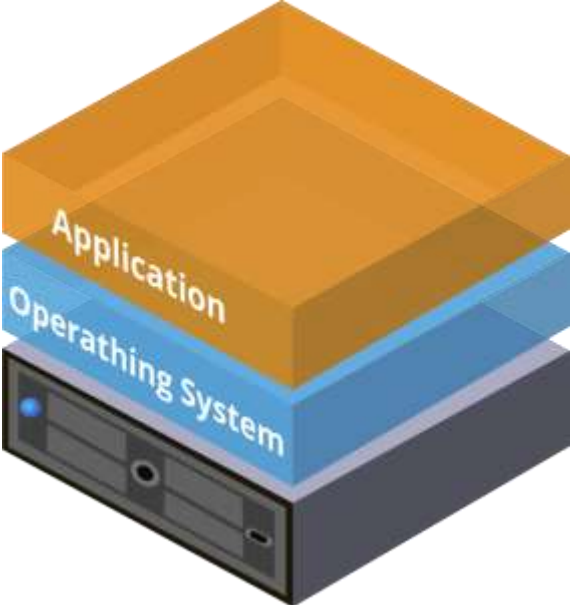
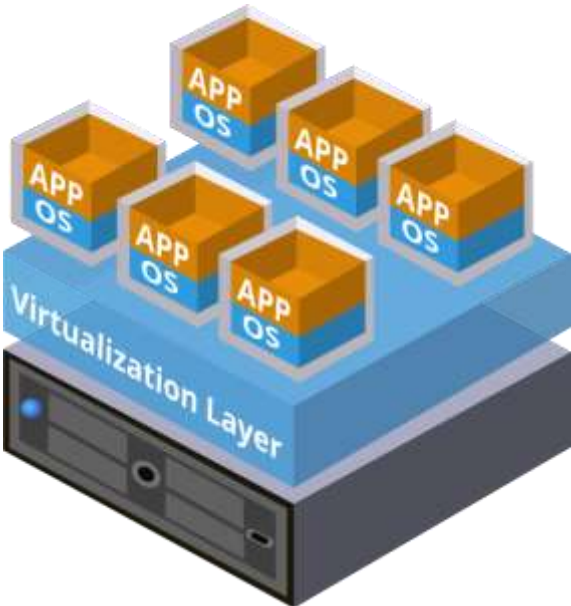




Virtualization & Containerization



Traditional Architecture



Virtual Architecture

Versioning

- There are a lot of challenges that we face when running different operating systems and even different versions of operating systems.
- This comes with complexities of modern businesses processing which relies on a large variety of software and hardware.
- But this complexity doesn't end with operating systems. Even if there was complete standardization of hardware and operating systems there is still the challenge of managing the software.
- It can easily be the case that one machine running a single operating system will run two different software applications. Each of these applications will have their set of dependencies which includes separate libraries and operating system libraries. If one of the software applications is updated and it requires an update of its supporting packages and if that support update breaks the other software then there is a real problem.

Boot Machines

- One solution to having many more environments than physical machines is the use of multiple-boot machines.
- These are computers that are configured with multiple installed operating systems and which are capable of selecting one of these during the boot process. To support this idea there are a number of boot-managers. This approach pushes our complexity to the boot manager and disk configurations.
- Even when this approach works, there are concerns. Often operating systems need control of their own disk and sometimes they even must have control of the primary boot disk. And there are security issues. If you boot a Linux operating system and there are other operating system disks in the same machine, Linux can mount those drives and make data visible without honoring those operating systems' controls.

Virtual Machines

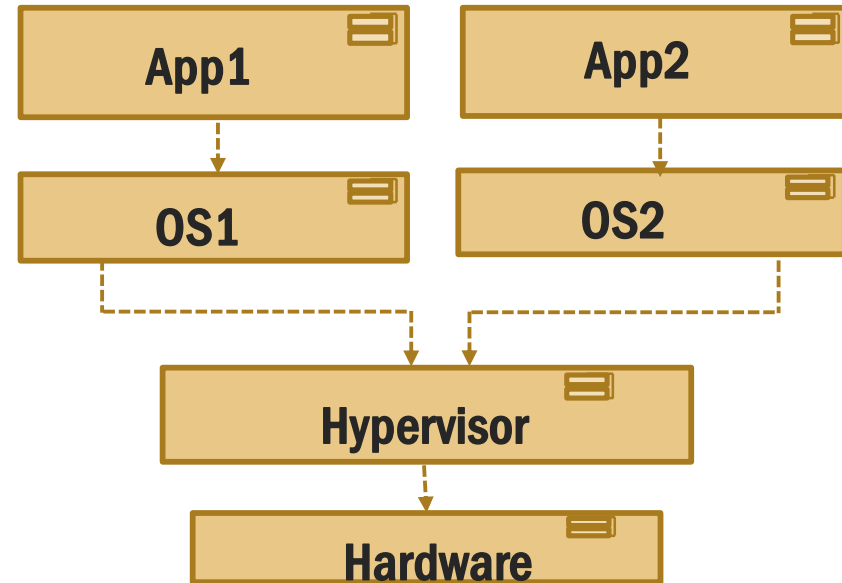
- Virtual machines themselves are not a new idea. In fact, they have been around since the 1960s. Even the company VMWare has been in existence for almost 20 years.
- The essential idea of the early virtualization was to give each running program the appearance of having the whole machine to themselves.
- This has an obvious appeal as it makes designing and writing programs much easier. While these independent programs are running, the virtualization part manages the actual sharing of the computer and all of its resources.

