Waterford Institute of Technology

Cloud Computing 2

Bachelor of Science (Hons) Applied Computing

Terraform & Ansible AWS Provisioning

Ciaran Roche - 20037160

April 18, 2019

Contents

1 Introduction

From work carried out in labs, the following spec was laid out. Using Terrafrom to provision AWS instances and Ansible to configure a HAProxy EC2 instance which load balances traffic between two webserver instances. Ansible roles to be used for haproxy load balancer and webserver. This paper documents the work carried out completing the above task.

2 Theory

2.1 Ansible

Ansible is an open-source IT automation engine developed by Red Hat. It is fast becoming the de-facto in automation as it allows for scalability, consistency and reliability of infrastructure and environments through the automation of the following –

- **Provisioning** – Setting up various servers and services within defined infrastructure
- **Configuration Management** – Changing configuration of an application, device, or service. Along with the starting, stoping and restarting of services. Installing and updating etc.
- **Application Deployment** – Making SysAdmin work easier by automating deployments of applications from testing to production systems.

For this assignment we will be focusing on Configuration Management and leaving the Provisioning to Terraform.
Something I find is a major selling point to ansible is the fact that it is agentless. Meaning there is no software to install on the services that we are trying to automate. Mix this with Ansible using YAML under the hood to allow for an easy to read understandable configuration for users.

2.2 Terraform

Everywhere you look in the software ecosystem, you can see people raving about Terrafrom. Google trends show a drastic and consistent rise in its popularity since is launch in April 2014. So what is Terraform? It is a tool for developing, changing and versioning infrastructure safely and efficiently. Developed by HashiCorp and written in GoLang, its rise in popularity is often attributed to its simple syntax that allows for easy modularity and works against a multitude of cloud providers. From my time playing around with Terraform it is easy to draw similarities to data structs in Terrafrom to those in GoLang, feeding back into the popularity of GoLang and Terraform. Some of the features of Terraform are as follows –

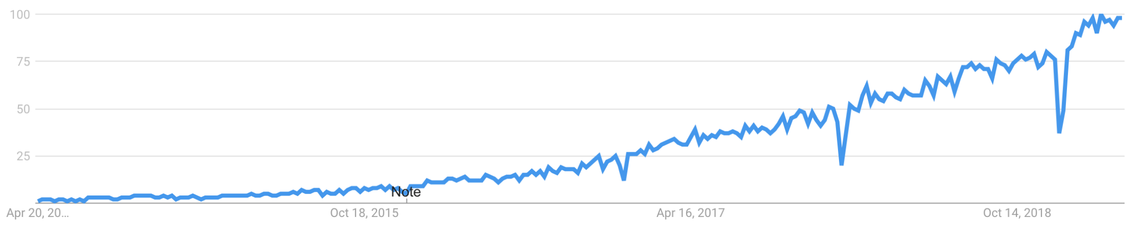


Figure 1: *Google trend – Terraform*

- Define infrastructure in config/code.
- Platform agnostic
- Adheres to many coding principles, such as source control and testing.
- Has a mature and enthusiastic community behind it

I was tempted to add speed of operations to the above list, as most resources online reinforce the speed of Terraform. Which from my experience with the exercise is correct. But from other work I have done with Terraform on GKE, the speed is just not there. It was a case where it was quicker to provision my services manually then use Terraform on GKE. This could be due to a many different factors, but made me second guess the decision of adding it to the above list. Despite this, working with Terraform and AWS the value and speed of quickly provisioning services is apparent.

3 Practical

3.1 Prerequisites

Before starting the Practical I dived into the cloud blueprint example at Appendix A. Just getting this running took quite some time and troubleshooting. The first error encounter was due to how I installed Ansible on my machine, using homebrew. This limited ansible access to my local python resources. So I had to reinstall Ansible using pip, once done I was able to execute the playbook, but ran into an error with the configuration of the right haproxy AMI. The first AMI I created using Ubuntu 16, installed HaProxy which turned out to be an older version which was not compatible with the Ansible Playbook. Updated HaProxy to overcome that error. The next error was due to Python not being installed on my HaProxy AMI. I updated these and started again, running into another error. This time it was a weird parsing error which occurred on the secondIP role. As a sanity check I decided to update my AMI to Ubuntu 18, installed latest HaProxy and Python, and everything this time round ran as expected. Some time was spent with this example running gaining a understanding of the configuration and files.

