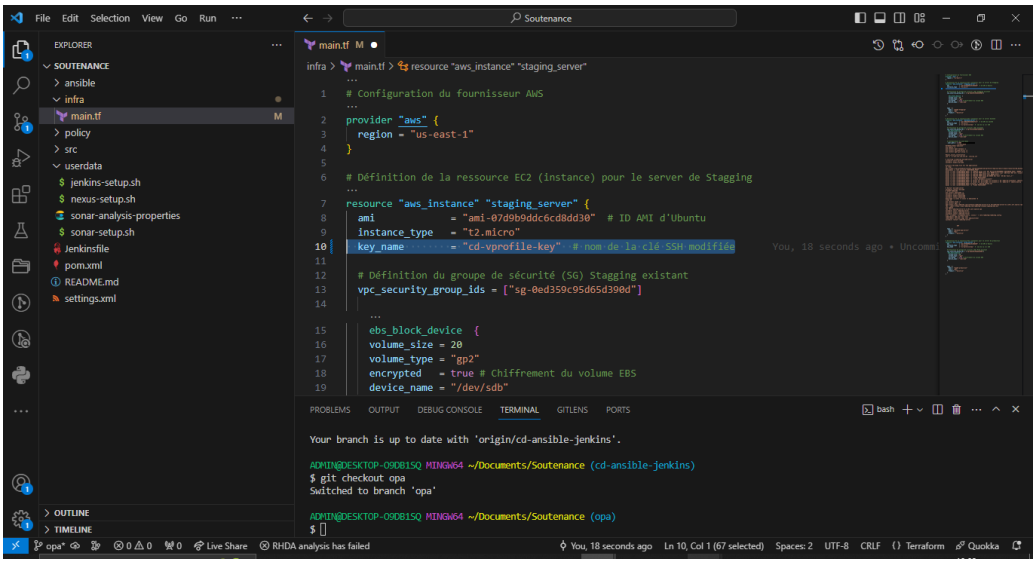


FIGURE 3.12 : Modification du code

Lorsque nous lançons le job, il se solde par un échec (voir figure 3.13).

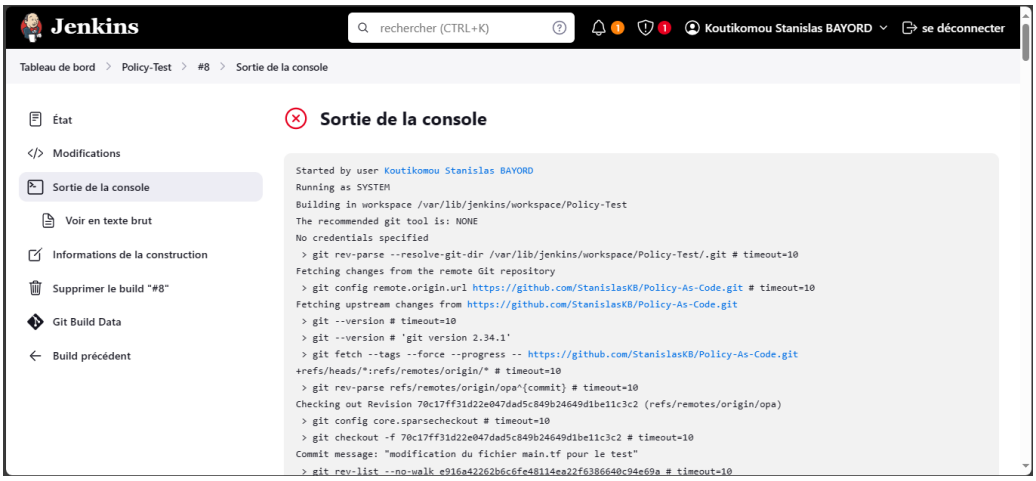


FIGURE 3.13 : Résultat d'échec après la modification du code

En observant les logs, nous constatons clairement que la politique liée à la clé SSH retourne false (allow_ssh_key=false). Donc l'évaluation de la conformité des politiques est juste et OPA retourne les valeurs qui doivent être retournées (voir figure 3.14).

