Songlin Jiang

Implementing a Virtual Network System among Containers

INTRODUCTION

Capture audience's attention:
Good morning! I guess many of you may have taken the Network Security Course last semester or the Lab course in Networking and Security this semester. For the network testbed, you must have been wondering, like, why it takes me over 10 minutes to get it started! Why it tells me that my computer memory is not enough! Especially if you are a Mac M1 or M2 user, you must have been really really upset since you can't start the testbed at all! Now, I'm here to save us all. The cure is to use containers. Let's stop using virtual machines!

Establish credibility:
Hi everyone! My name is Songlin Jiang and my tutor is Tuomas Aura. Today I'm going to discuss how to Implement a Virtual Network System among Containers.

Purpose:
It's still a common practice to build and test network systems configurations using virtual machines, so that engineers can experiment in a virtual environment before setting up the physical system. In addition, virtual networks are sometimes also needed for automated integration tests, as developers may want to test the usability and performance of their software under some specific network topology.

Creating a virtual computing environment can be slow and troublesome, and containers can fix these issues effectively.

As we all know, everything is going cloud. Due to little research on implementing network systems using containers, I investigated the possibilities of implementing virtual network systems based on Docker containers to overcome the disadvantages of virtual machines. My research also analyzes the functional and security limitations when virtual networks are implemented this way.

Overview:
So, I have divided this talk into five parts. First, we will talk a bit about docker container networking systems. Second, we will look into the VPN case study and check its details for implementation. Third, we will talk about the evaluation. Finally, I will present my conclusions as well as Q&A.

Now, let's look at the Docker network systems to see what we can do with Docker.

BODY

1. **Docker Networking System**

   A. Network Drivers
      - available drivers
      - initial screening choice

Docker employs a pluggable networking subsystem. Several drivers can provide the Docker network functionality. The default drivers include the bridge, host, overlay, IPVLAN, MACVLAN, and none.

The none, host, and overlay drivers do not cater to our needs. Firstly, the none driver disables all networking. Secondly, the host driver is unnecessary as it can have security implications due to its nature of sharing the same network stack with the host machine. In addition, the overlay is not an appropriate option as we are simulating the network system only on one host machine.

As a result, the possible candidates are bridge, IPVLAN, and MACVLAN. We finally chose MACVLAN.

   B. final choice: MACVLAN

Existing research has conducted benchmarks for different virtualization technologies for the networking overhead. The result shows that the MACVLAN driver performs best in throughput while requiring the least CPU resources compared to the other network drivers. MACVLAN is a trivial bridge that does not need to learn as it already knows every MAC address it can receive. IPVLAN is similar to MACVLAN, except IPVLAN assigns the same MAC address to all containers attached to it. In contrast, MACVLAN assigns a different MAC address to each attached Docker container.

Moreover, MACVLAN networks are the best choice when migrating from a VM setup, as MACVLAN makes the container appear as a physical device with its own MAC address.

MACVLAN bridge mode allows the testbed network to run layer 2 protocols, such as the ARP and Link Layer Discovery Protocol. MACVLAN also supports address configuration and discovery protocols, as well as other multicast protocols, such as DHCP and Precision Time Protocol.

C. Routing, Firewall and IPv6

- net_admin capability
- IPv6

By default, Docker does not allow manipulating container network devices and setting routing tables or firewalls inside the containers. These features can be enabled by assigning the net_admin capability to the container. IPv6 is also supported in Docker containers.

For networking, it seems like everything can get done in a virtual machine can also be done inside a container. Now, let's take a look at the VPN case study for implementing these into reality.

2. Case Study: VPN

A. VPN Overview

- topology
- implementation

Our research simulates a scenario where the IoT devices in two sites, A and B, would like to connect to the server in cloud S. The topology is based on the Network Security course Project 2, where site A, B, and cloud S both use the private IP addresses to improve security and save the IPv4 addresses. Gateways A, B, and S connect sites A, B, and S to the public Internet. The router in the topology represents the Internet between the sites and the cloud. The address space between the gateway and the router simulates public, routable IPv4 addresses, although they are all private. Site A, B, and cloud S use the router to access the Internet.

In order to make the clients in both site A and site B connect to the cloud server safely, we implemented a virtual network testbed based on Docker containers. We experiment with two types of VPN: site-to-site and host-to-host, using strongSwan, a VPN implementation based on IPsec.

B. Host to Host

A host-to-host VPN connects different gateways together. IP packets then get routed to different clients within the corresponding site according to the routing table of the gateway.

C. Site to Site

A site-to-site VPN connects different network systems located at different sites together directly. In this case, the address spaces of sites A, B and the cloud network S should not overlap.

D. Site to Site in IPv6

Site to Site in IPv6 is similar to Site to Site VPN in IPv4 but replaces all the address space in IPv6.

Now that we have looked at the case study topology, you must be very interested in our evaluation result. So here it is.

