EARTHLY
Super Simple Builds

Using Terraform with GitHub Actions

7 minute read Updated: June 26, 2023

Keanan Koppenhaver

In this Series

Table of contents

We're Earthly. We make building software simpler and therefore faster. This article is about GitHub Actions, if you'd like to see how Earthly can improve your GitHub Actions builds then check us out.

GitHub Actions is a powerful tool that allows software developers to automate almost everything inside a GitHub repository. From running tests to linting your code, to automatically commenting on pull requests and issues, it's a complete solution that helps projects of all kinds to operate more efficiently.

If you're managing IT infrastructure, you're likely using Terraform, a popular tool for managing infrastructure as code. Thankfully, GitHub Actions and Terraform can work together to create powerful, automated workflows for creating and maintaining even the most complicated deployments.

Using these tools together can help you automate your Terraform pipelines, which is important for making sure they run as frequently and consistently as you would like them to, making your pipelines more repeatable and reliable.

In this article, you'll learn how GitHub Actions and Terraform work together so that you can benefit from this powerful combination.

Using Terraform and GitHub Actions Together

If you're an experienced developer looking to better understand how to use Terraform and GitHub Actions, this tutorial is for you. This means you should already have some knowledge about what Terraform is and how it works to provision infrastructure. In addition, you should have a basic idea of how GitHub Actions works.

Prerequisites

Before you begin, you'll need the following:

- **A GitHub account:** GitHub is where you will store the code used for this project and where you will run your GitHub Actions workflow. You can take a look at the code that goes along with this tutorial in this repository.

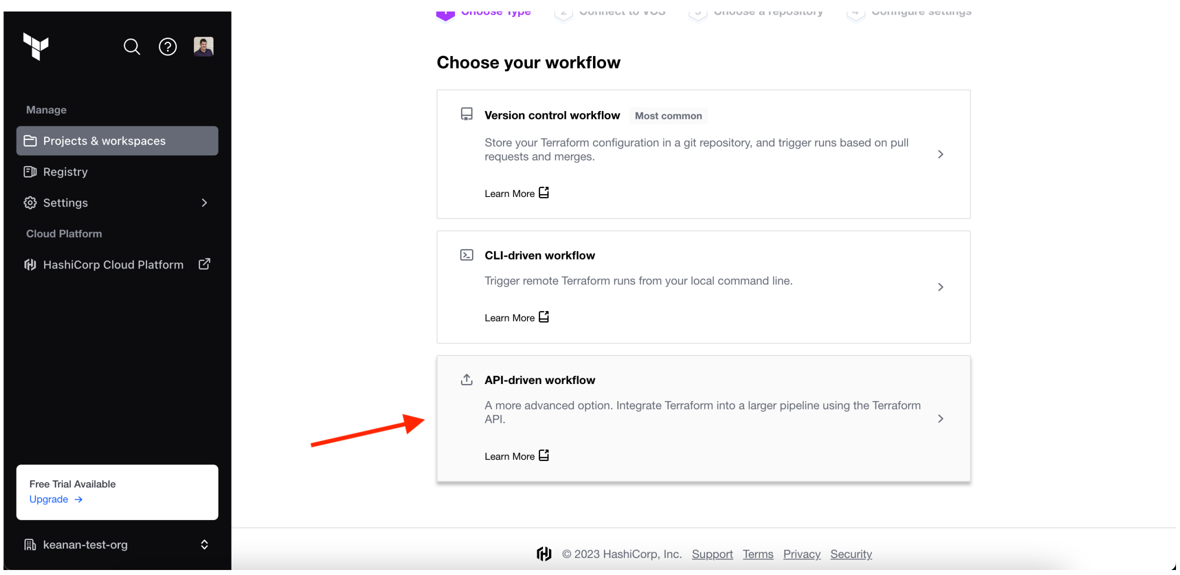
- **An Amazon Web Services (AWS) account and credentials:** Here, you'll be using resources that fall under the AWS free plan; however, if your account does not qualify for the AWS free plan, following this tutorial might incur some charges in your AWS account, so proceed with care.
- **A Terraform Cloud account:** Since this tutorial uses Terraform Cloud to create your infrastructure configuration, you'll need to have a Terraform Cloud account before you begin.

Once you have all these accounts set up, you're ready to get started.

