



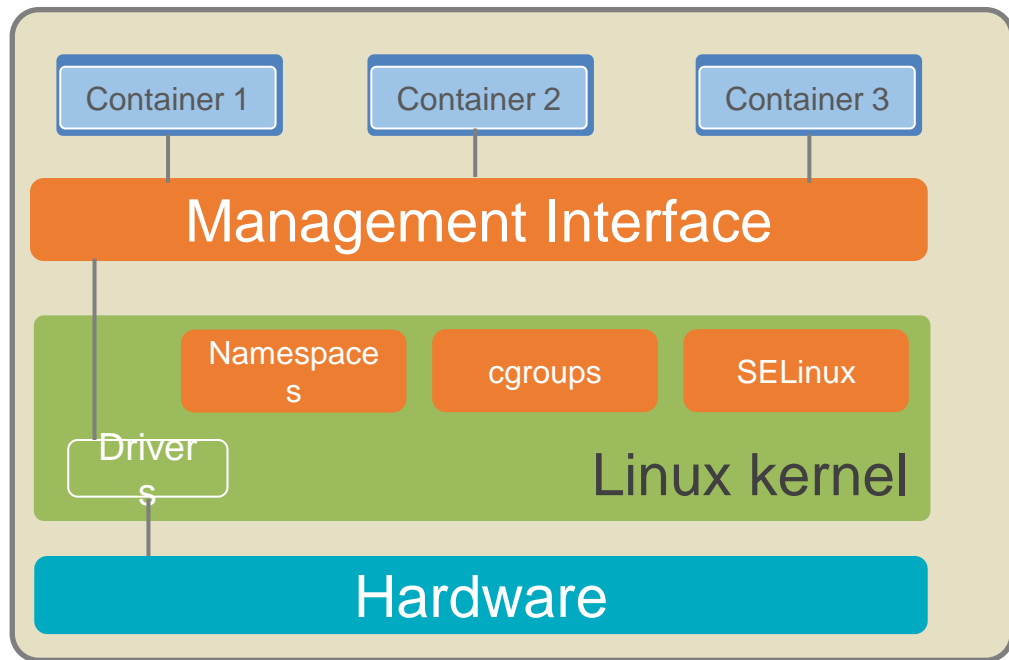
Docker – Containers

Containers

- Containers are operating system level virtualization
 - Allows for multiple isolated user space instances called containers
 - They share a single kernel
 - Can be added or removed at any time
- Containers consist of a self contained Linux file system
 - Can be from any Linux distribution which is compatible with the host kernel
 - Usually contain a single application such as a server
- Operating system level virtualization is lightweight
 - Is often used in Cloud Computing

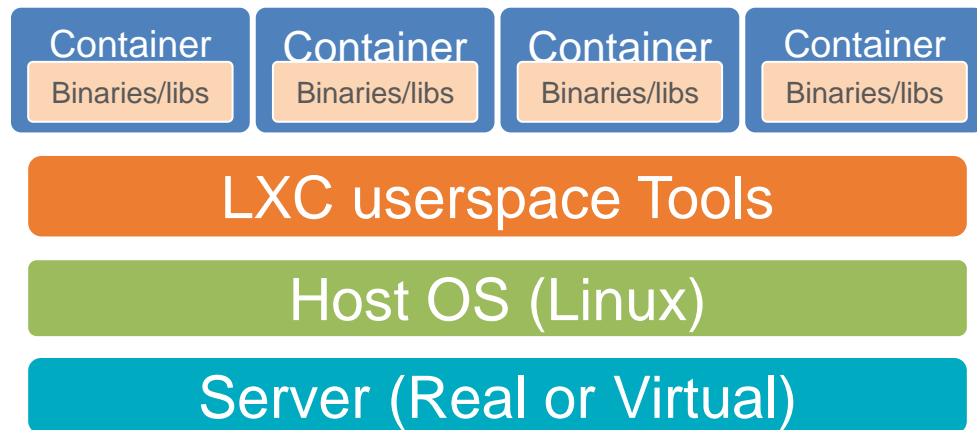
Container Implementation

- Operating system level virtualization uses a set of tools
 - A virtualization subsystem
 - A cgroup hierarchy for each container
 - The container is mounted into the filesystem
- A program inside the container is executed
 - Using chroot to restrict it to the container file system
 - The cgroup constrains use of resources and isolates the container from the rest of the system



