Assignment 3.2 - Container Orchestrations

Katerina Chinnappan, Rohit Shaw, Kjell Zijlemaker

Objective

The objective of this assignment is to prepare 3 VMs and create a Kubernetes cluster which will then host the URL shortener web service from assignment 1.1 with at least 3 replicas of this service.

Implementation

The development enviroment of this assignment is in OpenNebula as in assignment 2. We reused the VMs which were deployed for the Hadoop cluster.

Installation

To connect the three VMs within the Kubernetes cluster, it was required to first install docker and Kube-Admin on all three VMs and disable the swap-memory because of incompatibility reasons.

Configuration

On the master node, Calcio was used for networking accross the cluster for supporting the joining of the slave nodes. In the hosts file, all hostnames were required to be changed to the IP addresses accordingly. After this, the DNS worked and the join command was required be given on the slaves.

On the slaves, the join command has to be given to connect to the master node: *kubeadm join 10.100.0.49:6443 –token 7ytct9.hhsfd8mqzcodfcch –discovery-token-ca-cert-hash sha256:c92e23a86ff50d3a7f430 ae09f505c82fe8cb4f01eef174210fc7c0d4fd8bf84*

To deploy a service on the cluster, it was required to create a docker image which could then be used for deploying the URL shortener. This image was pushed to the Docker Hub and could then be used to deploy the REST service with the deployment.yaml configuration file. Finally, the namespace.yaml and service.yaml files made sure that the service was exposed to the public. In our configuration, these IP addresses are: 10.100.0.52 and 10.100.0.64.

Result

After installation and the configuring of the nodes, any user can access the service on the IPs given in the previous section.

The overall experience was good. The installation and configuration was easier than the previous assignment in which Hadoop was required to be installed. The join command which has to be executed on the slaves is more intuitive than the approach which was required when joining nodes in the Hadoop cluster. No XML files had to be configured either, which took a lot of time in the configuration of the cluster.

However, when configuring the Kubernetes cluster, a networking module had to be installed to let the slaves join the cluster. At first, this was not explained in some of the tutorials which was confusing because the DNS discovery would not work. After some time we were able to solve this fairly easy by installing the Calcio module. The DNS discovery would then be automatically enabled.

Adding new containers, replications and exposing services from docker images was also a nice experience. After creating such an image, creating a deployment from the service and exposing it just took a few commands and is highly scalable because of this simplicity. Kubernetes automatically assigns the service on multiple nodes for load balancing and availability.

Conclusion

To conclude, the Kubernetes cluster was faster to create than the Hadoop cluster in the previous assignment. It is possible to create multiple containers and services from docker images and push them onto the cluster automatically with a few commands from the master node. The Kubernetes cluster will then use all the nodes in the cluster for load-balancing and replication. Because this can be done in just a few commands and by configuring just three YAML files, it can be said that Kubernetes can indeed be seen as a very interesting shift in publishing Cloud applications in companies which are required to make use of a cluster of nodes and need to make use of scalable solutions and high availability.