K8s lab part 3, lab 1 – Deploying containers using K8s

1. First we create a folder where we store our yaml files:

mkdir ~/yaml

cd ~/yaml

1. Figure out what we want to containerize. In this example, we will use a simple nginx container. Browse to <https://hub.docker.com> and search for “nginx”. We will use the standard “*nginx*” container.
2. To get this container running, we will build out our very first yaml file that will build us a pod. Please note the spacing!!! Indentation in yaml is very important (like Python is too for example):

nano nginx.yml

---

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod

spec:

containers:

- name: nginx-container

image: nginx

ports:

- containerPort: 80

1. We will apply this yaml file to k8s now. Save and close, then perform these actions:

kubectl get pods (this should only return the snapshot helper pod)

kubectl apply -f nginx.yml

kubectl get pods (this should return a pod called “nginx-pod”).

* Make sure the status of the pod is in the “ready” state. If this fails (stuck on “ContainerCreating”), check what is going wrong by using:

kubectl describe pod nginx-pod

1. Now we have a container running. But how do we ever get to this container from the network? Answer: We need a K8s service. Before we create the service, we will have to “tag” the pod or pods we want to use by adding a label to the pod. To do this, edit our file and add the label part (check indentation!!):

nano nginx.yml

---

apiVersion: v1

kind: Pod

metadata:

name: nginx-pod

**labels:**

**app: nginx**

spec:

containers:

- name: nginx-container

image: nginx

ports:

- containerPort: 80

1. With the label attached, just run the “kubectl apply” command again to update the container:

kubectl apply -f nginx.yml

1. Now we can add a service which will bridge between our pod and the rest of the world:

nano service.yml

---

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

type: NodePort

selector:

app: nginx

ports:

- port: 80

targetPort: 80

nodePort: 30007

1. Apply the service:

kubectl apply -f service.yml

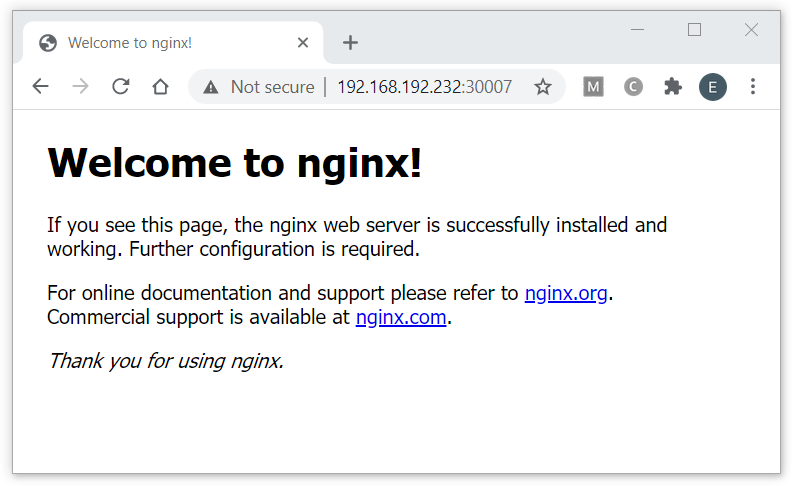
kubectl get svc -o wide

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE SELECTOR

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 6d22h <none>

nginx-service NodePort 10.106.63.171 <none> 80:30007/TCP 2m22s app=nginx

1. Now the service is up as a “nodePort”. You should be able to browse to any of the worker nodes on the IP address 30007 and see the container’s web response:



1. Now clean up the environment by removing the service and the pod:

kubectl delete svc nginx-service

kubectl delete pod nginx-pod

K8s lab part 3, lab 2– Load Balancing a K8s deployment

1. A deployment is basically a pod that is contained in a ReplicaSet which is in turn contained into a Deployment. In order to build a deployment we will have to build another yaml file that will describe the pod, the ReplicaSet and the actual deployment. Please note the spacing!!! Indentation in yaml is very important (like Python is too for example):

nano nginx-deploy.yml

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deploy

labels:

app: nginx

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

name: nginx-pod

labels:

app: nginx

spec:

containers:

- name: nginx-container

image: nginx

ports:

- containerPort: 80

1. We will apply this yaml file to k8s now. Save and close, then perform these actions:

kubectl apply -f nginx-deploy.yml

kubectl get deployments.apps (This shows you the deployment)

kubectl get pods -o wide (this should return three pods called “nginx-deploy-xxxxxx” running across the worker nodes).

