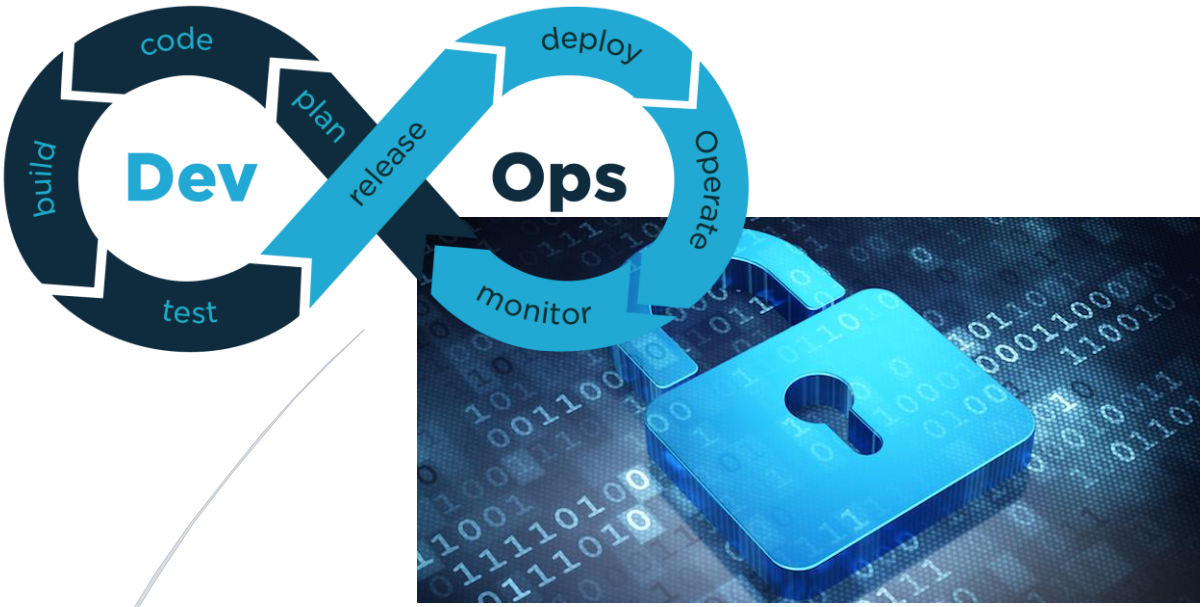
EVALUATION EN COURS DE FORMATION

PARCOURS ADMINISTRATEUR SYSTEME DEVOPS



Sécuriser l'Infrastructure

Evaluation :	**ECF3**
Version :	du 10/11/2025
Auteur :	FT

Evaluation	Parcours	ADMINISTRATEUR SYSTEME DEVOPS	Auteur	FT
ECF3	Titre	**Sécuriser l'Infrastructure**	Du	10/11/2025

Sommaire

Table des matières

Evaluation	Parcours	ADMINISTRATEUR SYSTEME DEVOPS	Auteur	FT
ECF3	Titre	**Sécuriser l'Infrastructure**	Du	10/11/2025

1. PRESENTATION

1.1. DESCRIPTION DE LA COMPETENCE – PROCESSUS DE MISE EN ŒUVRE

- À la demande d'un client ou de son responsable, mettre en place des pratiques de sécurité pour le provisioning de l'infrastructure.
- En tenant compte d'un cahier des charges, garder les secrets séparément du code provisioning, automatisé leurs usages et mettre en place un système de registre privé pour nos prochains modules.
- Mettre tout cela en service, tester le bon fonctionnement de toutes les parties et documenter son travail.

1.2. CONTEXTE(S) PROFESSIONNEL(S) DE MISE EN ŒUVRE

Ce travail s'effectue seul, en relation avec les équipes en charge du réseau et de la sécurité.

1.3. CRITERES DE PERFORMANCE

- Avoir un **repo git** sur lequel le code se trouve
- Avoir une **branche dev** sur laquelle toutes les étapes ont été commit
- Créer une **pull request**
- Avoir setup **Terraform cloud** en version control pour ce même repo git
- Terraform cloud doit être configuré sur la branche dev (pour la correction)
- **Aucun fichier** finissant par **tfvars** n'apparaît dans le repo git
- **Aucun secret ne fait partie des variables** du repo et/ou du Terraform cloud
- Le code terraform est configuré pour interagir avec un **serveur vault** (par exemple sur votre vm local)
- Vérifier que lors de **chaque merge** un **Plan** soit automatiquement lancer par **Terraform** cloud
- La configuration est conforme au cahier des charges
- Les documentations sont mises à jour
- Savoir-faire techniques, savoir-faire organisationnels, savoir-faire relationnels
- Documenter le code fourni
- Diagnostiquer un dysfonctionnement et le corriger
- Consulter de la documentation technique rédigée en anglais

2. ECF2 - QUESTIONS

2.1. QUELS SONT LES DANGERS D'EXPOSER SES VARIABLES / SECRETS ?

L'exposition des variables ou secrets représente un risque majeur de sécurité.

Les dangers principaux sont les suivants :

- **Compromission de services**
  Si une clé API ou des secrets sont exposés, un attaquant peut l'utiliser pour accéder au système (base de données, service cloud, etc) et de ce fait peut modifier la configuration ou le rendre inaccessible.

- **Accès non autorisés**
  Des informations comme des mots de passe, des clés API, des tokens d'authentification qui sont exposées peuvent permettre à des personnes mal intentionnées d'y accéder et de les utiliser à des fins frauduleuses ce qui rend vulnérable les systèmes cibles.

- **Vulnérabilités / Failles de sécurité**
  Un secret compromis peut être une porte d'entrée vers d'autres vulnérabilités comme injecter du code malveillant, ou déployer du code compromis.

- **Utilisation des ressources ou données**

  Un secret exposé peut donner accès à une base de données, des fichiers, contenant des informations sensibles (utilisateurs, mots de passe, …) et d'utiliser les ressources à des fins malhonnêtes.

- **Conséquences financières**

  Beaucoup de services cloud (AWS, GCP, Azure) facturent à l'usage et si des secrets sont volés, des personnes malintentionnées peuvent exploiter ses ressources et entrainer des conséquences financières en lançant des services non désirés.

2.2. POURQUOI GARDE-T-ON NOTRE CODE SUR UN VCS ? (EX: GITHUB / GITLAB / BITBUCKET)

Le code est gardé sur un VCS (Version Control System) qui est donc un système de gestion de versions, pour suivre le développement de logiciels.

Les raisons d'utiliser ce type d'outil sont les suivantes :

- **Historique des versions**

  Cela permet de garder une trace de toutes les modifications apportées au code, comme les ajouts ou les suppressions, et de les enregistrer avec un horodatage et un commentaire.

  Cela permet donc de revenir en arrière, de savoir qui a fait quoi, quand et pourquoi.

- **Effectuer un travail collaboratif**

  Plusieurs personnes peuvent travailler sur un même projet en parallèle par le biais de branches séparées, de suivre les tâches et de pouvoir effectuer des fusions quand les changements sont validés.

- **Gestions des Branches**

  Comme indiqué précédemment, il est possible de créer des branches pour ajouter des fonctionnalités au code principal, ou de corriger des bugs, pour ensuite fusionner ces changements ou corrections.

- **Sauvegarde centralisée**

  Le code n'est pas stocké uniquement sur la machine locale mais est aussi hébergé dans un environnement distribué, ce qui permet de stocker une copie complète du projet dans un espace sécurisé.

2.3. POURQUOI CREER UNE BRANCHE DE DEVELOPPEMENT ?

Il est possible de créer des branches pour ajouter des fonctionnalités au code principal, ou de corriger des bugs, pour ensuite fusionner ces changements ou corrections.

Cela permet donc plus précisément d'effectuer les actions suivantes :

- **Isoler le travail**

  Une branche permet de séparer le travail du code principal (branche master ou main) pour développer, tester sans impacter le projet principal.

- **Effectuer des développements en parallèle**

  Cela permet de travailler sur des fonctionnalités différentes et que chacun ait sa propre branche sans impacter le code principal ou le travail des autres collaborateurs.

- **Déploiement progressif**

  Une fois qu'une fonctionnalité est terminée sur une branche, elle peut être soumise à une "pull request", pour être lue et validée avant de l'intégrer à la branche principale.

  Cela permet des revues de code simplifiées dans une démarche Agile, en organisant le flux de travail de manière structurée et progressive.

2.4. À QUOI SERT UNE PULL REQUEST ?

Une Pull Request (sur GitHub) ou Merge (sur Gitlab) permet l'intégration d'une branche dans une autre, d'assurer la qualité et de travailler en équipe efficacement.

Cette demande est donc une demande de relecture à l'équipe ou au responsable du projet, pour valider les changements apportés dans une branche dans une autre branche qui est généralement la principale.

2.5. POURQUOI UTILISER UN GESTIONNAIRE DE SECRETS TEL QUE VAULT?

Les raisons principales d'utiliser un gestionnaire de secrets comme Vault sont les suivantes :

- **Centralisation des secrets**

  Les secrets sont stockés et chiffrés dans un "coffre-fort" sécurisé ce qui permet de protéger les données sensibles (mots de passe, tokens d'accès, clés API, certificats SSL, etc).

- **Contrôle d'accès granulaire**

  Vault permet de définir qui a accès à quoi de manière "fine", ce qui permet de limiter d'éventuelles compromissions.

- **Sécurité renforcée**

  Vault utilise des mécanismes de chiffrement, ou d'intégration avec des solutions d'authentification comme LDAP ou IAM cloud pour récupérer les secrets de manière sécurisé et automatisée sans les stocker localement.

- **Traçabilité des accès**

  Des logs ou journaux d'audit sont enregistrés à chaque demande d'accès pour savoir qui a accédé à quoi, quand et d'où.

2.6. COMMENT DEMARRER UN SERVEUR VAULT ?

Deux méthodes sont possibles pour démarrer un serveur Vault, la méthode avec un démarrage rapide, pour du développement ou des tests, en utilisant le mode "dev" qui utilise un stockage en mémoire (les données sont perdues lors d'un arrêt du serveur).

Mais pour un environnement de production, la configuration devra être plus robuste avec une configuration de sécurité appropriée, en mettant en place un stockage persistant.