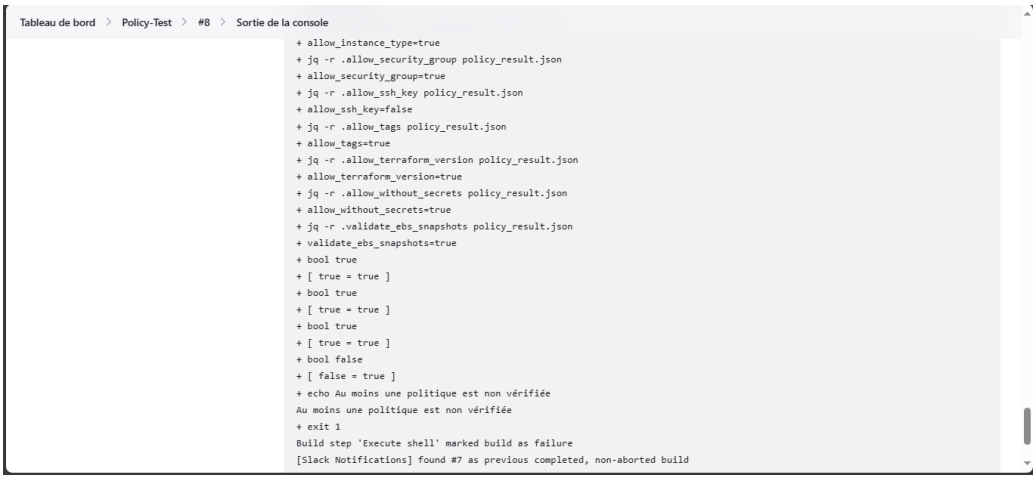


FIGURE 3.14 : Données des logs du job

3.2.2 Test du pipeline en entier

Pour cette section, nous allons procéder aux tests pour s'assurer que lorsqu'on effectue un nouveau commit, le pipeline se lance seul de façon automatique sans aucune intervention manuelle. Voici à quoi ressemble notre espace Slack avant le début du test (voir figure 3.15).

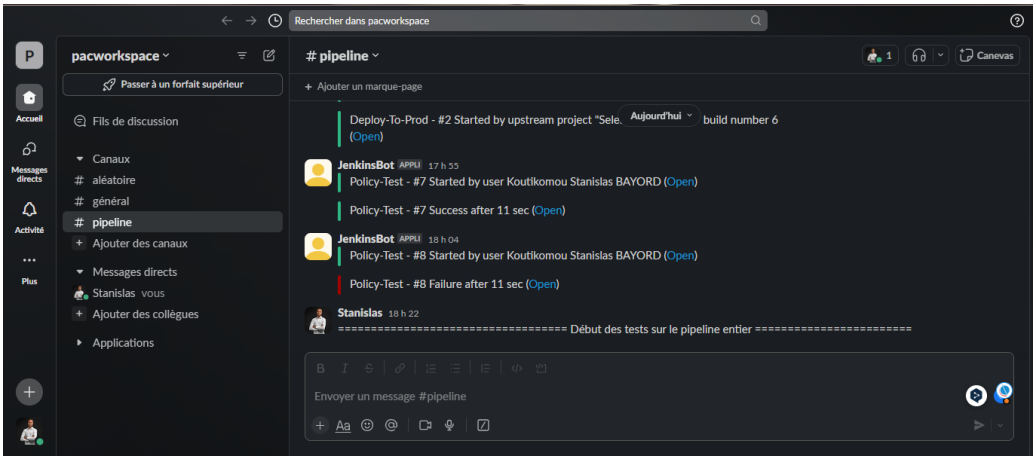


FIGURE 3.15 : Espace Slack avant le début des tests

Voici ce à quoi ressemble notre espace Slack quand le pipeline est déclenché par un administrateur Jenkins. On constate que c'est bien notifié que le pipeline a été lancé par un utilisateur. Par ailleurs, le processus se termine avec un succès pour tous les jobs. (voir figure 3.16 et 3.17)