Hypervisors

- At the core of virtualization is the hypervisor.
- This is specialized software that is responsible for managing the actual, very low-level hardware sometimes using specialized reserved CPU instructions.
- The job of the hypervisor and the virtualization system is to make it appear to the hosted software that there is a computer system with certain specific characteristics.

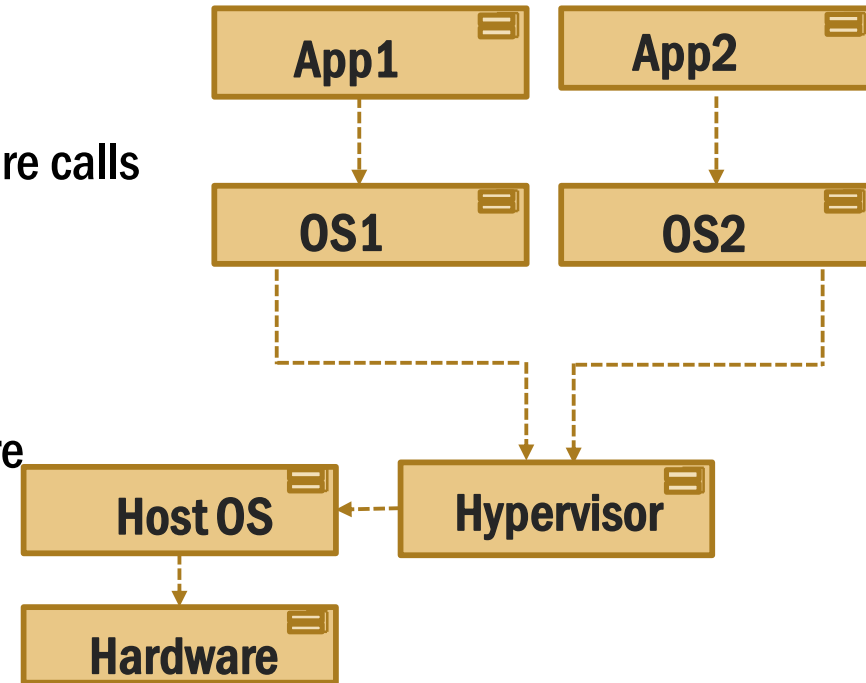
Type 1 Hypervisors

- Virtualization software is called a Hypervisor. It provides the guest OS with configurable access to real and emulated hardware.
- Type-1 or native or bare metal hypervisors
 - Run directly on the host hardware
 - Usually based on a modified Linux kernel
 - Guest operating systems run as processes on the host
- Usually used on high end servers
 - Products include Oracle VM Server, Citrix XenServer, VMware ESX/ESXi, and Microsoft Hyper-V



Type 2 Hypervisor

- **Type-2 or hosted hypervisors**
 - Run as a process on a conventional OS
 - Often need to map the guest OS hardware calls onto host OS system calls
- **Used on desktop and laptop computers**
 - Products include VMware Player, VMware Workstation, and Oracle VirtualBox



Linux Virtualization Unix Jails

The chroot mechanism confines a process and its sub-processes to a restricted area of the file system

UNIX has long had the facility to confine applications

It is possible for processes with root privileges to jail break

An application and its dependencies can be isolated from other processes

This is known as Operating System level virtualization

Container technologies enable the construction of secure jails

A namespace wraps global system resources

- Processes can be put into namespaces
- Processes in a namespace appear to have their own instances of global resources
- They are important for implementing containers

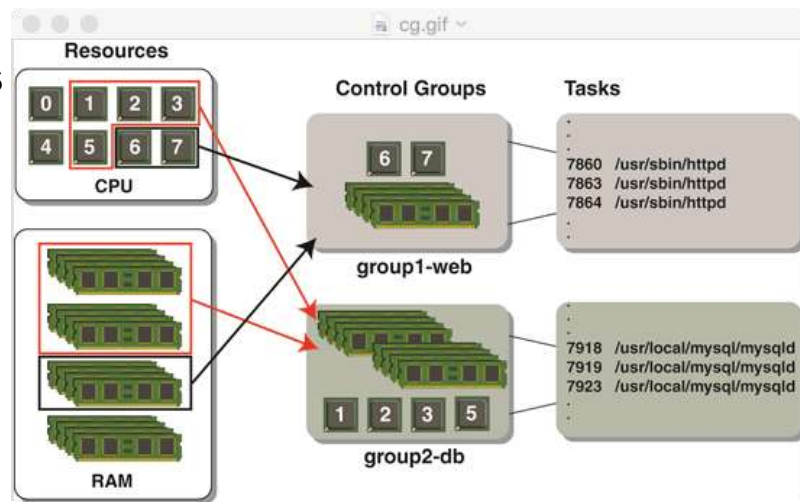
Processes can move between namespaces

- New namespaces can be created for a process when it is created
- Processes can join an existing namespace
- Processes can be moved to a new namespace

The two most important of these namespaces are the user and group ID namespace and the process identifier (or PID) namespace.