Operating System Level Virtualization

- Operating system level virtualization is where an operating system kernel can support multiple isolated user space instances
 - Instances are called containers or jails
 - There is little overhead as the kernel implements the containers
- There are numerous implementations
 - chroot has been available in UNIX since 1982
 - FreeBSD jail
 - Linux Containers (LXC) command line tools using Linux cgroups
 - LXD a container hypervisor built on LXC
 - Docker is a suite of tools for creating and managing containers



- Docker is a very light-weight software container and containerization platform
- Docker containers provide a way to run software in isolation
- What does a Docker Container contain and provide?
 - Initially – only a base Linux operating system
 - A boundary or a “jail” to contain running software
 - Like a good jail, there are no unauthorized entries or exits
 - A Docker Image is the foundation for any particular Docker container
- What is outside of Docker?
 - Other Docker containers
 - The operating system, kernel
 - Any other operating software



Docker was originally a Linux application



It uses the kernel container functionality



It requires a 64 bit installation using a kernel version 3.10 or later



Docker runs on many popular Linux distributions



It is available as RPM, APT, or binary versions

Docker for OS X




Docker runs natively on OS X



DMG install application running in user space



Is built on the xhyve hypervisor



Requires a 2010 or newer Mac with Intel MMU and EPT support



Requires OS X 10.10.3 Yosemite or newer



Requires at least 4GB of RAM

Docker for Windows



Docker runs natively on Windows



Requires later versions of Windows 10 Pro or Enterprise



Docker requires the Hyper-V package



This is Microsoft's hypervisor for Windows



It virtualizes the Docker environment and Linux kernel space



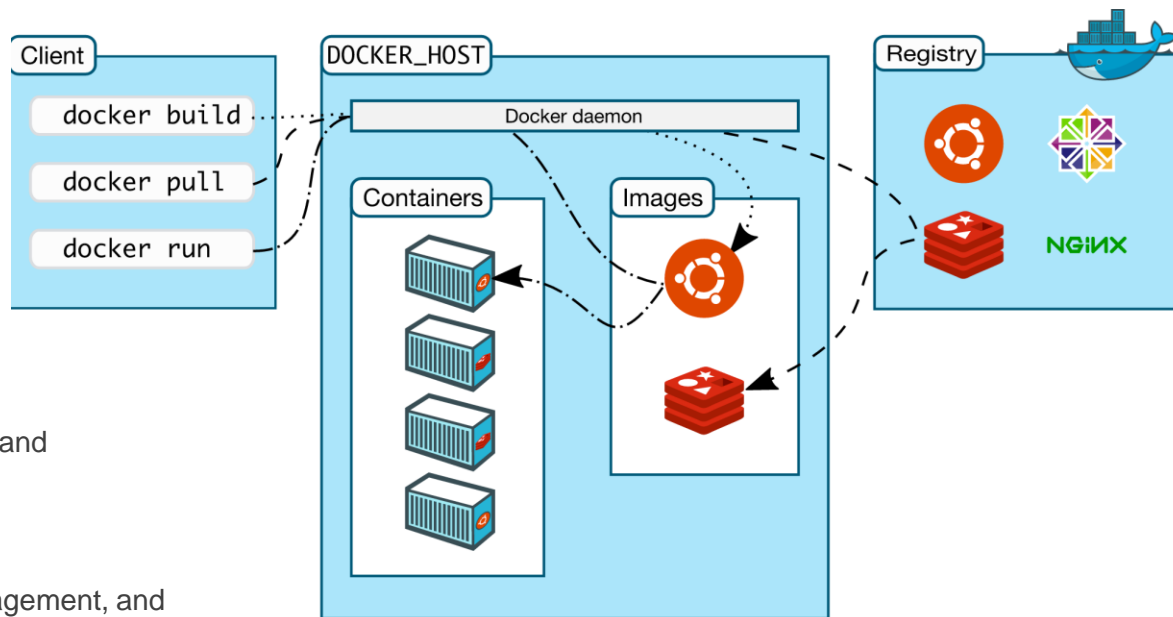
Docker can't run alongside VirtualBox VMs

