

Lab 5: Continuous Integration (CI) and Continuous Delivery (CD) with Kubernetes

Objectif : Ce TD présente une approche complète de CI/CD, où l'on apprend à automatiser les tests d'application et d'infrastructure, à configurer une authentification OIDC avec AWS, ainsi qu'à automatiser les déploiements avec GitHub Actions et OpenTofu, en comprenant les différentes stratégies de déploiement.

Partie 1 : Intégration Continue (CI)

L'objectif principal de cette section est d'établir une intégration continue, qui implique de fusionner fréquemment le code dans la branche principale (souvent plusieurs fois par jour). Cela permet de détecter rapidement les problèmes d'intégration et de les résoudre.

Étapes principales :

1. **Préparation du projet** : On commence par se rendre dans le répertoire principal du projet, on crée un dossier pour la partie CI, puis on y copie l'application d'exemple à tester.

```
$ cd ~/devops_base  
$ git checkout main  
$ git pull origin main
```

```
$ mkdir -p td5  
$ cp -r td4/scripts/sample-app td5/scripts/sample-app
```

2. **Création du workflow GitHub Actions** : On crée un workflow GitHub Actions (.github/workflows/app-tests.yml) pour exécuter des tests automatiques sur chaque push. À chaque push, le workflow se déclenche. Il installe les dépendances (npm install). Il exécute les tests (npm test) pour vérifier que l'application fonctionne.

```
$ mkdir -p .github/workflows
$ cd .github/workflows
```

```
# .github/workflows/app-tests.yml (Example 5-2)
name: Sample App Tests

on: push

jobs:
  sample_app_tests:
    name: "Run Tests Using Jest"
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Install dependencies
        working-directory: td5/scripts/sample-app
        run: npm install
      - name: Run tests
        working-directory: td5/scripts/sample-app
        run: npm test
```

3. **Exécution du workflow** : On teste le workflow en ajoutant du code et en créant une pull request (PR) sur GitHub. Si un test échoue, il est corrigé, ce qui démontre l'importance de la détection précoce des erreurs.

```
$ git add td5/scripts/sample-app .github/workflows/app-tests.yml
$ git commit -m "Add sample-app and workflow"
$ git push origin main
```

```
$ git checkout -b test-workflow
```

On modifie l'application pour introduire une erreur volontaire :

```
// Example 5-3: Update the app response text (td5/scripts/sample-app/app.js)
res.send('DevOps Labs!');
```

On pousse cette modification :

```
$ git add td5/scripts/sample-app/app.js
$ git commit -m "Introduce intentional error"
$ git push origin test-workflow
```

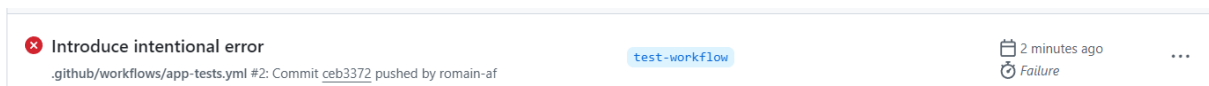
Voici nos captures d'écrans illustrant nos manipulations et les résultats obtenus :

```
sample-app/app.js
PS C:\Users\Romain\devops-base> git diff td5/scripts/sample-app/app.js*
diff --git a/td5/scripts/sample-app/app.js b/td5/scripts/sample-app/app.js
index b2d8a58..392a592 100644
--- a/td5/scripts/sample-app/app.js
+++ b/td5/scripts/sample-app/app.js
@@ -7,7 +7,7 @@ const port = 8080;
  const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
-    res.end('Hello, World!\n');
+    res.end('DevOps Labs!\n');
  });

  server.listen(port, hostname, () => {
PS C:\Users\Romain\devops-base> |
```

```
PS C:\Users\Romain\devops-base> git commit -m "Introduce intentional error"
[test-workflow ceb3372] Introduce intentional error
1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\Romain\devops-base> git push origin test-workflow
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 437 bytes | 437.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote:
remote: Create a pull request for 'test-workflow' on GitHub by visiting:
remote:   https://github.com/romain-af/devops-base/pull/new/test-workflo
remote:
To https://github.com/romain-af/devops-base.git
 * [new branch]      test-workflow -> test-workflow
```

Comme attendu, nous obtenons une erreur lors de notre push.