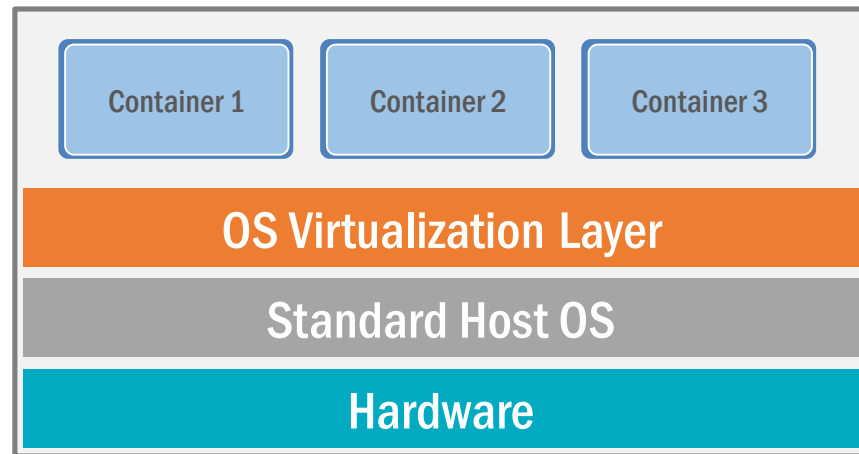
Control Groups

- Control groups (Cgroups) are a Linux kernel mechanism for partitioning the use of resources for a collection of processes
 - Started by Google in 2006, originally called process containers
 - Merged into the Linux kernel 2.6.24 in 2008
- A Cgroup associates a set of processes with a set of subsystems
- A subsystem is a resource controller, which:
 - Schedules a resource or applies limits
 - Can be anything that acts on a group of processes



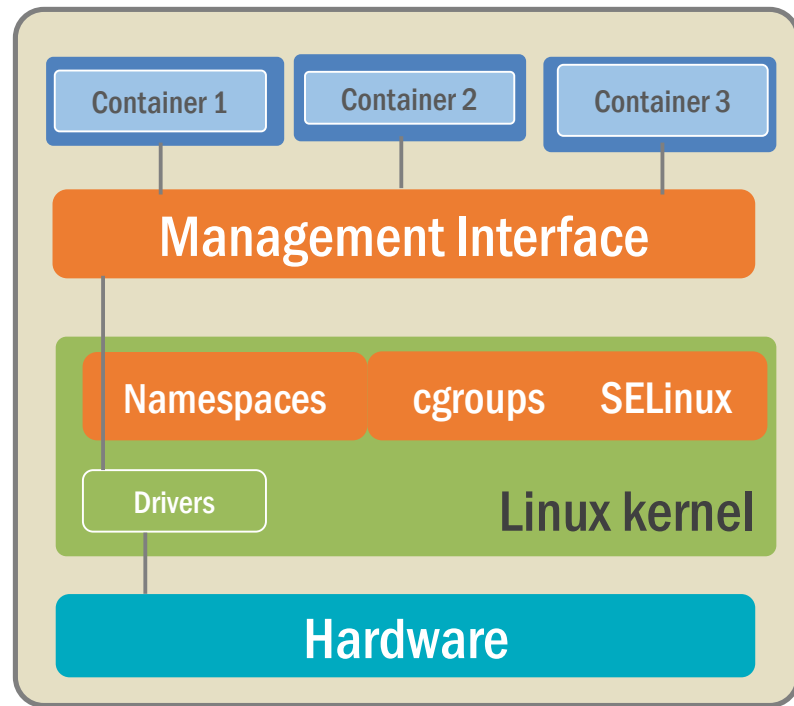
Containers

- 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



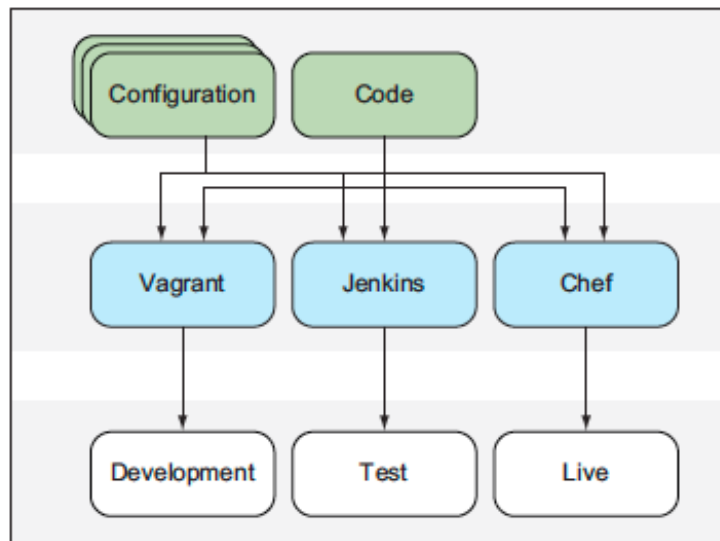
Union File Systems

- **A Union File System (Unionfs) is a Linux and *BSD file system**
 - It allows a union mount of file systems
 - Files and directories of the separate file systems are called branches
 - Branches are overlaid to form a single file system
- **Branches are overlaid with a priority**
 - Branches can be read only or read write
 - Same path directories are merged
 - Same path files only, the highest priority file is visible