Setting Up Terraform Cloud

When you first set up your Terraform Cloud account and log in, you'll start by creating an organization. An organization is an entity that will hold the workspace that you'll be creating. The organization allows you to collaborate with others and establish an internal workflow that streamlines the process of managing your infrastructure.

Next, you need to create a new workspace and make sure to select **API-driven workflow** when choosing the type. This will ensure that GitHub Actions is able to connect to Terraform later on:



Choose your workflow

Version control workflow Most common
Store your Terraform configuration in a git repository, and trigger runs based on pull requests and merges.

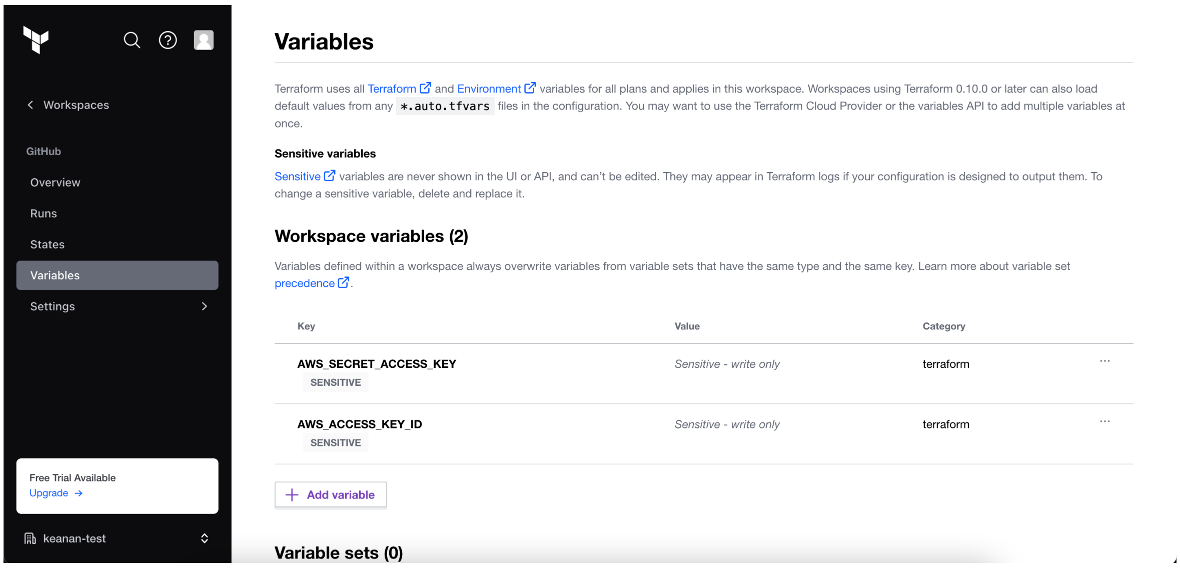
CLI-driven workflow
Trigger remote Terraform runs from your local command line.

API-driven workflow
A more advanced option. Integrate Terraform into a larger pipeline using the Terraform API.

A screenshot showing the setup of a Terraform Cloud Workspace

Next, you need to name your workspace. In this example, the workspace is named "ghactions-terraform-demo". This will create an empty workspace.

Next, you need to add your AWS secrets (*ie* `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`) to your Terraform workspace under the **Variables** section of the sidebar. Make sure you add them as sensitive variables so that only Terraform can use them to authenticate with AWS.



Variables

Terraform uses all Terraform and Environment variables for all plans and applies in this workspace. Workspaces using Terraform 0.10.0 or later can also load default values from any `*.auto.tfvars` files in the configuration. You may want to use the Terraform Cloud Provider or the variables API to add multiple variables at once.

Sensitive variables

Sensitive variables are never shown in the UI or API, and can't be edited. They may appear in Terraform logs if your configuration is designed to output them. To change a sensitive variable, delete and replace it.

Workspace variables (2)

Variables defined within a workspace always overwrite variables from variable sets that have the same type and the same key. Learn more about variable set precedence.