On met à jour le fichier de test app.test.js pour qu'il corresponde à la nouvelle réponse :

```
expect(response.text).toBe('DevOps Labs!');
```

Puis on push la correction et cette dernière marche :

4. Gestion des identifiants machine et authentification sécurisée : L'automatisation des tests d'infrastructure nécessite une connexion sécurisée à AWS. On ne doit jamais utiliser des identifiants utilisateurs classiques pour l'authentification, car ils sont statiques et potentiellement vulnérables.

Deux approches sont possibles :

1. Identifiants Machine : Un compte utilisateur AWS avec des permissions limitées
2. Identifiants à Provisionnement Automatique (OIDC - OpenID Connect)

5. Mise en place de l'authentification OIDC avec AWS : On configure AWS pour autoriser GitHub Actions à s'authentifier automatiquement en tant qu'identité AWS grâce à OpenID Connect.

On utilise le module Terraform github-aws-oidc pour déclarer GitHub comme un **fournisseur OIDC** :

```
provider "aws" {  
  region = "us-east-2" # Replace with your desired region  
}  
  
module "oidc_provider" {  
  source      = "github.com/your_github_name/devops-  
base//td5/scripts/tofu/modules/github-aws-oidc"  
  provider_url = "https://token.actions.githubusercontent.com"  
}
```

Une fois OIDC activé, on crée des **rôles IAM** pour **autoriser GitHub Actions** à exécuter les tests et déployer l'infrastructure.

On définit les rôles IAM dans Terraform :

```

provider "aws" {
  region = "eu-north-1"
}

module "oidc_provider" {
  source = "github.com/romain-af/devops-base//td5/scripts/tofu/modules/github-aws-oidc"

  provider_url = "https://token.actions.githubusercontent.com"
}

module "iam_roles" {
  source = "github.com/romain-af/devops-base//td5/scripts/tofu/modules/gh-actions-iam-roles"

  name          = "lambda-sample"
  oidc_provider_arn = module.oidc_provider.oidc_provider_arn

  enable_iam_role_for_testing = true

  # TODO: fill in your own repo name here!
  github_repo      = "romain-af/devops-base"
  lambda_base_name = "lambda-sample"

  enable_iam_role_for_plan = true
  enable_iam_role_for_apply = true

  # TODO: fill in your own bucket and table name here!
  tofu_state_bucket      = "tofu-state-romain"
  tofu_state_dynamodb_table = "tofu-state-lock"
}

```

On déploie ensuite les infrastructures :

```

cd td5/scripts/tofu/live/ci-cd-permissions
tofu init
tofu apply

```

Après l'exécution, on obtient trois ARN de rôles IAM :

- `lambda_test_role_arn` (pour tester l'infrastructure)
- `lambda_deploy_plan_role_arn` (pour simuler un déploiement)
- `lambda_deploy_apply_role_arn` (pour appliquer un déploiement)

On voit néanmoins sur ces screens que l'authentification OIDC est correctement mise en place et AWS reconnaît maintenant GitHub Actions comme un fournisseur d'identité.

Nom	Région AWS	Analyseur d'accès IAM	Date de création
tofu-state-romain	Europe (Stockholm) eu-north-1	Afficher l'analyseur pour eu-north-1	02 Feb 2025 06:59:45 PM CET

Tables (1)

Info

Rechercher des tables

N'importe quelle clé de balise

N'importe quelle valeur de b...

< 1 >

<input type="checkbox"/>	Nom	Statut	Clé de partition	Clé de tri	Index	Régions de réplication	Protection contre la suppression	Favori	M
<input type="checkbox"/>	tofu-state-lock	Actif	LockID (S)	-	0	0	Désactivé	☆	À

6. Exécution des tests automatisés d'infrastructure avec OpenTofu : Après la configuration de l'authentification et des rôles, on automatise les tests d'infrastructure avec OpenTofu.

