TÉCNICO LISBOA

AGISIT 19/20	LAB ASSIGNMENT	Number:	1
Multi-Node Scenarios		Issue Date:	31 Dec 2019
Container Orchestration: Kubernetes		Due Date:	31 Dec 2019
Authors: Duarte Galvão, Pedro Cerejo, Rui Ribeiro		Revision:	0.0

1 Introduction

In this lab experiment the students will be building a multi-node Kubernetes cluster for testing purposes using Vagrant. This cluster will be composed by a Master Node (orchestrator) and several worker nodes. Ansible will be used to automate software configuration management for the nodes.

1.1 Kubernetes

Kubernetes is an open-source system for automating deployment, scaling and management of containerized applications. It provides a framework to run distributed systems resiliently taking care of scaling and failover, providing deployment patterns, load balancing, automated rollouts and rollbacks, and more. In the end of the lab, one will be able to setup a production-like environment for development purpose. A multi node cluster setup can help solve problems related to application design and architecture, enabling teams to execute tests on multiple versions of the application, or to reproduce issues easily.

A Kubernetes cluster is a set of machines, called nodes, that run containerized applications managed by Kubernetes. A cluster has at least one worker node and at least one master node. Worker nodes host the pods (set of running containers running) The master node(s) manages the worker nodes and the pods in the cluster.

For this experiment we'll only use one master node, but multi master nodes could be used to define better failover strategies and provide high availability to the cluster itself.

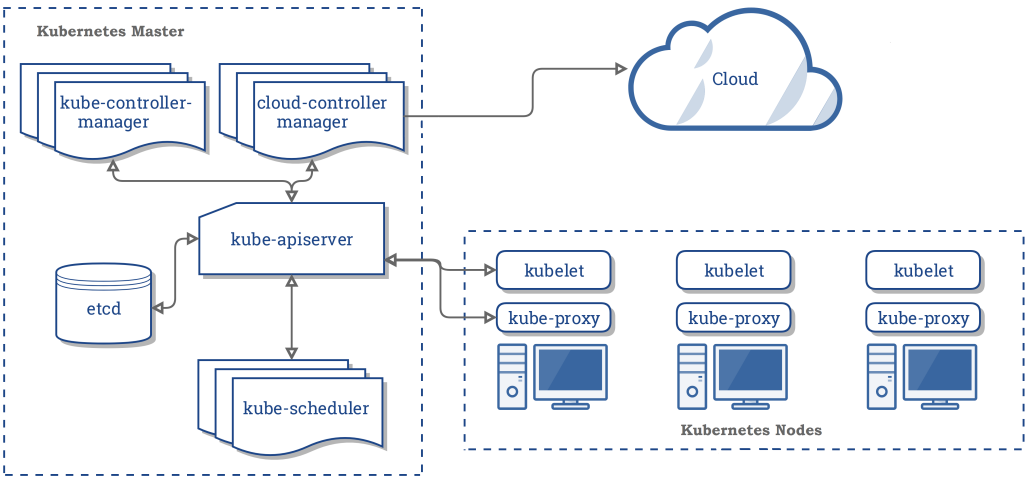

Figure 1: Kubernetes Components

1.1.1 Master Components

kube-apiserver

Exposes the Kubernetes API. kube-apiserver validates and configures data for api objects (pods, services, replication controllers, etc).

etcd

Consistent and highly available key value store used as Kubernetes' backing store for all cluster data. Despite not being a kubernetes specific component, a distributed key value store, in this case etcd, is mandatory for kubernetes to work.

kube-controller-manager

This component is responsible to run controllers. Controllers are processes that identify the shared state of the cluster through the api-server and make changes to that state attempting to move the current state to a desired state.

- Node Controller: Responsible for noticing and responding when nodes go down.
- Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
- Endpoints Controller: Populates the Endpoints object (that is, joins Services and Pods).
- Service Account and Token Controllers: Create default accounts and API access tokens for new namespaces.

cloud-controller-manager

Runs controllers that interact with the underlying cloud providers.

1.1.2 Node components

kubelet

An agent that checks containers running in a pod. The kubelet takes a set of PodSpecs (explained better later in the experiment) and ensures the containers described in those PodSpecs are running and healthy.

kube-proxy

A network proxy that runs on each node of the cluster and, via a defined set of network rules, allows communication to the Pods from inside or outside of the cluster.

