openSUSE®

Provisioning Kubernetes Cluster with Terraform and Ansible on openSUSE Cloud Infrastucture

Ragil Setianjaya
ragilsanjaya99@gmail.com

Ragil Setianjaya

- Graduated from Vocational High School 1 Cihampelas, Bandung
- Part of GLiB (GNU/Linux Bogor)
- Work at Boer Technology. PT
- openSUSE user since 2017

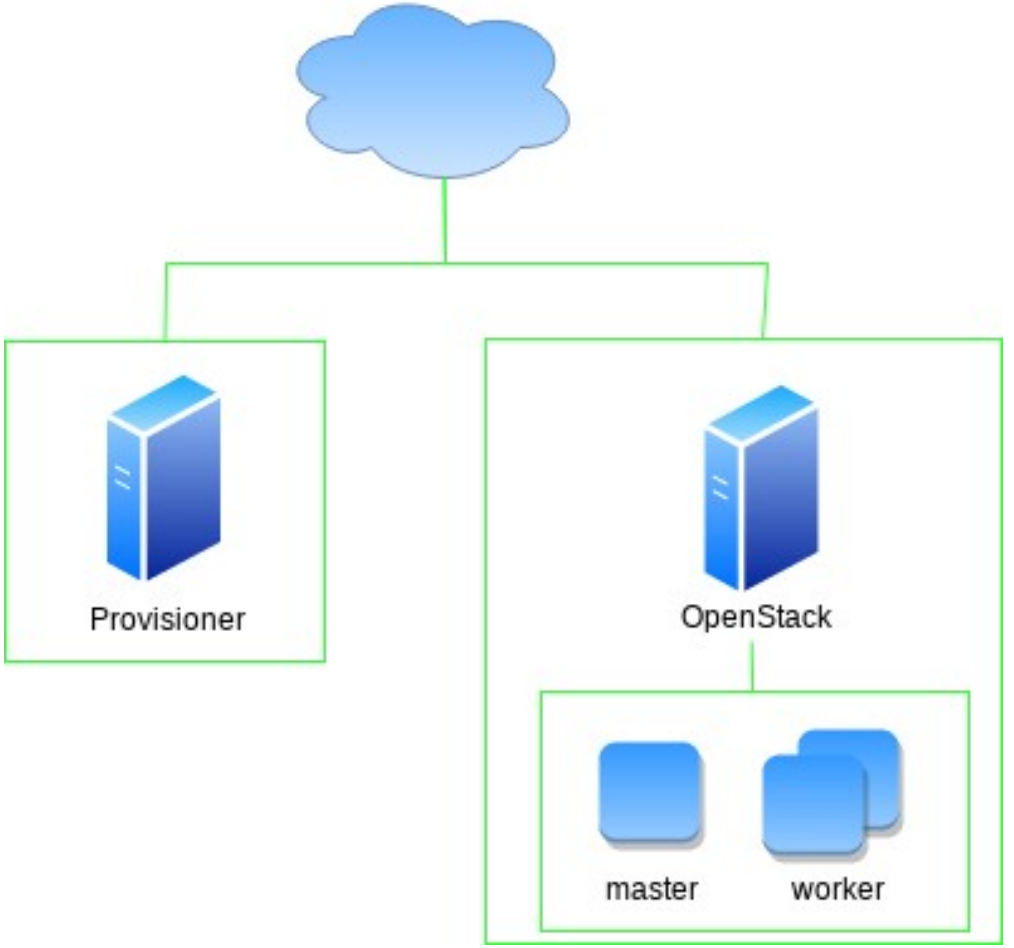
Environments

- OpenStack cloud server
- 1 VM terraform + ansible server (provisioner)

  Requirements provisioner server :

  - Python 2.7 (or newer)
  - Ansible 2.7 (or newer)
  - Jinja 2.9 (or newer)
  - Terraform 0.11 .XX

Topology



openSUSE

OpenStack

Open Source Software for creating IaaS private, public, community or hybrid cloud.