On copie les modules d'infrastructure des TDs précédents. On met ensuite à jour les variables pour que les ressources puissent être configurées dynamiquement :

```
variable "name" {
  description = "The base name for the function and all other resources"
  type        = string
  default     = "lambda-sample"
}
```

Puis, on remplace les valeurs statiques par var.name :

```
#
module "function" {
  # ... (other params omitted) ...
  name = var.name
}

module "gateway" {
  # ... (other params omitted) ...
  name = var.name
}
```

On définit un **workflow GitHub Actions** (.github/workflows/infra-tests.yml) pour tester l'infrastructure :

```
# name: Infrastructure Tests

on: push

jobs:
  opentofu_test:
    name: "Run OpenTofu tests"
    runs-on: ubuntu-latest
    permissions:
      id-token: write
      contents: read
    steps:
      - uses: actions/checkout@v2

      - uses: aws-actions/configure-aws-credentials@v3
        with:
          # TODO: fill in your IAM role ARN!
          role-to-assume: arn:aws:iam::111111111111:role/lambda-sample-tests
          role-session-name: tests-${{ github.run_number }}-${{ github.actor }}
          aws-region: us-east-2

      - uses: opentofu/setup-opentofu@v1


      - name: Tofu Test
        env:
          TF_VAR_name: lambda-sample-${{ github.run_id }}
        working-directory: td5/scripts/tofu/live/lambda-sample
        run: |
          tofu init -backend=false -input=false
          tofu test -verbose
```

Le workflow s'exécute à chaque push. Il s'authentifie à AWS avec OIDC (sans clé d'accès). Il exécute les tests OpenTofu (tofu test).



On pousse le workflow et on vérifie l'exécution :

```
git add .
git commit -m "Add infrastructure tests workflow"
git push origin opentofu-tests
```

Puis, on crée un pull request et on observe le test dans l'onglet Actions de GitHub :

 Fix infrastructure tests and workflow

main

 now
 17s

...

Sample App Tests #11: Commit `1deb97e` pushed by romain-af

Le pull final est bien passé après les modifications et nous pouvons passer à la phase de déploiement CD.

Partie 2 : Livraison Continue (CD)

Une fois les tests validés, on met en place un pipeline CI/CD pour déployer automatiquement les modifications.

Un pipeline de déploiement automatise les différentes étapes du déploiement, de l'engagement du code (*commit*) jusqu'à la mise en production (*release*).

Il est préférable d'exécuter le pipeline sur un serveur de déploiement dédié afin d'assurer la cohérence, la répétabilité et la sécurité du processus.

1. Configuration du Backend Distant pour l'État d'OpenTofu

On crée un bucket S3 et une table DynamoDB pour stocker l'état d'OpenTofu de manière distante.

On configure AWS avec OpenTofu :

```
1 provider "aws" {
2   region = "eu-north-1"
3 }
4
5 module "state" {
6   source = "github.com/romain-af/devops-base/td5/scripts/tofu/modules/state-bucket"
7
8   # TODO: fill in your own bucket name!
9   name = "tofu-state-romain"
10 }
```

Pour faire un `tofu init` = et `apply`, on a du recréer deux nouvelles permissions afin d'y accéder depuis powershell avec tous les droits :

Vérifier

Les politiques suivantes seront attachées à cet utilisateur. [En savoir plus](#)

Détails de l'utilisateur

Nom d'utilisateur
romain_af

Résumé des autorisations (2)

< 1 >

Nom	Type	Utilisé comme
AmazonS3FullAccess	Gérées par AWS	Stratégie des autorisations
AmazonDynamoDBFullAccess	Gérées par AWS	Stratégie des autorisations

[Annuler](#)

[Précédent](#)

[Ajouter des autorisations](#)

On peut maintenant créer un fichier backend avec nos propres infos :

```
terraform {
  backend "s3" {
    # TODO: fill in your own bucket name here!
    bucket      = "tofu-state-romain"
    key         = "td5/scripts/tofu/live/tofu-state"
    region      = "eu-north-1"
    encrypt     = true
    # TODO: fill in your own DynamoDB table name here!
    dynamodb_table = "tofu-state-lock"
  }
}
```

Suite au run de la commande tofu init, nous observons bien le message suivant indiquant la bonne configuration du backend en tant que backend distant par défaut :

```

PS C:\Users\Romain\devops-base\td5\scripts\tofu\live\tofu-state> tofu init -reconfigure
>>

Initializing the backend...

Successfully configured the backend "s3"! OpenTofu will automatically
use this backend unless the backend configuration changes.
Initializing modules...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Installing hashicorp/aws v5.51.1...
- Installed hashicorp/aws v5.51.1 (signed, key ID 0C0AF313E5FD9F80)

Providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://opentofu.org/docs/cli/plugins/signing/

OpenTofu 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.

OpenTofu has been successfully initialized!

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

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

```

2. Création des Rôles IAM pour le Déploiement

On définit les permissions IAM pour permettre à GitHub Actions d'exécuter tofu plan et tofu apply.

