

Week 2 - Docker

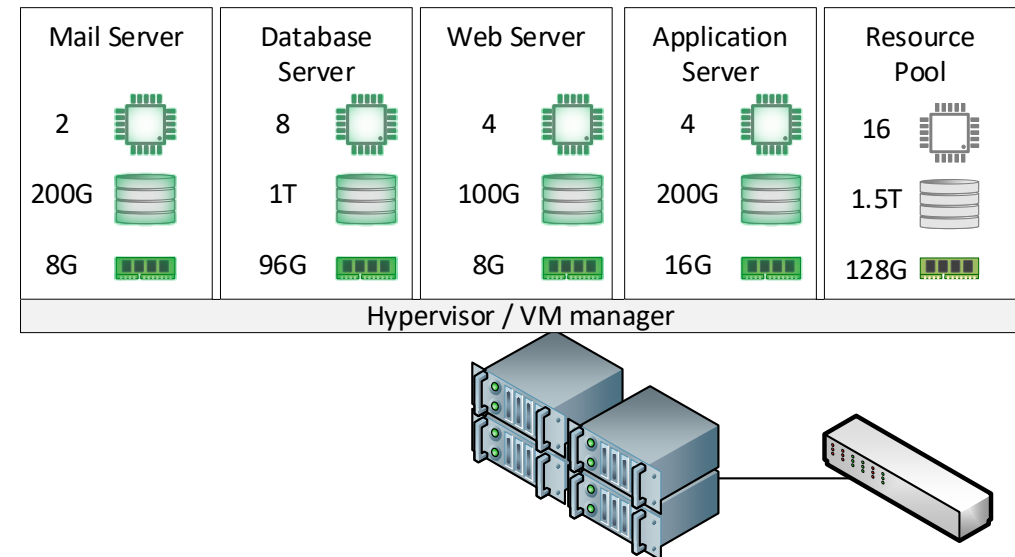
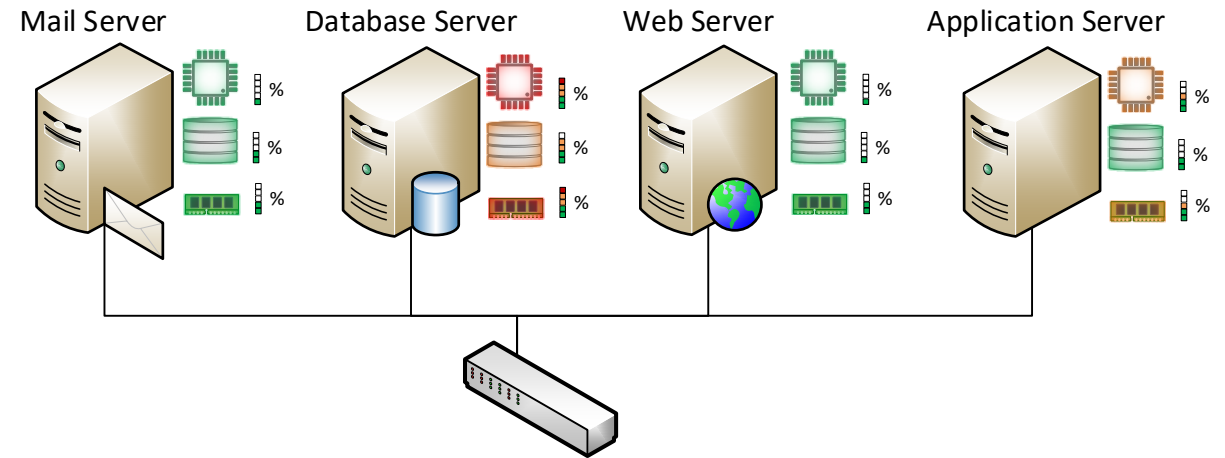
Dr Kiki Adhinugraha

Outline

- Introduction
- Virtualisation, how it works?
- Virtual Machine vs Container
- Containers
- Docker

Virtualisation

- Traditional IT management:
 - Resources are bounded to hardware
 - Over-specifications/Below-specifications
 - Some servers might be overloaded, while others are idle
 - Idle resources are wasted
 - High initial cost
 - High maintenance
- Virtualisation
 - Resource distribution
 - Resources are allocated based on workload
 - Dynamic allocation at anytime
 - Robust, no single point of failure



Why Virtualisation?

