



CLOUD: GET



START



CLOUD ROLES



AWS (NO



CODING FOR



BOOKS &

STARTED

LEARNING AWS

EXPLAINED

EXPERIENCE)

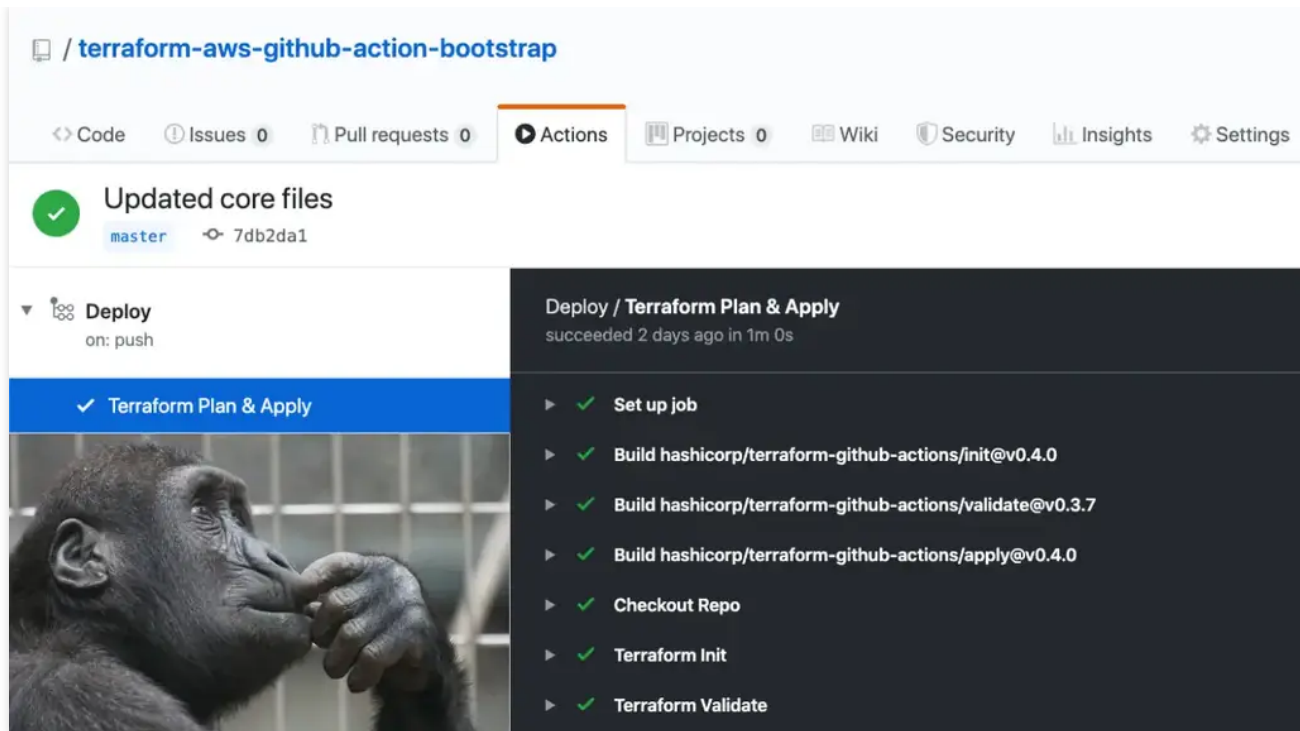
CLOUD?

COURSES

THE ULTIMATE TERRAFORM WORKFLOW: SETUP TERRAFORM (AND REMOTE STATE) WITH GITHUB ACTIONS

Running Terraform in a CI Server can be incredibly useful when you're trying to automate or experiment with cloud resources. One of the easiest, cheapest and most accessible setups I've found is using Github Actions and S3 for state.

◀ ◻ ▶



The screenshot shows the GitHub repository page for `terraform-aws-github-action-bootstrap`. The repository is on the `master` branch with commit `7db2da1`. A green checkmark indicates that the core files have been updated. The `Actions` tab is selected, showing a workflow named `Deploy / Terraform Plan & Apply` that succeeded 2 days ago in 1m 0s. The workflow steps are listed as follows:

- ▶ **Set up job**
- ▶ **Build hashicorp/terraform-github-actions/init@v0.4.0**
- ▶ **Build hashicorp/terraform-github-actions/validate@v0.3.7**
- ▶ **Build hashicorp/terraform-github-actions/apply@v0.4.0**
- ▶ **Checkout Repo**
- ▶ **Terraform Init**
- ▶ **Terraform Validate**

A small image of a gorilla is visible on the left side of the workflow details.

