Openstack - Infrastructure-as-Code

Terraform + Ansible to create a Kubernetes Cluster

Thomas Stadler[1], Klaus Thüringer[2], and Stephan Pilwax[3]

[1]2110781014@fh-burgenland.at
[2]2110781007@fh-burgenland.at
[3]2110781003@fh-burgenland.at

June 8, 2022

Contents

1 Introduction

In the introduction, the given task and the chosen approach are described.

1.1 Scope

Openstack Assignment by Peter Buzanits (translated from German):

On the Openstack installation of the University of Applied Sciences Burgenland, on which we have already practised infrastructure with its own network(s) and multiple instances. One or more services should run on it. Ideally these services are related to the topic of the work that is created for the InEn-ILV. Therefore, the delivery groups are the same as in the the ILV.

For example, something could be demonstrated that is described in the work. Or the results are presented in a wiki or similar. If this is difficult to do, it can also be done independently of the of the ILV.

Caution. Everything for the submission should take place in the project InEn-GruppeX (X is placeholder for the group name here). The project InEn-21107810xx is still available for personal experiments until the end of the semester.

Furthermore, a report is to be submitted in which the infrastructure is described. This should be designed in such a way that someone who takes over this infrastructure infrastructure (e.g. as a successor to the administrator in a company) can manage it without further inquiries. So all aspects like networks, IP addresses, security groups, etc. should be described. Also the services should be accessible for use after reading this report.

There will be a draft version of the report, which should already have the complete scope. This will allow me to provide feedback before the final submission. The services do not have to be working at the time of the draft.

If you were not there for the last exercise and do not know your login credentials, please send me an email.

1.2 Approach

For the given task - setting up infrastructure on an Openstack [1] instance provided by the university - the chosen approach was to define the infrastructure and all configuration as code by the concept of Infrastructure-as-Code (IaC) [2]. To achieve this, the following open source software tools are used:

- Hashicorp Terraform [3]
- Red Hat Ansible [4]

Terraform is used to execute plans written in the Hashicorp Configuration Language [5], which is a superset of the common JSON configuration language. By nature, it is a declarative language, which means that you define a desired end result, instead of writing imperative statements. In this way, the code can achieve the feature of idempotency, which means that it can be applied multiple times without changing the result beyond the initial application.

These Terraform plans define all infrastructure components within Openstack like Network, Routers, Subnets, Floating IPs, Instances, etc., and their parameters like network CIDRs, IP addresses, vCPU count, disk and memory space etc.

Terraform plans can be applied, upon the Terraform engine will make the according HTTP calls to the Openstack API, which will result in the infrastructure being scheduled for provisioning.

After successful provisioning of the infrastructure resources on Openstack, Terraform will return the newly created Floating IP addresses, which are associated with the compute instances.

These IP addresses are written into the Ansible inventory [6], which is a configuration file in the ini-style used by Ansible. In the Ansible inventory, the hosts, which Ansible should target, are defined, as well as additional variables if needed, like which user to connect with or which port the SSH daemon listens at. For connecting to the hosts, Ansible uses simple SSH connections. Ansible uses so called Ansible Playbooks [7] to define the configuration, which are written in the YAML syntax.

Ansible Playbooks are made up of many small tasks, which most of the time basically represent shell commands, with the difference, that they are defined as YAML, which has the advantage of being easy to read and understand even for non-programmers. Additionally, Ansible Playbooks also declarative by nature, which means a desired end result is defined. Ansible then transparently takes care of achieving the desired outcome, no matter the starting point, all while keeping the code base as simple as possible. With Ansible Playbooks being very easy to read, it is often times not needed to write more sophisticated documentation. Ansible excels at automating on the operating system layer, which means running system updates, installing additional software, configuring. All this automation can be done on multiple hosts, without having to login to any host by hand.

1.3 Result

The end result after applying this infrastructure code is a Kubernetes Cluster consisting of one master (control plane) and two worker nodes. The virtual machine instances are provisioned via Terraform on Openstack. IP addresses of the machines are handed off to Ansible, which continues the provisioning process at the operating system level after Terraform successfully starts the Ubuntu 20.04 machine images. Ansible connects via the SSH key pair, which is provided to Terraform prior to `terraform apply` by the operator. Ansible updates the base system packages, installs necessary components for Kubernetes like `kubectl` and overlay network, and finally create a Kubernetes cluster and joining all nodes together. The root user on the master (control plane) machine will have access to kubeadm and kubectl.

2 Infrastructure-as-Code Setup & Usage

This section explains the infrastructure code and how to use it.

