WASP Cloud Computing Module:
Ericsson Research Data Center Practical Session

PRACTICAL PART TWO: MAKE A WORKING KUBERNETES CLUSTER (seriously)

Goal: Kubernetes is complicated, but everyone is using it – why be left out? This practical won't do anything special, but we can at least see some of the basic steps involved in setting up k8s across multiple nodes… you can use this is the basis for any future work you do on ERDC.

This practical is heavily taken from *https://blog.alexellis.io/kubernetes-in-10-minutes/*

If you haven't already, log back into your VM at the ER DC

Install Docker

We could use a few different container runtimes, but Docker is simple. Install it by typing

```
sudo apt-get update                          (updates your repository)
sudo apt-get install -qy docker.io           (actually installs Docker)
```

You'll see quite a of text, and it'll take a minute or two, but when it's over, Docker will be installed.

Install Kubernetes

First we need to tell our Linux OS where the right Kubernetes package repository is. We do this with:

```
sudo apt-get install -y apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
```

(do not forget the " -" at the end of the second line)

We now need to update the packages list:

```
echo "deb http://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list
```

The character in the middle is a pipe "|" – make sure you don't type I or L instead.

Now let's update the actual repository (again):

```
sudo apt-get update                    (updates your repository)
```

This will take a minute or so, just like last time.

Now we can install *kubelet*, *kubeadm* and *kubernetes-cni* – all the components we need. *Kubelet* is responsible for scheduling, managing and running containers on your cluster. The *kubeadm* tool is a utility used to configure the components that make up a working cluster. The *kubernetes-cni* package represents the networking components which are not built into Kubernetes directly.

```
sudo apt-get install -yq kubelet kubeadm kubernetes-cni
```

This will also take a minute or so… but then… that's it, k8s is installed on your VM!

Creating the VM wasn't easy, was it? But we can now cheat and make things much faster to help create our Kubernetes cluster.

- Go back to the OpenStack interface: https://xerces.ericsson.net/
- Click "Compute >> Instances"
- Click "Create Snapshot" on the right-hand side of your Instance
- Call it whatever you like, maybe "Kubernetes VM Image"
- Click "Create Snapshot" in the bottom right

This saves a copy (snapshot) of your VM… with Docker and Kubernetes now installed on it. We can use this snapshot to quickly "Clone" the VM and make a few more:

- Go to "Compute >> Instances"
- Click "Launch Instance"
- Call the instance whatever you want, maybe "CloudTest2" or similar
- Click "Next" in the bottom right

- *You are now in the "Source tab"*
- In "Select Boot Source", select "Instance snapshot"
- Under the "Available" section at the bottom, click the "up arrow" to the right of your snapshot
- Make sure "Delete Volume on Instance Delete" is set to "Yes"
- Click "Next" in the bottom right

- *You are now in the "Flavor" tab*
- Click the "up arrow" to the right of "c2m4" (or pick a different flavor if you prefer!)
- Click "Next" in the bottom right

- *You are now in the "Networks" tab*
- Click the "up arrow" next to the network you created earlier (NOT "internet")
- Click "Next" in the bottom right

- *You are now in the "Network Ports" tab*
- Like last time, there is nothing to do here.
- Click "Next" in the bottom right

- You are now in the "Security Groups" tab
- Under "Available" click the "up arrow" next to the security group you created earlier
- Click "Next" in the bottom right

- You are now in the "Key Pair" tab
- The key you created earlier should already be allocated.
- If it isn't, click the "up arrow" to the right of it, in the "Available" section
- Now click "Launch Instance"

Done! Now you have a second VM! That was easy – now make at least one more, so you have three (or more) VMs, all using the same snapshot you created the second with. When you know what to click, you can literally go through all the above steps in about 20 seconds – try it yourself.

Sometimes you have to click "Compute >> Instances" again to see all the Image Names.

Now, we need to give each new VM a Floating IP address so we can connect to it:

- Go to "Compute >> Instances"
- To the right of each new VM, click the drop down menu next to "Create snapshot"
- Click "Associate Floating IP"
- Click the "+" to the right of the "No floating IP addresses allocated" text box
- Click "Allocate IP" in the bottom right
- Click "Associate" in the bottom right

Don't forget to do this step for each of the new VMs you have created.

Now go to "Network >> Network Topology" and gaze in wonder at your network, and all the VMs that you have created.

Keep the original VM terminal window open, and connect to the new VMs using the same method as before (using SSH in Linux, or PuTTy in Windows). Go to "Compute >> Instances" and select the IP address starting with "129.192" to log in to each VM. We have given them all the same private key, so you can log into all three (or more) using the same credentials.

Now back to the original VM terminal window.

We are going to make the first VM we created the "primary node" (or master) for Kubernetes. We're going to let K8s "advertise" itself to everyone on the internal (i.e. private) network.

Let's find our internal IP address. Type the following:

```
ifconfig
```

You should find the VM's internal IP address listed next to "ens3", called "inet". It will look *something* like 192.168.0.X

Using this IP address, now type in the following (replace THE_INTERNAL_ADDRESS with the IP above)

```
sudo kubeadm init --apiserver-advertise-address=THE_INTERNAL_ADDRESS
```

This command is the "big one" – it's going to set up a k8s primary node on your VM. You will have to wait a few minutes while it starts…. don't worry about any timeout message you might get. Make sure you don't close the window or type anything else when installation is complete – the message you are given at the bottom of the text is important!

You should now have a k8s master node running!

