A diagram of a software system

AI-generated content may be incorrect.

A diagram of a process

AI-generated content may be incorrect.

A computer screen with text on it

AI-generated content may be incorrect.

Here’s a **production-grade, automated, multi-stage GitHub Actions pipeline** integrating:

**✅ Your Required Steps – Fully Automated**

**✅ Step 1: Developer Pushes to dev**

Triggers the following checks:

* ✅ Terraform fmt, validate
* ✅ Security scan with tfsec
* ✅ Container/IaC scan with Trivy
* ✅ Terraform Cloud (TFC) Plan in dev workspace
* ✅ Auto-create PR from dev → main

**✅ Step 2: Optional Auto-Merge PR**

* Manual or automated approval via GitHub Environments or merge-method

**✅ Step 3: On main merge:**

* Terraform Cloud auto-triggers prod workspace deployment

**📁 Final Structure**

.github/

├── workflows/

│ └── terraform-multistage.yml

infrastructure/

├── main.tf

├── variables.tf

├── outputs.tf

**📜 GitHub Actions: .github/workflows/terraform-multistage.yml**

name: Terraform Multi-Stage Pipeline with Security & Auto PR

on:

push:

branches: [ "dev", "main" ]

env:

TF\_API\_TOKEN: ${{ secrets.TF\_API\_TOKEN }}

TF\_CLOUD\_ORGANIZATION: your-tfc-org

TF\_WORKING\_DIR: ./infrastructure

ARM\_CLIENT\_ID: ${{ secrets.ARM\_CLIENT\_ID }}

ARM\_CLIENT\_SECRET: ${{ secrets.ARM\_CLIENT\_SECRET }}

ARM\_SUBSCRIPTION\_ID: ${{ secrets.ARM\_SUBSCRIPTION\_ID }}

ARM\_TENANT\_ID: ${{ secrets.ARM\_TENANT\_ID }}

jobs:

# ----------------------------

# Stage 1 - Linting & Security Checks (dev)

# ----------------------------

lint-security:

if: github.ref == 'refs/heads/dev'

runs-on: ubuntu-latest

name: Run Format, Validate, TFSec, Trivy

steps:

- name: Checkout code

uses: actions/checkout@v4

- name: Setup Terraform

uses: hashicorp/setup-terraform@v3

with:

cli\_config\_credentials\_token: ${{ secrets.TF\_API\_TOKEN }}

- name: Terraform Format

run: terraform fmt -check -recursive

working-directory: ${{ env.TF\_WORKING\_DIR }}

- name: Terraform Validate

run: terraform validate

working-directory: ${{ env.TF\_WORKING\_DIR }}

- name: Run tfsec

uses: aquasecurity/tfsec-action@v1.0.0

with:

working\_directory: ${{ env.TF\_WORKING\_DIR }}

- name: Install Trivy

run: |

sudo apt-get install wget -y

wget https://github.com/aquasecurity/trivy/releases/latest/download/trivy\_0.47.0\_Linux-64bit.deb

sudo dpkg -i trivy\_0.47.0\_Linux-64bit.deb

- name: Trivy Filesystem Scan

run: trivy fs ${{ env.TF\_WORKING\_DIR }} --exit-code 0 --severity HIGH,CRITICAL

# ----------------------------

# Stage 2 - Terraform Plan (TFC Dev)

# ----------------------------

tfc-plan-dev:

if: github.ref == 'refs/heads/dev'

needs: lint-security

runs-on: ubuntu-latest

name: Terraform Cloud Plan - dev

steps:

- name: Checkout code

uses: actions/checkout@v4

- name: Trigger Terraform Cloud Plan

run: |

curl -s \

--request POST \

--header "Authorization: Bearer $TF\_API\_TOKEN" \

--header "Content-Type: application/vnd.api+json" \

--data '{

"data": {

"attributes": {

"message": "Triggered from GitHub Action - dev",

"plan-only": true

},

"type": "runs",

"relationships": {

"workspace": {

"data": {

"type": "workspaces",

"name": "dev"

}

}

}

}

}' https://app.terraform.io/api/v2/runs

# ----------------------------

# Stage 3 - Auto Create PR

# ----------------------------

create-pr:

if: github.ref == 'refs/heads/dev'

needs: tfc-plan-dev

runs-on: ubuntu-latest

name: Create PR from dev → main

steps:

- name: Checkout repo

uses: actions/checkout@v4

- name: Create PR

uses: peter-evans/create-pull-request@v6

with:

token: ${{ secrets.GITHUB\_TOKEN }}

branch: dev

base: main

title: "[Auto] Promote IaC changes from dev to main"

body: "This PR was automatically created after passing checks and TFC plan."