Exercise 1 : Install Docker on Linux & Windows

- Use Installation document to install docker on Linux Ubuntu and Windows 10

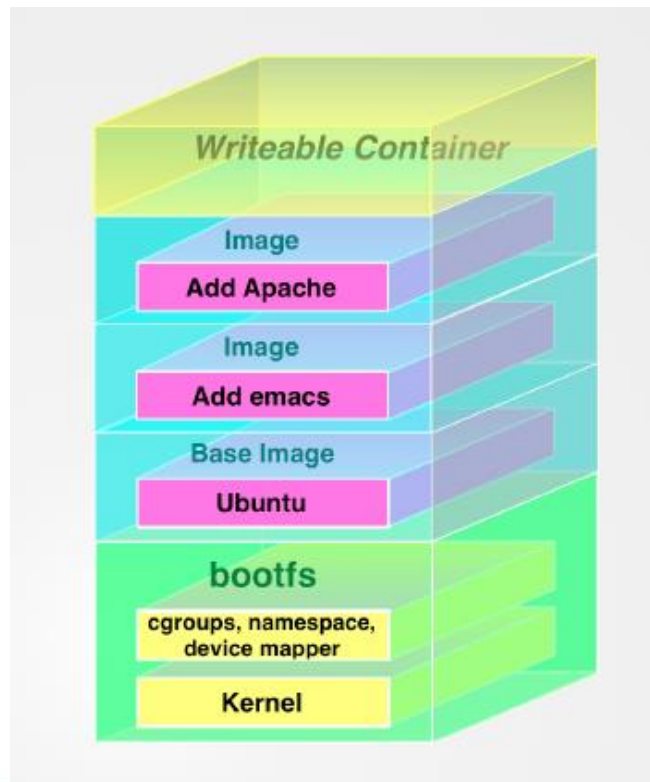
Docker Architecture

- Docker uses a client-server architecture
- Client
 - Is the primary user interface which communicates using a REST API
 - Over HTTP
 - Over local Unix socket
- Server
 - Is the Docker daemon
 - Responsible for building, running, and distributing containers
- Registry
 - Responsible for the storage, management, and delivery of Docker Images
 - Docker Hub
 - Private
 - Other vendors



Docker Images & Containers

- Docker images are read-only templates
 - Foundation is a simplified version of the Linux operating system
 - Changes to foundation, such as application installations added to the Image
 - Images are the templates or build commands for Docker
- Docker containers are running environments
 - Has OS, environment, program, network, etc.
 - Runs (probably one) application
 - All required software contained in image
 - Can have boot-up configuration
 - They can be run, started, stopped, and deleted



Docker Images

- Docker images are built in layers
 - Each layer is a file system
 - The layers are combined in a union file system to make a single image
 - Images are the build component of Docker
- Images start from a base image
 - Foundation is usually a specifically prepared Linux operating system
 - Custom base images can also be created
 - Docker Image are then built by adding layers:
 - Interactively
 - Defined in a directive file called a “Docker File”

