KubeNow Documentation

Release 0.2.1

mcapuccini

May 04, 2017

Getting Started

Welcome to KubeNow's documentation! This is a place where we aim to help you to provision Kubernetes, the KubeNow's way. If you are new to Kubernetes, and to cloud computing, this is going to take a while to grasp the first time. Luckily, once you get the procedure, it's going to be very quick to spawn your clusters.

CHAPTER 1

Prerequisites

Install provisioning tools

There are some software pieces that you need to install on your local machine, in order to provision Kubernetes with KubeNow:

- Terraform (0.9.4 or higher) to fire-up the virtual infrastructure on the host cloud
- Ansible (2.2.0.0 or higher) to provision the VMs (e.g. install and configure networking, reverse proxy etc.)
- *Python modules*: `dnspython`, `jmespath`, `apache-libcloud`, `shade`. You can istall them using pip:

```
sudo pip install dnspython jmespath apache-libcloud shade
```

Get KubeNow

KubeNow 0.2.1 is distributed via GitHub:

```
git clone https://github.com/mcapuccini/KubeNow.git
cd KubeNow
git checkout 0.2.1
```

All of the commands in this documentation are meant to be run in the KubeNow directory.

CHAPTER 2

Bootstrap Kubernetes on a host cloud

This step is slightly different for each host cloud. Here you find a section for each of the supported providers.

Sections

Bootstrap on OpenStack

Prerequisites

In this section we assume that:

- You have downloaded and sourced the OpenStack RC file for your tenancy: `source project-openrc.sh` https://docs.openstack.org/user-guide/common/cli-set-environment-variables-using-openstack-rc.html#download-and-source-the-openstack-rc-file
- You have a floating IP quota that allows to allocate a public IP for each master and edge node (at least 2 in total)
- You installed the glance command-line client in your local machine: https://docs.openstack.org/user-guide/common/cli-install-openstack-command-line-clients.html

Import the KubeNow image (only the first time you are deploying)

The first time you are going to deploy KubeNow, you'll have to import its cloud image. This considerably speeds up the following bootstraps, as all of the required software will already be installed on the instances.

You can import the latest image build by running the following Ansible playbook:

```
ansible-playbook playbooks/import-openstack-image.yml
```

If everything goes well, you will see the new image in the OpenStack web interface (Compute > Images). As an alternative, you can check that the image is present using the OpenStack command line client:

```
glance image-list
```

Bootstrap Kubernetes

Now we are going to provision the required virtual infrastructure in OpenStack using Terraform. This procedure will inject enough information in each instance, to independently provision itself.

Start by creating a `terraform.tfvars` file. There is a template that you can use for your convenience: `mv terraform.tfvars.os-template terraform.tfvars`. In this configuration file you will need to set:

Cluster configuration

- **cluster_prefix**: every resource in your tenancy will be named with this prefix
- **kubenow_image**: name of the image that you previously imported using Ansible
- **ssh_key**: path to your public ssh-key to be used (for ssh node access)
- **floating_ip_pool**: a floating IP pool label
- **external_network_uuid**: the uuid of the external network in the OpenStack tenancy
- **dns_nameservers**: (optional, only needed if you want to use other dns-servers than default 8.8.8.8 and 8.8.4.4)
- **kubeadm_token**: a token that will be used by kubeadm, to bootstrap Kubernetes. You can run generate_kubetoken.sh to create a valid one.

If you are wondering where you can correctly get a *floating_ip_pool* and an *external_network_uuid*, then one way is to inquiry your OpenStack settings. You can easily and quickly achieve this by installing one of the OpenStack command-line interface called *nova* . Installation occurs via `pip`, however please refer to the offical OpenStack documentation in depth details.

