Computer networks and the Internet — TTM4200

A. Murad, D. Palma, S. Fuglesang, C. Lewin

Contents

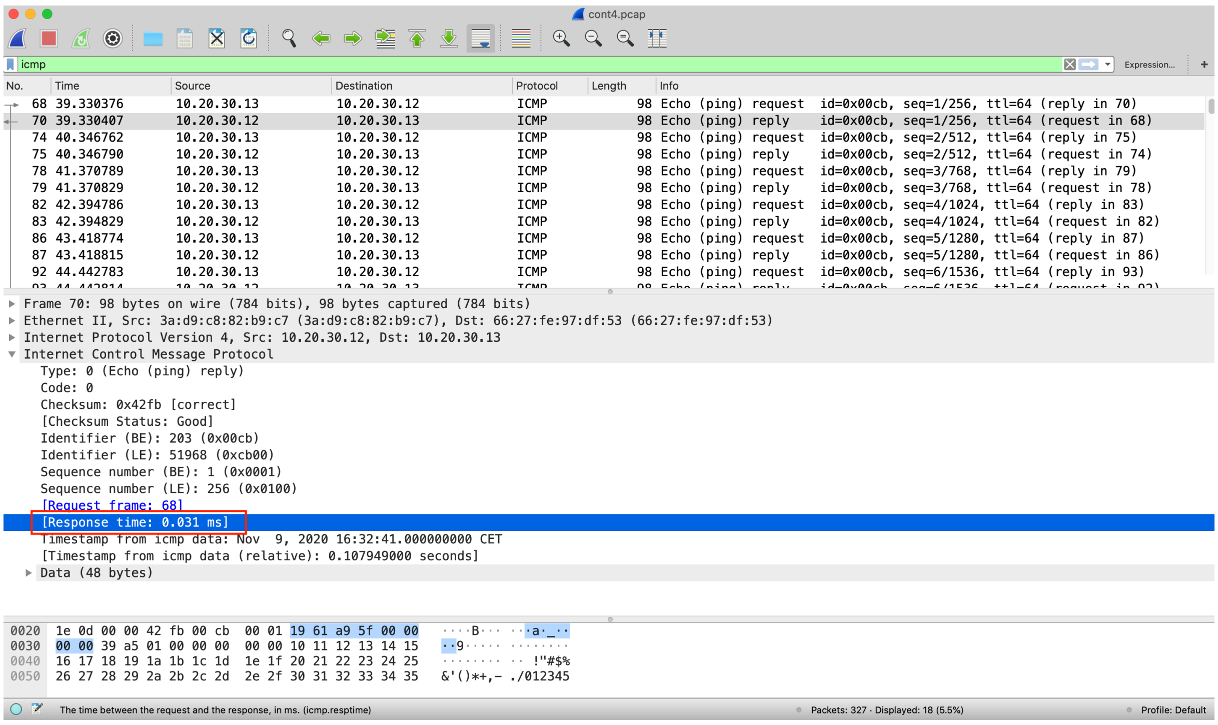
Introduction

The goal of this lab is to experiment with the main tools we are going to use in this course, including Docker, Docker Compose, Tcpdump, Wireshark and Tmux. The lab has several **milestones**. Make sure you reach each one before advancing to the next.

For delivery, submit a PDF report where you answer **only** those steps that are marked with **REPORT(%)**. Additionally, you should submit the codes or capture files, if they were **explicitly** asked for. The percent point gives you an indication of the score of that question. Lab1 counts for **2 points** of your final score in this course.

The answers should be clear and explicit. First, write the section number and title, then question number and the question itself, and then your answer. The screenshots should be annotated. You should provide a single screenshot for each question, i.e. do **not** refer to a single screenshot for multiple questions. An example of an excellent answer:

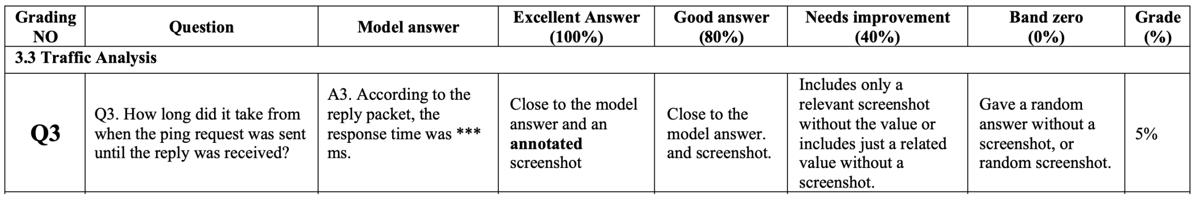
```
3.3 Traffic Analysis
Q3. How long did it take from when the ping request was sent until the reply was received?
A3. According to the reply packet, the response time was 0.031ms, as shown in the
following screenshot.
```



```
cont4.pcap
icmp
No.  Time       Source       Destination  Protocol  Length  Info
  68 39.330376  10.20.30.13  10.20.30.12  ICMP      98 Echo (ping) request  id=0x00cb, seq=1/256, ttl=64 (reply in 70)
  70 39.330407  10.20.30.12  10.20.30.13  ICMP      98 Echo (ping) reply    id=0x00cb, seq=1/256, ttl=64 (request in 68)
  74 40.346762  10.20.30.13  10.20.30.12  ICMP      98 Echo (ping) request  id=0x00cb, seq=2/512, ttl=64 (reply in 75)
  75 40.346790  10.20.30.12  10.20.30.13  ICMP      98 Echo (ping) reply    id=0x00cb, seq=2/512, ttl=64 (request in 74)
  78 41.370789  10.20.30.13  10.20.30.12  ICMP      98 Echo (ping) request  id=0x00cb, seq=3/768, ttl=64 (reply in 79)
  79 41.370829  10.20.30.12  10.20.30.13  ICMP      98 Echo (ping) reply    id=0x00cb, seq=3/768, ttl=64 (request in 78)
  82 42.394786  10.20.30.13  10.20.30.12  ICMP      98 Echo (ping) request  id=0x00cb, seq=4/1024, ttl=64 (reply in 83)
  83 42.394829  10.20.30.12  10.20.30.13  ICMP      98 Echo (ping) reply    id=0x00cb, seq=4/1024, ttl=64 (request in 82)
  86 43.418774  10.20.30.13  10.20.30.12  ICMP      98 Echo (ping) request  id=0x00cb, seq=5/1280, ttl=64 (reply in 87)
  87 43.418815  10.20.30.12  10.20.30.13  ICMP      98 Echo (ping) reply    id=0x00cb, seq=5/1280, ttl=64 (request in 86)
  92 44.442783  10.20.30.13  10.20.30.12  ICMP      98 Echo (ping) request  id=0x00cb, seq=6/1536, ttl=64 (reply in 93)

▶ Frame 70: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
▶ Ethernet II, Src: 3a:d9:c8:82:b9:c7 (3a:d9:c8:82:b9:c7), Dst: 66:27:fe:97:df:53 (66:27:fe:97:df:53)
▶ Internet Protocol Version 4, Src: 10.20.30.12, Dst: 10.20.30.13
▼ Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x42fb [correct]
    [Checksum Status: Good]
    Identifier (BE): 203 (0x00cb)
    Identifier (LE): 51968 (0xcb00)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Request frame: 68]
    [Response time: 0.031 ms]
    Timestamp from icmp data: Nov  9, 2020 16:32:41.000000000 CET
    [Timestamp from icmp data (relative): 0.107949000 seconds]
  ▶ Data (48 bytes)

0020  1e 0d 00 00 42 fb 00 cb  00 01 19 61 a9 5f 00 00   ····B··· ··a·_··
0030  00 00 39 a5 01 00 00 00  00 00 10 11 12 13 14 15   ··9····· ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········ ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345

The time between the request and the response, in ms. (icmp.resptime)    Packets: 327 · Displayed: 18 (5.5%)    Profile: Default
```

Figure 1: Annotated Screenshot

We will score your answers according to a rubric, for example:



Grading NO	Question	Model answer	Excellent Answer (100%)	Good answer (80%)	Needs improvement (40%)	Band zero (0%)	Grade (%)
3.3 Traffic Analysis							
Q3	Q3. How long did it take from when the ping request was sent until the reply was received?	A3. According to the reply packet, the response time was *** ms.	Close to the model answer and an **annotated** screenshot	Close to the model answer. and screenshot.	Includes only a relevant screenshot without the value or includes just a related value without a screenshot.	Gave a random answer without a screenshot, or random screenshot.	5%

Figure 2: Rubric for Q3

Setting up Virtual Machine

- Start by downloading and importing the VM (Link) and configure it according to you computer's specifications (as you did in TTM4175. Additionally, to access files of your computer from within the VM, you should share folders between them. Figure 3 summarizes the process of setting up the VM (adapt the settings to your own case).
- Start the VM and login using the password "ttm4200". Do not upgrade if prompted to upgrade to newer Ubuntu version. Change the password if you haven't done so (`passwd`).
- By default, the shared folder will be mounted to "/media/sf_shared" (if Auto-mount is selected). Add the user "ttm4200" to "vboxsf" group, then reboot your VM:

```
sudo usermod -a -G vboxsf ttm4200
sudo reboot
```

> *You can mount the shared folder in any directory of your choice (random guide).*