1. Now we can add a service which will bridge between our pod and the rest of the world. In this case we will add the service to the same yml file:

nano nginx-deploy.yml

and **add** this section to the yaml file (just like the service we had before):

---

apiVersion: v1

kind: Service

metadata:

name: nginx-service

spec:

type: NodePort

selector:

app: nginx

ports:

- port: 80

targetPort: 80

nodePort: 30007

1. Apply the service by just applying the deployment file again. You should see the deployment was unchanged but the service was created:

kubectl apply -f nginx-deploy.yml

deployment.apps/nginx-deploy unchanged

service/nginx-service created

kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 6d23h

nginx-service NodePort 10.97.248.118 <none> 80:30007/TCP 69s

1. Now once again the service is up as a “nodePort”. You should be able to browse to any of the worker nodes on the IP address 30007 and see the container’s web response. But this is a bit boring eh? All containers return the exact same page… So maybe use an example where the containers return the name of the instance they are running on. In order to do that, edit the deployment file again, find the “ image: nginx” and replace the image with “nginxdemo/hello”:

nano nginx-deploy.yml

<…>

containers:

- name: nginx-container

**image: nginxdemos/hello**

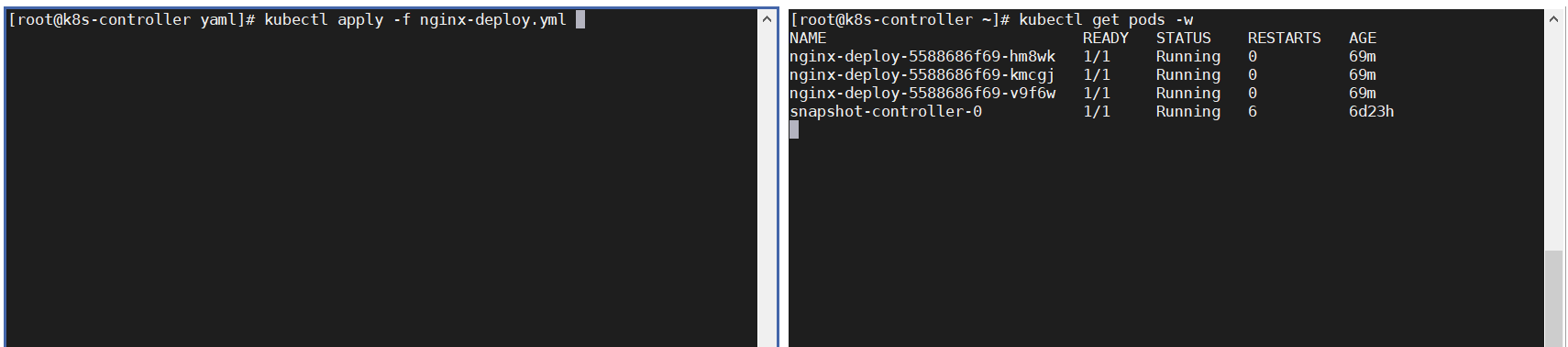
ports:

- containerPort: 80

<…>

1. Now let’s see what a deployment is all about. To show that the best, open up a second SSH session to the controller node, and display them next to each other (using “Split” from MobaXterm). In the right window run this command to continuously watch the pods:

kubectl get pods -w



1. Next switch to the left window and apply the deployment:

kubectl apply -f nginx-deploy.yml

The right window responds by creating NEW containers (with the new image) and then terminate the old ones as new ones become ready:

[root@k8s-controller ~]# kubectl get pods -w

NAME READY STATUS RESTARTS AGE

nginx-deploy-5588686f69-hm8wk 1/1 Running 0 69m

nginx-deploy-5588686f69-kmcgj 1/1 Running 0 69m

nginx-deploy-5588686f69-v9f6w 1/1 Running 0 69m

snapshot-controller-0 1/1 Running 6 6d23h

nginx-deploy-6b88f9db64-vdp58 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-vdp58 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-vdp58 0/1 ContainerCreating 0 0s

nginx-deploy-6b88f9db64-vdp58 1/1 Running 0 3s

nginx-deploy-5588686f69-kmcgj 1/1 Terminating 0 73m

nginx-deploy-6b88f9db64-8xw2h 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-8xw2h 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-8xw2h 0/1 ContainerCreating 0 0s

nginx-deploy-5588686f69-kmcgj 0/1 Terminating 0 73m

nginx-deploy-6b88f9db64-8xw2h 1/1 Running 0 4s

nginx-deploy-5588686f69-hm8wk 1/1 Terminating 0 73m

nginx-deploy-6b88f9db64-rrs7p 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-rrs7p 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-rrs7p 0/1 ContainerCreating 0 0s

nginx-deploy-5588686f69-hm8wk 0/1 Terminating 0 73m

nginx-deploy-6b88f9db64-rrs7p 1/1 Running 0 3s

nginx-deploy-5588686f69-v9f6w 1/1 Terminating 0 73m

nginx-deploy-5588686f69-hm8wk 0/1 Terminating 0 73m

nginx-deploy-5588686f69-v9f6w 0/1 Terminating 0 73m

nginx-deploy-5588686f69-kmcgj 0/1 Terminating 0 73m

…

1. Now check the output in the browser once again. You should see the other nginx image displaying a message similar to this:



Refreshing the page should actually cycle you through the different containers! I could not get this to work in Chrome. Using Firefox instead makes it simple: Just hold the <SHIFT> key while clicking the refresh icon on the browser.

1. Want more instances? Just go back to the MobaXterm. Clear the right screen and restart the kubectl watch. Then go to the left screen and enter this command:

kubectl scale deployment nginx-deploy --replicas=7

Just watch the number of replicas shoot up to 7:

NAME READY STATUS RESTARTS AGE

nginx-deploy-6b88f9db64-8xw2h 1/1 Running 0 19m

nginx-deploy-6b88f9db64-rrs7p 1/1 Running 0 19m

nginx-deploy-6b88f9db64-vdp58 1/1 Running 0 20m

snapshot-controller-0 1/1 Running 6 6d23h

nginx-deploy-6b88f9db64-gbjr8 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-znljt 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-mbzmk 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-gbjr8 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-znljt 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-rt472 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-mbzmk 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-znljt 0/1 ContainerCreating 0 0s

