



Aalto University
School of Science

Implementing a Virtual Network System among Containers

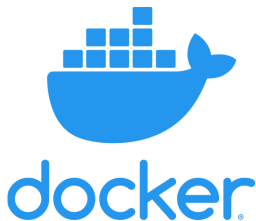
Songlin Jiang
songlin.jiang@aalto.fi

Tutor: Tuomas Aura

April 21, 2023

Introduction

- Virtual machines (VM) are commonly used for **building and testing** network system configurations.
- **containers** can replace **VMs** to overcome VMs' drawbacks. However, **no one has tried** to realize that using **Docker**.
- Rise of **container technologies** has attracted great attention.



Overview

1 Docker Networking System

- Network Drivers
- Routing, Firewall and IPv6

2 Case Study: VPN

- VPN Overview
- Host to Host
- Site to Site in IPv6

3 Evaluation

- Usability and Portability
- Performance
- Security and Limitations

4 Conclusion

5 Q&A



Network Drivers

- Docker employs a pluggable networking subsystem.
- Default drivers for Docker networking system include the **bridge**, **host**, **overlay**, **IPVLAN**, **MACVLAN**, and **none**.
- Possible candidates for implementing the virtual network system on one host machine are **bridge**, **IPVLAN**, and **MACVLAN**.
 - The **none** driver disables all networking.
 - The **host** driver shares the same network stack with the host.
 - The **overlay** driver creates a distributed network system among multiple Docker daemon hosts.

MACVLAN

We choose **MACVLAN** finally:

Table: Network Drivers Comparison

Items	bridge	IPVLAN	MACVLAN
Resources	High	Low	Lowest
MAC Address	Different	Same	Different
VM Migration	No	No	Yes

- MACVLAN is a trivial bridge that does not need to learn as it already knows every **MAC address** it can receive.
- MACVLAN also supports **address configuration** and **discovery protocols**, as well as other **multicast protocols**.

Routing, Firewall and IPv6

- By default, Docker does not allow manipulating container network devices and setting routing tables or firewalls inside containers.
 - However, **net_admin** capability in Linux allows us to:
 1. **Make interface configuration.**
 2. **Administrate IP firewall, masquerading, and accounting.**
 3. **Modify routing tables.**
 4. Bind to any address for transparent proxying.
 5. Set the type-of-service (TOS).
 6. Clear driver statistics.
 7. Set the promiscuous mode.
 8. Enable multicasting.
 9. Use `setsockopt(2)` to set several socket options.
 - **IPv6** is also supported in Docker containers.
-