Container Runtime

Software responsible for running containers. Kubernetes supports any container runtime that implements Kubernetes Container Runtime Interface (CRI) like Docker, containerd... More info on Kubernetes CRI can be found here.

1.2 Ansible

Ansible is used to automate many IT needs such as Cloud provisioning, Configuration management, Application deployment, Intra-service Orchestration, etc. Ansible models an IT infrastructure by describing how all systems inter-relate. Ansible is "agentless", meaning that it does not uses agents in the infrastructure nodes, making it easy to deploy – and most importantly, it uses a very simple language called YAML (Yet Another Markup Language), in the form of Ansible Playbooks. Ansible Playbooks are special text files describing the automation jobs in plain English. Ansible works by connecting to the Infrastructure Nodes and pushing out small programs, called "Ansible Modules" to them. These programs are written to be resource models of the desired state of the system. Ansible executes those modules (over SSH by default), and automatically removes them when finished.

Ansible will be used to automate the setup of **Kubernetes** nodes.

1.3 Vagrant

Vagrant is a tool that will allow us to create a virtual environment easily and it eliminates pitfalls that cause the works-on-my-machine phenomenon. It can be used with multiple providers such as Oracle VirtualBox, VMware, Docker, and so on. It allows us to create a disposable environment by making use of configuration files.

1.4 Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. It will be used as the containers that will host nginx in the worker nodes.

1.5 Nginx

NGINX is a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server. NGINX is known for its high performance, stability, rich feature set, simple configuration, and low resource consumption. In this lab experiment nginx will be deployed in the worker nodes by the kubernetes master node.

1.6 Preliminary Notes

The instructions in this document are applicable to Computers at the IST Labs. Nevertheless, one nice feature of the software stack we are going to use is that it is portable to many platforms including **YOUR OWN** personal computers, running the following Operating Systems:

- Microsoft Windows from version 10 up
- Apple macOS from versions 10.13 'High Sierra' up
- Debian-based Linux, such as Ubuntu (recommended) from versions 12.04 'Precise' up.

It is not recommended to apply this setup to a virtual machine (nested virtualization), although possible, as the configuration requires access to a hypervisor environment (recommended Virtualbox) in the host system. Before proceeding you should verify if you have a "clean" environment, i.e., no Virtual Machine "instances" running (using precious resources in your system), or inconsistent instances in Vagrant and Virtualbox. For that purpose run the vagrant global-status command and observe the results. It is **advisable to halt VMs** if running, and then **clean and destroy VMs from previous Lab experiments not related with this Lab**, as we may use in this Lab the same machine names.

Note: Avoid copying text strings from the command line examples or configurations in this document, as pasting them into your system or files may introduce/modify some characters, leading to errors or inadequate results.