Le main.tf est bien mis à jour :

```

C:\Users\Romain\devops-base\td5\scripts\tofu\live\ci-cd-permissions> main.tf
1  provider "aws" {
2    region = "eu-north-1"
3  }
4
5  module "oidc_provider" {
6    source = "github.com/romain-af/devops-base//td5/scripts/tofu/modules/github-aws-oidc"
7
8    provider_url = "https://token.actions.githubusercontent.com"
9  }
10
11
12  module "iam_roles" {
13    source = "github.com/romain-af/devops-base//td5/scripts/tofu/modules/gh-actions-iam-roles"
14
15    name = "lambda-sample"
16    oidc_provider_arn = module.oidc_provider.oidc_provider_arn
17
18    enable_iam_role_for_testing = true
19
20    # TODO: fill in your own repo name here!
21    github_repo = "romain-af/devops-base"
22    lambda_base_name = "lambda-sample"
23
24    enable_iam_role_for_plan = true
25    enable_iam_role_for_apply = true
26
27    # TODO: fill in your own bucket and table name here!
28    tofu_state_bucket = "tofu-state-romain"
29    tofu_state_dynamodb_table = "tofu-state-lock"

```

Les rôles IAM sont créés avec succès. GitHub Actions pourra maintenant interagir avec AWS en toute sécurité.

En faisant enfin un tofu apply :

```
Do you want to perform these actions?  
  OpenTofu will perform the actions described above.  
  Only 'yes' will be accepted to approve.  
  
Enter a value: yes
```

On selectionne yes et on obtient des valeurs de lambda en sortie (oublie de la capture d'écran)

3. Création des Workflows GitHub Actions pour le Déploiement

Nous allons maintenant créer deux fichiers dans notre workflow : `tofu_plan.yml` et `tofu_apply.yml`. Ces fichiers seront déclenchés lorsqu'une pull request sera effectuée sur la branche main, après des modifications dans le fichier `lambda-sample`.

Tofu_plan

```

name: Tofu Plan

on:
  pull_request:
    branches: ["main"]
    paths: ["td5/scripts/tofu/live/lambda-sample/**"]

jobs:
  plan:
    name: "Tofu Plan"
    runs-on: ubuntu-latest
    permissions:
      pull-requests: write
      id-token: write
      contents: read
    steps:
      - uses: actions/checkout@v2

      - uses: aws-actions/configure-aws-credentials@v3
        with:
          # TODO: fill in your IAM role ARN!
          role-to-assume: arn:aws:iam::111111111111:role/lambda-
sample-plan
github.actor }}
          role-session-name: plan-${{ github.run_number }}-${{
          aws-region: us-east-2

      - uses: opentofu/setup-opentofu@v1

      - name: tofu plan
        id: plan
        working-directory: td5/scripts/tofu/live/lambda-sample
        run: |
          tofu init -no-color -input=false
          tofu plan -no-color -input=false -lock=false

      - uses: peter-evans/create-or-update-comment@v4
        if: always()
        env:
          RESULT_EMOJI: ${ steps.plan.outcome == 'success' && '✅ '

```

Tofu_apply

```
permissions:
  pull-requests: write
  id-token: write
  contents: read
steps:
  - name: Checkout repository
    uses: actions/checkout@v2

  - name: Configure AWS credentials
    uses: aws-actions/configure-aws-credentials@v3
    with:
      role-to-assume: arn:aws:iam::111111111111:role/lambda-
sample-apply
      role-session-name: apply-${{ github.run_number }}-${{
github.actor }}
      aws-region: us-east-2

  - name: Setup OpenTofu
    uses: opentofu/setup-opentofu@v1

  - name: Tofu apply
    id: apply
    working-directory: td5/scripts/tofu/live/lambda-sample
    run: |
      tofu init -no-color -input=false
      tofu apply -no-color -input=false -lock-timeout=60m -auto-
approve

  - name: Find current PR
    uses: jwalton/gh-find-current-pr@master
    id: find_pr
    with:
      state: all

  - name: Create or update comment
    uses: peter-evans/create-or-update-comment@v4
    if: steps.find_pr.outputs.number
    env:
      RESULT_EMOJI: ${ steps.apply.outcome == 'success' &&
'✅' || '⚠️' }
```

Ce workflow est déclenché lors d'un push sur la branche main et si les fichiers modifiés se trouvent dans `td5/scripts/tofu/live/lambda-sample/**`.

