Introduction To Docker

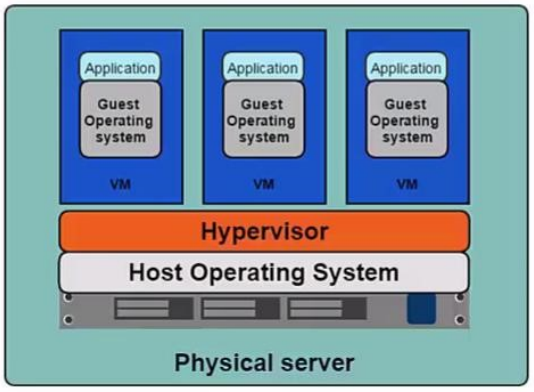
What is Docker?

***Docker** is a platform for developing, shipping and running applications using container virtualization technology*

- The Docker Platform consists of multiple products/tools
  - Docker Engine
  - Docker Hub
  - Docker Machine
  - Docker Swarm
  - Docker Compose
  - Kitematic

Limitations of VMs

- Each VM stills requires
  - CPU allocation
  - Storage
  - RAM
  - An entire guest operating system
- The more VM's you run, the more resources you need
- Guest OS means wasted resources
- Application portability not guaranteed

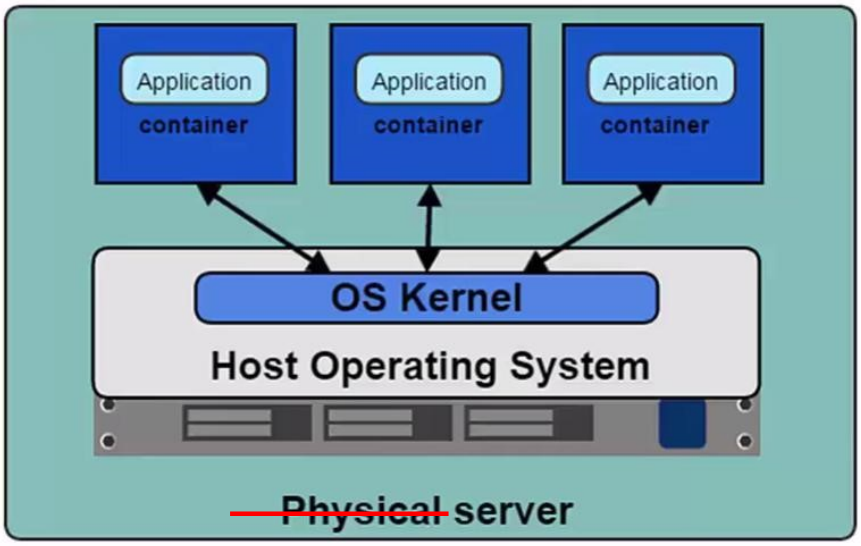


Introducing Containers

Container based virtualization uses the kernel on the host's operating system to run multiple guest instances

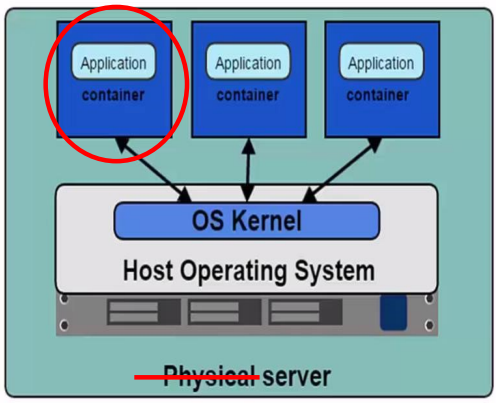
- Each guest instance is called a **container**
- Each container has its own
  - Root filesystem
  - Processes
  - Memory
  - Devices
  - Network ports