Once *nova* is installed, run the following command:

```
nova network-list
```

Depending on your tenancy settings you should get a similar output:

```
+--------------------------------------+----------------+------+
| ID                                   | Label          | Cidr |
+--------------------------------------+----------------+------+
| 5f274562-89b6-4ab2-a18f-94b159b0b85d | internal       | -    |
| d9384930-baa5-422b-8657-1d42fb54f89c | net_external   | -    |
+--------------------------------------+----------------+------+
```

Thus in this specific case the above mentioned fields will be set as following:

```
floating_ip_pool: net_external
external_network_uuid: d9384930-baa5-422b-8657-1d42fb54f89c
```

Master configuration

- **master_flavor**: an instance flavor for the master

Node configuration

- **node_count**: number of Kubernetes nodes to be created (no floating IP is needed for these nodes)
- **node_flavor**: an instance flavor name for the Kubernetes nodes
- **node_flavor_id**: this is an alternative way of specifying the instance flavor (optional, please set this only if **node_flavor** is empty)

Edge configuration

- **edge_count**: number of egde nodes to be created
- **edge_flavor**: an instance flavor for the edge nodes

If you are wondering yet again where you can fetch correct flavor label names then no worries, you are not being a stranger here. The nova command-line interface will come in handy. Just run the following command:

```
nova flavor-list
```

Depending on your tenancy settings you should get a similar output:

```
+--------+------------+-----------+------+-----------+------+-------+-------------+---
↪-------+
| ID     | Name       | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor |␣
↪Is_Public|
+--------+------------+-----------+------+-----------+------+-------+-------------+---
↪-------+
| 8c7ef1 | ssc.tiny   | 512       | 1    | 0         |      | 1     | 1.0         |␣
↪True     |
| 8d7ef2 | ssc.small  | 2048      | 20   | 0         |      | 1     | 1.0         |␣
↪True     |
| 8e7ef3 | ssc.medium | 4096      | 40   | 0         |      | 2     | 1.0         |␣
↪True     |
| 8f7ef4 | ssc.large  | 8192      | 80   | 0         |      | 4     | 1.0         |␣
↪True     |
| 8g7ef5 | ssc.xlarge | 16384     | 160  | 0         |      | 8     | 1.0         |␣
↪True     |
+--------+------------+-----------+------+-----------+------+-------+-------------+---
↪-------+
```

Based how many resources your applications require, then you may want to select the nodes' flavor accordingly. E.g.:

```
master_flavor: ssc.medium
edge_flavor:   ssc.medium
node_flavor:   ssc.large
```

Once you are done with your settings you are ready to bootstrap the cluster using Terraform:

```
terraform get openstack # get required modules (only the first time you deploy)
terraform apply openstack # deploy the cluster
```

While waiting for the bootstrap process to complete, it is important to avoid to abruptingly stop it (e.g by pressing `Ctrl+C` in the console), otherwise chances are that your local file terraform.tfstate will likely be corrupted which can lead to technical complication in successfully deploying the cluster.

If everything goes well, something like the following message will be printed:

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

To verify that each node connected to the master you can run:

```
ansible master -a "kubectl get nodes"
```

If all of the nodes are not yet connected and in the Ready state, wait a minute and try again. Keep in mind that booting the instances takes a couple of minutes.

Good! Now you have a minimal Kubernetes cluster up and running, and you are ready to *install the KubeNow core components*.

Bootstrap on Google Cloud (GCE)

Prerequisites

In this section we assume that:

- You have enabled the Google Compute Engine API: API Manager > Library > Compute Engine API > Enable
- You have created and downloaded a service account file for your GCE project: Api manager > Credentials > Create credentials > Service account key

Import the KubeNow image (only the first time you are deploying)

The first time you are going to deploy KubeNow, you'll have to import its cloud image. This considerably speeds up the following bootstraps, as all of the required software will already be installed on the instances.

You can import the latest image build by running the following Ansible playbook:

```
ansible-playbook -e "credentials_file_path=/full/path/to/service_account.json"␣
↪playbooks/import-gce-image.yml
```

If everything goes well, you will see the new image in the GCE web interface (Compute Engine > Images). As an alternative, you can check that the image is present using the Google Cloud command line client:

```
gcloud compute images list
```

Bootstrap Kubernetes

Now we are going to provision the required virtual infrastructure in Google Cloud using Terraform. This procedure will inject enough information in each instance, to independently provision itself.