Il utilise l'action `aws-actions/configure-aws-credentials@v3` pour s'authentifier auprès d'AWS via OIDC, en assumant le rôle IAM spécifié par le secret GitHub `APPLY_ROLE_ARN` (que vous devez créer et configurer avec l'ARN du rôle `apply`).

Le workflow exécute ensuite `terraform init` suivi de `terraform apply` pour appliquer les modifications.

Enfin, il utilise l'action `actions/github-script@v7` pour publier la sortie de la commande `terraform apply` sous forme de commentaire sur la pull request associée.

Nous avons rencontrés des difficultés par la suite car les fichiers git « taient trop volumineux ».

On commence donc par enlever terraform des commits effectués :

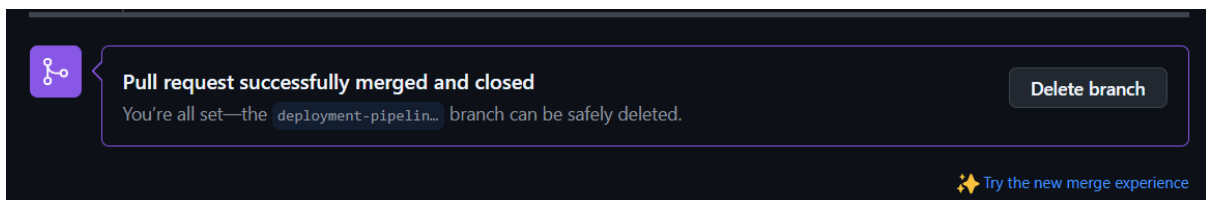
```
PS C:\Users\Romain\devops-base> git rm -r --cached td5/scripts/tofu/live/tofu-state/.terraform/
>>
rm 'td5/scripts/tofu/live/tofu-state/.terraform/modules/modules.json'
rm 'td5/scripts/tofu/live/tofu-state/.terraform/modules/state'
rm 'td5/scripts/tofu/live/tofu-state/.terraform/providers/registry.opentofu.org/hashicorp/aws/5.84.0/windows_amd64/CHANGELOG.md'
rm 'td5/scripts/tofu/live/tofu-state/.terraform/providers/registry.opentofu.org/hashicorp/aws/5.84.0/windows_amd64/LICENSE'
rm 'td5/scripts/tofu/live/tofu-state/.terraform/providers/registry.opentofu.org/hashicorp/aws/5.84.0/windows_amd64/README.md'
rm 'td5/scripts/tofu/live/tofu-state/.terraform/providers/registry.opentofu.org/hashicorp/aws/5.84.0/windows_amd64/terraform-provider-aws.exe'
rm 'td5/scripts/tofu/live/tofu-state/.terraform/terraform.tfstate'
```

On commit ces nouveaux fichiers de workflow directement sur la branche main, puis poussez-les sur GitHub :

```
$ git add .github/workflows
$ git commit -m "Add plan and apply workflows"
$ git push origin main
```

On va créer une nouvelle branche et on apporte une modification au module lambda pour le test.

On vérifie que le merge des deux branches a bien fonctionné



EXERCICE : Automatisation CI/CD avec OpenTofu et GitHub Actions

Voici une solution structurée qui nous permettra d'adapter notre pipeline afin de détecter automatiquement, lors d'un pull request ou d'un push, tous les dossiers contenant du code OpenTofu modifié (et non plus uniquement le module *lambda-sample*) et d'exécuter les commandes `tofu plan` et `tofu apply` pour chacun de ces dossiers en utilisant une stratégie de **matrix**.

Dans cette solution, nous allons créer deux workflows distincts :

- **tofu-plan.yml** : pour exécuter la commande tofu plan lors d'un pull request.
- **tofu-apply.yml** : pour exécuter la commande tofu apply lors d'un push sur la branche main.

