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

Author : Rémi Locquette

L'objectif de ce lab va être de se familiariser avec le CI/CD c'est à dire l'intégration constante de modifications sans impacter le travail se trouvant sur le git via la fenêtre git actions

Nous allons aussi utiliser des technologies de AWS et plus exactement l'OIDC, et enfin nous allons via OpenTofu déployer notre application automatiquement.

Part 1: Continuous Integration (CI)

Etape 1 : Intégration du Git

Pour commencer, il nous faut créer un git, qui est un cloud en ligne qui permet d'insérer des projets collaboratifs.

Un Depository, et un projet git, sur lequel il y a des branches du projet. Ma branche principale est la branche *Master* . Quand on commence un projet, on utilise les commandes :

```
git checkout master  #se place sur la collone master
git pull origin master  #importe le projet en ligne du git, sur le travail interne du projet
```

Ceci nous permet de mettre à jour le projet, dans le cas ou un membre du projet aurait mis à jour le git

Etape 2 : Github Actions workflow

Nous allons maintenant créer un dossier .github à la racine du projet. Dans ce dossier, nous allons créer un autre sous dossier, le fameux workflows. Ce dossier va contenir tous nos dossiers *.yml* qui vont gérer la partie test lors du push d'un projet.

```
mkdir -p .github/workflows
cd .github/workflows
touch app-tests.yml  #Créer notre fichier de test
```

```yaml
# .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
```

Ici on test le dossier sample-app, dont notre dossier contenant notre app : app.js

```javascript
// app.js
const express = require('express');
const app = express();
app.get('/', (req, res) => {
    res.send('Hello, World!');
});

app.get('/name/:name', (req, res) => {
    res.send(`Hello, ${req.params.name}!`);
});

app.get('/add/:a/:b', (req, res) => {
    const a = parseFloat(req.params.a);
    const b = parseFloat(req.params.b);

    if (isNaN(a) || isNaN(b)) {
        return res.status(400).send('Invalid input. Both parameters must be numbers.');
    }

    const sum = a + b;
    res.send(`The sum of ${a} and ${b} is ${sum}`);
});

module.exports = app;
```

Enfin on pousse notre projet sur notre git avec nos modification

git add td5/scripts/sample-app .github/workflows/app-tests.yml #On ajoute aussi notre scripts à tester

```
git commit -m "Add sample-app and workflow"
git push origin master
```

Ensuite pour tester, on crée une nouvelle commande ou on va implémenter une erreur

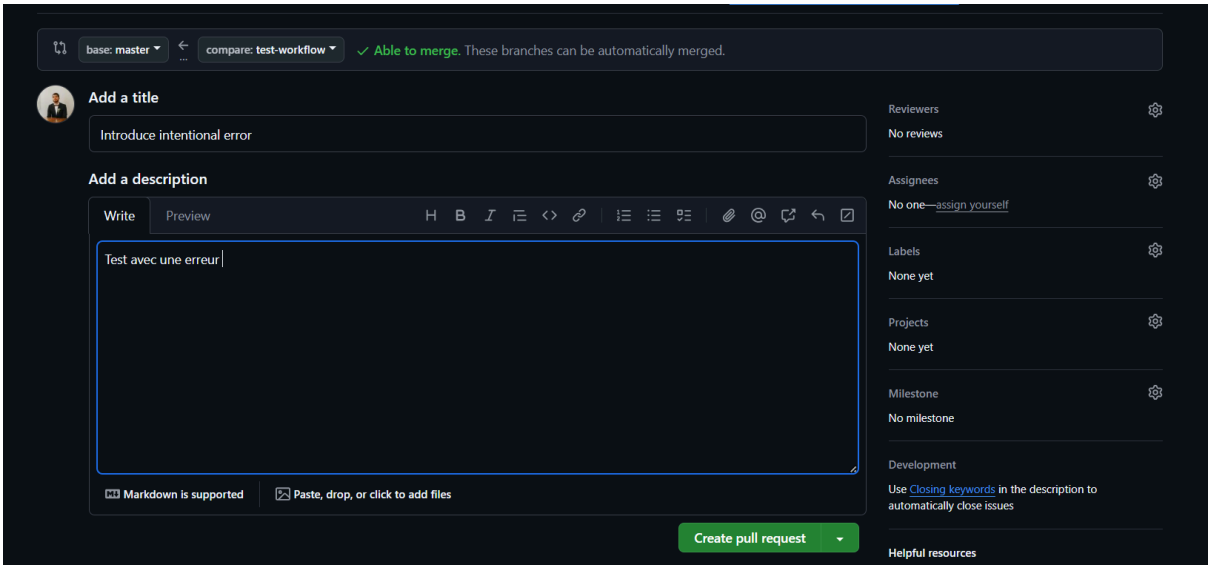
```
git checkout -b test-workflow #On crée une nouvelle branche test-workflow et on se place dessus
```