Start by creating a `terraform.tfvars` file. There is a template that you can use for your convenience: `mv terraform.tfvars.gce-template terraform.tfvars`. In this configuration file you will need to set:

Cluster configuration

- **cluster_prefix**: every resource in your project will be named with this prefix (the name must match `(?:[a-z](?:[-a-z0-9]{0,61}[a-z0-9])?)`, e.g. "kubenow-image")
- **kubenow_image**: name of the image that you imported using Ansible
- **kubeadm_token**: a token that will be used by kubeadm, to bootstrap Kubernetes. You can run *generate_kubetoken.sh* to create a valid one.
- **ssh_key**: path to your public ssh-key to be used (for ssh node access)

Google credentials

- **gce_credentials_file**: path to your service account file
- **gce_region**: the zone for your project (e.g. `europe-west1-b`)
- **gce_project**: your project id

Master configuration

- **master_flavor**: an instance flavor for the master (e.g. `n1-standard-1`)
- **master_disk_size**: master disk size in GB

Node configuration

- **node_count**: number of Kubernetes nodes to be created
- **node_flavor**: an instance flavor for the Kubernetes nodes (e.g. `n1-standard-1`)
- **node_disk_size**: nodes disk size in GB

Edge configuration

- **edge_count**: number of egde nodes to be created
- **edge_flavor**: an instance flavor for the edge nodes (e.g. `n1-standard-1`)
- **edge_disk_size**: edges disk size in GB

Once you are done with your settings you are ready to bootstrap the cluster using Terraform:

```
terraform get gce # get required modules (only the first time you deploy)
terraform apply gce # deploy the cluster
```

If everything goes well, something like the following message will be printed:

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

To verify that each node connected to the master you can run:

```
ansible master -a "kubectl get nodes"
```

If all of the nodes are not yet connected and in the Ready state, wait a minute and try again. Keep in mind that booting the instances takes a couple of minutes.

Good! Now you have a minimal Kubernetes cluster up and running, and you are ready to *install the KubeNow core components*.

Bootstrap on Amazon Web Services (EC2)

Prerequisites

In this section we assume that:

- You have an IAM user along with its *access key* and *security credentials* (http://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_create.html)

Bootstrap Kubernetes

Now we are going to provision the required virtual infrastructure in AWS (Amazon Web Services) using Terraform. This procedure will inject enough information in each instance, to independently provision itself.

Start by creating a `terraform.tfvars` file. There is a template that you can use for your convenience: `mv terraform.tfvars.aws-template terraform.tfvars`. In this configuration file you will need to set:

Cluster configuration

- **cluster_prefix**: every resource in your tenancy will be named with this prefix
- **kubenow_image**: name of the KubeNow image that you want to use
- **kubeadm_token**: a token that will be used by kubeadm, to bootstrap Kubernetes. You can run *generate_kubetoken.sh* to create a valid one.
- **ssh_key**: path to your public ssh-key to be used for ssh node access (e.g. `~/.ssh/id_rsa.pub`)
- **aws_region**: the region where your cluster will be bootstrapped (e.g. `eu-west-1`)
  - **Warning:** the image that you previously selected has to be available in this region
- **availability_zone**: an availability zone for your cluster (e.g. `eu-west-1a`)

Credentials

- **aws_access_key_id**: your access key id
- **aws_secret_access_key**: your secret access key

Master configuration

- **master_instance_type**: an instance type for the master (e.g. `t2.medium`)
- **master_disk_size**: edges disk size in GB

Node configuration

- **node_count**: number of Kubernetes nodes to be created
- **node_instance_type**: an instance type for the Kubernetes nodes (e.g. `t2.medium`)
- **node_disk_size**: edges disk size in GB

Edge configuration

- **edge_count**: number of egde nodes to be created
- **edge_instance_type**: an instance type for the edge nodes (e.g. `t2.medium`)
- **edge_disk_size**: edges disk size in GB

Network configuration (optional)

Define the following variables if you want KubeNow to be deployed with an existing VPC and subnet, or security group(s).

