Performance Analysis of MQTT on Docker
a Comparison Between the Running Results on Virtual Machine and Docker

Lu Le, Yiling Zhang, Jizhou Yang, Kai Zheng and Xin Huang
Department of Computer Science and Software Engineering
Xi’an Jiaotong-Liverpool University
Suzhou, China
{Lu.Le12, Yiling.Zhang11, Jizhou.Yang12, Kai.Zheng14}@student.xjtlu.edu.cn, Xin.Huang@xjtlu.edu.cn

Abstract — the rapid development of Internet of Things (IoT) has brought about a variety of message transport protocols and virtualized platforms to realize them. The configuration and running performance of transport protocols can distinguish according to the virtualization technique they chose. Therefore, the comprehension of these IoT protocols and virtual platforms may assist users to instantly configure and establish their required environment and to adjust suitable message transport strategies for different situations. This paper realizes the Message Queuing Telemetry Transport (MQTT) protocol in Java on two commonly-deployed environments: Virtual Machine (VM) and Docker. Both of these are frequently used virtualization technologies. In this paper, the protocol MQTT and these two platforms VM and Docker are firstly introduced; Afterwards, the differences between these two platforms and their respective advantages are elaborated. Finally, the performance of MQTT is reached and analyzed.

Keywords: IoT, MQTT, Docker, Virtual Machine, Linux Container

1. INTRODUCTION

The IoT attempts to achieve a wireless control to physical items in real world by means of transport protocols and virtualization technology [1]. In IoT, the protocols are function as the message communication rules among different objects while the virtualization technology decides the platforms that these rules base on.

The virtualization technology is not only used to establish the platforms for the devices to communicate with, it also can be applied in other Internet-based fields such as Cloud Computing [2, 3], which enables the IoT to be organized more flexible. Virtualization technology, typically the VM, has been the main trend among programmers. Through importing a concise software layer into the physical layer and the user interface, this technique aims to reduce the coupling between the layers and relieve the problem of implanting complicate software on various hardware platforms [4]. In other words, virtualization divides a single hosting environment into several isolated VMs that run their own environment and occupy their respective hardware resources. Apart from the traditional VM, the Docker container, which is another virtualization technology released in 2003, attracts an growing attention [5].

However, when configuring the virtual environment, some problems may raise. For example, duplicated internal memory can be yielded; the boot time may extend; and the configuration procedure itself can be tedious. Moreover, the running performance of transport protocols may also be influenced by these issues.

Therefore, this paper focuses on the new popularized virtualization technologies: Docker. Research into their effects on the communication performance of MQTT. The basic knowledge of Docker will firstly be introduced, then will show the process of setting MQTT server on Docker and finally the performance of Java-implemented MQTT on these two platforms will be analyzed to support users for their specialized establishment of virtual platform.

2. BACKGROUND

2.1 MQTT

MQTT protocol is a lightweight instant message protocol for tiny sensors and mobile devices [6]. It is developed by IBM and designed for information transport in IoT. This protocol supports most of the mainstream platforms, and is able to connect not only each item inside the IoT, such as the sensors, but also the outside staff, the actors for instance. For example, houses are connected using MQTT protocol through Twitter[5].

On account of its small size and efficient distributing capability, MQTT is chosen as the protocol to test the property of virtual containers.

2.2 Docker

Docker is a platform that allows users to pack, distribute and manage Linux applications within containers, and has become unparalleled and has been employed by increasing practitioners from all over the world[2].

The kernel of Docker is developing on the basis of Linux Container (LXC) [7]. LXC is known as an extremely
lightweight virtualization technology: it enables a host to run multiple Operation Systems simultaneously while occupying relatively small hardware resources. Consequently, this thereafter improves the portability and standardization of Docker.

3. REALIZATION OF SETTING MQTT SERVERS ON DOCKER
In this part, the detailed procedure of running MQTT on Docker are listed. With the analysis result, users are able to comprehend the characteristics of each environment, especially for Docker.