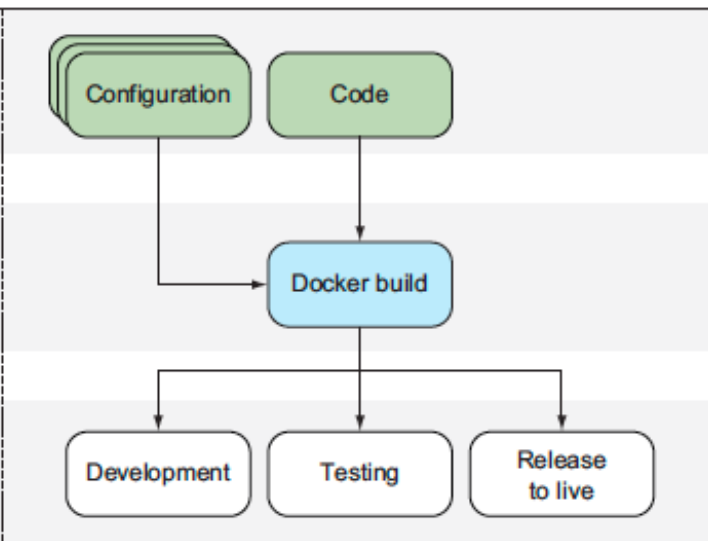
Docker

- 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

Life before Docker



Life with Docker





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 instances can't be accessed remotely due to limited network support

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 specific features

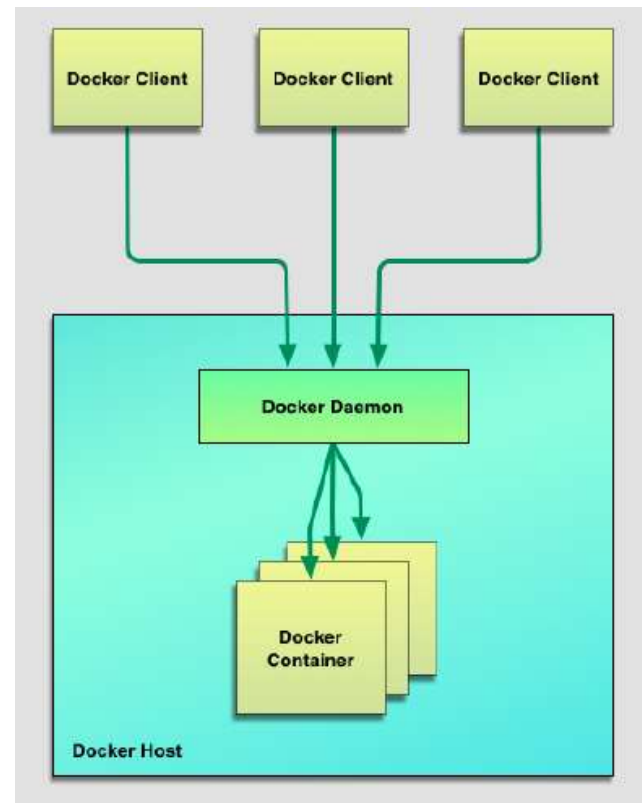


Docker can't run alongside VirtualBox VMs

Docker Toolbox

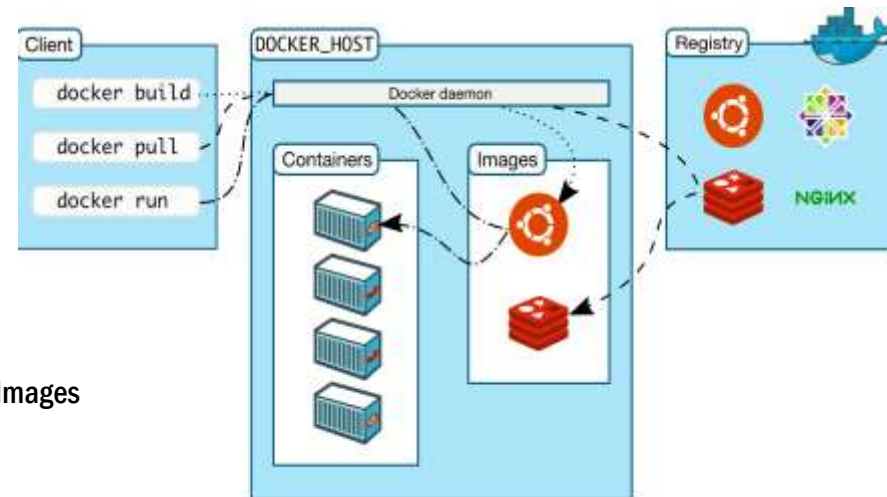
For computers not meeting these specification, Docker Toolbox is available.