Chaque workflow sera divisé en deux parties principales :

1. Détection des dossiers modifiés

Un job « detect » parcourt la liste des fichiers modifiés dans le répertoire `td5/scripts/tofu/live/` pour en déduire les dossiers racines (exemple : `td5/scripts/tofu/live/lambda-sample`). Il produit en sortie une matrice (au format JSON) contenant l'ensemble des dossiers modifiés.

2. Exécution en mode matrix

Un job principal utilise la stratégie matrix en se basant sur la sortie du job de détection. Pour chaque dossier, le job exécute les commandes tofu init et ensuite tofu plan ou tofu apply selon le cas, puis publie le résultat (par exemple sous forme de commentaire sur le pull request).

Voici le détail des changements, la démarche ainsi que l'intégralité du code à ajouter.

1. Workflow Tofu Plan

Ce workflow s'exécute lors d'un pull request sur la branche main pour tout changement intervenu dans le répertoire `td5/scripts/tofu/live/**`.

A. Démarche et explications

1. Détection des dossiers modifiés :

- **Action checkout :**

Récupération du code source pour disposer de l'historique Git.

- **Script de détection :**

Un script Bash utilise la commande `git diff` entre le commit de base et le HEAD pour lister les fichiers modifiés dans `td5/scripts/tofu/live/`.

Ensuite, il extrait les 4 premiers segments du chemin pour déterminer le dossier du module (par exemple, pour un fichier `td5/scripts/tofu/live/lambda-sample/src/index.js`, le dossier racine sera `td5/scripts/tofu/live/lambda-sample`).

Enfin, le script construit une liste unique de ces dossiers sous forme de JSON (grâce à `jq`) et la définit en sortie du job (via `::set-output` ou, dans les versions récentes, en écrivant dans la variable spéciale `$GITHUB_OUTPUT`).

2. Exécution du plan pour chaque dossier modifié :

- **Utilisation de la stratégie matrix :**

Le job « tofu_plan » récupère la liste de dossiers détectés (via la sortie du job précédent) et utilise cette liste pour lancer en parallèle un sous-job pour chaque dossier.

- **Configuration AWS :**

Chaque job configure l'authentification AWS en assumant le rôle défini dans vos GitHub Secrets (nommé ici PLAN_ROLE_ARN).

- **Exécution de tofu plan :**

Dans le dossier spécifique, la commande tofu init est d'abord exécutée pour initialiser le module, puis tofu plan est lancé. La sortie est récupérée et postée en commentaire sur le pull request grâce à l'action [peter-evans/create-or-update-comment](#).

B. Code complet du fichier .github/workflows/tofu-plan.yml

```
name: Tofu Plan (Matrix)

on:
  pull_request:
    branches: ["main"]
    paths:
      - "td5/scripts/tofu/live/**" # Déclenche pour tout changement dans le dossier OpenTofu

jobs:
  # 1. Job de détection des dossiers modifiés
  detect:
    runs-on: ubuntu-latest
    outputs:
      matrix: ${ steps.detect.outputs.matrix }
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Détecter les dossiers OpenTofu modifiés
        id: detect
        shell: bash
        run: |
          # Pour un pull request, on peut comparer le HEAD actuel avec le commit de base du PR.
          BASE=${{ github.event.before }}
          echo "Commit de base : $BASE"

          # Récupérer la liste des fichiers modifiés dans le répertoire td5/scripts/tofu/live/
          CHANGED=$(git diff --name-only $BASE HEAD | grep '^td5/scripts/tofu/live/' || true)
          echo "Fichiers modifiés :"
          echo "$CHANGED"

          # Extraire le dossier racine (ici on prend les 4 premiers segments du chemin)
          DIRS=$(echo "$CHANGED" | awk -F/ '{print $1"/"$2"/"$3"/"$4}' | sort | uniq)
          echo "Dossiers détectés :"
          echo "$DIRS"

          if [ -z "$DIRS" ]; then
```



```

    if [ -z "$DIRS" ]; then
        echo "Aucun dossier OpenTofu modifié."
        echo "::set-output name=matrix::[]"
        exit 0
    fi

    # Construire une matrice JSON à partir des dossiers
    MATRIX=$(printf '%s\n' $DIRS | jq -R . | jq -s .)
    echo "Matrice JSON générée : $MATRIX"
    echo "::set-output name=matrix::$MATRIX"

- name: Afficher la matrice générée
  run: echo "Matrice = ${ steps.detect.outputs.matrix }"

# 2. Job principal pour exécuter tofu plan en mode matrix
tofu_plan:
  needs: detect
  runs-on: ubuntu-latest
  strategy:
    matrix:
      folder: ${ fromJson(needs.detect.outputs.matrix) }
  steps:
    - name: Checkout repository
      uses: actions/checkout@v3

    - name: Configurer AWS pour le plan
      uses: aws-actions/configure-aws-credentials@v3
      with:
        role-to-assume: ${ secrets.PLAN_ROLE_ARN } # ARN stocké dans vos GitHub Secrets
        role-session-name: plan-${ github.run_number }-${ github.actor }
        aws-region: us-east-2