# ----------------------------

# Stage 4 - TFC Deploys Prod Automatically on Main Merge

# ----------------------------

# Handled automatically by Terraform Cloud’s VCS integration

**✅ Additional Notes:**

* tfsec and Trivy ensure security compliance
* Optional **auto-merge PR** can be added using:
* - name: Auto merge PR
* uses: pascalgn/automerge-action@v0.14.3
* TFC triggers apply for prod workspace on main update if configured

**📦 Want me to zip this structure again with this config + updated flowchart PDF?**

Let me know — I can generate the entire downloadable template for you now.

Here’s a **multi-stage GitHub Actions workflow** setup for your dev → main promotion using **Terraform Cloud**. Each stage performs:

**🔁 Full Multi-Stage Pipeline Flow**

# .github/workflows/multi-stage-iac.yml

name: Multi-Stage IaC Pipeline with Terraform Cloud

on:

push:

branches: [ "dev", "main" ]

env:

TF\_CLOUD\_ORGANIZATION: your-tfc-org

TF\_API\_TOKEN: ${{ secrets.TF\_API\_TOKEN }}

TF\_WORKING\_DIR: ./infrastructure

ARM\_CLIENT\_ID: ${{ secrets.ARM\_CLIENT\_ID }}

ARM\_CLIENT\_SECRET: ${{ secrets.ARM\_CLIENT\_SECRET }}

ARM\_SUBSCRIPTION\_ID: ${{ secrets.ARM\_SUBSCRIPTION\_ID }}

ARM\_TENANT\_ID: ${{ secrets.ARM\_TENANT\_ID }}

jobs:

# ----------------------

# Stage 1: Dev Validation & Plan

# ----------------------

validate-dev:

if: github.ref == 'refs/heads/dev'

name: Validate & Plan - Dev

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v4

- name: Setup Terraform

uses: hashicorp/setup-terraform@v3

with:

cli\_config\_credentials\_token: ${{ secrets.TF\_API\_TOKEN }}

- name: Terraform Init

working-directory: ${{ env.TF\_WORKING\_DIR }}

run: terraform init

- name: Terraform Validate

working-directory: ${{ env.TF\_WORKING\_DIR }}

run: terraform validate

- name: Terraform FMT Check

working-directory: ${{ env.TF\_WORKING\_DIR }}

run: terraform fmt -check

- name: Terraform Plan (Dev)

working-directory: ${{ env.TF\_WORKING\_DIR }}

env:

TF\_WORKSPACE: dev

run: terraform plan -var="environment=dev"

- name: Create PR to Main

uses: peter-evans/create-pull-request@v6

with:

token: ${{ secrets.GITHUB\_TOKEN }}

base: main

branch: dev

title: "[Auto] Promote IaC changes to main"

body: "Auto-created PR from dev branch after plan success."

# ----------------------

# Stage 2: Main Branch Plan (PR Check)

# ----------------------

validate-main:

if: github.ref == 'refs/heads/main'

name: Validate & Plan - Main

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v4

- name: Setup Terraform

uses: hashicorp/setup-terraform@v3

with:

cli\_config\_credentials\_token: ${{ secrets.TF\_API\_TOKEN }}

- name: Terraform Init

working-directory: ${{ env.TF\_WORKING\_DIR }}

run: terraform init

- name: Terraform Validate

working-directory: ${{ env.TF\_WORKING\_DIR }}

run: terraform validate

- name: Terraform Plan (Prod)

working-directory: ${{ env.TF\_WORKING\_DIR }}

env:

TF\_WORKSPACE: prod

run: terraform plan -var="environment=prod"

# ----------------------

# Stage 3: Apply on Merge to Main (Prod)

# ----------------------

apply-prod:

if: github.ref == 'refs/heads/main'

name: Deploy - Production

needs: validate-main

runs-on: ubuntu-latest

environment:

name: production

url: https://portal.azure.com

steps:

- name: Checkout code

uses: actions/checkout@v4

- name: Setup Terraform

uses: hashicorp/setup-terraform@v3

with:

cli\_config\_credentials\_token: ${{ secrets.TF\_API\_TOKEN }}

- name: Terraform Init

working-directory: ${{ env.TF\_WORKING\_DIR }}

run: terraform init

- name: Terraform Apply (Prod)

working-directory: ${{ env.TF\_WORKING\_DIR }}

env:

TF\_WORKSPACE: prod

run: terraform apply -auto-approve -var="environment=prod"

**🧩 Features Covered**

| **Feature** | **Description** |
| --- | --- |
| ✅ Stage 1 - dev validation | Run on dev push: validate & plan |
| ✅ Auto PR to main | Promotes via peter-evans/create-pull-request |
| ✅ Stage 2 - main validation | Ensures plan works on main |
| ✅ Stage 3 - main deploy | Final apply to **production** |
| 🔒 Supports GitHub Environments | Uses environment: production for approval gates |

**📦 Want this inside your ZIP repo?**

I can:

* Add this to .github/workflows/multi-stage-iac.yml
* Update your ZIP and PDF

Would you like me to do that now?

Here's how to **create two workspaces (dev and prod) in Terraform Cloud (TFC)** manually and programmatically:

**✅ Option 1: Manual Setup in Terraform Cloud UI**

1. **Log in to Terraform Cloud** → [https://app.terraform.io](https://app.terraform.io/)
2. **Select your organization**
3. Click **"New Workspace"**
4. For each workspace:
   * **Name**:
     + dev
     + prod
   * **Workflow type**: Version control workflow
   * **Repository**: Connect to your GitHub repository where this pipeline resides
   * Click **"Create Workspace"**
5. After creating both, go to:
   * Workspace > Variables tab
   * Set the following variables for both dev and prod:

| **Type** | **Key** | **Value** |
| --- | --- | --- |
| Variable | environment | dev or prod |
| Variable | location | "East US" |
| Variable | resource\_group\_name | "your-rg-name" |
| Secret | ARM\_CLIENT\_ID | From your Azure app reg |
| Secret | ARM\_CLIENT\_SECRET | From your Azure app reg |
| Secret | ARM\_SUBSCRIPTION\_ID | From your Azure |
| Secret | ARM\_TENANT\_ID | From your Azure |

**✅ Option 2: Automate with tfc CLI or API (Preferred for Scripting)**

If you're automating everything, install [Terraform Cloud CLI (tfc)](https://developer.hashicorp.com/terraform/cloud-docs/tfc-cli).

# Log in

tfc login

# Set organization name

ORG\_NAME="your-org-name"

# Create dev workspace

tfc workspace create \

--name dev \

--organization $ORG\_NAME \

--vcs-repo-id "your-github-org/your-repo-name" \

--vcs-repo-branch "dev" \

--vcs-repo-oauth-token-id "your-oauth-token-id"

# Create prod workspace

tfc workspace create \

--name prod \

--organization $ORG\_NAME \

--vcs-repo-id "your-github-org/your-repo-name" \

--vcs-repo-branch "main" \

--vcs-repo-oauth-token-id "your-oauth-token-id"

You can get your OAuth token ID under:  
**Settings > VCS Providers > Token ID**

**✅ Next Step (Optional Automation):**

Want me to generate a Terraform file that **provisions these two workspaces using the Terraform Cloud Provider** itself? (IaC style)?

Let me know!

Absolutely! Let’s go into **minute step-by-step detail**—from **developer push** to **Azure deployment**, all **fully automated** using GitHub, GitHub Actions, and Terraform Cloud.

**🎯 Goal:**

The moment a developer pushes to the dev branch:

1. ✅ Run all checks (fmt, validate, tfsec, Trivy, etc.)
2. ✅ Validate & Plan in **Terraform Cloud (TFC)** using dev workspace
3. ✅ Automatically create a PR from dev → main
4. ✅ Automatically merge the PR (optional, can be manual approval)
5. ✅ TFC detects main change → deploys to Azure (prod workspace)

**⚙️ Pre-requisites Setup (One-Time)**

| **Task** | **Tool** |
| --- | --- |
| 🔑 Terraform Cloud Account | [Terraform Cloud](https://app.terraform.io/) |
| 🏷️ Create 2 workspaces in TFC | dev and prod |
| 🔐 GitHub repo secrets | TFC\_API\_TOKEN, ARM\_CLIENT\_ID, ARM\_CLIENT\_SECRET, ARM\_SUBSCRIPTION\_ID, ARM\_TENANT\_ID |
| 📄 Configure .terraform/workspace.tf | with remote backend for TFC |
| 🔐 Branch protection on main | Require checks before merge |

