|  |  |
| --- | --- |
| Name : | Pierre-Antoine RidderstapVadym TkachenkoYanina PetrovaGabriel Rafael |
| Date : | 12/06/2023 |



Kubernetes Cluster Upgrade Without Disruption

Technical Document

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Revision Table | Description | Date | Author | Checked by | Approved by |
|  |  |  |  |  |  |

Table of Contents

[Introduction 3](#_Toc136942248)

[Deployment 4](#_Toc136942249)

[Prerequisite configuration 4](#_Toc136942250)

[Cluster deployment 4](#_Toc136942251)

[Upgrade 6](#_Toc136942252)

# Introduction

This technical document describes in detail how deployment and upgrade of the Kubernetes cluster are done using Kubespray. In the deployment section, the Terraform configuration for creating EC2 instances is discussed, followed by the upgrade process.

# Deployment

## Prerequisite configuration

First, to set up the Kubernetes cluster you need to clone the [Kubespray repository](https://github.com/kubernetes-sigs/kubespray), followed by installation of Anible and its dependencies. To do this, run the following command from cloned repository’s root folder:

**pip install -r requirements.txt**

Then, install latest version of Terraform by HashiCorp. Terraform must be used to assist provide our infrastructure, because Kubespray does not automatically construct virtual machines. We begin by setting up an SSH key pair on AWS for Ansible.

Once the keypair is generated execute following commands:

 **cd [repository]/contrib/terraform/aws/**

**cp credentials.tfvars.example credentials.tfvars**

**nano credentials.tfvars**

## Cluster deployment

Modify the **credentials.tfvars** file, filling in the AWS\_ACCESS\_KEY\_ID, AWS\_SECRET\_ACCESS\_KEY and the private\_key.pem name generated previously. Contents of your **credentials.tfvars** should look similar to *Figure 1*:



Figure 1: Modifying credentials.tfvars

Next step would be to edit the **terraform.tfvars**, where you should specify the structure of the cluster and number of instances for each Kubernetes component. In our case, the following configuration was specified:

///

***#Global Vars***

***aws\_cluster\_name = "devtest"***

***#VPC Vars***

***aws\_vpc\_cidr\_block = "10.250.192.0/18"***

***aws\_cidr\_subnets\_private = ["10.250.192.0/20", "10.250.208.0/20"]***

***aws\_cidr\_subnets\_public = ["10.250.224.0/20", "10.250.240.0/20"]***

***#Bastion Host***

***aws\_bastion\_num = 1***

***aws\_bastion\_size = "t3.small"***

***#Kubernetes Cluster***

***aws\_kube\_master\_num = 3***

***aws\_kube\_master\_size = "t3.medium"***

***aws\_kube\_master\_disk\_size = 50***

***aws\_etcd\_num = 3***

***aws\_etcd\_size = "t3.medium"***

***aws\_etcd\_disk\_size = 50***

***aws\_kube\_worker\_num = 4***

***aws\_kube\_worker\_size = "t3.medium"***

***aws\_kube\_worker\_disk\_size = 50***

***#Settings AWS ELB***

***aws\_nlb\_api\_port = 6443***

***k8s\_secure\_api\_port = 6443***

***default\_tags = {***

 ***# Env = "devtest"***

 ***# Product = "kubernetes"***

***}***

***inventory\_file = "../../../inventory/hosts"***

***///***

Follow this by creating the terraform plan and applying it to create instances in the AWS:

**terraform init**

**terraform plan -out mysuperplan -var-file=credentials.tfvars**

Once the plan has been applied, we can use Kubespray to deploy the cluster:

Next, load the SSH key, which were created in AWS earlier on. First, create a file (in our case, it will be located at ~/.ssh/user/kubespray.pem) and paste the private part of the key created at AWS there.

***ssh-add ~/.ssh/user/kubespray.pem***

To create the Kubernetes cluster, from repository’s root folder run:

***ansible-playbook -i ./inventory/hosts ./cluster.yml -e ansible\_user=admin -e kube\_version=[version] -b --become-user=root***

You might require a different value for **ansible\_user** depending on the image used by AWS for creating instances. You can check for specific value from AWS EC2 management console.

* The playbook above performs a list of tasks, such as:
* Configure openssh-server on bastion
* Deploys the container engine
* Sets up etcd and configures certificates
* Configures networking and Kubernetes control plane

Once the playbook has finished running, Kubernetes cluster should be successfully deployed and can be accessed by connecting via ssh to one of the master instances.

# Upgrade



Figure 2 Process Diagram

Following the deployment instructions should result in a Kubernetes cluster, which is up and running, as shown in the Figure 2. Once a new version of Kubernetes is out, you can upgrade your cluster to a new version using Ansible.

First, make sure to perform a “**git checkout”** for a newest version of Kubespray repository. Then you can upgrade the cluster to a new Kubernetes version by running a following command and specifying the ***kube\_version***:

***ansible-playbook -i ./inventory/hosts ./upgrade-cluster.yml -e ansible\_user=admin -e kube\_version=[version] -b --become-user=root***

This performs a graceful upgrade of the cluster, when each component is laid down in a fixed order and upgraded to a new version. Ansible performs a following list of tasks contained in the ***upgrade-cluster.yml*** playbook:

* Prepares nodes for upgrade
* Handles upgrades to master nodes to maintain backward compatibility
* Upgrade calico and external cloud provider on all masters, calico-rrs, and nodes
* Finally handle worker upgrades