

Class 6: Automating Terraform with GitHub Actions

Step 1: Preparing AWS

1. Log in to AWS

- Access your AWS account at [AWS Console](#).

2. Navigate to IAM Dashboard

- Once logged in, type "IAM" in the search bar and navigate to the Identity and Access Management dashboard.

3. Set Up IAM Permissions

- Click **User Groups** in the left-hand menu.
- Create a new user group with these permissions
 - **AdministratorAccess**
 - **AmazonAPIGatewayAdministrator**
 - **SystemAdministrator**
- Save the user group.

4. Create an IAM User

- Go back to the IAM dashboard and select **Users** from the left-hand menu.
- Click **Create Users**:
 - Enter the user name **Terraform**.
 - Click "Next"
- Assign the user to the previously created user group.
- Finish creating the user.

5. Generate Access Keys

- Click the new user.
- Under the **Summary**, click "Create Access Key."
- Command Line Interface (CLI)
- Click "I understand the above recommendation and want to proceed to create an access key."
- Create access key
- Save the Access Key ID and Secret Access Key.

Step 2: Setting Up Terraform Cloud

1. Sign Up or Log In

- Go to [Terraform Cloud](#). If you don't already have an account, click "Sign Up."

2. Create an Organization

- Once logged in, navigate to the **Organizations** tab.

- Click "Create New Organization."
- Name the organization appropriately and click "Create."

3. Set Up a Workspace

- Inside the organization, create a new workspace:
 - Default Project
 - Choose "API-Driven Workflow" if you are integrating with GitHub.
 - Name the workspace (e.g., "learn-terraform-github-actions").
- Complete the setup.

4. Add AWS Environment Variables

- Navigate to the banner on the left side.
- Select "Variables"
- Under "Workspace variables," add the following environment variables:
 - `AWS_ACCESS_KEY_ID`
 - `AWS_SECRET_ACCESS_KEY`
- Paste the values saved from Step 1.
- Select "Environment variable"
- Select "Sensitive"

Environment Variables

These variables are set in Terraform's shell environment using `export`.

Key	Value	
<code>AWS_ACCESS_KEY_ID</code> SENSITIVE	Sensitive - write only	...
<code>AWS_SECRET_ACCESS_KEY</code> SENSITIVE	Sensitive - write only	...

[+ Add variable](#)

5. Generate a Terraform Cloud API Token

- Go to your **Account Settings** in Terraform Cloud.
 - Left side banner
 - Click on profile picture
 - Click "Account settings"
 - Click "Tokens"
- Under "Tokens," generate a new token and name it (e.g., "GitHub Actions").
- Save the token for later use in GitHub Actions.

Step 3: Setting Up GitHub Repository

1. Fork the Repository

- Navigate to [Theo's repository](#) and click "Fork."
- Create fork

2. Add Terraform Cloud Token as a GitHub Secret

- In your forked repository, navigate to **Settings > Secrets and Variables (Left side banner) > Actions**.
- Click "New Repository Secret."
- Name it **TF_API_TOKEN** and paste the Terraform Cloud API token.
- Add secret

The screenshot shows the GitHub repository settings page. The left sidebar has a 'Settings' menu with 'Secrets and variables' highlighted. The main content area is titled 'Actions secrets and variables'. It has two tabs: 'Secrets' and 'Variables'. A 'New repository secret' button is visible. Below this, there are sections for 'Environment secrets' (empty) and 'Repository secrets' (containing 'TF_API_TOKEN' updated 2 hours ago).

3. Clone the Repository

- On your local machine, create a folder for the project.
- Open a terminal and run the following command:

```
git clone <repository-url>
```

- Navigate into the repository directory:

```
cd <repository-name>
```

Step 4: Updating Terraform Configuration Locally

1. Open the Repository in Visual Studio Code

- Use `cd` to navigate to the repository folder in your terminal.
- Open the repository in Visual Studio Code:

```
code .
```

2. Update Terraform Configuration

- Open the `main.tf` file.
- Locate the `cloud` block and update it with:
 - Your Terraform Cloud organization name.
 - Your workspace name.

```
cloud {  
  
  organization = "your_organization"  
  
  workspaces {  
    name = "your_workspace"  
  }  
}
```

3. Save and Commit Changes

- Save the updated file.
- Create a new branch:

```
git checkout -b 'update-tfc-org'
```

- Stage and commit your changes:

```
git add main.tf  
git commit -m "Use our HCP Terraform organization"
```