- **vpc_id**: ID of an existing VPC (to be defined together with `subnet_id`)
- **subnet_id**: ID of an existing subnet (to be defined together with `vpc_id`)
- **additional_sec_group_ids**: list of one or many existing security groups

Once you are done with your settings you are ready to bootstrap the cluster using Terraform:

```
terraform get aws # get required modules (only the first time you deploy)
terraform apply aws # deploy the cluster
```

If everything goes well, something like the following message will be printed:

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

To verify that each node connected to the master you can run:

```
ansible master -a "kubectl get nodes"
```

If all of the nodes are not yet connected and in the Ready state, wait a minute and try again. Keep in mind that booting the instances takes a couple of minutes. **Warning** if you are using the free tier, the cluster will take a little bit more to bootstrap (~5 minutes).

Good! Now you have a minimal Kubernetes cluster up and running, and you are ready to *install the KubeNow core components*.

CHAPTER 3

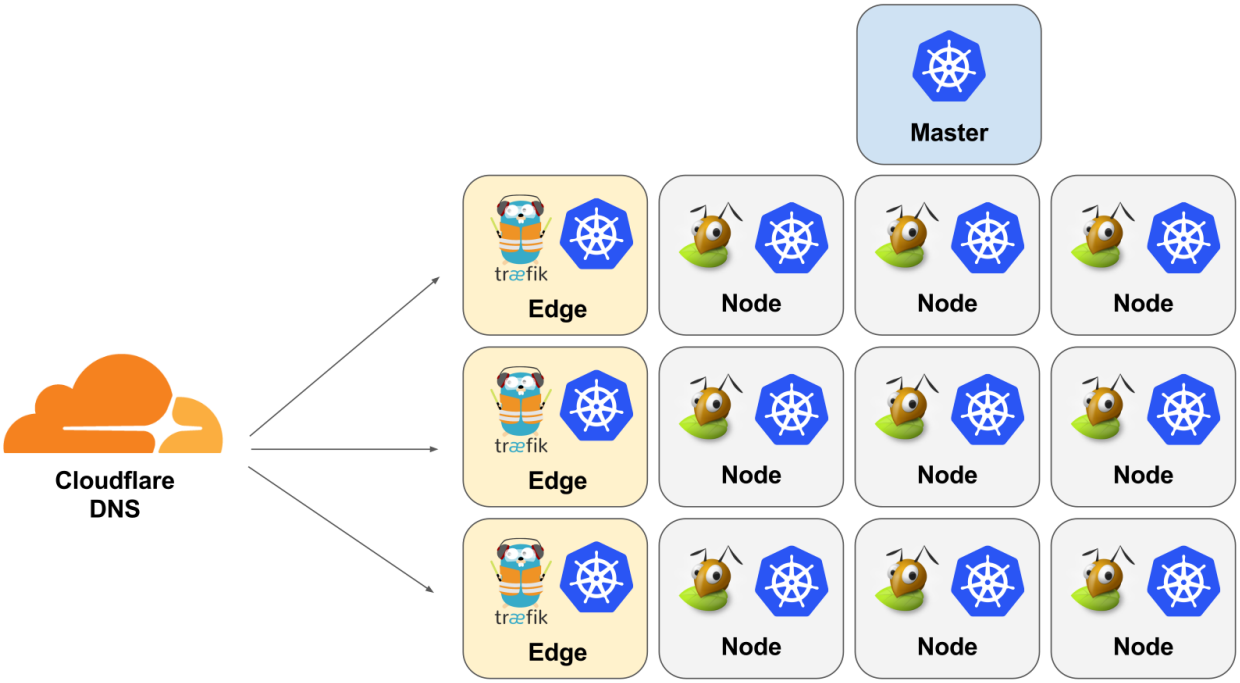
Install core components

At this point you should have a core Kubernetes cluster up and running, on your cloud provider. The Kubernetes cluster that you deploy in the previous step is not ready to use, as it lacks the overlay network service that will allow your containers to communicate. KubeNow comes with an install-core.yml playbook that installs the overlay network along with some other useful components.

The core components include:

- Weave overlay network (tag: *minimal*)
- Traefik HTTP reverse proxy and load balancer (**only on the edge nodes**, tag: `traefik`)
- Cloudflare Cloudflare dynamic DNS configuration (tag: `cloudflare`)
- GlusterFS GlusterFS distributed files system (**on every node but the master and the edge**, tag: `glusterfs`).
  - The playbook also configures a persistent volume called `shared-volume` that points to the GlusterFS endpoints.