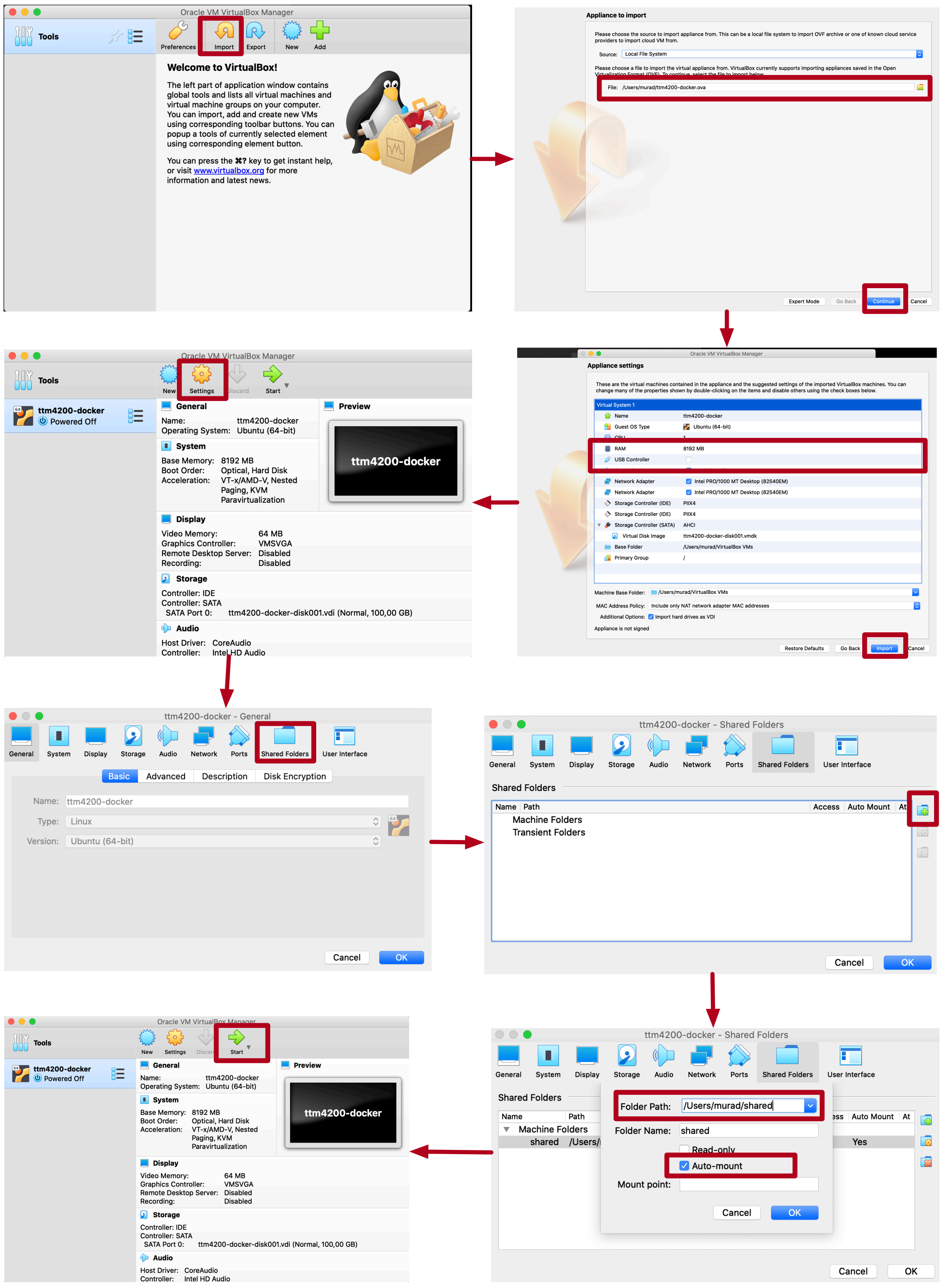
- Before starting lab1, you are encouraged to remove any containers and networks that you may have built before in your VM. This is to prevent unnecessary conflicts later in this lab:

```
docker rm $(docker ps -aq)
docker network prune
```

- From Blackboard, download "lab1.zip" to your VM and place it in your home directory ("/home/ttm4200"). Unzip this folder and navigate to it:

```
cd /home/ttm4200
unzip lab1.zip && cd lab1
```

> *All the provided commands in this lab assume that your present working directly (`pwd`) is "/home/ttm4200/lab1", and they won't work if you are inside another directory.*



Welcome to VirtualBox!

The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

You can press the ⌘? key to get instant help, or visit www.virtualbox.org for more information and latest news.

Appliance to import

Please choose the source to import appliance from. This can be a local file system to import OVF archive or one of known cloud service providers to import cloud VM from.

Source: Local File System

Please choose a file to import the virtual appliance from. VirtualBox currently supports importing appliances saved in the Open Virtualization Format (OVF). To continue, select the file to import below.

File: /Users/murad/ttm4200-docker.ova

Appliance settings

These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

Virtual System 1	
Name	ttm4200-docker
Guest OS Type	Ubuntu (64-bit)
CPU	1
RAM	8192 MB
USB Controller	☐
Network Adapter	☑ Intel PRO/1000 MT Desktop (82540EM)
Network Adapter	☑ Intel PRO/1000 MT Desktop (82540EM)
Storage Controller (IDE)	PIIX4
Storage Controller (IDE)	PIIX4
Storage Controller (SATA)	AHCI
Virtual Disk Image	ttm4200-docker-disk001.vmdk
Base Folder	/Users/murad/VirtualBox VMs
Primary Group	/

Machine Base Folder: /Users/murad/VirtualBox VMs

MAC Address Policy: Include only NAT network adapter MAC addresses

Additional Options: ☑ Import hard drives as VDI

Appliance is not signed

General
Name: ttm4200-docker
Operating System: Ubuntu (64-bit)

System
Base Memory: 8192 MB
Boot Order: Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display
Video Memory: 64 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage
Controller: IDE
Controller: SATA
SATA Port 0: ttm4200-docker-disk001.vdi (Normal, 100,00 GB)

Audio
Host Driver: CoreAudio
Controller: Intel HD Audio

Name: ttm4200-docker
Type: Linux
Version: Ubuntu (64-bit)

Name	Path	Access	Auto Mount	At
Machine Folders				
Transient Folders				

Name	Path	Access	Auto Mount	At
Machine Folders				
shared	/Users/		Yes	

Folder Path: /Users/murad/shared
Folder Name: shared
☐ Read-only
☑ Auto-mount
Mount point:

Figure 3: VM Set up

1 Milestone 1 – Getting Started with Docker

In this lab, you should get acquainted with Docker (https://docs.docker.com/), since we will use it extensively in all future labs. Read this overview (https://docs.docker.com/engine/docker-overview/) as well as the support document to get an idea of what Docker is.