```

```

    aws-region: us-east-2

- name: Installer OpenTofu
  uses: opentofu/setup-opentofu@v1

- name: Exécuter tofu plan dans le dossier ${{ matrix.folder }}
  id: plan
  working-directory: ${{ matrix.folder }}
  shell: bash
  run: |
    echo "Initialisation du module dans $PWD"
    tofu init -no-color -input=false
    echo "Exécution de tofu plan..."
    tofu plan -no-color -input=false -lock=false > plan.txt
    echo "Sortie de tofu plan :"
    cat plan.txt
    # Récupérer la sortie dans une variable d'output
    echo "stdout<<EOF" >> $GITHUB_OUTPUT
    cat plan.txt >> $GITHUB_OUTPUT
    echo "EOF" >> $GITHUB_OUTPUT

- name: Publier le résultat dans un commentaire de PR
  uses: peter-evans/create-or-update-comment@v4
  if: always()
  with:
    issue-number: ${{ github.event.pull_request.number }}
    body: |
      ## Résultat de `tofu plan` pour le dossier **${{ matrix.folder }}**
      ...
      ${{ steps.plan.outputs.stdout }}
      ...

```

2. Workflow Tofu Apply

Ce workflow s'exécute lors d'un push sur la branche main et déploie les changements pour chacun des dossiers OpenTofu modifiés.

A. Démarche et explications

1. Détection des dossiers modifiés :

La méthode est similaire à celle du workflow précédent. Le job de détection utilise la commande `git diff` pour récupérer la liste des fichiers modifiés, puis en déduit les dossiers racines (module).

2. Exécution de tofu apply en mode matrix :

- **Configuration AWS :**

Le job configure les identifiants AWS en utilisant le rôle stocké dans le secret APPLY_ROLE_ARN.

- **Exécution des commandes :**

Dans chaque dossier, le job effectue tofu init puis tofu apply avec l'option -autoapprove pour appliquer automatiquement les changements. La sortie est capturée et affichée, puis postée en commentaire sur le pull request (si disponible).

B. Code complet du fichier .github/workflows/tofu-apply.yml

```
name: Tofu Apply (Matrix)

on:
  push:
    branches: ["main"]
    paths:
      - "td5/scripts/tofu/live/**" # Déclenche pour tout changement dans le dossier OpenTofu

jobs:
  # 1. Job de détection des dossiers modifiés
  detect:
    runs-on: ubuntu-latest
    outputs:
      matrix: ${ steps.detect.outputs.matrix }
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Détecter les dossiers OpenTofu modifiés
        id: detect
        shell: bash
        run: |
          # Pour un push, utiliser le commit précédent pour la comparaison
          BASE=${{ github.event.before }}
          echo "Commit de base : $BASE"

          # Liste des fichiers modifiés dans td5/scripts/tofu/live/
          CHANGED=$(git diff --name-only $BASE HEAD | grep '^td5/scripts/tofu/live/' || true)
          echo "Fichiers modifiés : "
          echo "$CHANGED"

          # Extraire le dossier racine pour chaque module (4 premiers segments du chemin)
          DIRS=$(echo "$CHANGED" | awk -F/ '{print $1"/"$2"/"$3"/"$4}' | sort | uniq)
          echo "Dossiers détectés : "
          echo "$DIRS"

          if [ -z "$DIRS" ]; then
```

```

    if [ -z "$DIRS" ]; then
        echo "Aucun dossier OpenTofu modifié."
        echo "::set-output name=matrix::[]"
        exit 0
    fi

    # Créer la matrice JSON
    MATRIX=$(printf '%s\n' $DIRS | jq -R . | jq -s .)
    echo "Matrice JSON générée : $MATRIX"
    echo "::set-output name=matrix::$MATRIX"

- name: Afficher la matrice générée
  run: echo "Matrice = ${ steps.detect.outputs.matrix }"

# 2. Job principal pour exécuter tofu apply en mode matrix
tofu_apply:
  needs: detect
  runs-on: ubuntu-latest
  strategy:
    matrix:
      folder: ${ fromJson(needs.detect.outputs.matrix) }
  steps:
    - name: Checkout repository
      uses: actions/checkout@v3

    - name: Configurer AWS pour l'apply
      uses: aws-actions/configure-aws-credentials@v3
      with:
        role-to-assume: ${ secrets.APPLY_ROLE_ARN } # ARN stocké dans vos GitHub Secrets
        role-session-name: apply-${ github.run_number }-${ github.actor }
        aws-region: us-east-2