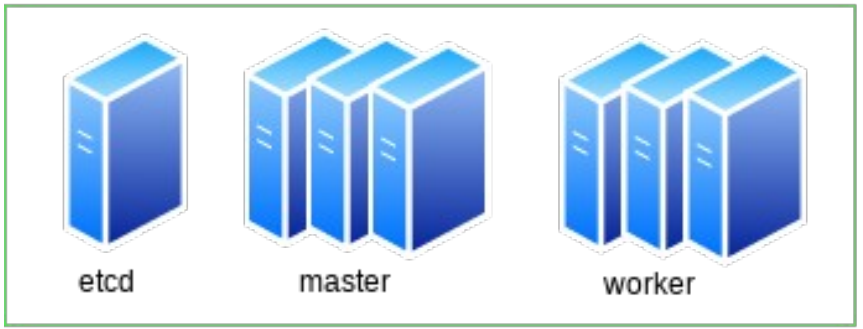
SUSE OpenStack Cloud

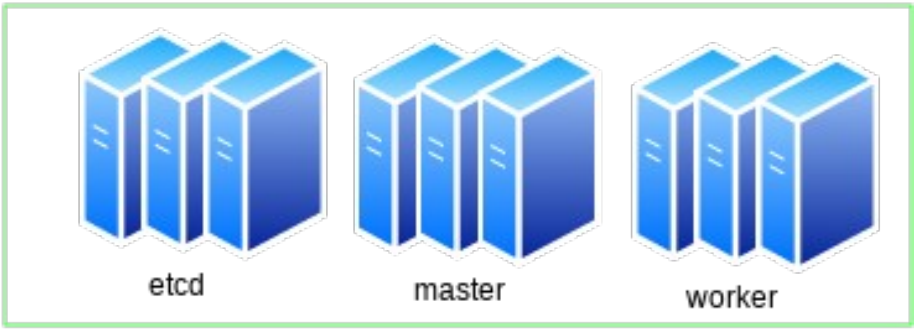
SUSE OpenStack Cloud provides an easy to deploy and manage heterogeneous cloud infrastructure for provisioning your development, test and production workloads in a way that is supportable, compliant and secure.

SUSE Cloud Versions

Product Release	General Support Ends
SUSE OpenStack Cloud 5	15 May 2017
SUSE OpenStack Cloud 6	15 Apr 2018
SUSE OpenStack Cloud 7	31 Dec 2019
SUSE OpenStack Cloud 8	31 May 2021
SUSE OpenStack Cloud 9	29 Apr 2022

Kubernetes Configuration Models





openSUSE

Terraform

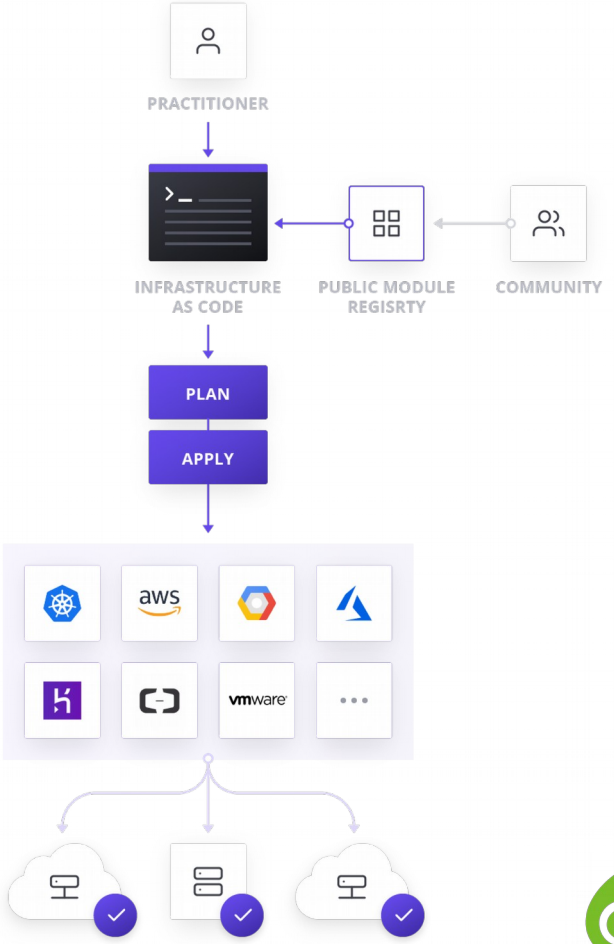
Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions.