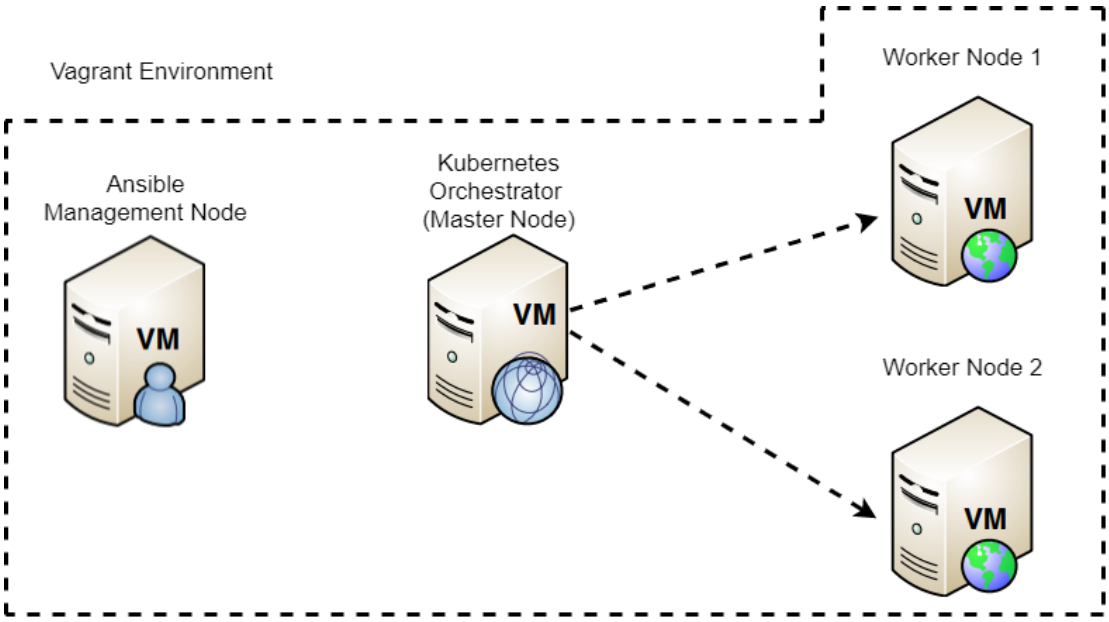

Figure 2: Vagrant Environment

2 Setting up the Vagrant-Ansible Environment

As good practice it's usual to create a folder for each vagrant environment. To start this project create a directory and name it, for example, **kuberneteslab**. Download the `AGISIT_19_20_LAB_GUIDE_1000_support_files.zip` and uncompress the content to the newly created folder.

```
|-- VagrantFile
|-- bootstrap-mgmt.sh
|-- bootstrap-targets.sh
|-- kuberntes
    |-- ansible.cfg
    |-- inventory.ini
    |-- master-playbook-yml
    |-- site.yml
    |-- ssh-addkeys.yml
    |-- workers-join-cluster.yml
```

The folder contains a `VagrantFile`, a file named `bootrstrap_mgmt.sh` and a folder named `kubernetes`. As defined in the `VagrantFile` we will create a **Management Node** named **mgmt** (responsible for configuring and installing the required software in the other nodes via **Ansible**), an **Orchestrator Node** named **orchestrator** (will act as the **Kubernetes Master Node**) and two **Worker Nodes** (that will be running **Nginx**).

```
IMAGE_NAME = "ubuntu/xenial64"

# Increase Kubernet numworkers if you want more than 2 worker nodes
NUM_WORKERS = 2
# VirtualBox settings
# Increase vmmemory if you want more than 256/512mb memory in the vm's
NODE_VM_MEMORY = 1024
MGMT_VM_MEMORY = 512
# Increase numcpu if you want more cpu's per vm
# The number of cores per vm should greater or equal than 2
NUM_CPU = 2
# Create the VMs
Vagrant.configure("2") do |config|

    config.ssh.insert_key = false
```

```
    config.vbguest.auto_update = true
    config.vm.box_check_update = false

    # Create management (mgmt) node
    config.vm.define "mgmt" do |mgmt|
      mgmt.vm.box = IMAGE_NAME
      mgmt.vm.network "private_network", ip: "192.168.56.10"
      mgmt.vm.hostname = "mgmt"
      if Vagrant::Util::Platform.windows? then
        # Configuration SPECIFIC for Windows 10 hosts
        mgmt.vm.synced_folder "kubernetes", "/home/vagrant/kubernetes",
          id: "vagrant-root", ouner: "vagrant", group: "vagrant",
          mount_options: ["dmode=775,fmode=664"]
        else
        # Configuration for Unix/Linux hosts
        mgmt.vm.synced_folder "kubernetes", "/home/vagrant/kubernetes"
      end

      mgmt.vm.provider "virtualbox" do |vb|
        vb.name = "mgmt"
        vb.cpus = NUM_CPU
        opts = ["modifyvm", :id, "--natdnshostresolver1", "on"]
        vb.customize opts
        vb.memory = MGMT_VM_MEMORY
      end

      mgmt.vm.provision "shell", path: "bootstrap-mgmt.sh"
    end

    # Create orchestrator (master) node
    config.vm.define "orchestrator" do |master|
        master.vm.box = IMAGE_NAME
        master.vm.network "private_network", ip: "192.168.56.11"
        master.vm.network "forwarded_port", guest: 80, host: 8080
        master.vm.hostname = "orchestrator"
```

```
        master.vm.provider "virtualbox" do |vb|
          vb.name = "orchestrator"
          vb.cpus = NUM_CPU
          opts = ["modifyvm", :id, "--natdnshostresolver1", "on"]
          vb.customize opts
          vb.memory = NODE_VM_MEMORY
        end

        master.vm.provision "shell", path: "bootstrap-targets.sh"
    end # orchestrator

    # Create worker nodes
    (1..NUM_WORKERS).each do |i|
        config.vm.define "worker-#{i}" do |node|
            node.vm.box = IMAGE_NAME
            node.vm.network "private_network", ip: "192.168.56.#{20 + i}"
            node.vm.network "forwarded_port", guest: 80, host: 8080+i
            node.vm.hostname = "worker-#{i}"

            node.vm.provider "virtualbox" do |vb|
              vb.name = "worker-#{i}"
              vb.cpus = NUM_CPU
              opts = ["modifyvm", :id, "--natdnshostresolver1", "on"]
              vb.customize opts
              vb.memory = NODE_VM_MEMORY
            end

            node.vm.provision "shell", path: "bootstrap-targets.sh"
        end # i
    end # workers
end
```

