Infrastructure as code - from Terraform to AWS

Terraform seems to be the go to tool these days for anything infrastructure as code, but is it possible to become an expert quickly? — by Craig Godden-Payne@beardy.digital

Craig Godden-Payne
May 2 · 6 min read ★

Photo by Michal Pech on Unsplash

> *Terraform enables you to safely and predictably create, change, and improve infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned*

If you've worked with CloudFormation before, it is a similar concept as in it provisions infrastructure, but it has much greater power, as it can be run against multiple cloud providers, such as Amazon Web Services, Microsoft Azure and Google Cloud Platform.

Photo by Pero Kalimero on Unsplash

I find the best way to learn a new tool, or to improve is to get some hands on experience.

This post hopes to guide you through a basic scenario, of setting up some infrastructure using terraform in the AWS cloud platform.

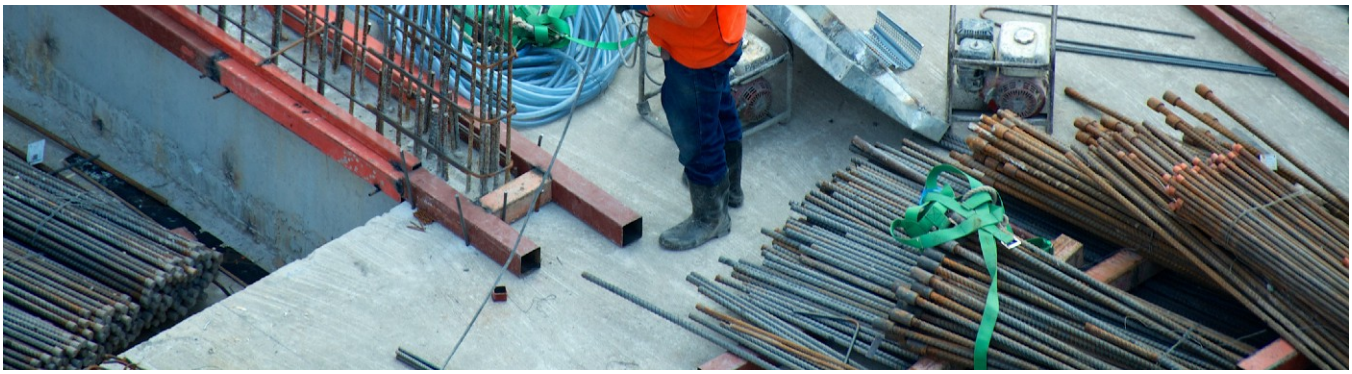


Photo by Etienne Girardet on Unsplash

. . .

Prerequisites

Download terraform. It's pretty simple on any operating system, and usually just involves downloading the binaries and adding them to the path of your shell.

Download Terraform - Terraform by HashiCorp

Download Terraform

- Terraform by HashiCorp Download Terraformwww.terraform.io

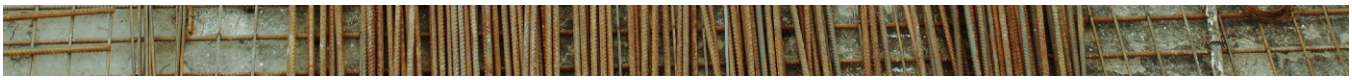


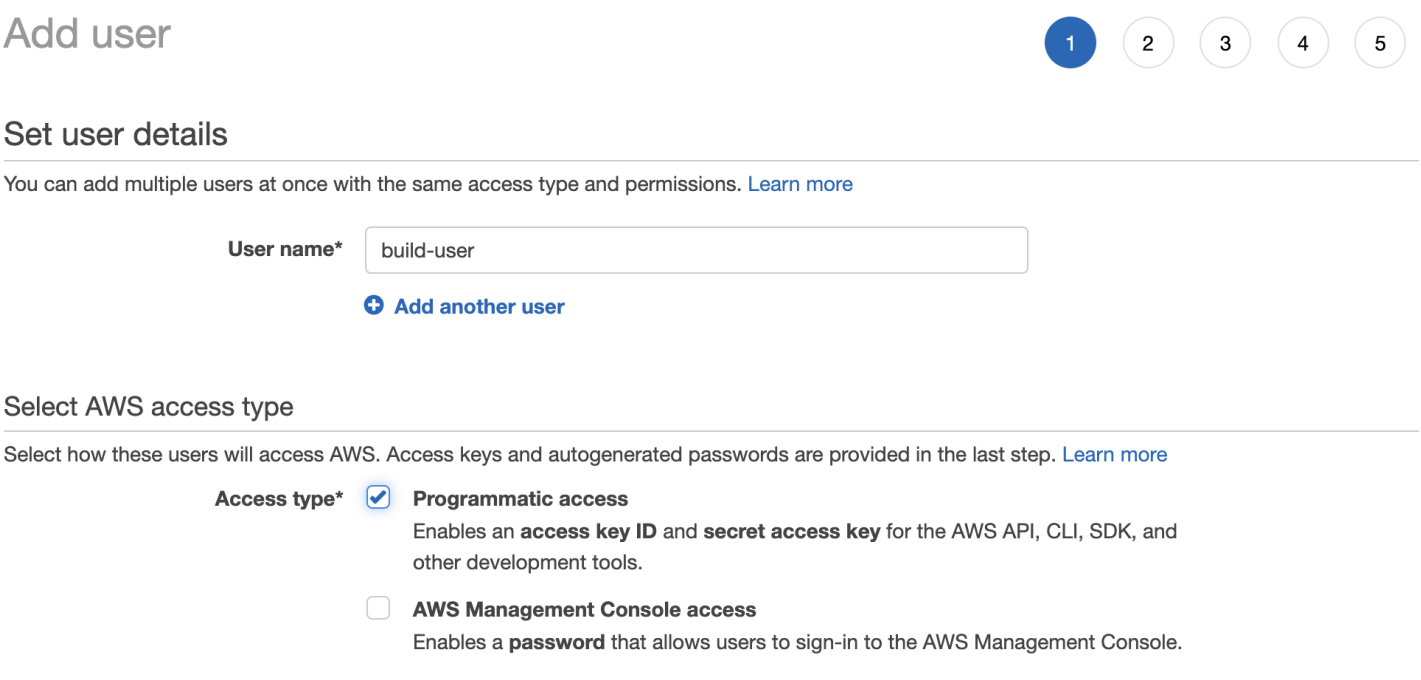
Photo by Guilherme Cunha on Unsplash

. . .