4. Push Your Changes

- Push the branch to GitHub:

```
git push -u origin update-tfc-org
```

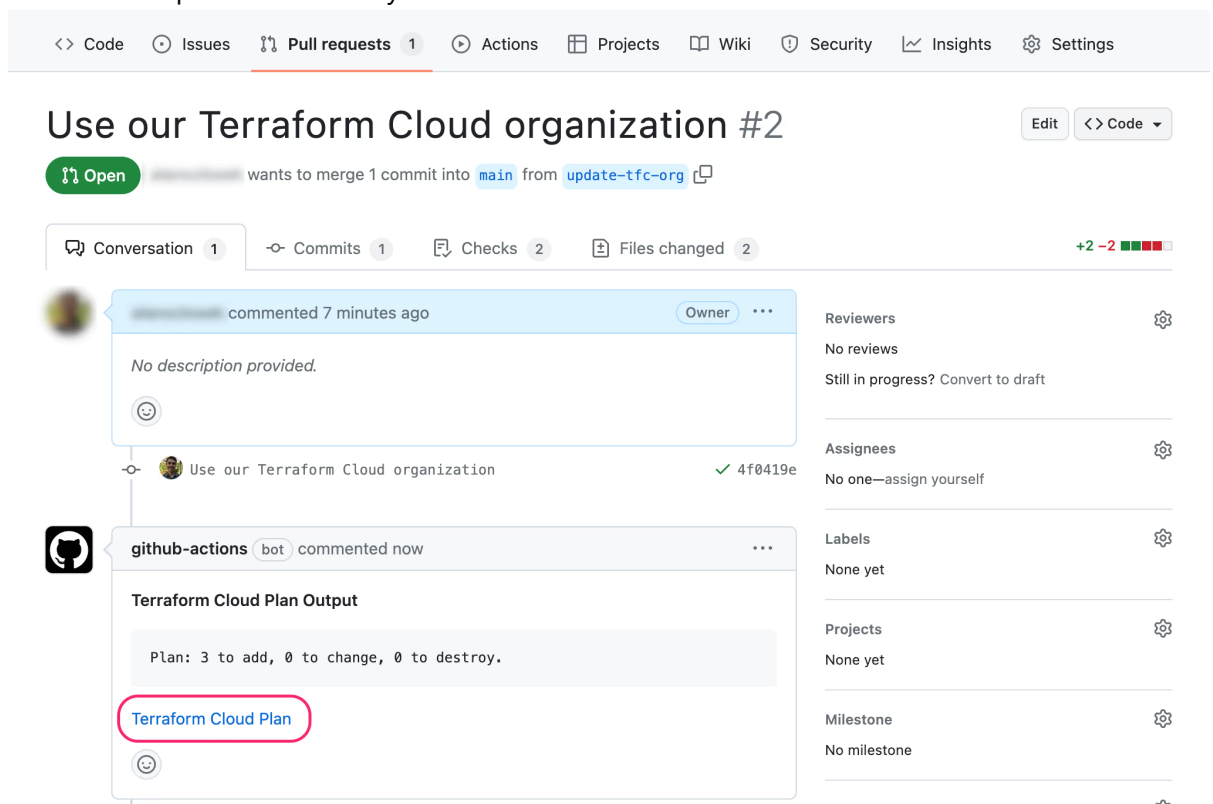
Step 5: Deploying Terraform Plan via GitHub Actions

1. Create a Pull Request (PR)

- On GitHub, go to your repository.
- Click "Pull Requests"
- Click "New"
- For "base repository" select your repo
- For compare: select "update-tfc-org"
- Create it.

2. Wait for Workflow Validation

- On the pull request page wait for the check to be done
- Ensure it completes successfully.



3. Merge the Pull Request

- Once the workflow completes, merge the PR into the main branch.

4. Verify Deployment

- Check the `terraform.yml` workflow in GitHub Actions to confirm success.

■ Click "Actions" on your repo

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

Actions

New workflow

All workflows

Terraform Apply

Terraform Plan

Your Fork

All workflows

Showing runs from all workflows

11 workflow runs

Event

Status

Branch

Actor

Merge pull request #2 from

Terraform Apply #4: Commit 58b53c2 pushed by

main

now

In progress

■ Click on the latest run

Summary

Jobs

Terraform Apply

Run details

Usage

Workflow file

Terraform Apply

succeeded now in 1m 34s

Search logs

Apply

51s

[id=i-07e0a62f76f4949a3]","@module":"terraform.ui","@timestamp":"2023-05-09T17:34:16.799370Z","hook":{"resource":{"addr":"aws_instance.web","module":"","resource":"aws_instance.web","implied_provider":"aws","resource_type":"aws_instance","resource_name":"web","resource_key":null},"action":"create","id_key":"id","id_value":"i-07e0a62f76f4949a3"},"elapsed_seconds":32},"type":"apply_complete"}

38 {"@level":"info","@message":"Apply complete! Resources: 3 added, 0 changed, 0 destroyed.","@module":"terraform.ui","@timestamp":"2023-05-09T17:34:17.206604Z","changes":{"add":3,"change":0,"remove":0,"operation":"apply"},"type":"change_summary"}

39 {"@level":"info","@message":"Outputs: 1","@module":"terraform.ui","@timestamp":"2023-05-09T17:34:17.206737Z","outputs":{"web-address":{"sensitive":false,"type":"string","value":"ec2-54-186-147-12.us-west-2.compute.amazonaws.com:8080"},"type":"outputs"}}

