RIMEL – Suivie de projet

Équipe E

HashiCorp Terraform

Sujet

Question générale

Quelles bonnes pratiques de Terraform sont vérifiables ?

Sous-questions

1) Comment définir une bonne pratique en Terraform ?
2) Quelles pratiques sont encouragées en Terraform ?
3) Est-ce que Terraform est toujours bien utilisé en pratique ?

Auteurs

- João Brilhante
- Enzo Briziarelli
- Charly Ducrocq
- Quentin Larose
- Ludovic Marti

UNIVERSITÉ CÔTE D'AZUR

POLYTECH NICE-SOPHIA

Table des matières

I. Analyse des bonnes pratiques

1. Critères de sélection

Une bonne pratique est :

- Justifiée ;
- Non essentielle au fonctionnement de Terraform ;
- Valide pour les versions récentes de Terraform ;
- Appuyée par des exemples (optionnel) ;
- Mesurable (optionnel).

2. Tableau des bonnes pratiques

Description	Justification	Exemples	Mesurabilité	Tags
Dans le bloc “required_providers”, il est recommandé de toujours expliciter l’attribut “version”. [source 1]	Cela permet de contraindre Terraform et de l'empêcher d’installer un fournisseur incompatible avec la configuration. La dernière version sera installée par défaut.		Facile	- Modules
Utiliser des fichiers de variables. [source 1]	Cela permet de facilement gérer les variables d'environnement sans lancer terraform avec une longue liste de paires clé-valeur.		Facile	- Variables - Conventions
Ne jamais écrire des identifiants en clair dans un fichier de configuration de Terraform. [source 1]	Pour ne pas s’exposer à des fuites et à du piratage.		Moyen	- Security - Variables - Resources
Nommer les variables entièrement en minuscules séparées par des underscores. [source 1] [source 2]	Par convention. Certaines ressources cloud peuvent avoir des contraintes de nommage.	“ma_variable”, “une_ressource”	Moyen	- Variables - Convention
Utiliser un format consistant dans les différents fichiers de configuration. [source 1]	Lisibilité et cohérence. Terraform fournit un outil de formatage automatique des fichiers de configuration.		Difficile	- Lisibility
Utiliser les fins de ligne de Unix plutôt que celles de Windows. [source 1]	Par convention.			- Convention
Utiliser plusieurs fichiers de configuration .tf [source 1]	Cela permet de répartir les responsabilités dans des fichiers distincts et de les rendre plus lisibles.	Notre projet de Cloud !	Facile	- Structure - Lisibility
Utiliser les modules officiels de Terraform autant que possible et les modifier si besoin.	Cela permet d'alléger la configuration en utilisant des ressources fonctionnelles et bien		Moyen	- Modules

[source 1] [source 2] [source 3]	conçues tout en évitant de réinventer la roue.			
Documenter le code. [source 1] [source 2] [source 3]	Cela permet une meilleure compréhension et maintenabilité.	Commenter les lignes et/ou utiliser un README.	Facile	- Lisibility
Créer des ressources avec un nom différent pour chaque environnement et chaque ressource. [source 1]	Définition facile de la ressource avec un nom significatif et unique, et possibilité de construire plus de la même pile d'applications pour différents développeurs sans changer beaucoup. Par exemple, vous mettez à jour l'environnement pour dev, staging, uat, prod, etc.		Difficile	- Conventions
Isoler les ressources indépendantes dans différentes compositions. [source 1]	Les ressources doivent être isolées par dépendance, c.a.d celles n'étant pas liées entre elles (non interdépendantes) ne doivent pas être placées dans une même composition. Permet d'isoler plus efficacement l'infrastructure et d'éviter qu'une erreur/panne se propage.		Moyen	- Resources - Structure - Modules
Construire des modules de ressources atomiques. [source 1]	Un module de ressource va englober des ressources et des données, donc afin d'éviter les effets de bord indésirables, les modules doivent être le plus atomique possible, donc avec le moins de dépendance. Cela permet ainsi de garantir une composition atomique et rejoint le point précédent concernant l'isolation des ressources.		Moyen	- Modules - Structure
Spécifier explicitement les dépendances entre les ressources. [source 1]	Via le mot clé 'locals' on spécifie explicitement à Terraform une relation de dépendance entre ressources, dans le cas où Terraform n'aurait pas automatiquement reconnu cette dépendance. Cela permet notamment de définir un ordre dans les dépendances pour, par exemple, expliciter la dépendance d'une base de données avec un service, afin d'être garantie que Terraform déploie en priorité la base de données. Ainsi, le		Moyen	- Variables

