DevOps Technical Exercise – Sports Interactive

Web Server Provisioning

For this exercise I have decided to use the following tools:

* Terraform
* AWS
* Ansible
* Jenkins
* Nginx

First, I want to provision my EC2 instance using Terraform. For this I will need an AWS account, AWS CLI, an IAM user, and both an access key ID and a secret access key - I create these 2 keys using the AWS console and then add them in my .aws/credentials file. I then get started with my Terraform configuration.

A screenshot of a computer

Description automatically generated with medium confidence

I have decided to create the following AWS resources in Terraform:

* IAM user
* S3 bucket
* EC2 instance

There is also a policy for both the S3 bucket and the IAM user. The state policy ensures the S3 bucket is locked down to protect the state file which will eventually be stored in there. For now, we will store the state locally as the S3 bucket doesn’t exist yet, so we can comment out the backend code for the meantime.

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated with medium confidence

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

This can be locked down further in future if required.

Timeline

Description automatically generated

I then run a terraform init in the terraform directory to initialise the working terraform directory. Then a terraform plan which will show me all the changes that will be made to the current infrastructure – in this case it will show that it will create my resources. Finally, when I am happy with the plan I run a terraform apply to apply these changes in AWS.

Once my resources have been created, I uncomment my s3 backend configuration.

Text

Description automatically generated

I create new access keys for my terraform IAM user in the AWS console and use these in my .aws/credentials file (I will also use these later in my pipeline). I can either then do a terraform init -migrate-state, or a plan then apply to change from my local state to the newly created backend s3 bucket.

Text

Description automatically generated

This will create a small EC2 instance with an Amazon Linux image.

I then want to use Ansible to provision my EC2 instance into a web server. It would be nicer if I customised an AMI so Ansible was pre-installed but for this exercise I will simply SSH in and install it manually. To do this I need SSH access to my EC2 instance. To do this in Terraform you can create a VPC and provide the id as a vpc\_id, but according to the terraform docs this is an optional field, and if empty will trigger a new resource, so for this exercise I will let Terraform create the VPC for me.

Text

Description automatically generated

This security group will allow any IP address to SSH into the EC2 instance. Obviously this would be a security risk but to save time I will temporarily attach this security group to my EC2 instance.

Text

Description automatically generated

I can now SSH onto my EC2 instance. I then run the following commands to install Ansible…

sudo yum update

sudo amazon-linux-extras enable ansible2

sudo yum install -y ansible

I can now edit the /etc/ansible/hosts file to configure Ansible to provision the web server locally.

[webserver]

localhost

I then want to create a playbook that Ansible can run to install nginx.

Text

Description automatically generated

This will check to see if nginx is installed, if not it installs it. Then it checks to see if the nginx service has started, if not it will start.

To prove this is working, we want a basic html page that can be used to show the nginx web server is doing what we expect it to do.

I added some extra config files and ansible functionality to accomplish this.

Text

Description automatically generated

\Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

It would be much better to setup a Jenkins master in AWS or Kubernetes, but for this exercise I will deploy my Jenkins server using Docker on my local machine.

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated



I haven’t actually ran any of this other than the Dockerfile which seems to work, so I imagine I would have some troubleshooting around the git repo and the workspace directory, as well as the SSH functionality in the python script. However, in interest of time (and due to the fact I’ve already spent 4 hours on this already) I’ve decided to leave it there.

I am slightly unhappy that this is unfinished, if I had longer to spend on the assessment, I would have worked through what I had done practically as well to ensure everything worked as expected. Although I did enjoy refamiliarizing myself with Ansible, and creating a pipeline from scratch, I will revisit this personal project in future. I also couldn’t quite get the nginx web server to show my index html, this is something else I could have got working with more time.

Git repo: https://github.com/freddypaxton/web-server-project