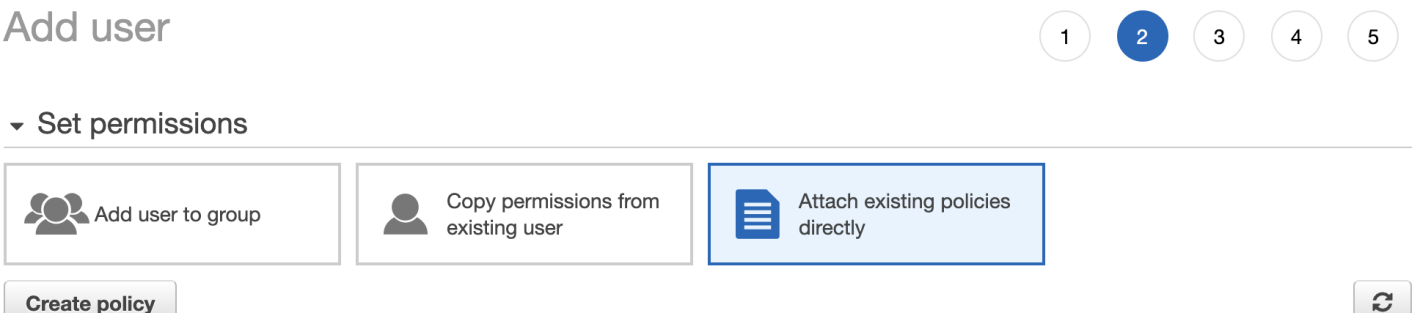
Creating a user in AWS

In order to use terraform to provision infrastructure within AWS, you will need a way for terraform to authenticate with AWS. To do this, we are going to create a user in AWS, and use access key id and secret access key to authenticate

- Create a user, give it a sensible name (such as build-user) and make it an admin. In a real life scenario, it would be better to give this user the least amount of privilege, but for a demo, admin is fine.
- Generate some credentials, you will use these credentials for authentication.



Create a build user



Filter policies | Search | Showing 526 results

	Policy name	Type	Used as
☑	AdministratorAccess	Job function	Permissions policy (1)
☐	AlexaForBusinessDeviceSetup	AWS managed	None
☐	AlexaForBusinessFullAccess	AWS managed	None
☐	AlexaForBusinessGatewayExecution	AWS managed	None
☐	AlexaForBusinessPolyDelegatedAccessPolicy	AWS managed	None
☐	AlexaForBusinessReadOnlyAccess	AWS managed	None
☐	AmazonAPIGatewayAdministrator	AWS managed	None
☐	AmazonAPIGatewayInvokeFullAccess	AWS managed	None

Cancel | Previous | Next: Tags

Add Administrator Access

Add user

1 2 3 4 5

Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

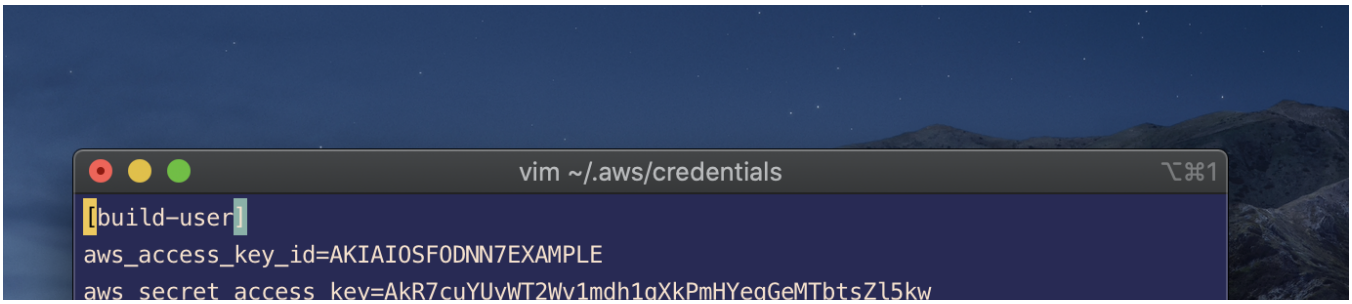
Users with AWS Management Console access can sign-in at: https://beardy-digital.signin.aws.amazon.com/console

Download .csv

	User	Access key ID	Secret access key
▸	build-user	AKIAX7VLMYDF4J5JPBET	AkR7cuYUyWT2Wv1mdh1q XkPmHYegGeMTbtsZl5kw Hide

Generating credentials

- Create and open a file located at ~/.aws/credentials and add the required credentials



- In order to use these credentials with terraform, make sure to set the AWS_PROFILE environment to the correct profile. You can do this like so:
- AWS_PROFILE=build-user terraform plan

Photo by Jonny Caspari on Unsplash

. . .

Working with Terraform

Terraform is not fussy with project layout, it will interpret any files within the directory it uses, so it is up to you how you setup your project.