	service est certain de pouvoir y accéder.			
Maintenir régulièrement la configuration de Terraform à jour. [source 1] [source 2]	La communauté de développement Terraform est très active et la publication de nouvelles fonctionnalités est fréquente. Ainsi, il devient difficile de mettre à jour Terraform si plusieurs versions majeures ont été ignorées. Pour cette raison, il est recommandé de maintenir régulièrement la configuration de Terraform à jour. De plus, cela permet de bénéficier des derniers correctifs de sécurité.		Moyen	- Security
Ne jamais partager le fichier .tfstate sur le repository. [source 1] [source 2]	Le fichier .tfstate est un fichier généré par Terraform lorsque des commandes qui modifient l'état de l'infrastructure sont exécutées. En effet, Terraform a besoin d'un fichier .tfstate pour connaître l'état de l'infrastructure avant de faire des modifications. Cependant, partager le fichier .tfstate sur le repository présente plusieurs risques. Premièrement, il est possible d'exposer les secrets de la configuration de l'application, tels que des mots de passe, des chaînes de connexion à la base de données, etc. Et deuxièmement, il est possible d'exécuter Terraform sur un état périmé ou ancien que quelqu'un a oublié de retirer du repository. Dans ce cas, il existe une alternative au partage du fichier .tfstate qui consiste à configurer une fonctionnalité appelée "backend" Terraform.		Facile	- Security - Teamwork

Sauvegarder l'état de l'infrastructure sur un backend Terraform. [source 1] [source 2]	La plupart du temps, plusieurs développeurs travaillent sur un projet. Pour leur donner accès au fichier d'état Terraform de manière sécurisée, celui-ci doit être stocké dans un emplacement centralisé et distant. Dans ce cas, il est possible de configurer un backend Terraform qui fera office de source unique de vérité sur l'état Terraform de votre infrastructure. En plus de permettre le partage sécurisé de l'état de l'infrastructure, un backend Terraform permet de faciliter le versionnement de l'état (pour revenir rapidement à un état connu en cas de problème) et de mettre en place un verrou sur l'état (pour éviter que plusieurs développeurs déploient des modifications au même temps).		Moyen	- Teamwork
Mettre en place un verrou sur l'état Terraform à l'aide d'un backend Terraform. [source 1] [source 2]	Si plusieurs développeurs travaillent sur un même projet, il est possible que plusieurs membres décident d'exécuter la configuration Terraform en même temps. Cela peut conduire à la corruption du fichier d'état Terraform ou même à une perte de données. Dans ce cas, il est possible de configurer un verrou sur l'état Terraform à l'aide d'un backend Terraform. Pour un backend configuré avec AWS, la mise à jour de la configuration du backend consiste simplement à indiquer un nouveau nom de table Dynamo DB.		Difficile	- Teamwork
Exécuter la configuration de Terraform à l'aide d'une chaîne de déploiement continu. [source 1]	L'exécution du code à l'aide d'une chaîne de déploiement continu présente de nombreux avantages, notamment un processus reproductible et un historique des modifications. Le concept de builds est également très utile lorsqu'il est appliqué à Terraform, car il permet de mieux voir ce qui a été exécuté sur votre			- Modules