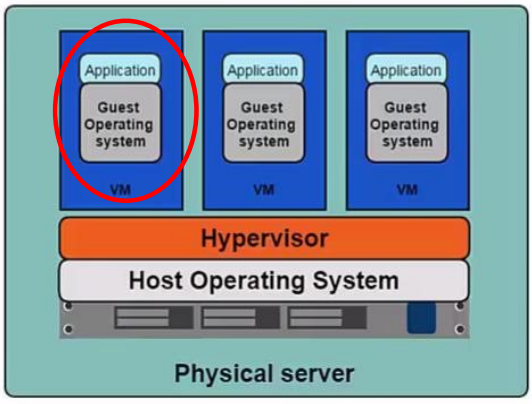
Containers



Containers vs VMs

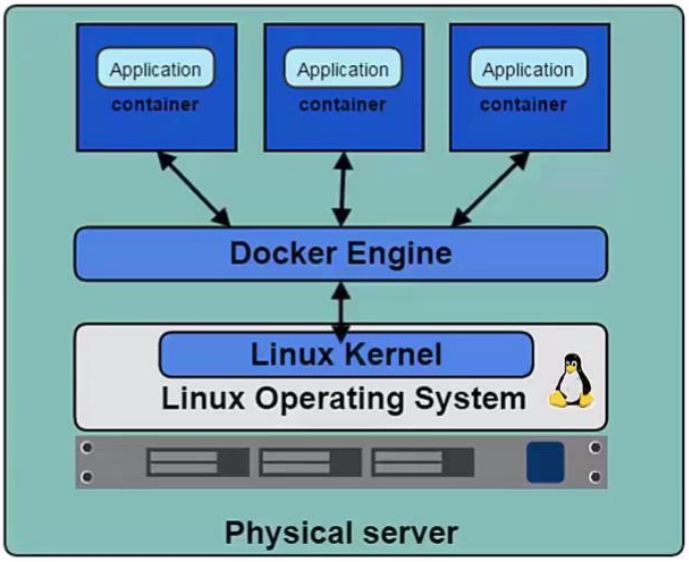
- Containers are more lightweight
- No need to install guest OS
- Less CPU, RAM, storage space required
- More containers per machine than VMs
- Greater portability





Docker and the Linux Kernel

- **Docker Engine** is the program that enables containers to be built, shipped and run.
- Docker Engine uses Linux Kernel namespaces and control groups
- Namespaces give us the isolated workspace



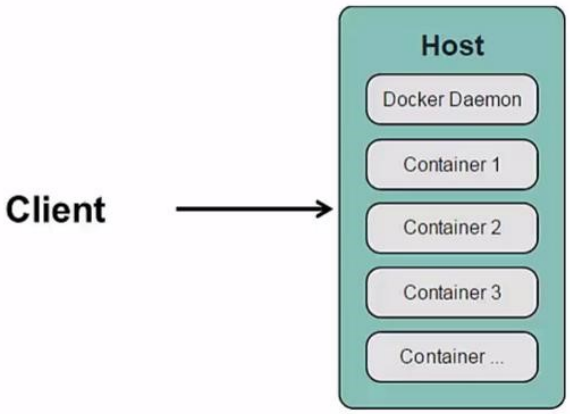
Install Docker

EXERCISE

1. Follow the instructions at https://docs.docker.com/installation/ to install the latest Docker maintained Docker package on your preferred operating system
2. Run the `hello-world` container to test your installation
   ```
   sudo docker run hello-world
   ```
3. Add your user account to the `docker` group
   ```
   sudo usermod -aG docker <user>
   ```
4. Logout of your terminal and log back in for the changes to take effect
5. Verify that you can run the hello-world container without using `sudo`
   ```
   docker run hello-world
   ```

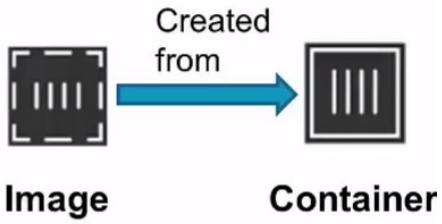
Docker Client and Daemon

- Client / Server architechture
- Client takes user inputs and send them to the daemon
- Daemon builds, runs and distributes containers
- Client and daemon can run on the same host or on different hosts
- CLI client and GUI (Kitematic)

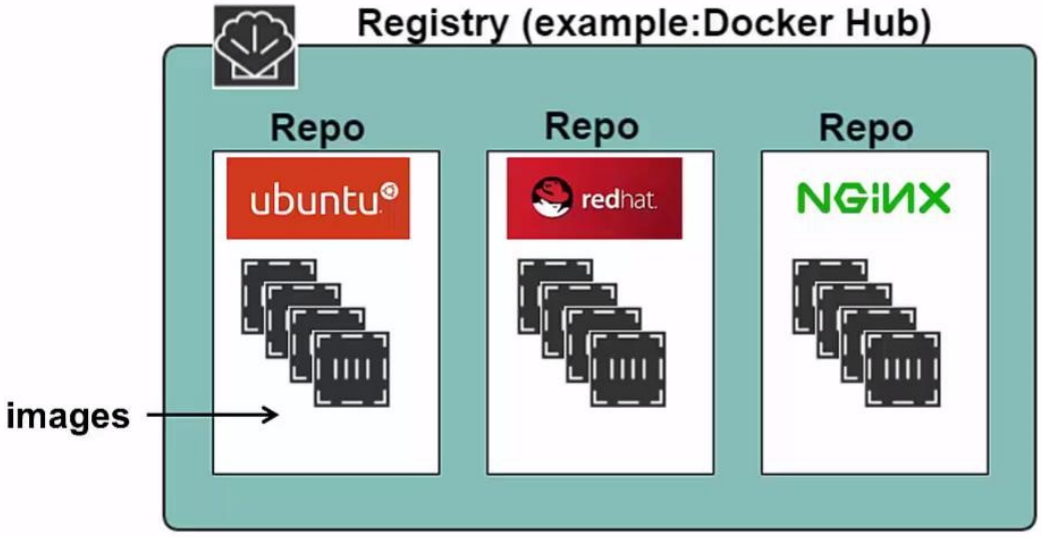


Docker Containers and Images

- Images
  - Read only template used to create containers
  - Built by you or other Docker users
  - Stored in the Docker Hub or your local Registry
- Containers
  - Isolated application platform
  - Contains everything needed to run your application
  - Based on one or more images