- **Démarrage en mode "dev"**
  - Démarrage rapide : **vault server -dev** **# un root-token est fourni**
  - Configuration de la CLI
    - export VAULT_ADDR='http://127.0.0.1:8200'
    - export VAULT_TOKEN='<root-token>'
  - Vérifier le status : **vault status**
  - Gestion des secrets:
    - création : **vault kv put -mount=secret test key1=password** (dans le chemin secret/data/test)
    - pour le lire : **vault kv get -mount=secret test**
    - lister les secrets : **vault secrets list**

- **Démarrage en mode "prod"**

  Il faut créer un fichier de configuration (vault.hcl) pour définir le backend de stockage persistant (ex : S3, File), le listener HTTPS, les paramètres de sécurité, initialiser le serveur Vault (pour générer un ensemble de clés de déverrouillage et un token root).

  - Exemple de fichier **vault.hcl** (simplifié)

```
storage "raft" {          # raft : backend de stockage intégré pour conserver les données de Vault
  path    = "/opt/vault/data"
  node_id = "vault-server-1"
}
cluster_addr = "http://127.0.0.1:8201"
listener "tcp" {
  address     = "0.0.0.0:8200"
  tls_disable = "false"
  tls_cert_file = "/opt/vault/tls/vault.crt"
  tls_key_file  = "/opt/vault/tls/vault.key"
}
```

  - Démarrage du serveur : **vault server -config=vault.hcl**

2.7. QU'EST-CE QUE L'ENCRYPTAGE SSL?

SSL signifie **Secure Sockets Layer**, c'est un protocole de chiffrement qui sert à sécuriser la communication entre un navigateur et un serveur web, il a été remplacé par **TLS** (**Transport Layer Security**).

On identifie un site web utilisant SSL/TLS grâce à la présence de "HTTPS" au début de son adresse web (au lieu de "HTTP") et un petit cadenas affiché dans la barre d'adresse, en utilisant un certificat SSL nécessaire pour garantir l'établissement d'une connexion sécurisée.

Cela permet d'authentifier le serveur, de protéger les données sensibles et d'empêcher les attaques de type "man-in-the-middle" (interception de la connexion), et de garantir la confidentialité et la sécurité des communications sur internet.

SSl combine 2 types de chiffrement :

Phase	Type de chiffrement	Rôle
Échange de clé	Asymétrique 🔐	Protéger la clé de session
Communication	Symétrique 🔑	Vitesse + sécurité

3. Partie 1 : Gestion des Variables

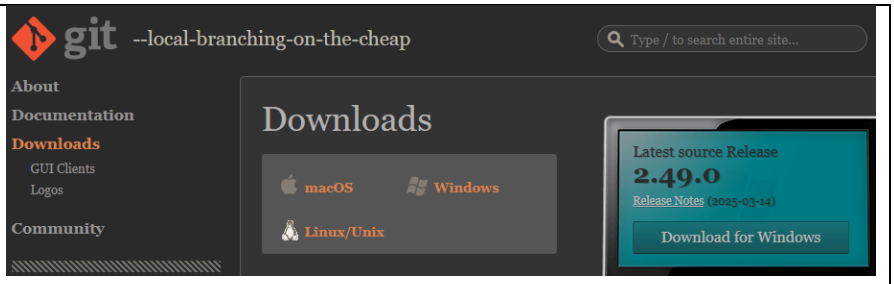
3.1. Creation du repo GIT

a. Repository GIT (Windows)

GIT peut être téléchargé via le lien suivant :

https://git-scm.com/downloads

L'installation se fera dans un environnement Windows.



- Création du répertoire de travail dans Git Bash: **git init terraformcloud-aws-vm**

```
$ git init terraformcloud-aws-vm
Initialized empty Git repository in C:/terraformcloud-aws-vm/.git/
```

- Création du fichier README.md

cd terraform-aws-vm

touch README.md

```
$ cd terraformcloud-aws-vm/

ulukai@DESKTOP-8VK8LT7 MINGW64 /c/terraformcloud-aws-vm (master)
$ touch README.md
```

- Ajouter des actions dans README.md qui seront effectuées

```
  GNU nano 8.3                                    README.md
Créer une repo git
Créer une branche dev
Pushez chaques prochaines étapes sur dev (pour la correction)
Créer une organisation et un workspace Terraform Cloud
Connecter ce workspace au repo précédemment mentionnée Terraform Cloud
Choisissez un provider parmi cette liste https://registry.terraform.io/browse/providers
Créer un compte sur le provider cloud AWS : https://aws.amazon.com/
Créez un module contenant une VM linux simple configuré à ce provider
SE référer à la documentation terraform pour celui-ci: https://registry.terraform.io/providers/hashicorp/aws/latest/docs
(Pas besoin de provisionner quoi que ce soit, faire juste en sorte que l'initialisation du provider fonctionne
et de pouvoir planifier une machine virtuelle sur le provider)
A la place de passer les clés api directement, passez les via des default variables dans un fichier nommé variables.tf
Enlevez ces variables du code et passer les via Terraform Cloud
```

- Ajout du fichier dans le dépôt et commit :

git add .

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md
```

git commit -m "Initial commit terraformcloud-aws-vm project"

```
$ git commit -m "Initial commit terraformcloud-aws-vm project"
[master (root-commit) 945bdd4] Initial commit terraformcloud-aws-vm project
 1 file changed, 13 insertions(+)
 create mode 100644 README.md
```

```
ulukai@DESKTOP-8VK8LT7 MINGW64 /c/terraformcloud-aws-vm (master)
$ git status
On branch master
nothing to commit, working tree clean
```

- Renommer la Branche principale : **git branch -M main**

```
ulukai@DESKTOP-8VK8LT7 MINGW64 /c/terraformcloud-aws-vm (master)
$ git branch -M main

ulukai@DESKTOP-8VK8LT7 MINGW64 /c/terraformcloud-aws-vm (main)
```

b. Repository GitHub

Le compte d'accès Github sera le suivant : https://github.com/fabrice-git-hub

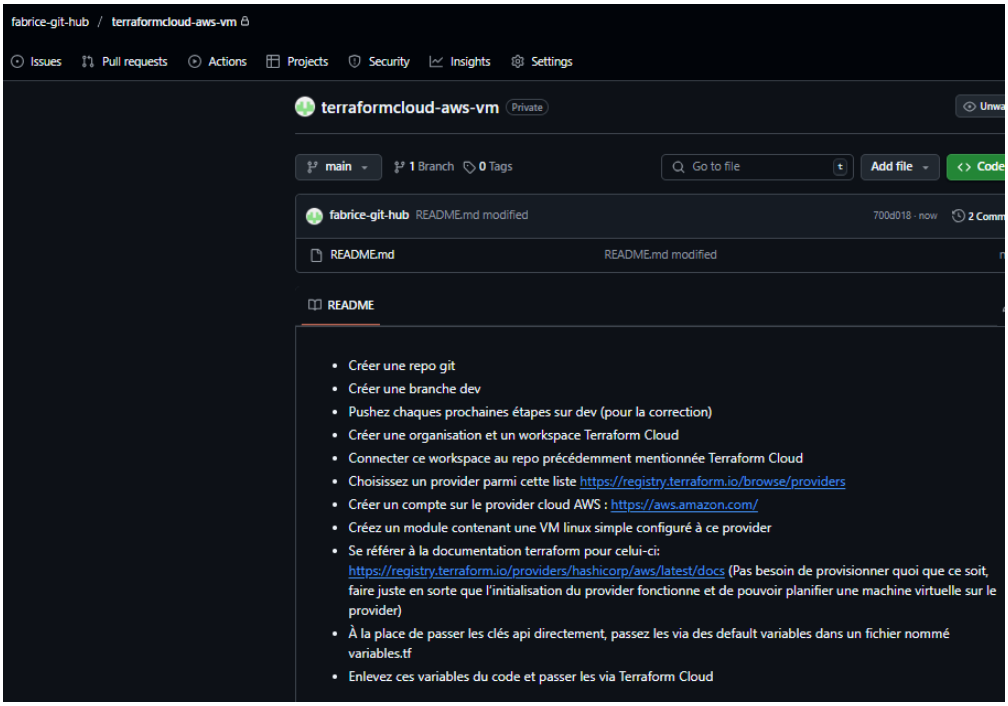
- Ajout Remote VCS et push sur GitHub

git remote add origin https://github.com/fabrice-git-hub/ec3.git

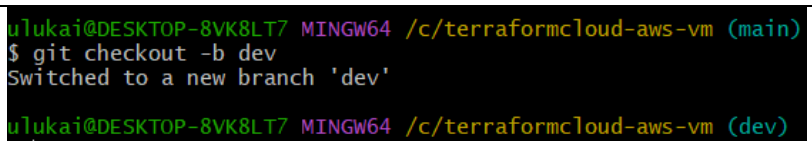
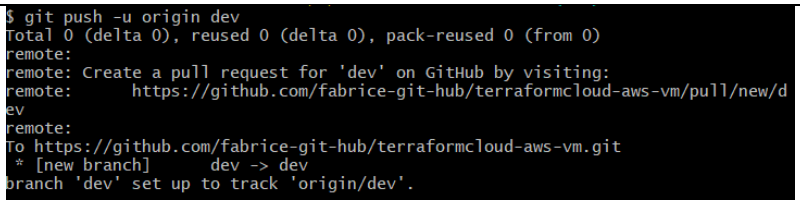
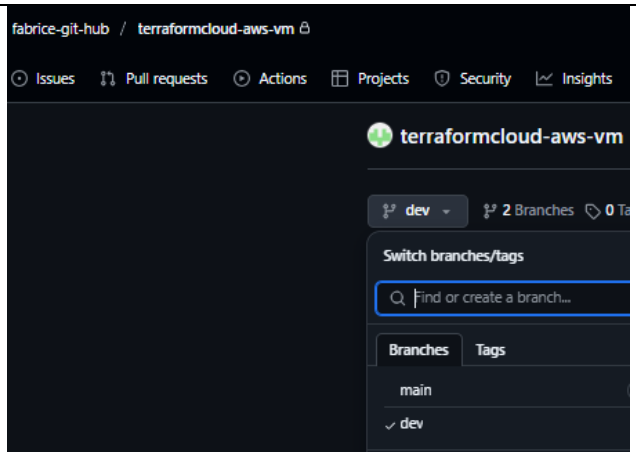
git push -u origin main

```
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 739 bytes | 123.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/fabrice-git-hub/terraformcloud-aws-vm.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