3. Evaluation

A. Usability and Portability

Containers are much more efficient and lightweight than virtual machines because containers share the Linux kernel with the host. In contrast, virtual machines employ hardware virtualization and have their own kernel instance, which consumes more resources and can be very slow when starting up.

The Docker setup can run in M1 or M2-based macOS with Docker Desktop. In contrast, the VM ones cannot run due to the limited support of VirtualBox for ARM64.

We can use Docker Buildx to create the testing environment images for multiple platforms with only one machine, then upload them to Docker Hub for reusing. Developers do not need to be aware of the different processor architectures when starting the container-based testbed. In contrast, VM monitors like VirtualBox usually do not have a centralized image hub. The developer must choose the right image based on the architecture before running the VM testbed.

B. Performance

We use the latest Docker Engine to run the containers and the latest VirtualBox 7 to run the Virtual Machines. The table reflects the average situation for all three implementations. It shows that our container solution significantly

reduces the fresh boot time for the whole VPN system by nearly 90%. It dramatically reduces memory consumption by nearly 94% as well.

C. Security and Limitations

We can't visit the host network interfaces inside containers, and vice versa. The container network stacks are isolated from the host machine.

There are also some limitations of the Docker networking model, which cause several observable differences compared to VMs. However, these limitations generally have workarounds to bypass and will not stop us from adopting container solutions.

That's all I have learned and implemented during my research journey.

CONCLUSION

Summarize the main points:
So, in conclusion, today we've discussed docker container networking systems, and we finally chose to use the MACVLAN network driver and add the net_admin capabilities to containers for realizing routing and firewalls. IPv6 is also supported in Docker. Second, we talked about implementing the testbed environment for both site-to-site and host-to-host VPN systems in IPv4 and IPv6. Then we evaluate by comparing VirtualBox-based Vagrant with Docker Compose in realizing the same networking features. The result shows that migrating the test from the Virtual Machines to the Docker containers can save lots of CPU time and memory for the VPN system, increasing a lot of usability and Portability. At the same time, the host machine still has no security risk increase, and there's always a workaround for the limitations of the Docker networking model.

Estimate feasibility:
Containers are much more lightweight than virtual machines. As a mature virtualization technology, containers can realize every functionality we require, similar to virtual machines. I think the container is a suitable replacement for virtual machines in implementing virtual network systems.

Return to the problem or need:
I hope the achievement of this study can inspire researchers and engineers to migrate their testing environment related to network systems from VMs into containers.

Encourage questions:
Thank you for your attention! You can scan the QR code if you want to have my code for the case study implementation. If you have any questions, I would now be happy to answer them.

Your evaluation demonstrates significant performance improvements when using containers compared to VMs. However, you only provided data on boot time and memory consumption. Did this migration have the same effects on the other metrics, such as Storage usage, CPU usage and network throughput?

I have considered those other metrics as well. However, generally, there are no significant differences for those other metrics between VM ones and container ones. Theoretically, container ones should perform slightly better than VMs because VMs run their own kernel instances. In contrast, containers share the kernel with the host machine.

For example, there are actually few storage usage differences if the amount of software installed on the container image equals the VM image. The only difference is that VM images have a working kernel binary and drivers.

VMs should perform worse than containers for CPU usage and network throughput because of the kernel overhead.

In conclusion, as the paper focuses on the implementations as well as the security and limitations, and it's generally a recognized conclusion that containers can perform better than VMs, I didn't use all the other metrics but chose the metrics that can significantly represent the user experience instead.

While your paper focuses on the benefits of using containers over VMs for testbed networks, are there any specific use cases or scenarios where VM-based testbeds might still be more advantageous or necessary?

I don't think there should exist many specific use cases or scenarios where VM-based testbeds might still be more advantageous or necessary, except, because of the limitations I mentioned for the Docker networking model, you may not have access to docker containers and thus can't change the IP addresses manually to do the workaround, in this case, VMs may be a good choice.

In addition, Docker network interfaces usually have a preconfigured IP address, which may not be very convenient for DHCP implementations as network interfaces should come with no IP address in this case.

The only security concern you mentioned is when we run containers inside containers. Did you encounter any other security concerns to worry about compared to VMs?

No, I didn't. But I can't guarantee there are no vulnerabilities for the kernel namespace, control groups, and SELinux. As a result, generally, VMs are considered more secure than containers since they provide another layer of security by having their own instances of the kernel.

Is there any limitation to using MacVLAN? If we don't need L2 protocols and CPU is not our bottleneck, is there an advantage to using IPvlan?

MACVLAN is suitable for most scenarios. However, suppose you want advanced flood control and forwarding database manipulation. In that case, you should use the bridge driver since these functionalities are specific to the bridge driver.

I don't think there are many scenarios to use IPVLAN unless you run an advanced network scenario, such as advertising the service you run in the container with the Bird Internet routing, which is the BGP daemon, running in the same container.