Building My First Infrastructure through Terraform

**What is Terraform?**

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions. The infrastructure Terraform can manage includes low-level components such as compute instances, storage, and networking, as well as high-level components such as DNS entries, SaaS features, etc.

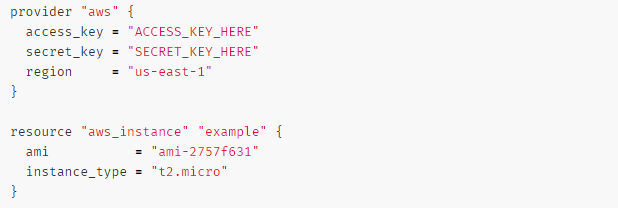
**Installing Terraform:**

To use Terraform it must first be installed in your machine. Find the appropriate package for your system and download the package from <https://www.terraform.io/downloads.html>. You will find a zip file which when unzipped will have a terraform binary file. And finally set path for Terraform binary file in PATH of the environmental variables(if Windows). Set PATH appropriate to your OS in which you are installing. To verify your installation is completed properly, open terminal and type terraform which will give the output with the list of commands terraform uses. Now Terraform is successfully installed in the machine.

**Build First Infrastructure:**

Let’s start building infrastructure through Terraform. As a first step we will take AWS as provider and start building infrastructure in AWS. Pre-Requisites for building infrastructure in AWS is having just an AWS account. Charges will apply if you are running code out of free tier. The set of files used to describe infrastructure in Terraform is simply known as a Terraform configuration. We will write a configuration to bring up a single AWS instance. Below is the code for bringing up single AWS instance.

A provider is responsible for creating and managing resources. Providers are usually where we want to build our infrastructure. Some common providers are AWS, Microsoft Azure etc. Save the below with the extension .tf and save it in the path where terraform binary file is kept.

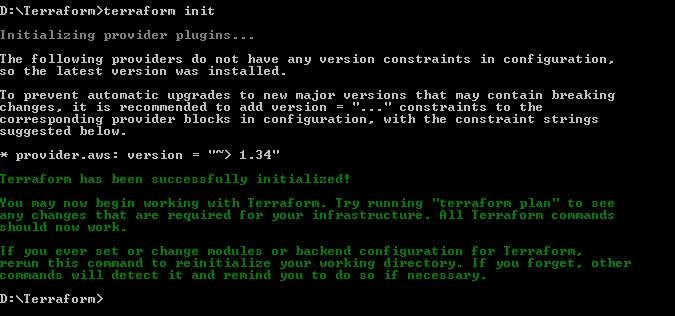


Paste your access key and secret key from your AWS account. Region is where the instance should be running.

The resource block defines a resource that exists within the infrastructure. In this example EC2 instance is the resource we are bringing up. The resource block has two strings before opening the block: the resource type and the resource name. In our example, the resource type is "aws\_instance" and the name is "example". I have actually used the below configuration for setting up infrastructure with t2.micro.

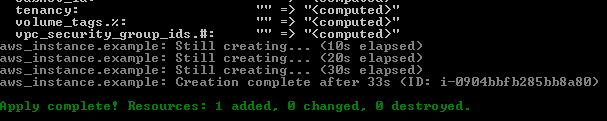
ami-b374d5a5

**Initialization:**

The first command to run a new configuration is terraform init. The terraform init command will automatically download and install any Provider binary for the providers in use within the configuration, which in this case is just the aws provider.

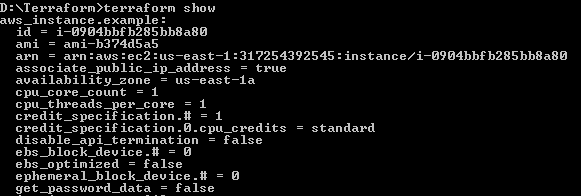
In the same directory where the configuration file is kept run terraform apply. Once the command is run it will show the execution plan how terraform will change the infrastructure according to the code. The output has a + next to aws\_instance.example, meaning that Terraform will create this resource. If terraform apply failed with an error, read the error message and fix the error that occurred. At this stage, it is likely to be a syntax error in the configuration.