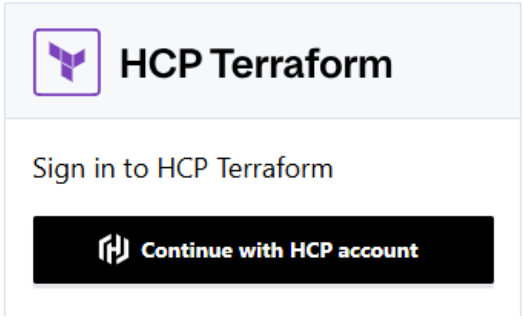
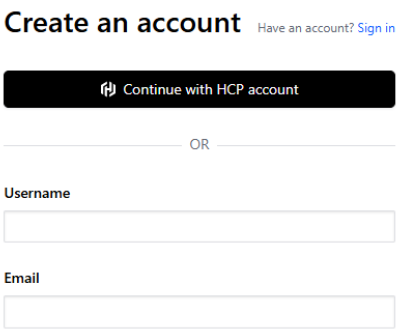
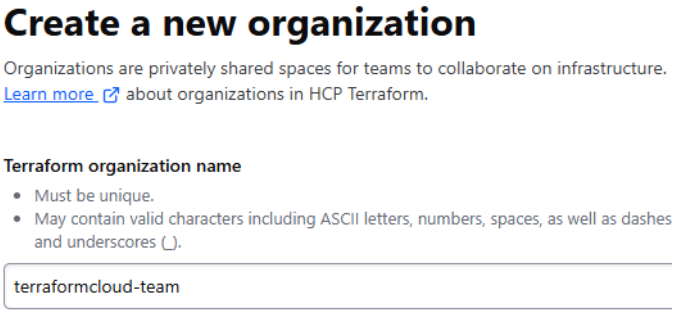
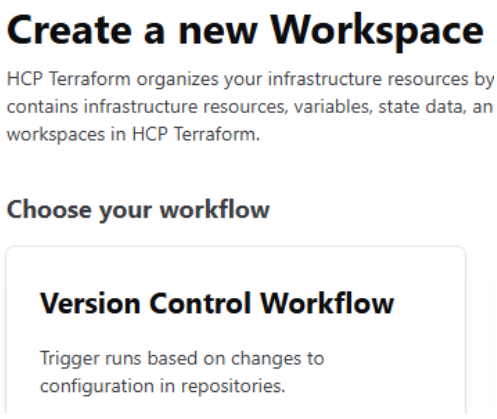
- Vérification sur Github

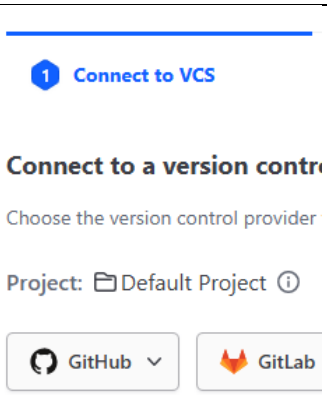
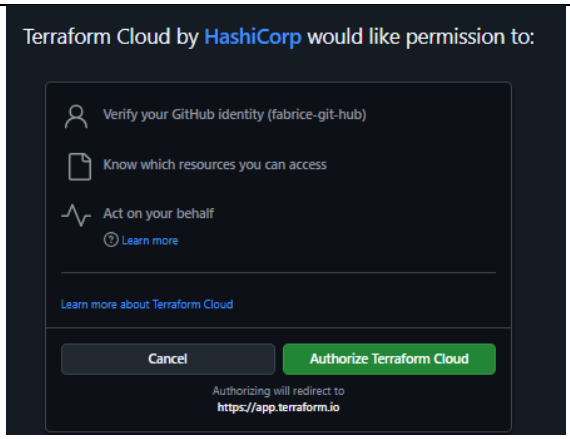
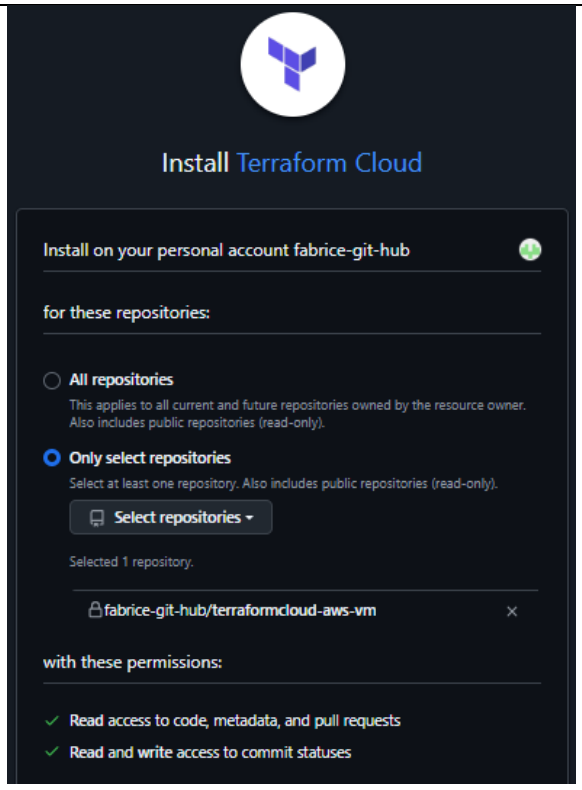


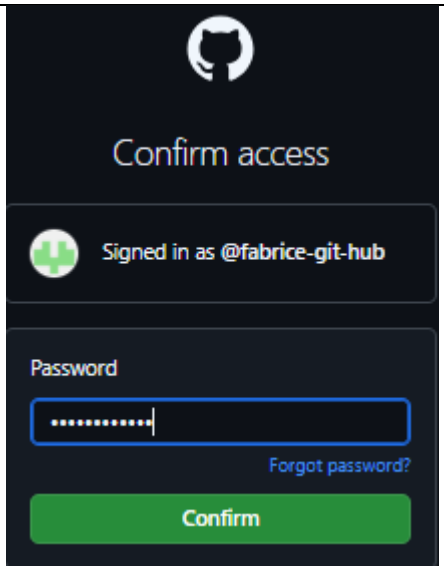
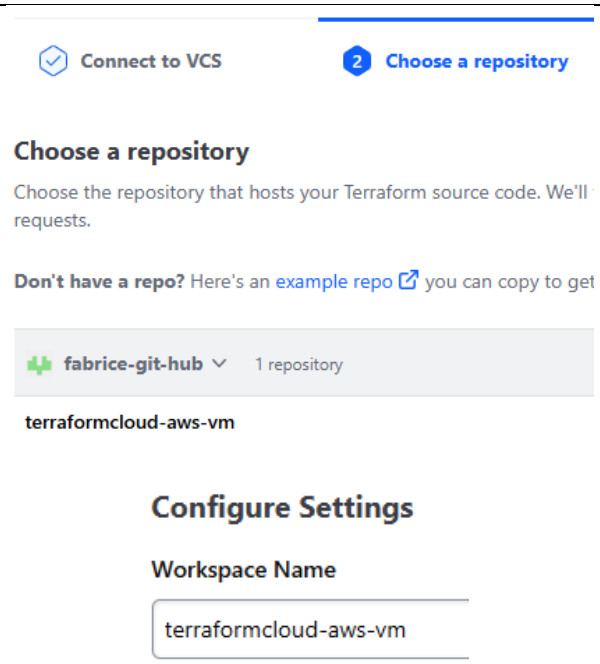
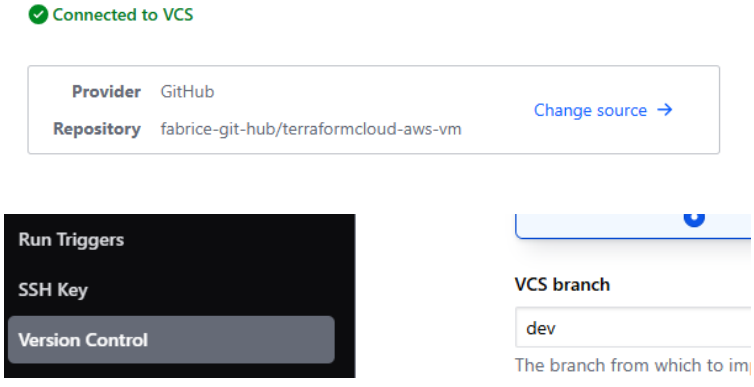
3.2. CREATION DE LA BRANCHE "DEV"

git checkout -b dev	
git push -u origin dev **Chaque prochaine étape sera "pushée"sur dev**	
• Vérification sur Github	

3.3. CREATION DE L'ORGANISATION ET DU WORKSPACE TERRAFORM CLOUD

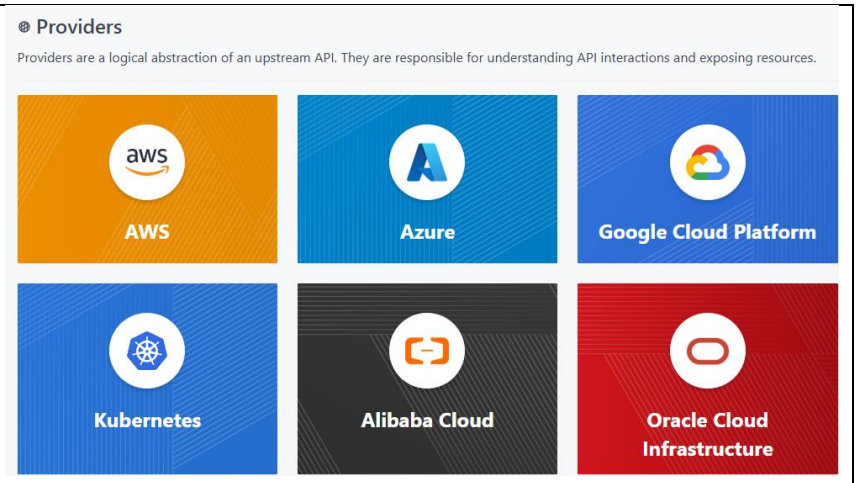
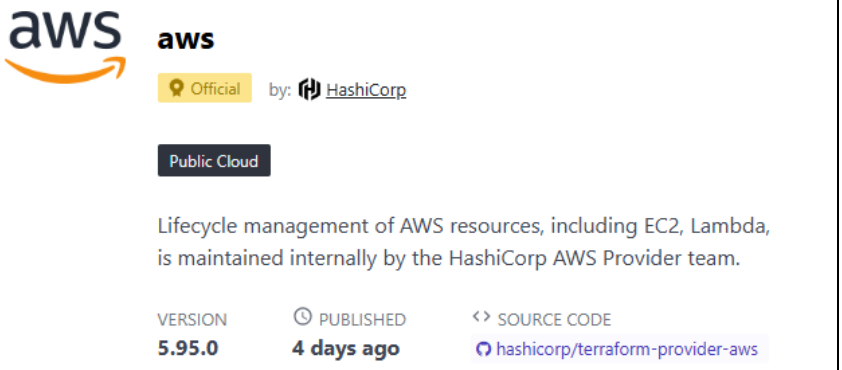
• Se connecter sur [Terraform Cloud]	
• Créer un nouveau Compte	
• Création de l'organisation "**terraformcloud-team**"	
• Création du Workspace "**terraformcloud-aws-vm**" • Sélectionner **Version Control Workflow**	