Key	Value	Category
AWS_SECRET_ACCESS_KEY SENSITIVE	Sensitive - write only	terraform
AWS_ACCESS_KEY_ID SENSITIVE	Sensitive - write only	terraform

+ Add variable

Variable sets (0)

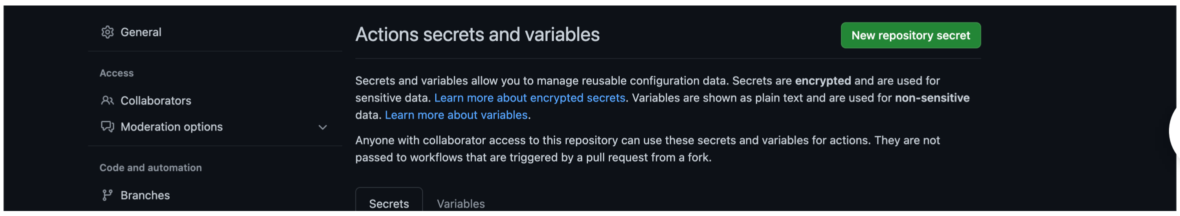
Environment variables after getting added to Terraform Cloud

Once you have all that configured, head over to the **Tokens page** and generate an API token. This is what you'll use to connect GitHub Actions to Terraform Cloud, so make sure you store this information in a safe place.

Creating a GitHub Repository and Configuring Your Action

Now that you've set up Terraform and your API tokens are stored somewhere, head over to GitHub and set up a new repository.

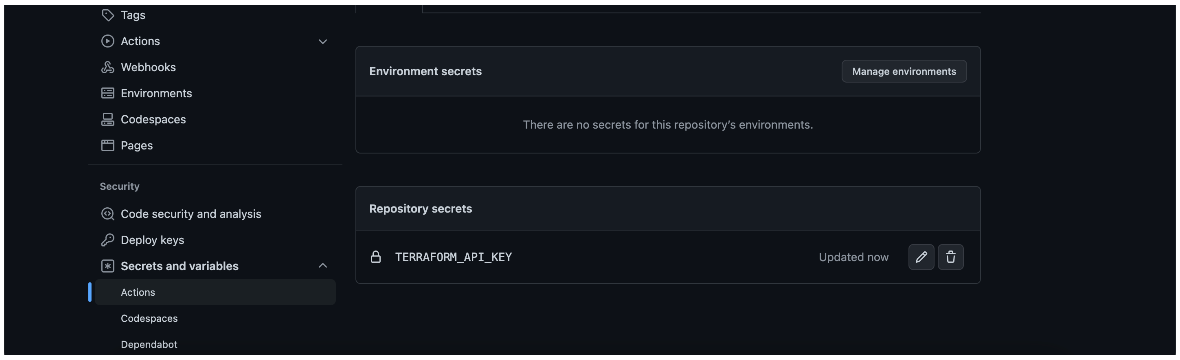
Once you have your repository set up, go to **Settings > Secrets and Variables > Actions** and create a new secret. This is where you need to add the API key you retrieved from Terraform in the previous step. In this example, the secret is `TERRAFORM_API_KEY`, but you can name it whatever you like. If you're using the sample code as a reference, ensure that you replace the name of the secret with whatever you have named yours:



Actions secrets and variables

Secrets and variables allow you to manage reusable configuration data. Secrets are **encrypted** and are used for sensitive data. Learn more about encrypted secrets. Variables are shown as plain text and are used for **non-sensitive** data. Learn more about variables.

Anyone with collaborator access to this repository can use these secrets and variables for actions. They are not passed to workflows that are triggered by a pull request from a fork.



Environment secrets

There are no secrets for this repository's environments.

Repository secrets

TERRAFORM_API_KEY Updated now

A screenshot showing a properly configured Terraform API key as a GitHub secret

Now, you can clone your repository to your local system and create two important files within the directory that you just cloned down from GitHub: `.github/workflows/terraform.yml` and `main.tf`.

To fill the contents of these files, you can use the **Terraform example repo**, but here's an explanation of the content in each piece:

Terraform.yml

This is the actual file that your GitHub action will run through every time it's triggered. There are a few important code snippets in this file:

terraform.yml Copy

```
on:
  push:
    branches:
      - main
```

This defines what triggers your workflow. In this case, your workflow is only triggered when code is pushed to the main branch or when any pull requests to the main branch are merged:

terraform.yml Copy

```
- name: Setup Terraform
        uses: hashicorp/setup-terraform@v1
        with:
          # terraform_version: 0.13.0:
          cli_config_credentials_token: $
```

This is where GitHub Actions sets up the Terraform CLI and uses your Terraform API key to get everything configured to provision your infrastructure. If you named your secret in GitHub something different, this is where you need to replace that name.