By the end of this milestone, you will build a system as shown in Figure 4 and Figure 5. Refer back to these figures as often as needed to understand how the lab is constructed.

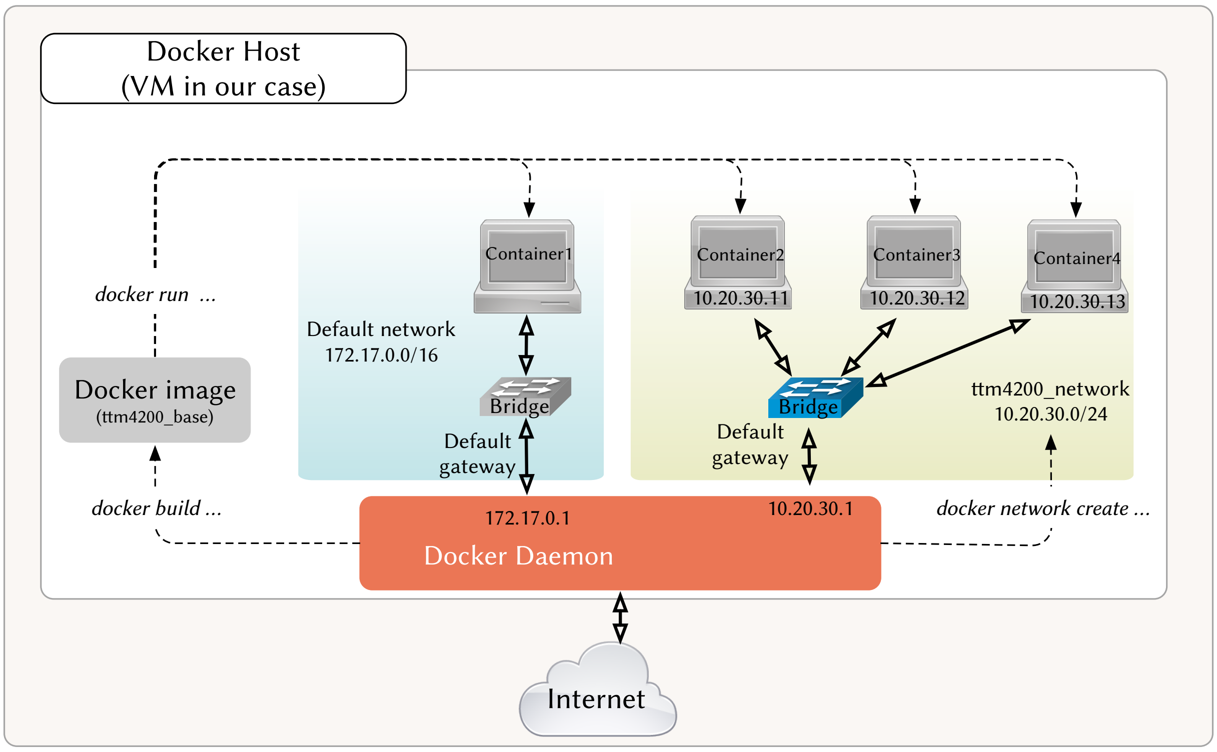
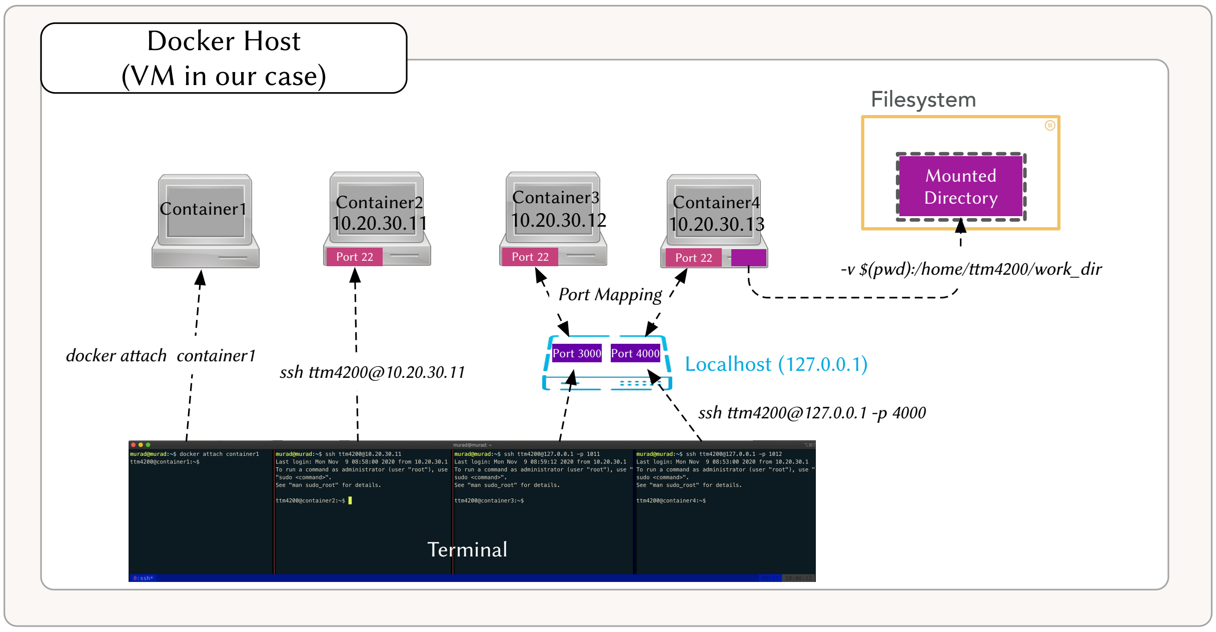