What I find is good is grouping infrastructure over multiple files, logically by the type of infra. There are no hard or fast rules about this, but I try to use common sense. It is not uncommon for me to have a project like this:

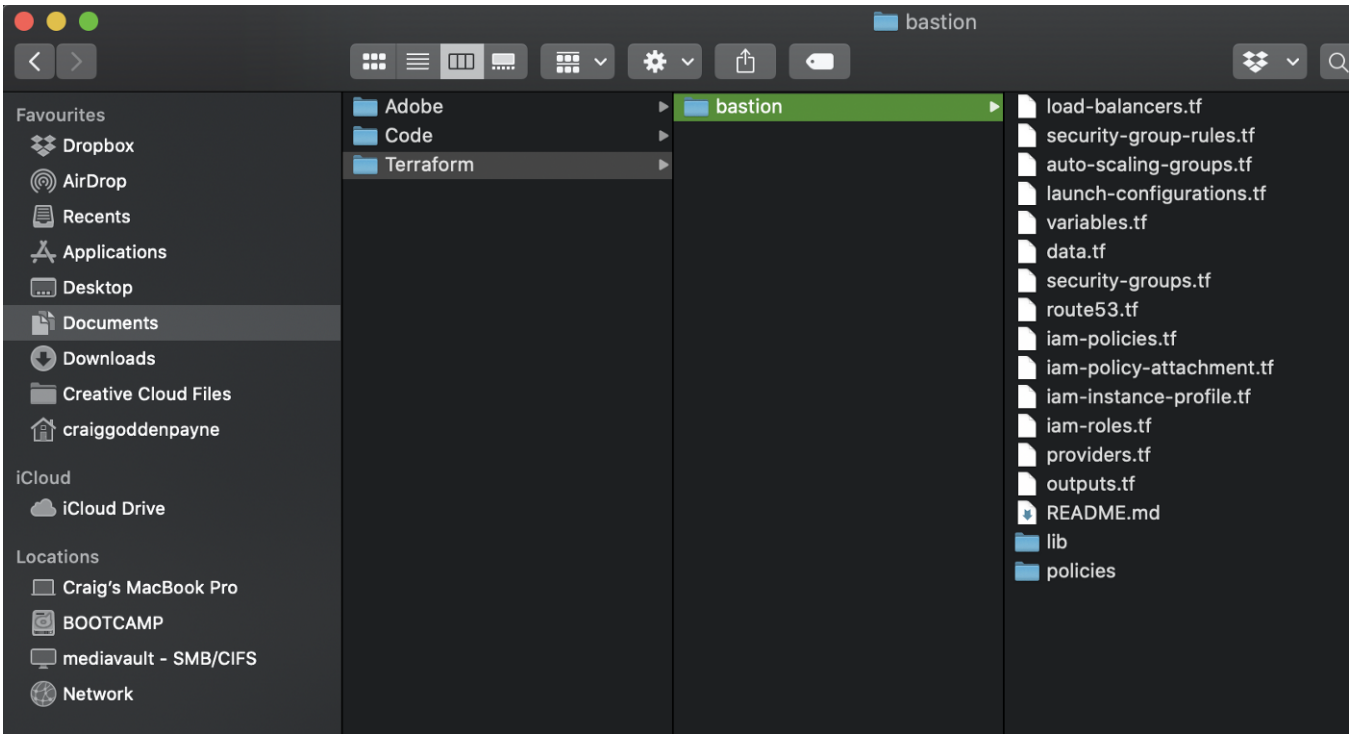


Photo by EJ Yao on Unsplash

. . .

Working with State

One important feature to understand about terraform is the terraform state. Without a state, terraform has a hard job of tracking what changes between deployments.

There are different backends for storing the state, but its common to use something like S3 as it is relatively cheap, and can have versioning enabled, so you are able to look back at previous versions if you need to.

You can also easily add dynamo db as a backing store for a lock table, which has the benefits of not allowing multiple users running the same terraform deployment at the same time.

```
terraform {
  backend "s3" {
    bucket         = "my-terraform-state-bucket"
    key            = "my-application/state/terraform.tfstate"
    region         = "eu-west-2"
    dynamodb_table = "my-terraform-lock-table"
    encrypt        = true
  }
}
```

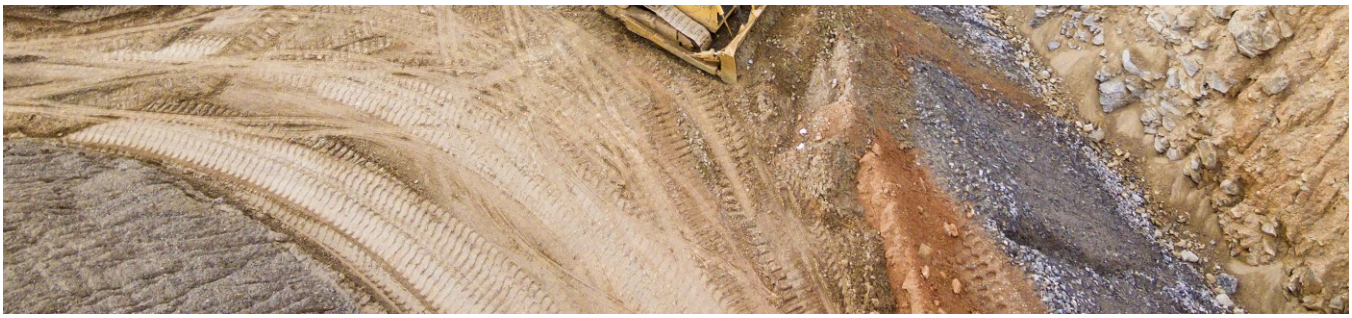


Photo by Shane McLendon on Unsplash

Lets set something simple up

As mentioned above, let's build something in terraform. Now we have the user created to be able to authenticate with AWS we will start with building a basic network.

I tend to use Jetbrains IDEs as I find the terraform support probably one of the best out of the IDEs I have used.

