Docker

Nikita Neveditsin, SMU, 2019

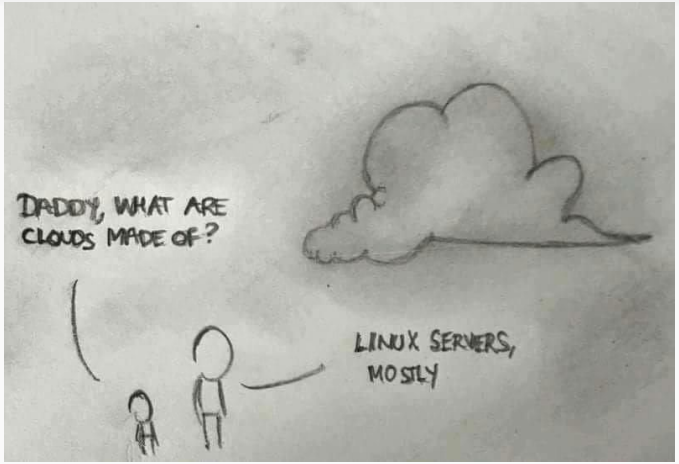
nikita.neveditsin@smu.ca

Docker: what is it?

Docker is a computer program that performs operating-system-level virtualization, also known as "containerization" [wiki]
Also used for automatic deployment of applications

Advantages of virtualization

- Electricity saving
- Saving money for hardware
- Better security
- No need to reboot on desktops/laptops to run other OS
- Can run old operating systems and software on new hardware
- Easy to make restore points
- Easy to migrate to other hardware
- Cloud services (IaaS, PaaS, SaaS)



[1]

Brief history of virtualization

- Virtualization is not something new - virtual machines have been used since 1960s (IBM IBM System/360-67, VM-370);
- Gerald J. Popek and Robert P. Goldberg in their 1974 article "Formal Requirements for Virtualizable Third Generation Architectures" introduced virtualization requirements for computer architecture [1]. Roughly speaking, CPU should support passing control to **VMM** (or **hypervisor**) when guest machine executes “**sensitive CPU instructions**” (Supervisor mode)

Brief history of virtualization

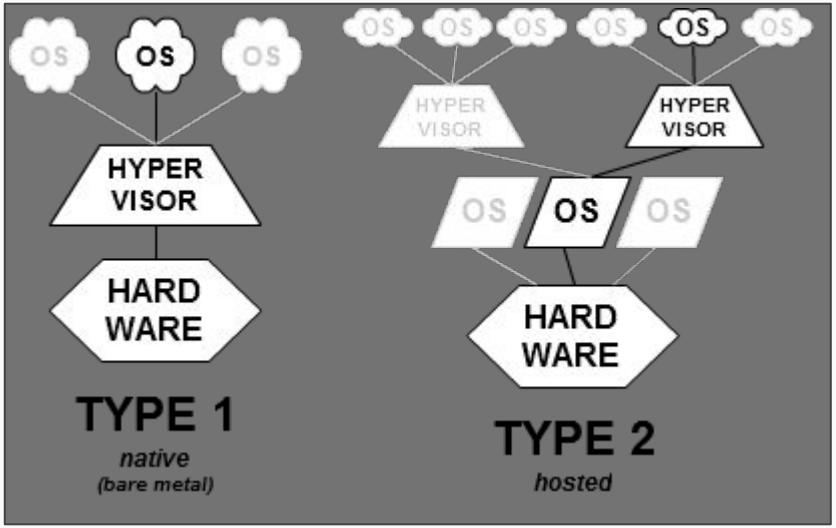
- Popek-Goldberg virtualization requirements were not met by x86 architecture until 2005 when Intel introduced **Intel VT(-x)** technology and AMD introduced **AMD-V**. So, history of hardware virtualization on x86 platform starts from the year 2005 only. Modern CPUs support Hypervisor mode (Ring -1) instructions.
- Before 2005, VMWare supported virtualization using **binary translation** (closer to *emulation*)

Brief history of virtualization

Virtualization != Emulation

Virtualization: definitions

Hypervisor = Virtual Machine Monitor: creates, runs and manages VM



2

Virtualization: definitions

- **Host machine** - the machine which hosts virtual machines and runs hypervisor
- **Guest machine** - the virtual machine itself
- **Nested virtualization** - when virtual machine acts as a host machine for other virtual machines

Virtualization: definitions

- **Full Virtualization** - classic virtualization. Guest operating system **does not know** that it is being run on a virtual machine
- **Paravirtualization** - Guest operating system **knows** that it is being run on a virtual machine. Mostly used for devices

Virtualization: performance

- If your app running on VM is **CPU-intensive** (e.g., heavy computations), then performance hit is usually very low (~1%)
- If your app uses a lot of **I/O** operations (databases, intensive networking, etc.), then you may lose up to 30% comparing to non virtualized environment. However, there are some technologies allowing to reduce performance loss with IO (like SR-IOV, IOMMU (AMD-Vi and VT-d))

Virtualization: software

- VMWare Workstation/ VMWare Workstation Player - for desktops (Windows/Linux), player is freeware: type 2 hypervisor

- VMWare ESXi - for servers: type 1 hypervisor

Virtualization: software

- Oracle VirtualBox - for desktops (Windows/Linux/Mac/Solaris), freeware: type 2 hypervisor