But learning a new technology can be frustrating especially when the anxiety of: “Am I doing this right?” strikes. In this article I’ll walk you through how to get a Terraform project running in Github Actions from start to finish — with all the details you need to understand what’s happening and why.

By the end of this article you will have a running Terraform project on Github Actions using remote state.

WHAT WE'RE BUILDING TODAY

After writing quite a lot about Terraform I've had repeated requests to have a simple Terraform workflow. So today I'm sharing how to simply setup a workflow using Terraform and Github Actions that I have used personally not just for general experimentation but also learning and real applications.

Note: All the code for today's article is within the following Github repo so be sure to open it up for reference!

Terraform is a great way to learn Cloud, check out this article to understand why: [5 Important Reasons To Learn Terraform Before Cloud Computing](#).

TERRAFORM, AWS & GITHUB ACTIONS — WHY?

But, before we get into the setup, let's quickly recap on what each of these technologies does and why you'd want to use them.

WHY TERRAFORM?



Terraform logo

Terraform is a CLI tool that allows you to create infrastructure declaratively as code. When your infrastructure is written as code you can apply the same quality assurance processes like code review, and build pipelines.

One of the main reasons I really like Terraform is that it's cloud platform agnostic, you can use it with AWS, GCP, etc. That means once you understand Terraform you don't need to re-learn your infrastructure as code tooling if you want to use another cloud service.

Need a refresher on Infrastructure As Code? Check out: [Infrastructure As Code: A Quick And Simple Explanation](#). Or for detail on declarative infra as code? Check out: [Declarative vs. Imperative Infrastructure As Code](#)

WHY AWS?



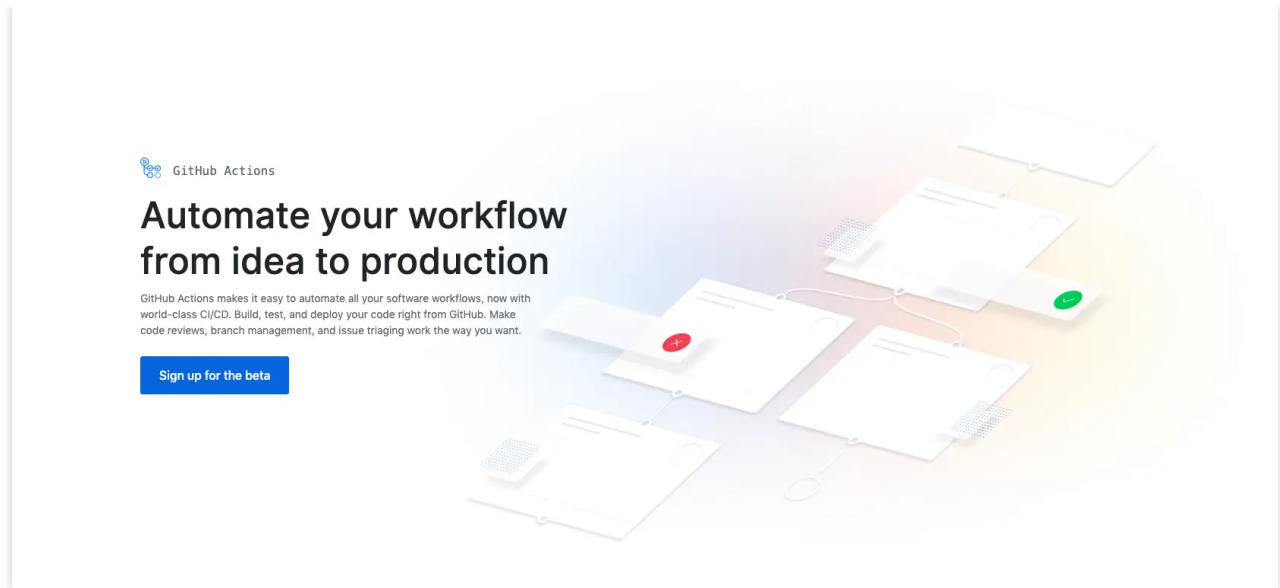
And then the question is: why AWS? Well there's two reasons: one is because... it's AWS (more on this later), and also simply because I needed to integrate with something in order to create the demo!

You can infact re-use today's CI setup with all other Cloud providers too. We'll cover how you can do that in detail at the end of the article if you're interested.

But, the reason I picked AWS is because it's most likely the cloud provider that you want to learn, or already use. Not only that, but [AWS is the market leader](#) and has [the most mature Terraform community](#).