Pro tip: you can use the tags to install or skip certain components while running install-core.yml. For more informations please refer to the Ansible tags documentation.



This kind of deployment is particularly convenient, as only the master node, and the edge nodes (that run Traefik) need to be associated to public IPs (which can be scarce). Therefore, the end user will access the microservices running in the Kubernetes nodes, through an edge node that will act as a reverse proxy. The Cloudflare service will loadbalance the requests over the edge nodes.

Cloudflare account configuration

In this stack, the edge nodes act as getaways to access some services running in the Kubernetes nodes. Typically, you want the end user to access your services through a domain name. One option is to manually configure the DNS services, for a domain name, to load balance the requests among the edge nodes. However, doing this for each deployment can be tedious, and prone to configuration errors. Hence, we recommend to sign up for a free account on Cloudflare, that you can use as dynamic DNS service for your domain name.

Once you have your Cloudflare account, start by creating a `playbooks/roles/cloudflare/vars/conf.yml` configuration file. There is a template file that you can use for your convenience:

```
mv playbooks/roles/cloudflare/vars/conf.yml.template playbooks/roles/cloudflare/vars/
↪conf.yml
```

In this configuration file you need to set:

- **cf_mail**: the mail that you used to register your Cloudflare account
- **cf_token**: an authentication token that you can generate from the Cloudflare web interface
- **cf_zone**: a zone that you created in your Cloudflare account. This typically matches your domain name (e.g. somedomain.com)
- **cf_subdomain** (optional): you can set a subdomain name for this cluster, if you don't want to use the whole domain for this purpose

Deploy the stack

Once you are done with the Cloudflare account configuration, you can deploy the stack via Ansible:

```
ansible-playbook playbooks/install-core.yml
```

To make sure that each service is running you can run the following command:

```
ansible-playbook playbooks/infra-test.yml
```

It may happen that the test will return the following error:

```
FAILED! => {"failed": true, "msg": "An unhandled exception occurred while running
the lookup plugin 'dig'. Error was a <class 'ansible.errors.AnsibleError'>, original␣
↪message:
Can't LOOKUP(dig): module dns.resolver is not installed"}
```

If this is the case, then as it can be deduced by the error message, the dns python module is missing. Installing the module `dnspython` via `pip` (i.e. `pip install dnspython`) will fix the glitch and allow the test to perform correctly. See the issue reported in the Ansible GitHub discussion for in depth details.

CHAPTER 4

Deploy your first application

In this guide we are going to deploy a simple application: cheese-deployent. This deployment defines 3 services with a 2 replication factor. Traefik will load balance the requests among the replicas in the Kubernetes nodes. For more details about the cheese deployment, please refer to: https://docs.traefik.io/user-guide/kubernetes.

Start by substituting `domain_name` with `yourdomain.com` in `cheese-deployent.yml` (where *yourdomain.com* is the domain that points to the edge nodes, through CloudFlare):

```
sed -i -e 's/domain_name/yourdomain.com/g' examples/cheese-deployment.yml
```

Now, copy the `cheese-deployent.yml` file into the master node:

```
ansible master -m copy -a "src=examples/cheese-deployment.yml dest=/home/ubuntu"
```

Finally, deploy the application using kubectl:

```
ansible master -a "kubectl apply -f /home/ubuntu/cheese-deployment.yml"
```

If everything goes well you should see some front-ends and back-ends showing up in the Traefik UI, and you should be able to access the services at:

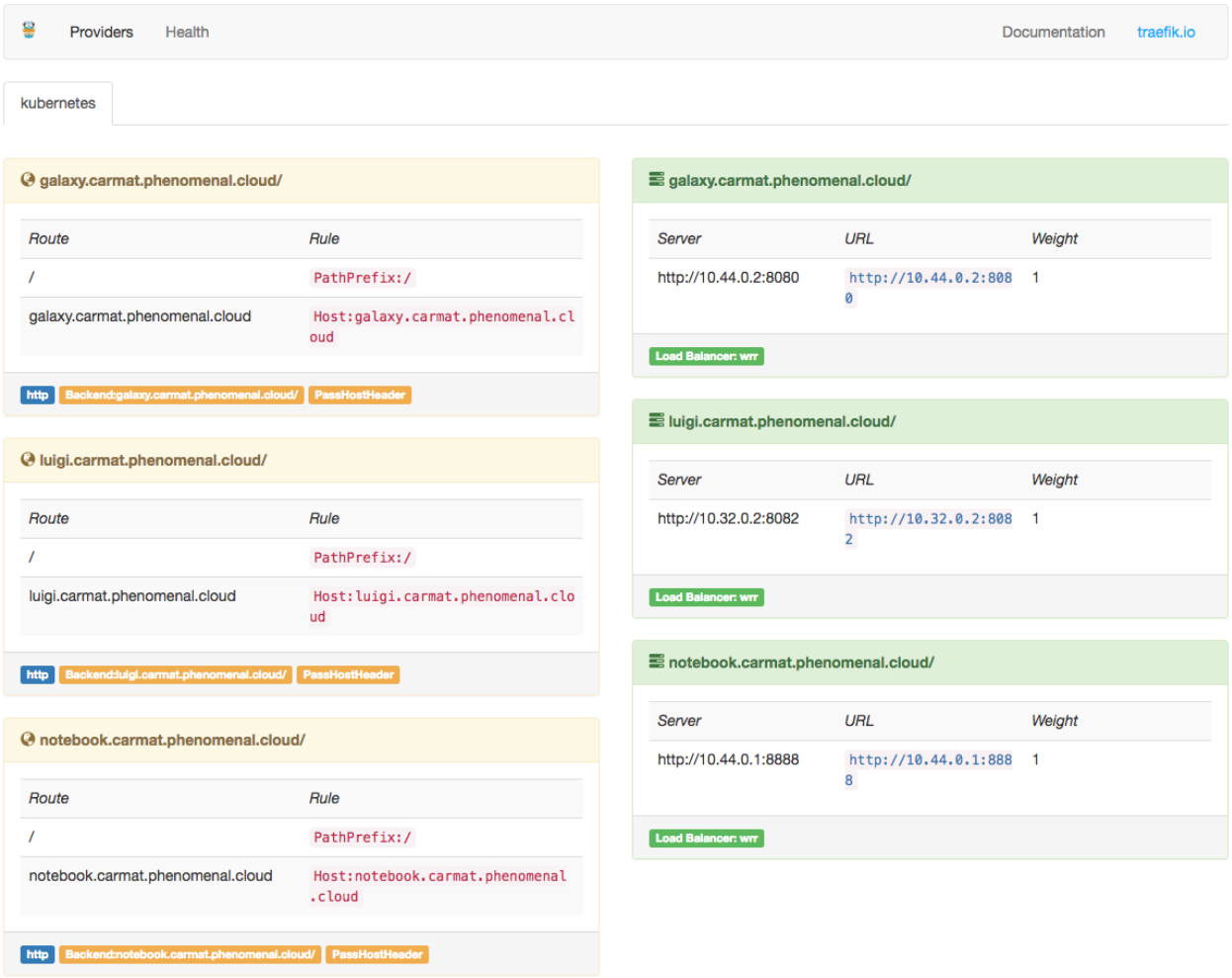
- http://stilton.yourdomain.com
- http://cheddar.yourdomain.com
- http://wensleydale.yourdomain.com

One simply and quick way to access the Traefik UI is to tunneling via ssh to one of the edge nodes with the following command:

```
ssh -N -f -L localhost:8080:localhost:8080 ubuntu@<edge1-floating-ip>
ssh -N -f -L localhost:8081:localhost:8081 ubuntu@<edge2-floating-ip>
...
```

If you are wondering how to get the *edge-floating-ip*, then open the *inventory* file which is created after having spun up the cluster in the KubeNow folder

By visitng for instance your localhost:8080 in your browser you should be displayed with the following UI:



where on the left highlighted in yellow you can find your deployed frontend apps whereas on the right highlighted in green the backend services are listed

CHAPTER 5

Clean after yourself

Cloud resources are typically pay-per-use, hence it is good to release them when they are not used. Here we show how to destroy a KubeNow cluster.