- Cost
 - \$ 0 set up cost. Pay as you go
 - \$ 0 hardware maintenance
- Efficiency
 - Consolidate resources (HW/SW)
- Uptime
 - Robust (Snapshot, clone, suspend/resume)
 - Hardware redundancy
- Deployment
 - Simple deployment
 - Live migration
- Energy Savings
 - Data centre is very expensive

Virtual Machine

- At the base is the host, underlying hardware
- Virtual machine manager (VMM)
 - Also known as the hypervisor
 - Creates and runs virtual machines
 - Provides an interface that is identical to the host (except in the case of paravirtualization)
 - Usually, guest is an operating system
 - Configure resources available to each guest operating system E.g. two virtual CPUs, 4GB of RAM, 10GB of disk, etc.

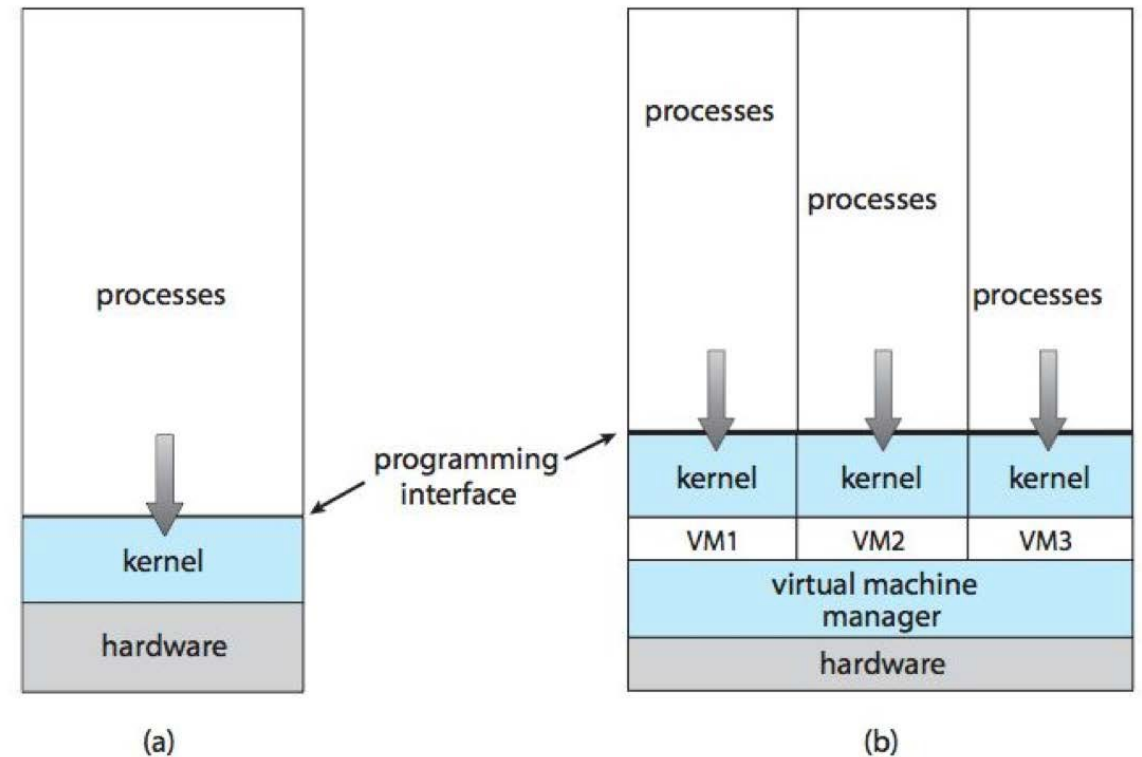
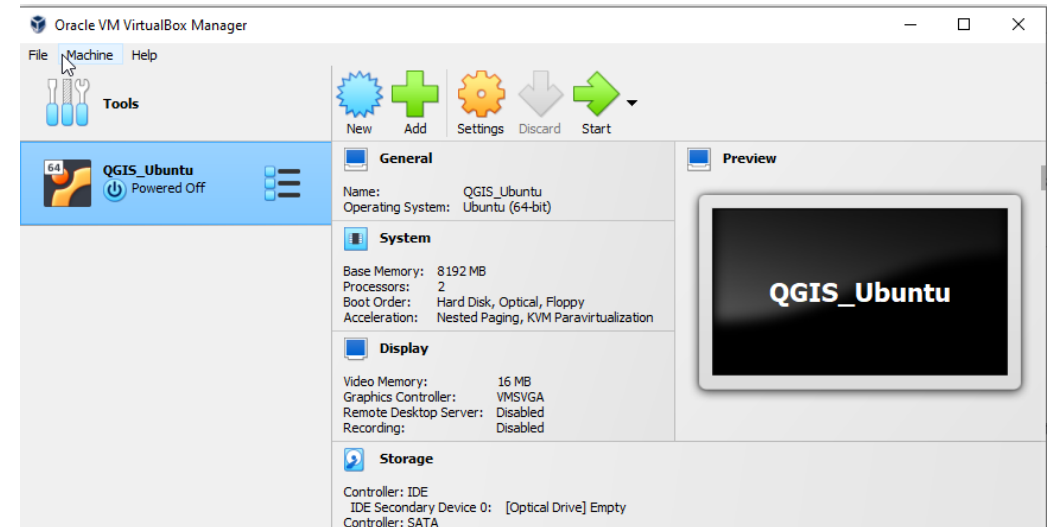


