Report on Kubernetes experiment

Group 16: You Hu 12036234, Wenchen Lai 12482625

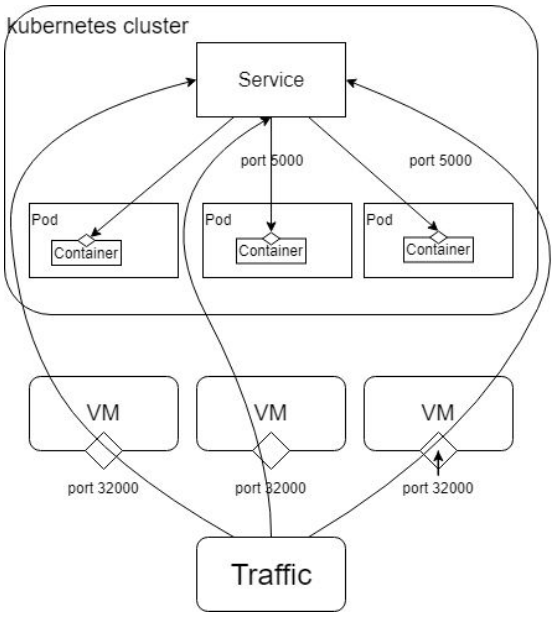
After the experiment on Docker, an obvious goal come out. Like what we expected to VMs, we hope to scale up the services fitting to the requirements and organize the processing units automatically.

In Kubernetes, multiple containers are grouped as Pod, which is the unit providing independent service. The idea of Pod is to achieve scalability, load balance, and fault tolerance. To get better know how Kubernetes work, we performed an experiment to deploy URL-shortener on 3 VMs cluster and expose it as service.

We installed 3 Ubuntu VMs on our laptop. First, we installed one VM and installed all the Kubernetes components and Docker on it with appropriate configurations. And then we exported the VM as OVF file to install another two VMs. Note that in the previous experiment for Hadoop cluster, we installed all three VMs manually as we tried to use “Clone” to duplicate VM but there would be a conflict in MAC address. However, the OVF approach avoided this kind of problem. Besides, all VMs are assigned IP address by DHCP which means, the IP addresses may change after restarting or resuming from pending status.

We chose one VM as Master and initialized the cluster with CIDR IP pool configuration 10.244.0.0/16. And then we installed addon Canal which utilizing Calico and flannel networking together as a unified networking solution. After that, we let all worker nodes joined the cluster.

Before we deployed URL-shortener, we had uploaded the image to the Docker hub. Therefore, all nodes can get the the image from Docker hub in the deployment. Then we wrote the YAML files for deployment and service explosion. In the deployment file, we explicitly configured that all container would open port 5000 which our service will listen to. And in the service file, we configured that the mode is NodePort in which all nodes open port 32000, and users can get served by accessing to any nodes’ port 32000. The structure is shown below.



In our experiment, we found that the different results for the same operation were returned. We dug into the reason, we think it is because the scheduler lets different pods served requests and at the same time, the URL mappings are stored as Python dictionary. Therefore the storage will be not consistent.
Comparing to OpenNebula, Kubernetes focuses more on service while OpenNebula is more about managing virtual resources. After some research, we found in the industry it is not the one-or-another situation. They can be combined together, even though we need to sacrifice some performance, we can get more reliability and elasticity at the same time.