Figure 4: Docker Image, Containers, and Networks



```
murad@murad:~$ docker attach container1
ttm4200@container1:~$
```

```
murad@murad:~$ ssh ttm4200@10.20.30.11
Last login: Mon Nov  9 08:58:00 2020 from 10.20.30.1
To run a command as administrator (user "root"), use
"sudo <command>".
See "man sudo_root" for details.

ttm4200@container2:~$
```

```
murad@murad:~$ ssh ttm4200@127.0.0.1 -p 1011
Last login: Mon Nov  9 08:59:12 2020 from 10.20.30.1
To run a command as administrator (user "root"), use "
sudo <command>".
See "man sudo_root" for details.

ttm4200@container3:~$
```

```
murad@murad:~$ ssh ttm4200@127.0.0.1 -p 1012
Last login: Mon Nov  9 08:53:00 2020 from 10.20.30.1
To run a command as administrator (user "root"), use "
sudo <command>".
See "man sudo_root" for details.

ttm4200@container4:~$
```

Figure 5: Docker Attach, SSH, Port Mapping, and Volumes

1.1 Creating Docker Images

To build a docker image, we need to to write a Dockerfile (https://docs.docker.com/engine/reference/builder/) that contains all the commands for assembling an image. In this course, all Dockerfiles will be provided to you. We will always build our images on top of Ubuntu (https://hub.docker.com/_/ubuntu/).

- Build an image from the Dockerfile in the directory named “ttm4200_base”:

```
docker build --tag ttm4200_base ./ttm4200_base/
```

- Check how many images are built locally:

```
docker images
```

1.2 Running Containers

- Run a container and name it “container1” from the image you have just built. Run the container interactively, and in the background (detached mode):

```
docker run --interactive --tty --detach --name=container1 --hostname=container1 \
    ttm4200_base
```

- Attach a terminal to the container. This will allow you to control the container interactively in your terminal.

```
docker attach container1
```

> *The password for the root user inside the container is set to "ttm4200"*

- Check the container IP address, MAC address, routing table, and ARP table.

```
ip address
ip link
ip route
ip neighbor
```

> *By default, Docker uses the 172.17.0.0/16 subnet for container networking, and assigns the first address (172.17.0.1) to the host (the machine which is running Docker daemon, the VM in your case).*

- Ping the Docker host, then again check the ARP table.
- Detach from the container (press `Ctrl+p` followed `Ctrl+q`). This will turn the container from interactive mode into daemon mode.

  > *If you press `Ctrl+c` or `exit`, you will kill the container. This is one of the common pitfalls (especially in later labs)*

1.3 Docker Network

- Create a new docker network with a specific name and subnet:

```
docker network create --subnet=10.20.30.0/24 ttm4200_net
```

- Run a second container (name it "container2") and connect it to the new network, and assign it a static IP address:

```
docker run -itd --name=container2 --hostname=container2 --net=ttm4200_net \
    --ip=10.20.30.11 ttm4200_base
```

- Connect to the container using SSH. The ssh server is already installed and configured in the Dockerfile. It is configured to accept only a public key authentication to the user "ttm4200", and the private key is already in the VM ("/home/ttm4200/.ssh"). You can connect directly using the container ip address:

```
ssh ttm4200@10.20.30.11
```

1.4 Publishing Container's Ports

By default, a container does not make its port available to the outside world (or the Docker host). To do so, you have to map a container port to a port on the Docker Host.

- Run a third container ("container3") and bind its SSH port to an unused port in your Docker Host (preferably a number above 1024, to avoid conflicts), and make it listen on localhost only (127.0.0.1).

  ```
  docker run -itd --name=container3 --hostname=container3 --net=ttm4200_net \
      --ip=10.20.30.12 -p 127.0.0.1:3000:22 ttm4200_base
  ```

  > *if you don't specify 127.0.0.1 in the port mapping, it will publish the port on all interfaces of the Docker Host (0.0.0.0). This is less ideal from a security perspective.*

- Connect to the container using SSH, but using the published port on the localhost of Docker Host:

  ```
  ssh ttm4200@127.0.0.1 -p 3000
  ```

1.5 Docker Volumes

To preserve data generated by a container, you can either copy them to your host machine or use *Volumes* for persisting data. A Docker volume is basically a mechanism to mount a file system (directory) between a Docker host and a container. Thus, any data generated will be stored automatically in your host machine.