Figure 16.1 System models. (a) Nonvirtual machine. (b) Virtual machine.

Virtual Machine Popularity

- Virtual Machine is popular for
 - Running multiple guest OS in one host
 - Trying system functionality on different OS
 - Deploying or testing system
 - Simulation
- Popular Virtual Machine
 - Oracle Virtual Box
 - VMWare
 - Citrix



Virtual Machine Resource Allocation

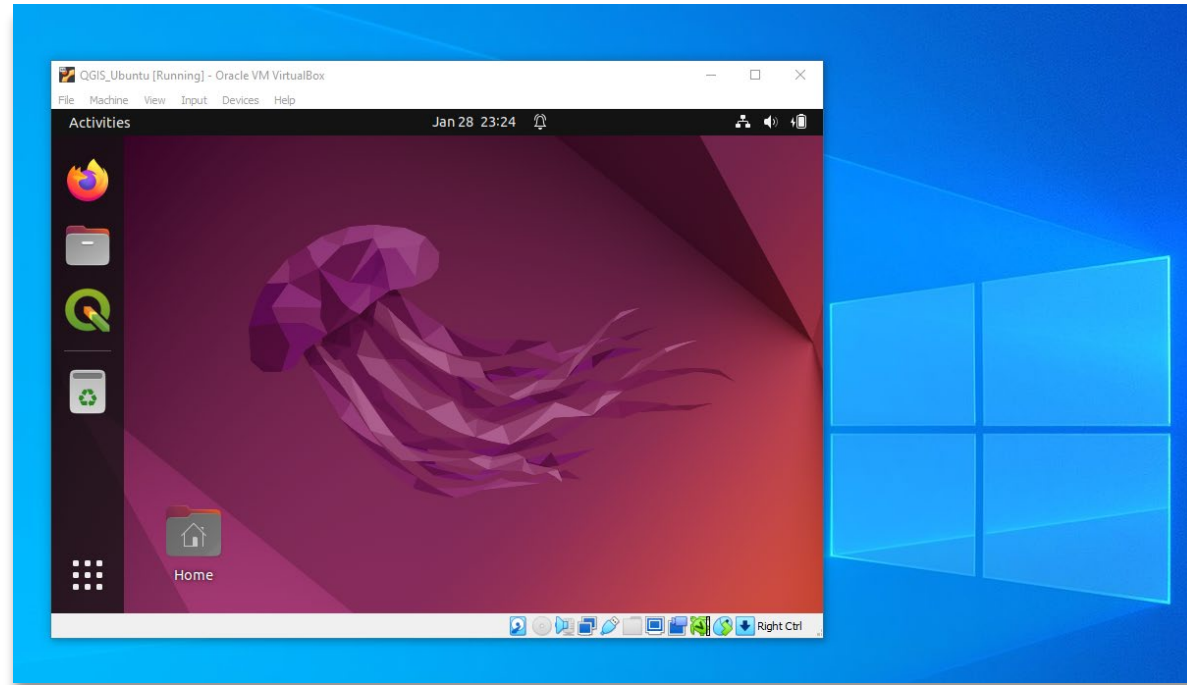
- A Virtual Machine requires dedicated resources that are taken from the host machine.
- The resources are locked exclusively for this virtual machine:
 - CPU – number of cores
 - RAM
 - Storage
 - Video Memory
 - Ports
- Only CPU, RAM, Video memory and ports are configurable after the VM is created

Virtual Machine Characteristic

- Although a VM is created inside a host OS, a VM is treated as an external machine
 - Host OS do not have access to the guest OS in VM, vice versa
 - Host OS and guest OS can share files using shared folders
 - Host OS and guest OS have their own IP Address. If a service is running in the guest OS, the host OS can access the service using guest OS's IP Address
 - When guest OS run out of space, the only option is to recreate the guest OS with larger storage, or attach another storage to the guest OS.
This is similar when your laptop run out of space. It is impossible to enlarge the harddrive.

Virtual Machine Example

- This example shows a Linux Ubuntu Virtual Machine that is running on Windows 10 as a Host OS, using Oracle VirtualBox.



Containers – Not just a Virtualisation

- While Virtual Machine is a revolutionary concept in utilising resources, VM may cause several issues:
 - VM reserves the host resources exclusively, hence will significantly decrease the host performance
 - It's difficult to expand the VM storage
 - File sharing between host and guest could be problematic as VMs are treated as an external machine
 - Multiplexing VM will multiplex the number of resources
 - Applications need to be manually installed and configured

Containers

- Containers are a lighter-weight, more agile way of handling virtualisation — since they don't use a hypervisor, you can enjoy faster resource provisioning and speedier availability of new applications.
- Packages together everything needed to run a single application or microservice (along with runtime libraries they need to run)
- Containers use a form of operating system (OS) virtualisation