New to AWS? Check out: [Where \(And How\) to Start Learning AWS as a Beginner](#)

WHY GITHUB ACTIONS?



GithubActionsBeta

Lastly, the question is why Github Actions? Firstly it's because Github Actions integrates with Github without needing to setup any other accounts, that's less work for you and makes the setup quicker.

Not only is Github Actions quick and easy to setup, but importantly Github Actions is free (for low usage), which means that it's likely to have the widest accessibility for anyone wanting to setup this workflow.

But aside from those two reasons Github Actions is still a solid CI tool and it has some nice features such as being container based (but that's a topic for another day!

HOW THE ARTICLE IS STRUCTURED

Okay that's enough introduction about what we're going to do today, let's actually go ahead and jump into the thick of it. We've got a lot to cover, so in order to break everything down we'll do it in two parts...

In part one we're going to setup a Terraform back-end with S3. We need our backend in order to run Terraform in CI (don't worry, we'll discuss why in just a moment). Then, in part two we're going to go ahead and put our Terraform pipeline in Github actions and use our previously created remote state

Before you embark, I must give you a word of warning: setting up Terraform with CI can be quite confusing. There's no simple way to do it (yet) so we have to do a few (seemingly convoluted) steps to get everything working. But stick with the article to the end as all will be revealed along the way.

The process of setting up Terraform with CI and remote state can be pretty fiddly but I'll do my best to talk you through what we're doing and why at each step so you have a good understanding of what's going. Trust the process!

Sound good? Let's get to work!

PART 1: CREATE THE BACKEND

So the first thing we'll need to do is create our Terraform backend. But before we can do that we need to understand what a backend is. so let me explain...

WHAT IS A TERRAFORM BACKEND?

Terraform works by comparing existing resources in your cloud provider (using the cloud API's) against a stored state file. The state file is a representation of the resources that you manage with Terraform.

Think of state like an inventory list. When you take something out of your inventory you remove it from your list. And by having a list you can quickly check what's in your inventory without needing to manually look through it.

The state file file is the inventory. And it's actually a basic JSON file. Whenever you run Terraform it updates your state file, which is by default stored on the machine running it. But if we want to collaborate with others or use Terraform in a CI tool storing the file locally won't work.

In that case we'll need to store the state in a "remote" location, which is where the idea of remote state comes in. And because we're using AWS, we can use S3 as our remote state location.

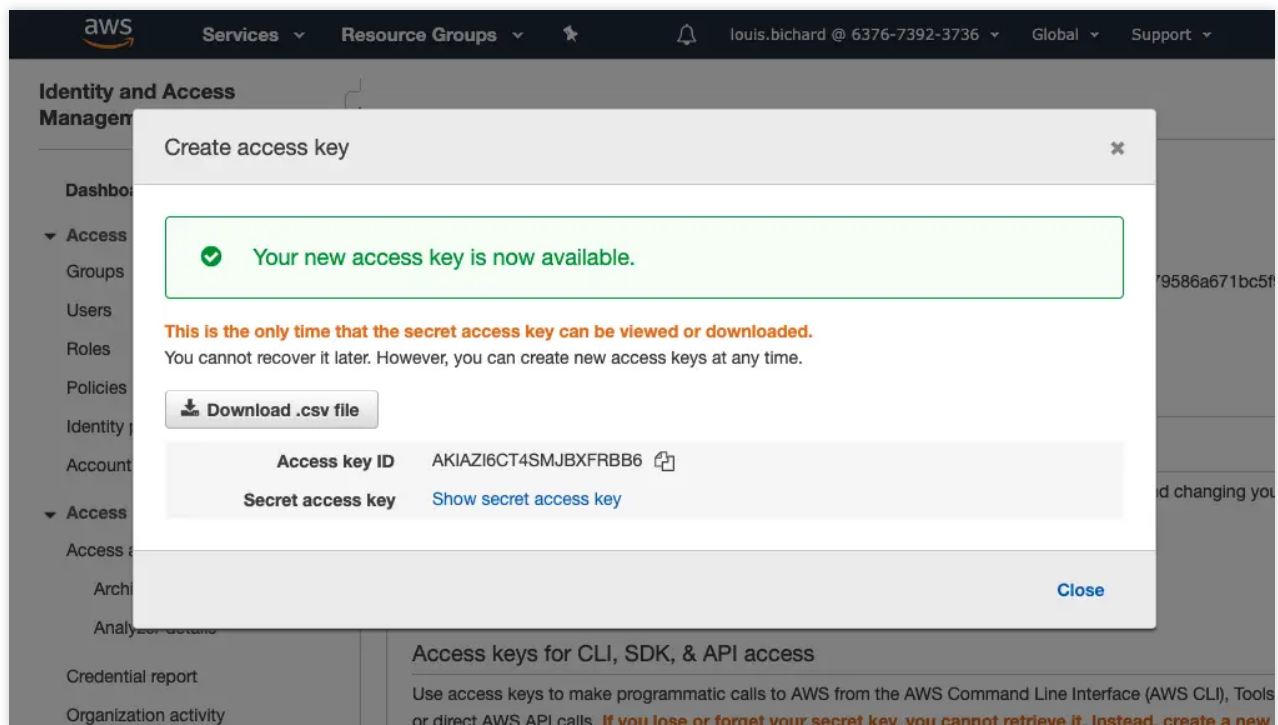
So the first thing we'll need to do in order to get our Terraform setup running in Github Actions CI is to setup our remote state bucket.