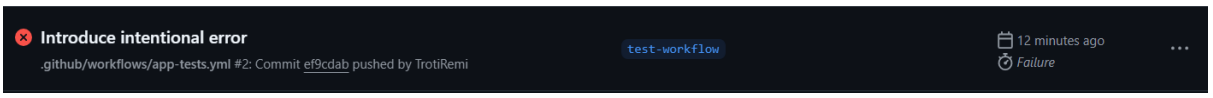
On ajoute notre erreur en changeant par exemple Hello, World ! en DevOps Labs!

Et on repush, mais sur la nouvelle branche

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

On va ensuite sur notre Git et on merge la branche test-workflow avec notre branche principale, master



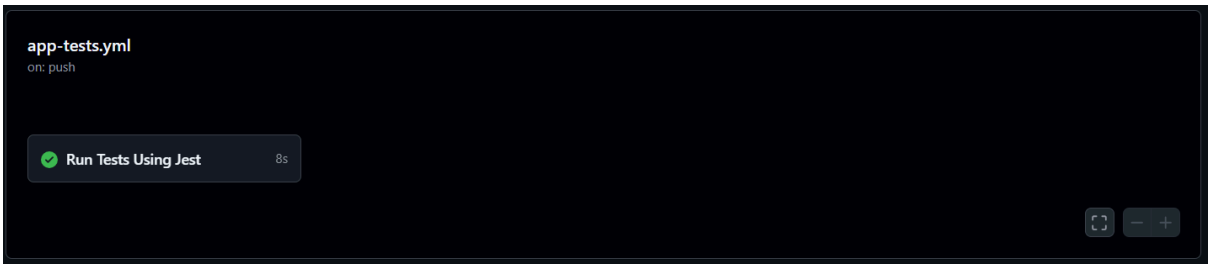


On voit ici en effet une erreur, comme souhaité. Maintenant on va corriger l'erreur. On ne va pas remettre la valeur de app.js, mais on va plutôt changer notre *app.test.js :*