The rest of the workflow file goes through the remainder of the Terraform configuration and applies the configuration specified in `main.tf`, which sets up and deploys a micro Amazon Elastic Compute Cloud (Amazon EC2) server on AWS.

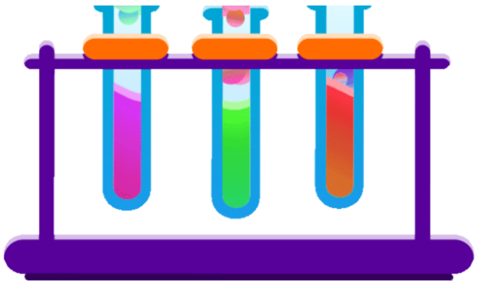
Main.tf

There are a few key sections of the `main.tf` that will help you understand how Terraform is going to provision your infrastructure:

- The `terraform` block has all the information about what version of Terraform is running, the information needed to connect this workflow to Terraform Cloud and information about any other dependencies that this particular configuration has.
- The `data "aws_ami" "ubuntu"` block tells Terraform and AWS which operating system and Amazon Machine Image (AMI) to install on the newly-created instance.
- The `resource "aws_instance" "web"` block determines what commands are run on the AWS instance after it's provisioned by Terraform.

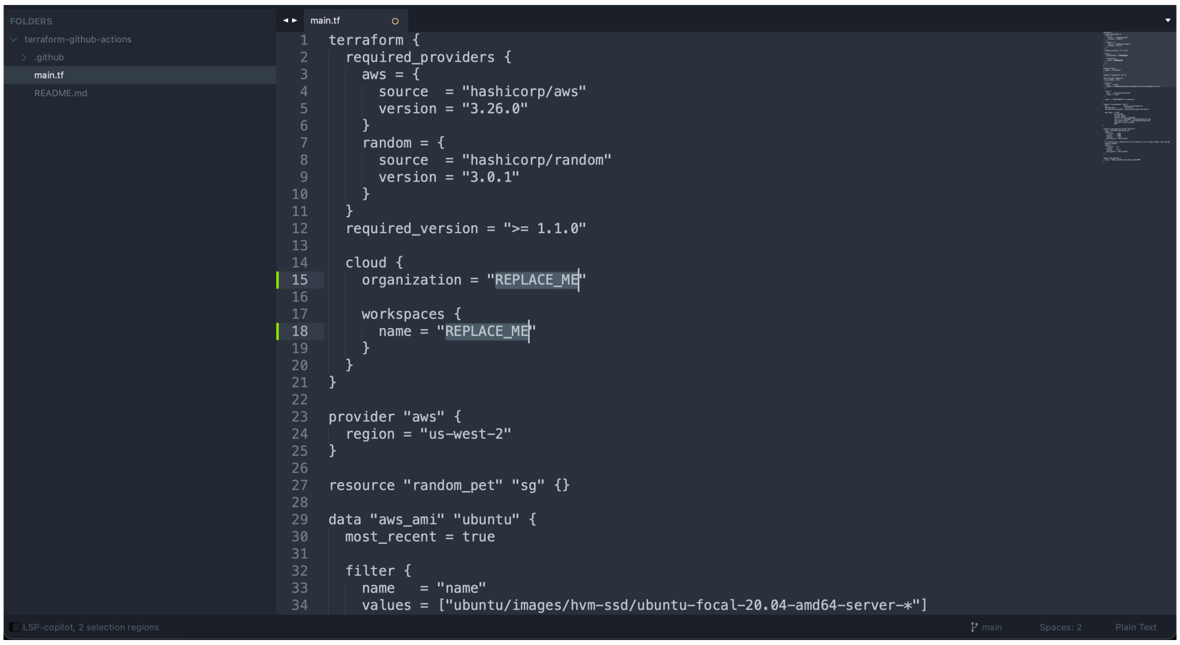
Together, this is all the configuration information needed for your GitHub Actions workflow to execute the provisioning of a new cloud resources via Terraform.

Let's Test It



Now that you have everything configured, it's time to make sure your GitHub action runs and provisions your resources with Terraform.

