Implementación Automatizada con Terraform Cloud, GitHub Actions y AWS

Autor: Acosta Hernan Andres
Fecha: 31/03/2025

Objetivo

Este proyecto demostró la integración de:
HCP Terraform (gestión centralizada de estado)
GitHub Actions (CI/CD automatizado)
AWS (proveedor cloud)
----------///----------

```
[Desarrollador]
   ↓ (Push código)

[Repositorio GitHub]
   ↓ (Trigger automático)

[GitHub Actions]
  ↓ (Ejecuta)

[Terraform Plan] → [Muestra cambios en Pull Request]
   ↓ (Tras aprobación)

[Merge a rama principal]
   ↓ (Trigger automático)

[GitHub Actions Apply]
   ↓ (Aplica cambios)

[HCP Terraform]
   ↓ (Gestiona infra)

[Recursos en AWS]
```

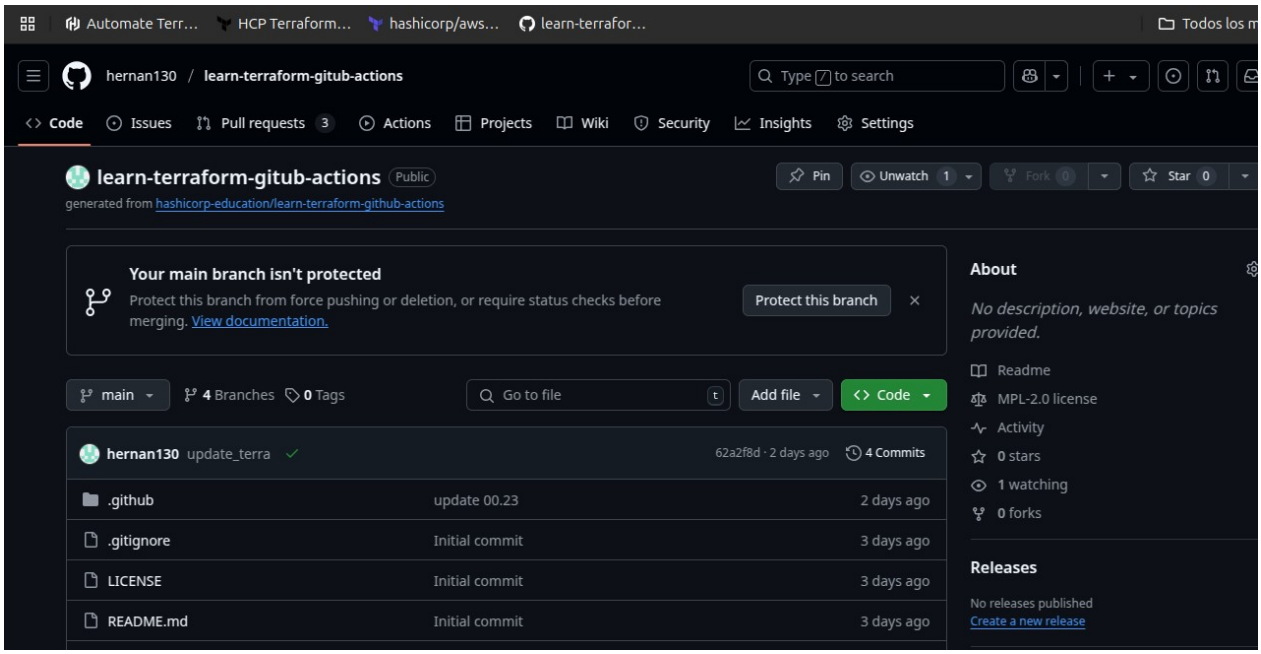
----------//////////----------

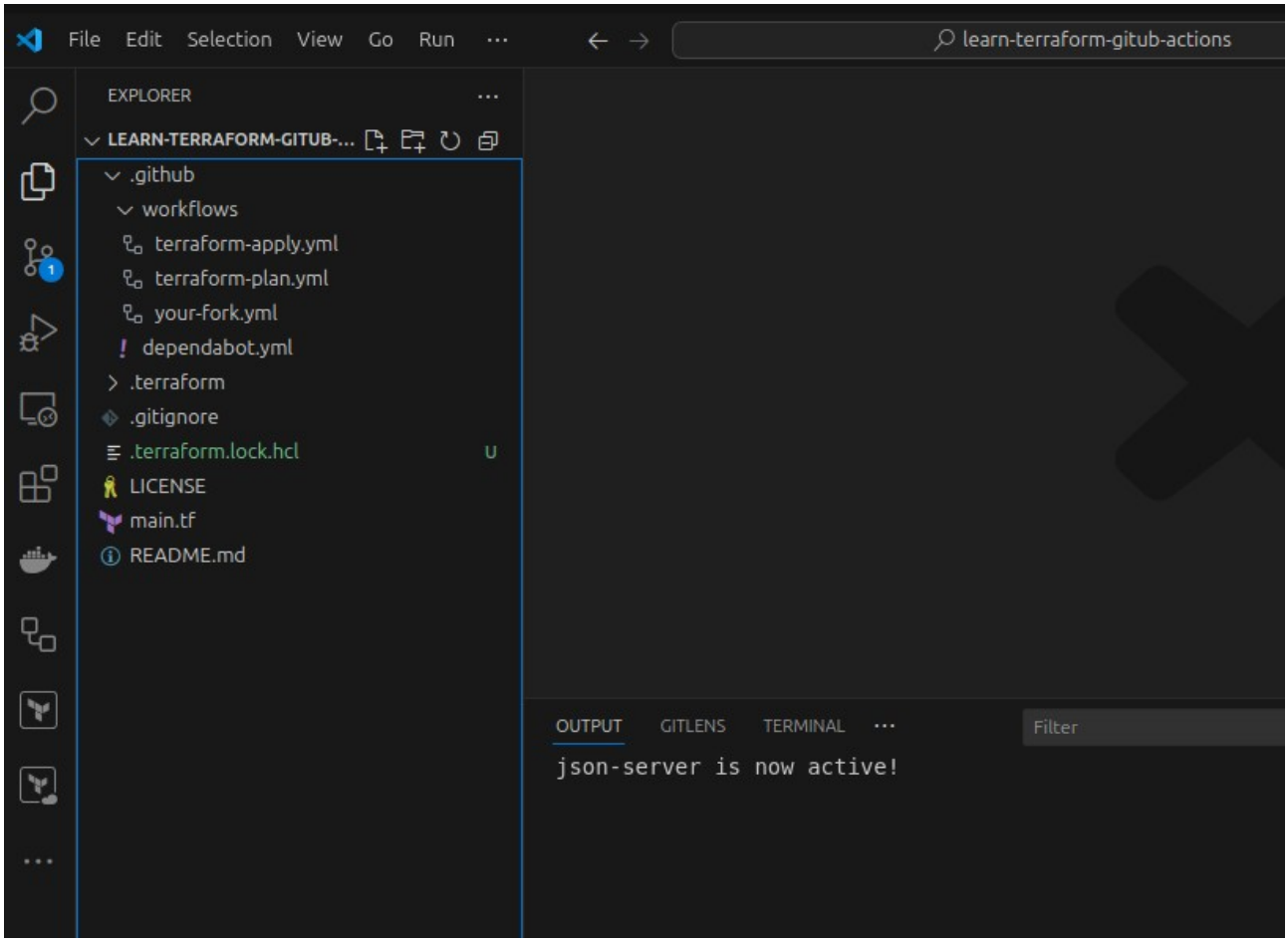
Flujo de trabajo implementado:

1. El desarrollador realiza un Push al repositorio de GitHub.
2. GitHub Actions detecta los cambios y ejecuta un Terraform Plan.
3. Se revisa el resultado en el Pull Request.
4. Si se aprueba, se realiza el merge a la rama main .
5. GitHub Actions ejecuta Terraform Apply y despliega los cambios en AWS.

como primera medida se utilizo la plantilla del repositorio propuesto((Post Hashicorp) - https://developer.hashicorp.com/terraform/tutorials/automation/github-actions)

