Kubernetes deployment with Terraform on cloud.ca

Clément Contini

January 26, 2018

Objectives

Objective: Present how to use an *infrastructure as code* tool to deploy a Kubernetes cluster.

This presentation will consist of:

- Explaining the technologies involved
- Detailing the Terraform configuration used
- Seeing the results in a demo

About us

Clément Contini

- Software Engineering degree from ÉTS
- I work on Cloud Infrastructure at CloudOps and cloud.ca

CloudOps

- Cloud Infrastructure experts since 2005
- Design, build and manage public, private and hybrid cloud solutions

cloud.ca

- Infrastructure as a service platform, based in Montreal
- Launched in August 2014

Terraform

Infrastructure as code tool:

- Cross-platform
- Configurations can be versioned and reused
- Generates execution plans
- A cloud.ca provider is available

Similar tools:

- OpenStack Heat
- AWS CloudFormation

Figure 1: Terraform logo

Kubernetes

- Container orchestration system
- Originates from Google's Borg

Similar tools:

- Docker Swarm
- $\sim$ Rancher
- $\sim$ HashiCorp Nomad

Figure 2: Kubernetes logo

Demonstration

The demo will consist in deploying a Kubernetes cluster using the *kubeadm* toolkit.

We will deploy:

- a cloud.ca environment with a VPC, networks and all the necessary virtual machines
- A Kubernetes master and several workers on the VMs
- Automatically start a Vault deployment in the cluster with 3 replicas

Let's start the demo, and I will explain the configuration while it's building the cluster.

Demonstration

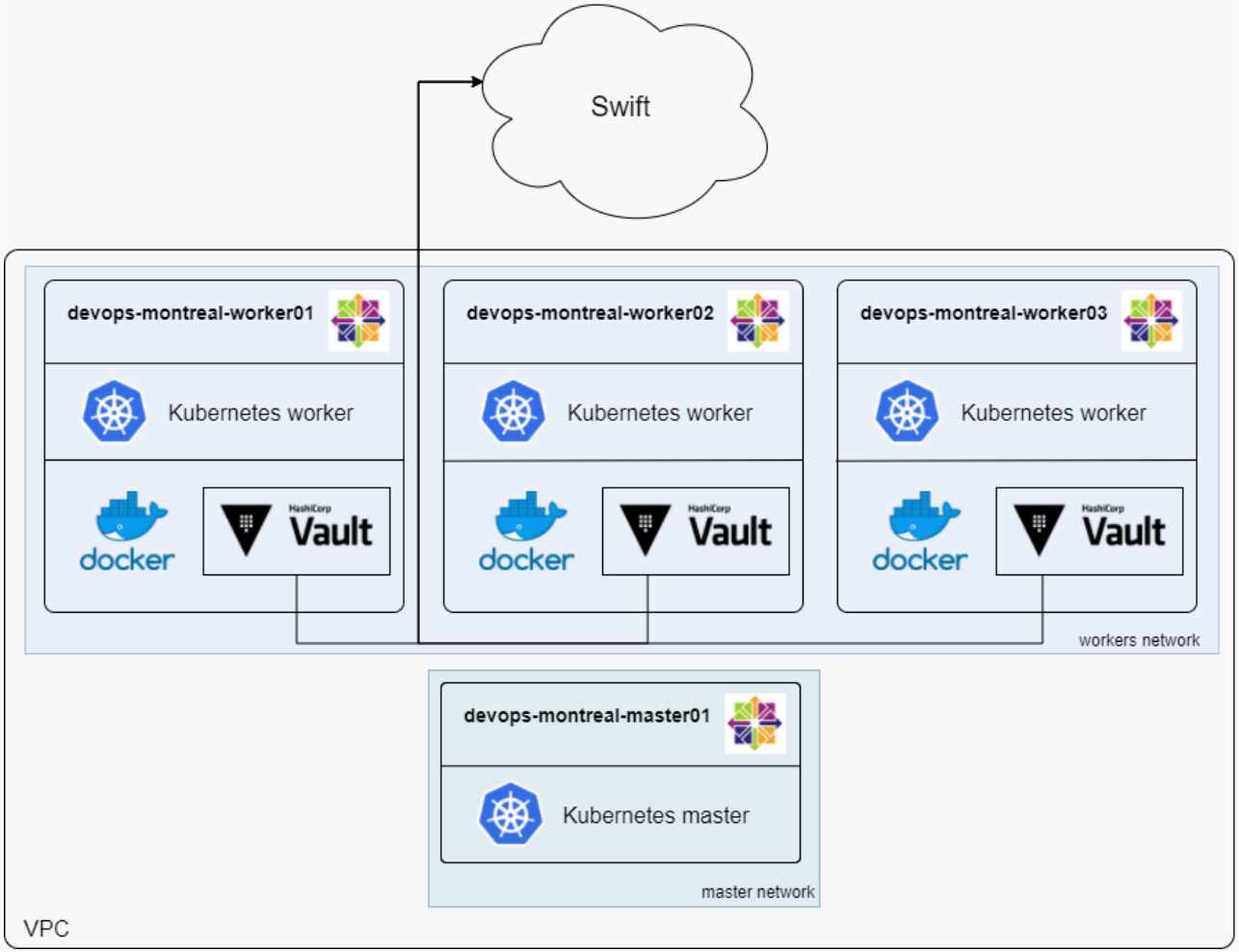

Figure 3: Deployment of the Kubernetes cluster

Environment

We first create an environment in cloud.ca for our deployment.