To release the resources, please run:

```
terraform destroy <cloud-provider>
```

`<cloud-provider>` can be “openstack”, “gce” or “aws”.

To delete the Clouflare DNS records, please run:

```
ansible-playbook playbooks/clean-cloudflare.yml
```

Warning: if you create a new cluster before deleting the DNS records, the Ansible inventory will be replaced and you will have to delete the records manually.

CHAPTER 6

Terraform troubleshooting

Since Terraform applies changes incrementally, when there is a minor issue (e.g. network timeout) it's sufficient to rerun the command. However, here we try to collect some tips that can be useful when rerunning the command doesn't help.

Contents

Corrupted Terraform state

Due to network issues, Terraform state files can get out of synch with your infrastructure, and cause problems. Since Terraform apply changes increme. A possible way to fix the issue is to destroy your nodes manually, and remove all state files and cached modules:

```
rm -R .terraform/
rm terraform.tfstate
rm terraform.tfstate.backup
```

CHAPTER 7

OpenStack troubleshooting

Contents

Console logs on OpenStack

Can't get the status from the nodes with `ansible master -a "kubectl get nodes"`? The nodes might not have started all right. Checking the console logs with nova could help.

List node IDs, floating IPs etc.:

```
nova list
```

Show console output from node of interest:

```
nova console-log <node-id>
```

Missing DomainID or DomainName to authenticate by Username

When running terraform and/or packer you may be prompted with the following error:

```
You must provide exactly one of DomainID or DomainName to authenticate by Username
```

If this is the case, then setting either `OS_DOMAIN_ID` or `OS_DOMAIN_NAME` in your environment should fix the issue. For further information, please refer to this document: https://www.terraform.io/docs/providers/openstack/index.html.

CHAPTER 8

Kubernetes troubleshooting

Here you can find some frequently used commands to list the status and logs of kubernetes. If this doesn't help, please refer to http://kubernetes.io/docs.

Contents

List kubernetes pods

```
# If you are logged into the node via SSH:
kubectl get pods --all-namespaces

# With Ansible from your local computer:
ansible master -a "kubectl get pods --all-namespaces"
```

Describe status of a specific pod

```
# If you are logged into the node via SSH:
kubectl describe pod <pod id> --all-namespaces

# With Ansible from your local computer:
ansible master -a "kubectl describe pod <pod id> --all-namespaces"
```

Get the kubelet service log

```
# If you are logged into the node via SSH:
sudo journalctl -r -u kubelet

#  With Ansible from your local computer:
ansible master -a "journalctl -r -u kubelet"
```

CHAPTER 9

More troubleshooting

Contents

SSH connection errors

In case of SSH connection errors:

- Make sure to add your private SSH key to your local keyring:

```
ssh-add ~/.ssh/id_rsa
```

- Make sure that port 22 is allowed in your cloud provider security settings.

If you still experience problems, checking out the console logs from your cloud provider could help.

Figure out hostnames and IP numbers

The bootstrap step should create an Ansible inventory list, which contains hostnames and IP addresses:

```
cat inventory
```

CHAPTER 10

Image build instructions

KubeNow uses prebuilt images to speed up the deployment. Even if we provide some prebuilt images you may need to build it yourself. The image building process involves the use of Packer (0.12.3 or higher), so you will need to install it on your workstation. The procedure is slightly different for each host cloud. Here you find a section for each of the supported providers.

Sections

Build KubeNow image on OpenStack

Prerequisites

In this section we assume that:

- You have downloaded and sourced the OpenStack RC file for your tenancy: `source project-openrc.sh`

Every OpenStack installation it's a bit different, and the RC file you get to download from the interface might be incomplete. Please make sure that all of these environment variables are set in the RC file:

```
OS_USERNAME
OS_PASSWORD
OS_AUTH_URL
OS_USER_DOMAIN_ID
OS_DOMAIN_ID
OS_REGION_NAME
```

```
OS_PROJECT_ID
OS_TENANT_ID
OS_TENANT_NAME
OS_AUTH_VERSION
```