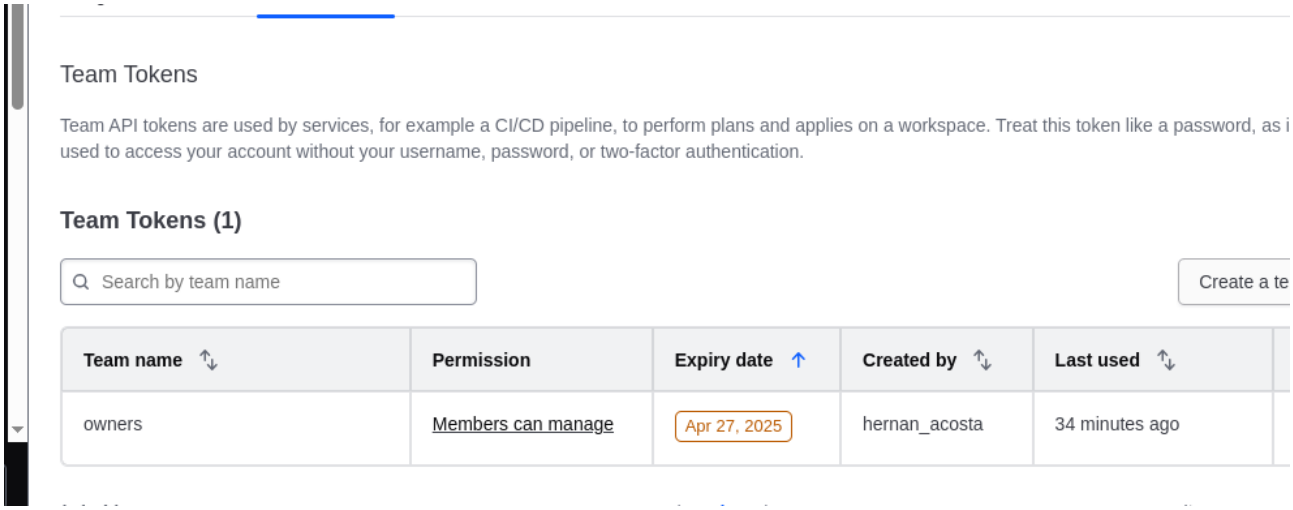
para crear un repositorio personal y luego clonarlo a nivel localmente





Configuración Realizada

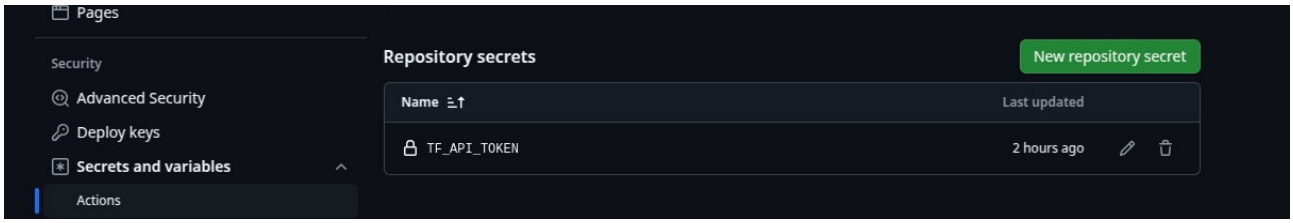
3.1 HCP Terraform



Creación de workspace y token de API:

Variables de AWS configuradas como sensibles:

Secrets del repositorio con TF_API_TOKEN:

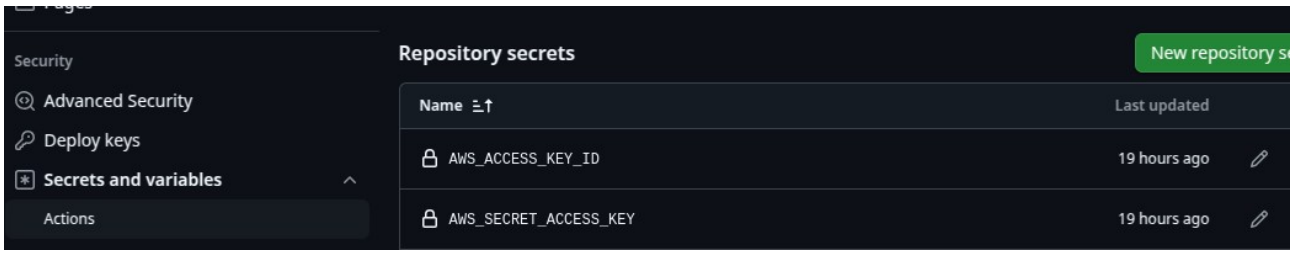


Configuración de Secrets en GitHub

Se almacenaron las credenciales de AWS en Settings > Secrets > Actions:

•AWS_ACCESS_KEY_ID

•AWS_SECRET_ACCESS_KEY



- Workflows en .GitHub/Workflows/:
  terraform-plan.yml
- name: "Terraform Plan"

on:
Pull_Request:

env:
TF_CLOUD_ORGANIZATION: "mi-org-tf"
TF_API_TOKEN: "${{ secrets.TF_API_TOKEN }}"
TF_WORKSPACE: "learn-terraform-github-actions"
CONFIG_DIRECTORY: "./"

jobs:
terraform:
if: github.repository != 'hashicorp-education/learn-terraform-github-actions'
name: "Terraform Plan"
runs-on: ubuntu-latest
permissions:
so GitHub can check out this repo using the default github.token
contents: read
pull-requests: write
steps:
- name: Checkout
uses: actions/checkout@v3