**🛠️ 1. Developer Pushes Code to dev Branch**

**🔁 GitHub Trigger:**

on:

push:

branches: [ "dev" ]

**🔍 2. Run Checks: Terraform Format, Validate, tfsec, Trivy**

**✅ GitHub Actions: .github/workflows/dev-pr-check.yml**

name: Dev CI with Terraform Cloud

on:

push:

branches: [ "dev" ]

jobs:

dev-ci:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- name: Setup Terraform

uses: hashicorp/setup-terraform@v3

- name: Terraform Init

run: terraform init

- name: Terraform Format

run: terraform fmt -check

- name: Terraform Validate

run: terraform validate

- name: tfsec

uses: aquasecurity/tfsec-action@v1.0.0

- name: Trivy Scan

uses: aquasecurity/trivy-action@master

with:

scan-type: "config"

scan-ref: "./"

- name: Terraform Plan (Dev)

run: terraform plan -var="environment=dev"

env:

TF\_WORKSPACE: dev

💡 This uses the dev Terraform Cloud workspace behind the scenes.

**🤖 3. Auto-Create Pull Request from dev → main**

**✅ Add to same dev-pr-check.yml:**

- name: Create Pull Request

uses: peter-evans/create-pull-request@v6

with:

token: ${{ secrets.GITHUB\_TOKEN }}

title: "Promote VNet changes to main"

base: main

branch: dev

body: |

Auto-generated PR after successful checks on dev branch.

🧠 This PR will trigger further checks.

**✅ 4. PR Triggered – Run Same Checks on main**

**📁 .github/workflows/pr-check.yml**

on:

pull\_request:

branches: [ "main" ]

types: [opened, synchronize, reopened]

jobs:

validate-main:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- name: Setup Terraform

uses: hashicorp/setup-terraform@v3

- name: Terraform Init

run: terraform init

- name: Terraform Validate

run: terraform validate

- name: Terraform Plan (Prod)

run: terraform plan -var="environment=prod"

env:

TF\_WORKSPACE: prod

✅ Ensures everything works in main/prod environment.

**⚡ 5. (Optional) Auto-Merge Pull Request After Approval**

**📁 .github/workflows/automerge.yml**

name: Automerge PR

on:

pull\_request:

types: [labeled]

jobs:

automerge:

runs-on: ubuntu-latest

steps:

- uses: pascalgn/automerge-action@v0.15.6

env:

GITHUB\_TOKEN: ${{ secrets.GITHUB\_TOKEN }}

To trigger auto-merge:

* Add label automerge to the PR (via UI or automation)

**🚀 6. Merge to main Triggers Terraform Cloud to Deploy**

**Terraform Cloud** is connected to the repo.  
When main updates:

* TFC runs terraform plan
* Then applies changes to **Azure** using prod workspace

If **manual approval is enabled** on the TFC workspace:

* Someone approves in UI Otherwise:
* Auto-applies to Azure

**🎯 Automation Summary Flow**

| **Step** | **Trigger** | **Tools Used** | **Result** |
| --- | --- | --- | --- |
| 1 | Push to dev | GitHub Actions | Format, validate, plan |
| 2 | Auto-create PR | create-pull-request | PR from dev → main |
| 3 | PR checks | GitHub Actions | Validate, plan in prod |
| 4 | (Optional) Auto-merge | automerge-action | PR auto-merged |
| 5 | Merge to main | TFC | Apply to Azure prod |

**🚦 Bonus: Branch Protection (GitHub UI)**

* Require status checks (PR checks)
* Require pull request review (if manual)
* Protect main from direct pushes

Would you like this:

1. As a downloadable template (repo)
2. As a visual PDF/flowchart
3. Or a video-like step-through

Let me know and I’ll generate it!

✅ Here's your complete GitHub .github workflows folder packaged for download:

📦 [Download github\_terraform\_auto\_flow.zip](sandbox://mnt/data/github_terraform_auto_flow.zip)

Now generating the **visual flowchart** of this CI/CD automation…

✅ Here's your visual flowchart of the automated Terraform Cloud CI/CD pipeline:

Here’s a **step-by-step, end-to-end guide**—explained in **layman’s terms**—to go from **creating a GitHub repository** to **deploying an Azure Virtual Network (VNet)** into **production using GitHub Actions and Terraform Cloud**. This is perfect for someone who’s new to Infrastructure as Code (IaC) or automation.