- Oracle VM Server - for servers: type 1 hypervisor

Virtualization: software

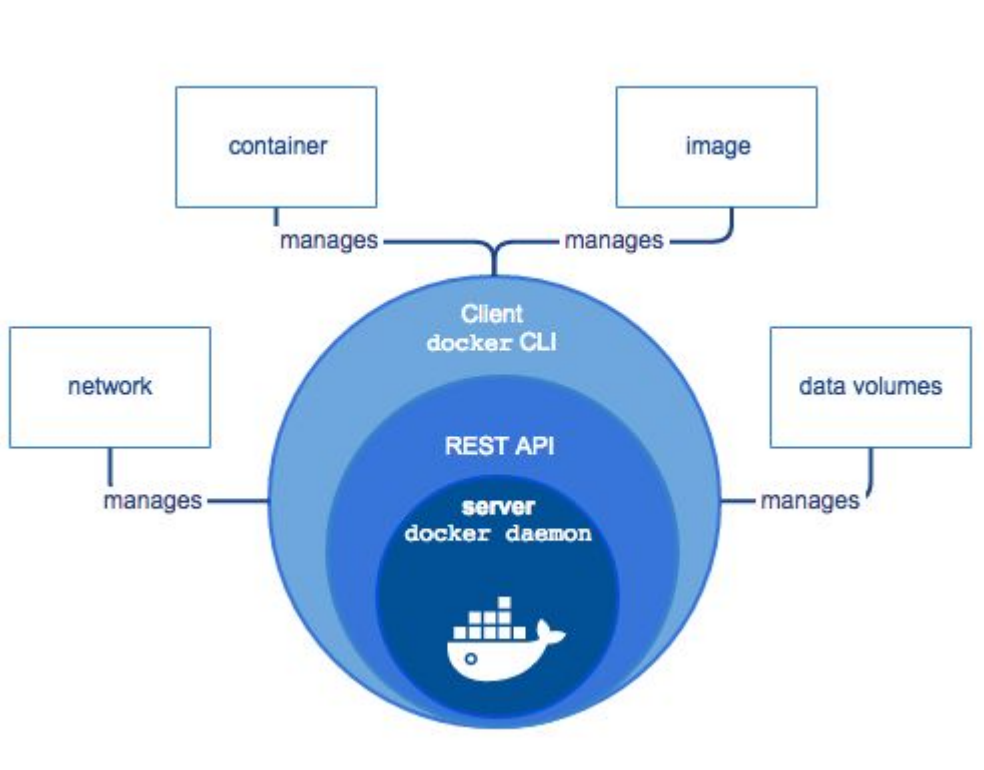
- Microsoft Hyper-V - for desktops/servers (Windows Server 2008+, Windows 8+ (only selected editions)).
- How to enable on Windows 10:

https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v

Virtualization: software

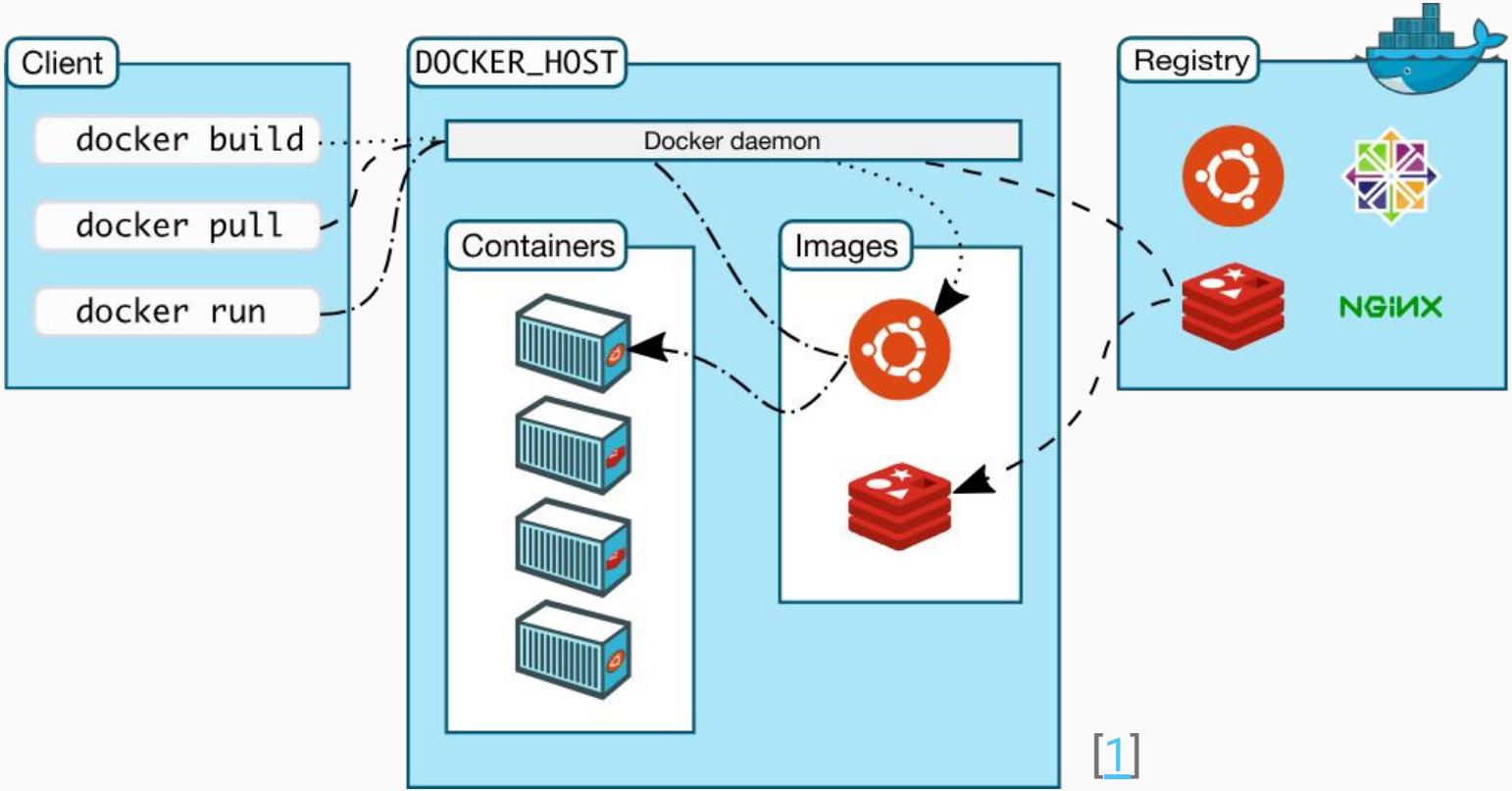
- Xen - type 1 hypervisor, open-source
- KVM (+QEMU) - type 2 hypervisor, Linux kernel module, open-source