Terraform

https://www.terraform.io

How Terraform Works

Terraform allows infrastructure to be expressed as code in a simple, human readable language called HCL (HashiCorp Configuration Language). Terraform CLI reads configuration files and provides an execution plan of changes, which can be reviewed for safety and then applied and provisioned. Extensible providers allow Terraform to manage a broad range of resources, including hardware, IaaS, PaaS, and SaaS services.



openSUSE

How Terraform Works (1)

Configuration files describe to Terraform the components needed to run a single application or your entire datacenter. Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, Terraform is able to determine what changed and create incremental execution plans which can be applied.

https://www.terraform.io/intro/use-cases.html

openSUSE

How Terraform Works (2)

```
root@ra-terraform:~/terraform# terraform init

Initializing provider plugins...
- Checking for available provider plugins on https://releases.hashicorp.com...
- Downloading plugin for provider "openstack" (1.23.0)...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.openstack: version = "~> 1.23"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

```
root@ra-terraform:~/terraform# ls -la
total 28
drwxr-xr-x  3 root root 4096 Oct  3 15:41 .
drwx------ 10 root root 4096 Oct  3 15:10 ..
-rw-r--r--  1 root root  595 Oct  3 15:10 instance.tf
-rw-r--r--  1 root root  155 Oct  3 13:57 provider.tf
drwxr-xr-x  3 root root 4096 Oct  3 14:49 .terraform
-rw-r--r--  1 root root 5325 Oct  3 15:41 terraform.tfstate
root@ra-terraform:~/terraform# cat terraform.tfstate
```

```
root@ra-terraform:~/terraform# terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.


------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + openstack_compute_floatingip_associate_v2.floating_ip
      id:                   <computed>
      floating_ip:          "${openstack_networking_floatingip_v2.floating_ip.address}"
      instance_id:          "${openstack_compute_instance_v2.instance.id}"
      region:               <computed>

  + openstack_compute_instance_v2.instance
      id:                   <computed>
      access_ip_v4:         <computed>
      access_ip_v6:         <computed>
      all_metadata.%:       <computed>
      availability_zone:    <computed>
      flavor_id:            <computed>
      flavor_name:          "flavor1"
      force_delete:         "false"
      image_id:             <computed>
      image_name:           "openSUSE15.1"
      key_pair:             "ra-terraform"
      name:                 "test-openSUSE15.1"
      network.#:            "1"
      network.0.access_network:   "false"
      network.0.fixed_ip_v4:      <computed>
      network.0.fixed_ip_v6:      <computed>
      network.0.floating_ip:      <computed>
      network.0.mac:              <computed>
      network.0.name:             <computed>
      network.0.port:             <computed>
      network.0.uuid:             "63de2d6b-cbc1-4c73-8177-6ab0b31902c6"
      power_state:          "active"
```

Terraform Use Cases

- Multi-Cloud Deployments

  It's often attractive to spread infrastructure across multiple clouds to increase fault-tolerance. By using only a single region or cloud provider, fault tolerance is limited by the availability of that provider. Having a multi-cloud deployment allows for more graceful recovery of the loss of a region or entire provider.

`https://www.terraform.io/intro/use-cases.html`

Providers

- Alibaba Cloud
- AWS
- Azure
- Cloudflare
- Cloudstack
- DigitalOcean
- Google Cloud Platform
- Hetzner Cloud
- HuaweiCloud
- Kubernetes
- Linode
- Naver Cloud
- Nutanix
- 1 & 1
- OpenStack
- OpenTelekomCloud
- Oracle Cloud Platform
- Rundeck
- Scaleway
- VMware NSX-T

Provider Configuration file

```
...
provider "openstack" {
  user_name   = "admin"
  tenant_name = "admin"
  password    = "password"
  auth_url    = "http://openstack.com/"
}
...
```

```
...
provider "google" {
  project = "acme-app"
  region  = "us-central1"
}
...
```

```
...
provider "aws" {
  version = "~> 2.0"
  region  = "us-east-1"
}
...
```

https://www.terraform.io/docs/providers/index.html

Terraform Kubernetes Variable

Variable	Description
cluster_name	All OpenStack resources will use the Terraform variable cluster_name (default example) in their name to make it easier to track.
dns_nameservers	An array of DNS name server names to be used by hosts in the internal subnet.
number_of_k8s_masters, number_of_k8s_masters_no_floating_ip	Number of nodes that serve as both master and etcd. These can be provisioned with or without floating IP addresses.
number_of_k8s_masters_no_etcd	Number of nodes that serve as just master with no etcd.
number_of_etcd	Number of pure etcd nodes.
number_of_k8s_nodes	Kubernetes worker nodes.

Thank You

openSUSE