**🛠️ 1. Create a GitHub Repository**

This is your project folder in the cloud.

1. Go to [GitHub](https://github.com/)
2. Click **"New Repository"**
3. Name it something like azure-vnet-infra
4. Choose **private** or **public**
5. Check **Initialize with README**

🎉 Now you have a place to store your code!

**🧱 2. Write Your Terraform Code**

Terraform is like a blueprint to tell Azure what to build.

Create a few files:

* main.tf → Defines what Azure resources (e.g., VNet) to create.
* variables.tf → Takes input values (like region, name).
* terraform.tfvars → Your actual input values.
* versions.tf → Sets the Terraform and provider versions.

📄 Example main.tf:

provider "azurerm" {

features {}

}

resource "azurerm\_virtual\_network" "main\_vnet" {

name = "my-vnet"

location = var.location

resource\_group\_name = var.resource\_group

address\_space = ["10.0.0.0/16"]

}

**🔐 3. Push the Code to GitHub**

You upload your files from your computer to the GitHub repository.

Use VS Code or terminal:

git init

git remote add origin https://github.com/your-username/azure-vnet-infra.git

git add .

git commit -m "Initial VNet setup"

git push -u origin main

**☁️ 4. Create a Terraform Cloud Workspace**

Terraform Cloud will run your code securely for you.

1. Go to [Terraform Cloud](https://app.terraform.io/)
2. Create an **organization**
3. Create a **workspace** (e.g., vnet-prod)
4. Set the workspace to use **Version Control Workflow**
5. Connect it to your GitHub repo

🔐 Go to **Settings > Variables**:

* Add Azure credentials as environment variables (like ARM\_CLIENT\_ID, ARM\_CLIENT\_SECRET, etc.)

**🤖 5. Create GitHub Actions Workflow**

This is the automation that tells GitHub to run your Terraform plan when a PR is opened or merged.

📁 .github/workflows/vnet-pr-check.yml

name: PR CI for VNet

on:

pull\_request:

branches: [ "main", "dev" ]

jobs:

validate:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- name: Setup Terraform

uses: hashicorp/setup-terraform@v3

with:

terraform\_version: 1.5.6

- name: Terraform Init & Validate

run: |

terraform init

terraform validate

You can also add terraform plan to see what changes will be made.

**🔁 6. Open a Pull Request (PR)**

Think of this as saying: "Hey team, I want to make a change!"

1. Create a new branch: dev
2. Make a change (e.g., change VNet name)
3. Push it
4. Open a PR from dev → main

GitHub Actions now: ✅ Validates the code  
✅ Shows what will be created/changed

**✅ 7. Merge the PR**

Once approved, you merge it into main — the production branch.

**🚀 8. Terraform Cloud Automatically Deploys to Azure**

Once the PR is merged into main, Terraform Cloud detects the change and:

1. Pulls the latest code from GitHub
2. Runs terraform plan
3. Applies it — i.e., **actually creates the VNet in Azure**

You get a UI to approve or auto-apply the changes.

**🔒 9. Add Branch Protection Rules**

To prevent accidental changes:

* Require PR reviews
* Require passing CI checks before merge
* Protect main from direct push

**🎯 Final Outcome:**

* Your code lives in GitHub
* PRs run checks automatically
* Merged code gets deployed via Terraform Cloud
* Azure creates the Virtual Network — production ready!

Would you like:

* A working codebase as a ZIP?
* A visual diagram of this whole flow?
* Extension to include Subnets, NSGs, or Azure Firewall?

Let me know how far you want to take this!

Your .github folder with dynamic environment-based PR workflows integrated with Terraform Cloud is ready.

📦 [Download github\_pr\_workflow.zip](sandbox://mnt/data/github_pr_workflow.zip)

This includes:

* pr-check.yml: Terraform validation + security scans with dynamic workspace (dev or prod) based on PR target branch
* auto-review.yml: Auto-assign reviewers and apply labels
* auto\_assign.yml: Config file for reviewer assignment

Let me know if you'd like to add:

* Auto-merge for bot PRs
* Slack or Teams notifications
* GitHub Environments and secrets setup guide

Here is the visual diagram of the GitHub version control flow:

Now, here’s a basic **Terraform script for GitHub management**, which includes repository creation, teams, collaborators, and branch protection rules.

