KubeNow Documentation

Release master

mcapuccini

Oct 23, 2018

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

1.1 Install Docker

KubeNow provisioners are distributed via Docker and they are based on Kubernetes *1.9.2*. Please start by installing Docker on your workstation: Install Docker.

1.2 Get KubeNow

In order to get the provisioning tools please run:

```
docker pull kubenow/provisioners:master
```

We wrote up a handy CLI that wraps around the Docker container above, you can install it with a one-liner:

```
curl -Lo kn https://raw.githubusercontent.com/kubenow/KubeNow/master/bin/kn &&
↪ chmod +x kn && sudo mv kn /usr/local/bin/
```

CHAPTER 2

Deploy KubeNow on a host cloud

The following steps are slightly different for each host cloud. Here you find a section for each of the supported providers.

Sections

2.1 Deploy on OpenStack

2.1.1 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 at least one public IP

2.1.2 Deployment configuration

First we need to initialize a deploy configuration directory by running:

```
kn init openstack my_deployment
```

The configuration directory contains a new SSH key pair for your deployments, and a Terraform configuration template that we need to fill in.

Locate into `my_deployment` :

cd my_deployment

In the configuration file `config.tfvars` you will need to set at least:

Cluster configuration

- **cluster_prefix**: every resource in your tenancy will be named with this prefix
- **floating_ip_pool**: a floating IP pool label
- **external_network_uuid**: the uuid of the external network in the OpenStack tenancy

If you are wondering where you can get `floating_ip_pool` and `external_network_uuid`, then one way is to inquiry your OpenStack settings by running:

```
kn openstack network list
```

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

```
+--------------------------------------+----------------+...
| ID                                   | Label          |...
+--------------------------------------+----------------+...
| 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 it follows:

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

Master configuration

- **master_flavor**: an instance flavor for the master
- **master_extra_disk_size [optional]**: adds an extra disk with specified size to the node

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_extra_disk_size [optional]**: adds an extra disk with specified size to the node
- **node_assign_floating_ip [optional]**: adds a floating ip to this node

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 openstack command-line interface will come in handy. Just run the following command:

```
kn openstack flavor list
```

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

```
+--------+------------+...
| ID     | Name       |...
+--------+------------+...
| 8c7ef1 | ssc.tiny   |...
| 8d7ef2 | ssc.small  |...
| 8e7ef3 | ssc.medium |...
| 8f7ef4 | ssc.large  |...
| 8g7ef5 | ssc.xlarge |...
+--------+------------+...
```

You may want to select the favor according to much resources you'd like to allocate. E.g.:

```
master_flavor = "ssc.medium"
node_flavor = "ssc.large"
```

2.1.3 Deploy KubeNow

Once you are done with your settings you are ready deploy your cluster running:

```
kn apply
```

The first time you are going to deploy it will take longer, since the KubeNow image needs to be imported. Future deployments will be considerably faster, since the image will be already present in your user space.

To check that your cluster is up and running you can run:

```
kn kubectl get nodes
```

As long as you are in the `my_deployment` directory you can use `kubectl` over SSH to control Kubernetes. If you want to open an interactive SSH terminal onto the master then you can use the `kn ssh` command:

```
kn ssh
```

If everything went well, now you are ready to *deploy your first application*.

2.2 Deploy on Google Cloud (GCE)

2.2.1 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

2.2.2 Deployment configuration

First we need to initialize a deploy configuration directory by running:

```
kn init gce my_deployment
```

The configuration directory contains a new SSH key pair for your deployments, and a Terraform configuration template that we need to fill in.

Locate into `my_deployment` :

```
cd my_deployment
```

In the configuration file `config.tfvars` you will need to set at least:

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")

Google credentials

- **gce_project**: your project id
- **gce_zone**: some GCE zone (e.g. `europe-west1-b`)

Master configuration

- **master_flavor**: an instance flavor for the master (e.g. `n1-standard-2`)
- **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-2`)
- **node_disk_size**: nodes disk size in GB

In addition, when deploying on GCE you need to copy your service account file in the deployment configuration directory:

```
# assuming that you are in my_deployment
cp /path/to/service-account.json ./
```

2.2.3 Deploy KubeNow

Once you are done with your settings you are ready deploy your cluster running:

```
kn apply
```

The first time you are going to deploy it will take longer, since the KubeNow image needs to be imported. Future deployments will be considerably faster, since the image will be already present in your user space.

To check that your cluster is up and running you can run:

```
kn kubectl get nodes
```

As long as you are in the `my_deployment` directory you can use `kubectl` over SSH to control Kubernetes. If you want to open an interactive SSH terminal onto the master then you can use the `kn ssh` command:

```
kn ssh
```

If everything went well, now you are ready to *deploy your first application*.

2.3 Deploy on Amazon Web Services (AWS)

2.3.1 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)

2.3.2 Deployment configuration

First we need to initialize a deploy configuration directory by running:

```
kn init aws my_deployment
```

The configuration directory contains a new SSH key pair for your deployments, and a Terraform configuration template that we need to fill in.

Locate into `my_deployment` :

cd my_deployment

In the configuration file `config.tfvars` you will need to set at least:

Cluster configuration

- **cluster_prefix**: every resource in your tenancy will be named with this prefix
- **aws_region**: the region where your cluster will be bootstrapped (e.g. `eu-west-1`)
- **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

2.3.3 Deploy KubeNow

Once you are done with your settings you are ready deploy your cluster running:

```
kn apply
```

To check that your cluster is up and running you can run:

```
kn kubectl get nodes
```

As long as you are in the `my_deployment` directory you can use `kubectl` over SSH to control Kubernetes. If you want to open an interactive SSH terminal onto the master then you can use the `kn ssh` command:

```
kn ssh
```

If everything went well, now you are ready to *deploy your first application*.

2.4 Deploy on Microsoft Azure

2.4.1 Prerequisites

In this section we assume that:

- You have created an application API key (Service Principal) in your Microsoft Azure subscription: (https://www.terraform.io/docs/providers/azurerm/authenticating_via_service_principal.html#creating-a-service-principal)

2.4.2 Deployment configuration

First we need to initialize a deploy configuration directory by running:

```
kn init azure my_deployment
```

The configuration directory contains a new SSH key pair for your deployments, and a Terraform configuration template that we need to fill in.

Locate into `my_deployment` :

cd my_deployment

In the configuration file `config.tfvars` you will need to set at least:

Cluster configuration

- **cluster_prefix**: every resource in your tenancy will be named with this prefix
- **location**: some Azure location (e.g. `West Europe`)

Azure credentials

- **subscription_id**: your subscription id
- **client_id**: your client id (also called appId)
- **client_secret**: your client secret (also called password)
- **tenant_id**: your tenant id

Master configuration

- **master_vm_size**: the vm size for the master (e.g. `Standard_DS2_v2`)

Node configuration

- **node_count**: number of Kubernetes nodes to be created
- **node_vm_size**: the vm size for the Kubernetes nodes (e.g. `Standard_DS2_v2`)

2.4.3 Deploy KubeNow

Once you are done with your settings you are ready deploy your cluster running:

```
kn apply
```

The first time you are going to deploy it will take longer, since the KubeNow image needs to be imported. Future deployments will be considerably faster, since the image will be already present in your user space.

To check that your cluster is up and running you can run:

```
kn kubectl get nodes
```

As long as you are in the `my_deployment` directory you can use `kubectl` over SSH to control Kubernetes. If you want to open an interactive SSH terminal onto the master then you can use the `kn ssh` command:

```
kn ssh
```

If everything went well, now you are ready to *deploy your first application*.

CHAPTER 3

Deploy your first application

In this guide we are going to deploy a simple application: cheese. We will deploy 3 web pages with a 2 replication factor. The master node will act as a reverse proxy, load balancing the requests among the replicas in the Kubernetes nodes.

The simple cluster that we just deployed uses nip.io as base domain for incoming HTTP traffic. First, we need to figure out our cluster domain by running:

```
grep domain inventory
```

The command will return something like `domain=37.153.138.137.nip.io`, meaning that our cluster domain name in this case would be `37.153.138.137.nip.io`.

In KubeNow we encourage to deploy and define SaaS-layer applications using Helm. The KubeNow community maintain a Helm repository that contains applications that are developed and tested for KubeNow: https://github.com/kubenow/helm-charts. To deploy the cheese application you can run the following command, substituting `<your-domain>` with the domain name that you got above:

```
kn helm install --name cheese --set domain=<your-domain> kubenow/cheese
```

If everything goes well you should be able to access the web pages at:

- http://stilton.<your-domain>
- http://cheddar.<your-domain>
- http://wensleydale.<your-domain>

3.1 Traefik reverse proxy

KubeNow uses the Traefik reverse proxy as ingress controller for your cluster. Traefik is installed on one or more nodes, namely edge nodes, which have a public IP associated. In this way, we can access services with a few floating IP without needing LBaaS, which may not be available on certain cloud providers.

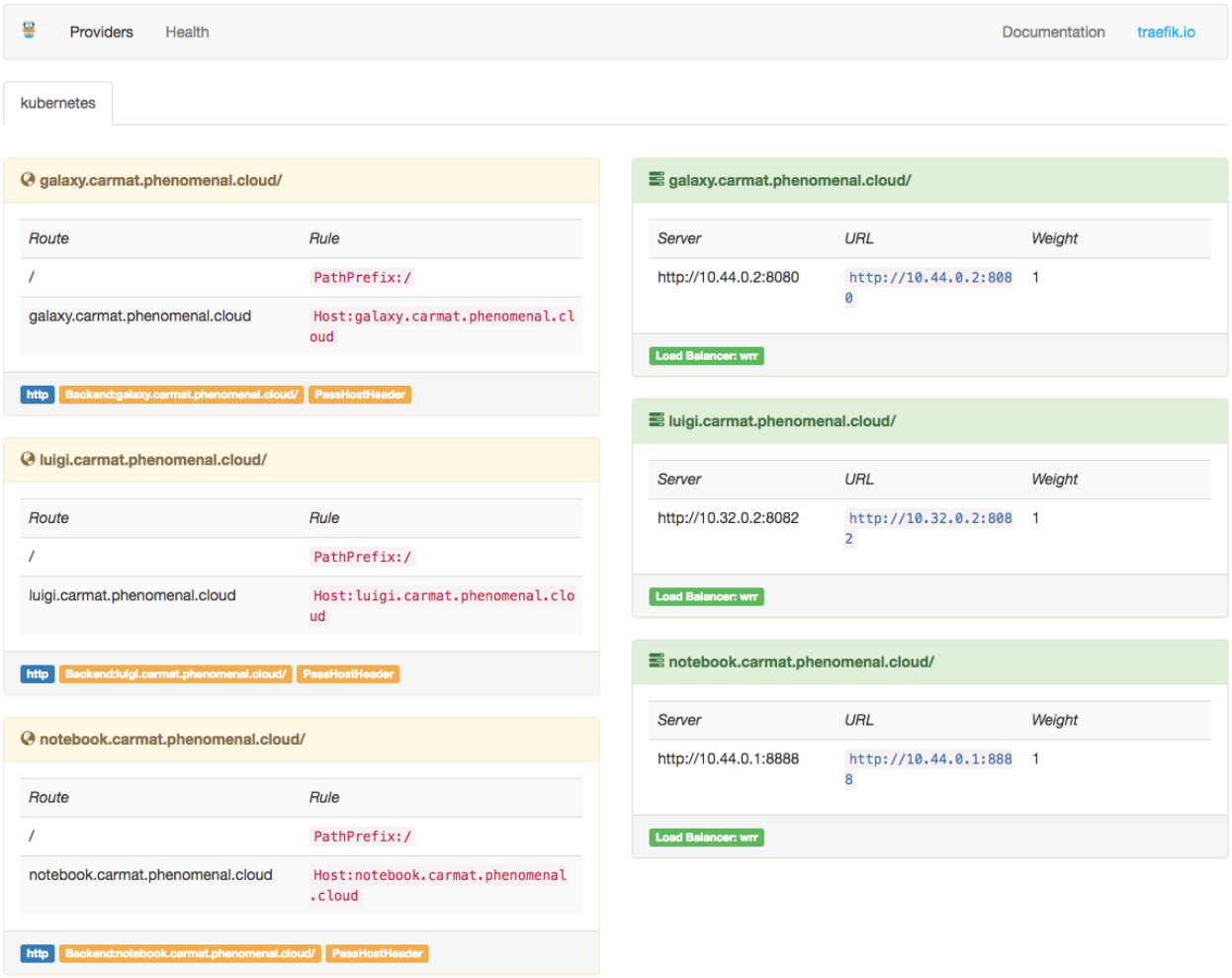
In the default setting KubeNow doesn't deploy any edge node, and it runs Traefik in the master node.

3.1.1 Accessing the Traefik UI

One simple and quick way to access the Traefik UI is to tunnel via SSH to one of the edge nodes with the following command:

```
ssh -N -f -L localhost:8081:localhost:8081 ubuntu@<your-domain>
```

Using SSH tunnelling, the Traefik UI should be reachable at http://localhost:8081, and it should look something like this:



In the left side you can find your deployed frontends URLs, whereas on the right side the backend services.

CHAPTER 4

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:

```
kn destroy
```

Warning: if you delete the cluster configuration directory (`my_deployment`) the cluster status will be lost, and you'll have to delete the resources manually.

CHAPTER 5

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

5.1 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 6

OpenStack troubleshooting

Contents

6.1 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>
```

6.2 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 7

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

7.1 List kubernetes pods

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

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

7.2 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:
kn ansible master -a "kubectl describe pod <pod id> --all-namespaces"
```

7.3 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:
kn ansible master -a "journalctl -r -u kubelet"
```

CHAPTER 8

More troubleshooting

Contents

8.1 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.

8.2 Figure out hostnames and IP numbers

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

```
cat inventory
```

CHAPTER 9

Edge Nodes

Edge nodes are specialized service nodes with an associated public IP address, and they run Traefik acting as reverse proxies, and load balancers, for the services that are exposed to the Internet. In the default settings, we don't deploy any edge node enabling the reverse proxy logic in the master node instead. However, in production settings we recommend to deploy one or more edge nodes to reduce the load in the master.

To deploy edge nodes, it is sufficient to uncomment the following lines in the `config.tfvars` file, and to set the desired number of edge nodes, along with an available instance flavor:

```
# Master configuration (mandatory in general, above all for single-server setup)
master_flavor = "your-master-flavor"
master_as_edge = "false"

# Edge configuration
edge_count = "2"
edge_flavor = "your-edge-flavor"
```

Please notice that we set `master_as_edge = "false"` to disable Traefik in the master node.

CHAPTER 10

GlusterFS Nodes

GlusterFS nodes are specialized service nodes. They run only GlusterFS and they are attached to a block storage volume to provide additional capacity. In the default settings, we don't deploy GlusterFS nodes, as it is not required in many use cases. However, GlusterFS can be particularly convenient when a distributed file system is needed for container synchronization.

To deploy GlusterFS nodes, it is sufficient to uncomment the following lines in the `config.tfvars` file, and to set the desired number of edge nodes, along with an available instance flavor and block storage disk size:

```
# Gluster configuration
glusternode_count = "1"
glusternode_flavor = "your-glusternode-flavor"
glusternode_extra_disk_size = "200" # Size in GB
```

10.1 How to claim a GlusterFS volume

KubeNow is configured to employ the Kubernetes dynamic volume provisioning, enabling GlusterFS storage volumes to be created on-demand. In addition, GlusterFS is configured as default StorageClass, meaning that when a user creates a PersistentVolumeClaim with unspecified `storageClassName` (i.e. left empty), the *DefaultStorageClass* admission controller automatically adds the GlusterFS *storageClassName*.

In practice, users can request GlusterFS dynamically provisioned storage by simply leaving the *storageClassName* field empty within their *PersistentVolumeClaim* template. An example follows:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: name-you-choose
 spec:
   accessModes:
     - ReadWriteOnce
   storageClassName: # left empty
   resources:
```

(continues on next page)

(continued from previous page)

```
    requests:
      storage: 1Gi
```

CHAPTER 11

Single-Node Deployments

When resources are scarce, or for testing purpose, KubeNow enables single-node deployments. In fact, it is possible to deploy the master node only, which will automatically enabled for service scheduling.

You can achieve this by commenting all of the lines for the other instance types (i.e. edge node, gluster node and worker node) in the `config.tfvars` file, leaving the master node only as it is shown below:

```
# Master configuration (mandatory in general, above all for single-server setup)
master_flavor = "your-master-flavor"
master_as_edge = "true"