- It requires a 64 bit operating system
- It is a local installation of Docker-machine, a client program to control the Docker daemon, and a local VirtualBox compatible version
 - A local install of VM such as VirtualBox hosts Linux which runs Docker Engine
 - A remote computer prepared to run Docker Engine



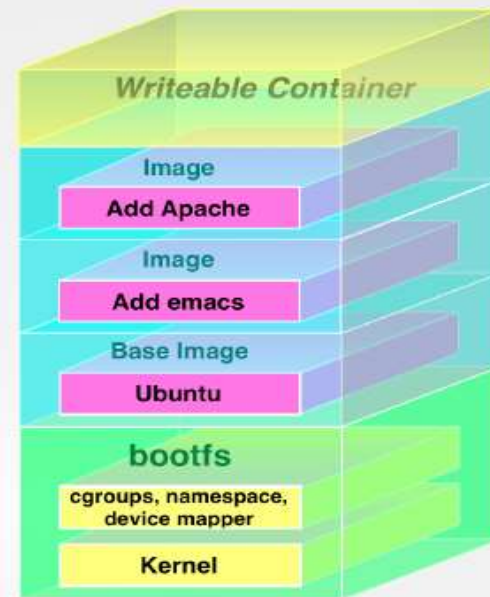
Docker Architecture

- 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 and 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 “DockerFile”

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

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
```

Running a Container Interactively

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 centosC1
docker rm centosC1
```

Attaching Running Containers

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

On first terminal:

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

On second terminal:

```
docker attach centosC1  
> date
```

Notice that you now have two attached terminal sessions to the same running container.

Stopping Running Containers

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 centosC1
```

Pausing Running 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
```

On first terminal:

```
> whoami  
> date
```

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
```


Removing Containers

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
```

Exercise 1 – Find & Run Containers

- **Install Docker for Linux on your virtual machine.**
- **Pull a Linux image from Docker Hub**
- **Run a container without a command**
- **Run a container with a command**
- **Try out some of the commands**

Docker Images

- 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

```
docker search centos
```

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
```

Verifying Changes and Commit

- 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 centosC1 centos-net
```

Verifying Changes

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 inspect -f {{.Architecture}} centos
```


Automating Builds

- 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/
```

Add

- 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
```

Env

- 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
```

Command

- The Dockerfile `CMD` command sets the default application.

```
CMD /bin/bash
```


Building

- 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 - Dockerfile

- Install openssh on your VM.
- Create ssh keys.
- Create a Docker build directory containing a Dockerfile and public key.
- Build a Docker image.
- Run a container from the image.
- Connect to the container using ssh.

- 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 Registry

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**

Using Docker Hub Repository

- 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
```


Using Docker Hub Repository

- 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**

Creating Private Registry

- 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
```

Running the Registry

- 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
```

Using 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 – Docker Registry

- Pull the latest CentOS image.
- Start it as a container.
- Install the docker-registry RPM.
- Write a script to run the registry.
- Start the registry.
- Push an image into the registry.
- Verify that the image can be found by searching the registry.

Exporting a Container

```
root@test01:~# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
2fa250ffb93b	ubuntu:14.04	"/bin/bash"	About an hour ago
c1803ff982ca	ubuntu:14.04	"/bin/bash"	22 hours ago
f76c9b9f4a41	hello-world	"/hello"	8 days ago
15cde3dab21c	hello-world	"tail -f /dev/null"	8 days ago
6206c23a6b92	hello-world	"/hello"	8 days ago

```
root@test01:~# docker export 2fa250ffb93b > update.tar
root@test01:~# ls
update.tar  WordPress
root@test01:~#
```

Export a stopped container into a tarball/ tarfile

Importing a Container

```
root@test01:~# docker import - update < update.tar
sha256:6f174220363d05aae66a0c5baa63d18eafa3c891014fc4d15b252626b3ae90de
root@test01:~# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
update	latest	6f174220363d	24 seconds ago
ubuntu	update	e41846db43d7	About an hour ago
<none>	<none>	fb74b3578bd3	22 hours ago
ubuntu	14.04	302fa07d8117	3 weeks ago
hello-world	latest	48b5124b2768	3 months ago

```
root@test01:~#
```

Upload the tar file on a web server and let your collaborator download it and use the import command on his Docker host.

Save Command

```
root@test01:~# docker save -o update1.tar update
root@test01:~# ls -l
total 383564
-rw----- 1 root root 196383744 May  5 17:26 update1.tar
-rw-r--r-- 1 root root 196375552 May  5 16:07 update.tar
drwxr-xr-x 2 root root      4096 May  3 19:52 WordPress
root@test01:~# docker rmi update
Untagged: update:latest
Deleted: sha256:6f174220363d05aae66a0c5baa63d18eafa3c891014fc4d15b252626b3ae90d
e
Deleted: sha256:dea4ebd431871a1550ce3f9a593ea2c9e3786222c73638cbba472ef9a157e08
0
root@test01:~# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
ubuntu	update	e41846db43d7	2 hours ago
210.5 MB			
<none>	<none>	fb74b3578bd3	23 hours ago
188 MB			
ubuntu	14.04	302fa07d8117	3 weeks ago
188 MB			
hello-world	latest	48b5124b2768	3 months ago
1.84 kB			

Load Command

```
root@test01:~# docker load < update1.tar  
dea4ebd43187: Loading layer 196.4 MB/196.4 MB
```

```
Loaded image: update:latest
```

```
root@test01:~# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
SIZE			
update	latest	6f174220363d	About an hour ago
187.3 MB			
ubuntu	update	e41846db43d7	2 hours ago
210.5 MB			
<none>	<none>	fb74b3578bd3	23 hours ago
188 MB			
ubuntu	14.04	302fa07d8117	3 weeks ago
188 MB			
hello-world	latest	48b5124b2768	3 months ago
1.84 kB			

Copying Data to and from Containers

- Use docker cp command to copy files from a running container to the docker host
- The command allows you to copy files to and from the host to container
- The usage of docker cp command is as follows :