You can tweak the memory values for each node but that is a confortably number for this experiment. The number of cores per machine should be at least 2, otherwise kubernetes related playbooks might not run properly.

2.1 Lauching the Vagrant environment

Before lauching the Vagrant environment run `vagrant status` to check if all the nodes mentioned before. It should look something like this:

```
Current machine states:

mgmt                      not created (virtualbox)
orchestrator              not created (virtualbox)
worker-1                  not created (virtualbox)
worker-2                  not created (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

The machines are all in the state "not created" so run `vagrant up` to start them. This will launch the machines into the Vagrant environment, install ansible in the **Management Node** and configure a "hosts" inventory (via the post-install script `bootrstrap-mgmt.sh`), and copy some usefull ansible playbooks to that node.
When `vagrant up` finishes, run `vagrant ssh mgmt` to login to the Management Node.

2.2 Establishing a SSH Trust

In order to work with Ansible and a lot of configuration tools it is a MUST to establish a "**password less**" but secure access to the other machines in the network. One of the playbooks added to the **mgmt** node, `ssh-addkeys.yml` will help us with it. First generate a new RSA key using by running the following command:

```
vagrant@mgmt:~$ ssh-keygen -t rsa -b 2048
```

NOTE: when asked for the password press don't insert any. Just press **ENTER** in response to the prompts. This will generate a RSA keys with 2048 bytes. After that we need to add the inventory defined machines as ssh **known_hosts**. This can be achieved by running `ssh-keyscan` for all the VM's of the infrastructure and pipe the output to `.ssh/known_hosts`:

```
vagrant@mgmt:~$ ssh-keyscan orchestrator worker-1 worker-2 >> .ssh/
    known_hosts
# worker-1:22 SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.8
# orchestrator:22 SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.8
```

```
# orchestrator:22 SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.8
# orchestrator:22 SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.8
# worker-1:22 SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.8
# worker-1:22 SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.8
# worker-2:22 SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.8
# worker-2:22 SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.8
# worker-2:22 SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.8
```

You can now verify that the **known_hosts** file contains a bunch of keys from the remote machines and also verify the contents of **.ssh** to check if the RSA keys have been created. To deploy the key previously generated in the remote machines run the command `ansible-playbook` with the `--ask-pass` option (because it is the first time and we do not have "**password less**" login configured yet in those remote machines). The password to use is **vagrant**.

```
vagrant@mgmt:~/kubernetes$ ansible-playbook ssh-addkeys.yml --ask-pass
SSH password:

PLAY [targets] *****************************************************

TASK [Remove require tty - alt] ************************************
ok: [worker-1]
ok: [orchestrator]
ok: [worker-2]

TASK [Validate the sudoers file before saving  not to require password]
*********************************************************************
changed: [orchestrator]
changed: [worker-1]
changed: [worker-2]

TASK [install ssh key] *********************************************
changed: [worker-1]
changed: [worker-2]
changed: [orchestrator]

PLAY RECAP *********************************************************
```

```
orchestrator : ok=3  changed=2  unreachable=0  failed=0  ...  ignored=0
worker-1     : ok=3  changed=2  unreachable=0  failed=0  ...  ignored=0
worker-2     : ok=3  changed=2  unreachable=0  failed=0  ...  ignored=0
```

To check if everything works as expected and if ansible can remotely execute commands on the other machines run the ping command targeting all hosts:

```
vagrant@mgmt:~/kubernetes$ ansible all -m ping
```

You should be able to see PONG responses from every host.