	infrastructure à des fins d'audit, de débogage et de collaboration.			
Ne jamais modifier le fichier d'état Terraform manuellement. [source 1]	Le fichier d'état est une représentation de votre infrastructure dans le monde réel. Il arrive parfois que certaines situations vous obligent à modifier cet état. Beaucoup sont tentés, lorsqu'ils doivent déplacer ou renommer un état, de plonger dans le fichier d'état lui-même et de commencer à le bidouiller. Mais attention, il existe un moyen beaucoup plus sûr ! La CLI de Terraform vous offre des commandes qui vous permettent de supprimer ou de déplacer. Celle-ci pourra effectuer un certain nombre de vérifications et effectuera les modifications à tous les endroits nécessaires.	Par exemple, si vous souhaitez renommer un bloc de ressources, vous devrez réassigner votre état Terraform au nouveau nom de la ressource.	Impossible	- Structure - Language
Utiliser le bon type de données pour les variables. [source 1]	L'utilisation de types de données appropriés dans Terraform facilite la validation des entrées et la documentation de l'utilisation.		Difficile	- Variables - Conventions

II. Analyse des sources de bonnes pratiques

1. Critères de sélection

La sélection d'une source est influencée par :

- La réputation de l'auteur ;
- L'expérience de l'auteur ;
- L'absence de conflit d'intérêt.

2. Tableau des sources

Nom	Auteurs	Réputation	Expérience	Conflit d'intérêt ?
HashiCorp Learn - Terraform - Build Infrastructure - Terraform AWS Example				
Terraform by HashiCorp - Plugin Development - Naming Best Practices				
Geekflare - 10 Terraform Best Practices for Better Infrastructure Provisioning	Avi			
UpCloud - Terraform best practices for beginners	UpCloud			
Open Up The Cloud - 10 Terraform Best Practices: For Secure & Fast Infrastructure	Lou Bichard			
Terraform Best Practices	Anton Babenko			HashiCorp Ambassador
Cloud Posse Developer Hub - Terraform Best Practices	Cloud Posse			

III. Sélection des outils d'analyse

1. Définitions

Nous définissons une **mauvaise pratique** comme toute pratique allant à l'encontre d'une ou plusieurs des bonnes pratiques que nous avons sélectionnées.

2. Critères de sélection

La sélection d'un outil d'analyse est influencée par :

- La licence de l'outil ;
- La maturité de l'outil ;
- Les types d'analyses proposés ;
- Le nombre de mauvaises pratiques détectées ;
- Le type de mauvaises pratiques détectées (syntaxe ou sécurité) ;
- La possibilité d'ajouter des règles de détection personnalisées ;
- La complexité, la testabilité et l'expressivité des règles personnalisées ;
- L'utilisabilité générale de l'outil.

3. Outils d'analyses

a. TFLint

Description

TFLint est un linter Terraform construit sous la forme d'un framework où chaque fonctionnalité est fournie par un plugin.

Fonctionnalités

Les fonctionnalités principales sont les suivantes :

- Détecter les erreurs de configuration pour les principaux fournisseurs de cloud.
- Signaler l'utilisation d'une syntaxe dépréciée, les déclarations inutiles.
- Signaler les bonnes pratiques, les conventions de nommage.

La détection d'erreurs de configuration Terraform prend en charge les fournisseurs de cloud suivants :

- AWS
- Azure
- Google Cloud Platform

b. Checkov

Description

Checkov est un outil d'analyse statique de code pour les fichiers d'infrastructure-as-code. Il analyse les infrastructures cloud provisionnées à l'aide de Terraform, Terraform Plan, AWS CloudFormation, AWS SAM, Kubernetes, Helm charts, Kustomize, Dockerfile, Serverless ou Azure Resource Manager (ARM) et détecte les erreurs de sécurité et de conformité à l'aide d'une analyse basée sur des graphes.

Fonctionnalités

Les fonctionnalités principales sont les suivantes :

- Détecter les erreurs de configuration en matière de sécurité et de conformité pour de nombreux fournisseurs de cloud.
- Détecter les informations d'identification AWS dans les données utilisateur EC2, les variables d'environnement Lambda et les fournisseurs Terraform.
- Identifier les secrets à l'aide d'expressions régulières, de mots-clés et d'une détection basée sur l'entropie.
- Évaluer les paramètres des fournisseurs Terraform pour réglementer la création, la gestion et les mises à jour des IaaS, PaaS ou SaaS gérés par Terraform.