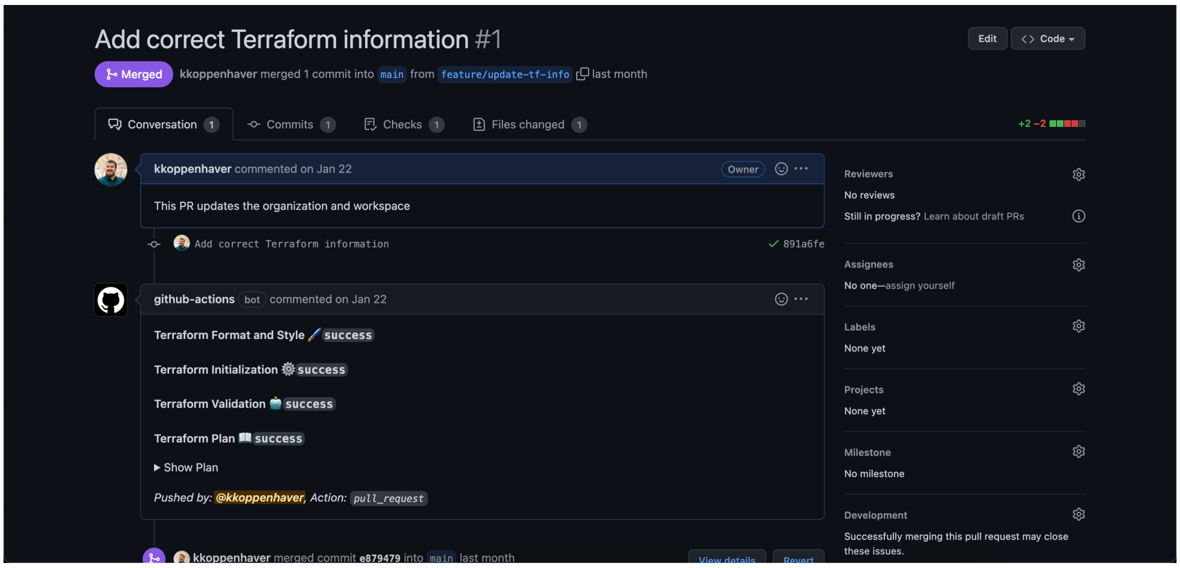
Create a new branch in your GitHub repository or locally via the command line. Inside this branch, navigate to your `main.tf` file and replace the `organization` and the `workspace` parameter value from the default placeholder `REPLACE_ME` with the actual organization and workspace names you created in Terraform Cloud earlier. Then commit and push your branch to GitHub:



```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "3.26.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.0.1"
    }
  }
  required_version = ">= 1.1.0"

  cloud {
    organization = "REPLACE_ME"

    workspaces {
      name = "REPLACE_ME"
    }
  }
}

provider "aws" {
  region = "us-west-2"
}

resource "random_pet" "sg" {}

data "aws_ami" "ubuntu" {
  most_recent = true

  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*"]
```

Replacing the placeholders for **`organization`** and **`workspace`** name

Once you push your branch, create a pull request (PR) against the main branch of your codebase with your changes and replacements. You'll be able to see the output of the Terraform plan inside your PR, informing you of what resources will be created and/or destroyed with that particular PR. At this point, no action has been taken yet, but this offers a preview of what will happen once the PR is merged:


Add correct Terraform information #1

Merged kkoppenhaver merged 1 commit into `main` from `feature/update-tf-info` last month

Conversation 1 · Commits 1 · Checks 1 · Files changed 1 · +2 −2

kkoppenhaver commented on Jan 22 (Owner)