Create a file, lets call it providers.tf and let terraform know which region you want to deploy to:

```
provider aws {
  region = "eu-west-2"
}
```

Now create another file, let's call it vpc.tf, and inside let's create a VPC.

```
resource "aws_vpc" "vpc" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_support   = "true"
  enable_dns_hostnames = "true"
}
```

This example will create a VPC with IP range of 10.0.0.0 — 10.0.255.255, a total of 65536 (hopefully enough!!)

Photo by Jonny Caspari on Unsplash

. . .

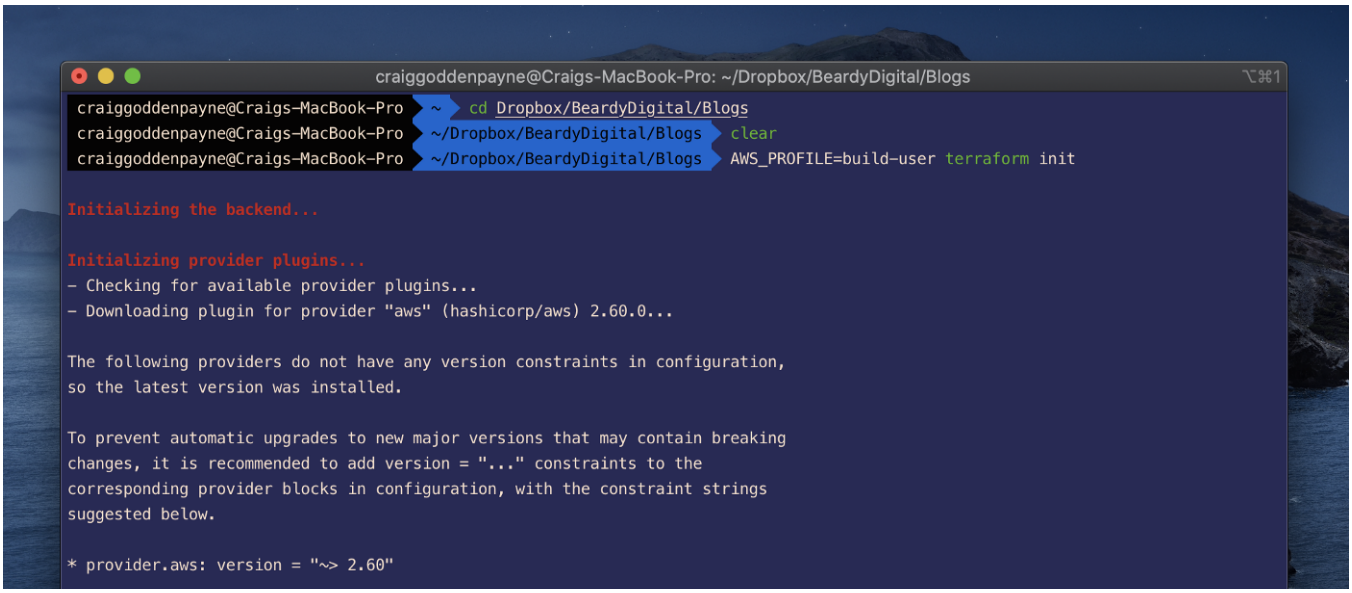
Let's see what will be deployed

So now we have the credentials called build-user setup and terraform installed, all we need to do is in the console, navigate to the code directory with the vpc.tf code.

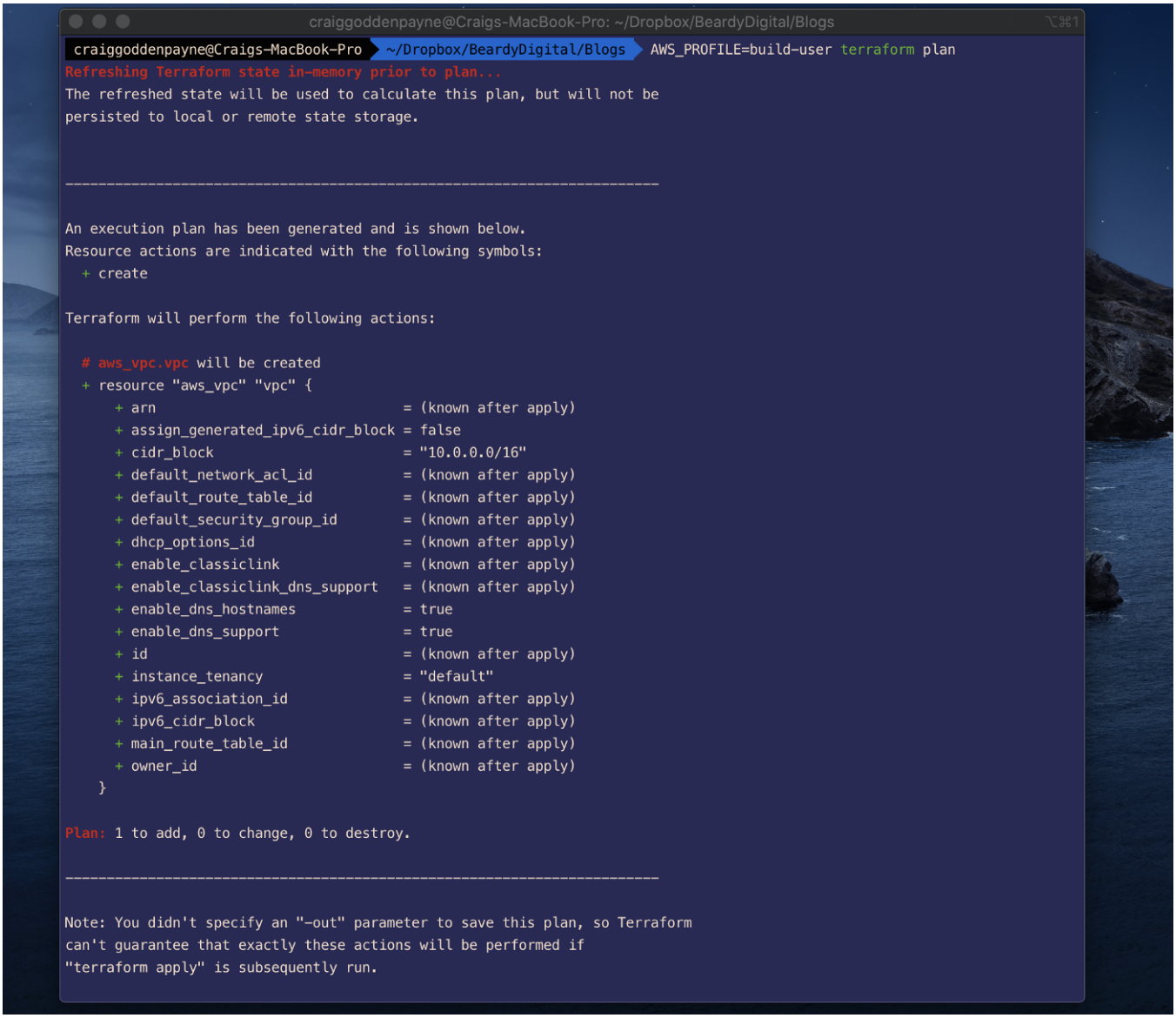
Run the command

```
AWS_PROFILE=build-user terraform init

AWS_PROFILE=build-user terraform plan
```