<https://youtu.be/EUitQ8DaZW8>

Containers

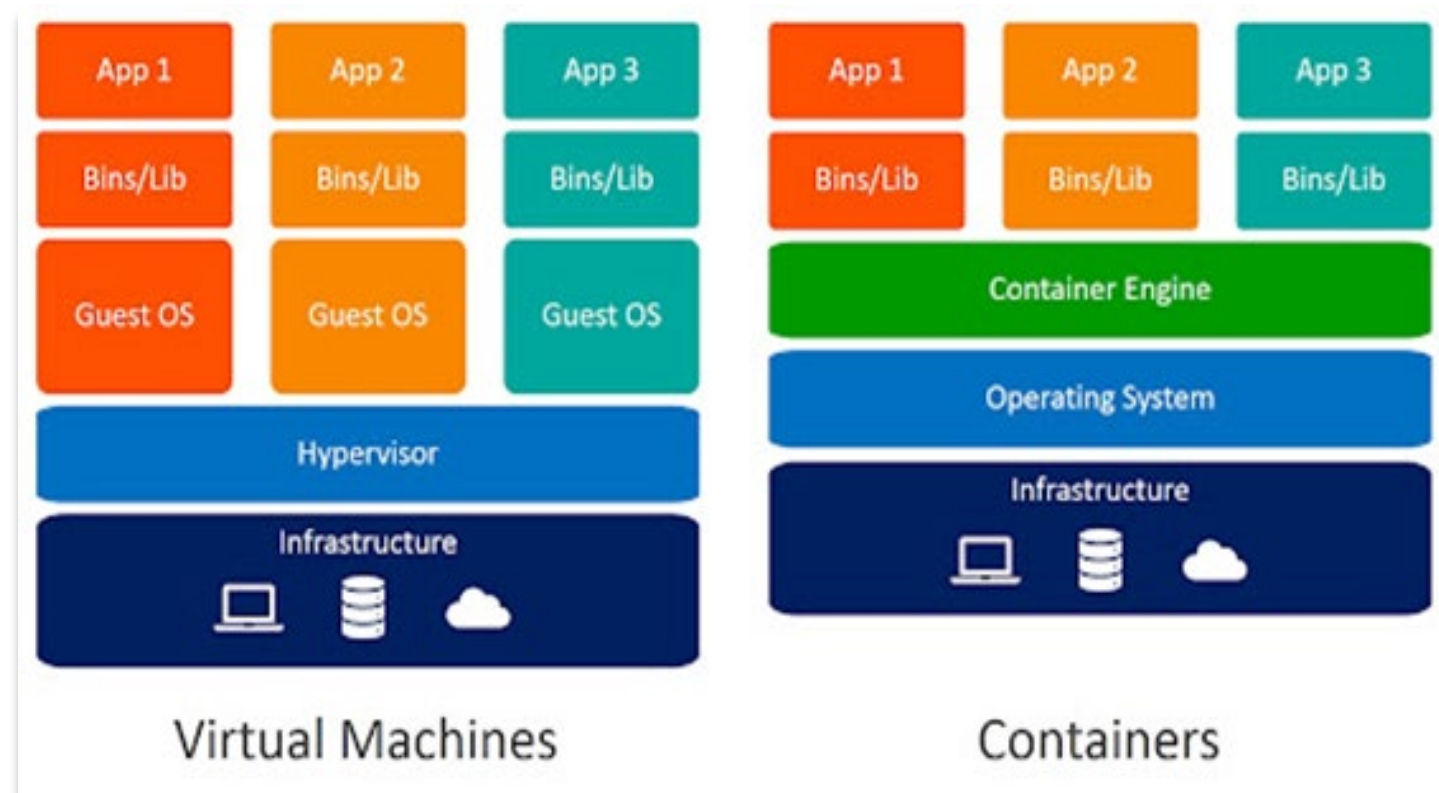
- Virtual machines are great but they are heavy weight (high overhead)
 - They require a hypervisor layer
- In contrast a container is like a light weight virtual machine which uses the host operating system by using the normal systems call interface
 - No need for a hypervisor
- This means containers can not run very different operating systems
 - For example if the host operating system is Linux then the container can not be windows.
 - However, a host operating system running red hat Linux can have a container running Ubuntu Linux

Containers

- Containers used to have problems with isolating users from each other because multiple containers runs within in the same virtual machine.
- Recently Linux has added new features such as
 - Control groups
 - Namespaces
- The result is containers now have strong isolation
 - Their own network stacks
 - Their own storage stacks
- Recent container technologies include
 - OpenVZ, Solaris Zones, lxc, etc.
- Although containers are light weight and offers many benefits like virtual machines in the past they were not used much since they are hard to setup.
- However that all changed with the introduction of Docker!

