Terraform Journey

**EC2 INSTANCE CREATION**

My first project has been downloading/installing terraform to work on my windows machine. This included being able to access terraform from powershell. The only way this was possible was by adding a new %path% for the location of the terraform.exe. I was then able to go into terraform console and use terraform from PS. AWS CLI was needed as well for terraform to take advantage of AWS.

Next was downloading other prerequisites such as a windows subsystem for linux and setting up AWS FREE tier IAM user to be able to deploy and use EC2 and S3 in their entirety by assigning a policy to them. I also downloaded a .csv of the keys needed to be able to commit changes to AWS from terraform.

The file format is .tf and can be edited using notepad. This file outlines the provider/region (AWS in this case), the resources (an EC2 instance in this case), and a tag for it.

The commands I’ve gone over so far involve formatting the file aka terraform fmt, terraform init, terraform validate, then terraform apply. Also inspected state to see metadata and end result of creation and the specifics of the server/instance.

After playing around I have successfully create a running EC2 instance in AWS and I believe to get rid of the resource I now would need to terraform destroy.

**Changing Configuration**

So far I’ve only edited the .tf file manually and changed the AMI. I formatted one more time and then since the terraform was already initialized with init I chose to do terraform apply. This shut down my last instance, committed new changes, and started running the new instance with the updated ami. With terraform show I could now see the details of the newly created instance.

I then used terraform destroy to get rid of the resources created. This should not affect other projects. Terraform automatically recognizes dependencies and destroys it in a proper order.

**Variables**

Now I’m introducing variables. In this case an assigned variable was defined in string format to name the server. That variable was then called later in a tag to name the instance. (ie: var.Example\_Name). This is static however so not complex. The complexity starts when functions are added. The variable can also be changed from the CLI that you are using to deploy these infrastructure changes, for me It is power shell right now. (example: **terraform apply -var "instance\_name=YetAnotherName"** )

**IP addresses and IDs & Querying them**

Created a separate outputs.tf and applied it. This essentially defined what the outputs would be and the values were gotten by calling things associated with the EC2 instance AKA aws\_instance.app\_server in this case.

***output "instance\_public\_ip" {***

***description = "Public IP address of the EC2 instance"***

***value = aws\_instance.app\_server.public\_ip***

***}***

Example above.

**GitHub Repos/Pushing/Pulling/CLI installs**

The first thing I did today was I wanted to figure out how to commit changes and push/pull things to my github account. I used trial and error and ChatGPT to guide me through errors. I ran into errors of files being too big, the origin not being correct, having a private vs public repo, etc… I had to install the git CLI as well to use through powershell. After selecting my directory I can set the origin and push whatever is in the directory to the repo through HTTPS through the link that github gives you. Some commands were git init for the directory, git add . to add the files in the directory to what will be pushed later on. Git remote origin, git remote -v to see the name of the origin identified, and the git repos https link as well. I’ve tested it using this file here and will continue to use the master section to push/pull files to get down the process of doing it. Gotta figure out how to automate this for future CI/CD home lab creation. When I update this document I’m able to create another commit and then push to the master section of my repo. It keeps track of when it’s committed and updated.

**Visual Studio Code / Terraform Relational Database / GitHub Practice**

Today I downloaded visual studio code while looking for a IDE autocomplete tool for terraform. I watched a video on creating a RDB using AWS free tier. I defined two .tf files. One for defining provider and creds. The other was to define the resource I would be deploying aka the RDB. It was a mysql database using free tier resources. I used terraform init, plan, validate, apply, and destroy. The build was successful. I now know you can control everything from the initialized directory you are working in. I then wanted to push the file for the rdb into github just to document my progress as well as this document. I ran into an error where github actually recognizes the different repositories and wouldn’t let me push to the master unless I allowed it using *--allow-unrelated-histories* . I think git merge was also an option. I was able to push to the repo in github however and my understanding of how these tools work is increasing. Practicing pushing and pulling to different resources is extremely useful right now. I cannot wait to start doing automated CI/CD actions. One step at a time. Next will be figuring out deploying a few other types of resources in terraform first and trying to combine deployments instead of one resource at a time.