- name: Upload Configuration
uses: hashicorp/tfc-workflows-github/actions/upload-configuration@v1.0.0
id: plan-upload
with:
workspace: ${{ env.TF_WORKSPACE }}
directory: ${{ env.CONFIG_DIRECTORY }}
speculative: true

- name: Create Plan Run
uses: hashicorp/tfc-workflows-github/actions/create-run@v1.0.0
id: plan-run
with:
workspace: ${{ env.TF_WORKSPACE }}
configuration_version: ${{ steps.plan-upload.outputs.configuration_version_id }}
plan_only: true

```
- name: Get Plan Output
uses: hashicorp/tfc-workflows-github/actions/plan-output@v1.0.0
id: plan-output
with:
plan: ${{ fromJSON(steps.plan-run.outputs.payload).data.relationships.plan.data.id }}

- name: Update PR
uses: actions/github-script@v6
id: plan-comment
with:
github-token: ${{ secrets.GITHUB_TOKEN }}
script: |
// 1. Retrieve existing bot comments for the PR
const { data: comments } = await github.rest.issues.listComments({
owner: context.repo.owner,
repo: context.repo.repo,
issue_number: context.issue.number,
});
const botComment = comments.find(comment => {
return comment.user.type === 'Bot' && comment.body.includes('Terraform Cloud Plan Output')
});
const output = `#### Terraform Cloud Plan Output
\`\`\`
Plan: ${{ steps.plan-output.outputs.add }} to add, ${{ steps.plan-output.outputs.change }} to
change, ${{ steps.plan-output.outputs.destroy }} to destroy.
\`\`\
[Terraform Cloud Plan](${{ steps.plan-run.outputs.run_link }})
`;
// 3. Delete previous comment so PR timeline makes sense
if (botComment) {
github.rest.issues.deleteComment({
owner: context.repo.owner,
repo: context.repo.repo,
comment_id: botComment.id,
});
}
github.rest.issues.createComment({
issue_number: context.issue.number,
owner: context.repo.owner,
repo: context.repo.repo,
body: output
});
```

- terraform-apply.yml)

```
name: "Terraform Apply"

on:
push:
branches:
```

```
- main

env:
TF_CLOUD_ORGANIZATION: "mi-org-tf"
TF_API_TOKEN: "${{ secrets.TF_API_TOKEN }}"
TF_WORKSPACE: "learn-terraform-github-actions"
CONFIG_DIRECTORY: "./"


jobs:
terraform:
if: github.repository != 'hashicorp-education/learn-terraform-github-actions'
name: "Terraform Apply"
runs-on: ubuntu-latest
permissions: # granular permissions
# so GitHub can check out this repo using the default github.token
contents: read
steps:
- name: Checkout
uses: actions/checkout@v3

- name: Upload Configuration
uses: hashicorp/tfc-workflows-github/actions/upload-configuration@v1.0.0
id: apply-upload
with:
workspace: ${{ env.TF_WORKSPACE }}
directory: ${{ env.CONFIG_DIRECTORY }}

- name: Create Apply Run
uses: hashicorp/tfc-workflows-github/actions/create-run@v1.0.0
id: apply-run
with:
workspace: ${{ env.TF_WORKSPACE }}
configuration_version: ${{ steps.apply-upload.outputs.configuration_version_id }}

- name: Apply
uses: hashicorp/tfc-workflows-github/actions/apply-run@v1.0.0
if: fromJSON(steps.apply-run.outputs.payload).data.attributes.actions.IsConfirmable
id: apply
with:
run: ${{ steps.apply-run.outputs.run_id }}
comment: "Apply Run from GitHub Actions CI ${{ github.sha }}"
```