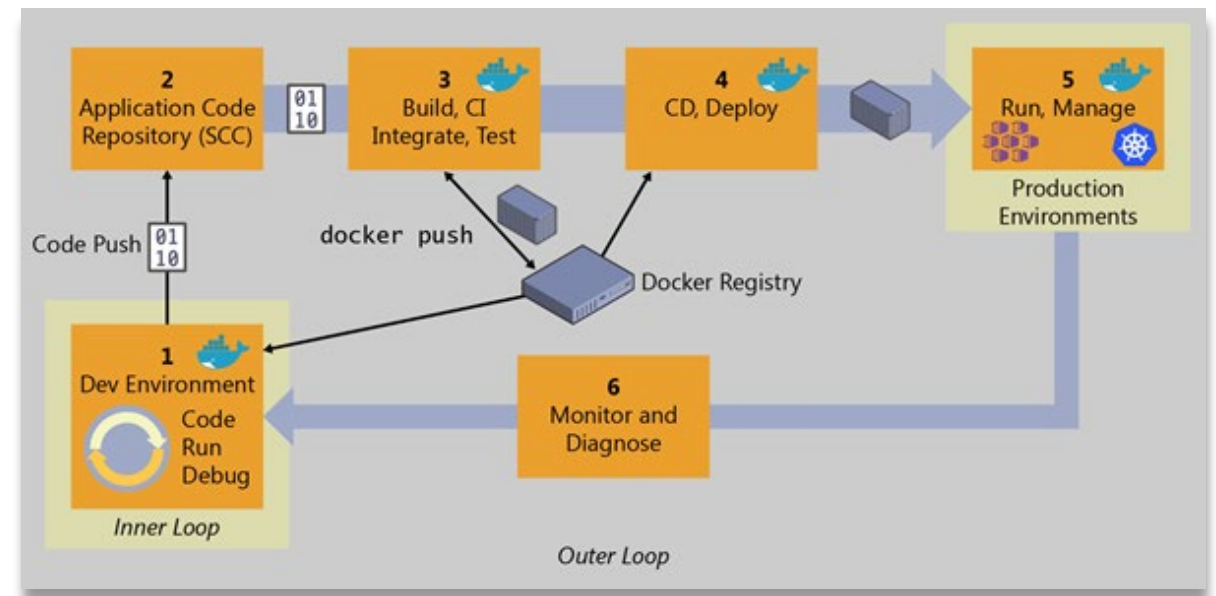
Virtual Machine vs Containers

- VM
 - Hypervisor virtualises physical hardware
 - Requires Guest OS
 - Hardware must be configured independently
 - May contain more than one service
- Docker
 - Lightweight
 - Uses Container engine
 - Utilises host OS and hardware → Platform independent
 - Contains one service for each container



