

Containers

107021129 黃明瀧

Cloud Computing Homework 2

A. Screenshots for Task 1

```
[rx570][~/Sync/cloud-compute-hw-2/socket-server]$ docker images | head -n2
REPOSITORY          TAG             IMAGE ID        CREATED        SIZE
localhost/socket-107021129  latest         ad78de4bc46e   49 minutes ago 13.2 kB
[rx570][~/Sync/cloud-compute-hw-2/socket-server]$
```

Figure 1: List containing the container for task 1

```
[rx570][~/Sync/cloud-compute-hw-2/socket-server]$ docker run --rm -d -p 8080:8080 localhost/socket-107021129:latest
de4956ca49af344110434b62c1de86c3ac5df622baafa839c3630a3f91e90e51
[rx570][~/Sync/cloud-compute-hw-2/socket-server]$ ./client
Hello message sent
Hello from server
[rx570][~/Sync/cloud-compute-hw-2/socket-server]$
```

Figure 2: Running the container and the client

B. Screenshots for Task 3

The client was modified to read destination IP and port from stdin.

```
s107021129@cc-jump:~/task1$ bat socket-deployment.yml
File: socket-deployment.yml
Size: 446 B
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: s107021129-deployment
5  spec:
6    selector:
7      matchLabels:
8        app: s107021129-socket
9    replicas: 5
10   template:
11     metadata:
12       labels:
13         app: s107021129-socket
14         tier: backend
15         track: stable
16     spec:
17       containers:
18       - name: s107021129-socket
19         image: socket-107021129:latest
20         imagePullPolicy: Never
21         ports:
22         - containerPort: 8080
23
s107021129@cc-jump:~/task1$
```

Figure 3: Deployment configuration

```
s107021129@cc-jump:~/task1$ bat socket-service.yml
File: socket-service.yml
Size: 192 B
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: s107021129-service
5  spec:
6    type: NodePort
7    selector:
8      app: s107021129-socket
9    ports:
10     - protocol: TCP
11       targetPort: 8080
12       port: 30008
13
s107021129@cc-jump:~/task1$
```

Figure 4: Service configuration

```

s107021129@cc-jump:~/task1$ kubectl get deployment
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
s107021129-deployment  5/5     5             5           40m
s107021129@cc-jump:~/task1$ kubectl get service
NAME      TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes  ClusterIP   10.96.0.1     <none>        443/TCP          80m
s107021129@cc-jump:~/task1$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
s107021129-deployment-55646bcb84-6dp5x  1/1     Running   0           40m
s107021129-deployment-55646bcb84-8jtrg   1/1     Running   0           40m
s107021129-deployment-55646bcb84-gmg5s   1/1     Running   0           40m
s107021129-deployment-55646bcb84-pwzmr   1/1     Running   1 (9m25s ago) 40m
s107021129-deployment-55646bcb84-rkkwm   1/1     Running   0           40m
s107021129@cc-jump:~/task1$ ./client
172.18.0.38 32400
Hello message sent
Hello from server
s107021129@cc-jump:~/task1$

```

Figure 5: `kubectl` and client outputs

C. Performance of Container vs VM (Task 2)

All the experiments are ran for at least 3 times, and the average values is presented in the following table.

Item	VM	Container
Sysbench CPU (Events/s)	18486	18630
Sysbench Memory (Events/s)	61141029	65703919
Sysbench File IO (Events/s)	17737	1066502
iperf (Gbps)	1.19	9.68

In terms of CPU and memory performance, there is no dramatic difference between the VM and the container. With the help of KVM, the VM inside QEMU run pretty much as well as the container, with very minor performance hit.

File IO speed in the VM is severely impacted by the fact that the file operations goes through the qcow2 disk image format, which, as described in the documentation of QEMU, “has the largest overhead compared to raw images when it needs to grow the image”. Since the experiments were done in a fresh VM, the images had not allocated the space required for the files written by sysbench, and did the allocation on-the-fly while sysbench runs, causing file IO to be very slow.

`iperf` shows that the container has a huge advantage (with more than 8 times better throughput) in

communicating with the host machine via network, since the network traffic is essentially travelling between processes on the host itself.

D. Difference between Docker Container and VM

As opposed to virtual machines, which are themselves fully virtualized systems, Docker containers are essentially a bunch of carefully managed and restricted processes running on the same host. This enables containers to be much more lightweight and enables better sharing of the computational resources on the host (for example, memory allocation and overcommitting is much more easier), at the cost of having less isolation and security.

Docker containers typically have different use cases than virtual machines. Docker containers are nowadays mainly used for deploying stable development and production environments, which is not commonly done using virtual machines due to the bloat of virtual machine images.

E. Explain “Deployment”, “Service” and “Pod”

Deployment

Deployment in Kubernetes is a declarative way to specify how to create and manage the *pods* that constitutes a container app. Typically things like the container images to be used and the number of replicas are specified in a deployment.

Service

A *service* in Kubernetes groups one or more pods in a cluster as a logical application, presenting them to the outside world as a network service. In this assignment, a [NodePort](#) service is used to expose the socket service on all nodes.

Pod

A *pod* in Kubernetes is an instance (or several instances) of running Docker containers, and is the smallest unit of execution. When there are multiple containers inside a single pod, the containers share network and storage resources.

F. Kubernetes

Kubernetes is a system for managing containerized applications and services. It provides us with many features such as *service discovery* (exposing container with a DNS name), *rollouts* and *rollbacks* (gradually reaping old instances and creating new ones), *fault tolerance* (restarts unhealthy instances), *load-balancing*, and more.

The need of Kubernetes stems from the fact that managing containers *across multiple servers* at scale is hard. Kubernetes presents a tightly integrated interface and declarative configurations, which allows the operators to do so in a maintainable way.

G. Container Technology

H. Container Data