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

On repull notre projet, puis on le re merge, et on vérifie si ca a bien marché

Ici on voit que le projet tourne mais n'a pas encore compilé



Et ici on voit notre résultat final qui a marché

Etape 3 : Open Tofu

On passe à nouveau à notre branche master et on pull nos données

git checkout master
git pull origin master

Puis on va tester avec open tofu en crée une nouvelle branche test

git checkout -b opentofu-tests

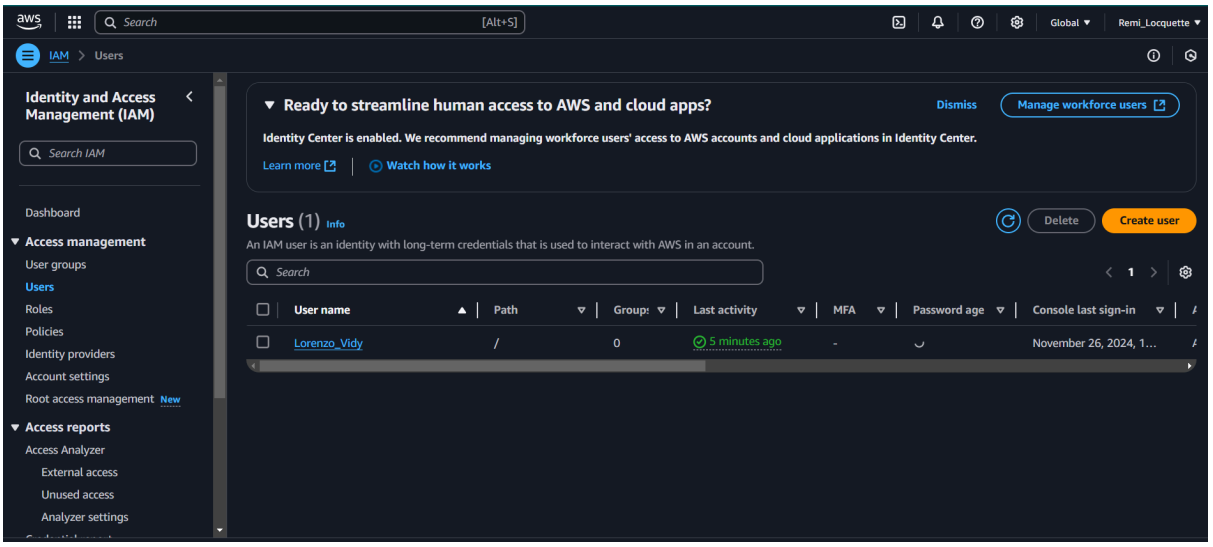
On crée dedans un dossier *ci-cd-permissions* puis dedans un dossier tofu : *main.tf*

```
provider "aws" {
  region = "us-east-2"
}

module "oidc_provider" {
  source       = "git::https://github.com/TrotiRemi/devops-lab.git//td5/scripts/tofu/modules/github-aws-oidc?ref=master"
  provider_url = "https://token.actions.githubusercontent.com"
}
```

Ici ma source est branché sur mon git justement sur mon OIDC. On ajoute ensuite un *iam_roles*

```
module "iam_roles" {
  source                      = "../../modules/gh-actions-iam-roles"
  name                        = "lambda-sample"
  oidc_provider_arn           = module.oidc_provider.oidc_provider_arn
  enable_iam_role_for_testing = true
  enable_iam_role_for_plan    = true
  enable_iam_role_for_apply   = true
  github_repo                 = "TrotiRemi/devops-lab"
  lambda_base_name            = "lambda-sample"
  tofu_state_bucket           = "mon-bucket-tofu"
  tofu_state_dynamodb_table   = "mon-tofu-state-lock"
}
```

Avant de faire cela, il faut créer un utilisateur AWS. Je vais donc sur mon compte et rajoute un utilisateurs avec des droits basics



Quand cela est fait, on ajoute une output.tf pour définir la sortie :

```
output "lambda_test_role_arn" {
  description = "The ARN of the IAM role for testing"
  value       = module.iam_roles.lambda_test_role_arn
}
```

On va aussi avoir besoin de notre arn, comme vue sur l'output, pour cela on lance cette commande :

```
$ aws iam list-open-id-connect-providers --region us-east-2
{
    "OpenIDConnectProviderList": [
        {
            "Arn": "arn:aws:iam::122610477635:oidc-provider/token.actions.githubusercontent.com"
        }
    ]
}
```

On push à nouveau nos modifications dans cette branche :
git add .
git commit -m "bucket et table"
git push origin opentofu-tests

Et enfin on lance notre appli dans le dossier ci-cd-permissions avec

tofu init #Initialise notre tofu
tofu apply #lance l'appli

```
Initializing the backend...
Upgrading modules...
- iam_roles in ..\..\modules\gh-actions-iam-roles
Downloading git::https://github.com/TrotiRemi/devops-lab.git?ref=master for oidc_provider...
- oidc_provider in .terraform\modules\oidc_provider\td5\scripts\tofu\modules\github-aws-oidc

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Finding latest version of hashicorp/tls...
- Installing hashicorp/aws v5.84.0...
- Installed hashicorp/aws v5.84.0 (signed, key ID 0C0AF313E5FD9F80)
- Installing hashicorp/tls v4.0.6...
- Installed hashicorp/tls v4.0.6 (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 created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that OpenTofu can guarantee to make the same selections by default when
you run "tofu init" in the future.

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.
module.oidc_provider.data.tls_certificate.github: Reading...
```

Initialisation de tofu

Quand on lance le tofu apply, on peut vérifier nos output :

Outputs:

lambda_deploy_apply_role_arn = "arn:aws:iam::122610477635:role/lambda-sample-apply"
lambda_deploy_plan_role_arn = "arn:aws:iam::122610477635:role/lambda-sample-plan"
lambda_test_role_arn = "arn:aws:iam::122610477635:role/lambda-sample-tests"

Etape 4 : Automatiser les tests

On veut maintenant automatiser nos tests. Pour cela on copie les deux fichier lambda-sample et test-endpoint dans notre dossier tofu. Pour rappel, un dossier tofu aura à l'intérieur un main.tf qui sera la base du code. un variables.tf, qui sont les variables qu'on va utiliser. Et enfin un output.tf, les sorties de notre application.