• Sélectionner le VCS **Github**	
• Autoriser **Terraform Cloud**	
• Sélectionner le repository : **terraformcloud-aws-vm**	

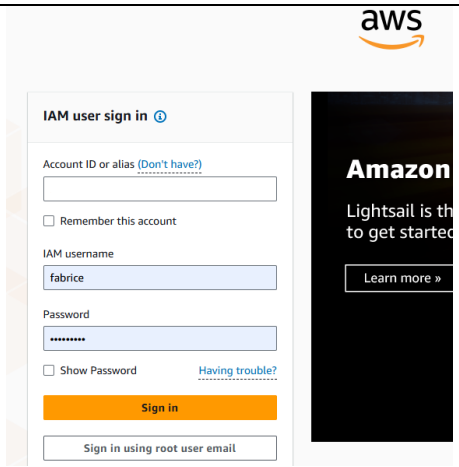
• Se connecter à **Github**	
• Finalisation de la création du Workspace	
• Configuration de Terraform Cloud sur la branche DEV	

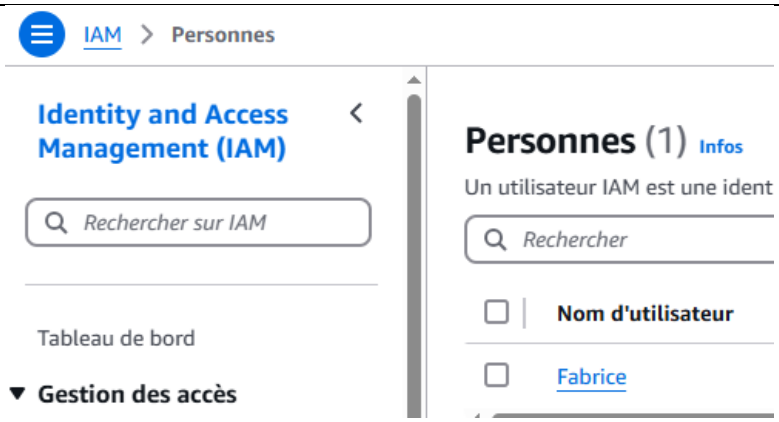
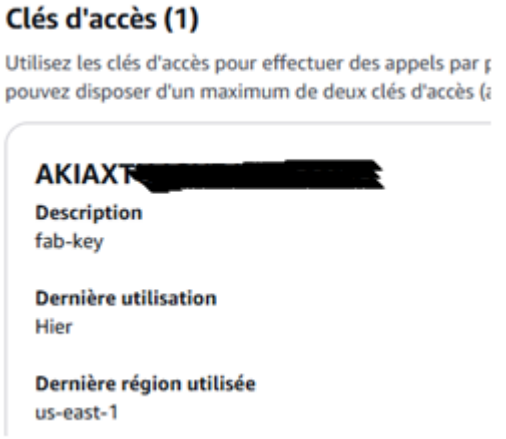
3.4. CHOIX DU PROVIDER

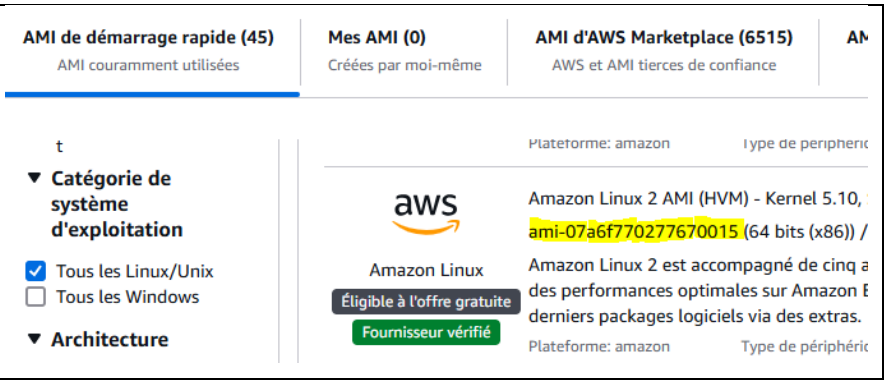
Sélectionner le provider sur [Terraform Providers]

• Choix du Provider	
• Dernière Version du provider AWS Terraform **5.95.0** • La version du binaire Terraform qui sera utilisée en local sera la **1.11.4**	

3.5. CREATION D'UN COMPTE SUR LE PROVIDER CLOUD AWS

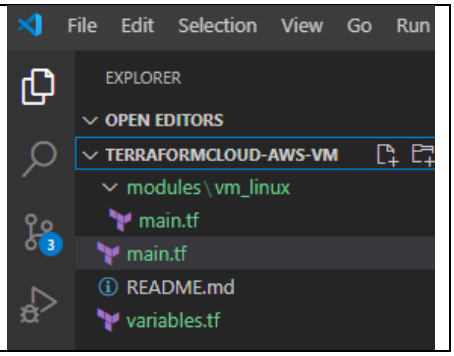
• Se connecter à [https://aws.amazon.com/] • Région choisie : us-east-1	

• Un compte root aura été créé ainsi qu'un compte IAM qui sera le compte de connexion utilisé pour les échanges avec terraform.	
• Création de clés d'accès (Access Key ID et Secret Access Key) pour permettre à Terraform de provisionner les ressources.	

• Choix de la VM Linux (dans le catalogue des images Amazon) qui sera utilisée	

3.6. CREATION D'UN MODULE CONTENANT UNE VM LINUX SIMPLE

- La documentation terraform servira de référence :
  https://registry.terraform.io/providers/hashicorp/aws/latest/docs

• L'outil **Visual Studio Code** sera utilisé pour l'écriture des fichiers de configuration .tf	

- Actions qui seront exécutées :
  - ✓ Pas de provisionnement
  - ✓ Faire juste une initialisation du provider
  - ✓ Lancer un "terraform plan" de la machine virtuelle
    (Pas besoin de provisionner quoi que ce soit, faire juste en sorte que l'initialisation du provider fonctionne et de pouvoir planifier une machine virtuelle sur le provider AWS).

a. Création du répertoire "modules" contenant le dossier "vm_linux"

• Création des répertoires dans terraformcloud-aws-vm : **mkdir -p modules/vm_linux**	
• Accès au sous-dossier : **cd modules/vm_linux**	• Création du fichier terraform : **touch main.tf**

```
$ mkdir -p modules/vm_linux

ulukai@DESKTOP-8VK8LT7 MINGW64 /c/terraformcloud-aws-vm (dev)
$ cd modules/vm_linux

ulukai@DESKTOP-8VK8LT7 MINGW64 /c/terraformcloud-aws-vm/modules/vm_linux (dev)
$ touch main.tf
```

- Le fichier **maint.tf** (dans "**modules/vm_linux**") contiendra la configuration de base pour la création d'une instance EC2 Linux (**AMI Amazon Linux 2**)

main.tf .\ U ● | main.tf ...\vm_linux U ×

modules > vm_linux > main.tf > ...

```
1  resource "aws_instance" "vm_linux" {
2    ami           = "ami-07a6f770277670015" # Image Amazon Linux 2
3    instance_type = "t2.micro"              # taille de l'instance EC2 (1 vCPU/ 1 GB)
4    tags = {
5      Name = "vm_linux"
6    }
7  }
```

b. Création du fichier principal main.tf (./terraformcloud-aws-vm)

Ce fichier contiendra la **configuration du provider AWS** ainsi que le renseignement des **clés API**.

- Accès au répertoire racine :

  cd /c/terraformcloud-aws-vm

- Création du fichier terraform : **touch main.tf**

```
ulukai@DESKTOP-8VK8LT7 MINGW64 /c/terraformcloud-aws-vm/modules/vm_linux (dev)
$ cd /c/terraformcloud-aws-vm/

ulukai@DESKTOP-8VK8LT7 MINGW64 /c/terraformcloud-aws-vm (dev)
$ touch main.tf
```

main.tf .\ U ● main.tf ...\vm_linux U

main.tf > ...

```hcl
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "5.95.0" # version du provider AWS terraform
    }
  }
  required_version = "1.11.4" # version du binaire terraform en local
}
provider "aws" {
  region     = "us-east-1" # déclaration de la région
  access_key = "AKIA                    "
  secret_key = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
}

module "vm_linux" {
  source = "./modules/vm_linux" # répertoire contenant le main.tf pour création de l'instance EC2
}
```

c. Initialisation et plan Terraform

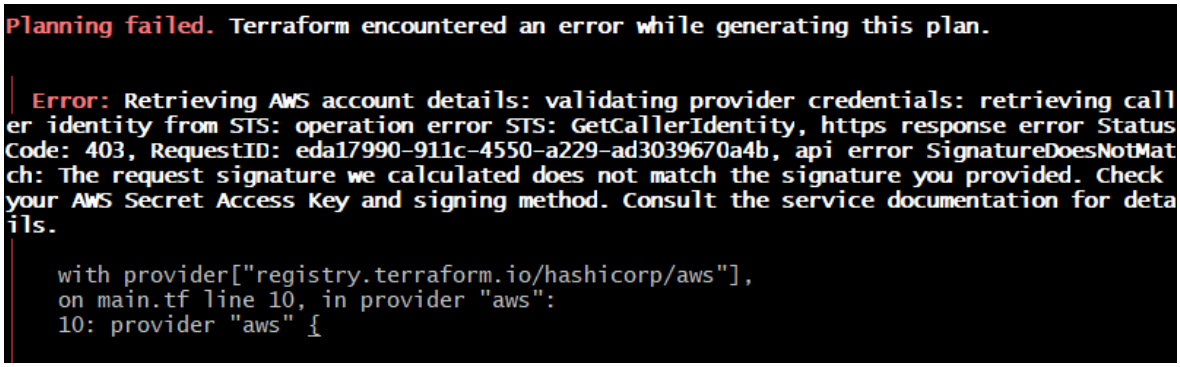
- **Initialisation** et téléchargement du plugin du provider AWS

```
ulukai@DESKTOP-8VK8LT7 MINGW64 /c/terraformcloud-aws-vm (dev)
$ terraform init -upgrade
Initializing the backend...
Upgrading modules...
- vm_linux in modules\vm_linux
Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.95.0"...
- Installing hashicorp/aws v5.95.0...
- Installed hashicorp/aws v5.95.0 (signed by HashiCorp)
Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

- Lancement du **plan terraform**

Résultat du terraform plan en ANNEXE - 5.1

Dans le cas d'un problème de connexion et d'authentification avec la clé, le message suivant apparaitra:



3.7. PASSAGE DES CLES API DANS LES VARIABLES

- Dans le répertoire racine : **terraformcloud-aws-vm**
- Création des fichiers des variables : **touch variables.tf** et **terraform.tfvars**

```
ulukai@DESKTOP-8VK8LT7 MINGW64 /c/terraformcloud-aws-vm (dev)
$ ls
README.md  main.tf  modules/  terraform.tfvars  variables.tf
```

- Modification **main.tf**

main.tf .\ U ● | main.tf ...\vm_linux U | variables.tf U | terraform.tfvars ●

main.tf > ...