Overview

1 Docker Networking System

- Network Drivers
- Routing, Firewall and IPv6

2 Case Study: VPN

- VPN Overview
- Host to Host
- Site to Site in IPv6

3 Evaluation

- Usability and Portability
- Performance
- Security and Limitations

4 Conclusion

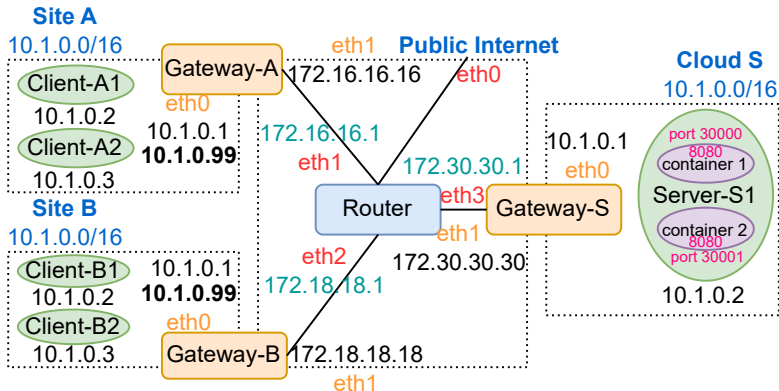
5 Q&A



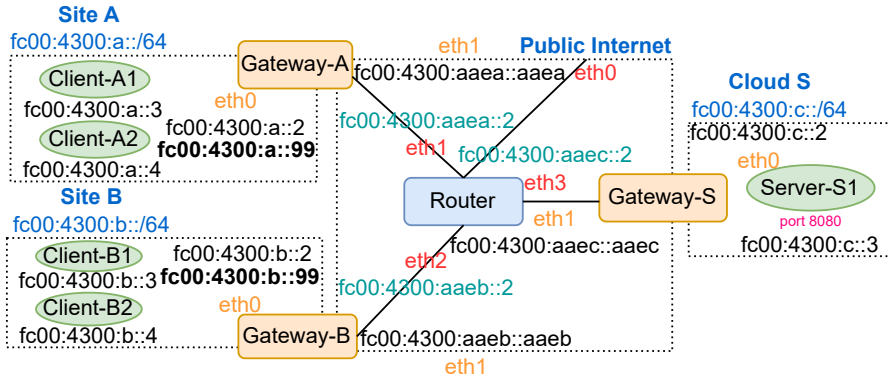
VPN Overview

- Simulate IoT devices in **sites A, B** connect to server in **cloud S**.
 - **Site A, B**, and **cloud S** both use the private IP addresses to improve security and save the IPv4 addresses.
 - **Gateways A, B, S** connect **sites A, B, S** to the public Internet.
 - The **router** represents the Internet between the sites and cloud.
 - The address space between the **gateway** and **router** simulates public, routable IPv4 addresses, although they are all private.
 - **Site A, B**, and **cloud S** use the **router** to access the Internet.
- We experiment with 2 types of VPN using strongSwan (IPsec):
 1. **Site to Site**
 2. **Host to Host**

Host to Host



Site to Site in IPv6



Overview

1 Docker Networking System

- Network Drivers
- Routing, Firewall and IPv6

2 Case Study: VPN

- VPN Overview
- Host to Host
- Site to Site in IPv6

3 Evaluation

- Usability and Portability
- Performance
- Security and Limitations

4 Conclusion

5 Q&A



Usability and Portability

With **Docker Compose**, the Docker orchestration tool, containers **outperforms** VirtualBox-based Vagrant in many aspects:

Table: Functionalities Comparison

Items	Vagrant + VirtualBox	Docker Compose
Resources	Heavy	Lightweight
Kernel	Own	Shared (Namespaces)
Scalability	Hard	Easy
M1/M2 Support	Limited	Fully
Image Hub	Unavailable	Available (Docker Hub)
Seamless	No	Yes

Performance

Table below shows the metrics for the performance evaluation:

Table: Performance Test Result in Average

Solution	Boot Time ¹	Memory ²
Docker Compose	75 s	278 MB
Vagrant + VirtualBox	689 s	4.5 GB

The container based solution *reduces*:

- 1. Fresh boot time** by nearly 90%
- 2. Memory consumption** by nearly 94%

Security and Limitations

- The container network stacks are **isolated** from host.
- Limitations of the Docker networking model include:
 1. Disallowing to assign **overlapped IP address ranges** by Docker, even for network interfaces that won't directly connected.
 2. Disallowing to assign **IP address ending in ".1"** by Docker, as these addresses are reserved by Docker for gateways or routers.
- However, we can always choose to configure the IP addresses manually inside containers to bypass these limitations.

Overview

1 Docker Networking System

- Network Drivers
- Routing, Firewall and IPv6

2 Case Study: VPN

- VPN Overview
- Host to Host
- Site to Site in IPv6

3 Evaluation

- Usability and Portability
- Performance
- Security and Limitations

4 Conclusion

5 Q&A



Conclusion

- It's possible to have a virtual network system with Docker!
- Containers are much more **lightweight** than virtual machines.
- Containers **realized all functionalities** for VPN case study.
- Container is a suitable **replacement** for virtual machines.



Thanks for listening!

Any Questions?



Figure: Scan to get the case study implementation source code