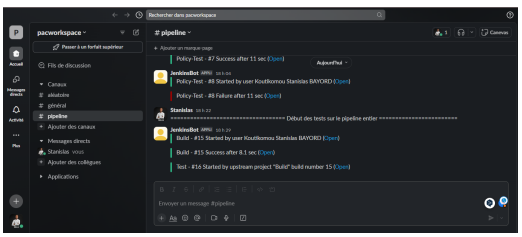


FIGURE 3.16 : Pipeline déclenché par un utilisateur manuellement

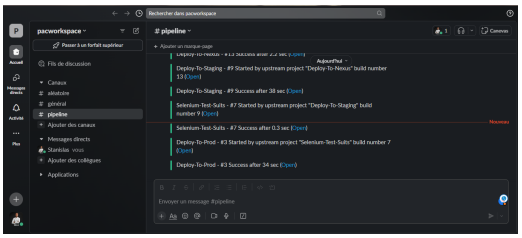


FIGURE 3.17 : Résultat pipeline déclenché par un utilisateur manuellement

Nous allons effectuer une modification dans un fichier et effectuer un commit pour observer le résultat. Après le commit, nous constatons que le pipeline est automatiquement lancé et se termine par un succès pour tous les jobs. (voir figure 3.18 et 3.19)

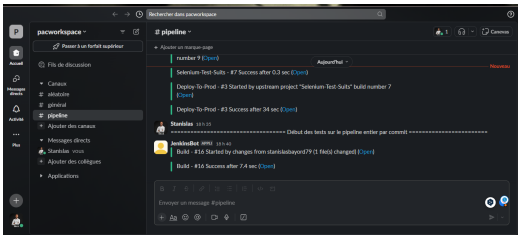


FIGURE 3.18 : Pipeline déclenché automatiquement par un commit

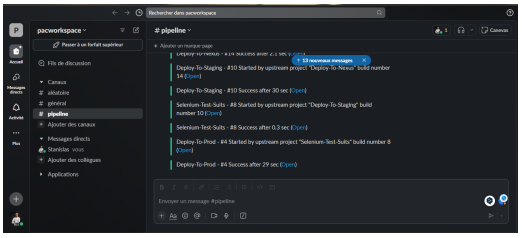


FIGURE 3.19 : Résultat pipeline déclenché automatiquement par un commit

À présent, nous allons encore modifier la clé SSH et effectuer un commit. Le pipeline se lance automatiquement, mais il s'arrête lors de l'exécution du job d'évaluation des politiques. (voir figure 3.20 et 3.21) Cela est dû au fait que nous avons modifié la clé SSH. Il suffit de remodifier et de mettre la clé idéale pour voir le pipeline se terminer avec succès.

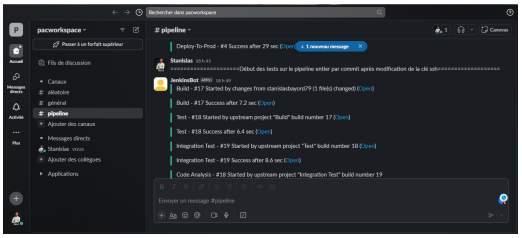


FIGURE 3.20 : Pipeline déclenché automatiquement par un commit après modification

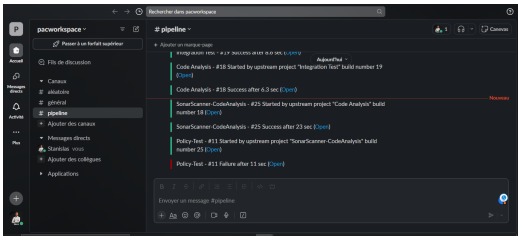


FIGURE 3.21 : Résultat pipeline déclenché automatiquement par un commit après modification

Conclusion

Au terme de ce chapitre, nous avons réalisé une plongée approfondie dans l'intégration concrète des politiques définies au sein de notre pipeline CI/CD. Nous avons démarré en démontrant de près le processus d'intégration ainsi que tous les éléments intervenants. L'accent a été mis sur la praticité, la facilité d'intégration et l'automatisation du processus. Les tests d'évaluation de conformité sont exécutés de manière automatique, permettant une détection précoce des éventuelles violations de politiques. Cette approche proactive renforce la conformité du pipeline et contribue à la construction d'un environnement de déploiement fiable.