- Run a fourth container ("container4") and mount the "ttm4200_base" directory to a directory in the container:

  ```
  docker run -itd --name=container4 --hostname=container4 --net=ttm4200_net \
    --ip=10.20.30.13 -p 127.0.0.1:4000:22 \
    -v $(pwd)/ttm4200_base:/home/ttm4200/work_dir ttm4200_base
  ```

- Stop and remove all the containers as well as the network.

  ```
  docker stop $(docker ps -aq)
  docker rm $(docker ps -aq)
  docker network prune
  ```

2 Milestone 2 – Docker Compose

In this course, we are going to use Docker extensively to build several different services using containers. For handling multiple containers at once, we need to use Docker Compose. In Docker Compose, we will create a YAML file to configure the application's services, and with a single command, we can create all the images and run the containers at once. Additionally, we will be able to stop and remove all containers and the network in a single command. For further reference, see the documentation from docker-compose.yml (https://docs.docker.com/compose/compose-file/compose-file-v2/), and the support document.

2.1 Creating Docker Compose Files

- Complete the “docker-compose.yml” file that defines exactly all we did in Milestone 1 (see Figure 4 and Figure 5), specifically:
  - Build an image, called “ttm4200_base”, from the Dockerfile in the “ttm4200_base” directory
  - Create a network, called “ttm4200_net”, with the subnet “10.20.30.0/24”.
  - Run “container1” interactively, with a hostname of “container1”, and connected it to the default network.
  - Run “container2” interactively, with a hostname of “container2”, and connected to “ttm4200_net”.
  - Run “container3” interactively, with a hostname of “container3”, connected to “ttm4200_net”, and its SSH port mapped to a port in Docker Host.
  - Run “container4” interactively, with a hostname of “container4”, connected to “ttm4200_net”, its SSH port mapped to a port in Docker Host, and the “ttm4200_base” directory is mounted to “/home/ttm4200/work_dir” inside the container.
  - Additionally, assign IPv6 addresses to containers connected to “ttm4200_net”, within the subnet of “fd00::/64”.
  - To make things easier, the network, building the image, “container1” and “container4” are already configured. You must complete “container2” and “container3”!

```
version: '2.3'
#specify a custom network
networks:
#the name of the created network
    ttm4200_net:
        enable_ipv6: true
        ipam:
            config:
                - subnet: 10.20.30.0/24
                - subnet: "fd00::/64"

services:

    #Building the base image
    image:
        #Build an image from the Dockerfile in the "ttm4200_base" directory
        build: ./ttm4200_base
        #Name the image "ttm4200_base", and use it
        image: ttm4200_base


    container1:
        #use the image "ttm4200_base"
        image: ttm4200_base
        #It will depends the service "image", that built the image
        depends_on:
            - "image"
        #Name of the container "container1"
        container_name: container1
        #Name the host machine "container1"
        hostname: container1
        #Run in "--interactive" mode
        stdin_open: true
        #Run in "--tty", pseudo terminal
        tty: true
        # Add networking capabilities, instead of privileged
        cap_add:
            - NET_ADMIN

    container2:
    ################################################
    # TODO: Your code here!


    ################################################

    container3:
    ################################################
    # TODO: Your code here!


    ################################################

    container4:
    ################################################
        image: ttm4200_base
        depends_on:
            - "image"
        container_name: container4
        hostname: container4
        stdin_open: true
        tty: true
        cap_add:
            - NET_ADMIN
        #map ssh port to a port on Docker Host (port 400)
        ports:
            - "127.0.0.1:4000:22"
        # mount the "ttm4200_base" directory on Docker Host to the container
        volumes:
            - ./ttm4200_base/:/home/ttm4200/work_dir
        #Connect to the network (ttm4200_net and assign IPv4 and IPv6 addresses)
        networks:
            ttm4200_net:
                ipv4_address: 10.20.30.13
                ipv6_address: "fd00::13"

    ################################################
```

> *Docker Compose might use a different default network subnet other than 17.17.0.0/16, depending on if this subnet is already in use*

Q1. **REPORT(20%):** Submit your docker-compose file along with the report. We will check that it is working, and that it corresponds to what is shown in Figure 4 and 5.

2.2 Running Docker Compose Files

- Create and start the containers using the docker-compose file:

```
docker-compose up -d --build
```

  The “-d” option is to run containers in the background (detached mode), and “--build” is to build the images before starting containers.