Ici on va crée notre variables.tf avec comme variables : name composé d'une description, et de type String

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

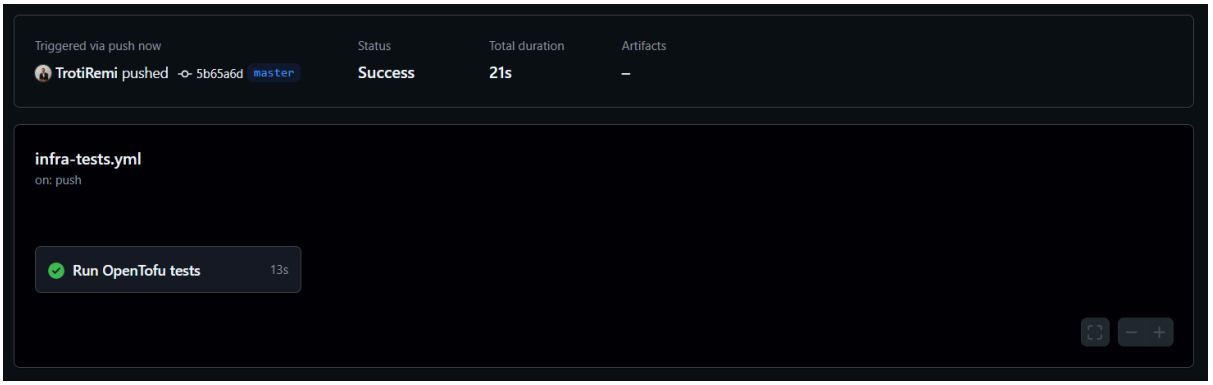
On modifie ensuite notre main.tf dans le dossier live pour ajouter nos deux variables. Enfin, quand nos deux variables sont ajoutées, on peut maintenant créer un nouveau test dans .github/workflows à la source de notre projet. La seule chose à mettre ici sera de remplacer role-to-assume: avec la valeur de notre arn de notre aws.

On peut retrouver cette info avec la commande ci dessous :

```
$ aws iam get-user
{
    "User": {
        "Path": "/",
        "UserName": "Lorenzo_Vidy",
        "UserId": "AIDARZDBHNJBZPADQL6VH",
        "Arn": "arn:aws:iam::122610477635:user/Lorenzo_Vidy",
        "CreateDate": "2024-11-26T13:37:25+00:00",
        "PasswordLastUsed": "2024-11-26T13:39:09+00:00",
        "Tags": [
            {
                "Key": "AKIARZDBHNJBSF665GQE",
                "Value": "test_1"
            },
            {
                "Key": "AKIARZDBHNJB3WFKFGER",
                "Value": "My first key"
            }
        ]
    }
}
```

```yaml
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:
          role-to-assume: arn:aws:iam::122610477635:role/lambda-sample-tests #Mon arn
          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: ch5/tofu/live/lambda-sample
        run: |
          tofu init -backend=false -input=false
          tofu test -verbose
```

Quand ceci est fait, on re-commit les changements avant de faire notre pull request sur notre branche main, pour vérifier que le test a bien marché.



Sur la fenêtre action le test est en vert, on peut donc passé à la partie suivantes

Part 2: Continuous Delivery (CD)

Etape 1 : Implémentation du Bucket et d'une table S3

Notre objectif cette fois sera de s'occuper de la partie déploiement. Nous voulons automatiser le processus (Continuous Delivery). Pour cela nous avons plusieurs stratégies pour plusieurs types de déploiements.

Nous utiliserons aussi des pipelines qui eux s'occuperont de l'automatisation des déploiements via les commits.