Conclusion Générale

Dans l'univers en pleine évolution du développement logiciel, la sécurisation des infrastructures informatiques s'impose comme un impératif essentiel. Face à la montée en puissance des pratiques DevOps, promouvant rapidité et agilité, la sécurité doit évoluer. C'est dans ce contexte que de nouvelles approches de sécurité comme l'approche de Politique en tant que Code ont vu le jour.
L'objet de ce mémoire a été d'étudier cette approche ainsi que sa mise en œuvre dans un environnement DevOps. Notre travail a débuté par un état de l'art sur l'approche étudiée, établissant ainsi les bases conceptuelles. Dans la deuxième partie, nous avons approfondi l'étude en présentant un pipeline spécifique, un outil de mise en œuvre de Politique en tant que Code et enfin, nous avons identifié, défini et codifié quelques politiques pour ce pipeline. Dans la troisième partie, nous avons intégré ces politiques et effectué des tests de conformité. Ce projet apporte une vision concrète de la PaC au sein de l'environnement DevOps.
Le travail effectué dans ce mémoire est loin d'être parfait. Il pourrait être amélioré, corrigé et approfondi. Il pourrait être approfondi à travers une étude sur un pipeline incluant des technologies comme **Docker, Kubernetes, Kafka,** etc. Également, définir des politiques pour chaque étape du pipeline.

Bibliographie

[1] Jimmy Ray. Policy as code : Improving cloud-native security. O'Reilly Media, Inc, 2023.

Webographie

[2] All4test. Commencer avec selenium webdriver (java). https://www.all4test.fr/blog-du-testeur/commencer-avec-selenium-webdriver/. Consulté en Janvier 2024.

[3] AWS. Qu'est-ce que l'infrastructure informatique ? https://aws.amazon.com/fr/what-is/it-infrastructure. Consulté en Décembre 2023.

[4] BDM. Slack. https://www.blogdumoderateur.com/tools/slack/. Consulté en Janvier 2024.

[5] Blent AI. Ansible : qu'est-ce que c'est ? https://blent.ai/blog/a/ansible-quest-ce-que-cest. Consulté en Janvier 2024.

[6] CrowdStrike. Politique en tant que code (policy as code ou pac). https://www.crowdstrike.fr/cybersecurity-101/security-operations/policy-as-code/. Consulté en Décembre 2023.

[7] CrowdStrike. Qu'est-ce que le devops ? https://www.crowdstrike.fr/cybersecurity-101/cloud-security/devops/. Consulté en Décembre 2023.

[8] CyberArk. Qu'est-ce que la sécurité du devops. https://www.cyberark.com/fr/what-is/devops-security/. Consulté en Décembre 2023.

[9] DevUniversity. Jenkins : Guide complet de cet outil ci/cd open-source. https://www.devuniversity.com/blog/jenkins-guide-complet-de-cet-outil-ci-cd-open-source. Consulté en Janvier 2024.

[10] JavaEE Goffinet. Introduction à tomcat. https://javaee.goffinet.org/tomcat-01-introduction-tomcat/. Consulté en Janvier 2024.

[11] Jenkins - Le Guide Complet. Les outils d'analyse de qualité du code populaires pour java et groovy. https://jenkins-le-guide-complet.github.io/html/sect-code-quality-tools.html. Consulté en Janvier 2024.

[12] Marclabs. Sonarqube, un outil d'analyse de code source surpuissant. https://marclabs.com/sonarqube-un-outil-danalyse-de-code-source-surpuissant/. Consulté en Janvier 2024.

[13] Next Decision. Qu'est-ce que git ? https://www.next-decision.fr/wiki/qu-est-ce-que-git. Consulté en Janvier 2024.

[14] Open Policy Agent. Open policy agent introduction. https://www.openpolicyagent.org/docs/latest/. Consulté en Décembre 2023.

[15] Paloalto Networks. What is policy as code? https://www.paloaltonetworks.com/cyberpedia/what-is-policy-as-code. Consulté en Décembre 2023.

[16] Redhat. Comprendre le devops. https://www.redhat.com/fr/topics/devops. Consulté en Décembre 2023.

[17] Redhat. Introduction to policy as code with automation. https://www.redhat.com/sysadmin/policy-as-code-automation. Consulté en Décembre 2023.

[18] Redhat. L'approche ci/cd : qu'est-ce que c'est? https://www.redhat.com/fr/topics/devops/what-is-ci-cd. Consulté en Décembre 2023.