We create:

- a cloudca_environment
- a cloudca_vpc
- 2 cloudca_public_ip

```
1 resource "cloudca_environment" "kubernetes" {
2   service_code      = "${var.service_code}"
3   organization_code = "${var.organization_code}"
4   name              = "${format("%s-env", var.prefix)}"
5   description       = "Environment for a Kubernetes cluster"
6   admin_role        = ["${var.admin}"]
7   read_only_role    = ["${var.read_only}"]
8 }
```

Listing 1: Environment definition

Variables

Some variables need to be defined (for example the api key) and some can simply use the default. The values are provided in the *terraform.tfvars* file.

```
service_code      = "compute-qc"
organization_code = "system"
admin             = [ "ccontini" ]
read_only         = [ ]
prefix            = "devops-montreal"
```

Listing 2: Variables assignment

Variables

We need to declare the variables that will help us configure the environment for our use case:

- service_code
- organization_code
- prefix
- admin
- read_only

```
1 variable "service_code" {}
2 variable "organization_code" {}
3 variable "admin" { type = "list" }
4 variable "read_only" { type = "list" }
5 variable "prefix" {}
```

Listing 3: Variables definition

Master

We then declare all resources required to start the Kubernetes master:

- A network and its associated set of ACLs
- An instance
- A port forwarding rule

```
1 resource "cloudca_network" "master" {
2   environment_id   = "${cloudca_environment.kubernetes.id}"
3   name             = "${format("%s-network-master", var.
      ↪ prefix)}"
4   description      = "Network for the master nodes"
5   vpc_id           = "${cloudca_vpc.kubernetes.id}"
6   network_offering = "Standard Tier"
7   network_acl_id   = "${cloudca_network_acl.master.id}"
8 }
```

Listing 4: Network definition for the master node

Master

```
1  resource "cloudca_instance" "master_node" {
2    environment_id          = "${cloudca_environment.kubernetes
       ↪  .id}"
3    name                    = "${format("%s-master01", var.
       ↪  prefix)}"
4    network_id              = "${cloudca_network.master.id}"
5    template                = "${var.template_name}"
6    compute_offering        = "${var.default_offering}"
7    cpu_count               = "${var.master_vcpu}"
8    memory_in_mb            = "${var.master_ram}"
9    root_volume_size_in_gb = "${var.master_disk}"
10   user_data               = "${data.template_file.cloudinit.
       ↪  rendered}"
11 }
```

Listing 5: Master node instance definition

Master

The master node is configured using a cloudinit script. It will:

- Create a user
- Install docker and Kubernetes
- start a bootstrap script to initialize the master

```
1  data "template_file" "cloudinit" {
2    template = "${file("templates/cloudinit.tpl")}"
3
4    vars {
5      public_key = "${replace(file("./id_rsa.pub"), "\n", "")}
         ↪ "
6      username   = "${var.username}"
7    }
8  }
```

Listing 6: Cloudinit template interpolation

Master

```
1 resource "cloudca_port_forwarding_rule" "
    ↪ management_master_ssh" {
2   environment_id      = "${cloudca_environment.kubernetes.id}
      ↪ "
3   public_ip_id        = "${cloudca_public_ip.master_ip.id}"
4   public_port_start   = 2200
5   private_ip_id       = "${cloudca_instance.master_node.
      ↪ private_ip_id}"
6   private_port_start  = 22
7   protocol            = "TCP"
```

Listing 7: Master node port forwarding rule

Once the port forwarding rule is created, we use it to provision the node with bootstrap scripts and a deployment manifest that will be applied when the Kubernetes cluster starts.

Master

```
1   connection {
2     type        = "ssh"
3     user        = "${var.username}"
4     private_key = "${file("./id_rsa")}"
5     host        = "${cloudca_public_ip.master_ip.ip_address}
          ↪ "
6     port        = 2200
7   }
8
9   provisioner "file" {
10    content     = "${data.template_file.bootstrap_master.
          ↪ rendered}"
11    destination = "/home/${var.username}/bootstrap.sh"
12  }
13
14  provisioner "file" {
15    source      = "manifests"
16    destination = "/home/${var.username}/manifests"
17  }
```

Listing 8: Provisioning of the master node

Workers

We create the workers, the same way we created the master, with:

- A network and its associated set of ACLs
- Several instance (we will use the *count* keyword)
- A port forwarding rule

```
1  resource "cloudca_instance" "worker_nodes" {
2    environment_id          = "${cloudca_environment.kubernetes
       ↪  .id}"
3    name                    = "${format("%s-worker%02d", var.
       ↪  prefix, count.index + 1)}"
4    network_id              = "${cloudca_network.worker.id}"
5    template                = "${var.template_name}"
6    compute_offering        = "${var.default_offering}"
7    cpu_count               = "${var.worker_vcpu}"
8    memory_in_mb            = "${var.worker_ram}"
9    root_volume_size_in_gb = "${var.worker_disk}"
10   count                   = "${var.worker_count}"
11   user_data               = "${data.template_file.cloudinit.
       ↪  rendered}"
12 }
```

Listing 9: Workers instance definition

Vault deployment

Let's check what was deployed on Kubernetes.

Some alternatives

Some alternative solutions that could be use to create and manage a Kubernetes cluster:

- We could use the Kubernetes Terraform provider to manage the resources inside the Kubernetes cluster.
- We could use Rancher to deploy Kubernetes on cloud.ca using a GUI

cloud.ca plans to integrate more tightly with Kubernetes to be able to create resources for the containers directly with Kubernetes:

- Volumes
- Load-balancing rules

Questions

Any questions ?

Thanks for attending this meetup !

The source code is available at:
`https://github.com/vilisseranen/terraform-kubeadm`.