**📁 Terraform for GitHub Management**

**🔸 main.tf**

provider "github" {

token = var.github\_token

owner = var.github\_owner

}

resource "github\_repository" "enterprise\_app" {

name = "enterprise-app-infra"

description = "Enterprise Infrastructure Repo"

visibility = "private"

auto\_init = true

}

resource "github\_branch\_protection" "main" {

repository\_id = github\_repository.enterprise\_app.node\_id

pattern = "main"

required\_status\_checks {

strict = true

contexts = ["Terraform Validate", "CodeQL"]

}

required\_pull\_request\_reviews {

dismiss\_stale\_reviews = true

required\_approving\_review\_count = 2

}

enforce\_admins = true

}

resource "github\_branch\_protection" "dev" {

repository\_id = github\_repository.enterprise\_app.node\_id

pattern = "dev"

required\_status\_checks {

strict = false

contexts = ["Terraform Validate"]

}

required\_pull\_request\_reviews {

required\_approving\_review\_count = 1

}

enforce\_admins = false

}

**🔸 variables.tf**

variable "github\_token" {

type = string

description = "GitHub personal access token"

sensitive = true

}

variable "github\_owner" {

type = string

description = "GitHub organization or user"

}

**🔸 terraform.tfvars**

github\_token = "ghp\_your\_access\_token"

github\_owner = "your-org-name"

Let me know if you’d like to:

* Add GitHub Actions workflows via Terraform
* Manage teams, CODEOWNERS, or secrets
* Apply this setup for multiple repositories in a module-based structure

Perfect! Let’s go step-by-step for a **complete, secure, and DevSecOps-focused Azure IaC implementation** using:

* ✅ GitHub (version control with branching, PRs, branch protection),
* ✅ GitHub Actions (multi-environment CI/CD workflows),
* ✅ Terraform Cloud (remote backend, RBAC, execution).

**🧱 Step 1: GitHub Repo Setup**

**✅ 1.1. Create a GitHub Repository**

Create a new private repo called azure-iac-infra.

**✅ 1.2. Set up directory structure**

azure-iac-infra/

├── .github/workflows/terraform.yml

├── environments/

│ ├── dev/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ ├── backend.tf

│ │ └── terraform.tfvars

│ ├── staging/

│ └── prod/

├── modules/

│ └── network/

├── policies/

│ ├── sentinel/

│ └── conftest/

**🌳 Step 2: Branching Strategy**

| **Branch** | **Purpose** |
| --- | --- |
| main | Production |
| staging | QA/Staging |
| dev | Development |
| feature/\* | New features |
| hotfix/\* | Urgent fixes |

**✅ Set Branch Protection (via GitHub UI or gh CLI)**

For main and staging:

* Require PR reviews
* Require passing checks
* Restrict pushes (allow only CI bots or admins)

**☁️ Step 3: Terraform Cloud Setup**

**✅ 3.1. Create Terraform Cloud Organization**

Go to [Terraform Cloud](https://app.terraform.io/), create an organization.

**✅ 3.2. Create Workspaces**

* dev
* staging
* prod

**✅ 3.3. Set Backend Config**

Each environments/dev/backend.tf:

terraform {

backend "remote" {

organization = "your-org-name"

workspaces {

name = "dev"

}

}

}

**✅ 3.4. Authenticate Terraform Cloud**

In GitHub, create a secret:

TF\_API\_TOKEN = <your Terraform Cloud User API Token>

**🔐 Step 4: Terraform RBAC Setup**

**✅ 4.1. Team-Based Access in TFC**

* DevOps: Full access to all workspaces
* Developers: Plan-only
* Security: Read + Policy Management
* Auditors: Read-only

**✅ 4.2. Azure Role Assignment via Terraform**

resource "azurerm\_role\_assignment" "example" {

principal\_id = var.principal\_id

role\_definition\_name = "Contributor"

scope = azurerm\_resource\_group.example.id

}

**🚀 Step 5: GitHub Actions CI/CD Pipeline**

Create a file .github/workflows/terraform.yml:

name: Terraform CI/CD

on:

push:

branches:

- dev

- staging

- main

pull\_request:

branches:

- "\*"

jobs:

terraform:

name: Terraform Plan & Apply

runs-on: ubuntu-latest

defaults:

run:

shell: bash

env:

TF\_TOKEN\_app\_terraform\_io: ${{ secrets.TF\_API\_TOKEN }}

ENV: ${{ github.ref\_name }}

steps:

- name: Checkout Code

uses: actions/checkout@v4

- name: Set Environment Path

run: echo "TF\_ENV=environments/${{ github.ref\_name }}" >> $GITHUB\_ENV

- name: Setup Terraform

uses: hashicorp/setup-terraform@v2

with:

terraform\_version: 1.6.6

- name: Terraform Init

run: terraform init

working-directory: ${{ env.TF\_ENV }}

- name: Terraform Format Check

run: terraform fmt -check

working-directory: ${{ env.TF\_ENV }}

- name: Terraform Validate

run: terraform validate

working-directory: ${{ env.TF\_ENV }}

- name: Terraform Plan

run: terraform plan -out=tfplan

working-directory: ${{ env.TF\_ENV }}

- name: Terraform Apply (Auto for dev/staging)

if: github.ref\_name != 'main'

run: terraform apply -auto-approve tfplan

working-directory: ${{ env.TF\_ENV }}

- name: Manual Approval (prod only)

if: github.ref\_name == 'main'

run: echo "Manual approval required in TFC"

**🧪 Step 6: DevSecOps Integrations**

**✅ 6.1. Add TFSec (Security Scan)**

- name: Run TFSec Scan

uses: aquasecurity/tfsec-action@v1.0.0

with:

working\_directory: ${{ env.TF\_ENV }}

**✅ 6.2. Add Checkov (Policy as Code)**

- name: Checkov Scan

uses: bridgecrewio/checkov-action@master

with:

directory: ${{ env.TF\_ENV }}

**✅ 6.3. Conftest with OPA**

Install conftest and validate:

- name: Install conftest

run: |

wget https://github.com/open-policy-agent/conftest/releases/download/v0.36.0/conftest\_0.36.0\_Linux\_x86\_64.tar.gz

tar -xzf conftest\_0.36.0\_Linux\_x86\_64.tar.gz

sudo mv conftest /usr/local/bin/

- name: Run Conftest

run: conftest test ${{ env.TF\_ENV }}

Store your OPA policies under policies/conftest/.

**🔒 Step 7: Azure Credentials Setup**

Use [OIDC-based GitHub-Azure login](https://learn.microsoft.com/en-us/azure/developer/github/connect-from-azure?tabs=azure-cli%2Clinux):

* Register your GitHub repo as an Azure Federated Identity
* Create a secret: AZURE\_CREDENTIALS (if using JSON-based auth instead)

**📦 Step 8: Example Terraform Code (main.tf)**

provider "azurerm" {

features = {}

}

resource "azurerm\_resource\_group" "example" {

name = "rg-${var.env}"

location = "East US"

}

output "resource\_group\_name" {

value = azurerm\_resource\_group.example.name

}

**📘 Step 9: Optional Enhancements**

* Enable **Terraform Drift Detection**
* Enable **Cost Estimation** in Terraform Cloud
* Add Slack/MS Teams webhook for notifications
* Add audit logging via GitHub/Azure

Would you like me to:

1. Generate a **sample GitHub repo with all the files**?
2. Help you **integrate Terraform Sentinel policies**?
3. Set up **OIDC-based Azure login from GitHub Actions**?

Let me know which you'd like first!

Here’s a **complete, end-to-end implementation plan** for Azure Infrastructure as Code (IaC) using:

* **GitHub** (version control, branch protection, PRs, branching strategy),
* **GitHub Actions** (CI/CD workflows with DevSecOps focus),
* **Terraform Cloud** (backend and execution),
* **RBAC integration** across all stages.