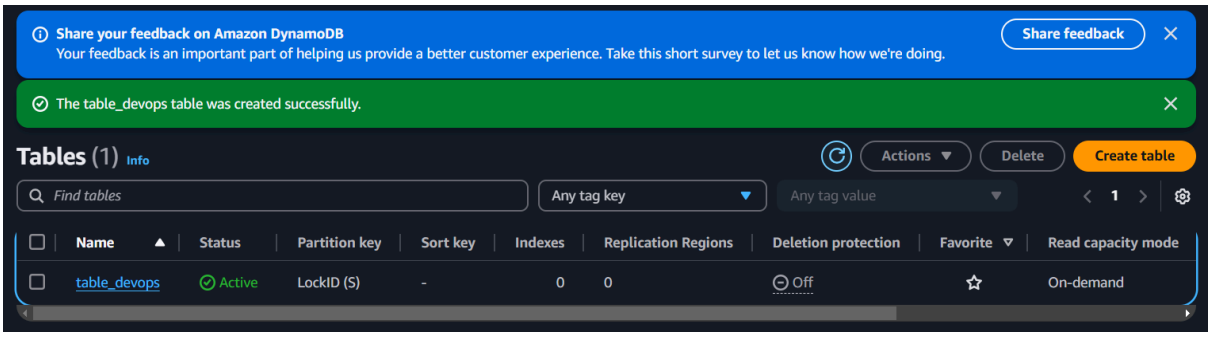
Nous allons commencer par utiliser les pipelines avec GitHub Actions. Pour cela nous avons besoin de notre bucket et notre S3.

```
git checkout master  #se place sur la collone master
git pull origin master  #importe le projet en ligne du git, sur le travail interne du projet
```

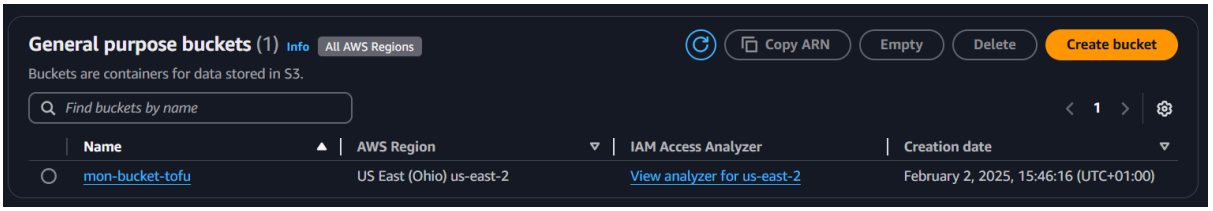
On veut crée maintenant deux éléments :

- tofu_state_bucket
- tofu_state_dynamodb

Pour avoir ces deux valeurs, il nous faut aller sur AWS et créer justement notre bucket et notre S3.



On peut créer un dossier dans module state-bucket avec nos infos.



Enfin on ajoute des droits à notre utilisateurs pour bucket et S3 pour pouvoir les utiliser

Dans ce dossier tofu-state on va créer un fichier tofu *backend.tf.* On va mettre dans ce même projet, justement notre bucket et notre table

```
terraform {
  backend "s3" {
    bucket         = "mon-bucket-tofu" #le bucket qu'on a eu juste avant
    key            = "ch5/tofu/live/tofu-state"
    region         = "us-east-2"
    encrypt        = true
    dynamodb_table = "table_devops" #la table qu'on a eu juste avant
  }
}
```

On veut enfin s'assurer que notre bucket est importé de aws à notre projet tofu, on utilise donc cette commande:

```
Rémi@DESKTOP-NIKT10P MINGW64 /c/tmp/git-practice/devops-lab/td5/scripts/tofu/live/tofu-state (master)
$ tofu import module.state.aws_s3_bucket.tofu_state mon-bucket-tofu
Acquiring state lock. This may take a few moments...
module.state.aws_s3_bucket.tofu_state: Importing from ID "mon-bucket-tofu"...
module.state.aws_s3_bucket.tofu_state: Import prepared!
  Prepared aws_s3_bucket for import
module.state.aws_s3_bucket.tofu_state: Refreshing state... [id=mon-bucket-tofu]

Import successful!

The resources that were imported are shown above. These resources are now in
your OpenTofu state and will henceforth be managed by OpenTofu.

Releasing state lock. This may take a few moments...
```