Docker: architecture



[1]

Docker: architecture (cont-d)



[1]

Docker: just a service (daemon)

t_test@lnx: ~

```
t_test@lnx:~$ systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2019-02-11 15:21:09 UTC; 25min ago
     Docs: https://docs.docker.com
 Main PID: 892 (dockerd)
    Tasks: 20
   Memory: 29.9M
      CPU: 2.183s
   CGroup: /system.slice/docker.service
           └─892 /usr/bin/dockerd -H fd://
t_test@lnx:~$
```

Docker: underlying technologies

- Control groups (since 2007): resource usage limits
- Union file system (since 2004): implements union mounts
- Linux Namespaces (since 2002): isolation of processes, mounts, etc.

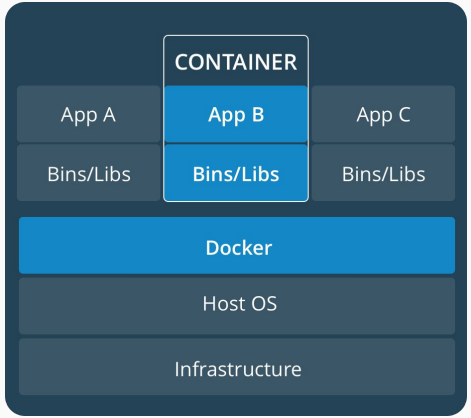
Docker: alternatives

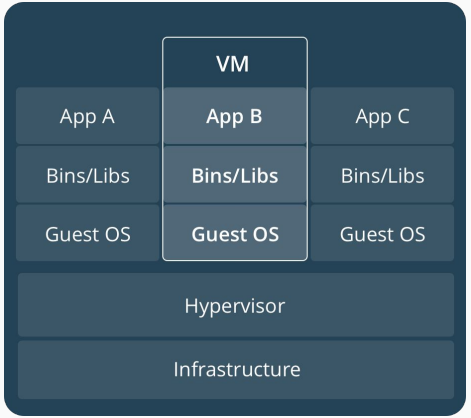
- LXC (since 2008)
- FreeBSD Jails (since 2000)
- chroot (since 1982)
- And others...

Docker: docker VS virtual machine

A **container** runs **natively on Linux** and **shares the kernel of the host machine** with other containers. It runs a discrete process, taking no more memory than any other executable, making it lightweight.

By contrast, a virtual machine (VM) runs a full-blown “guest” operating system with virtual access to host resources through a hypervisor. In general, VMs provide an environment with more resources than most applications need [1]





[1]

Docker: concepts. Containers VS Images

An **image** is a **read-only template** with instructions for creating a Docker container. An image is an executable package that includes everything needed to run an application: the code, a runtime, libraries, environment variables, and configuration files. Often, an image is *based on another image*, with some additional customization [1]

A **container** is a **runtime instance** of an image - what the image becomes in memory when executed (that is, an image with state, or a user process). You can see a list of your running containers with the command, **docker ps**, just as you would in Linux. A container is defined by its image as well as any configuration options you provide to it when you create or start it. **When a container is removed, any changes to its state that are not stored in persistent storage disappear** [1]

Docker: application deployment: Dockerfile

Dockerfile defines what goes on in the environment inside your container. Access to resources like networking interfaces and disk drives is virtualized inside this environment, which is isolated from the rest of your system, so you need to map ports to the outside world, and be specific about what files you want to “copy in” to that environment. However, after doing that, you can expect that the build of your app defined in this Dockerfile behaves exactly the same wherever it runs [1]

Docker: create an image from a Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org Flask Redis
# Make port 80 available to the world outside this container
EXPOSE 80

# Run app.py when the container launches
CMD ["python", "app.py"]
```

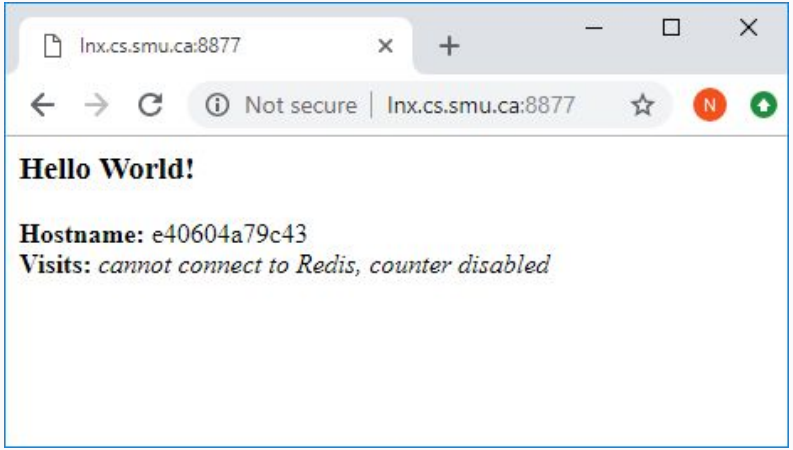
Exercise

Build your first docker image and run a container using the following commands.