```
vagrant@mgmt:~/kubernetes$ ansible all -m ping
orchestrator | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
worker-2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
worker-1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

3 Now, on your own: Setting up the Kubernetes cluster and it's nodes

In order to setup the cluster we'll use **kubeadm**. **kubeadm** is a "simple" tool that provides best-practice ways to create a **Kubernetes** cluster. It performs the actions necessary to get a minimum viable, secure cluster up and running in a user friendly way. **kubeadm** setup commands will run via the ansbile playbooks.

3.1 Installing the required software

Start by analysing `site.yml`, `master-playbook.yml` and `worker-playbook.yml` playbooks. The main playbook, named `site.yml` has one play that is responsible for installing the required kubernetes related software in both master and worker nodes. That software represents components briefly explained in the introduction. `site.yml` The `master-playbook.yml` is an example of a playbook that could be responsible to initialize the cluster itself using `kubeadm` and setup the network. However run the commands instead to understand how it works.

```
vagrant@mgmt:~/kubernetes$ ansible-playbook site.yml

PLAY [all] ***************************************************************

TASK [Gathering Facts] ***************************************************
ok: [worker-1]
ok: [worker-2]
ok: [orchestrator]

TASK [Install packages that allow apt to be used over HTTPS] ***********
ok: [worker-1]
ok: [worker-2]
ok: [orchestrator]

TASK [Add an apt signing key for Docker] ***********************************
ok: [worker-2]
ok: [worker-1]
ok: [orchestrator]

...
```

```
PLAY RECAP *********************************************************************
orchestrator : ok=15 changed=11  unreachable=0  failed=0  ...  ignored=0
worker-1     : ok=15 changed=11  unreachable=0  failed=0  ...  ignored=0
worker-2     : ok=15 changed=11  unreachable=0  failed=0  ...  ignored=0
```

To verify that everything was installed properly run the following commands (check the version of the install main software):

```
kubeadm version
kubelet --version
kubectl version
```

3.2 Initializing the cluster

To intialize the cluster we will run **kubeadm** in the orchestrator. Login in another terminal to orchestrator using `vagrant ssh orchestrator` and then run `kubeadm init` with the required options.

```
vagrant@orchestrator:~$ sudo kubeadm init
    --apiserver-advertise-address="192.168.56.11"
    --apiserver-cert-extra-sans="192.168.56.11"
    --node-name orchestrator --pod-network-cidr="192.168.0.0/16"
    --ignore-preflight-errors=true
W1231 14:21:33.953919    5073 validation.go:28]
    Cannot validate kubelet config - no validator is available
W1231 14:21:33.954834    5073 validation.go:28]
    Cannot validate kube-proxy config - no validator is available
[init] Using Kubernetes version: v1.17.0
[preflight] Running pre-flight checks
[WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker
        cgroup driver. The recommended driver is "systemd".
        Please follow the guide at https://kubernetes.io/docs/setup/cri/
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed
    of your internet connection
[preflight] You can also perform this action in beforehand using
    'kubeadm config images pull'
```

```
...

[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a
regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options
    listed at:
    https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following
    on each as root:

kubeadm join 192.168.56.11:6443 --token ********************** \
    --discovery-token-ca-cert-hash
    ****************************************************************
```

Now we'll need to copy our config file and setup the container networking provider and the network policy engine

To do so run:

```
mkdir -p /home/vagrant/.kube
sudo cp -i /etc/kubernetes/admin.conf /home/vagrant/.kube/config
sudo chown vagrant:vagrant /home/vagrant/.kube/config



****
```

```
kubectl create
    -f https://docs.projectcalico.org/v3.11/manifests/calico.yaml
```

Run `kubectl get nodes` to check the current nodes in the cluster. It should look similar to this.

```
vagrant@orchestrator:~$ kubectl get nodes
NAME           STATUS     ROLES    AGE     VERSION
orchestrator   NotReady   master   3m49s   v1.17.0
```

3.3 Join worker nodes to the cluster

As you've seen, when a Kubernetes cluster is setup a token is generated. That token is required in order for nodes to join the cluster. Analyse and run the `workers-join-cluster.yml` playbook. It will get the join command from the cluster node and copy it into the shared folder. The worker nodes will run said command and will join the cluster.