```

```

    aws-region: us-east-2

- name: Installer OpenTofu
  uses: opentofu/setup-opentofu@v1

- name: Exécuter tofu apply dans le dossier ${ matrix.folder }
  id: apply
  working-directory: ${ matrix.folder }
  shell: bash
  run: |
    echo "Initialisation du module dans $PWD"
    tofu init -no-color -input=false
    echo "Exécution de tofu apply..."
    tofu apply -no-color -input=false -lock-timeout=60m -autoapprove > apply.txt
    echo "Sortie de tofu apply :"
    cat apply.txt
    # Récupérer la sortie dans une variable d'output
    echo "stdout<<EOF" >> $GITHUB_OUTPUT
    cat apply.txt >> $GITHUB_OUTPUT
    echo "EOF" >> $GITHUB_OUTPUT

- name: Publier le résultat dans un commentaire de PR (si applicable)
  uses: peter-evans/create-or-update-comment@v4
  if: github.event_name == 'pull_request'
  with:
    issue-number: ${ github.event.pull_request.number }
    body: |
      ## Résultat de `tofu apply` pour le dossier **${ matrix.folder }**
      ~~~
      ${ steps.apply.outputs.stdout }
      ~~~

```

3. Points Importants et Résumé des Changements

- **Organisation de la détection :**

Nous avons ajouté un job « detect » qui parcourt les fichiers modifiés et détermine, grâce à un script Bash, quels dossiers (correspondant aux modules OpenTofu) ont été affectés. Le résultat est converti en JSON et utilisé pour créer une **matrix**.

- **Utilisation de la stratégie matrix :**

La matrix permet de lancer en parallèle des jobs sur chaque dossier modifié, ce qui est très utile lorsqu'un pull request touche plusieurs modules. Ainsi, chaque module bénéficie d'un traitement isolé pour tofu plan ou tofu apply.

- **Configuration AWS sécurisée :**

Les workflows utilisent les actions aws-actions/configure-aws-credentials@v3 pour configurer l'authentification AWS via des rôles IAM, dont les ARNs sont stockés dans les secrets GitHub (PLAN_ROLE_ARN et APPLY_ROLE_ARN).

- **Publication des résultats :**

Les sorties des commandes tofu plan et tofu apply sont récupérées et publiées en commentaire sur le pull request grâce à l'action [peter-evans/create-or-update-comment](#). Cela permet aux reviewers de voir directement le résultat de la commande dans l'interface GitHub.

- **Installation d'OpenTofu :**

L'action opentofu/setup-opentofu@v1 installe la version d'OpenTofu nécessaire à l'exécution des commandes dans chaque dossier.

Conclusion

En appliquant ces changements, notre pipeline sera désormais capable de :

1. **Détecter automatiquement** tous les dossiers modifiés dans le répertoire OpenTofu (plutôt qu'un seul module statique).
2. **Utiliser la stratégie matrix** pour exécuter en parallèle les commandes tofu plan et tofu apply sur chaque module modifié, permettant ainsi de gérer des pull requests impactant plusieurs modules simultanément.
3. **Gérer l'authentification AWS de manière sécurisée** en s'appuyant sur des rôles IAM configurés via OpenID Connect (OIDC).