La détection d'erreurs de configuration Terraform prend en charge les fournisseurs de cloud suivants :

- AWS
- Azure
- Google Cloud Platform
- Kubernetes
- GitHub
- Digital Ocean
- Oracle Cloud Infrastructure
- OpenStack
- Linode

c. TFSec

Description

TFSec est un outil d'analyse statique de code pour les fichiers Terraform permettant de repérer les problèmes de sécurité potentiels.

Fonctionnalités

Les fonctionnalités principales sont les suivantes :

- Détecter la présence de données sensibles lors de l'utilisation de n'importe quel fournisseur.
- Détecter les erreurs de configuration pour de nombreux fournisseurs de cloud.
- Analyser les modules Terraform.
- Évaluer les expressions ainsi que les valeurs littérales.
- Évaluer les fonctions Terraform, par exemple, concat().

La détection d'erreurs de configuration Terraform prend en charge les fournisseurs de cloud suivants :

- AWS
- Azure
- Google Cloud Platform
- Kubernetes
- GitHub
- DigitalOcean
- CloudStack
- Oracle Cloud Infrastructure
- OpenStack

d. Terrascan

Description

Terrascan est un outil d'analyse statique de code pour les fichiers d'infrastructure-as-code. Il analyse les infrastructures cloud provisionnées à l'aide de Terraform, Kubernetes, Helm, Kustomize et Dockerfile.

Fonctionnalités

Les fonctionnalités principales sont les suivantes :

- Analyser l'infrastructure-as-code pour détecter les erreurs de configuration.
- Détecter les vulnérabilités de sécurité et les violations de conformité.
- Réduire les risques avant de provisionner l'infrastructure.

La détection d'erreurs de configuration Terraform prend en charge les fournisseurs de cloud suivants :

- AWS
- Azure
- Google Cloud Platform
- Kubernetes

- GitHub

e. Regula

Description

Regula est un outil d'analyse statique de code pour les fichiers d’infrastructure-as-code. Il analyse les infrastructures cloud provisionnées à l'aide de Terraform, Terraform Plan, AWS CloudFormation, Kubernetes ou Azure Resource Manager (ARM).

Fonctionnalités

Les fonctionnalités principales sont les suivantes :

- Analyser l'infrastructure-as-code pour détecter les erreurs de configuration.
- Détecter les vulnérabilités de sécurité et les violations de conformité.
- Réduire les risques avant de provisionner l'infrastructure.

La détection d’erreurs de configuration Terraform prend en charge les fournisseurs de cloud suivants :

- AWS
- Azure
- Google Cloud Platform
- Kubernetes

f. Snyk IaC

Description

Snyk Infrastructure as Code (Snyk IaC) aide les développeurs à rédiger des configurations Terraform, AWS CloudFormation, Kubernetes et Azure Resource Manager (ARM) sécurisées avant de toucher à la production.

4. Comparaison des outils d'analyse

a. Présentation

	TFLint	Checkov	TFSec	Terrascan	Regula	Snyk IaC
Prix	Gratuit	Gratuit	Gratuit	Gratuit	Gratuit	Gratuit / Payant
Open source	Oui	Oui	Oui	Oui	Oui	Non (sauf CLI)
Licence	MPL 2.0	Apache 2.0	MIT	Apache 2.0	Apache 2.0	Apache 2.0 (CLI)
Langage	Go	Python	Go	Go	Open Policy Agent	TypeScript (CLI)
Date de création	10/2016	11/2019	03/2019	09/2017	12/2019	10/2015 (CLI)
Popularité	2.8k stars	3.7k stars	3.8k stars	2.7k stars	621 stars	3.7k stars (CLI)
Soutenu par	Indépendant	Bridgecrew (entreprise)	Aqua Security (entreprise)	Accurics (entreprise)	Fugue (entreprise)	Snyk (entreprise)