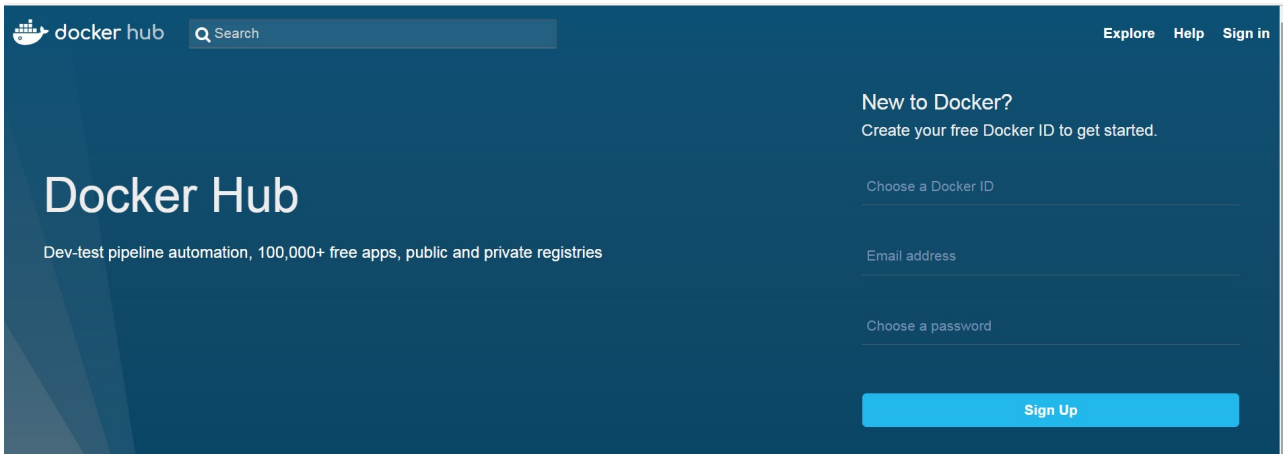


Registry and Repository



Docker Hub

Docker Hub *is the public registry that contains a large number of images available for your use*



Docker Orchestration

- Three tools for orchestrating distributed applications with Docker
- Docker Machine
  - Tool that provisions Docker hosts and installs the Docker Engine on them
- Docker Swarm
  - Tool that clusters many Engines and schedules containers
- Docker Compose
  - Tool to create and manage multi-container applications

Benefits of Docker

- Separation of concerns
  - Developers focus on building their apps
  - System admins focus on deployment
- Fast development cycle
- Application portability
  - Build in one environment, ship to another
- Scalability
  - Easily spin up new containers if needed
- Run more apps on one host machine

Create a Docker Hub Account

EXERCISE

1. Go to https://hub.docker.com/account/signup/ and signup for an account if you do not already have one.
   No credit card details are needed
2. Find your confirmation email and activate your account
3. Browse some of the repositories
4. Search for some images of your favourite dev tools, languages, servers etc…
   a) (examples: Java, Perl, Maven, Tomcat, NGINX, Apache)

Display Local Images

- Run **`docker images`**
- When creating a container Docker will attempt to use a local image first
- If no local image is found, the Docker daemon will look in Docker Hub unless another registry is specified

Image Tags

- Images are specified by **repository:tag**
- The same image may have multiple tags
- The default tag is `latest`
- Look up the repository on Docker Hub to see what tags are available

Creating a Container

- Use **`docker run`** command
- Syntax
  `sudo docker run [options] [image] [command] [args]`
- Image is specified with `repository:tag`

Examples

```
docker run ubuntu:14.04 echo "Hello World"
docker run ubuntu ps ax
```