```
root@test01:~# docker cp
"docker cp" requires exactly 2 argument(s).
See 'docker cp --help'.
```

```
Usage:  docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-
        docker cp [OPTIONS] SRC_PATH|- CONTAINER:DEST_PATH
```

Copy files/folders between a container and the local filesystem

```
root@test01:~#
```

Copying Data to and from Containers

- Copy data from Container to Host

```
root@test01:~# docker run -d --name testcopy ubuntu sleep 360
ab6e79055874b95ce464d453b87d9c6c3ad6de5861567539d3d43c92e0880e49
root@test01:~# docker exec -it testcopy /bin/bash
root@ab6e79055874:/# cd /root
root@ab6e79055874:~# echo 'I am in the container' >file.txt
root@ab6e79055874:~# exit
exit
root@test01:~# docker cp testcopy:/root/file.txt .
root@test01:~# cat file.txt
I am in the container
root@test01:~#
```

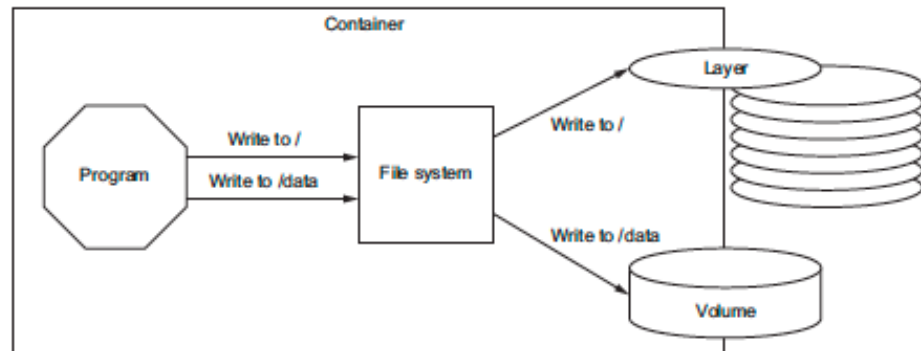
Copying Data to and from Containers

- Copy data from Host to Container

```
root@test01:~# echo 'I am in the host' > host.txt
root@test01:~# docker cp host.txt testcopy:/root/host.txt
root@test01:~#
```

Docker Data Volumes

- A data volume is a directory structure mounted into a Docker container
- It is not part of the union file system
- It can be shared and reused between containers
- It is written to directly
- It does not become part of an image if the container is committed
- It persists even if the container is deleted



Creating Data Volumes

- Data volumes can be created explicitly using the create command
- Data volumes can be created implicitly using the run command
- Any number of data volumes can be created and mounted in a container
- The `--volume` or `-v` option can be used with the Docker run command
- The mount point directory gets created automatically

```
docker volume create --name mydata
```

```
docker run -it --name centosC1 --volume mydata:/opt/data centos /bin/bash
```

Locating Data Volumes

- Data volumes can be located using `Docker inspect` on a container
- Look for the **Mounts** section in the output.
- By default they have rather long names and have been shortened using:

```
"Mounts": [  
  {  
    "Name": "6f9f9518...d25561f58bdc3e",  
    "Source": "/var/lib/docker/volumes/6f9f9518...d25561f58bdc3e/_data",  
    "Destination": "/opt/data",  
    "Driver": "local",  
    "Mode": "",  
    "RW": true,  
    "Propagation": ""  
  }  
],
```

```
docker inspect centosC1
```

Managing Data Volumes

- Data volumes can be listed and removed
 - `ls -q` just prints the volume name
- Volumes can't be removed if in use by containers
- Data Volumes can be created and named
 - They can be mounted into a container explicitly

```
docker volume ls
```

```
docker volume rm $(docker volume ls -q)
```

```
docker volume create --name jabber  
docker volume ls
```

```
docker run -it --name centosC1 --volume jabber:/opt/data centos /bin/bash
```

Copying Data Volumes

- Data volumes can be mounted from an existing container
- All mounted data volumes are mounted
- They have the same mount points

```
docker run -it --rm --volumes-from centosC1 centos /bin/bash
```

Mounting File Systems

- Directories and files can be mounted from the host filesystem
 - Specify local_path:container_path
 - Both paths must be absolute
 - On Windows use /c/Users etc
- Editors such as vi can't be used to edit explicitly mounted files as they try and change the inode
 - It is better to mount the parent directory