b. Règles intégrées

	TFLint	Checkov	TFSec	Terrascan	Regula	Snyk IaC
Types de d'analyse	HCL	HCL ou Plan	HCL	HCL	HCL ou Plan	HCL ou Plan
Règles intégrées	1000+ règles	1600+ règles	200+ règles	500+ règles	200+ règles	400+ règles
AWS	Oui	Oui	Oui	Oui	Oui	Oui
Azure	Oui	Oui	Oui	Oui	Oui	Oui
GCP	Oui	Oui	Oui	Oui	Oui	Oui
Kubernetes	Non	Oui	Oui	Oui	Oui	Oui
GitHub	Non	Oui	Oui	Oui	Non	Non
Digital Ocean	Non	Oui	Oui	Non	Non	Non
OCI	Non	Oui	Oui	Non	Non	Non
OpenStack	Non	Oui	Oui	Non	Non	Non
CloudStack	Non	Non	Oui	Non	Non	Non
Linode	Non	Oui	Non	Non	Non	Non

c. Règles personnalisées

	TFLint	Checkov	Chekov 2.0+	TFSec	Terrascan	Regula	Snyk IaC
Règles persos.	Oui	Oui	Oui	Oui	Oui	Oui	Oui
Langage	Go	Python	YAML	YAML ou JSON	Rego	Rego	Rego
Complexité*	Moyenne (Go)	Faible	Faible	Faible	Moyenne (Rego)	Moyenne (Rego)	Moyenne (Rego)
Testabilité	Bonne (Go)	Bonne (Python)	Moyenne (Python + YAML + Terraform)	Mauvaise (faisable manuellement)	Mauvaise (faisable manuellement)	Très bonne (intégrée)	Très bonne (intégrée)
Expressivité	Bonne	Moyenne (pas de support pour les règles multi-ressources)	Bonne	Moyenne (les types de tests sont limités)	Bonne	Bonne	Bonne

Les champs marqués par (*) sont subjectifs.

d. Utilisabilité

	TFLint	Checkov	TFSec	Terrascan	Regula	Snyk IaC
Ignorer des règles via le code	Oui	Oui	Oui	Oui	Oui	Oui
Ignorer des règles via la CLI	Oui	Oui	Oui	Oui	Oui	Oui
Extension VS Code	Non	Oui	Oui	Non	Non	Oui
Extension IntelliJ	Non	Oui	Non	Non	Non	Oui

e. Conclusion

1. Checkov

Avantages :

- Gratuit ;
- Open source ;
- Populaire ;
- Nombreuses règles intégrées ;

- Nombreux fournisseurs de cloud pris en charge ;
- Règles personnalisées simples à écrire (Python ou YAML),
- Règles personnalisées testables et expressives (YAML),
- Bonne utilisabilité grâce à des extensions d'IDE.

Inconvénients :

- Pas de support pour les vérifications multi-ressources pour les règles personnalisées en Python ;
- Testabilité moyenne à cause de nombreux fichiers à mettre en place pour les règles personnalisées en YAML.

2. TFSec

Avantages :

- Gratuit ;
- Open source ;
- Populaire ;
- Nombreux fournisseurs de cloud pris en charge ;
- Règles personnalisées simples à écrire (YAML ou JSON) ;
- Bonne utilisabilité grâce à une extension VS Code.

Inconvénients :

- Peu de règles intégrées par défaut ;
- Règles personnalisées difficilement testables ;
- Règles personnalisées manquant d'expressivité.

3. Terrascan

Avantages :

- Gratuit ;
- Open source ;
- Populaire ;
- Nombreuses règles intégrées ;
- Règles personnalisées très expressives grâce à Rego (langage conçu pour définir des politiques de contrôle pour le cloud).

Inconvénients :

- Peu de fournisseurs de cloud pris en charge (comparé aux alternatives) ;
- Règles personnalisées plus complexes à écrire à cause de Rego. Ce langage n'est pas connu par l'équipe et requiert un temps d'apprentissage non négligeable (comparé aux alternatives) ;
- Règles personnalisées difficilement testables.

IV. Sélection des projets

1. Critères de sélection

La sélection des projets est influencée par :

- Taille du projet ;
- Popularité du projet ;
- Expérience de l'équipe de développement ;
- Régularité des mises à jour.