terraform init



terraform plan

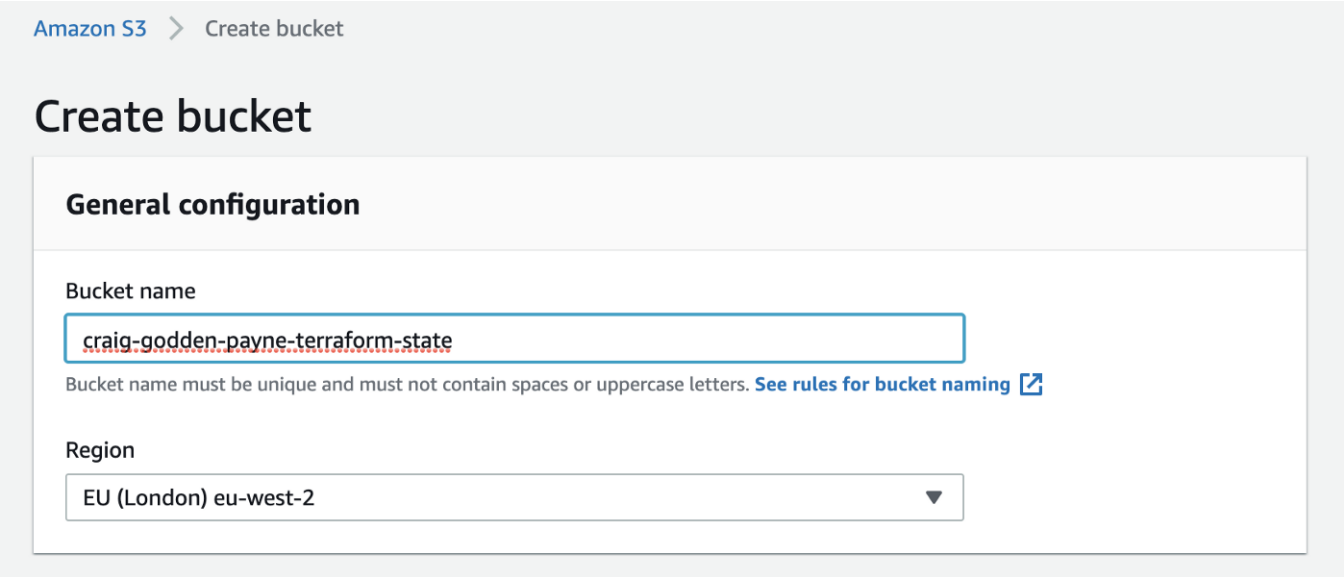
Cool, so looks like a VPC will be created when we apply, but without setting up a remote state, we will get in a world of pain, as terraform will not know the current state of what is in AWS.

We can setup the S3 state as mentioned earlier.

. . .

Setting up the state

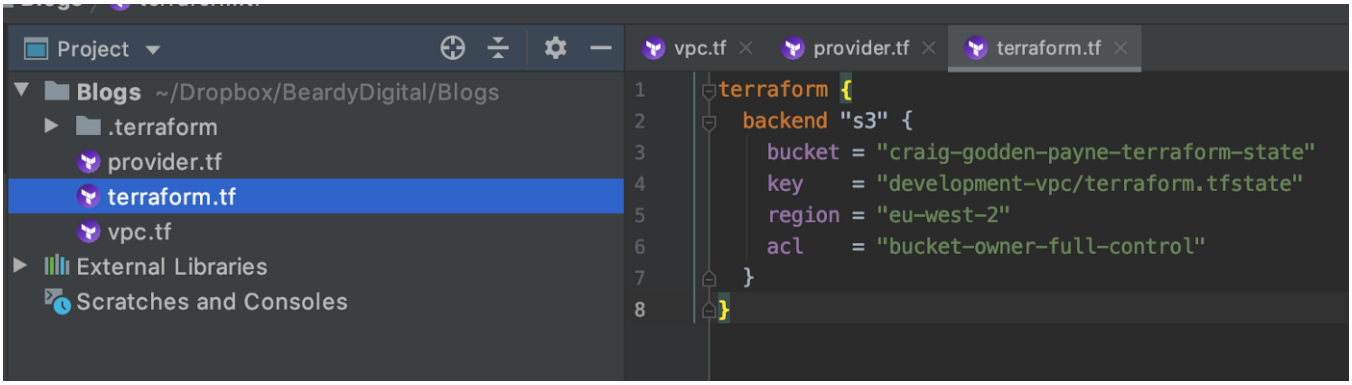
In the console, create an S3 bucket for storing terraform state files



Create an S3 bucket

Add a file called terraform.tf and populate the content with the following, substituting your own bucket name:

```
terraform {
  backend "s3" {
    bucket = "craig-godden-payne-terraform-state"
    key    = "development-vpc/terraform.tfstate"
    region = "eu-west-2"
    acl    = "bucket-owner-full-control"
  }
}
```



. . .