STEP 1: CLONE THE REPO

First up you'll need to clone the repo. The repo contains a minimal Terraform setup and some Github Action workflows that execute a Terraform plan on pull requests to master, and deploy on merge to master.

```
git clone git@github.com:loujaybee/terraform-aws-github-action-bootstrap.git
```

STEP 2: LOCALLY SET YOUR AWS ACCESS KEYS



Create AWS Access Key

Now, in order to access AWS we'll need to get access to our AWS account. The way we'll do

You'll need to download those from your AWS account security credentials page and set them on your local environment.

In bash you can do it like this...

```
set AWS_ACCESS_KEY_ID=YOUR_ACCESS_KEY
```

```
set AWS_SECRET_ACCESS_KEY=YOUR_SECRET_KEY
```

Important note: With your access keys an attacker can do anything your user can do with your AWS account. So be as vigilant with them as you would be with the password to your online bank account.

If you're a little uncertain about what AWS keys are and how they work, be sure to check out this article: [AWS access keys — 5 Tips To Safely Use Them](#).

STEP 3: INSTALL TERRAFORM

```
~/P/loujaybee [~] terraform -v
Terraform v0.12.9


Your version of Terraform is out of date! The latest version
is 0.12.20. You can update by downloading from www.terraform.io/downloads.html
~/P/loujaybee [~] █
```

Terraform Version

Next up in order to run Terraform you'll need to install Terraform locally. Go ahead and [follow the instructions on their website](#) for that. Now when you run...

```
terraform -v
```

STEP 4: RUN TERRAFORM INIT

```
~/P/L/terraform-aws-github-action-bootstrap  terraform init

Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "aws" (hashicorp/aws) 2.48.0...
```

Execute Terraform Init

Now with Terraform installed, go into the root of the previously cloned bucket and run...

```
terraform init
```

STEP 5: SET YOUR BUCKET NAME

```
loujaybee > terraform-aws-github-action-bootstrap > main.tf > ...
1  provider "aws" {
2    |   version = "~> 2.0"
3    |   region  = "eu-central-1"
4  }
5
6  # terraform {
7  #   backend "s3" {
8  #     bucket = "example-terraform-project-name-bootstrap-terraform-state"
9  #     key    = "default-infrastructure"
10  #     region = "eu-central-1"
11  #   }
12  # }
13
14  0 references
15  resource "aws_s3_bucket" "terraform_state" {
16  |   bucket = "example-terraform-project-name-bootstrap-terraform-state"
17  |
18  |   versioning {
19  |     enabled = true
20  |   }
21  }
```