```
mkdir do
cd do
wget lnx.cs.smu.ca/docker/Dockerfile
wget lnx.cs.smu.ca/docker/app.py

docker build -t USER_friendlyhello .

docker run -p MMDD:80 friendlyhello
Or run detached:
docker run -d -p MMDD:80 friendlyhello
```



Where **USER** is your username, **MM** is your month of birth and **DD** is day of birth
Try to access the app from your browser

Docker: stop container

t_test@lnx: ~/do

```
t_test@lnx:~/do$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
        NAMES
e40604a79c43        friendlyhello       "python app.py"     36 seconds ago      Up 35 seconds       0.0.0.0:8877->8
0/tcp   jovial_mccarthy
t_test@lnx:~/do$ docker container stop e40604a79c43
e40604a79c43
t_test@lnx:~/do$
```

Docker: remove container

- If you **stop** the container, you can find it in using **docker container ps -a** command. You can start it again using **docker start** command (even after reboot)

If you remove a container, you will lose any changes that you made inside it

t_test@lnx: ~/do

```
t_test@lnx:~/do$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                      PORTS
AMES
e40604a79c43        friendlyhello       "python app.py"     6 minutes ago       Exited (137) 5 minutes ago
ovial_mccarthy
t_test@lnx:~/do$ docker container rm e40604a79c43
e40604a79c43
t_test@lnx:~/do$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAM
ES
t_test@lnx:~/do$
```

Docker: repository

- You can share your docker image
- You can pull an existing image from centralized repository
- Works almost like git
- You can create your docker account for free in 2 minutes: https://cloud.docker.com/

cssmuadm@lnx: ~/do

```
cssmuadm@lnx:~/do$ sudo docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't ha
ve a Docker ID, head over to https://hub.docker.com to create one.
Username: neveditsin
Password:
Login Succeeded
cssmuadm@lnx:~/do$ sudo docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
friendlyhello       latest              31ef9269a205        36 minutes ago      132MB
python              2.7-slim            d0d1b97dd328        2 days ago          120MB
centos              latest              49f7960eb7e4        3 days ago          200MB
cssmuadm@lnx:~/do$ sudo docker tag friendlyhello neveditsin/tutorial:latest
cssmuadm@lnx:~/do$ sudo docker image ls
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
friendlyhello         latest              31ef9269a205        37 minutes ago      132MB
neveditsin/tutorial   latest              31ef9269a205        37 minutes ago      132MB
python                2.7-slim            d0d1b97dd328        2 days ago          120MB
centos                latest              49f7960eb7e4        3 days ago          200MB
cssmuadm@lnx:~/do$ sudo docker push neveditsin/tutorial:latest
The push refers to repository [docker.io/neveditsin/tutorial]
bb66a5460732: Pushed
6d1afcb5260f: Pushed
27e0ff1bf791: Pushed
20f93bdcee9c: Layer already exists
21b24882d499: Layer already exists
db9dabc5cfee: Layer already exists
d626a8ad97a1: Layer already exists
latest: digest: sha256:ee64adfc0523543ea46350d9aea7aaa402da6fef6a286334d24b3c8813878178 size: 17
87
cssmuadm@lnx:~/do$
```

Docker: interactive use: just like a VM!

@5c5d86f2500d:/

```
t_test@lnx:~$ docker pull centos
Using default tag: latest
latest: Pulling from library/centos
a02a4930cb5d: Pull complete
Digest: sha256:184e5f35598e333bfa7de10d8fb1cebb5ee4df5bc0f970bf2b1e7c7345136426
Status: Downloaded newer image for centos:latest
t_test@lnx:~$ docker run -it centos bash
[root@5c5d86f2500d /]# cd ~
[root@5c5d86f2500d ~]# ls
[root@5c5d86f2500d /]# uname -a
Linux 5c5d86f2500d 4.4.0-141-generic #167-Ubuntu SMP Wed Dec 5 10:40:15 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
[root@5c5d86f2500d /]#
```

- -i means interactive
- -t means allocate pseudo-tty

Docker: commits

@350969c379fa:/home

```
t_test@lnx:~$ docker run -it centos bash
[root@972d02aa6fb8 /]# cd /home
[root@972d02aa6fb8 home]# ls
[root@972d02aa6fb8 home]# mkdir test
[root@972d02aa6fb8 home]# exit
exit
t_test@lnx:~$ docker commit 972d02aa6fb8 centos/test
sha256:adbd54aee5f490ea69a23e1fd71c5d66e50eb8cc38d46ab23f9770d5b1139543
t_test@lnx:~$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos/test         latest              adbd54aee5f4        5 seconds ago       202MB
friendlyhello       latest              638fec51c24f        About an hour ago   131MB
python              2.7-slim            413ee88c678e        5 days ago          120MB
centos              latest              1e1148e4cc2c        2 months ago        202MB
t_test@lnx:~$ docker stop 972d02aa6fb8
972d02aa6fb8
t_test@lnx:~$ docker rm 972d02aa6fb8
972d02aa6fb8
t_test@lnx:~$ docker run -it centos/test
[root@350969c379fa /]# cd /home
[root@350969c379fa home]# lks
bash: lks: command not found
[root@350969c379fa home]# ls
test
[root@350969c379fa home]#
```

Docker: remove image

t_test@lnx: ~

```
t_test@lnx:~$ docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos/test         latest              adbd54aee5f4        3 minutes ago       202MB
friendlyhello       latest              638fec51c24f        About an hour ago   131MB
python              2.7-slim            413ee88c678e        5 days ago          120MB
centos              latest              1e1148e4cc2c        2 months ago        202MB
t_test@lnx:~$ docker image rm centos/test
Error response from daemon: conflict: unable to remove repository reference "centos/test" (must force) - container 350969c37
9fa is using its referenced image adbd54aee5f4
t_test@lnx:~$ docker container rm 350969c379fa
350969c379fa
t_test@lnx:~$ docker image rm centos/test
Untagged: centos/test:latest
Deleted: sha256:adbd54aee5f490ea69a23e1fd71c5d66e50eb8cc38d46ab23f9770d5b1139543
Deleted: sha256:1c51005e3be4151e469dd952080c98f10586ab8233eb7874e4ba9bdf8ec7caa5
t_test@lnx:~$
```

Docker: basic commands list

```
## List Docker CLI commands
docker
docker container --help

## Display Docker version and info
docker --version
docker version
docker info

## Execute Docker image
docker run hello-world

## List Docker images
docker image ls

## List Docker containers (running, all, all in quiet mode)
docker container ls
docker container ls --all
docker container ls -aq
```

Docker: basic commands list (cont-d)