Awesome, lets deploy it

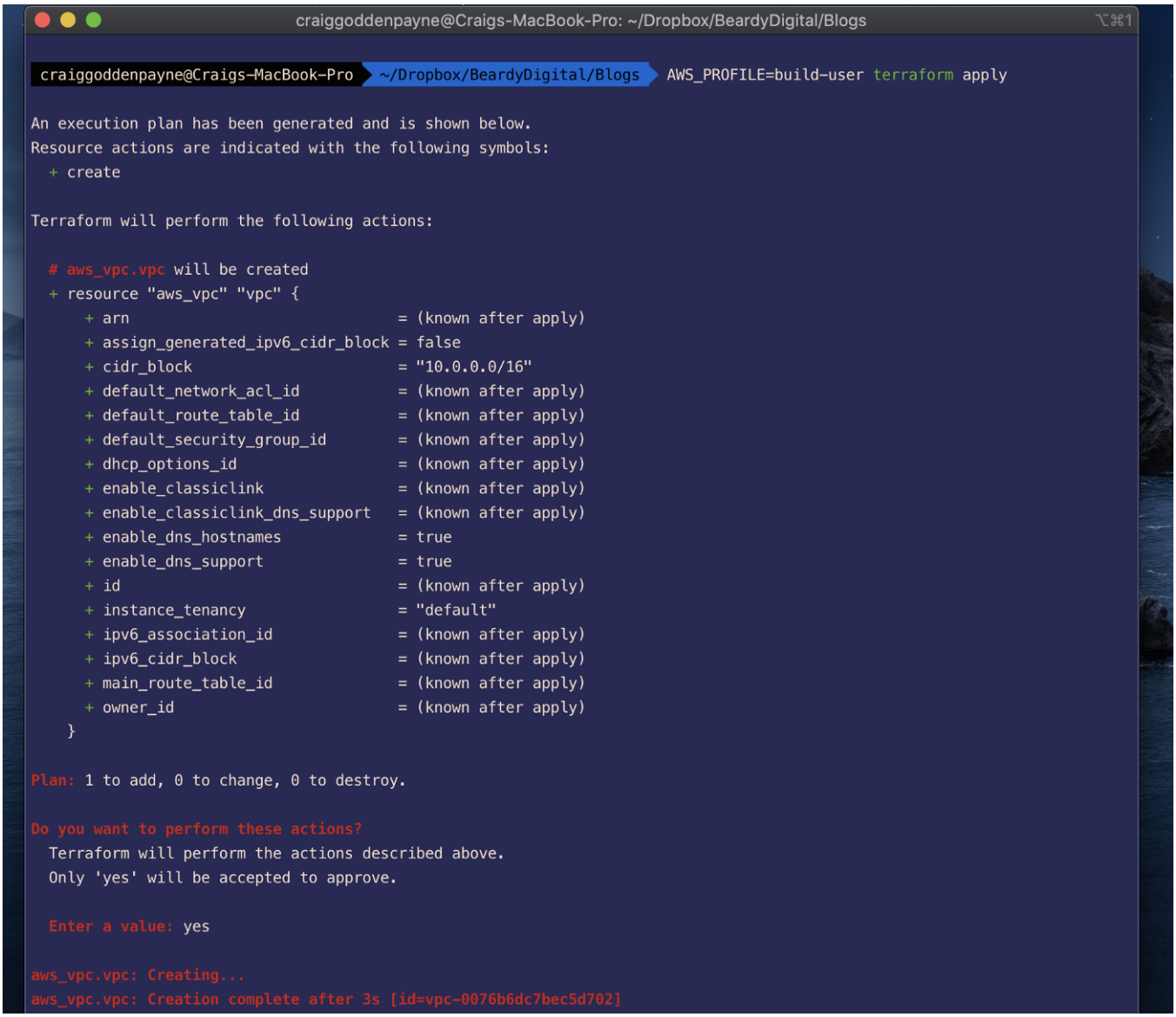
Now we have the state managed in S3, run the plan again to make sure that everything is as expected

Now, lets build some infra!