et on fait la même chose pour notre table :

```
Rémi@DESKTOP-NIKT10P MINGW64 /c/tmp/git-practice/devops-lab/td5/scripts/tofu/live/tofu-state (master)
$ tofu import module.state.aws_dynamodb_table.tofu_locks table_devops
Acquiring state lock. This may take a few moments...
module.state.aws_dynamodb_table.tofu_locks: Importing from ID "table_devops"...
module.state.aws_dynamodb_table.tofu_locks: Import prepared!
  Prepared aws_dynamodb_table for import
module.state.aws_dynamodb_table.tofu_locks: Refreshing state... [id=table_devops]

Import successful!
```

On peut maintenant relancer les commandes :

tofu init #Initialise notre tofu
tofu apply #lance l'appli

Cela peut mettre un peu de temps (dans mon cas 9 minutes)

```
module.state.aws_dynamodb_table.tofu_locks: Still modifying... [id=table_devops, 8m30s elapsed]
module.state.aws_dynamodb_table.tofu_locks: Still modifying... [id=table_devops, 8m40s elapsed]
module.state.aws_dynamodb_table.tofu_locks: Still modifying... [id=table_devops, 8m50s elapsed]
module.state.aws_dynamodb_table.tofu_locks: Modifications complete after 8m57s [id=table_devops]

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
```

Maintenant on va faire la même chose mais dans le dossier lambda-sample:

```
terraform {
  backend "s3" {
    bucket         = "mon-bucket-tofu"
    key            = "ch5/tofu/live/lambda-sample"
    region         = "us-east-2"
    encrypt        = true
    dynamodb_table = "table_devops"
  }
}
```

On peut maintenant relancer les commandes :

tofu init #Initialise notre tofu
tofu apply #lance l'appli

```
Rémi@DESKTOP-NIKT10P MINGW64 /c/tmp/git-practice/devops-lab/td5/scripts/tofu/live/lambda-sample (master)
$ tofu apply
Acquiring state lock. This may take a few moments...

No changes. Your infrastructure matches the configuration.      Acquiring state lock. TAcquiring state lock. This may take a few Ac
ake a fewAcquiring state lock. This may take a few moments...

No changes. Your infrastructure matches the configuration.

OpenTofu has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

Enfin on change notre main.tf dans ci-cd-permission pour ajouter dans notre iam_roles nos deux éléments

```
module "iam_roles" {
  source                      = "../../modules/gh-actions-iam-roles"
  name                        = "lambda-sample"
  oidc_provider_arn           = module.oidc_provider.oidc_provider_arn
  enable_iam_role_for_testing = true
  enable_iam_role_for_plan    = true
  enable_iam_role_for_apply   = true
  github_repo                 = "TrotiRemi/devops-lab"
  lambda_base_name            = "lambda-sample"
  tofu_state_bucket           = "mon-bucket-tofu"
  tofu_state_dynamodb_table   = "mon-tofu-state-lock"
}
```

On ajoute enfin nos output qu'on veut afficher lors du apply

```
output "lambda_deploy_plan_role_arn" {
  description = "The ARN of the IAM role for deployment plan"
  value       = module.iam_roles.lambda_deploy_plan_role_arn
}