We have a few little things to do before we can add the other VMs and create our cluster though. Thankfully, we can copy and paste from the output text of the command we just ran. First of all, we need to set up our environment. This is easy, copy and paste the text given under "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 won't get any output when you have finished those three commands, but they set up the correct local of the configuration we need as a non-root user.

Now let's look at the other text it gave us. You will see something like:

```
kubeadm join 192.168.1.248:6343 --token 7tgcg0.343443uxrbwdz7fuj \
        --discovery-token-ca-cert-hash sha256:01bcae1545445a153d0ed05e8cf415229f013331a4d02e4c77ed4
```

This is the only command you need to give to join your other VMs to your k8s cluster! Copy it, and go to the terminal window for your second VM. Paste it there, but make sure you add "sudo " to the beginning, as you are not root! So something like:

```
sudo kubeadm join 192.168.1.248:6343 --token 7tgcg0.343443uxrbwdz7fuj \
        --discovery-token-ca-cert-hash sha256:01bcae1545445a153d0ed05e8cf415229f013331a4d02e4c77ed4
```

It'll take a few seconds and then you'll get a "This node has joined the cluster" message. Repeat this step for your other VMs (but NOT the primary VM).

Congratulations! Your multi-node Kubernetes cluster is now operational, running on the ER DC!

We can test to make sure all our nodes have joined the cluster. Go to the primary VM, and type:

`kubectl get node` **(note this ONLY works on the primary node)**

You should see all your nodes listed. It'll look something like this:

```
ubuntu@cloudtest1:~$ kubectl get node
NAME         STATUS     ROLES                  AGE     VERSION
cloudtest1   NotReady   control-plane,master   11m     v1.21.0
cloudtest2   NotReady   <none>                 2m11s   v1.21.0
cloudtest3   NotReady   <none>                 84s     v1.21.0
ubuntu@cloudtest1:~$
```

You will note that the node status for every node in your cluster is marked "**NotReady**". That doesn't sound good, but it's because we haven't set up the Pod networking yet. This is going to be a little complicated, so best to just type the following three commands onto the PRIMARY NODE:

```
sudo mkdir -p /var/lib/weave

head -c 16 /dev/urandom | shasum -a 256 | cut -d" " -f1 | sudo tee /var/lib/weave/weave-passwd

kubectl create secret -n kube-system generic weave-passwd --from-file=/var/lib/weave/weave-passwd
```

If you type (or paste) the above correctly, you will see the message "**secret/weave-passwd created**".

Now type the following:

```
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl
version | base64 | tr -d '\n')&password-secret=weave-
passwd&env.IPALLOC_RANGE=192.168.0.0/24"
```

If you copy/paste the above correctly, you will get a message along the lines of "**weave-net-created**".

Now we can test if it works. Type:

```
kubectl get node
```

And you should see something like this:

```
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.apps/weave-net created
ubuntu@cloudtest1:~$ kubectl get node
NAME         STATUS   ROLES                  AGE     VERSION
cloudtest1   Ready    control-plane,master   18m     v1.21.0
cloudtest2   Ready    <none>                 9m41s   v1.21.0
cloudtest3   Ready    <none>                 8m54s   v1.21.0
ubuntu@cloudtest1:~$
```

Finally, we've done it! We have multiple VMs running at the Ericsson data center, and have used them to create a multi-node Kubernetes cluster! Now we are ready to run containers, and watch how k8s balances them across nodes, etc.

Want to see what containers are currently running on your cluster (including all the hidden ones)?

```
kubectl get all –A
```

Running a container

Let's make sure that we don't have any pods running in our cluster at the moment. DO ALL OF THE FOLLOWING ON YOUR PRIMARY (Control Plane) VM:

```
kubectl get pods
```

You should see "No resources found in default namespace" – that's fine, we've not deployed anything yet! We'll deploy an example contain, which shows "developers how to package a Node.js and Express.js microservice". First, we need to install "Helm" – this is a tool that is used to manage, install and upgrade Kubernetes applications:

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash
```

Helm is now installed! Now let's use Helm to download and install the example service/container:

```
helm repo add expressjs-k8s https://alexellis.github.io/expressjs-k8s/
helm repo update
helm install test-app expressjs-k8s/expressjs-k8s
```

If the service is deployed correctly, you'll get a message on your terminal showing you a command to set up a port forward. We'll do this:

```
kubectl port-forward -n default deploy/test-app-expressjs-k8s 8088:8080 &
```

(don't forget the " &" at the end of that command!)

We haven't set up networking to use our local Web Browser to view this service, but we can access it remotely on the terminal, using:

```
curl http://127.0.0.1:8088
```

Now let's find out where the container pod is located:

```
kubectl get pod -o wide
```

Under "Node" you'll see where k8s has placed the Pod – it'll be somewhere on your cluster!

Let's scale it up! Set the number of replicas to whatever you like – I've used 5:

```
kubectl scale deploy/test-app-expressjs-k8s --replicas=5
```

Now let's check out the locations again:

```
kubectl get pod -o wide
```

You should see that k8s has now scaled the container further across your cluster!

You can also play with high availability – what happens if you go into OpenStack and "pause" one of your VMs? You'll find that after a 5 minute wait (which you can change), Kubernetes realises that that VM is down, and will automatically replicate the Pods to other nodes in the network.

That's it – you've installed a network of VMs, you've installed a Kubernetes cluster across it, and you've deployed and scaled a Kubernetes container Pod!

This is just the beginning, but you can build and learn more from here – well done!

When you've had enough, go back to the OpenStack page in your web browser. You can "tear down" your cluster by deleting the instances, the snapshots, the security groups, key pairs, etc.