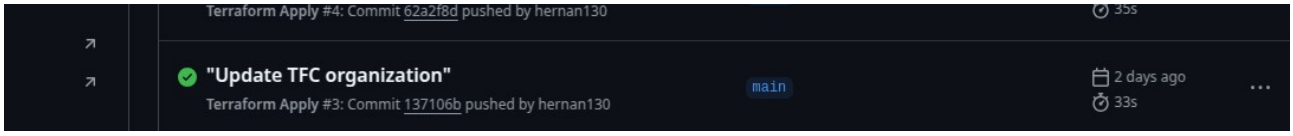
---

4. Flujo de Trabajo

4.1 Terraform Plan (PR)

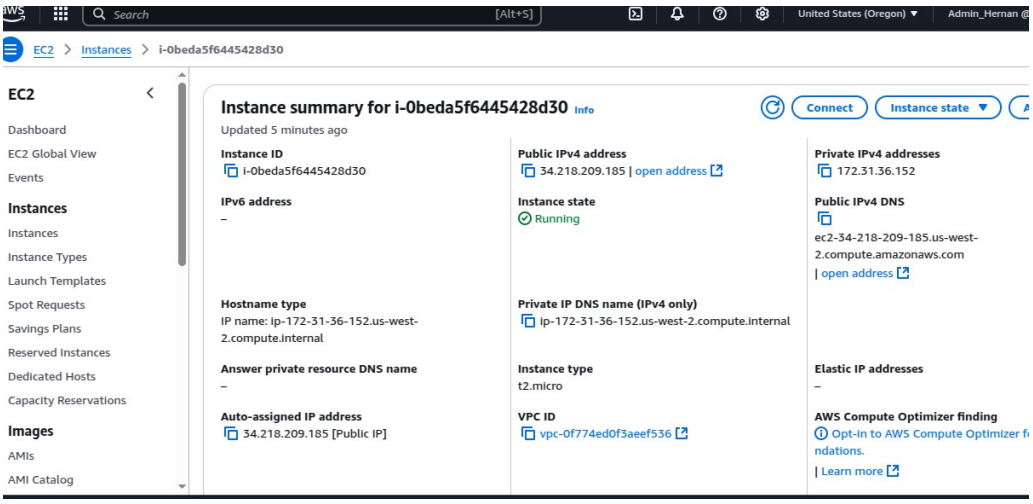
1. Se creó un Pull Request con cambios :

GitHub Actions



Recursos creados en AWS:

<u>Instancia EC2 t2.micro con acceso HTTP:</u>



Conclusión

La implementación fue exitosa, logrando:

- Automatización completa del ciclo de vida de IaC
- Integración entre Terraform Cloud y GitHub
- Creación verificable de recursos en AWS

Lo que mejoraría en esta implementación es la estructura

Estructura original

```
learn-terraform-github-actions/
├── .github/
│   ├── workflows/            # Directorio de flujos de GitHub Actions
│   │   ├── terraform-plan.yml   # Workflow para ejecutar terraform plan
│   │   └── terraform-apply.yml  # Workflow para ejecutar terraform apply
│   └── dependabot.yml         # Configuración para actualizaciones automáticas de
dependencias
├── .gitignore                # Archivos excluidos del control de versiones
├── LICENSE                   # Licencia del proyecto (probablemente MIT)
├── README.md                 # Documentación principal del proyecto
└── main.tf                   # Archivo principal de configuración de Terraform
```

Estructura con modificaciones

```
learn-terraform-github-actions/
├── .github/
│   ├── workflows/
│   │   ├── terraform-plan.yml
│   │   └── terraform-apply.yml
│   └── dependabot.yml
├── modules/                  # Módulos reutilizables (opcional)
│   └── example-module/
│       ├── main.tf
│       ├── variables.tf
│       └── outputs.tf
├── environments/             # Configuración por entorno
│   ├── dev/
│   │   ├── main.tf           # Config específica de dev
│   │   ├── terraform.tfvars    # Variables para dev
│   │   └── backend.tf        # Backend config para dev
│   └── prod/
│       ├── main.tf
│       ├── terraform.tfvars
│       └── backend.tf
```

```
├── main.tf                  # Configuración principal
├── providers.tf             # Configuración de proveedores
├── variables.tf             # Variables globales
├── outputs.tf               # Outputs globales
├── terraform.tfvars.example    # Ejemplo de variables (sin valores sensibles)
├── .gitignore
├── LICENSE
└── README.md
```

Mejoras clave:

1. **Separación clara de componentes:**
   - providers.tf para la configuración de proveedores
   - variables.tf para declaración de variables
   - outputs.tf para outputs
2. **Estructura por entornos (dev/prod) con:**
   - Configuraciones específicas por entorno
   - Variables separadas
   - Backend configuration independiente
3. **Módulos opcionales para componentes reutilizables**

Ejemplo de terraform.tfvars sin valores reales para documentación

4. **README.md debería explicar:**
   - Cómo usar los workflows de GitHub Actions
   - Estructura del proyecto
   - Requisitos previos
   - Ejemplos de uso