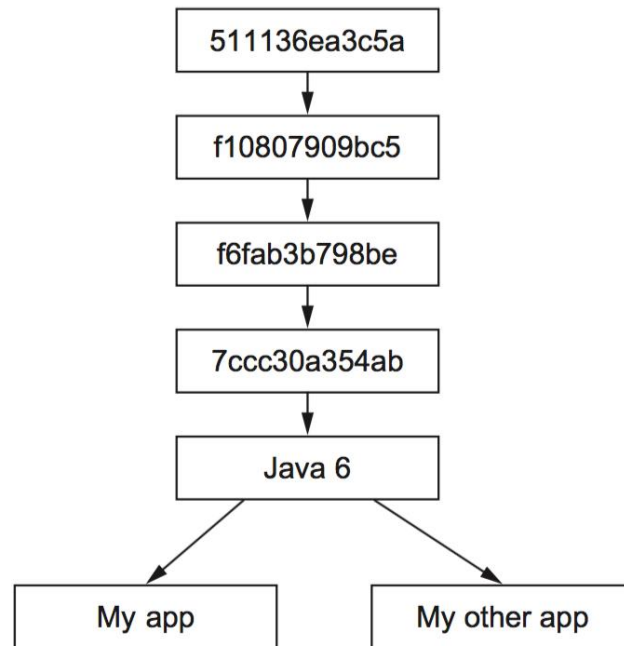


Figure 3.7 The full lineage of the two Docker images used in section 3.3.1

Running Containers

- The docker `run` command starts a container based on a named Docker Image
 - Docker first looks for a local copy of the image
 - If it does not exist it is pulled from a Docker Registry
 - The default Registry is the Docker Hub Registry
 - A new container is created using the file system from the image
 - A read-write layer is added to the top of the file system
 - A network interface is created and an IP address is assigned from a pool
 - Standard input, output, and error streams are connected
 - A specified application is executed
- Docker container appears as a child of the daemon process, `ms` are connected

Pulling & Running Containers

- A Docker Image must be located on the local computer
 - It may have been created locally
 - It may have been pulled from a Registry
 - It may be missing
- The `pull` command insures that the specified image is on the local computer
 - It will transfer all constituent layers of the image as separate transfers
- The `run` command creates and initiates a container based on the image
 - The example runs the latest CentOS image
 - It runs the command `command`

```
docker pull centos:latest
```

```
docker run centos:latest whoami  
docker run centos:latest pwd  
docker run centos:latest date
```

To use a container interactively requires switches to the run command

- -i or --interactive keeps STDIN open
- -t or --tty allocates a pseudo TTY

```
docker run -i -t centos:latest /bin/bash
```

Listing Running Containers

- Running containers can be listed using the ps command
 - Note that the container has been given a name
- The stats command shows running container resource usage, use ^C to exit

```
docker ps  
docker stats
```


Naming Containers

Containers can be explicitly named using the `--name` switch

- By default, Docker makes up a comical name such as `hungry_lumiere`
- Most commands will accept either the name or the ID of the container
- Note: The `ps` command accepts a switch `-a` for all containers

```
docker run -it --name centosC1 centos:latest /bin/bash
docker stop centos
docker rm centos
```

Attaching to Running Container

- Attaching to a container attaches to the contained process's STDIN, STDOUT, and STDERR
 - You can attach with either the container ID or its name
 - Several command prompts can attach to the same container process
 - All tty sessions see the same input and outputs
 - The container ID is obtained using `ps`

On first terminal:

You can detach from a container and leave it running using `^p ^q`

```
docker run -it --name centosC1 centos:latest /bin/bash  
> date
```

On second terminal:

```
docker attach centosC1  
> date
```

Containers can be stopped using the stop command

- It has an optional `-t` or `--time` parameter which defaults to 10 seconds
- The main process, `PID=1`, is sent a `SIGTERM`
- After the timeout interval, it is sent a `SIGKILL`

```
docker stop centos
```

Pausing Containers

- Docker containers may be paused
 - Use the `pause` command
 - All container processes are suspended as a group
 - It does not use the SIGSTOP, SIGCONT mechanism as processes can see the signals
 - It uses the cgroups freezer mechanism where all processes in a cgroup and its children are suspended without the processes being aware
- Process can be resumes using the unpause command

On first terminal:

```
docker run -it --name centosC1 centos:latest /bin/bash
```

On second terminal:

```
docker pause centosC1  
> date
```

On second terminal:

```
docker unpause centosC1
```

Starting Containers

- Once a container is stopped, it is still available
 - The `ps -a` command shows all containers
- A stopped container can be started with the `start` command
 - The `-i` or `--interactive` switch connects STDIN
 - The `-a` or `attach` switch attaches STDIN and STDERR

```
docker ps -a  
docker start -ai centosC1      OR  
Docker restart centosC1
```

Docker containers can be removed with the `rm` command

- The `-f` or `--force` switch causes running containers to be force stopped for removal
- The `ps -aq` command shows all containers IDs to enable all to be removed

```
docker rm centosC1  
docker rm $(docker ps -aq)
```

- Containers can be run in the background
 - Use the run command with the `-d` or `--detach` switch
- Signals can be sent using the kill command
 - The default signal is SIGTERM

```
docker run -d --name webserver nginx  
docker kill webserver
```