Terraform Backend Bucket Name

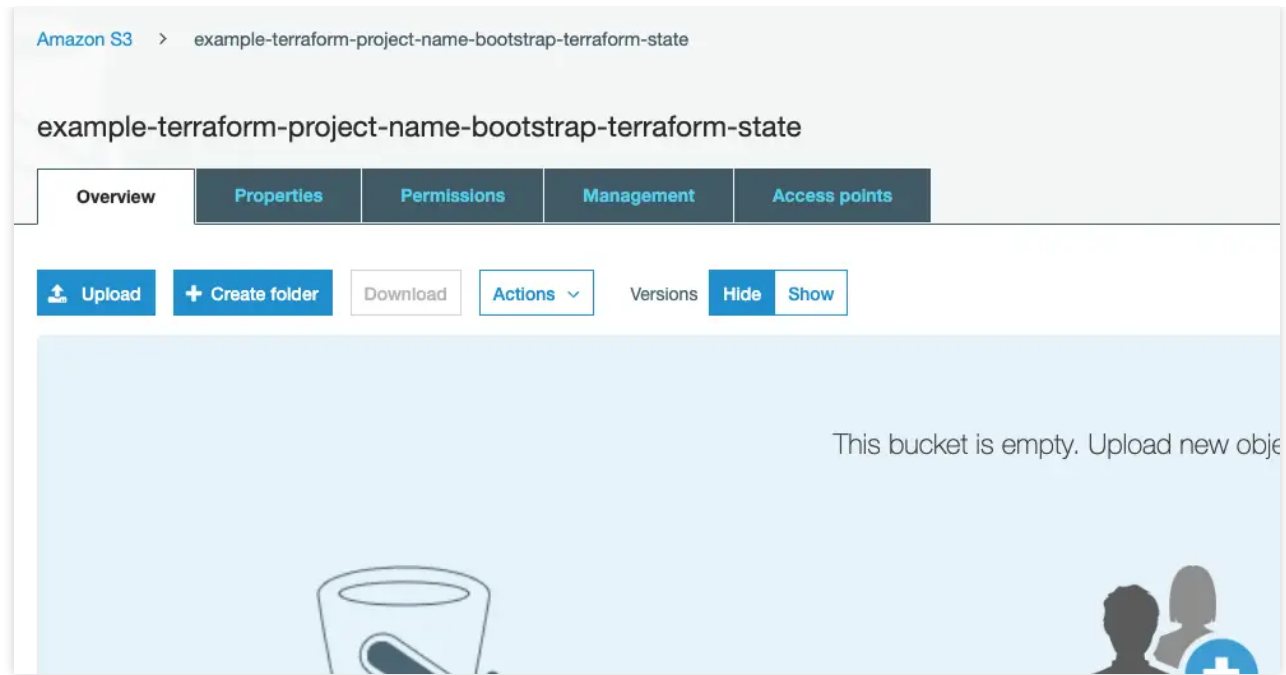
Next we'll need to update the bucket name in the repo. Since bucket names need to globally unique you'll need to update the bucket name.

So go ahead and open up the `main.tf` file and update the two bucket values (yes also the one that's commented out, we'll need it later) to be a name related to your project.

Something like...

`my-demo-terraform-backend` OR `my-home-automation-terraform-backend`

STEP 6: CREATE YOUR BUCKET



Backend Terraform S3 Bucket

And now with Terraform setup, your bucket name configured and your access to AWS setup you should be able to run your first Terraform code!

Run an apply to create your AWS bucket, like this...

```
terraform apply
```

And with that we conclude part one. So far you should have a bucket which we're going to use for our remote state we can now setup our CI pipeline to run terraform remotely rather than on our machine.

But, you've got a problem... You've just created an AWS resource on your local machine and you haven't stored the state file anywhere. We don't want to store the state file in Github as our CI can't commit to Github — so we need to push that state to our newly created bucket.

PART 2: SETUP TERRAFORM ON GITHUB ACTIONS

In this part we'll talk through two main things: moving our previously created local state to our new remote bucket, and then authorising Github Actions to run Terraform.

STEP 7: UNCOMMENT THE BACKEND CONFIGURATION

As we said before, keeping the state file on the machine that's running Terraform won't work for running Terraform in CI.

So to have Terraform working in Github Actions the first thing you'll want to do is uncomment the backend configuration in the `main.tf` file. The [backend configuration block](#) tells Terraform that we want to use AWS and S3 as our backend remote state storage.

STEP 8: TRANSFER YOUR STATE TO REMOTE

```
Do you want to copy existing state to the new backend?  
Pre-existing state was found while migrating the previous "local" backend to the  
newly configured "s3" backend. No existing state was found in the newly  
configured "s3" backend. Do you want to copy this state to the new "s3"  
backend? Enter "yes" to copy and "no" to start with an empty state.  
  
Enter a value: █
```


Now we want to re-run terraform init.