# Node configuration
# node_count = "3"
# node_flavor = "your-node-flavor"

# Edge configuration
# edge_count = "2"
# edge_flavor = "your-edge-flavor"

# Gluster configuration
# glusternode_count = "1"
# glusternode_flavor = "your-glusternode-flavor"
# glusternode_extra_disk_size = "200" # Size in GB
```

CHAPTER 12

Cloudflare DNS Records

Cloudflare runs one of the largest authoritative DNS networks in the world. In order to resolve domain names for exposed services, KubeNow can optionally configure the Cloudflare dynamic DNS service, so that a base domain name will resolve to the edge nodes (or the master node if no edge node is deployed).

To configure the Cloudflare dynamic DNS service, it is sufficient to uncomment the following lines in the `config.tfvars` file, specifying credentials, domain and subdomain:

```
# Cloudflare configuration (optional)
use_cloudflare = "true"
cloudflare_email = "your-cloudflare-email"
cloudflare_token = "your-cloudflare-token"
cloudflare_domain = "your-domain-name"
cloudflare_subdomain = "your-subdomain-name"
```

CHAPTER 13

Cloudflare: Proxied Traffic

Incoming container traffic can be optionally proxied through the Cloudflare servers. When operating in this mode Cloudflare provides HTTPS for container services, and it protects against distributed denial of service, customer data breach and malicious bot abuse.

To enable Cloudflare proxied traffic, it is sufficient to uncomment the following lines in the `config.tfvars` file, specifying DNS records to be proxied:

```
# Cloudflare proxy (optional)
cloudflare_proxied = "true"
cloudflare_record_texts = ["record1","record2",...] # Warn: wildcards are not␣
↪supported when using proxied records
```

CHAPTER 14

Alternative Boot Image

KubeNow now allows to specify an alternative boot image of your choice for your cluster VMs. However KubeNow doesn't support yet an automatic downloading and/or uploading for a custom image, therefore you need to make sure such custom (or non-custom image) is already present in the cloud provider image repository you are using for your deployment.

This can be achieved easily by changing just a couple of lines in the `config.tfvars` file created by the initial command `kn init` as explained here. It is also necessary and important to set the parameter `skip_image_import` equal to `"true"` in order to avoid triggering the default importing mechanism that KubeNow supports for our customed images.

In the following snippet of code you can see how these changes should look like in the `config.tfvars` for each currently supported cloud-provider (a generic Ubuntu Xenial 16.04 image is used in the examples below):

Amazon:

```
boot_image = "ubuntu/images/hvm-ssd/ubuntu-xenial-16.04-amd64-server-201803*"
skip_image_import = "true"
```

Google cloud:

```
boot_image = "ubuntu-1604-lts"
skip_image_import = "true"
```

OpenStack:

```
boot_image = "Ubuntu 16.04 LTS (Xenial Xerus) - latest"
skip_image_import = "true"
```

Microsoft Azure:

```
boot_image_public {
  publisher = "Canonical"
  offer     = "UbuntuServer"
  sku       = "16.04-LTS"
  version   = "latest"
```

(continues on next page)

(continued from previous page)

```
}
skip_image_import = "true"

# In this case you also need to comment out or remove the whole boot_image parameter
↪'s line
# boot_image = "KubeNow-xx"
```

CHAPTER 15

Manual Cluster Scaling

KubeNow is now featuring the ability to add or delete nodes to your cluster with the help of the command `kn scale`.

While your cluster is deployed and running, it is necessary first to update the number of nodes in the `config.tfvars` file created by the very first command `kn init` as explained here . Currently `kn-scale` is only supporting adding and removing *worker nodes* or *edge nodes* (not *master nodes*). Last but not least, it is important to have at least one `node` left with enough space for your running services.

Once the `config.tfvars` has been updated with the new desired number of nodes, then we simply run the command:

```
kn scale
```

What will happen depends on whether we are downscaling or upscaling.

In the former scenario the command will first drain resources out of one or more node/s by executing `kubectl drain`, then remove it/them via `kubectl delete node` and finally a call to `terraform apply` will shut it/them down.

In the latter case one or more new node/s is/are first created by executing `terraform apply` and then joined to the already deployed cluster via `kubeadm join` by using our default bootstrap script (more details of such script in the main KubeNow's GitHub repository).

CHAPTER 16

Provisioning

It is now possible to both automatically and manually provision the cluster's instances via the master node. This is done by simply adding a provision block in your `config.tfvars` (which by now you should be familiar with). If available, such block will be automatically executed when the command `kn apply` is run. Otherwise we can also manually execute it with the new command `kn provision`.

A provision block can have one or several `action{ }` sub-block/s. The latter will always have at least a `type` field, which can either be `ansible-playbook` or `local-exec`. Based on these two scenarios, it is then necessary to introduce a couple of specific variables as shown in the examples here below.

Note: The Ansible version being used is the one installed in the KubeNow Docker image.

16.1 Action type = "ansible-playbook"

In this scenario, the following *ansible-playbook* specific variables will be used, with the exception of `extra_vars` which is optional:

```
playbook    = "path-to-playbook (relative config-directory)"
extra_vars  = "json-obj of extra variables (complying to hcl-json-syntax, see https://
↪github.com/hashicorp/hcl)"
```

Example **without** `extra_vars`:

```
#
# This block is equivalent to the command:
#
# ansible-playbook $ANSIBLE_OPT playbooks/install-core.yml
#
provision = {
  "action" = {
    "type"       = "ansible-playbook"
    "playbook"   = "playbooks/install-core.yml"
  }
}
```

Example **with** `extra_vars`:

```
#
# This block is equivalent to the command:
#
# ansible-playbook $ANSIBLE_OPT -e "$extra_vars" playbooks/create-pvc.yml
#
provision = {
  "action" = {
    "type"       = "ansible-playbook"
    "playbook"   = "playbooks/create-pvc.yml"
    "extra_vars" = {
      "claim_name"  = "galaxy-pvc"
      "storage"     = "95G"
    }
  }
}
```

16.2 Action type = “local-exec”

In this other case, we only have one *local-exec* specific variable that must be present in order to work:

```
command  = "command to execute - can be path to a script (relative config-directory)"
```

Example:

```
#
# This block is equivalent to the command:
#
# plugins/phnmnl/KubeNow-plugin/bin/phenomenal-post-install.sh
#
provision = {
  "action" = {
    "type"       = "local-exec"
    "command"    = "plugins/phnmnl/KubeNow-plugin/bin/phenomenal-post-install.sh"
  }
}
```

CHAPTER 17

Ingress Port Opening

On some deployments you might need to manually configure ports in order to allow specific service traffic. This can be done by adding/modifying the field `ports_ingress_tcp` in the configuration file `config.tfvars` (which you should be familiar with by now). Default is equals to `["22", "80", "443"]`. Let's suppose your newly deployed service also requires ports `7443` and `9443` to be opened, then one would modify the `config.tfvars` as follows:

```
# Cluster configuration
....

ports_ingress_tcp = ["22", "80", "443", "7443", "9443"]

....
```

Hence we've modified the field `ports_ingress_tcp` so to read **ports_ingress_tcp = ["22", "80", "443", 7443", "9443"]**.

17.1 To Keep in Mind

It is important to consider potential security risks to avoid future issues. While opening ports does put you more at risk than having none open, you are only in danger if an attack can exploit the service that is using that port. A port is not an all access pass to a cluster/network if an attacker happens upon it. Security is a complex topic indeed and can vary from case to case. Nevertheless here are some best practices for a proper configuration:

- **Block by default**: block all traffic by default and explicitly allow only specific traffic to known services. This strategy provides good control over the traffic and reduces the possibility of a breach because of service misconfiguration.
- **Allow specific traffic**:in general the rules that you use to define network access should be as specific as possible. This strategy is referred to as *the principle of least privilege*, and it forces control over network traffic. In this case what you are specifying is a certain port (or list of them) for your services to be reachable from outside the cluster's network.

CHAPTER 18

Image building

KubeNow uses prebuilt images to speed up the deployment. Image continous integration is defined in this repository: https://github.com/kubenow/image.

The images are exported on AWS, GCE and Azure:

- `https://storage.googleapis.com/kubenow-images/kubenow-<version-without-dots>.tar.gz`
- `https://s3.amazonaws.com/kubenow-us-east-1/kubenow-<version-without-dots>.qcow2`
- `https://kubenow.blob.core.windows.net/system?restype=container&comp=list`

Please refer to this page to figure out the image version: https://github.com/kubenow/image/releases. It is important to point out that the image versioning is now disjoint from the main KubeNow repository versioning. The main reason lies in the fact that pre-built images require less revisions and updates compared to the main KubeNow package.