2.1 Terraform - Openstack provisioning

To get started, the Git Repository needs to be cloned.

```
$ git clone https://github.com/thomasstxyz/fhb-mcce-inenp-pt-openstack
```

Next, the credentials for authentication to the Openstack API, as well as the public ssh key will be read into environment variables.

```
$ export OS_USERNAME=my_user
$ export OS_PASSWORD=my_pass
$ export OS_PROJECT_ID=1234...
$ export TF_VAR_public_key="ssh-rsa ..."
```

The project id can be retrieved from the Openstack Horizon Dashboard under "Project - API Access" `http://172.20.41.1/horizon/project/api_access/`, upon pressing the button "View Credentials".

Now the setup is complete and Terraform should be able to authenticate to the Openstack API. Change into the terraform directory.

```
$ cd terraform
```

Use Terraform plan to look at what changes would be made by executing (this is just a dry run and does not make any changes).

```
$ terraform plan
```

A summary will be printed in the console, stating how many resources to add, change and destroy.

```
Plan: 21 to add, 0 to change, 0 to destroy.
```

If content with the dry run, the plan can be applied.

```
$ terraform apply
```

Again a summary will be printed, and the user is asked to confirm with yes.

After a successful Terraform apply, the IP addresses of the created compute instances are displayed on screen. Additionally, the addresses are written to the Ansible inventory file `../ansible/inventory`, which is needed by Ansible for the next step.

All resources can be deleted, by running Terraform destroy.

```
$ terraform destroy
```

2.2 Ansible - Kubernetes cluster & OS configuration

After provisioning of the Openstack instances via Terraform, Ansible creates a Kubernetes cluster. For this, the Ansible Role `geerlingguy.kubernetes` (https://github.com/geerlingguy/ansible-role-kubernetes) is used.

Change into the ansible directory

```
$ cd ansible
```

and install the project dependencies, which is only the kubernetes role.

```
$ ansible-galaxy role install -r roles/requirements.yml
```

Now the Ansible Playbook can be run to install dependencies, load kernel modules, do further configuration, and finally create a Kubernetes cluster on top of the compute instances.

```
$ ansible-playbook -i inventory playbook.yml
```

Note: There are certain `pre_tasks` specified in the `playbook.yml`, which are needed for this specific Openstack infrastructure. The hostname is mapped to `127.0.1.1` in the `/etc/hosts` file, which fixes a bug, where every time sudo is called, the process hangs for about ten seconds, which would significantly decrease the speed at which the Ansible playbook is run, because it calls sudo on every task. Furthermore, DNS nameservers are specified, because the Openstack DHCP server does not seem to provide them.

After successful playbook run, the Kubernetes cluster is ready for use.

Get started by logging into the master node

```
$ ssh ubuntu@<master_node_ip>
```

and switching to the root user.

```
$ sudo su - root
```

The `kubectl` command line tool can be used to interact with Kubernetes.

```
$ kubectl get nodes
$ kubectl cluster-info
```

3 Openstack Resource Specification

To get a full understanding of the Openstack resources, which are provisioned by Terraform, you have to look at the terraform plan files `*.tf` in the `terraform/` directory.

3.1 Resource Overview

Firstly, a private network is created with a subnet, which is where the compute instances are located. It is only a private network, in the sense that machines do not have public ip addresses. For this purpose, Floating IP addresses are created for each compute instance. Furthermore, a security group is created with multiple ingress rules.

- Ingress Port 22 - SSH
- Ingress Port 6443 - Kubernetes API

The SSH port is needed, in order for Ansible to connect. The Kubernetes API port is needed for the master (control plane) node, if the operator wants to manage the cluster from their own workstation.

The security group can and should be optimized, by specifying the remaining rules, which are needed by Kubernetes for in-cluster traffic. For the sake of simplicity, the same security group is used for the Kubernetes Master and Node. This should be sorted out before using this project in production.

A router is used to connect the private network with the provider network, which is where the Floating IP addresses are created. This network is accessible from the local area network (LAN), where the Openstack server is located. Another much needed resource is the public ssh key of the operator, which will be imported into Openstack by Terraform. Finally, two groups of Compute Instances are created. The `kube_master` and the `kube_node` resources. By default, one `kube_master` instance and two `kube_node` instances. The instances are installed from a `Ubuntu-20.04` image and the Openstack Nova instance type is `m1.big`.

References

[1] Open source cloud computing infrastructure. https://www.openstack.org/, 2022. (Accessed on 05/27/2022).

[2] Infrastructure as code. https://www.ibm.com/cloud/learn/infrastructure-as-code, 2022. (Accessed on 05/27/2022).

[3] Terraform - automate infrastructure on any cloud. https://www.terraform.io/, 2022. (Accessed on 05/27/2022).

[4] Red hat ansible automation platform. https://www.ansible.com/, 2022. (Accessed on 05/27/2022).

[5] Terraform language documentation. https://www.terraform.io/language, 2022. (Accessed on 05/27/2022).

[6] How to build your inventory. https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html, 2022. (Accessed on 05/27/2022).

[7] Intro to playbooks. https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html, 2022. (Accessed on 05/27/2022).