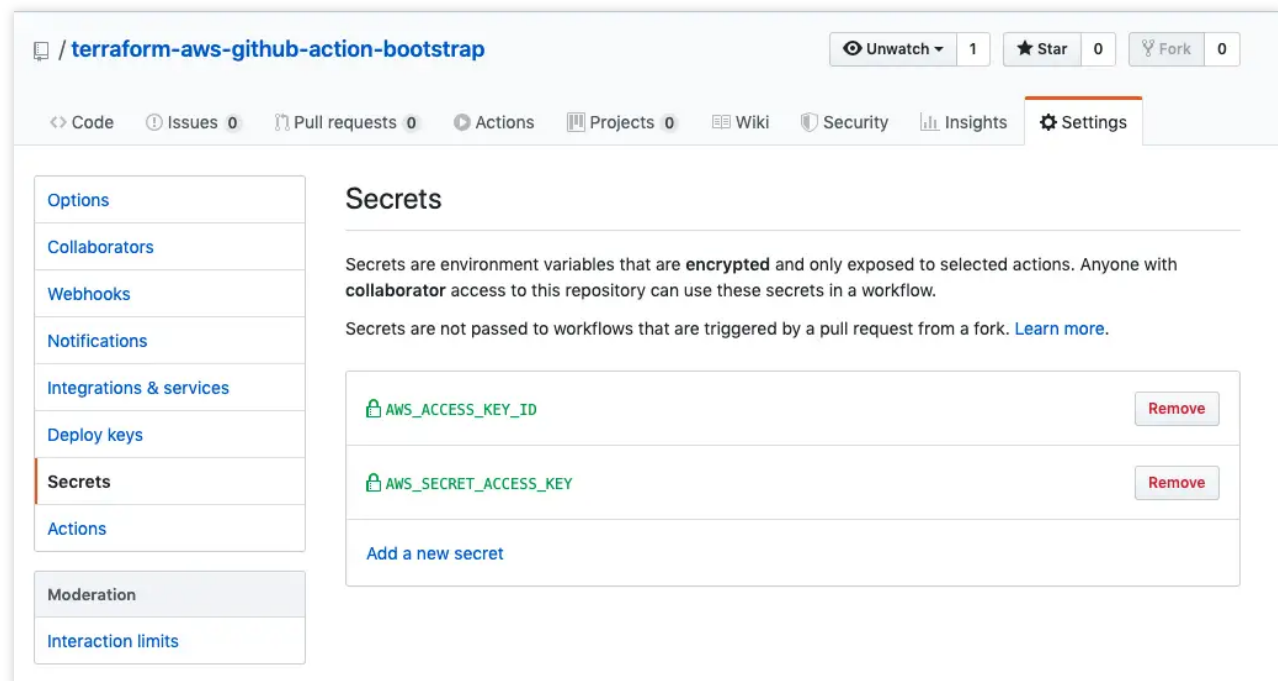
When we re-initialise terraform it's going to notice that our current setup (with a backend) is different to our original setup where we were using local state. And since Terraform is smart, it will ask us if we want to transfer our currently created state into our remote bucket.

Now wait... let's think about what we're doing for just a moment as this is pretty meta. Essentially we're going to move the resource information about our backend S3 into our backend S3. That's pretty weird, why would we want to do that?

Keeping even your backend S3 configuration in your state allows you to ensure that your backend bucket is also managed in Terraform. That's useful if we want to do things like update our bucket versioning, or configure permissions on our bucket, or implement S3 backups etc.

Now that means our backend S3 is setup and configured. We're now only missing one thing: Our CI server in Github Actions needs permissions to AWS so that it can execute our Terraform.

STEP 9: SETUP YOUR GITHUB ACTIONS SECRETS



The first thing you'll need to do is ensure that Github Actions has permissions to act upon your AWS account. So go ahead and add your AWS credentials to the secrets settings of your repo (these settings are configured per repo).

Use the exact same key names as you did on your local machine here and set your `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. That will allow the Terraform AWS plugin to authenticate itself to create your resources.

STEP 10: COMMIT, PUSH AND RUN

The screenshot displays the GitHub interface for the repository `terraform-aws-github-action-bootstrap`. The top navigation bar includes links for Code, Issues (0), Pull requests (0), Actions, Projects (0), Wiki, Security, Insights, and Settings. Below the navigation bar, a green checkmark icon and the text "Updated core files" are visible, along with the branch name `master` and commit hash `7db2da1`.

The left sidebar shows the "Deploy" workflow, triggered by "on: push", with a sub-item "Terraform Plan & Apply" marked with a green checkmark. The main content area displays the details of the "Deploy / Terraform Plan & Apply" action, which "succeeded 3 minutes ago in 1m 0s". The action steps are listed as follows:

- ▶ **Set up job**
- ▶ **Build hashicorp/terraform-github-actions/init@v0.4.0**
- ▶ **Build hashicorp/terraform-github-actions/validate@v0.3.7**
- ▶ **Build hashicorp/terraform-github-actions/apply@v0.4.0**
- ▶ **Checkout Repo**
- ▶ **Terraform Init**
- ▶ **Terraform Validate**
- ▶ **Terraform Apply**
- ▶ **Complete job**

Successful Terraform Apply In Github Actions

And now this last step is an easy one! Commit and push your changes (your bucket name, mainly) to your cloned repo, and watch Github Actions complete!

```
git add .
```

 to add all your local changes.

```
git commit -m "My changes"
```

 to commit your changes.

```
git push
```

 to send your changes to Github.