[19] Services Renater. Nexus. https://services.renater.fr/sourcesup/nexus. Consulté en Janvier 2024.

[20] Tekunda. The 5 pillars of salesforce devops. https://tekunda.com/blog/The-5-Pillars-of-Salesforce-DevOps. Consulté en Décembre 2023.

[21] Wikipedia. Apache maven. https://fr.wikipedia.org/wiki/Apache_Maven. Consulté en Janvier 2024.

Table des matières

Dédicace ii

Remerciements iii

Résumé iv

. iv

Abstract v

. v

Table des figures vi

Liste des tableaux vii

Liste des acronymes viii

Introduction 1

1 État de l'Art 3

Introduction 3

1.1 Concept de DevOps 3

1.1.1 Qu'est-ce que DevOps ? 3

1.1.2 L'Approche CI/CD 4

1.2 Sécurité des infrastructures informatiques en environnement DevOps 5

1.2.1 Qu'est-ce qu'une infrastructure informatique ? 5

1.2.2 Défis relatifs à la sécurité des infrastructures informatiques en environnement DevOps 5

1.2.3 Évolution des pratiques de sécurité dans DevOps 6

1.3 Politique en tant que Code 7

1.3.1 Explication du concept de Politique en tant que Code 7

1.3.2 En quoi consiste l'approche de la politique en tant que code ? 7

1.3.3 Avantages de la politique en tant que code 8

1.3.4 Politique en tant que Code VS Infrastructure en tant que Code (IaC) 9

1.4 Solutions pour la Mise en œuvre de la Politique en tant que Code 9

1.5 Choisir la solution PaC adéquate 10

1.5.1 Facteurs de sélection à prendre en compte 10

1.5.2 Méthode pour évaluer différentes solutions PaC 11

Conclusion 12

2 Pipeline et Solution PaC **13**
Introduction . . . 13
2.1 Présentation du pipeline CI/CD . . . 13
2.1.1 Scénario pour le pipeline . . . 13
2.1.2 Problème . . . 14
2.1.3 Solution . . . 14
2.1.4 Principaux outils/technologies du pipeline . . . 14
2.1.4.1 Jenkins . . . 14
2.1.4.2 Git . . . 15
2.1.4.3 Apache Maven . . . 15
2.1.4.4 CheckStyle . . . 15
2.1.4.5 Slack . . . 16
2.1.4.6 Sonatype Nexus Repository . . . 16
2.1.4.7 SonarQube . . . 16
2.1.4.8 Ansible . . . 17
2.1.4.9 Selenium . . . 17
2.1.4.10 Apache Tomcat . . . 17
2.1.5 Architecture du pipeline . . . 18
2.2 Présentation de la solution PaC choisie . . . 20
2.2.1 Qu'est-ce que Open Policy Agent ? . . . 20
2.2.2 Pourquoi utiliser OPA ? . . . 20
2.2.3 Comment OPA fonctionne ? . . . 20
2.2.4 Rego . . . 21
2.2.4.1 Modèle de Document d'OPA . . . 21
2.2.4.2 Règles . . . 21
2.2.4.3 Fonctions . . . 22
2.2.4.4 Fonctions intégrées . . . 23
2.2.4.5 Objets, collections et compréhensions . . . 23
2.2.4.6 Unification, affectation et comparaison . . . 24
2.3 Définition des politiques pour le pipeline CI/CD . . . 24
2.4 Codification des politiques définies . . . 26
Conclusion . . . 27

3 Tests et résultats **29**
Introduction . . . 29
3.1 Présentation des étapes et du processus d'intégration des politiques . . . 29
3.1.1 Les serveurs principaux et la partie CI . . . 29
3.1.2 Mise en place du job de test de politiques . . . 30
3.1.2.1 Création du Job des politiques . . . 30
3.1.2.2 Configuration du job . . . 31
3.1.2.3 Le job Terraform-prov . . . 33
3.1.3 La partie CD . . . 34
3.2 Présentation des tests et résultats . . . 34
3.2.1 Test du job d'évaluation des politiques . . . 34
3.2.2 Test du pipeline en entier . . . 36

Conclusion 38

Bibliographie 39

Webographie 40

Table des matières 42