- You created a private network with a router that connects it to the external network
- You have a Ubuntu 16.04 (Xenial) image in your tenancy
- You set up the default security group to allow ingress traffic on port 22

Build the KubeNow image

Start by creating a `packer-conf.json` file. There is a template that you can use for your convenience: `mv packer-conf.json.os-template packer-conf.json`. In this configuration file you will need to set:

- **image_name**: the name of the image that will be created after the build (e.g. "KubeNow")
- **source_image_name**: a Ubuntu Xenial image, already present in your tenancy
- **network**: the ID of a private network, already present in your tenancy
- **flavor**: an instance flavor to use, in order to build the image
- **floating_ip_pool**: a floating IP pool

Once you are done with your settings you are ready to build KubeNow using Packer:

```
packer build -var-file=packer-conf.json packer/build-openstack.json
```

If everything goes well, you will see the new image in the OpenStack web interface (Compute > Images). As an alternative, you can check that the image is present using the OpenStack command line client:

```
glance image-list
```

Build KubeNow image on GCE

Prerequisites

In this section we assume that:

- You have enabled the Google Compute Engine API: API Manager > Library > Compute Engine API > Enable
- You have created and downloaded a service account file for your GCE project: Api manager > Credentials > Create credentials > Service account key

Build the KubeNow image

Start by creating a `packer-conf.json` file. There is a template that you can use for your convenience: `mv packer-conf.json.gce-template packer-conf.json`. In this configuration file you will need to set:

- **image_name**: the name of the image that will be created after the build (the name must match `(?:[a-z](?:[-a-z0-9]{0,61}[a-z0-9])?)`, e.g. "kubenow-image")
- **source_image_name**: a Ubuntu Xenial image (this should already be in GCE, e.g. `ubuntu-1604-xenial-v20161013`)

- **account_file**: path to your service account file
- **zone**: the zone to use in order to build the image (e.g. `europe-west1-b`)
- **project_id**: your project id

Once you are done with your settings you are ready to build KubeNow using Packer:

```
packer build -var-file=packer-conf.json packer/build-gce.json
```

If everything goes well, you will see the new image in the GCE web interface (Compute Engine > Images). As an alternative, you can check that the image is present using the Google Cloud command line client:

```
gcloud compute images list
```

Build KubeNow image on Amazon Web Services (EC2)

Prerequisites

In this section we assume that:

- You have an IAM user along with its *access key* and *security credentials* (http://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_create.html)

Build the KubeNow image

The first time you are going to deploy KubeNow, you'll have to create its cloud image. This considerably speeds up the following bootstraps, as all of the required software will already be installed on the instances.

Start by creating a `packer-conf.json` file. There is a template that you can use for your convenience: `mv packer-conf.json.aws-template packer-conf.json`. In this configuration file you will need to set:

- **image_name**: the name of the image that will be created after the build (e.g. "kubenow-image")
  - **Warning:** the image_name must be unique in AWS, otherwise it will fail creating the new image
- **source_image_id**: an Ubuntu Xenial AMI ID
  - **Tip:** to figure out an Ubuntu Xenial AMI ID that works with your preferred region, you can use the Amazon EC2 AMI Locator
  - **Warning:** we support only *hvm:ebs-ssd* AMIs (other AMIs might work anyway)
- **aws_access_key_id**: your access key id
- **aws_secret_access_key**: your secret access key
- **region**: the region to use in order to create the image
  - **Warning:** the image that you previously selected has to be available in this region

Once you are done with your settings you are ready to build KubeNow using Packer:

```
packer build -var-file=packer-conf.json packer/build-aws.json
```

If everything goes well, something like the following will be printed out:

```
==> Builds finished. The artifacts of successful builds are:
--> amazon-ebs: AMIs were created:

eu-central-1: ami-XXXX
```

Tip: write down region and AMI ID for this KubeNow image build, as it will be useful in the next step.

In addition, you will see the new image in the Amazon web interface (EC2 Dashboard > Images > AMIs). You might need to change your location in the dashboard for your image to be shown.

As an alternative, you can check that the image is present using the amazon cloud command line client:

```
aws ec2 describe-images --owners self
```