`workers-join-cluster.yml` must be executed from the **mgmt** node. As usual run `vagrant ssh mgmt` to login into **mgmt**, go into the folder `kubernetes` and run `ansible-playbookworkers-join-cluster.yml`

```
vagrant@mgmt:~/kubernetes$ ansible-playbook workers-join-cluster.yml

PLAY [master] ******************************************************************
************************************************
TASK [Gathering Facts] *********************************************************
************************************************
ok: [orchestrator]

TASK [Generate join command] ***************************************************
************************************************
changed: [orchestrator]

TASK [Copy join command to local file] *****************************************
************************************************
changed: [orchestrator -> localhost]

PLAY [worker] ******************************************************************
```

```
************************************************
TASK [Gathering Facts] ***********************************************
************************************************
ok: [worker-2]
ok: [worker-1]

TASK [Copy the join command to server location] **********************
************************************************
changed: [worker-1]
changed: [worker-2]

TASK [Join the node to cluster] **************************************
************************************************
changed: [worker-1]
changed: [worker-2]

PLAY RECAP ***********************************************************
************************************************
orchestrator           : ok=3   changed=2   unreachable=0   ...   ignored=0
worker-1               : ok=3   changed=2   unreachable=0   ...   ignored=0
worker-2               : ok=3   changed=2   unreachable=0   ...   ignored=0
```

After it you can run `kubectl get nodes` again from the **orchestrator** and the output should be something like this:

```
vagrant@orchestrator:~$ kubectl get nodes
NAME           STATUS     ROLES    AGE     VERSION
orchestrator   Ready      master   3m39s   v1.17.0
worker-1       NotReady   <none>     40s   v1.17.0
worker-2       NotReady   <none>     39s   v1.17.0
```

3.4 Create a Kubernetes deployment

A **Deployment** instructs Kubernetes how to create and update instances of an application. In this case the worker nodes will run nginx (you can related o previous lab experiments to setup nginx to serve a webpage).

While logged in to the **orchestrator** run the following commands in order to create the deployment for nginx

```
vagrant@orchestrator:~$ kubectl create deployment nginx --image=nginx
deployment.apps/nginx created

vagrant@orchestrator:~$ kubectl get deployments
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   0/1     1            0           36s

vagrant@orchestrator:~$ kubectl create service nodeport nginx --tcp=80:80
service/nginx created

vagrant@orchestrator:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        21m
nginx        NodePort    10.96.170.228   <none>        80:30702/TCP   45s
```

The value associated with the ports for the service **nginx** will be the port where nginx is running in the worker nodes. (In this case the port was 30702)

To confirm that every node is running nginx perform a request from the orchestrator to one of the worker nodes.

```
vagrant@orchestrator:~$ curl worker-1:30702
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
```

```
<p>If you see this page, the nginx web server is successfully installed
and working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

After the creation of the service the command `kubectl get nodes` should return:

```
vagrant@orchestrator:~$ kubectl get nodes
NAME           STATUS   ROLES    AGE    VERSION
orchestrator   Ready    master   ...m   v1.17.0
worker-1       Ready    <none>   ...m   v1.17.0
worker-2       Ready    <none>   ...m   v1.17.0
```

4 Finishing the experiments

In order to terminate the services running in the cluster you should run `kubectl delete deployment nginx` in the orchestrator node.

After that stop the Virtual Machines and verify the global state of all active Vagrant environments on the system, issuing the following commands:

```
vagrant halt
```

Run `vagrant status` to see the current state of the machines and confirm if the status of the machines is "powered off". You can destroy every machine with `vagrant destroy` ####### where ###### represents the machine id.

NOTE: https://www.youtube.com/watch?v=W5rTeJcHrxI is the Youtube video link for the complementary tutorial.