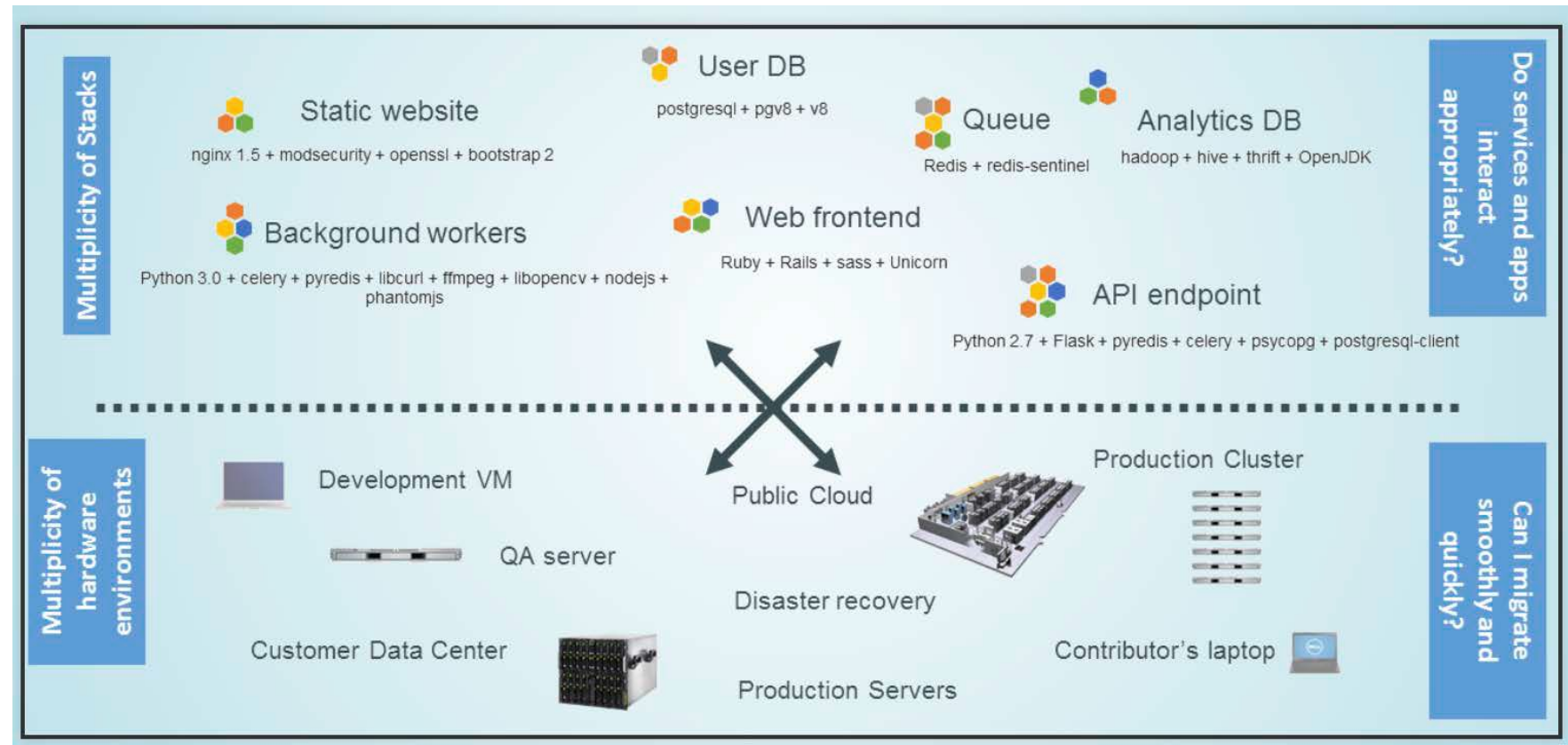
Docker

- Docker is an open source engine that automates the deployment of applications into containers
- Docker adds an application deployment engine on top of a virtual container execution environment.
- It is designed to provide a lightweight and fast environment in which to run your code as well as an efficient workflow to get the code from your laptop to your test environment and then into production
 - Laptop
 - Where you code.
 - Test environment
 - Usually involves running your program in the same kinds of machines that your system will be deployed on. For example Amazon Web Services.
 - Production
 - Where your code is currently running to serve user requests











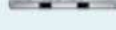




The Challenge

- An application uses multiple software stacks to work.
 - A software stack includes a set of libraries and dependencies that are needed to run a server
- These different software stacks need to be deployed on different hardware environments