Run a Simple Container

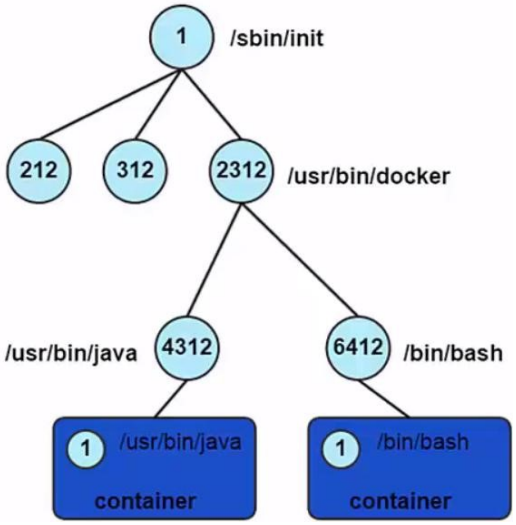
1. On your terminal type
   `docker run ubuntu:14.04 echo "hello world"`
2. Observe the output
3. Then type
   `docker run ubuntu:14.04 ps ax`
4. Observe the output
5. Notice the much faster execution time compared to the first container that was run. This is due to the fact that Docker now has the Ubuntu 14.04 image locally and thus does not need to download the image

Run a Container and get Terminal Access

1. Create a container using the ubuntu 14.04 image and connect to STDIN and a terminal
   ```
   sudo docker run -i -t ubuntu:14.04 /bin/bash
   ```
2. In your container, create a new user using your first and last name as the username
   ```
   adduser username
   ```
3. Add the user to the sudo group
   ```
   adduser username sudo
   ```
4. Exit the container
   ```
   exit
   ```
5. Notice how the container shut down
6. Once again run:
   ```
   sudo docker run -i -t ubuntu:14.04 /bin/bash
   ```
7. Try and find your user
8. Notice that it does not exist

Container Processes

- A container only runs as long as the process from your specified `docker run` command is running
- Your command's process is always PID 1 inside the container



Container ID

- Containers can be specified using their ID or name
- Long ID and short ID
- Short ID and name can be obtained using `docker ps` command to list containers
- Long ID obtained by inspecting a container

Find your Containers

- Use **`docker ps`** to list running containers
- The `-a` flag to list all containers (includes containers that are stopped)

```
testuser@new-docker:~$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS                      PORTS                              NAMES
c7e1d768c8f0        tomcat:latest       "catalina.sh run"        5 minutes ago       Up 5 minutes                0.0.0.0:8080->8080/tcp             reverent_fermat
7b6e57c6af9f        redis:latest        "/entrypoint.sh redi"    6 minutes ago       Up 6 minutes                0.0.0.0:6379->6379/tcp             silly_franklin
fe8a875623c2        nginx:latest        "nginx -g 'daemon of"    6 minutes ago       Up 6 minutes                443/tcp, 0.0.0.0:80->80/tcp        pensive_lumiere
b26004b82658        redis:latest        "/entrypoint.sh redi"    7 minutes ago       Exited (0) 6 minutes ago                               grave_perlman
4e69b0d11895        nginx:latest        "nginx -g 'daemon of"    7 minutes ago       Exited (0) 6 minutes ago                               cranky_ritchie
51c44ec7ba98        ubuntu:14.04        "ps ax"                  13 minutes ago      Exited (0) 13 minutes ago                              high_hodgkin
b266cb5d91b3        ubuntu:14.04        "ps a"                   13 minutes ago      Exited (1) 13 minutes ago                              cocky_almeida
24fe5183ff9b        ubuntu:14.04        "echo Hello world"       13 minutes ago      Exited (0) 13 minutes ago                              adoring_einstein
bccf114cb92c        hello-world:latest  "/hello"                 4 hours ago         Exited (0) 4 hours ago                                 desperate_tesla
d3bbb5187818        hello-world:latest  "/hello"                 4 hours ago         Exited (0) 4 hours ago                                 mad_hypatia
```

Running in Detached Mode

- Also known as running in the background or as a daemon
- Use `-d` flag
- To observe output use **`docker logs [container id]`**

Create a centos container and run the ping command to ping the container itself 50 times

```
docker run -d centos:7 ping 127.0.0.1 -c 50
```

List Your Containers

EXERCISE

1. Run
   `docker run -d centos:7 ping 127.0.0.1 -c 50`
2. List your containers by running `docker ps`
3. Notice the centos container running
4. Run `docker ps -a`
5. Notice all the containers created from the previous exercises

A More Practical Container

- Run a web application inside a container
- The `-P` flag to map container ports to host ports

Create a container using the tomcat image, run in detached mode and map the tomcat ports to the host port

```
docker run -d -P tomcat:7
```

Run a Web Application Container

EXERCISE

1. Run
   `docker run -d -P tomcat:7`
2. Check your image details by running
   `docker ps`
3. Notice the port mapping. The container's port 8080 is mapped to a random port on your host machine

```
PORTS
0.0.0.0:49155->8080/tcp
```

4. Go to `<your linux server url>:<port number>` and verify that you can see the Tomcat page

Command Reference

- **`docker run`**
  https://docs.docker.com/reference/commandline/cli/#run
  https://docs.docker.com/reference/run
- **`docker images`**
  https://docs.docker.com/reference/commandline/cli/#images
- **`docker ps`**
  https://docs.docker.com/reference/commandline/cli/#ps