Voila! That's it. Now you have a working Terraform pipeline up and running configured with a Terraform backend.

ADVANCED SETUP (WITH DYNAMODB LOCKS)

If you want to make your setup more advanced, you can [introduce Terraform locks using DynamoDB](#). Locks simply ensure that your Terraform is not executed by two people at the same time.

documentation if you think you need it!

REUSING THE SETUP FOR GCP, AZURE, ETC.



Cloud Vendor Logos

Before I go though, at the start of the article I did promise to mention about how you can re-use the setup for different cloud providers. And it's pretty simple. All you have to do is [add another Terraform configuration](#) into your `main.tf` file (like the AWS one we have) to configure your other accounts.

With a new provider setup all you have to do is add resources from your new provider and Terraform will create them. Terraform doesn't care that your backend is stored in AWS or S3 nor that you're running on Github Actions.

MORE ON TERRAFORM

If you're keen to learn more about Terraform and infrastructure as code I'd highly recommend my free [Terraform Kick Start email course](#). Or alternatively of these articles:

- [What Is Terraform Used For? The 3 Main Use Cases.](#)
- [How Long Does It Take To Learn Terraform? And How To Speed Up Your Learning.](#)

- [Should You Use Typescript To Write Terraform? \(The Terraform CDK\)](#)
- [Terraform: How To Rename \(Instead Of Deleting\) A Resource](#)
- [The Simplest Possible EC2 Web Server Setup Using Terraform \(On AWS\)](#)
- [What is Terraform? A Simple Definition.](#)
- [10 Terraform Best Practices: For Secure & Fast Infrastructure.](#)
- [Should You Commit the Terraform .tfstate File to Git?](#)
- [The Ultimate Terraform Workflow: Setup Terraform \(And Remote State\) With Github Actions](#)
- [What Is the Best Way to Learn Terraform?](#)

BECOME A TERRAFORM GURU!

And that is everything we'll cover today. I really hope that helped you to grasp the different parts you'll need for setting up Terraform and Github Actions.

I truly believe it's a great setup and it's very useful when learning both Terraform and AWS. Now you can create, destroy and update cloud resources simply, quickly and repeatably.

And that's everything. You should now have a nice setup for running Terraform in Github Actions with repeatability, traceability and better ease of use. Happy experimentation!

Speak soon Cloud Native Friends.

Lou Bichard



Hey I'm Lou! I'm a Cloud Software Engineer From London. I created Open Up The Cloud to help people get their start and grow their careers in cloud. I edit the monthly [Open Up The Cloud Newsletter](#) which I think you'd really like. You can find me on [Twitter](#) or [LinkedIn](#) , I'm always happy to chat. When I'm not writing you can usually find me picking up heavy things and putting them down, or riding two wheels

ALSO ON OPEN UP THE CLOUD

	2636 » Open Up The Cloud	cloud-computing-start » Open Up The Cloud	» Open Up TI
7 months ago ©2017 Lou Bichard - All Opinions Are My Own And Are Not Representative Of ...	7 months ago ©2017 Lou Bichard - All Opinions Are My Own And Are Not Representative Of ...	3 months ago ©2017 Lou Bichard - All Opinions Are My Own And Are Not Representative Of ...	7 months ago ©2017 Lou Bic Opinions Are N Are Not Repres

Sponsored

Do This Before Bed and Watch Your Belly Shrink

X-Loss Control

The Best Trick For Treating ED (See How)

Erectile Dysfunction | Search Ads

Top 15 At-Home Exercises to Lose Weight and Build Muscle

Lintmit.com

Remember Her From American Pie This Is Her Now

House Jogger

Farmer Stumbles Upon Egg On His Farmland - When Expert Sees It, He Turns Pale


Tips and Tricks


Movie stars who turned broke & some of them now work ordinary jobs.


The Family Breeze


What do you think?