- Docker has a number of base images on Docker Hub
 - Including many versions of Linux distributions
 - The image `centos:latest` will be used as a base
- Application images are created by adding layers to a base image
 - It can be done manually from a container running a shell
 - It can be automated using Docker's build process
- The Docker search facility can be used to search for images on Docker Hub

Listing and Removing Images

- Images can be listed using the images command
- Images can be deleted using the rmi command

```
docker images  
docker rmi centos-git
```

The CentOS base image is cut down

- It doesn't have the `which` or `ifconfig` commands
- We can install these from the command line using `yum`

```
docker run -it --name centosC1 centos:latest  
yum install -y which  
yum install -y net-tools  
exit
```

Building Image Interactively

- Any changes made interactively are only in the container
 - The diff command can be used to see which files have been changed
- The changed container can be committed to create an image
- The image can then be run
 - The run --rm option deletes the container on exit

```
docker run -it --name centosC1 centos:latest
yum install -y which
yum install -y net-tools
exit

docker diff centosC1
docker commit centosC1 centos-net
docker rm centosC1
docker run --rm -it --name centosC2 centos-net
```

The images can be inspected to see the layers

- The inspect command returns a JSON array by default
- It can be used on images and containers
- The `-s` or `--size` switch gives the size of a container
- The `-f` or `--format` switch can be used to extract the JSON fields

```
docker inspect centos-net
```

- Docker image creation can be automated
- Create a directory containing all the files required for the build
- Add a file called Dockerfile which defines the build process
- The directory becomes the build context
- Each command in the build file creates a layer of the image
- A new container is created at each stage

- Dockerfile contains build directives
 - FROM defines the starting image
 - MAINTAINER defines the email address of the builder

```
FROM centos:latest  
MAINTAINER phill@totaleclipse.eu
```

- The Dockerfile SHELL command defines the default shell to use
 - It must be specified in JSON form
 - The default for Linux and windows are shown
 - It takes the form SHELL ["executable", "parameters"]

```
SHELL ["/bin/sh", "-c"]
```

```
SHELL ["cmd", "/S", "/C"]
```

- The Dockerfile COPY command copies files into the container
 - The source files or directories must be in the build context
 - The source files can contain UNIX shell wildcards ? * []
 - Destination directories must end in a / and will get created if they don't exist

```
COPY jdk*.rpm /tmp/
```


- The Dockerfile ADD command copies files and remote file URLs into the container
 - The source files or directories must be in the build context or remote URLs
 - The source files can contain UNIX shell wildcards ? * []
 - Destination directories must end in a / and will get created if they don't exist
 - Local source files in tar or compressed tar format get unpacked

```
ADD apache-maven*.tar.gz /opt/
```

- The Dockerfile RUN command executes a Linux command
 - Multiple commands can be separated with a ; - needed for cd
 - Commands shouldn't block for input – commands have a –y switch which answers yes to all questions

```
RUN yum install -y which
```

```
RUN rpm -i /tmp/*.rpm
```

```
RUN cd /opt; ln -s apache-maven* maven
```

The Dockerfile ENV command sets environment variables in the image.

```
ENV JAVA_HOME=/usr/java/latest  
ENV PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/opt/maven/bin
```

EXPOSE

Exposes the port at which the container application will be made available

EXPOSE 22

COMMAND

The Dockerfile CMD command sets the default application.

```
CMD /bin/bash
```

- The build process requires a directory
 - All of the directory contents are transferred to the daemon
 - It must contain a Dockerfile build script
 - Images should be tagged image-name:version
 - A temporary container is created for each command in the build

```
docker build -t centos-java:latest .
```

Exercise 2 : Automate Build of Docker Image

- Use document to build Jtrac image and run a container using it

- A major issue using Docker is getting the correct versions of images
 - Docker uses registries to distribute images
 - Registries can be hosted or you can have a private version
- Considerations about registries:
 - Performance, rollout frequency, and number of images
 - Security issues, including access control and digitally signing images

- A registry is a service used to manage and distribute images
- It is based on a description
- Registries manage Docker images stored in repositories
- Repositories are collections of related images
 - Different versions of an application
 - All images have the name of the repository with a tag name to distinguish between images

- Docker Hub is the default registry
 - It has a root namespace for official images
 - Root images include versions of supported Linux distributions
 - For example the nginx images can be addressed in different ways

```
hub.docker.com/_/nginx:1.9  
nginx:1.9
```

Labels

- Labels are used to uniquely identify images
 - Labels look like URIs
 - Components separated by /
- Label components:
 - Registry FQDN
 - Namespace _ is for Docker Hub, r is for user
 - User or organization name
 - Repository name: tag
 - A tag is either a version number or a descriptive label

```
https://hub.docker.com/r/databliss/netkernel-se/
```

URI = Uniform Resource Identifier

FQDN = Fully Qualified Domain Name

Other Docker Registries

Docker Hub is not the only Docker registry.