```hcl
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "5.95.0" # version du provider AWS terraform
    }
  }
  required_version = "1.11.4" # version du binaire terraform en local
}
provider "aws" {
  region     = "us-east-1" # déclaration de la région
  access_key = var.aws_access_key
  secret_key = var.aws_secret_key
}

module "vm_linux" {
  source = "./modules/vm_linux" # répertoire contenant le main.tf pour création de l'instance EC2
}
```

- Fichier **variables.tf** (déclaration des variables)

variables.tf > variable "aws_secret_key"

```hcl
variable "aws_access_key" {
  type = string
  description = "AWS Access Key ID"
  sensitive = true      # empêche l'affichage de la valeur de la variable dans les sorties plan et apply
}

variable "aws_secret_key" {
  type = string
  description = "AWS Secret Access Key"
  sensitive = true      # empêche l'affichage de la valeur de la variable dans les sorties plan et apply
```

- Fichier **terraform.tfvars** (fournit les valeurs des variables)

terraform.tfvars > ...

```
aws_access_key = "AKIAX[illegible]"
aws_secret_key = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
```

- Création du fichier **.gitignore** et ignorer les fichiers et dossier suivants :

.gitignore

```
*.exe
*.hcl
*.tfvars
.terraform/
```

3.8. PASSAGE DES CLES API VIA TERRAFORM CLOUD

a. GIT

- Suppression des valeurs **aws_access_key** et **aws_secret_key** dans le fichier **terraform.tfvars**

terraform.tfvars > aws_secret_key

You, 14 minutes ago | 1 author (You)

```
aws_access_key = ""
aws_secret_key = ""
```

- Effectuer un Commit et Push sur dev :

git status

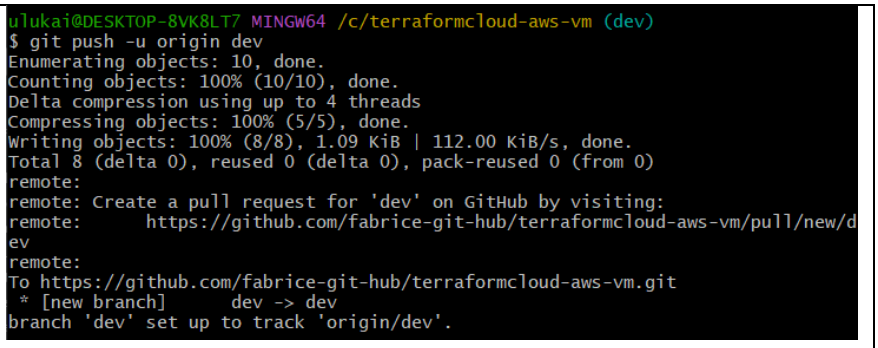
```
$ git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        main.tf
        modules/
        variables.tf
```

git add .

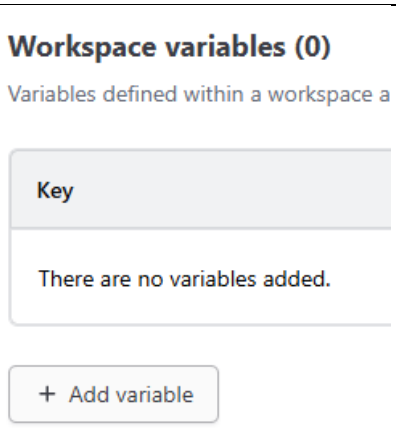
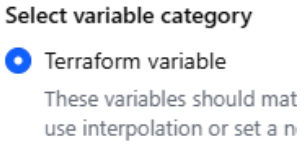
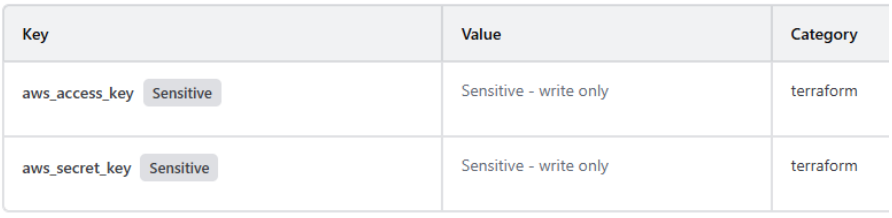
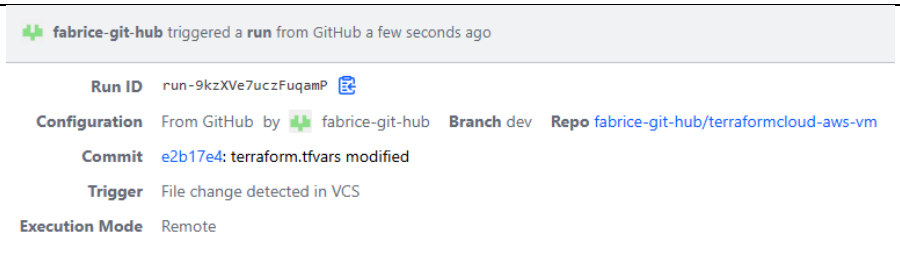
```
ulukai@DESKTOP-8VK8LT7 MINGW64 /c/terraformcloud-aws-vm (dev)
$ git status
On branch dev
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore
        new file:   main.tf
        new file:   modules/vm_linux/main.tf
        new file:   variables.tf
```

git commit -m "Ajout du module VM Linux"

```
ulukai@DESKTOP-8VK8LT7 MINGW64 /c/terraformcloud-aws-vm (dev)
$ git commit -m "Add VM Linux Module"
[dev 1f4e21d] Add VM Linux Module
 4 files changed, 35 insertions(+)
 create mode 100644 main.tf
 create mode 100644 modules/vm_linux/main.tf
 create mode 100644 variables.tf
```

git push origin dev	

b. Terraform Cloud

• Dans le Worspace Terraform Cloud/Variables -> **Add Variable**	
Sélectionner **Terraform Variable**	
Création des variables **aws_access_key** et **aws_secret_key** Renseigner le champ "**Value**" et marquer "Sensitive"	
Exécution automatique d'un PLAN après git push origin dev **Résultat du PLAN** pour provisionnement en ANNEXE - 5.2	

4. Partie 2 : VAULT - Passage de Secrets a la Volee

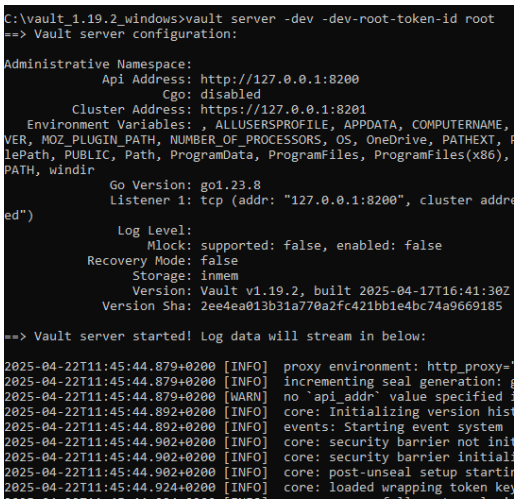
4.1. Mise en place de Vault (option UI configuree)

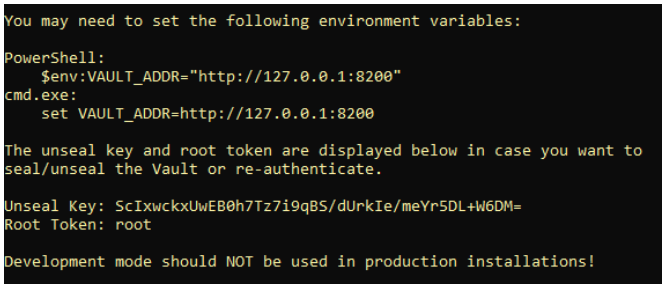
Pour les besoins de cette évaluation, VAULT sera lancé dans un environnement Windows en mode DEV.

- Téléchargement de Vault à l'adresse : Binaire Vault
- **Lancement de Vault** : **vault server -dev -dev-root-token-id root**
  - ✓ Le serveur Vault démarre et fournit une adresse pour accéder à l'UI **http://127.0.0.1:8200** ainsi qu'un token de root initial et les clés de déchiffrement.
  - ✓ Le serveur de développement écoute sur l'interface de bouclage 127.0.0.1 sur le port TCP 8200, sans TLS activé par défaut.

-dev : Active le mode développement, Vault s'exécute en mémoire et démarre en mode "unsealed".

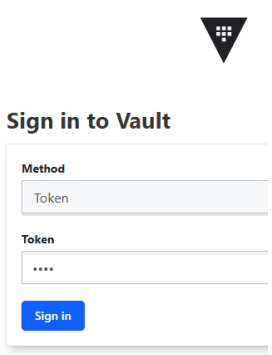
-dev -dev-root-token-id root : Initialise le jeton root, s'applique uniquement en mode « dev »et peut également être spécifié via la variable d'environnement VAULT_DEV_ROOT_TOKEN_ID.





- ✓ Ajout de la variable d'environnement qui demande au client de communiquer avec le serveur en mode dev : **set VAULT_ADDR=http://127.0.0.1:8200**

- **Connexion à l'UI**

URL : http://127.0.0.1:8200 Token d'accès : root	

4.2. Connecter le module Terraform au serveur Vault

Pour que Terraform interagisse avec le serveur Vault, il faut configurer le provider Vault dans le fichier de configuration **main.tf**.

Version du provider Vault : https://registry.terraform.io/providers/hashicorp/vault/latest/docs

- **Modification du fichier maint.tf**

✓ Ajout du provider	`provider "vault" {`
	`  address = "http://127.0.0.1:8200"`
	`  token   = "root"`
	`}`

✓ Ajout dans required_providers	`vault = {`
	`  source  = "hashicorp/vault"`
	`  version = "4.7.0" # version`
	`}`

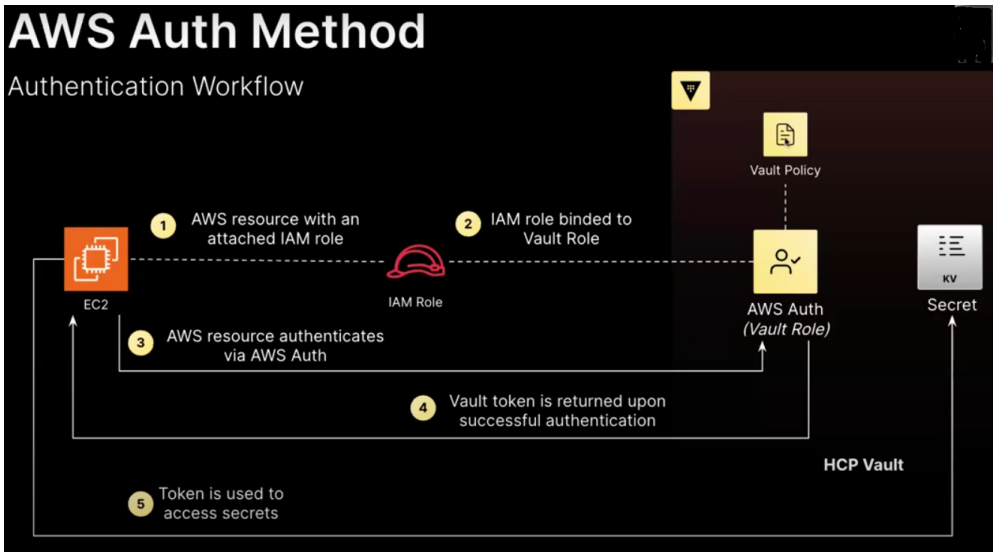
- Fichier **main.tf** complet en **ANNEXE - 5.3a**

4.3. Copie des cles API (UNSAFE et seulement pour la correction)

Copie des clés en local