Run the command:

```
AWS_PROFILE=build-user terraform apply
```

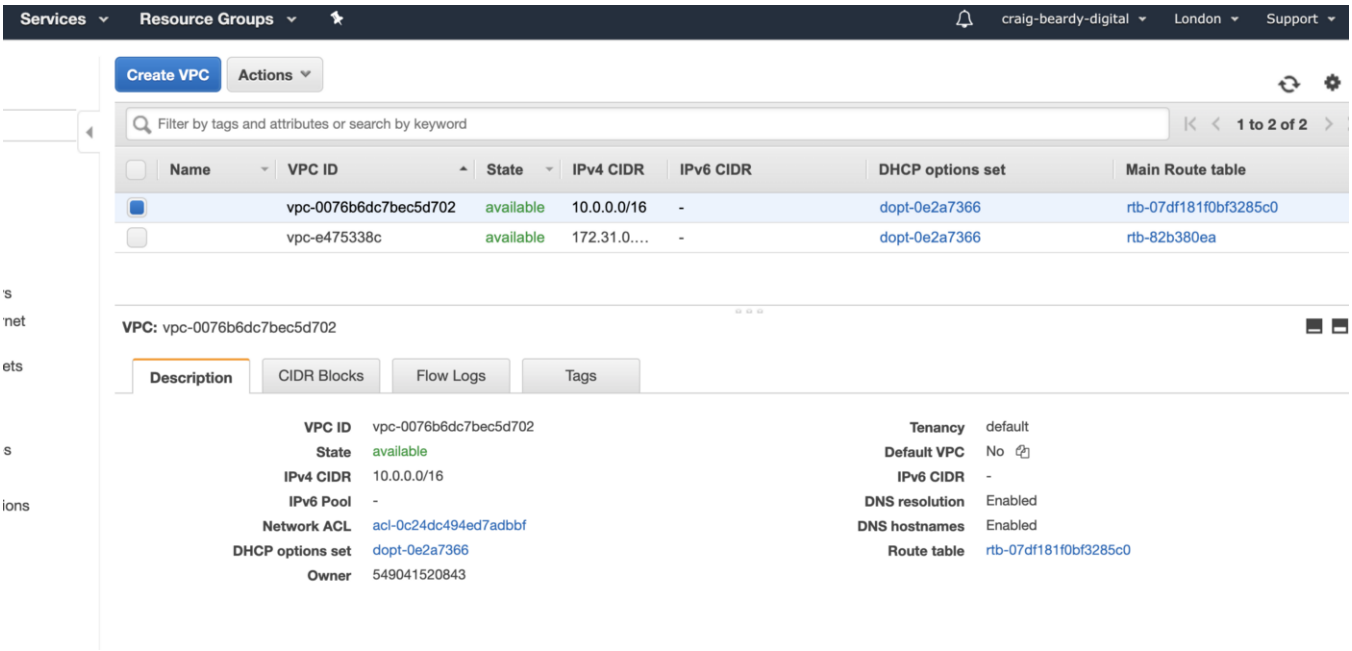
Before the infra is deployed, terraform will ask you if you want to make changes, so make sure to answer `yes` to this question.



```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
craiggoddenpayne@Craigs-MacBook-Pro ~/Dropbox/BeardyDigital/Blogs
```

You just deployed your first VPC

Have a look in the console:



The vpc we just created

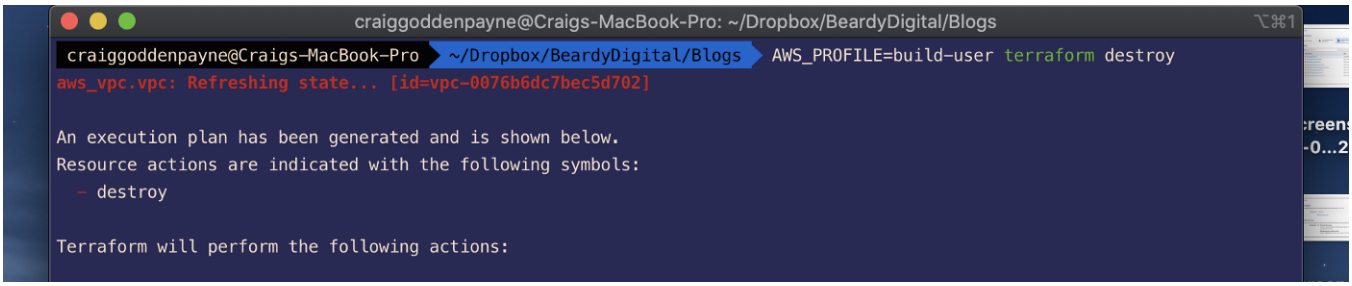
. . .