- Google Container Registry
 - Part of the Google Cloud Platform, good for access control and security
- Amazon EC2 Container Service
 - Part of Amazon AWS
- Quay
 - Has free and pay for plans
- Private Registry

- First it is necessary to log into the registry
 - You will be prompted for a password
 - The credentials will be stored in ~/.docker/config.json
 - The password is stored as a hash

```
docker login --username=user
```

- Images need to have the same name as the repository
 - Create a container from an image
 - Commit the image to the Docker Hub registry name
 - Alternatively, create a new tag for the image
- Images can then be pushed into the repository
 - It can take a while as all layers are pushed

```
docker create --name java centos-java  
docker commit java phill/question:java-1.0  
docker push phill/question
```

Using Docker Hub Repository

- Image tags can be viewed on the Docker Hub web site
- Images are retrieved
 - By using the docker pull command

```
docker pull phill/question:java-1.0
```

Deleting Images

- Images can be deleted from the command line
- Docker Hub doesn't allow images to be deleted at present
 - Can only delete the entire repository from the Settings menu

```
docker rmi phill/question:java-1.0
```


- Private registries are a good solution for the following cases:
 - Provide a local image cache to speed up image loading
 - Allow teams to share images locally
 - Store images specific to a project lifecycle stage, development, and UAT
 - Guarantee that the registry will be available for as long as required

- The easiest way to create a private registry is to use a prebuilt Docker container
 - Docker Hub has a number of registry images including the official one
- The registry images can be pulled from Docker Hub

```
docker pull registry:2
```

- The registry image can now be run as a container
 - It needs to be run as a daemon container
 - The `-p` option exposes the container's ports as local ports
 - It usually uses port 5000

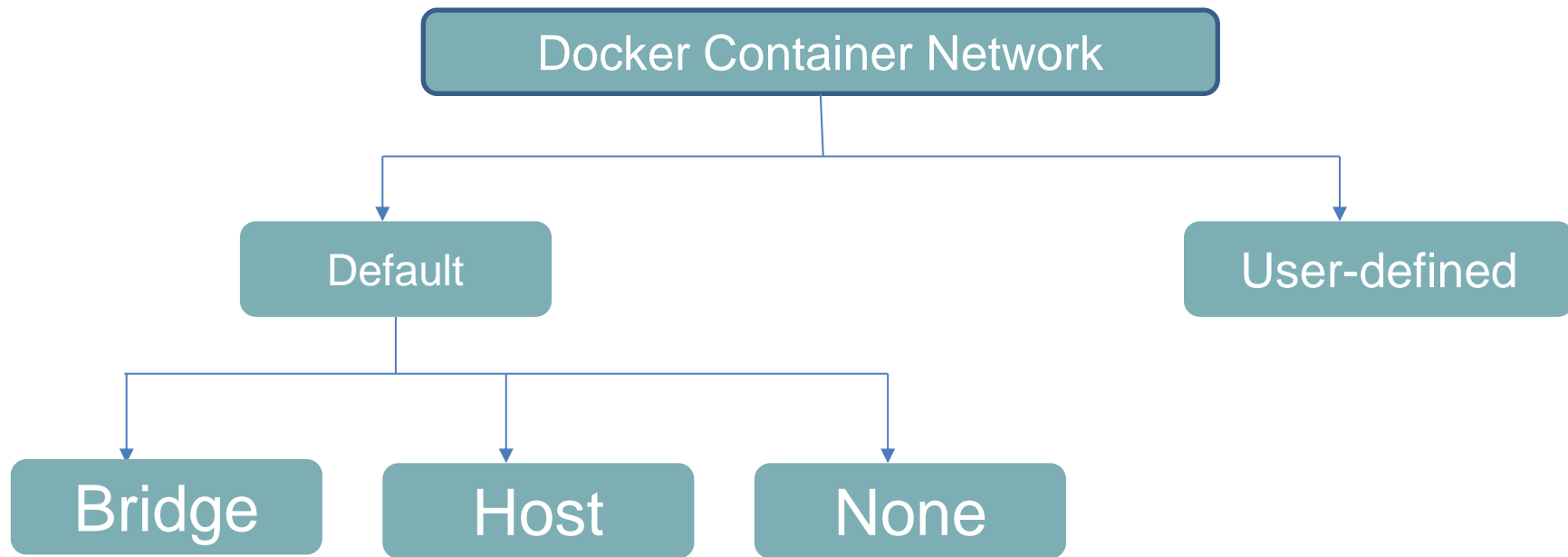
```
docker run -d -p 5000:5000 --name registry registry
```