```
set AWS_ACCESS_KEY_ID="AKIAXXXXXX"
set AWS_SECRET_ACCESS_KEY="XXXXXXXXXXXX"
```

4.4. Cles API renseignees par vault



a. Activer le moteur secret AWS dans Vault

Lancer la commande suivante pour générer des credentials temporaires AWS à la volée.

Ce moteur de secrets permet à Vault de générer des informations d'identification AWS dynamiques et à durée limitée.

vault secrets enable **-path=aws aws**

```
C:\vault_1.19.2_windows>vault secrets enable -path=aws aws
Success! Enabled the aws secrets engine at: aws/
```

b. Ajout des credentials dans Vault

- Configuration du moteur de secrets avec les clés d'accès AWS d'un utilisateur IAM disposant des permissions nécessaires :

vault write aws/config/root \
access_key=$AWS_ACCESS_KEY_ID \
secret_key=$AWS_SECRET_ACCESS_KEY \
region=us-east-1

```
C:\vault_1.19.2_windows>vault write aws/config/root access_key=$AWS_ACCESS_KEY_ID
Success! Data written to: aws/config/root

C:\vault_1.19.2_windows>vault write aws/config/root secret_key=$AWS_SECRET_ACCESS_KEY
Success! Data written to: aws/config/root

C:\vault_1.19.2_windows>vault write aws/config/root region=us-east-1
Success! Data written to: aws/config/root
```

- Détails de la configuration root du backend AWS monté dans Vault : **vault read aws/config/root**

```
C:\vault_1.19.2_windows>vault read aws/config/root
Key                           Value
---                           -----
access_key                    AKIAXTU4OY[redacted]
disable_automated_rotation    false
iam_endpoint                  n/a
identity_token_audience       n/a
identity_token_ttl            0s
max_retries                   -1
region                        us-east-1
role_arn                      n/a
rotation_period               0s
rotation_schedule             n/a
rotation_window               0
sts_endpoint                  n/a
sts_fallback_endpoints        []
sts_fallback_regions          []
sts_region                    n/a
username_template             {{ if (eq .Type "STS") }}{{
{{ else }}{{ printf "vault-%s-%s-%s" (printf "%s-%s" (.Dis
 | truncate 64 }}{{ end }}
```

c. Création d'un role AWS dans Vault

- Configurez un rôle Vault associé à un ensemble d'autorisations et à un type d'identifiant AWS.

vault write aws/roles/my-role credential_type=iam_user

```
C:\vault_1.19.2_windows>vault write aws/roles/my-role credential_type=iam_user
Success! Data written to: aws/roles/my-role
```

Explication :

Vault saura que lorsqu'on demandera des credentials avec ce rôle (my-role), il devra créer un utilisateur IAM dans AWS.

vault write aws/roles/my-role policy_document=-<C:\vault_1.19.2_windows\aws_role.json

aws_role.json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
      "iam:GetUser",
      "iam:CreateUser"
      ],
      "Resource": "*"
    }
  ]
}
```

```
C:\vault_1.19.2_windows>vault write aws/roles/my-role policy_document=-<C:\vault_1.19.2_windows\aws_role.json
Success! Data written to: aws/roles/my-role
```

Explication :

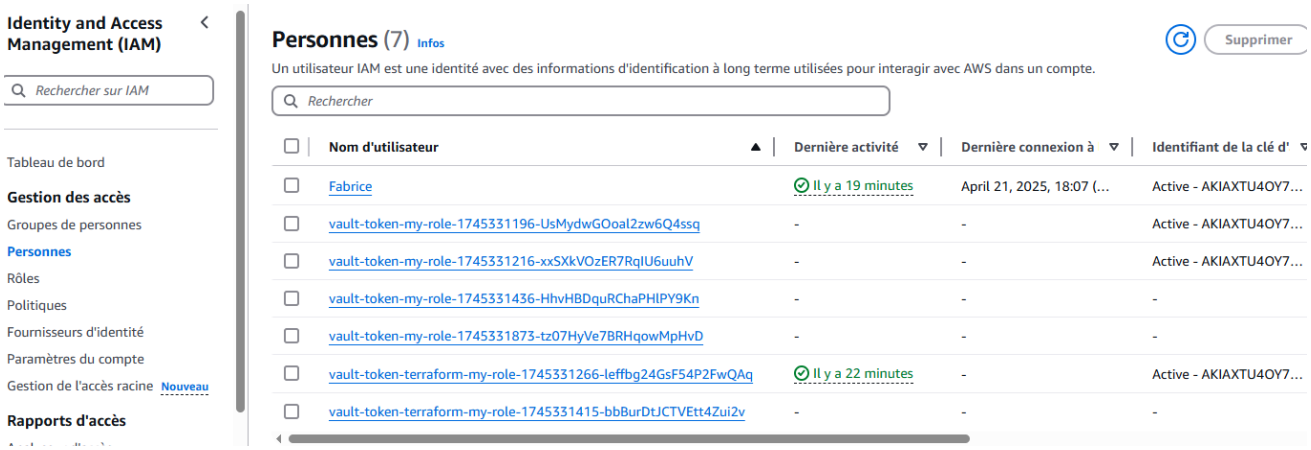
Cela met à jour le rôle AWS dans Vault avec une policy AWS définie dans un fichier JSON local.

 # policy_document : C'est une policy IAM au format JSON, qui définit ce que les utilisateurs/roles générés par Vault pourront faire sur AWS.

Vérification de la création du rôle
vault list aws/roles
	`C:\vault_1.19.2_windows>vault list aws/roles`
	`Keys`
	`----`
	`my-role`

Vérification obtention des informations d'identification AWS dynamiques et temporaires sécurisées gérées par Vault, basées sur la configuration du rôle IAM my-role
vault read aws/creds/my-role
	`C:\vault_1.19.2_windows>vault read aws/creds/my-role`
	`Key Value`
	`--- -----`
	`lease_id aws/creds/my-role/IJuTGCsB7c9Y5OWWXNz45OYV`
	`lease_duration 768h`
	`lease_renewable true`
	`access_key AKIAXTU`
	`secret_key JdRL5Vc`
	`session_token <nil>`

- Vérification dans la console AWS (à chaque nouvelle connexion de Vault à AWS un utilisateur est créé)



d. Modification du fichier main.tf

- Utilisation de la **data source Vault**
  Ce bloc demande à Vault de générer et récupérer des identifiants AWS dynamiques (clé d'accès et clé secrète) pour un rôle nommé my-role.

```
15  data "vault_aws_access_credentials" "creds" {
16    backend = "aws"
17    role    = "my-role"
18  }
    You, 5 seconds ago | 1 author (You)
19  provider "aws" {
20    region     = "us-east-1" # déclaration de la région
21    access_key = data.vault_aws_access_credentials.creds.access_key
22    secret_key = data.vault_aws_access_credentials.creds.secret_key
23  }
```

- Fichier **main.tf** complet en **ANNEXE - 5.3b**

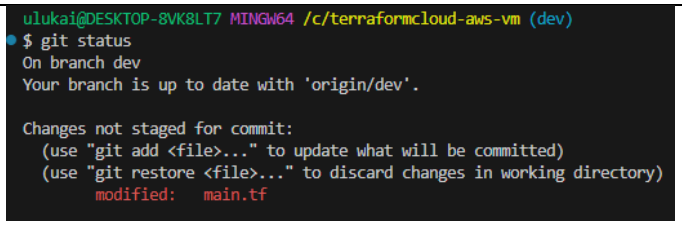
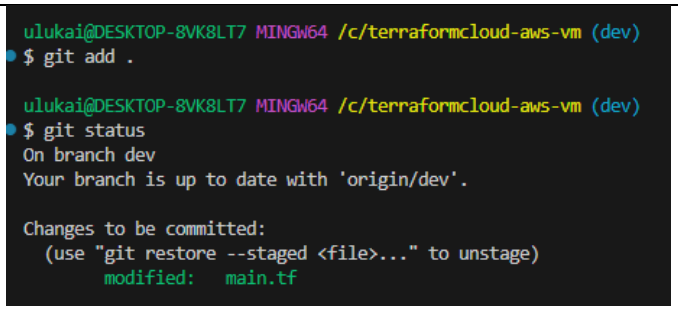
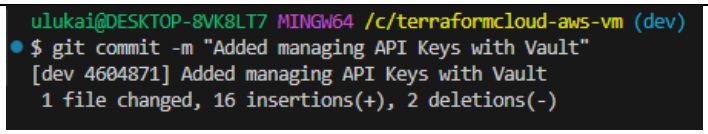
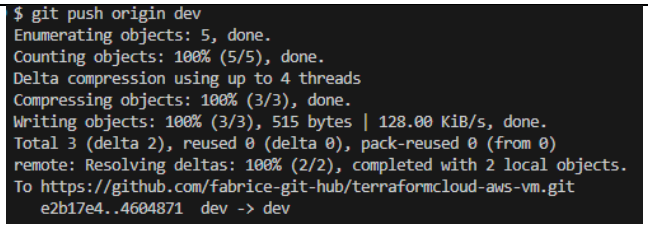
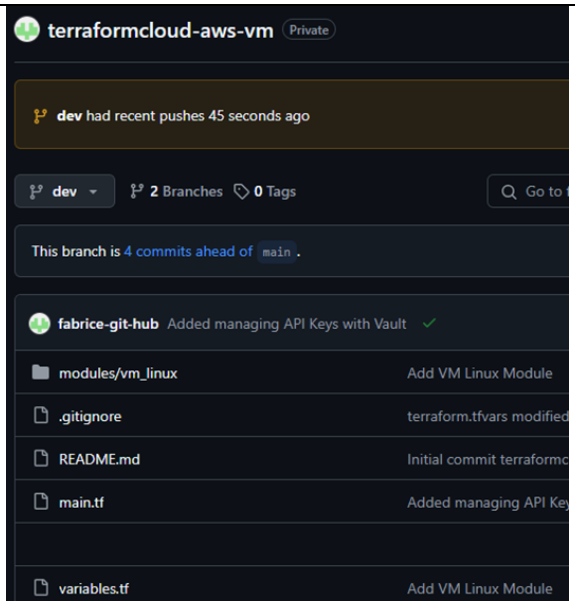
e. Terraform init

```
$ terraform init
Initializing the backend...
Initializing modules...
Initializing provider plugins...
- Finding hashicorp/vault versions matching "4.7.0"...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Installing hashicorp/vault v4.7.0...
- Installed hashicorp/vault v4.7.0 (signed by HashiCorp)
- Using previously-installed hashicorp/aws v5.95.0
```