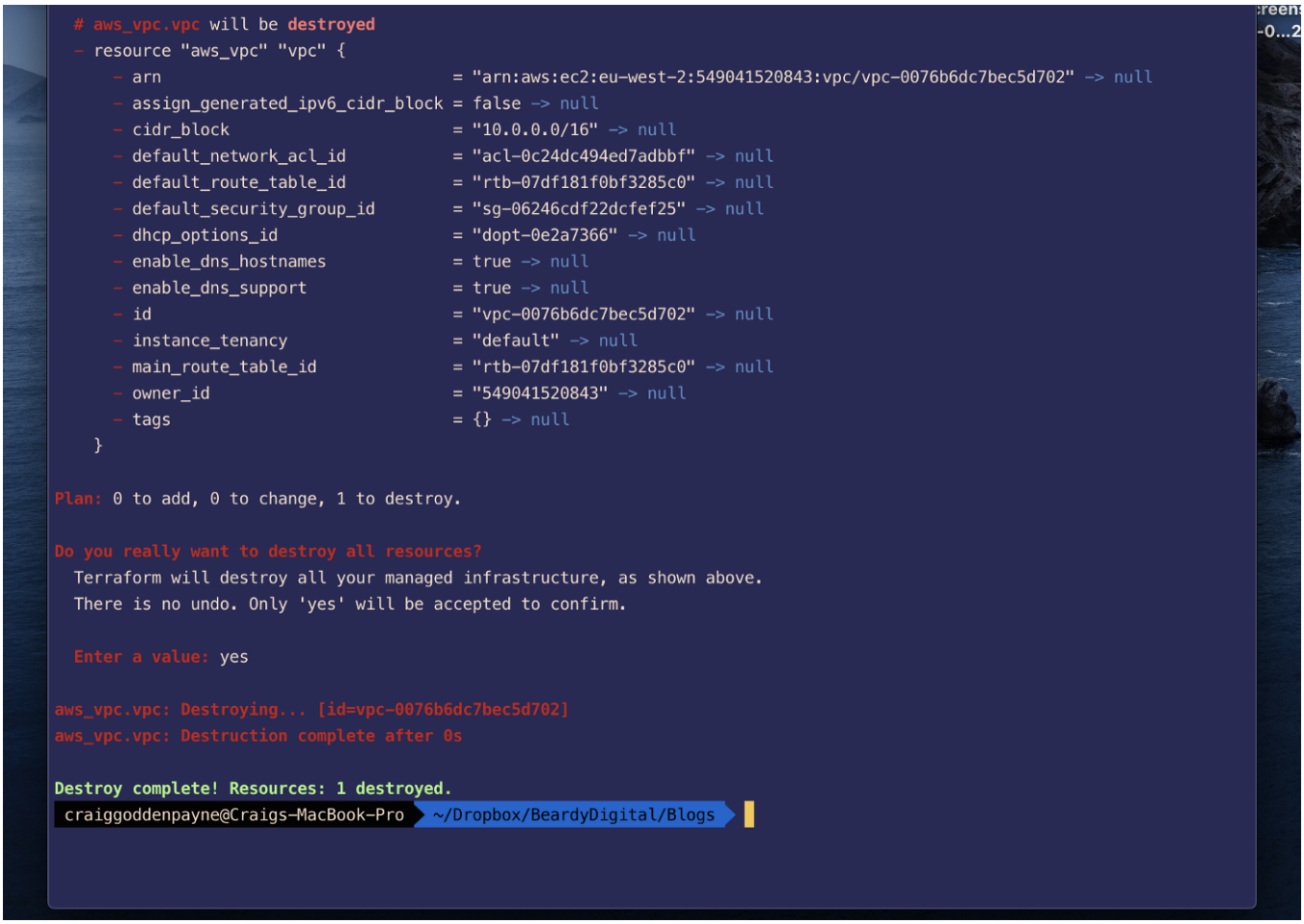
Lets tear it back down

Deployment is easy, tearing down the infra is just as easy with the terraform destroy command.

To remove the VPC, use the following command

```
AWS_PROFILE=build-user terraform destroy
```





terraform destroy

And there it does, all gone!

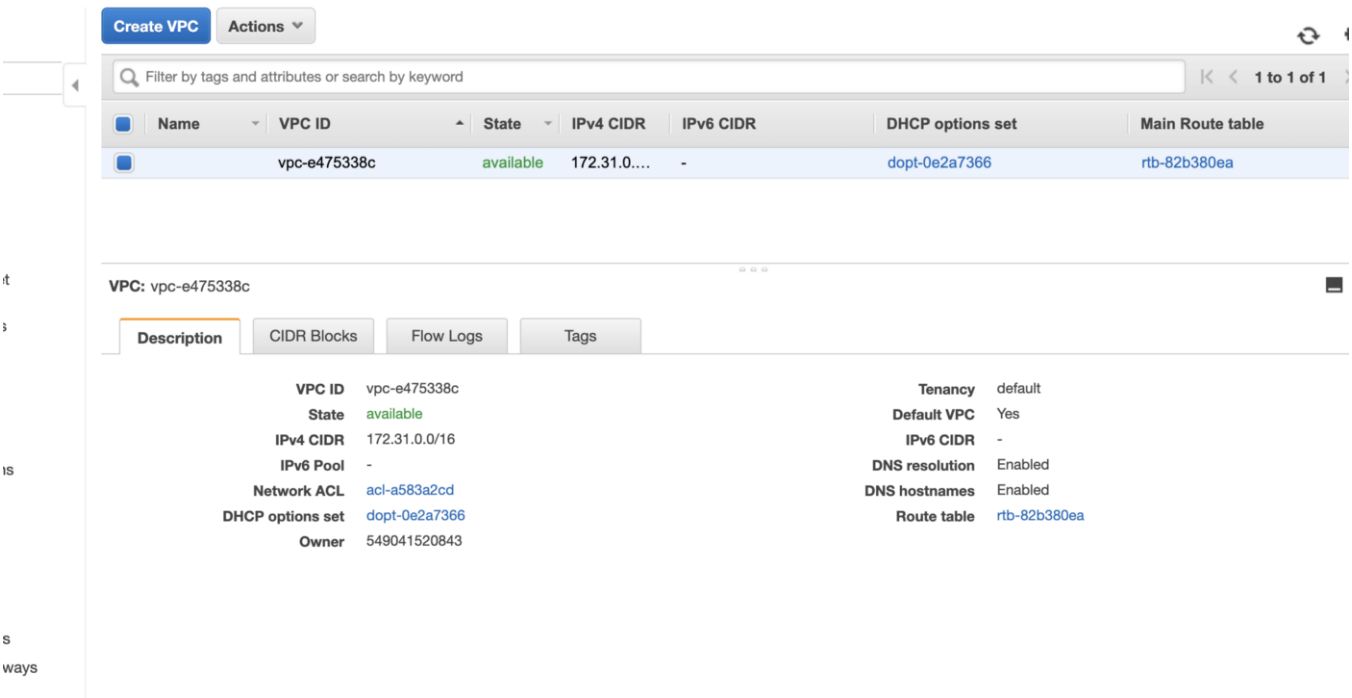


Photo by Markus Spiske on Unsplash

. . .

Conclusion

After reading this, you should be able to setup terraform, authenticate with AWS and be able to create infrastructure, manage its state and destroy infra within AWS.

To build on top of this example, just add more and more files containing AWS resources.

Terraform really is straightforward and the beauty is that you can repeat creating and tearing down infrastructure over and over, no longer you need to create things through the AWS console, and if you use in combination with source control, such as git, you can maintain a history of how the infrastructure has changed, and also who changed it!

Photo by Jamar Penny on Unsplash

AWS Terraform DevOps Development Infrastructure