- The registry is identified by hostname:port
 - For example localhost:5000
 - Images need to be named with the registry prefix
 - For example, localhost:5000/alpine
 - The image can then be pushed into the registry or pulled from it

```
docker pull alpine
docker tag alpine:latest localhost:5000/alpine:latest
docker push localhost:5000/alpine
```

Exercise 3 : Building Private Docker Registry

- Use document to build private docker registry and push images to it



Docker Container Networks - Default

- Docker creates three networks by default, which can't be removed
 - The none network is local to the container – it has localhost
 - The host network gives the container the same network as the host
 - The bridge network is the default
- A docker0 or bridge0 virtual interface is created on the host

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
9d6a9ab487ba	bridge	bridge	local
c7956146a031	host	host	local
115642b21a91	none	null	local

- The bridge network creates a subnet and a subnet mask.

```
$ docker network inspect bridge
[ {
    "Name": "bridge",
    "Id":
"9d6a9ab487ba1d00715bfa60833a9cf5daa564d9a02918424ca3d38e26b2b5f8",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
        "Driver": "default",
        "Options": null,
        "Config": [
            {
                "Subnet": "172.17.0.0/16",
                "Gateway": "172.17.0.1"
            }
        ]
    }
}]
```


- The bridge network assigns MAC and IP addresses to each container

```
$ docker network inspect bridge
...
  "Containers": {
    "eb6dc24ff73fff0da60e98b02aa28e76f92f316aeed73f774ef7b3b0220b5b69": {
      "Name": "centos",
      "EndpointID":
"5b4437f5c54b0923f4558ecc01f9326fafa0fbb4d1f8564131d6dac9bd47e0de",
      "MacAddress": "02:42:ac:11:00:02",
      "IPv4Address": "172.17.0.2/16",
      "IPv6Address": ""
    }
  },
```

- The bridge network supplies a /etc/hosts file for each container

```
127.0.0.1      localhost
::1           localhost ip6-localhost ip6-loopback
fe00::0        ip6-localnet
ff00::0        ip6-mcastprefix
ff02::1        ip6-allnodes
ff02::2        ip6-allrouters
172.17.0.2     eb6dc24ff73f
```

A container attached to a host network has the same network as the host

It has the same network configuration as the host

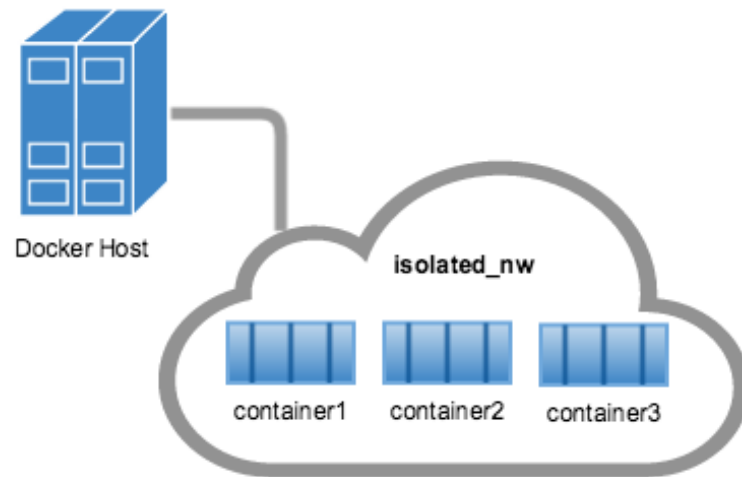
It is not used much any more

Default – None Network

- A container attached to a none network has no network
- It has only a localhost interface
- It can't communicate with other networked containers
- It is not used much any more