5 Responses



Upvote


Funny


Love


Surprised


Angry


Sad

1 Comment

 Login ▼

G

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name



Share

Best Newest Oldest



Will
a year ago



It's missing a big step part of the process: the git actions terraform configuration! I found a

Sponsored

Do This Before Bed and Watch Your Belly Shrink

X-Loss Control

The Best Trick For Treating ED (See How)

Erectile Dysfunction | Search Ads

Top 15 At-Home Exercises to Lose Weight and Build Muscle

Lintmit.com

Remember Her From American Pie This Is Her Now

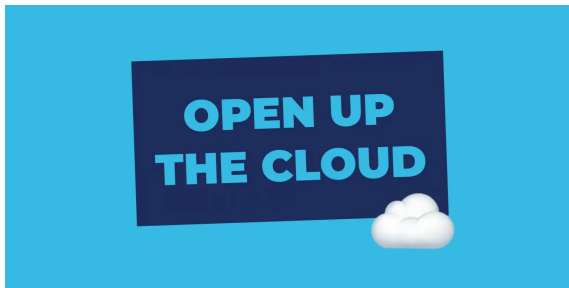
House Jogger

Farmer Stumbles Upon Egg On His Farmland - When Expert Sees It, He Turns Pale

Tips and Tricks

Abuja: Click Here To See The Price Of Solar Panels

Solar Panels | Search Ads



Hey! 🙌 Welcome to Open Up The Cloud, a social-enterprise dedicated to helping you get your start and grow your career in cloud (read [the mission](#)).

If you're new around here I recommend you check out the [start here page](#) as the best starting point.

... ..

New articles come out constantly on the blog. But, you can also find Open Up The Cloud on [YouTube](#), [Instagram](#), [Twitter](#) for more cloud content!

Speak soon, [Lou](#).

NEWSLETTER

[Open Up The Cloud Newsletter #30 \(January Recap 2022\)](#)

[Open Up The Cloud Newsletter #29 \(November Recap 2021\)](#)

[Open Up The Cloud Newsletter #28 \(October Recap 2021\)](#)

[In Serverless, Who Sets Up The Environment? What You Do & Don't Have Access To](#)

[How To Test AWS Lambda: Everything You Need To Get Started.](#)

[How To Debug AWS Lambda: A Detailed Overview](#)

AWS

[Which Is The Best AWS Region For Your Location? Based on Cost & Latency](#)

[Which AWS Region Is Cheapest? A Costing Report](#)

[How To Find Which AWS Region Is Closest To You?](#)

INFRA AS CODE

[What Is Terraform Used For? The 3 Main Use Cases.](#)

[How Long Does It Take To Learn Terraform? And How To Speed Up Your Learning.](#)

[Should You Use Typescript To Write Terraform? \(The Terraform CDK\)](#)

LEGAL INFORMATION

We are a participant in the Amazon Services LLC Associates Program, an affiliate advertising program designed to provide a means for us to earn fees by linking to [Amazon.com](https://www.amazon.com) and affiliated sites. We also participate in programs from other sites. Open Up The Cloud is compensated for referring traffic and business to these companies.

Sponsored Content



Belly Fat Removal Without Surgery in Abuja: the Price Might Surprise You

Liposuction | Search Ads



Bathroom Remodelling Trends in 2023 Might Totally Surprise You

Bathroom Remodelling | Search Ads



More People Switching To VoIP Phones (Take A Look At The Prices)

VOIP Phones | Search Ads



Do You Speak English? Work For A USA Company, Live In Abuja

Online Jobs | Search Ads



Abuja: Smart Beds Clearance Sale: Prices In Mexico Might Surprise You

Smart Beds In Mexico | Search Ads



Unsold Emergency Generators In Abuja (See Prices)

Emergency Generators | Search Ads

Recommended by

OPEN UP THE CLOUD

[About](#)
[Mission](#)
[Start Here](#)
[Recommended Books & Courses](#)
[Open Up The Cloud Newsletter](#)

ULTIMATE GUIDES

[Containers](#)
[Serverless](#)
[Infrastructure As Code](#)
[Observability & Monitoring](#)
[Amazon Web Services \(AWS\)](#)

SOCIAL (NOT JUST A BLOG!)

[LinkedIn](#)
[Twitter](#)
[YouTube](#)
[Instagram](#)
[Facebook](#)
[DEV.to](#)

CATEGORIES

[AWS](#)
[Careers](#)
[Certifications](#)
[Containers](#)
[Infrastructure As Code](#)
[Meta](#)
[Newsletter](#)
[NodeJS](#)
[Observability & Monitoring](#)
[Reader Questions](#)
[Serverless](#)
[Tech Book Summary](#)
[Uncategorized](#)