If the plan was created successfully, Terraform will now pause and wait for approval before proceeding. If anything in the plan seems incorrect or dangerous, it is safe to abort here with no changes made to your infrastructure. In this case the plan looks acceptable, so type yes at the confirmation prompt to proceed.



After this, Terraform is all done! You can go to the EC2 console to see the created EC2 instance. Terraform also wrote some data into the terraform.tf state file. This state file is extremely important; it keeps track of the IDs of created resources so that Terraform knows what it is managing. This file must be saved and distributed to anyone who might run Terraform.

You can inspect the current state using terraform show:



Great! You've built your first infrastructure with Terraform. You've seen the configuration syntax, an example of a basic execution plan, and understand the state file.

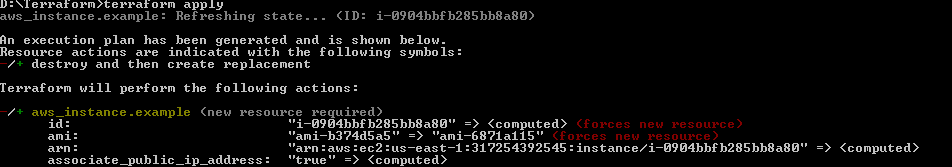
**Change Infrastructure:**

Changing the instance and bringing up different instance is so easy that you just need to modify the instance ami and run terraform apply. It will destroy the old instance and bring up the new instance in no time.

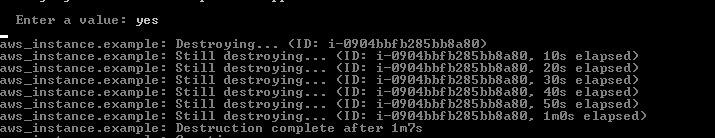
I have changed my instance to below ami id:

ami-6871a115

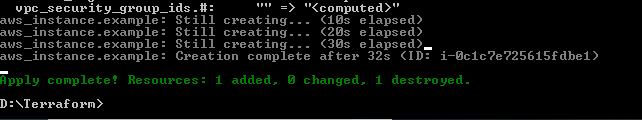
Once changed and running terraform apply will give me the following output. **Note:** Yes option is required to apply the changes once the execution plan is created.



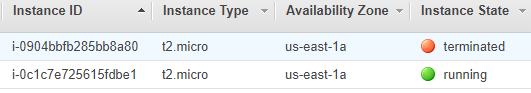
The -/+ option describes there has been a change in the infrastructure.



Once the state is changed it will give the following output.



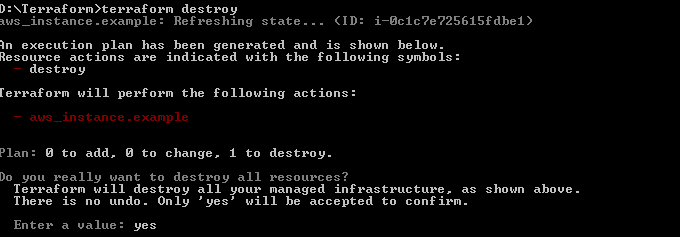
If you try checking the EC2 console the following output would be displayed.



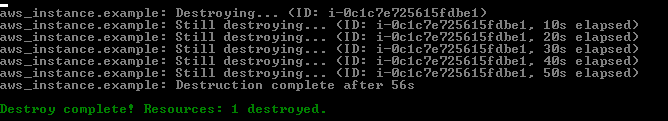
**Destroy Infrastructure:**

Destroying your infrastructure is a rare event in production environments. But if you're using Terraform to spin up multiple environments such as development, test, QA environments, then destroying is a useful action.

Terraform destroy is the command used for destroying the infrastructure.



The - prefix indicates that the instance will be destroyed. As with apply, Terraform shows its execution plan and waits for approval before making any changes.



That’s how I did my first Infrastructure as Code using Terraform.