//TODO : Utiliser des critères qui ont plus de sens :
Version de Terraform actuellement utilisée
Date de création du premier fichier Terraform

2. Tableau des projets

Projet	Taille	Popularité	Expérience	Régularité	Contributeurs
StubbornJava	1214 ko	218 stars	Déc. 2016	361 commits	4
Atlantis	43 567 ko	4.4k stars	Mai 2017	2014 commits	206
Docker Android	242 262 ko	4.2k stars	Déc. 2016	525 commits	38
DetectionLab	198 188 ko	3.3k stars	Déc. 2017	541 commits	6
StreamAlert	44 411 ko	2.7k stars	Jan. 2017	1900 commits	30

V. Analyse des projets

Méthode d'analyse

//TODO: Comment nous avons fait les analyses.
//Quels outils, quels scripts, quelle méthode ?

Résultats des analyses

//TODO: Remplir le tableau avec les mauvaises pratiques détectées par les outils sur les projets sélectionnées.
//Donner le total, le nombre de mauvaises pratiques par criticité (sévères, moyennes, peu sévères).

Checkov

Toutes les mauvaises pratiques

Projet	Succès	Échecs	Total	Taux de succès
StubbornJava	24	42	66	36%
Atlantis	1	1	2	50%
Docker Android	68	21	89	76%
DetectionLab	17	18	35	48%
StreamAlert	608	34	642	94%

Projet	Faible	Moyenne	Haute	Critique
StubbornJava	14	14	14	0
Atlantis	0	1	0	0
Docker Android	138	16	6	0
DetectionLab	27	5	10	1
StreamAlert	28	3	3	0

Bonne pratique de sécurité : Identifiants

Checkov fournit initialement une grande quantité de règles (quelques centaines) permettant de vérifier la qualité ou la sécurité du code analysé.

Parmis toutes ces règles, nous avons décidé de nous préoccuper uniquement de celles concernant la sécurité des identifiants ou des secrets :

```
CKV_AWS_41          CKV_AWS_45          CKV_AWS_46
CKV_AWS_58          CKV_AWS_149         CKV_AZURE_41
CKV_BCW_1           CKV_GIT_4           CKV_LIN_1
CKV_OCI_1           CKV_OPENSTACK_1     CKV_PAN_1
```

Avec ces différentes règles, nous sommes en mesure d'exécuter une analyse sur un projet afin de vérifier l'utilisation de cette bonne pratique.

```
checkov -d <directory> --compact --framework terraform --check
CKV_AWS_41,CKV_AWS_45,CKV_AWS_46,CKV_AWS_58,CKV_AWS_149,CKV_AZURE_41,CKV_BCW_
1,CKV_GIT_4,CKV_LIN_1,CKV_OCI_1,CKV_OPENSTACK_1,CKV_PAN_1
```

Projet	Succès	Échecs	Total	Taux de succès
StubbornJava	4	0	4	100%
Atlantis	1	0	1	100%
Docker Android	0	0	0	100%
DetectionLab	5	0	5	100%
StreamAlert	18	0	18	100%

Quelques détails

StubbornJava:

- CKV_AWS_2: "Ensure ALB protocol is HTTPS"

Cette erreur démontre l'absence d'utilisation du protocole HTTPS sur la ressource ALB. Cette ressource d'AWS est exposée à internet, et devient donc vulnérable si un acteur extérieur obtient son adresse, car il sera en mesure de contacter nos instances EC2 via le load balancer en utilisant la technique d'attaque d'interception de trafic dite du "man-in-the-middle".
L'utilisation du protocole HTTP standard nous rend donc vulnérable aux attaques et compromettant nos identifiants.

- CKV_AWS_103: "Ensure that load balancer is using TLS 1.2"

Cette erreur rejoint le point précédent, car une version de la TLS inférieur à 1.2 ne comporte pas de mécanismes essentiels à la sécurité de nos identifiants (chiffrement du trafic, connexions SSL).

VI. Sources

- https://blog.christophetd.fr/shifting-cloud-security-left-scanning-infrastructure-as-code-for-security-issues/
- https://www.revolgy.com/insights/blog/complete-guide-for-picking-the-right-tool-for-terraform-security-code-analysis
- https://www.linkedin.com/pulse/iac-static-analysis-tools-terraform-khalid-ibnelbachyr/