3.2 Configuring the lab

3.2.1 Terraform

After gaining an understanding of the blueprint I decided to abstract the necessary files from the blueprint to fit the spec of the exercise. The first aspect to this configuration is the provisioning handled by Terraform. To gain an understanding some terminology is needed.

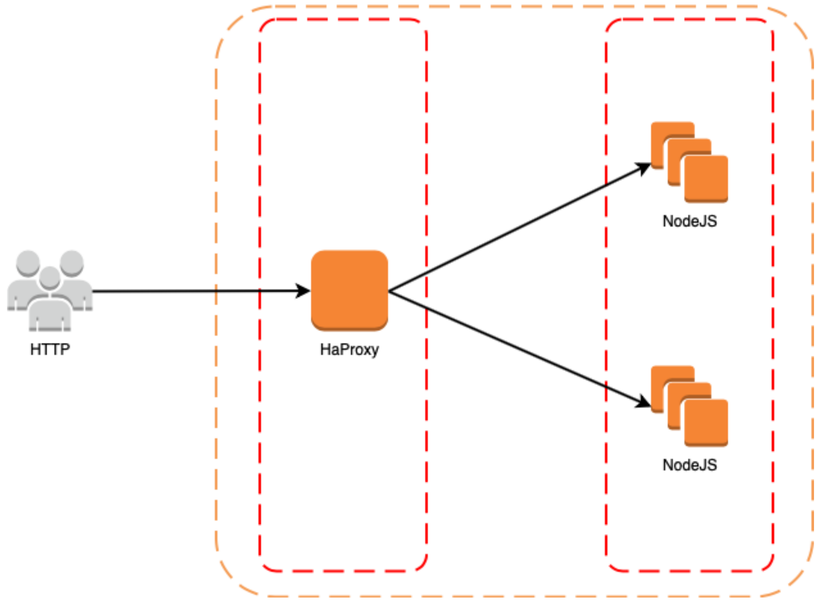


Figure 2: *Exercise Infrastructure*

As we know from above, Terraform is used to create, manage, etc, infrastructure. In order for Terraform to understand the API interactions of the infrastructure we need to declare specific providers. In the case of this exercise we are declaring an AWS provider and passing the region variable to it :

```
provider "aws" {
  region = "${var.aws_region}"
}
```

Maintaining common development practises such as DRY, terraform allow us to declare data soruces which we can call on anywhere within our configuration. An example of one used in this exercise can be seen below, we declare it as an aws_ami type, the second param is the name of the data type which we can refer to within our configuration. The structure of this data type following that layed out from terraform aws ami type, where we are declaring a specific ami to use. In this case it is a HaProxy AMI which I created.

```
data "aws_ami" "haproxy_aws_amis" {
  most_recent = true

  filter {
    name = "name"
    values = ["HaProxy-lb"]
  }

  owners = ["998447052043"]
}
```

Another type we need to understand is the terraform resources, these are the most important element in terraform as it is a representation of a specific piece of infrastructure. An example can be seen below, where we are declaring an aws_vpc, like with the data type, we tell terraform what kind this type is, then give it a name which we can call on. The body of the resource is declaring the nessesary variables needed to create a vpc

```
resource "aws_vpc" "default" {
  cidr_block = "20.0.0.0/16"
  enable_dns_hostnames = true

  tags {
    Name = "haproxy_test_vpc"
  }
}
```

A common practice with terraform is to declare outputs, these outputs feed us back information on completion of running terraform. An example of this from the exercise can be seen below where we are declaring

```
output "HAProxy␣nodes␣private␣IPs" {
  value = "${aws_instance.haproxy_node.*.private_ip}"
}
```

Like with the reusable data types we can declare variables within terraform. An example used in this exercise is shown below, where we declare a variable name, give it a description and a value.

```
variable "aws_region" {
  description = "Home␣AWS␣region"
  default = "eu-west-1"
}
```

The full code can be found in Appendix B to this exercise. Simply running the following commands will execute terraform and provision our infrastructure which resembles figure 2

```
$ terraform init
$ terraform apply
```

3.2.2 Ansible

Like with terraform above an understanding of some of Ansibles core principles are needed. As Ansible grew from a basic scripting format and the use cases became more complex the Ansible Playbook originated. This is a set structure in which multiple plays can be run. Think of a play as a task or a script. These plays are executed in a specific order, an example of this can be taking from this exercise. Here we can see we have 3 plays we want to execute, the first is a check to see if the correct version of Ansible is installed, this is a common pattern in Ansible playbooks to perform all requirement checks at the begining of a playbook. The following plays provide the configuration for HaProxy and the backend web app nodes.