```
docker run -it --rm --volume /opt/java:/opt/java centos /bin/bash
```

Options

- The `-v` or `--volume` parameter takes the form
 - `[host-path:]container-path[:options]`
- Options are comma separated
 - The option `ro` mounts the volume read-only
 - There is also `rw` for read-write which is the default
- If there are files under the container mount point they get copied into the volume
 - Unless a local file or directory is being mounted
 - Unless the `nocopy` option is added

Volume in Dockerfile

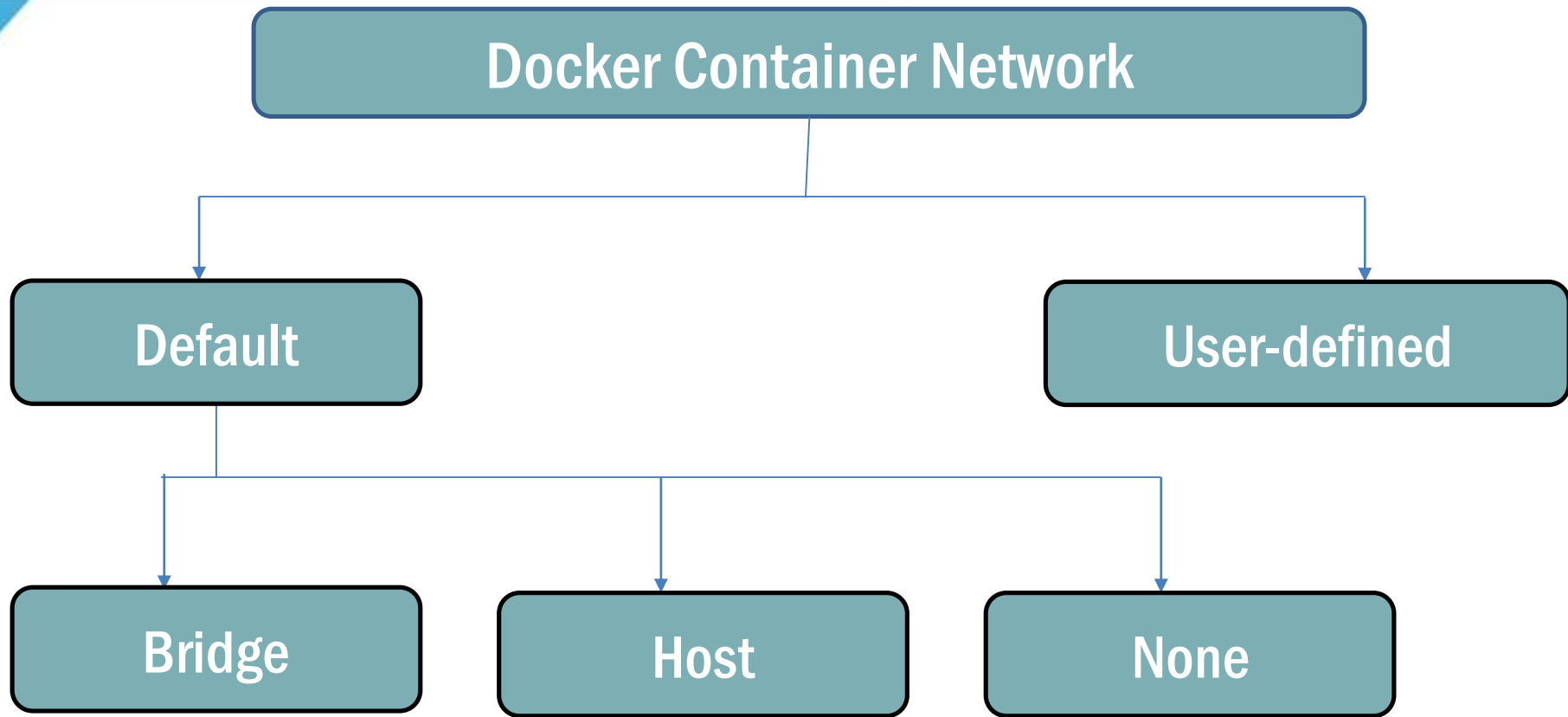
- Dockerfile has a **VOLUME** command and it specifies one or more mount points for data volumes
- Any content in the mount point will get copied to the volume
- Any changes to mount points after the **VOLUME** command get discarded

```
FROM centos
RUN mkdir /opt/data
RUN echo "Hello World" > /opt/data/greeting
VOLUME /opt/data
```

Exercise 4 – Docker Volumes

- Start a Docker container with a directory from your VM mounted into it.
- Change the contents of the directory and exit the container.
- See the changes to the file system.
- Create a Docker volume and start a container with it mounted.
- Add content to the volume and exit the container.
- Start a new container with the image mounted and see the new content.

- Docker containers normally contain a single application
- Multiple applications or parts of application can work together though networking between Docker containers
- This is “Docker Container Networking”
- Docker provides a number of networking options



Docker Network - 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

Default—Bridge Network

- 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"
      }
    ]
  }
}]
```

Default—Bridge Network

- 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": ""
    }
  },
```

Default—Bridge Network Hosts File

- 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
```

Default—Host Network

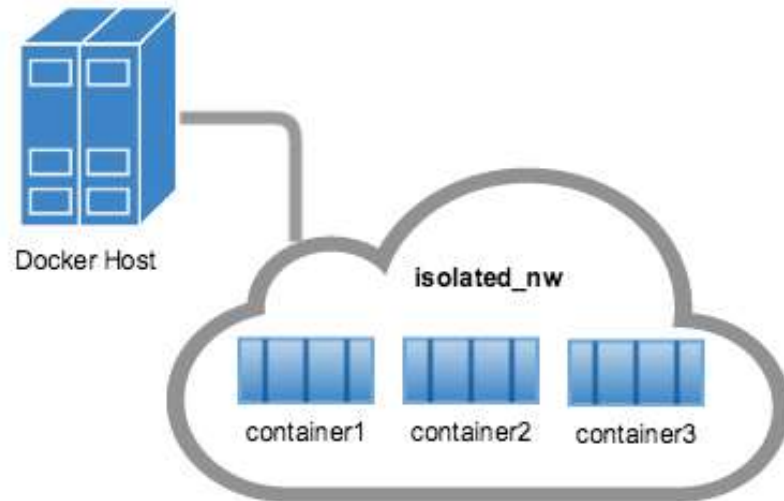
- A container attached to a host network has the same network as the host
- It has the same network configuration as the host
- It is not used much any more