nginx-deploy-6b88f9db64-rt472 0/1 Pending 0 0s

nginx-deploy-6b88f9db64-gbjr8 0/1 ContainerCreating 0 0s

nginx-deploy-6b88f9db64-mbzmk 0/1 ContainerCreating 0 0s

nginx-deploy-6b88f9db64-rt472 0/1 ContainerCreating 0 0s

nginx-deploy-6b88f9db64-rt472 1/1 Running 0 4s

nginx-deploy-6b88f9db64-mbzmk 1/1 Running 0 4s

nginx-deploy-6b88f9db64-znljt 1/1 Running 0 5s

nginx-deploy-6b88f9db64-gbjr8 1/1 Running 0 5s

nginx-deploy-6b88f9db64-gw9pl 1/1 Running 0 5s

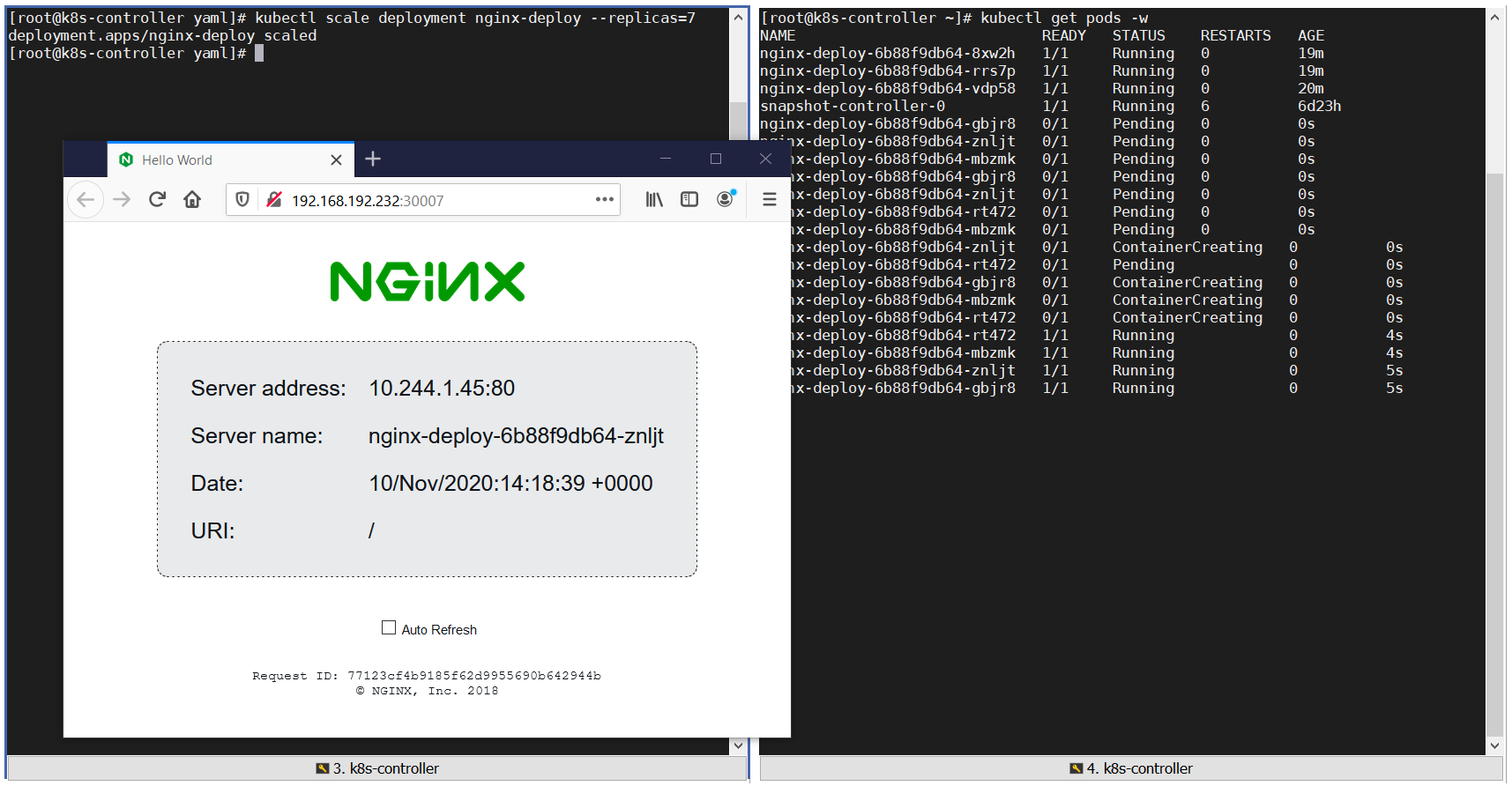
nginx-deploy-6b88f9db64-kph2s 1/1 Running 0 6s

nginx-deploy-6b88f9db64-kx9cd 1/1 Running 0 7s

nginx-deploy-6b88f9db64-s4gb7 1/1 Running 0 8s

nginx-deploy-6b88f9db64-ckhqk 1/1 Running 0 10s

And YES: As the service recognized all these pods (label!!) it will now happily load balance between 7 pods!



This concludes the lab on “deploying containers using K8s”.