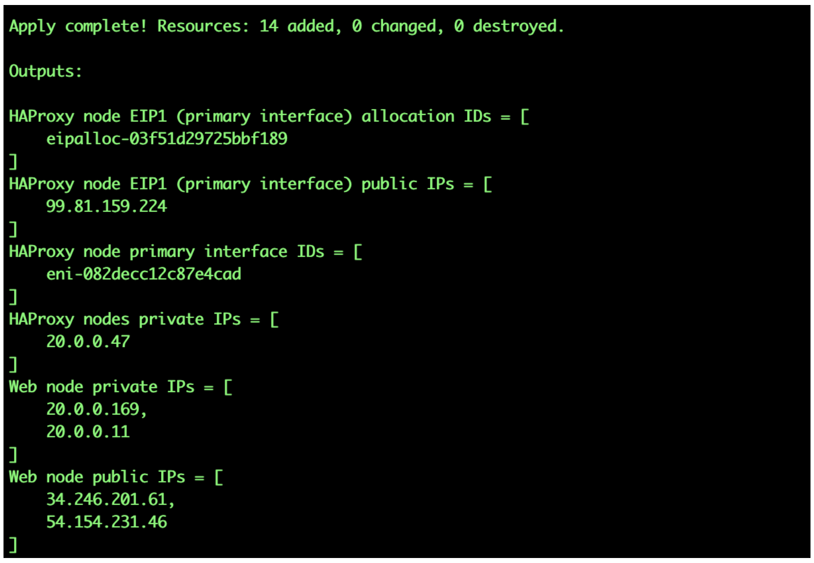
```
- name: require 2.6+ minimum Ansible version due to EC2 EIP facter
  hosts: all
  ...
- name: configure HAProxy LB nodes
  hosts: tag_Name_haproxy_lb_node
  ...
- name: configure Web backend nodes
  hosts: tag_Name_haproxy_web_node
  ...
```

To avoid this becoming unmangable the plays are broken into roles. Roles are reusable pieces of configuration which can be shared on Ansible Galaxy. Roles are not playbooks, while they can and often are executed independently there is no way of executing a role outside of a playbook. A role is broken into its own directory, within these directories we house all necessary files for that specific role. Each role has its own task directory, here is where we have that roles specific main configuration. An example seen below is taken from this exercise, for the HaProxy LB role. As we can see this role has two tasks, where configuration is copied from the role to the instance, then haproxy is restarted.

```
- name: generate new HAProxy LB configuration
  template:
    src: haproxy-lb.cfg.j2
    dest: /etc/haproxy/haproxy.cfg
    mode: 0664
    owner: root
    group: haproxy

- name: restart HAProxy LB service
  systemd:
    name: haproxy
    state: restarted
```

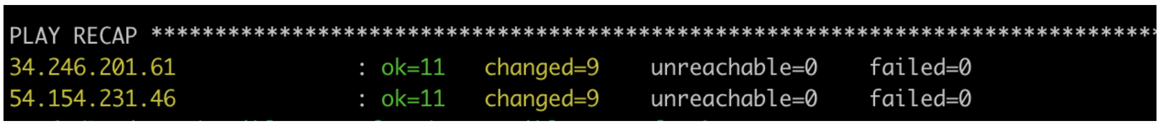
3.3 Verification

To verify completion of this lab, figure 3 shows the output from the Terraform script. Figure 4 shows the recap from the Ansible playbook, and Figure 5 has a view of the AWS EC2 console, and the result of visiting the HaProxy IP can be seen in Figure 6.



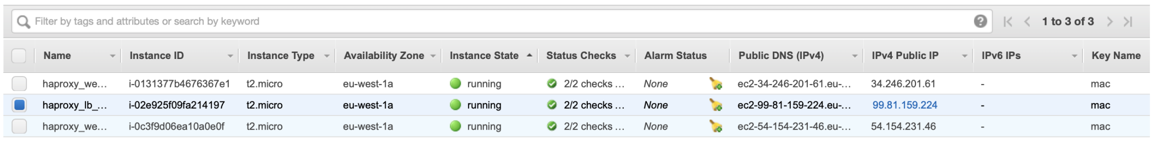
```
Apply complete! Resources: 14 added, 0 changed, 0 destroyed.

Outputs:

HAProxy node EIP1 (primary interface) allocation IDs = [
    eipalloc-03f51d29725bbf189
]
HAProxy node EIP1 (primary interface) public IPs = [
    99.81.159.224
]
HAProxy node primary interface IDs = [
    eni-082decc12c87e4cad
]
HAProxy nodes private IPs = [
    20.0.0.47
]
Web node private IPs = [
    20.0.0.169,
    20.0.0.11
]
Web node public IPs = [
    34.246.201.61,
    54.154.231.46
]
```

