ulm university universität
uulm

Tools & Concepts for Cloud Deployments

Exercise 7: Kubernetes; Reliability & Availability

Christopher B. Hauser

Institute of Information Resource Management, Ulm University

SummerSchool 2019, Curitiba

Exercise 7: Kubernetes; Reliability & Availability

Overview

Welcome to exercise 7. This time we will have the following lessons:

1. Container Orchestration with Kubernetes
2. Concept of Reliability and Availability

Lessons learned

Lesson 1: Container Orchestration with Kubernetes

In this lesson, we will install and use Kubernetes, the Container orchestration from Google.

Research: What is Kubernetes and how does it work?

Make yourself familiar with Kubernetes. Focus on the features and the concepts, described on the following links.

https://kubernetes.io/

https://kubernetes.io/docs/concepts/

Questions: Kubernetes

- What Featuers does Kubernetes provide?
- Where in our Cloud Stack do you place Kubernetes?

Cloud Stack	Example	Deployment Tool
Application Component	Mediawiki	?
Containers	Docker	?
Virtual Resource	Instance m1.small	?
Cloud Platform	OpenStack	-

Task: Setup Kubernetes

Cleanup your bwCloud tenant. Next, start *four virtual machines* from image Ubuntu 16.04, put them into your private network and add floating IPs. One of these four VMs will be your aster, the other three will be worker nodes.

We will now follow these guides to install Kubernetes:

- https://kubernetes.io/docs/setup/independent/install-kubeadm/
- https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/

Install Docker and Kubernetes on all your virtual machines:

```
# install docker
sudo apt-get install -y \
```

```
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
    | sudo apt-key add -
sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
sudo apt-get update
sudo apt-get install -y docker-ce

sudo usermod -aG docker ubuntu

# install kubelet and kubeadm
apt-get update && apt-get install -y apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-
key add -
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
EOF
apt-get update
apt-get install -y kubectl kubelet kubeadm kubernetes-cni
```

One node will become the Kubernetes Master. In this host, run `kubeadm init` as root to initialize a new kubernetes cluster. The initialization command will print important information for the cluster:

```
Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run (as a regular user):

  sudo cp /etc/kubernetes/admin.conf $HOME/
  sudo chown $(id -u):$(id -g) $HOME/admin.conf
  export KUBECONFIG=$HOME/admin.conf

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

```
You can now join any number of machines by running the following on each node
as root:

  kubeadm join --token 0a78c2.82b7a3558d6a38a6 192.168.5.6:6443
```

Run all the kubectl and kubeadm commands as root on all your hosts.

Run the first three commands, which sets the configuration file to access the kubernetes cluster. The last command `kubeadm join` will be required to add hosts to the new cluster. But before we continue to add hosts, we need to set up an overlay network for Kubernetes, to allow private IP networks for containers across hosts.

```
kubectl apply -f \
http://docs.projectcalico.org/v2.3/getting-started/kubernetes/installation
/hosted/kubeadm/1.6/calico.yaml
```

Kubernetes defines `pods`: a component which is executed by starting one or many containers from a specified image. With Kubernetes, you can list all running Pods:

```
kubectl get pods --all-namespaces
```

You should see some kube-* pods, which are core features of Kubernetes, and you should see some calico-* pods, which enable the overlay networking we just installed. If some of the Pods are still in STATUS "Pending", be patient until they are all in "Running" state.

In the mean while, log in to all other hosts an run the `kubectl join ...` command form the previous output as root. In the master, validate that the hosts are registered by running `kubectl get nodes`. You should get the following output:

```
root@kubernetes-1:~# kubectl get nodes
NAME           STATUS     AGE       VERSION
kubernetes-1   Ready      15m       v1.6.6
kubernetes-2   Ready      28s       v1.6.6
kubernetes-3   NotReady   6s        v1.6.6
kubernetes-4   NotReady   8s        v1.6.6
```

When all nodes are ready, we can start running containers on the kubernetes cluster. The most simple way to start a new container is by running:

```
kubectl run $nameOfYourPod --image=$imageToStartFrom --port=8080 ...

# e.g. start a web server with 3 replicas:
kubectl run testserver --replicas=3 --image=httpd:latest --port=80
```

```
# list the deployment status
kubectl get deployments

# list running pods
kubectl get pods

# show detailed information
kubectl describe pod $podNameFromList
```

The `kubectl describe` command shows the private IP address of the containers, and the ClusterIP of the deployment. Inside the virtual machine, the web server is accessible with this private IP / Cluster IP, but it will not be accessible form the outside. For accessible application deployments, Kubernetes provides services. To create a new service, we need to expose the pod:

```
# create service for web servers
kubectl expose deployment testserver --type=NodePort --target-port=80

# list services
kubectl get services
```

The output of the service list contains the port number, by which the exposed deployment is accessible on any of the hosts. In the example below the web server is accessible by any of the public node IPs and port 32495. An external load balancer then can distribute incoming requests among existing nodes in kubernetes.

```
NAME         CLUSTER-IP       EXTERNAL-IP   PORT(S)         AGE
kubernetes   10.96.0.1        <none>        443/TCP         2h
testserver   10.105.254.205   <nodes>       80:32495/TCP    6s
```

Task: Mediawiki on Kubernetes

In the previous exercises we started our own registry, built the Dockerfiles as images and pushed them to the registry. For running the Mediawiki application on kubernetes, these steps are required as well.

With the images available, you can run and expose deployments for MySQL and Apache2 with the Mediawiki application on Kubernetes. Try to deploy Mediawiki (Database and Webserver) on Kubernetes.

Use the `kubectl` and `docker` command to troubleshoot issues while deploying.

Lesson 2: Concept of Reliability and Availability

So far in the exercises we focused on scalability of stateless application layers (e.g. the web server). Handling state in distributed systems is generally a challenging task, especially in large scale systems with high demand on availability.

Research: Consistency Guarantees

Do some research on consistency guarantees: ACID and eventual consistency

Further, look at the trade offs between consistency and availability guarantees: CAP theorem and BASE (Basically Available, Soft state, Eventual consistency)

Questions: Consistency Guarantees

- What is ACID? What is eventual consistency?
- What are the three categories of CAP?
- What are examples for CA data store? What are examples for AP? And for CP?

Research: Distributed Database Systems

To store state information in Cloud applications, database systems are very popular. If the application has to scale, the database system eventually has to scale and hence distribute as well.

Do some research about distributed database systems, and the features the database systems have to support to be scalable.

Questions: Distributed Database Systems

- What is the difference between replication and sharding/partitioning?
- How does replication relate to the CAP theorem?
- How does sharding relate to the CAP theorem?
- How would the perfect Cloud database system look like?