40

41 View Run in Terraform Cloud: <https://app.terraform.io/app/hashicorp-learn/workspaces/learn-terraform-github-actions/runs/run-hvuR2hPZv1mF3JJ8>

42 {

43 "run_id": "run-hvuR2hPZv1mF3JJ8",

44 "run_link": "https://app.terraform.io/app/hashicorp-learn/workspaces/learn-terraform-github-actions/runs/run-hvuR2hPZv1mF3JJ8",

45 "run_status": "applied",

46 "status": "Success"

Post Checkout

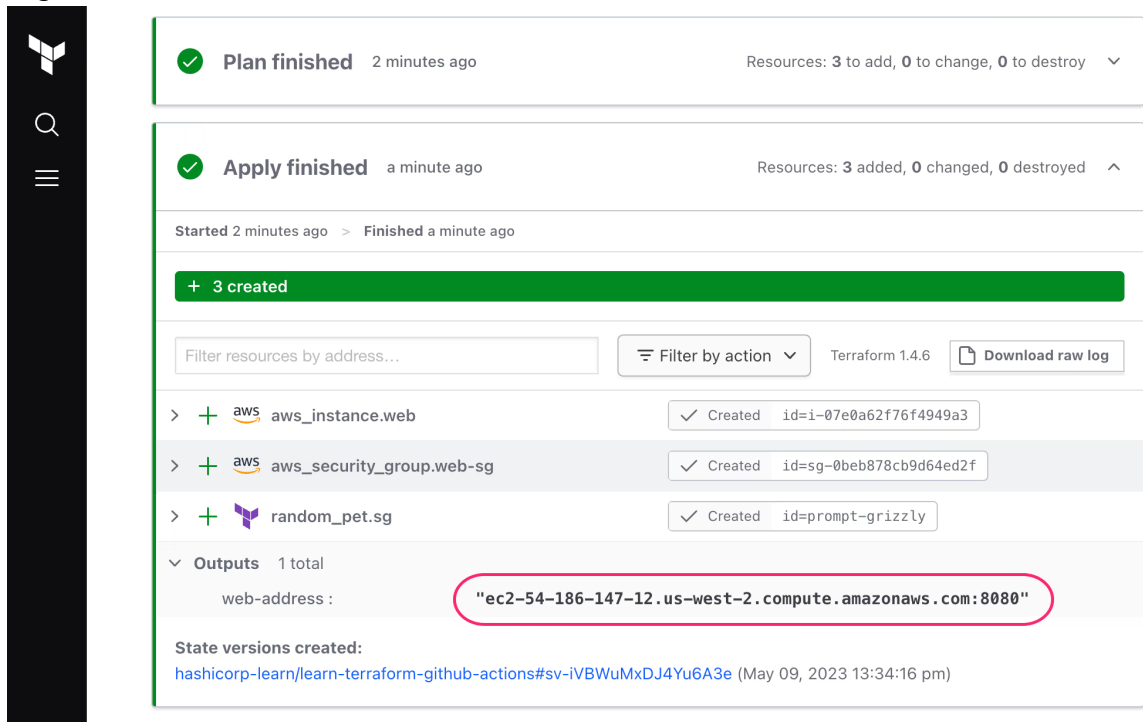
0s

Complete job

0s

6 / 8

- Log in to Terraform Cloud and review the "Runs" section for the latest run.



The screenshot displays the Terraform Cloud interface. On the left is a dark sidebar with the Terraform logo and navigation icons. The main content area shows the status of a Terraform run. At the top, a green checkmark indicates 'Plan finished' 2 minutes ago, with resources: 3 to add, 0 to change, 0 to destroy. Below this, another green checkmark indicates 'Apply finished' a minute ago, with resources: 3 added, 0 changed, 0 destroyed. A green bar shows '+ 3 created'. Below this, a table lists the created resources:

Resource	Action	ID
aws_instance.web	Created	id=i-07e0a62f76f4949a3
aws_security_group.web-sg	Created	id=sg-0beb878cb9d64ed2f
random_pet.sg	Created	id=prompt-grizzly

Below the resources table, the 'Outputs' section shows 1 total output. The 'web-address' output is highlighted with a red circle and contains the value: "ec2-54-186-147-12.us-west-2.compute.amazonaws.com:8080". At the bottom, it shows 'State versions created: hashicorp-learn/learn-terraform-github-actions#sv-ivBWuMxDJ4Yu6A3e (May 09, 2023 13:34:16 pm)'.

5. Validate Resources in AWS Console

- Log in to the AWS console and confirm that the resources specified in `main.tf` are created.

Step 6: Cleaning Up Resources

1. Destroy Resources in Terraform Cloud

- Navigate to your workspace settings in Terraform Cloud.
 - Left side banner
 - "Settings"
 - "Destruction and Deletion"
- Queue a destroy plan and confirm the action.

2. Monitor Resource Deletion

- Watch the "Runs" section in Terraform Cloud to ensure the destroy process completes.

3. Verify Deletion in AWS

- Confirm that all resources have been deleted.

Actually do this so you can get a job - Jourdan