output "lambda_deploy_apply_role_arn" {
  description = "The ARN of the IAM role for deployment apply"
  value       = module.iam_roles.lambda_deploy_apply_role_arn
}
```

Etape 2 : Github Actions workflow

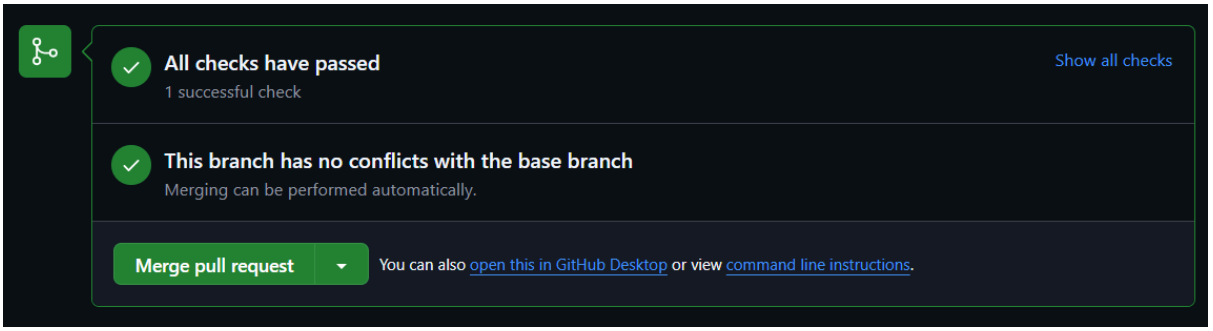
On va maintenant créer deux fichiers dans notre workflows, tofu_plan.yml et tofu_apply.yml. Les deux fichiers seront lancé quand il y aura un pull de la branche master après des modifications dans le fichier lambda-sample.

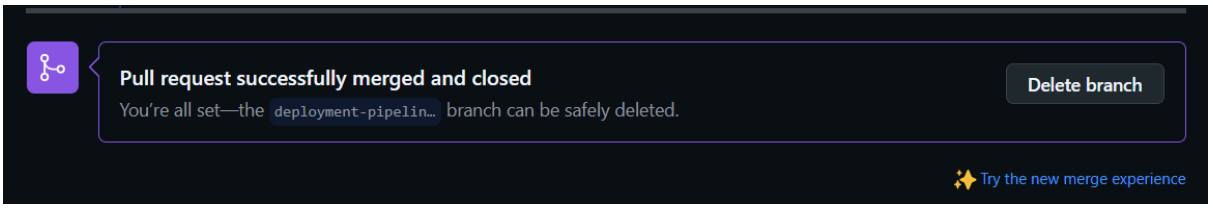
On push ces deux nouveaux fichiers dans notre branche master. Puis on change de branche, pour effectuer des modifications par exemple :

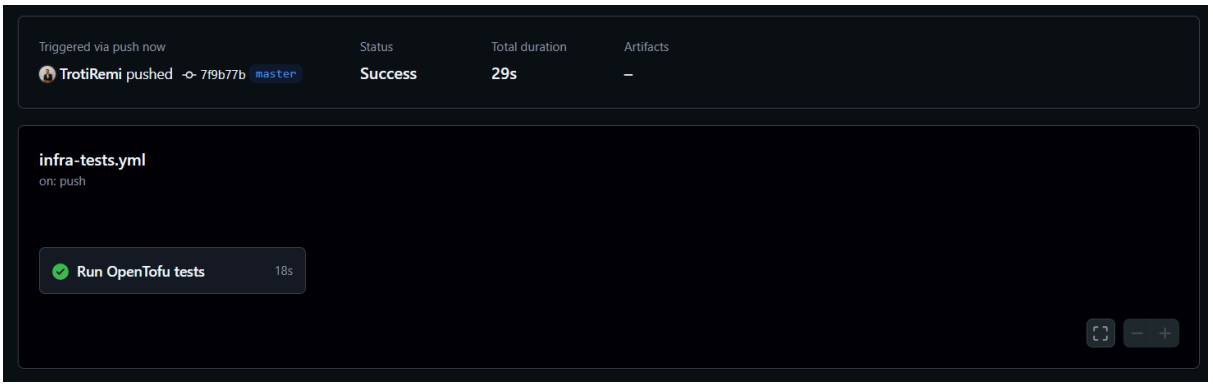
```
Rémi@DESKTOP-NIKT10P MINGW64 /c/tmp/git-practice/devops-lab (test-tofu-matrix)
$
```

```
exports.handler = (event, context, callback) => {
  callback(null, {statusCode: 200, body: "DevOps Labs!"});
};
```

On push enfin cette branche avant de la merge à nouveau avec notre branche master







Notre test a bien fonctionner

Etape 3 : Execice

On va maintenant faire deux choses, premièrement améliorer la pipeline pour detecter les changement automatiquement, en mettant le dossier source directement sur lambda-sample. On doit donc changer nos deux dossier apply et plan

Tofu Plan

```yaml
name: Tofu Plan
on:
  pull_request:
    branches: ["master"]
    paths:
      - "ch5/tofu/live/**"  # Detecte les changement dans tout les dossiers tofu
    types: [opened, synchronize, reopened]
```