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 Networks—User-defined

- 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 Networks

- 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
```

- **A container can be accessed remotely by port binding**
 - The listener port on the container is bound to a port on the host
- **The container can then be accessed through the host IP**
 - User name and password are required to be set up in the container for remote ssh access

```
docker run -it --name ssh -p 2222:22  
centos-ssh  
ssh -p 2222 root@localhost
```

Exercise 5 – Docker Networking

- **Ensure Docker is running on your machine.**
- **Create a bridged network and check if it is in the list of networks.**
- **Start an Alpine Linux container running a shell.**
- **Add the networking tools package and list the interfaces on the container.**
- **Connect the new bridged network to the container and verify that a new interface has been created.**
- **Disconnect the bridge network and verify that the interface gets removed.**

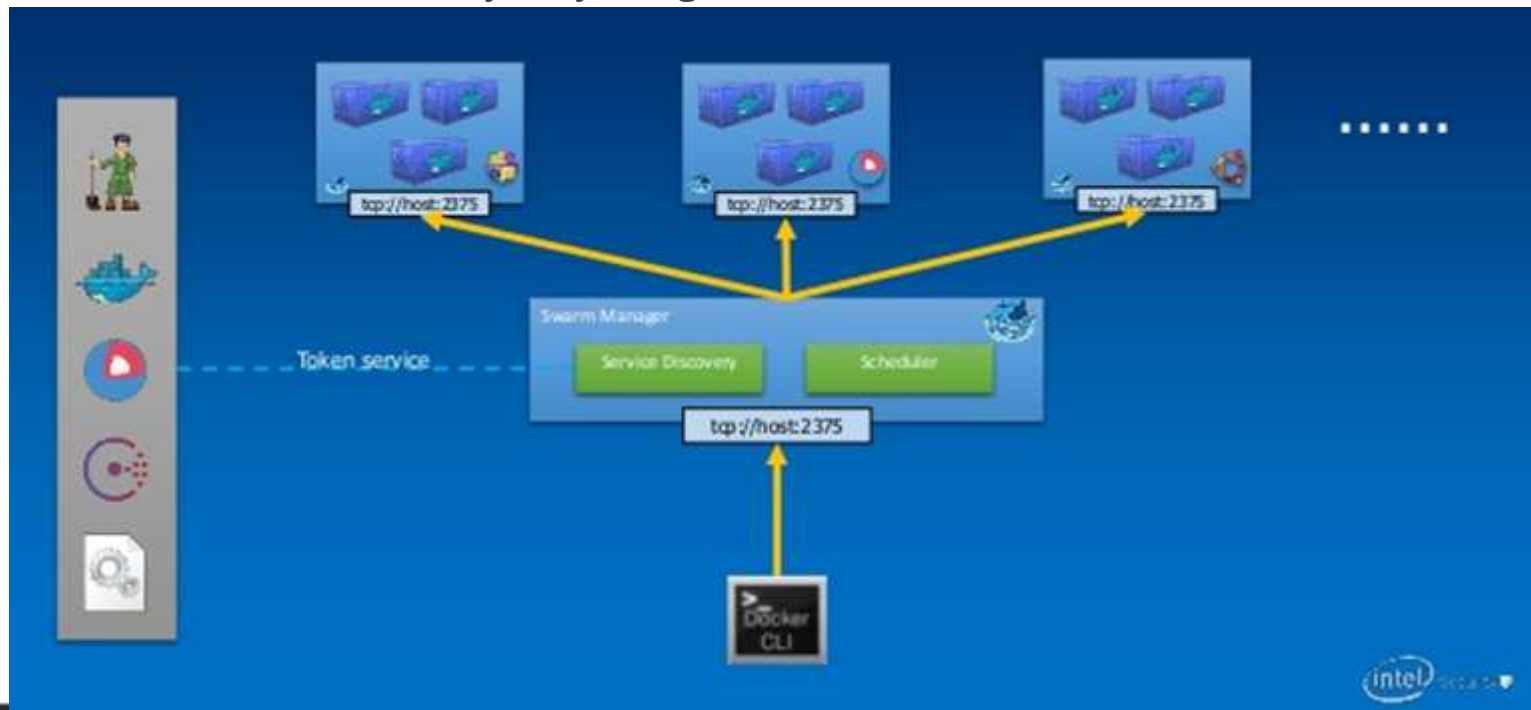
Docker Swarm

- Docker Swarm is clustering for Docker
- It turns several Docker hosts into a single virtual Docker host
- The regular Docker client works transparently with Swarm
- The Swarm is controlled by a Swarm Manager
- Each Docker node communicates with the manager
- It can be installed manually or by using Docker Machine



Docker Swarm Architecture

- The Swarm is controlled by a Swarm Manager
- Each Docker node communicates with the manager
- It can be installed manually or by using Docker Machine



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





THANK YOU

average 45%