```
## Create image using Dockerfile in the current directory
docker build -t friendlyhello .
## Run "friendlyname" mapping port 4000 to 80
docker run -p 4000:80 friendlyhello
## Same thing, but in detached mode
docker run -d -p 4000:80 friendlyhello
## List all running containers
docker container ls
## List all containers, even those not running
docker container ls -a
## Gracefully stop the specified container
docker container stop <hash>
## Force shutdown of the specified container
docker container kill <hash>
## Remove specified container from this machine
docker container rm <hash>
## Remove all containers
docker container rm $(docker container ls -a -q)
## Rename a container
docker rename oldname newname
## you can use container id instead of oldname
```

Docker: basic commands list (cont-d)

```
## List all images on this machine
docker image ls -a
## Remove specified image from this machine
docker image rm <image id>
## Remove all images from this machine
docker image rm $(docker image ls -a -q)
```

```
## Log in this CLI session using your Docker credentials
docker login
## Tag <image> for upload to registry
docker tag <image> username/repository:tag
## Upload tagged image to registry
docker push username/repository:tag
## Run image from a registry
docker run username/repository:tag
```

Exercise

- Create a centos container and run it in **interactive mode**
- Create a folder with your username in **/home** directory of the container
- Exit the container's shell
- Find your container's name
- Remove your container

Docker networking

- **bridge**: The default network driver. Bridge networks are usually used when your applications run in standalone containers that need to communicate.
- **host**: For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly. Only available for **swarm services** on Docker 17.06 and higher.
- **overlay**: Overlay networks connect multiple Docker daemons together and enable **swarm services** to communicate with each other.
- **macvlan**: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network.The best choice when dealing with legacy applications that expect to be directly connected to the physical network, rather than routed through the Docker host's network stack.
- **none**: disable all networking. [1]

Docker networking: bridge

t_test@lnx: ~

```
t_test@lnx:~$ docker run -dit --name t_test_centos1 centos bash
6ff7612a463b32146e33df07fd8b248163e5a99d510fdaaef9c0db61073d8e8e
t_test@lnx:~$ docker run -dit --name t_test_centos2 centos bash
a8552357d8d361b55223c8b5ee38d276614642c5f989f69f5a7755d3454107b5
t_test@lnx:~$ docker container ls
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAME
S
a8552357d8d3        centos              "bash"              19 seconds ago      Up 18 seconds                           t_te
st_centos2
6ff7612a463b        centos              "bash"              24 seconds ago      Up 23 seconds                           t_te
st_centos1
t_test@lnx:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAME
S
a8552357d8d3        centos              "bash"              28 seconds ago      Up 26 seconds                           t_te
st_centos2
6ff7612a463b        centos              "bash"              33 seconds ago      Up 31 seconds                           t_te
st_centos1
t_test@lnx:~$
```

Run 2 Centos Linux containers in detached interactive mode with bash shell

Docker networking: bridge (cont-d)

t_test@lnx: ~

```
t_test@lnx:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
682700c81569        bridge              bridge              local
91322875b1da        host                host                local
631b2c7ac4b0        none                null                local
t_test@lnx:~$
```

List available networks

Docker networking: bridge (cont-d)

docker network inspect NET
Show information about network (including connected containers)

t_test@lnx: ~

```
t_test@lnx:~$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "682700c8156931f20falea508226f67d87ad94ed10bb1b570d599d69234eac4a",
        "Created": "2019-02-11T15:21:09.247500548Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "6ff7612a463b32146e33df07fd8b248163e5a99d510fdaaef9c0db61073d8e8e": {
                "Name": "t_test_centos1",
                "EndpointID": "86d14328334a041a0a3b02ca394ff6a336aa9f3ca9d93b27a7d198359620f41e"
,
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            },
            "a8552357d8d361b55223c8b5ee38d276614642c5f989f69f5a7755d3454107b5": {
                "Name": "t_test_centos2",
                "EndpointID": "b3845cd9daad60bce2a6f64be196d1c4f8dc7691eaef46494b365923404760ef"
,
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            }
```

Docker networking: bridge (cont-d)


```
        "ConfigOnly": false,
        "Containers": {
            "6ff7612a463b32146e33df07fd8b248163e5a99d510fdaaef9c0db61073d8e8e": {
                "Name": "t_test_centos1",
                "EndpointID": "86d14328334a041a0a3b02ca394ff6a336aa9f3ca9d93b27a7d198359620f41e"
,
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            },
            "a8552357d8d361b55223c8b5ee38d276614642c5f989f69f5a7755d3454107b5": {
                "Name": "t_test_centos2",
                "EndpointID": "b3845cd9daad60bce2a6f64be196d1c4f8dc7691eaef46494b365923404760ef"
,
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            }
```

Docker networking: bridge (cont-d)

t_test@lnx: ~

```
t_test@lnx:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
a8552357d8d3        centos              "bash"              15 minutes ago      Up 15 minutes
6ff7612a463b        centos              "bash"              15 minutes ago      Up About a minute
t_test@lnx:~$ ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.162 ms
64 bytes from 172.17.0.2: icmp_seq=2 ttl=64 time=0.072 ms
^C
--- 172.17.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.072/0.117/0.162/0.045 ms
t_test@lnx:~$ ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.230 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.095 ms
^C
--- 172.17.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.095/0.162/0.230/0.068 ms
t_test@lnx:~$ ifconfig | grep 172
          inet addr:172.17.0.1  Bcast:172.17.255.255  Mask:255.255.0.0
t_test@lnx:~$
```

Try to ping the running containers

Note that host has IP address in the network as well

Docker networking: user-defined bridge net

t_test@lnx: ~

```
t_test@lnx:~$ docker  network create --driver bridge t_test_net
33d42547be61e29818e668009794dd32da1d0f578531ab191a08967899af3e99
t_test@lnx:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
682700c81569        bridge              bridge              local
91322875b1da        host                host                local
631b2c7ac4b0        none                null                local
33d42547be61        t_test_net          bridge              local
t_test@lnx:~$
```

Create a bridge network
“t_test_net”

Docker networking: user-defined bridge net

t_test@lnx: ~