This PR updates the organization and workspace

`Add correct Terraform information` ✓ 891a6fe

github-actions bot commented on Jan 22

Terraform Format and Style `success`

Terraform Initialization `success`

Terraform Validation `success`

Terraform Plan `success`

▶ Show Plan

Pushed by: @kkoppenhaver, Action: `pull_request`

kkoppenhaver merged commit e879479 into `main` last month

Reviewers: No reviews. Still in progress? Learn about draft PRs

Assignees: No one—assign yourself

Labels: None yet

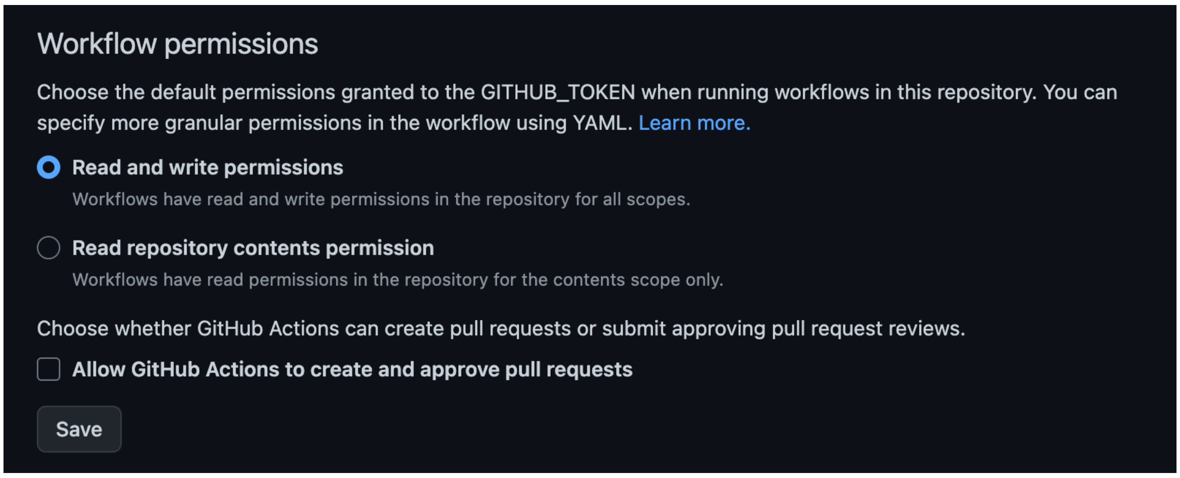
Projects: None yet

Milestone: No milestone

Development: Successfully merging this pull request may close these issues.

GitHub Actions info on a PR within the repository

> ***Please note:** If you get an error when your action runs, head over to your GitHub Actions settings and make sure that GitHub Actions has read and write permissions enabled for your repository:*


Workflow permissions

Choose the default permissions granted to the GITHUB_TOKEN when running workflows in this repository. You can specify more granular permissions in the workflow using YAML. Learn more.

[✓ **Read and write permissions**]
Workflows have read and write permissions in the repository for all scopes.

Read repository contents permission
Workflows have read permissions in the repository for the contents scope only.

Choose whether GitHub Actions can create pull requests or submit approving pull request reviews.

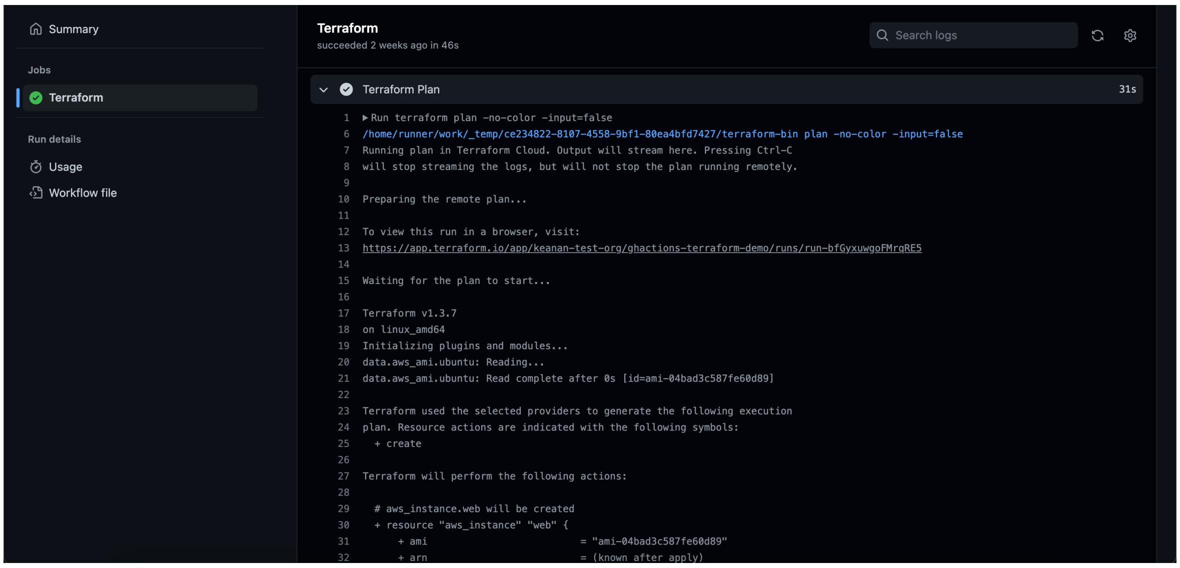
☐ **Allow GitHub Actions to create and approve pull requests**

A correctly configured settings page for GitHub Actions permissions

Once the GitHub action successfully runs on your pull request, you can merge the pull request into the `main` branch and Terraform will provision your infrastructure.