3.1 Running of MQTT
The following table implies that the MQTT protocol started running

```
Docker run -ti -p 1894:1894 -p 9011:9011 toke/mosquitto
```

Table 1. Initialization of MQTT server

This command in Linux means that the MQTT server is established and the message transporting port is 0.0.0.0:1894 and 9011. And to test if this server is established successfully. We transmit the message “123” through it

```
mosquitto_pub 0.0.0.0 -t test -m 123
```

Table 2. publish message on MQTT server

This command means that the publish message is “123”, which theme of it is “test” in the port 0.0.0.0

```
mosquitto_sub -h 0.0.0.0 -t test
```

Table 3 subscribe message on MQTT server

This command means the subscriber monitoring the port of 0.0.0.0 and finally received the message which theme of message is “test”. The result shows below. The message is successful transmit which means that the MQTT server is actually established.

```
123
```

Table 4 the result showing in subscriber

3.2 Parallel Running of MQTT on Docker
In this part, Java codes of MQTT protocol and the code of Docker file were used to realize the parallel running of MQTT.

After the initialization, 10 Docker containers were initialized. The above picture is the part of snapshot of ten containers running, the ten containers occupy different ports. And ten MQTT servers are established on those ten ports.

4. SYSTEM CONSTRUCTION ENVIRONMENT

<table>
<thead>
<tr>
<th>System</th>
<th>Ubuntu 14.04</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>3.7 GiB</td>
</tr>
<tr>
<td>Processor</td>
<td>Intel Core 2 Duo CPU <a href="mailto:E6750@2.66GHz">E6750@2.66GHz</a>*2</td>
</tr>
<tr>
<td>OS type</td>
<td>64GB</td>
</tr>
<tr>
<td>Disk</td>
<td>40.1GB</td>
</tr>
</tbody>
</table>

Table 1 system configuration Environment

5. RESULT ANALYSIS

5.1 Size - Comparison

Ten Docker containers were generated on the host and MQTT server were established on each container. We used the “top” command in Linux and find out CPU occupation rate is nearly one core. Afterwards, MQTT servers were set on two VMs in the host. After using the “top” command we can also see the CPU occupation rate is 101.1%.

Fig. 1 Number of Containers and VMs

The above graph demonstrates those 10 parallel MQTT Dockers running on the Ubuntu System When the occupation rate of CPU is one-core. However, using one core of CPU there are only two parallel MQTT servers running on two virtual machine on the same host.

5.2 Average boot-time

To measure the average boot-time, we realized the setting up of MQTT on two environment and checked the boot time. Each host has same condition of H/W and S/W. The boot-time is monitored by NMON tool.

Fig. 1 Parallel Running of MQTT
In Fig. 2, average boot-time is 1.23s of Docker container and as for VM, average boot-time is 15.38s. The figure can show that the average boot-time of docker is less than VM. VM needs the time of booting Guest OS. However, he boot-time of docker is shorter compare to VM for the reason of using Host OS.

5.3 Other Detail Information

When using the “top” command in Linux system. The detail information of CPU can be showed when ten Docker container and 2 VMs running respectively. Some detail information is reorganized below.

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RE</th>
<th>SHRS</th>
<th>S</th>
<th>%CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>31176</td>
<td>924</td>
<td>848</td>
<td>R</td>
<td>99.9%</td>
</tr>
</tbody>
</table>

Table 1. Resources Occupied by 10 Dockers Containers

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>PES</th>
<th>SHR</th>
<th>%CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>3217</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>2535428</td>
<td>1.083g</td>
<td>1.058g</td>
<td>101.1%</td>
</tr>
</tbody>
</table>

Table 2. Resources Occupied by two VMs

Obviously, single Docker container occupy less resources than one Virtual Machine. Some information we have already discussed. There are also some other detail information showing above.

Those comparisons can show that Dockers are relatively superior to the virtual machines while running IoT protocols. When using the same configuration of computer, the scale and the number of VMS can run is less than those of Dockers. The occupation rate of CPU and Memory of single VM is also proved to be fairly higher than the amount consumed by one Docker images. Furthermore, the boot-time of docker container is shorter than virtual machine.

6. CONCLUSION

In this paper, the researchers have run MQTT server on different environments and it can be seen from the results that Docker is a more suitable environment. The server running on it operation fast and occupy less.

To summarize, although the Docker is an immature application, it possesses its natural advantage in term of the insulation and consumption of source and the boot time. Docker is a light weight system compared with the traditional virtualization platform, which means it requires fairly fewer system resources (occupation rate of CPU and RAM) to maintain the system. User is able to choose customize the virtualized OS as their own service platform according to the demand. In the near future, the number of new projects combining the virtualization with IoT or Cloud Computing tends to increase; thus, the research of Docker is essential for IoT users.

REFERENCES