```
t_test@lnx:~$ docker network inspect t_test_net
[
    {
        "Name": "t_test_net",
        "Id": "33d42547be61e29818e668009794dd32da1d0f57853lab191a08967899af3e99",
        "Created": "2019-02-11T18:37:45.79525297Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
t_test@lnx:~$
```

Connect user-defined bridge net

Connect our centos containers to the t_test_net

t_test@lnx: ~

```
t_test@lnx:~$ docker network connect t_test_net
CENT1                t_test_centos1     t_test_centos2     zealous_rosalind
t_test@lnx:~$ docker network connect t_test_net t_test_centos1
t_test@lnx:~$ docker network connect t_test_net t_test_centos2
```

Connect user-defined bridge net

t_test@lnx: ~

```
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "6ff7612a463b32146e33df07fd8b248163e5a99d510fdaaef9c0db61073d8e8e": {
                "Name": "t_test_centos1",
                "EndpointID": "9e81863322b9ac9778cbe3d638957d1550774863dbe177af443449f5593953a6",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            },
            "a8552357d8d361b55223c8b5ee38d276614642c5f989f69f5a7755d3454107b5": {
                "Name": "t_test_centos2",
                "EndpointID": "51a51cbd96c8f388d79fc24ab8880ded4ce86f89bd451393873835df939693d6",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
t_test@lnx:~$
```

Connect user-defined bridge net (cont-d)

t_test@lnx: ~

```
t_test@lnx:~$ ping 172.18.0.2
PING 172.18.0.2 (172.18.0.2) 56(84) bytes of data.
64 bytes from 172.18.0.2: icmp_seq=1 ttl=64 time=0.559 ms
64 bytes from 172.18.0.2: icmp_seq=2 ttl=64 time=0.140 ms
^C
--- 172.18.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.140/0.349/0.559/0.210 ms
t_test@lnx:~$ ping 172.18.0.3
PING 172.18.0.3 (172.18.0.3) 56(84) bytes of data.
64 bytes from 172.18.0.3: icmp_seq=1 ttl=64 time=0.479 ms
64 bytes from 172.18.0.3: icmp_seq=2 ttl=64 time=0.134 ms
^C
--- 172.18.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.134/0.306/0.479/0.173 ms
t_test@lnx:~$ ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.340 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.073 ms
^C
--- 172.17.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.073/0.206/0.340/0.134 ms
t_test@lnx:~$ ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2) 56(84) bytes of data.
64 bytes from 172.17.0.2: icmp_seq=1 ttl=64 time=0.228 ms
^C
--- 172.17.0.2 ping statistics ---
```

Now centos containers have 2 IP addresses each: from default bridge network and from the user-defined one

Connect user-defined bridge net (cont-d)

t_test@lnx: ~

```
t_test@lnx:~$ docker network disconnect bridge
6ff7612a463b32146e33df07fd8b248163e5a99d510fdaaef9c0db61073d8e8e
a8552357d8d361b55223c8b5ee38d276614642c5f989f69f5a7755d3454107b5
t_test_centos1
t_test_centos2
t_test@lnx:~$ docker network disconnect bridge t_test_centos2
t_test@lnx:~$ ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
From 172.17.0.1 icmp_seq=9 Destination Host Unreachable
From 172.17.0.1 icmp_seq=10 Destination Host Unreachable
From 172.17.0.1 icmp_seq=11 Destination Host Unreachable
^C
--- 172.17.0.3 ping statistics ---
12 packets transmitted, 0 received, +3 errors, 100% packet loss, time 11079ms
pipe 3
t_test@lnx:~$
```

Let's disconnect the bridge network from centos2:
Now we can't ping it

Docker networking review

- Containers by default are created with default **bridge** network. They can access internet and other containers **which are on the same network** by their IP addresses. You can access containers by IP address from host as well

- User-defined bridge networks are usually used to connect containers running on the **same Docker host**. This is recommended for standalone containers running in **production**. Same as default bridge network, the user-defined bridge networks can access **internet** and other containers **which are on the same network** by their IP addresses. You can access containers by IP address from host as well

- Containers can be connected to multiple networks

Docker networking: basic commands