Verify That the EC2 Instance Was Provisioned

Once your pull request is merged and your infrastructure is provisioned, clicking on the GitHub action will show you all the information about the infrastructure you are creating or modifying:



```
▶ Run terraform plan -no-color -input=false
/home/runner/work/_temp/ce234822-8107-4558-9bf1-80ea4bfd7427/terraform-bin plan -no-color -input=false
Running plan in Terraform Cloud. Output will stream here. Pressing Ctrl-C
will stop streaming the logs, but will not stop the plan running remotely.

Preparing the remote plan...

To view this run in a browser, visit:
https://app.terraform.io/app/keanan-test-org/ghactions-terraform-demo/runs/run-bfGyxuwgoFMrqRE5

Waiting for the plan to start...

Terraform v1.3.7
on linux_amd64
Initializing plugins and modules...
data.aws_ami.ubuntu: Reading...
data.aws_ami.ubuntu: Read complete after 0s [id=ami-04bad3c587fe60d89]

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_instance.web will be created
  + resource "aws_instance" "web" {
      + ami                          = "ami-04bad3c587fe60d89"
      + arn                          = (known after apply)
```

Terraform output inside of GitHub Actions

If you're using the infrastructure only for testing purposes and want to avoid charges for the provisioned infrastructure, consider deleting the AWS instance once you've confirmed that it has been provisioned.

Conclusion

Now that you have a taste for automatically provisioning infrastructure, you could make your `main.tf` file much more complicated and provision all sorts of different resources. Load balancers, databases, and more are now all within your reach using the combination of GitHub Actions and Terraform.

And if you're looking to continue building out your automation pipeline, consider using Earthly. Earthly runs everywhere, including GitHub Actions and can improve the reliability of your CI/CD pipelines. It works great with GitHub Actions and Terraform.

Earthly + GitHub Actions

GitHub Actions are better with Earthly. Get faster build speeds, improved consistency, and local testing along with an easy-to-use syntax – no YAML – and better monorepo support.

Go to our homepage to learn more

Keanan Koppenhaver

Keanan is the CTO at Alpha Particle where he helps publishers modernize their technology platforms and build their developer teams.

Writers at Earthly work closely with our talented editors to help them create high quality tutorials. This article was edited by:

Mustapha Ahmad Ayodeji

Ahmad is a Software developer and a Technical writer with so much interest for Django related frameworks.

Updated: June 26, 2023

Published: May 16, 2023

You may also enjoy

Better Dependency Management in Python

6 minute read

Learn how Earthly can simplify dependency management in Python projects, ensuring consistency across different environments and streamlining the build and de...

Earthly used by Phoenix Project

less than 1 minute read

Learn how Earthly is revolutionizing the CI pipeline for the popular Phoenix project, making testing and continuous integration easier than ever. Discover ho...

The world deserves better builds

4 minute read

Learn how Earthly is revolutionizing the build process with its self-contained, reproducible, and parallel approach. Say goodbye to slow, brittle builds and ...

Can We Build Better?

4 minute read

Learn how to solve the problem of reproducible builds with Earthly, an open-source tool that encapsulates your build process in a Docker-like syntax. With Ea...

Better Together - Earthly + Github Actions

15 minute read

Learn how Earthly and Github Actions can work together to improve your Continuous Integration (CI) process. Discover the benefits of Earthly's local CI pipel...

Earthly Switches to Open-source

10 minute read

Earthly, a CI/CD framework, has announced that it is switching to an open-source license, allowing for greater community involvement and integration with var...

Earthly CI: Launching a new era for CI

10 minute read

Earthly CI is revolutionizing the world of CI/CD with its fast and efficient platform. Say goodbye to slow builds, complex pipelines, and outdated approaches...

G

28

Le
er

Products

Earthly
Earthly Cloud
Earthly Satellites
Check Status

Content

Blog
Newsletter
Videos & Webinars

Resources

Docs
Pricing
Customer Stories
Solutions
FAQ
Newsroom
Download

Made with 💙 on Planet Earth | We're **hiring**!