Figure 3: *Exercise Infrastructure*



```
PLAY RECAP *****************************************************************************
34.246.201.61              : ok=11   changed=9    unreachable=0    failed=0
54.154.231.46              : ok=11   changed=9    unreachable=0    failed=0
```

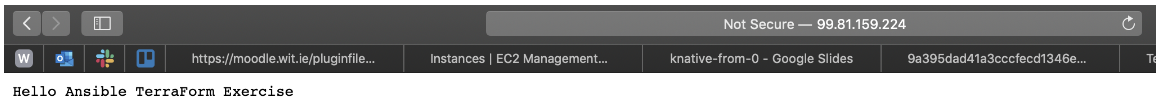
Figure 4: *Terraform Success Output*



Filter by tags and attributes or search by keyword — 1 to 3 of 3

	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IPv6 IPs	Key Name
☐	haproxy_we...	i-0131377b4676367e1	t2.micro	eu-west-1a	● running	✓ 2/2 checks ...	None	ec2-34-246-201-61.eu-...	34.246.201.61	-	mac
■	haproxy_lb_...	i-02e925f09fa214197	t2.micro	eu-west-1a	● running	✓ 2/2 checks ...	None	ec2-99-81-159-224.eu-...	99.81.159.224	-	mac
☐	haproxy_we...	i-0c3f9d06ea10a0e0f	t2.micro	eu-west-1a	● running	✓ 2/2 checks ...	None	ec2-54-154-231-46.eu-...	54.154.231.46	-	mac

Figure 5: *AWS console EC2*



Not Secure — 99.81.159.224
https://moodle.wit.ie/pluginfile...
Instances | EC2 Management...
knative-from-0 - Google Slides
9a395dad41a3cccfecd1346e...

```
Hello Ansible TerraForm Exercise
```

Figure 6: *Visit Website via HaProxy*

4 Summary

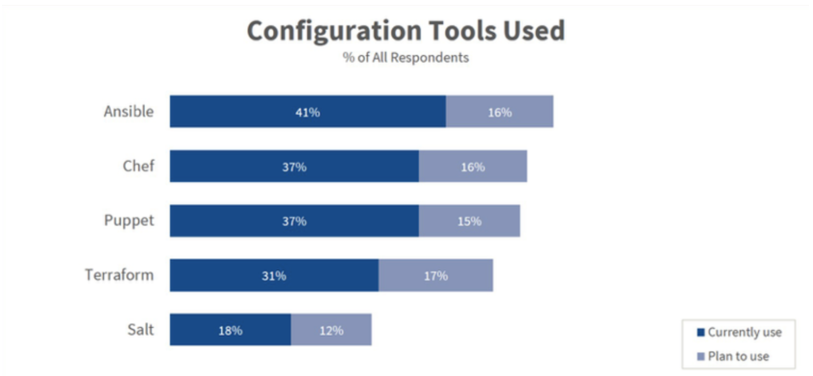


Figure 7: *Rightscale 2019 State of the Cloud Report.*

With the rise of cloud adoption by companies the need for configuration management tools is significant. Figure 7 shows the tools being used today, showing Ansible to be leading the poll. With this fact in mind, gives the importance of gaining experience with Ansible through this exercise. Having already completed similar exercises within the last year in manually configuring haproxy, it is easy to see the benefits configuration management has. Personally I found having the power to quickly spin up and tear down beneficial as it allowed for easier troubleshooting. During manual configuration I found it quite time consuming troubleshooting errors.
When comparing and looking at Terraform and Ansible, the both had a pretty low barrier to entry, being easy to upskil on. I leaned more toward Terraform, in the syntax being cleaner and easier to understand. This being just a personal opinion. Mixed with a much easier install and set up of the tool itself. Ansible on the other hand is quite a lot more mature, with that there is quite a lot of resources on line which helped in the upskilling.
After getting a taste of automation with this lab, along with some terraform work from my FYP, it will be quite hard to go back to manually configuring infrastructure again

Appendices

A Cloud BluePrints

https://github.com/haproxytech/cloud-blueprints/tree/master/aws_heartbeat

B Exercise Code

https://github.com/ciaranRoche/ansible-terraform-aws