```
## List all docker networks on this machine
docker network ls
## Remove specified network from this machine
docker network rm <network id>
## Show information about network
docker network inspect <network id>
## Connect running container to the existing network
docker network connect <network id> <container id>
## Disconnect running container from the existing network
docker network disconnect <network id> <container id>
```

Docker networking: clarifying points

- **EXPOSE** command in Dockerfiles servers as a kind of **metadata** for other applications and also tells users which services are running in the container. **It does not restrict containers from opening other ports**
- Flag -p **binds** container's port to some **local port on the host**: command `docker run -p 4000:80 friendlyhello` maps port 80 of container to port 4000 of the host machine
- Applications can open any ports inside your container (like MySQL server's port (3306), HTTP (80), HTTPS (443), and even SSH (22)). You can either access these apps using IP address of the container (e.g., curl 172.17.0.2:80) or BIND this port to your host and then use host's IP address to access the apps (e.g., curl lnx.cs.smu.ca:4000)

Exercise

- Create two centos containers: names should be USER_centos1 and USER_centos2 where USER is your username
- Attach to centos1 (docker attach CONTAINER_NAME), install **nc** with command **yum install nc** (answer Y)
- Listen on port 1234 with netcat redirected to some file (e.g., **nc 0.0.0.0 1234 > test**)
- Detach from the container using **Ctrl+P Ctrl+Q** sequence
- Send a message from your Linux account to the centos1 machine using nc (e.g., **echo "Hi there" | nc IP 1234** where IP is IP address of your docker container)
- Attach to the container again and check contents of the file where you redirected your netcat

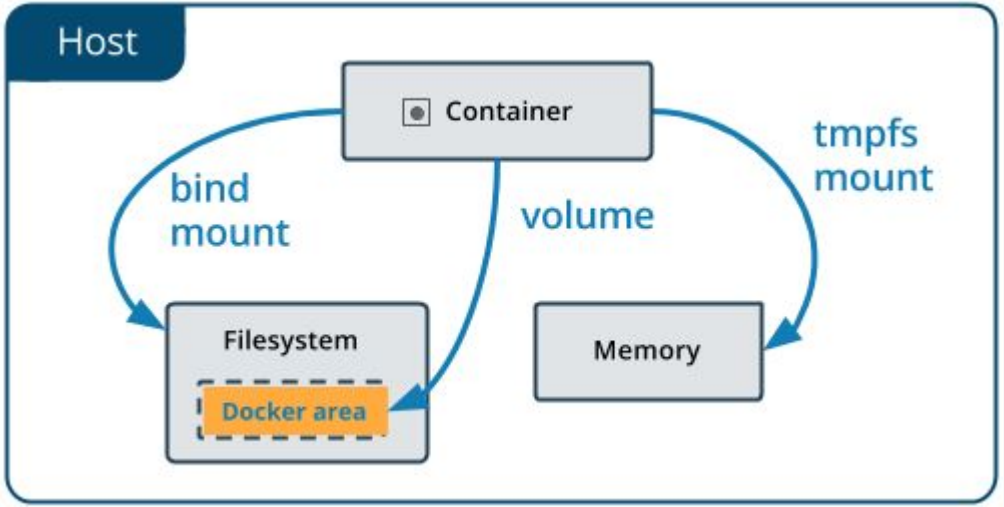
Docker: persistence

By default all files created inside a container are stored on a writable container layer. It means that:

- The data **doesn't persist** when that container is removed, and it can be difficult to get the data out of the container if another process needs it.
- A container's writable layer is tightly coupled to the host machine where the container is running. You can't easily move the data somewhere else.
- Writing into a container's writable layer requires a storage driver to manage the filesystem. The storage driver provides a union filesystem, using the Linux kernel. This extra abstraction reduces performance as compared to using data volumes, which write directly to the host filesystem.

Docker has two options for containers to store files in the host machine, so that the files are persisted even after the container is removed: volumes, and bind mounts [1]

Docker: persistence: volumes



1

Docker: persistence: volumes (cont-d)

t_test@lnx: ~

```
t_test@lnx:~$ docker volume create t_test_voll
t_test_voll
t_test@lnx:~$ docker volume ls
DRIVER              VOLUME NAME
local               t_test_voll
t_test@lnx:~$ docker volume inspect t_test_voll
[
    {
        "CreatedAt": "2019-02-11T19:49:18Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/t_test_voll/_data",
        "Name": "t_test_voll",
        "Options": {},
        "Scope": "local"
    }
]
```

Docker: persistence: volumes (cont-d)

- Volume is mounted as empty directory
- If target directory does not exist in container, Docker creates it
- Data written to it is accessible from host and can be accessible from other containers
- After removing container, data is still there

t_test@lnx: ~

```
t_test@lnx:~$ docker run -dit --name cent5 --mount source=t_test_vol1,target=/home/testvol centos bash
4771dd3ac3c3bab8ffab9917d29c1b0d95b561e4c08d1fdb18bb6885 1e0f762e
t_test@lnx:~$ docker attach cent5
[root@4771dd3ac3c3 /]# cd /home
[root@4771dd3ac3c3 home]# ls
testvol
[root@4771dd3ac3c3 home]# cd testvol/
[root@4771dd3ac3c3 testvol]# ls /bin > binlist
[root@4771dd3ac3c3 testvol]# exit
exit
t_test@lnx:~$ ls -l /var/lib/docker/volumes/t_test_vol1/_data
total 4
-rw-r--r-- 1 root root 3610 Feb 11 20:34 binlist
t_test@lnx:~$ tail /var/lib/docker/volumes/t_test_vol1/_data/binlist
zcat
zcmp
zdiff
zegrep
zfgrep
zforce
zgrep
zless
zmore
znew
t_test@lnx:~$
```

Docker: persistence: volumes (cont-d)

If the same volume is mounted again (it's not empty), then it's not populated with data from container's directory. Instead, the contents of volume is mounted to target directory

cssmuadm@lnx: ~/do