- Connect to the containers, as shown in figure 5.

```
docker attach container1
ssh ttm4200@10.20.30.11
ssh ttm4200@127.0.0.1 -p 3000
ssh ttm4200@127.0.0.1 -p 4000
```

3 Milestone 3 – Traffic Capture & Analysis Tools

3.1 Terminal multiplexer (`tmux`)

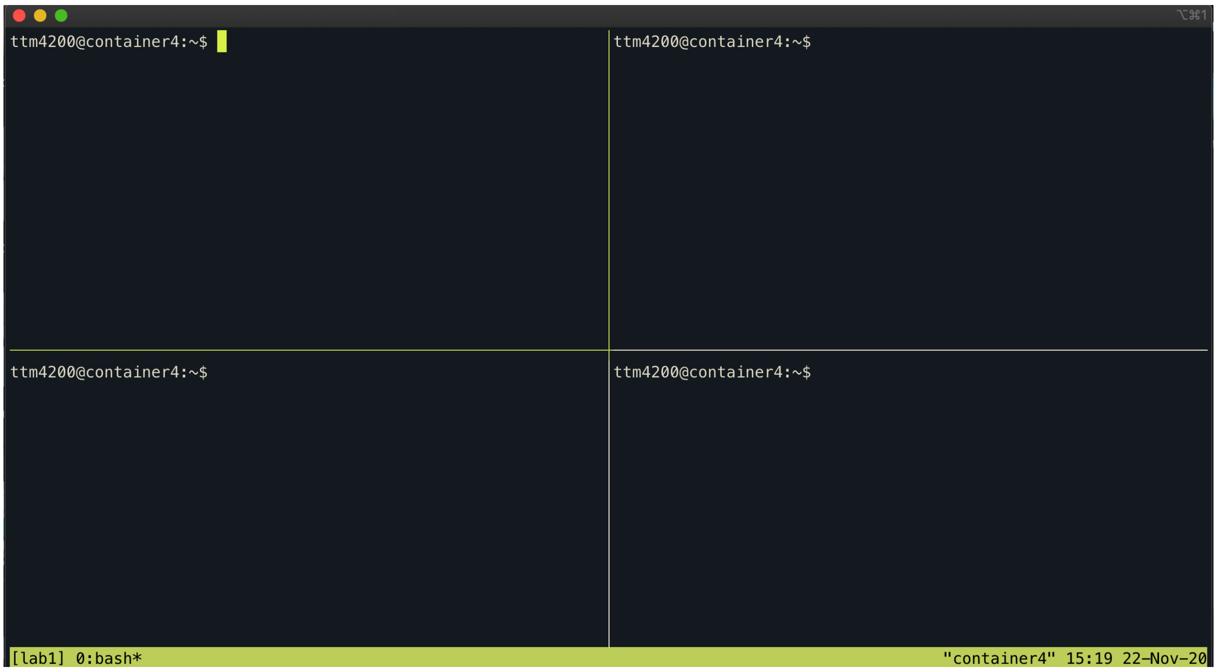
We will use tmux (http://man.openbsd.org/OpenBSD-current/man1/tmux.1) to access multiple terminal sessions simultaneously in a single window. It lets you switch easily between several programs in one terminal. Additionally, you can detach and re-attach from sessions, thus allowing you to run sessions in the background. This is really helpful if you are working on a remote server. Here (https://shortcutworld.com/tmux/linux/tmux_Shortcuts) are the most common `tmux` shortcuts.

- In “container4”, start a new `tmux` session (preferably a named session to distinguish between multiple sessions):

```
tmux new -s [session_name]
```

- Create four panes as shown in Figure 6.

> *To execute a command in* `tmux`*, a prefix must be used before the command (* `<prefix> + command` *). By default, the prefix is* `Ctrl+b`*. For example, to create a vertical split use* `<prefix> + %`*. That is, you have to press the keys* `Ctrl+b` *followed by the command* `%` *(without the control key). To create a horizontal split use* `<prefix> + "`*. To move between panes use* `<prefix> + Arrow Keys`*.*



```
ttm4200@container4:~$           ttm4200@container4:~$

ttm4200@container4:~$           ttm4200@container4:~$

[lab1] 0:bash*                          "container4" 15:19 22-Nov-20
```

Figure 6: Tmux Panes

- Detatch from the current session by using `<prefix> + d`.
- ***Optional:*** If you feel that `tmux` key combinations are awkward, you can customize it according to your personal preferences. For example, to change the prefix from `Ctrl+b` to `Ctrl+a`, simply add the following lines to `tmux` configuration file (".tmux.conf") inside the user's home directory ("/home/ttm4200"):

```
unbind C-b
set-option -g prefix C-a
bind-key C-a send-prefix
```

And to change the command for vertical split from `%` to `\`, horizontal split from `"` to `-`, add the following lines:

```
bind-key \ split-window -h
bind-key - split-window -v
```

And you can do many interesting customizations such as fast pane-switching, pane-resizing, etc. If you are interested see this blog (https://www.hamvocke.com/blog/a-guide-to-customizing-your-tmux-conf/). For these changes to take effect, you have to exit from `tmux`, i.e. kill all active sessions. In this course, to change the `tmux` configuration for all containers, you only need to change ".tmux.conf" inside the folder from which the image is built ("ttm4200_base", in this case). But you have to rebuild the image and containers for these configuration to take effect.

3.2 Traffic Capture

To capture network traffic, we will use tcpdump (https://www.tcpdump.org/manpages/tcpdump.1.html), which is a command-line packet capturing utility that allows us to sniff, capture and monitor any type of traffic on a network easily.

