Terraform GitHub Actions with OIDC for Multi-Environment Deployments - Azure

Terraform GitHub Actions with OIDC for Multi-Environment Deployments (dev/test/prod)

Overview

This guide explains how to set up a single Terraform codebase with GitHub Actions, OpenID Connect (OIDC) authentication, and environment-specific branching (dev, test, prod). It ensures each environment uses separate state files and credentials.

Step 1: Define Branches for Each Environment

Purpose: Segregate code and control deployment scope.

Branches:
- `dev`
- `test`
- `main` (for prod)

Federated Credential Subjects:
repo:<owner>/<repo>:ref:refs/heads/dev
repo:<owner>/<repo>:ref:refs/heads/test
repo:<owner>/<repo>:ref:refs/heads/main

Step 2: Configure Federated Credentials in Azure AD

Purpose: Allow GitHub Actions to authenticate to Azure via OIDC.

Use the following PowerShell snippet to register the App and Federated Credentials:

```powershell
$REPO = "qatip/terraform-az-galab"
$APP_NAME = "gha-terraform"
$BRANCHES = @("dev", "test", "main")
foreach ($BRANCH in $BRANCHES) {
  $subject = "repo:$REPO:ref:refs/heads/$BRANCH"
  # Create federated credential for this branch...
}
```

```

## Step 3: Store GitHub Secrets and Variables

Secrets:
- `AZURE_CLIENT_ID`
- `AZURE_TENANT_ID`
- `AZURE_SUBSCRIPTION_ID`

Variables:
- `STATE_RG`
- `STATE_STORAGE`
- `STATE_CONTAINER`

## Step 4: Add GitHub Actions Workflows

Use a single shared `plan.yml` and `apply.yml` across all branches.

Dynamic ENV selector:
```
env:
  ENV: ${{ github.ref_name == 'main' && 'prod' || github.ref_name }}
```

## Step 5: Configure Terraform Backend Dynamically

**main.tf:**
```
terraform {
  backend "azurerm" {}
}
```

**plan.yml / apply.yml:**
```
- name: Terraform init
  run: |
    terraform init   -backend-config="resource_group_name=${STATE_RG}"   -backend-
config="storage_account_name=${STATE_STORAGE}"   -backend-
config="container_name=${STATE_CONTAINER}"   -backend-config="key=${ENV}.tfstate"
```

## Step 6: Use tfvars for Environment-specific Config

Store the following files:
- `dev.tfvars`
```

- `test.tfvars`
- `prod.tfvars`

Terraform plan step:
```yaml
- name: Terraform plan
  run: |
    terraform plan -var-file="${ENV}.tfvars" -out="plan.tfplan"
```

Step 7: Destroy Logic (Optional)

Use a separate workflow `destroy.yml` with:
```yaml
run: terraform destroy -auto-approve -var-file="${ENV}.tfvars"
```

Summary

This setup allows you to:
- Maintain one Terraform codebase.
- Dynamically switch environment with branches.
- Use GitHub OIDC instead of static credentials.
- Isolate state files and tfvars for each environment.