f. Terraform plan (ANNEXE - 5.3c)

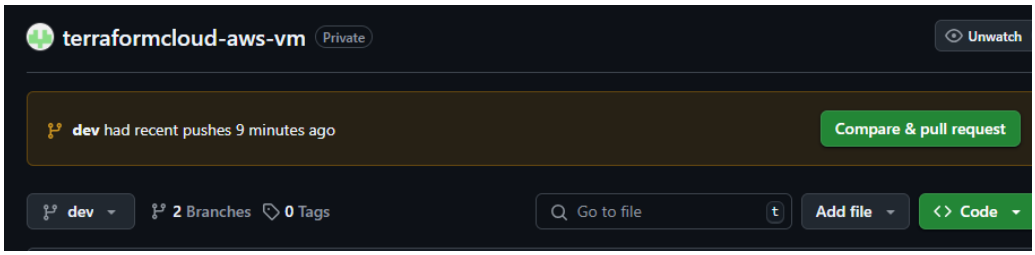
4.5. PULL REQUEST SUR MAIN(MASTER)

a. Push des modifications sur la branche DEV

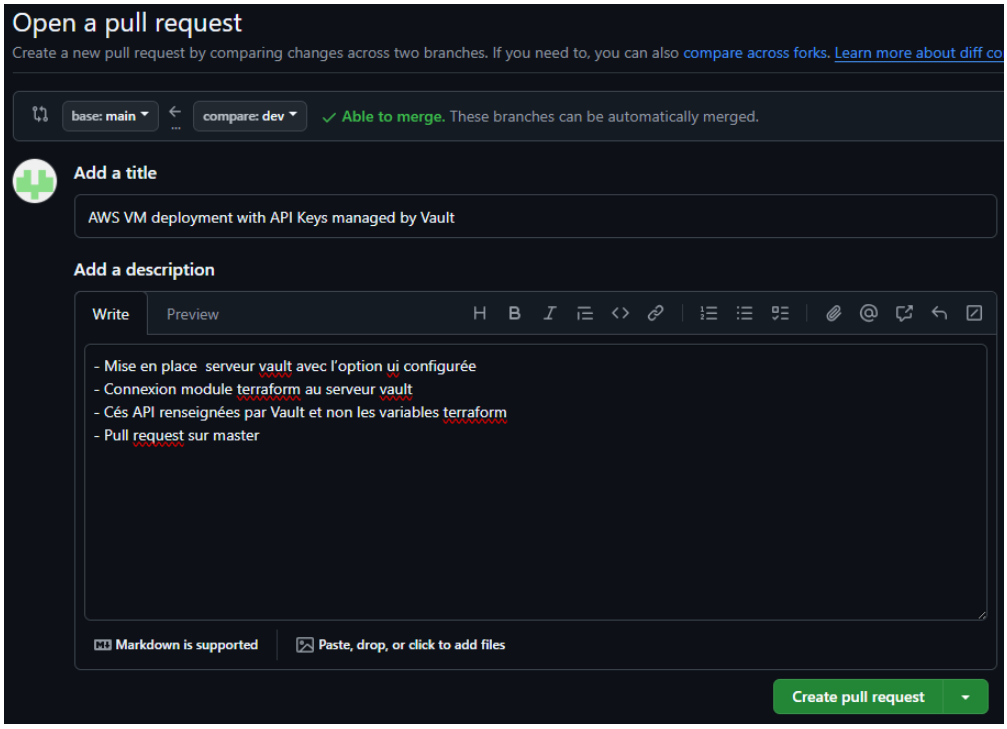
git status	
git add .	
git commit -m "Added managing API Keys with Vault"	
git push origin dev	
Vérification dans Github	

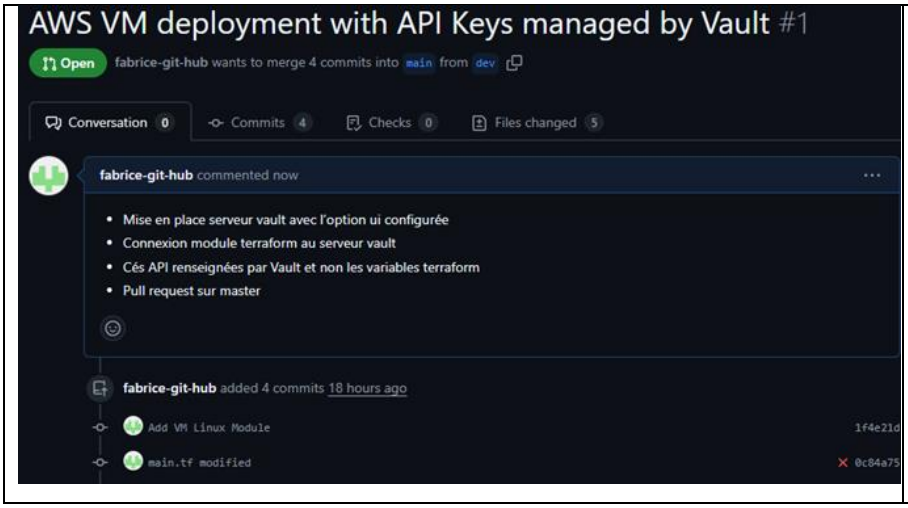
b. Pull Request sur Main (Master)

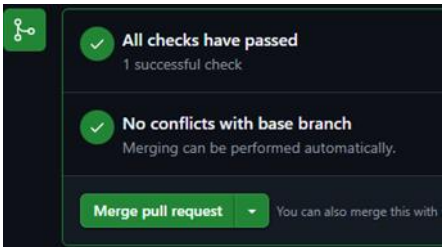
- Cliquer sur Compare & pull request



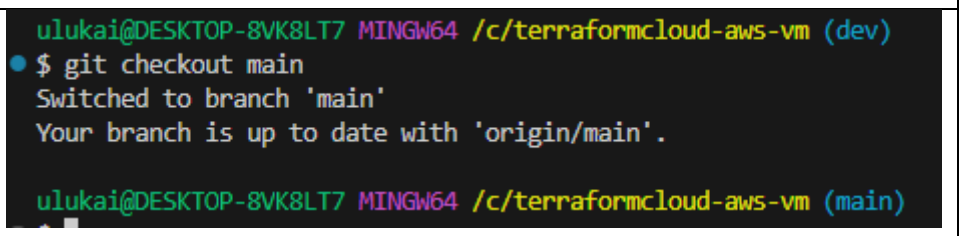
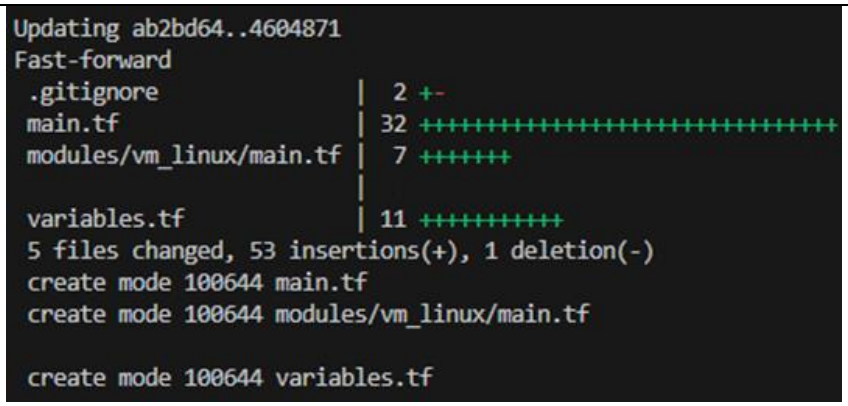
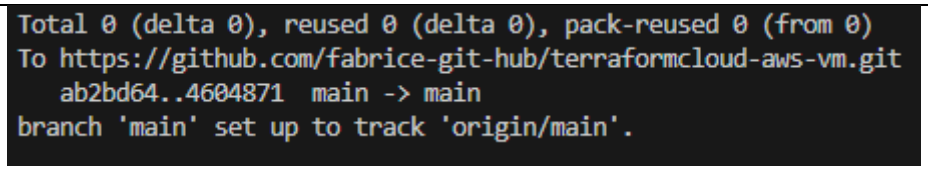
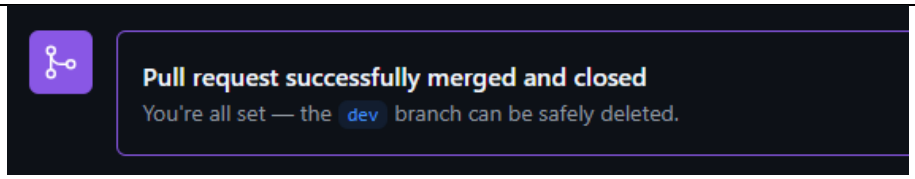
- Création Pull request







c. Merge Pull Request (Git CLI)

1. Mise à jour du dépôt local avec les dernières modifications : **git pull origin main**	
2. Bascule vers la branche de base de la pull request : **git checkout main**	
3. Fusionnez la branche principale avec la branche de base : **git merge dev**	
4. Envoyez les modifications : **git push -u origin main**	
Etat de la pull request sur Github	