- In "container4", attach to the previously created session:

```
tmux attach -t [session_name]
```

- In one pane, start packet capturing with tcpdump and dump the captured packets to a file:

```
sudo tcpdump -i eth0  -w work_dir/container4.pcap
```

- Switch to the other pane `Ctrl+b arrow key`, ping and traceroute "container3" using IPv4 and IPv6 addresses.

```
ping 10.20.30.12
ping -6 fd00::12
traceroute 10.20.30.12
```

> *We highly recommend getting comfortable with using tmux, as it will be worth the time you spent on learning it. Otherwise, you can just open multiple SSH connections*

- Stop the tcpdump capturing and copy the capture file to your host machine. You can use scp (https://haydenjames.io/linux-securely-copy-files-using-scp/):

```
scp -P 4000 ttm4200@127.0.0.1:~/work_dir/container4.pcap ./
```

> *Since the "ttm4200_base" directory is mounted on "container4" in "work_dir", any file*

> *saved to the "work_dir" inside the container, will be directly saved to the "ttm4200_base" folder*

- Using the docker-compose file, stop and remove the containers, as well as the network. You can do so with only one command:

```
docker-compose down
```

Q2. **REPORT(10%):** Submit your capture file "container4.pcap" along with the report. We will check it and make sure that it is unique. So please don't copy it from other teams.

3.3 Traffic Analysis

To analyze network traffic, we will use Wireshark (https://www.wireshark.org/), which is a free and open-source packet analyzer. You can refer to the course's book in "Chapter 1: Wireshark Lab: Getting Started", to become familiar with Wireshark.

- In your VM, open the file "container4.pcap" in Wireshark.
- Examine one ping request and its reply using "icmp" filter. Then answer these questions and validate your answer with an **annotated** screenshot from Wireshark.

  Q3. **REPORT(5%):** How long did it take from when the ping request was sent until the reply was received?

  Q4. **REPORT(5%):** Does the ICMP packet have source and destination port numbers?

  Q5. **REPORT(10%):** What are the ICMP type and code numbers in the ping request? What other fields does this ICMP packet have? How many bytes are the checksum, sequence number and identifier fields?

  Q6. **REPORT(10%):** What are the ICMP type and code numbers in the corresponding reply? What other fields does this ICMP packet have? How many bytes are the checksum, sequence number and identifier fields?

  Q7. **REPORT(10%):** Explain why there are ICMP messages: Destination Unreachable (Port unreachable), coming from your destination.

  > *These messages correspond to* `traceroute` *command. To find the reason, search for the working principles of* `traceroute`*, especially the value of the used port number*

- Examine one ping request and its reply using "icmpv6" filter. Then answer these questions and validate your answer with an **annotated** screenshot from Wireshark.

  Q8. **REPORT(5%):** How long did it take from when the ping request was sent until the reply was received?

Q9. **REPORT(5%):** Does the ICMP packet have source and destination port numbers?

Q10. **REPORT(10%):** What are the ICMP type and code numbers in the ping request? What other fields does this ICMP packet have? How many bytes are the checksum, sequence number and identifier fields?

Q11. **REPORT(10%):** What are the ICMP type and code numbers in the corresponding reply? What other fields does this ICMP packet have? How many bytes are the checksum, sequence number and identifier fields?

4 Optional Exercise

This exercise is to test your understanding of docker. It is not connected to the previous milestones.

- Using docker-compose, build images and run containers from Dockerfile in the directories named “node-server” and “client”. You can complete the following “docker-compose.yml” file to do so:

```
version: '2.3'
networks:
    ttm4200_net:
        enable_ipv6: true
        ipam:
            config:
                #configure subnet range (both IPv6 and IPv4)
                ========== FILL IN HERE ==========
services:
    node:
        build: ./node-server/
        container_name: node-server
        hostname: node-server
        cap_add:
            - NET_ADMIN
        ports:
            #map port 8080 on the host machine to port 8080 in the container
            ========== FILL IN HERE ==========
        networks:
            ttm4200_net:
                #assign the node service an IPv6 address and IPv4 address
                ========= FILL IN HERE =========d
    client:
        build: ./client
        container_name: client
        hostname: client
        user: root
        tty: true
        stdin_open: true
        cap_add:
            - NET_ADMIN
        volumes:
            - ./client/:/home
        networks:
            ttm4200_net:
                #assign the client service an IPv6 address and IPv4 address
                ========== FILL IN HERE =========
```

- Verify that both containers are running, and that a new network has been created.

> *You will have a network conflict if you have not removed the network in Milestone 3 (i.e. `docker-compose down`), since you are trying to build two networks with the same name.*

- Attach to client-container.
- Ping the node-server, using the name you assigned it when you ran the container.
- Ping the node-server again, this time using its IP-address.
- Test the node-server by typing “localhost:8080” in a browser of your choice (**needs to be a browser on the machine which is running Docker**).
- Download the .html-file from the node-server using the “wget” command

```
    wget <node-server ip-address>:8080
```

- Edit the “index.html” file located in “/node-server/views” to say “Hello from group <your group ID>”, instead of “Hello World!”.
- Copy the newly edited “index.html” file to the node-server container, and use the path “/app/views/index.html”.

> *You can either use `docker cp` command or the mounted volume.*

- Access “localhost:8080” in the browser.