```
jobs:
  detect-changes:
    name: "Detect Changes"
    runs-on: ubuntu-latest
    outputs:
      changed_folders: ${{ steps.filter.outputs.changed_folders }}
    steps:
      - uses: actions/checkout@v2

      - name: Detect changed OpenTofu folders
        id: filter
        uses: dorny/paths-filter@v2
        with:
          filters: |
            changed_folders:
              - "ch5/tofu/live/**"

  plan:
    name: "Tofu Plan"
    runs-on: ubuntu-latest
    needs: detect-changes
    if: needs.detect-changes.outputs.changed_folders == 'true'  #
tourne seulement si un changement est détecté
    strategy:
      matrix:
        folder: ${{
fromJson(needs.detect-changes.outputs.changed_folders) }}
    permissions:
      pull-requests: write
      id-token: write
      contents: read
    steps:
      - uses: actions/checkout@v2

      - uses: aws-actions/configure-aws-credentials@v3
        with:
          role-to-assume:
arn:aws:iam::122610477635:role/lambda-sample-plan
          role-session-name: plan-${{ github.run_number }}-${{
github.actor }}
          aws-region: us-east-2
```

```
      - uses: opentofu/setup-opentofu@v1

      - name: tofu plan
        id: plan
        working-directory: ${{ matrix.folder }}
        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' && '✅' ||
'⚠️' }}
        with:
          issue-number: ${{ github.event.pull_request.number }}
          body: |
            ## ${{ env.RESULT_EMOJI }} `tofu plan` output for `${{
matrix.folder }}`
            ```${{ steps.plan.outputs.stdout }}```
```

# Tofu Apply

```
name: Tofu Apply
on:
  push:
    branches: ["master"]
    paths:
      - "ch5/tofu/live/**" # Detecte les changement dans tout les
dossiers tofu

jobs:
  detect-changes:
    name: "Detect Changes"
    runs-on: ubuntu-latest
    outputs:
      changed_folders: ${{ steps.filter.outputs.changed_folders }}
    steps:
      - uses: actions/checkout@v2

      - name: Detect changed OpenTofu folders
```
```

```
        id: filter
        uses: dorny/paths-filter@v2
        with:
          filters: |
            changed_folders:
              - "ch5/tofu/live/**"

  apply:
    name: "Tofu Apply"
    runs-on: ubuntu-latest
    needs: detect-changes
    if: needs.detect-changes.outputs.changed_folders == 'true'
    strategy:
      matrix:
        folder: ${{
fromJson(needs.detect-changes.outputs.changed_folders) }}
    permissions:
      pull-requests: write
      id-token: write
      contents: read
    steps:
      - uses: actions/checkout@v2

      - uses: aws-actions/configure-aws-credentials@v3
        with:
          role-to-assume:
arn:aws:iam::122610477635:role/lambda-sample-apply
          role-session-name: apply-${{ github.run_number }}-${{
github.actor }}
          aws-region: us-east-2

      - uses: opentofu/setup-opentofu@v1

      - name: tofu apply
        id: apply
        working-directory: ${{ matrix.folder }}
        run: |
          tofu init -no-color -input=false
          tofu apply -no-color -input=false -lock-timeout=60m
-auto-approve

      - uses: jwalton/gh-find-current-pr@master
        id: find_pr
```

```
        with:
          state: all

      - uses: peter-evans/create-or-update-comment@v4
        if: steps.find_pr.outputs.number
        env:
          RESULT_EMOJI: ${{ steps.apply.outcome == 'success' && '✅' ||
'⚠️' }}
        with:
          issue-number: ${{ steps.find_pr.outputs.number }}
          body: |
            ## ${{ env.RESULT_EMOJI }} `tofu apply` output for `${{
matrix.folder }}`
            ```${{ steps.apply.outputs.stdout }}```
```

On push à nouveau sur notre branche master, puis on change de branche en effectuant une modification sur notre lambda_sample. On merge les deux branches, et on vérifie si le test a marché

## Conclusion :

On a réussi ici à créer un CI/CD qui va détecter automatiquement tout changement, pour vérifier si le projet est fonctionnel avec ces changements.

# FIN
```