**🌐 1. GitHub Repository Setup**

**🗂️ Repo Structure (Monorepo with environment folders)**

azure-iac-infra/

├── .github/

│ └── workflows/

│ └── terraform-pipeline.yml

├── environments/

│ ├── dev/

│ │ ├── main.tf

│ │ ├── variables.tf

│ │ └── backend.tf

│ ├── staging/

│ └── prod/

├── modules/

│ └── core\_network/

├── policies/

│ ├── sentinel/

│ └── conftest/

└── README.md

**🌳 Branching Strategy**

* main: Production branch
* staging: Pre-prod testing
* dev: Feature development
* feature/\*: New features
* hotfix/\*: Critical fixes

**✅ Branch Protection Rules**

* Enable for main and staging:
  + Require pull request reviews
  + Require status checks to pass before merging
  + Restrict who can push (e.g., Terraform Cloud, GitHub Actions only)

**⚙️ 2. Terraform Cloud Backend**

**🧰 Setup:**

* **Workspace for each environment**: dev, staging, prod
* Use **CLI-driven workflow**
* Enable **remote operations**, **state locking**, and **Terraform version pinning**
* Enable **variable sets** for reusability
* Use **Team-based access** for RBAC (more on that below)

**backend.tf (dev example):**

terraform {

backend "remote" {

organization = "your-org-name"

workspaces {

name = "dev"

}

}

}

**🔐 3. RBAC Implementation**

**Terraform Cloud**

* Teams: DevOps, Developers, Security, Auditors
* Permissions:
  + DevOps: Full Admin (Run, Manage Workspaces, Variables)
  + Developers: Plan only (No Apply)
  + Security: Manage policies (Sentinel/OPA)
  + Auditors: Read-only

**Azure**

* Use azurerm\_role\_assignment in Terraform:

resource "azurerm\_role\_assignment" "dev\_contributor" {

principal\_id = var.dev\_principal\_id

role\_definition\_name = "Contributor"

scope = azurerm\_resource\_group.dev.id

}

**🔄 4. GitHub Actions CI/CD Pipeline**

**Sample Workflow: .github/workflows/terraform-pipeline.yml**

name: Terraform Multi-Env Pipeline

on:

push:

branches:

- dev

- staging

- main

pull\_request:

branches:

- "\*"

jobs:

terraform:

name: Terraform ${{ github.ref\_name }}

runs-on: ubuntu-latest

env:

TF\_CLOUD\_ORGANIZATION: your-org-name

TF\_WORKSPACE: ${{ github.ref\_name }}

TF\_TOKEN\_app\_terraform\_io: ${{ secrets.TF\_API\_TOKEN }}

steps:

- name: Checkout

uses: actions/checkout@v3

- name: Setup Terraform

uses: hashicorp/setup-terraform@v2

- name: Terraform Format

run: terraform fmt -check

- name: Terraform Init

run: terraform init

- name: Terraform Validate

run: terraform validate

- name: Terraform Plan

run: terraform plan -out=tfplan

- name: Upload Plan as Artifact

uses: actions/upload-artifact@v3

with:

name: tfplan-${{ github.ref\_name }}

path: tfplan

- name: Manual Approval for Prod

if: github.ref == 'refs/heads/main'

uses: hmarr/auto-approve-action@v3

with:

github-token: ${{ secrets.GITHUB\_TOKEN }}

- name: Terraform Apply (non-prod)

if: github.ref != 'refs/heads/main'

run: terraform apply -auto-approve

**🛡️ 5. DevSecOps Enhancements**

**Static Security Scans**

* **TFSec**, **Checkov**: Add as pre-check steps before terraform plan.

**Compliance-as-Code**

* **OPA (Conftest)** or **Sentinel**:
  + Write rules to enforce naming conventions, tagging, disallowed resources, etc.

- name: Run Checkov

uses: bridgecrewio/checkov-action@master

with:

directory: ./environments/${{ github.ref\_name }}

- name: Run Conftest

run: conftest test ./environments/${{ github.ref\_name }}

**Secrets Management**

* Use **GitHub Secrets** for TF API Token, Azure Credentials
* Store Azure credentials securely as AZURE\_CREDENTIALS\_JSON

**📊 6. CI/CD Pipeline Flow per Environment**

| **Branch** | **Env** | **Actions** | **Apply Type** |
| --- | --- | --- | --- |
| dev | dev | fmt, validate, plan, apply | Auto |
| staging | staging | fmt, validate, plan, apply | Auto |
| main | prod | fmt, validate, plan | Manual PR review + approval |

**📘 7. Additional Recommendations**

* **Enforce drift detection** (Terraform Cloud supports it)
* **Cost Estimation**: Enable in Terraform Cloud
* **Add notifications** via Slack/Teams from GitHub Actions
* **Audit Trails**: Use GitHub audit logs + Terraform Cloud run logs

Would you like a **sample repo template** or a **dashboard view for CI/CD & compliance status**? I can generate those next.