```
cssmuadm@lnx:~/do$ sudo docker run -dit --name cent --mount source=volume1,target=/bin centos
 bash
5786d0dc7327945f5c84e8473f70fb771795428b8ba1222e24a685e6026bd6b2
docker: Error response from daemon: OCI runtime create failed: container_linux.go:348: starti
ng container process caused "exec: \"bash\": executable file not found in $PATH": unknown.
cssmuadm@lnx:~/do$ sudo docker container ls -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
       PORTS               NAMES
5786d0dc7327        centos              "bash"              20 seconds ago      Created
                           cent
cssmuadm@lnx:~/do$ sudo docker container stop cent
cent
cssmuadm@lnx:~/do$ sudo docker container rm cent
cent
cssmuadm@lnx:~/do$
```

Docker: persistence: volumes (cont-d)

If source volume does not exist, Docker creates new volume **and populates it with contents of target directory**

t_test@lnx: ~

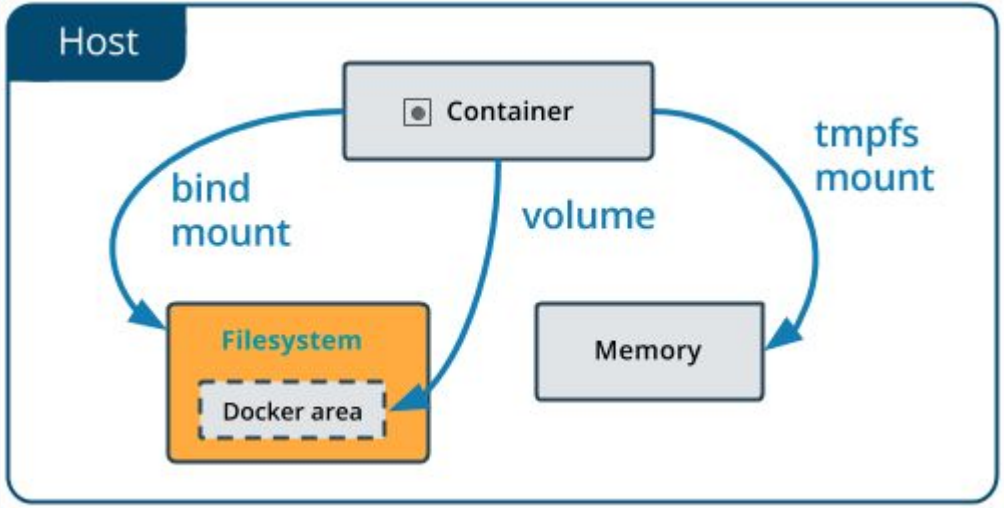
```
t_test@lnx:~$ docker run -dit --name cent6 --mount source=VOLUME2,target=/bin centos bash
5ela30la0d8baa595076a4de160fd7ae9c210f971ccd291d843833156a6e70b3
t_test@lnx:~$ docker volume ls
DRIVER              VOLUME NAME
local               VOLUME2
local               t_test_vol1
t_test@lnx:~$ docker volume inspect VOLUME2
[
    {
        "CreatedAt": "2019-02-11T20:44:02Z",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/VOLUME2/_data",
        "Name": "VOLUME2",
        "Options": null,
        "Scope": "local"
    }
]
t_test@lnx:~$ ls /var/lib/docker/volumes/VOLUME2/_data
[                                   idn                 setup-nsssysinit.sh
addr2line                           igawk               sg
alias                               info                sh
ar                                  infocmp             sha1sum
arch                                infokey             sha224sum
as                                  infotocap           sha256sum
awk                                 install             sha384sum
base64                              ionice              sha512sum
basename                            ipcmk               show-changed-rco
bash                                ipcrm               show-installed
bashbug                             ipcs                shred
```

Docker: persistence: volumes (cont-d)

```
## List all docker volumes on this machine
docker volume ls
## Remove specified volume from this machine
docker volume rm <volume id>
## Show information about volume
docker volume inspect <volume id>
```

```
## Start container with mounted volume
docker run --mount source=volume,target=/path/to/something <image id>
```

Docker: persistence: bind mounts



[1]

Docker: persistence: bind mounts (cont-d)

Host directory **/home/t_test is now** mounted as container's **/home/test_acct**

```
t_test@lnx: ~
t_test@lnx:~$ docker run -dit --name cent1 -v /home/t_test:/home/test_acct centos bash
a27f099cc0a21c47905acb4c4382f234b40f90f57f120d18d76c1bf2c722e7f7
t_test@lnx:~$ docker attach cent1
[root@a27f099cc0a2 /]# cd /home
[root@a27f099cc0a2 home]# ls
test_acct
[root@a27f099cc0a2 home]# cd test_acct/
[root@a27f099cc0a2 test_acct]# ls
binlist.txt  bz  do  do1  do2  nobz  nohup.out  scrip.sh  tmp
[root@a27f099cc0a2 test_acct]# cp /bin/
Display all 454 possibilities? (y or n)
[root@a27f099cc0a2 test_acct]# touch TEST
[root@a27f099cc0a2 test_acct]# exit
exit
t_test@lnx:~$ ls
binlist.txt  bz  do  do1  do2  nobz  nohup.out  scrip.sh  TEST  tmp
t_test@lnx:~$ ls -l
total 36
-rw-rw-r-- 1 t_test t_test 1432 Jan 31 14:45 binlist.txt
-rw-rw-r-- 1 t_test t_test  339 Feb  2 19:26 bz
drwxrwxr-x 2 t_test t_test 4096 Feb 11 16:31 do
drwxrwxr-x 2 t_test t_test 4096 Feb 11 16:45 do1
drwxrwxr-x 2 t_test t_test 4096 Feb 11 16:52 do2
-rw-rw-r-- 1 t_test t_test 1093 Feb  2 19:26 nobz
-rw------- 1 t_test t_test  126 Jan 31 14:14 nohup.out
-rwxrw-r-- 1 t_test t_test  210 Feb  2 19:21 scrip.sh
-rw-r--r-- 1 root   root      0 Feb 11 20:52 TEST
```

Docker: persistence: bind mounts vs volumes

Volumes are the preferred way to persist data in Docker containers and services. Some use cases for volumes include:

- Sharing data among multiple running containers. If you don't explicitly create it, a volume is created the first time it is mounted into a container. When that container stops or is removed, the volume still exists. Multiple containers can mount the same volume simultaneously, either read-write or read-only. Volumes are only removed when you explicitly remove them.
- When the Docker host is not guaranteed to have a given directory or file structure. Volumes help you decouple the configuration of the Docker host from the container runtime.
- When you want to store your container's data on a remote host or a cloud provider, rather than locally [l]

Docker: persistence: bind mounts vs volumes

Bind mounts are appropriate for the following types of use case:

- Sharing configuration files from the host machine to containers. This is how Docker provides DNS resolution to containers by default, by mounting **/etc/resolv.conf** from the host machine into each container.
- Sharing source code or build artifacts between a development environment on the Docker host and a container. For instance, you may mount a Maven target/ directory into a container, and each time you build the Maven project on the Docker host, the container gets access to the rebuilt artifacts.
- When the file or directory structure of the Docker host is guaranteed to be consistent with the bind mounts the containers require[l]

Exercise

Dump **/var** directory from your Centos container to subdirectory **‘cvar’** of your home directory at lnx.cs.smu.ca

Docker: uncovered topics

- Services
- Swarms
- Stacks
- Related networking topics

These features are useful for load-balancing, distributed computing and production environment. You should be able to easily learn how they work (in case if you need them) using official Docker tutorial