Docker Container Network – User Defined Network

- User defined networks can be created
 - Docker provides drivers including bridge
- Containers can only communicate with other containers on the same network
- Multiple networks can be created
- Containers can be connected to multiple networks
 - Can communicate with any container on any connected network



Creating User Defined Network

- New networks can be created
- The default driver is the bridge network
- A new subnet is created unless addresses are specified

```
$ docker network create isolated_bridge
b58db4ec8887a9187151c46850d69b58276a95d780e7465a24aaffb014f6ad8
```

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
9d6a9ab487ba	bridge	bridge	local
c7956146a031	host	host	local
db58db4ec888	isolated_bridge	bridge	local
115642b21a91	none	null	local

Using networks

- Networks can be specified only when a container is run
- A network can be added to an existing container
 - A new network interface is added
- A network can be disconnected from a container
- A user-defined network can be removed

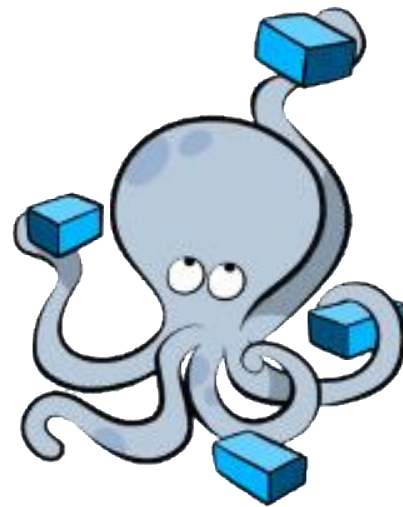
```
docker run -it --network isolated_bridge --name java centos-java
docker network connect isolated_bridge centos
docker network disconnect isolated_bridge centos
docker network rm isolated_bridge
```

Exercise 4 : Docker networking

- Networking between container on docker network

Docker Compose

- Docker Compose is a tool for running multi-container Docker applications
- A configuration file is used to define the services
- All of the services can be run using a single command
- Compose can manage the lifecycle of an application
 - Start, stop, and rebuild services
 - View the status of running services
 - Get the log output of running services
 - Run a command on a service



Installing Compose

- Run this command to download the latest version of Docker Compose:

```
cd ~  
sudo curl -L  
"https://github.com/docker/compose/releases/download/1.22.0/docker-  
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

- Apply executable permissions to the binary:

```
sudo chmod +x /usr/local/bin/docker-compose
```

Compose

- Create a directory for Docker Compose
- Create a YAML file called docker-compose.yml
- The docker-compose command needs to be run from the directory containing the YAML file
- Any number of containers can be specified

```
version: "2"
services:
  sshd:
    build: sshd
    image: centos-sshd:latest
    ports:
      - "2222:22"
```

Compose

- First of all create a directory with name of service containing a Dockerfile under Docker Compose folder
- Docker Compose would use this to build image

```
FROM centos:latest
RUN yum install -y openssh-server
RUN mkdir /var/run/sshd
RUN useradd -c "Student User" -m student
RUN echo "student:student" | chpasswd
RUN ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key -q -N ""
EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

Run Compose

- Docker Compose has an up option which runs all containers
- The `-d` option runs it in the background
- The down option terminates all managed containers

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

```
ssh -p 2222 student@localhost
```

```
docker-compose down
```

Exercise 5 : Docker Compose

- Use docker compose to run containers

- A single command can be executed on a container
- A new container is started from the image
- The `--rm` option removes the container after the command is run
- The `--name` option names the new container
- The `-p` option publishes container ports to the host
- A TTY is allocated unless the `-T` option is given

The background of the slide features a handshake in the center, with a teal-colored overlay on the left side. The background image is a cityscape with a bright light source on the left, creating a lens flare effect. In the upper right corner, there is a statistic that reads "Average 45%".

THANK YOU