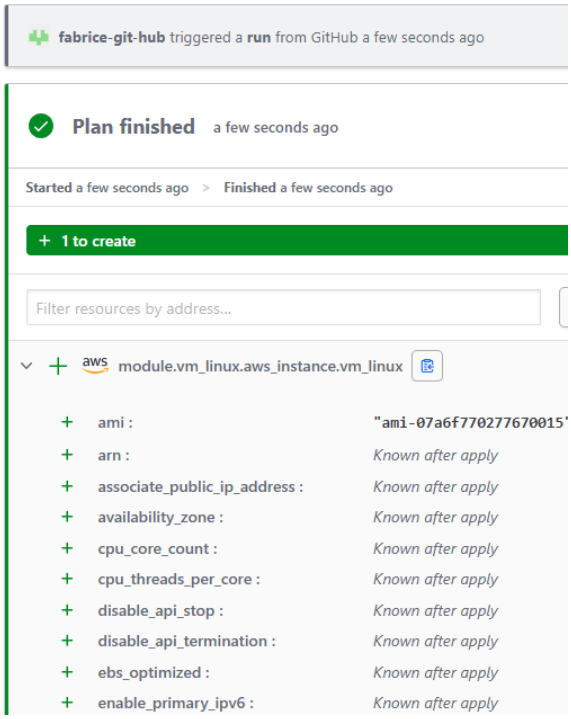
5. Annexe

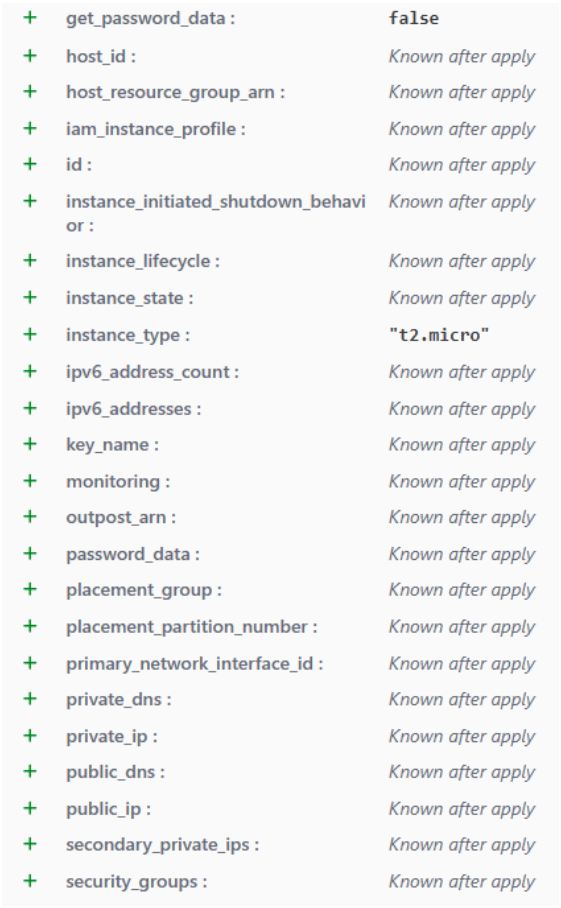
5.1. Creation d'un Module contenant une VM Linux simple

```
$ terraform plan

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # module.vm_linux.aws_instance.vm_linux will be created
  + resource "aws_instance" "vm_linux" {
      + ami                                  = "ami-07a6f770277670015"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + enable_primary_ipv6                  = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = (known after apply)
      + monitoring                           = (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + placement_partition_number           = (known after apply)
      + primary_network_interface_id         = (known after apply)
      + private_dns                          = (known after apply)
      + private_ip                           = (known after apply)
      + public_dns                           = (known after apply)
      + public_ip                            = (known after apply)
```

```
      + secondary_private_ips                = (known after apply)
      + security_groups                      = (known after apply)
      + source_dest_check                    = true
      + spot_instance_request_id             = (known after apply)
      + subnet_id                            = (known after apply)
      + tags                                 = {
          + "Name" = "vm_linux"
        }
      + tags_all                             = {
          + "Name" = "vm_linux"
        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)

      + capacity_reservation_specification (known after apply)

      + cpu_options (known after apply)

      + ebs_block_device (known after apply)

      + enclave_options (known after apply)

      + ephemeral_block_device (known after apply)

      + instance_market_options (known after apply)

      + maintenance_options (known after apply)

      + metadata_options (known after apply)

      + network_interface (known after apply)

      + private_dns_name_options (known after apply)

      + root_block_device (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

5.2. PASSAGE DES CLES API VIA TERRAFORM CLOUD





5.3. DECLARATION DE VAULT DANS TERRAFORM (MAIN.TF)

a. Connecter le module Terraform au serveur Vault

main.tf > terraform > required_providers > vault

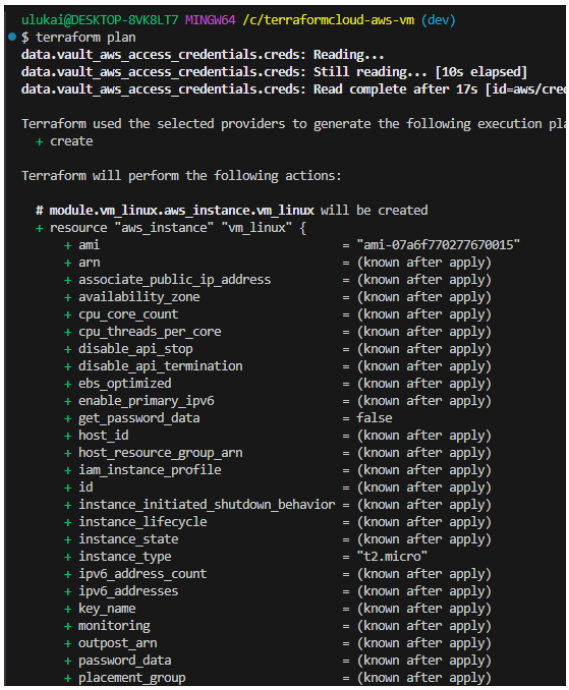
```
You, 4 seconds ago | 1 author (You)
terraform {
  You, 4 seconds ago | 1 author (You)
  required_providers {
    You, 13 hours ago | 1 author (You)
    aws = {
      source  = "hashicorp/aws"
      version = "5.95.0" # version du provider AWS terraform
    }
    You, 4 seconds ago | 1 author (You)
    vault = {
      source  = "hashicorp/vault"
      version = "4.7.0" # version du provider VAULT terraform
    }
  }
  required_version = "1.11.4" # version du binaire terraform en local
}
You, 12 hours ago | 1 author (You)
provider "aws" {
  region     = "us-east-1" # déclaration de la région
  access_key = var.aws_access_key
  secret_key = var.aws_secret_key
}

You, 4 seconds ago | 1 author (You)
provider "vault" {
  address = "http://127.0.0.1:8200"
  token   = "root"
}

You, 13 hours ago | 1 author (You)
module "vm_linux" {
  source = "./modules/vm_linux" # répertoire contenant le main.tf pour
}
```

b. Clés API renseignées par Vault

main.tf > module "vm_linux"

```
You, 14 minutes ago | 1 author (You)
terraform {
  You, 14 minutes ago | 1 author (You)
  required_providers {
    You, 15 hours ago | 1 author (You)
    aws = {
      source  = "hashicorp/aws"
      version = "5.95.0" # version du provider AWS terraform
    }
    You, 14 minutes ago | 1 author (You)
    vault = {
      source  = "hashicorp/vault"
      version = "4.7.0" # version du provider VAULT terraform
    }
  }
  required_version = "1.11.4" # version du binaire terraform en local
}

You, 14 minutes ago | 1 author (You)
data "vault_aws_access_credentials" "creds" {
  backend = "aws"
  role    = "my-role"
}
You, 14 minutes ago | 1 author (You)
provider "aws" {
  region     = "us-east-1" # déclaration de la région
  access_key = data.vault_aws_access_credentials.creds.access_key
  secret_key = data.vault_aws_access_credentials.creds.secret_key
}

You, 14 minutes ago | 1 author (You)
provider "vault" {
  address = "http://127.0.0.1:8200"
  token   = "root"
}

You, 15 hours ago | 1 author (You)
module "vm_linux" {
  source = "./modules/vm_linux" # répertoire contenant le main.tf pour création de l'instance EC2
}       You, 15 hours ago • Add VM Linux Module …
```

c. Terraform plan



```
ulukai@DESKTOP-8VK8LT7 MINGW64 /c/terraformcloud-aws-vm (dev)
$ terraform plan
data.vault_aws_access_credentials.creds: Reading...
data.vault_aws_access_credentials.creds: Still reading... [10s elapsed]
data.vault_aws_access_credentials.creds: Read complete after 17s [id=aws/cre

Terraform used the selected providers to generate the following execution pl
  + create

Terraform will perform the following actions:

  # module.vm_linux.aws_instance.vm_linux will be created
  + resource "aws_instance" "vm_linux" {
      + ami                                  = "ami-07a6f770277670015"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_stop                     = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + enable_primary_ipv6                  = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
      + instance_lifecycle                   = (known after apply)
      + instance_state                       = (known after apply)
      + instance_type                        = "t2.micro"
      + ipv6_address_count                   = (known after apply)
      + ipv6_addresses                       = (known after apply)
      + key_name                             = (known after apply)
      + monitoring                           = (known after apply)
      + outpost_arn                          = (known after apply)
      + password_data                        = (known after apply)
      + placement_group                      = (known after apply)
      + primary_network_interface_id         = (known after apply)
      + private_dns                          = (known after apply)
      + private_ip                           = (known after apply)
      + public_dns                           = (known after apply)
      + public_ip                            = (known after apply)
      + secondary_private_ips                = (known after apply)
      + security_groups                      = (known after apply)
      + source_dest_check                    = true
      + spot_instance_request_id             = (known after apply)
      + subnet_id                            = (known after apply)
      + tags                                 = {
          + "Name" = "vm_linux"
        }
      + tags_all                             = {
          + "Name" = "vm_linux"
        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + user_data_replace_on_change          = false
      + vpc_security_group_ids               = (known after apply)

      + capacity_reservation_specification (known after apply)

      + cpu_options (known after apply)

      + ebs_block_device (known after apply)

      + enclave_options (known after apply)

      + ephemeral_block_device (known after apply)

      + instance_market_options (known after apply)

      + maintenance_options (known after apply)

      + metadata_options (known after apply)

      + network_interface (known after apply)

      + private_dns_name_options (known after apply)

      + root_block_device (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```