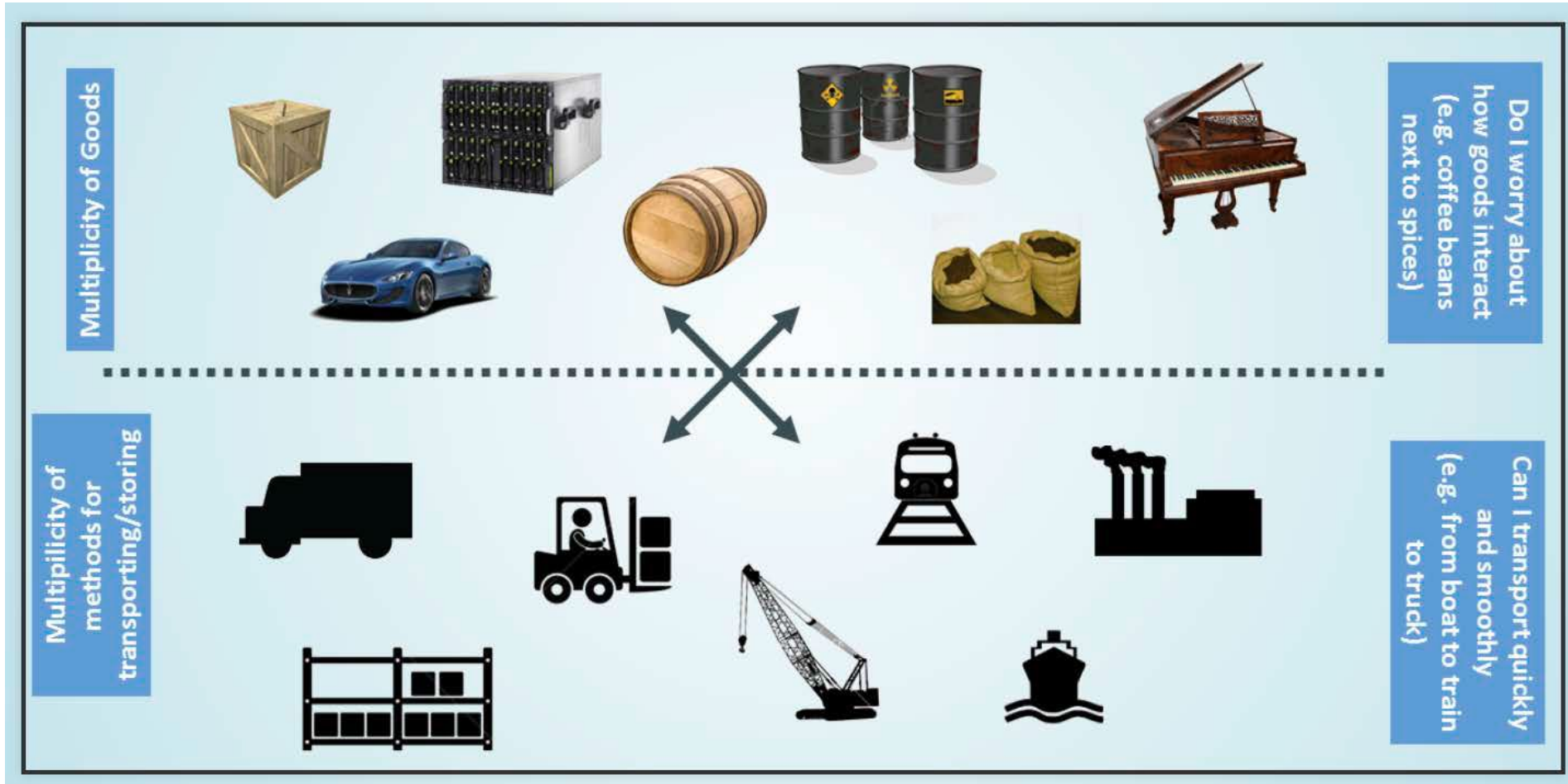


The Matrix of Hell














- You need to work out how to get each stack working on each hardware platform!

	Static website	?	?	?	?	?	?	?
	Web frontend	?	?	?	?	?	?	?
	Background workers	?	?	?	?	?	?	?
	User DB	?	?	?	?	?	?	?
	Analytics DB	?	?	?	?	?	?	?
	Queue	?	?	?	?	?	?	?
		Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
								

Cargo Transport Pre-1960



Also a Matrix of Hell

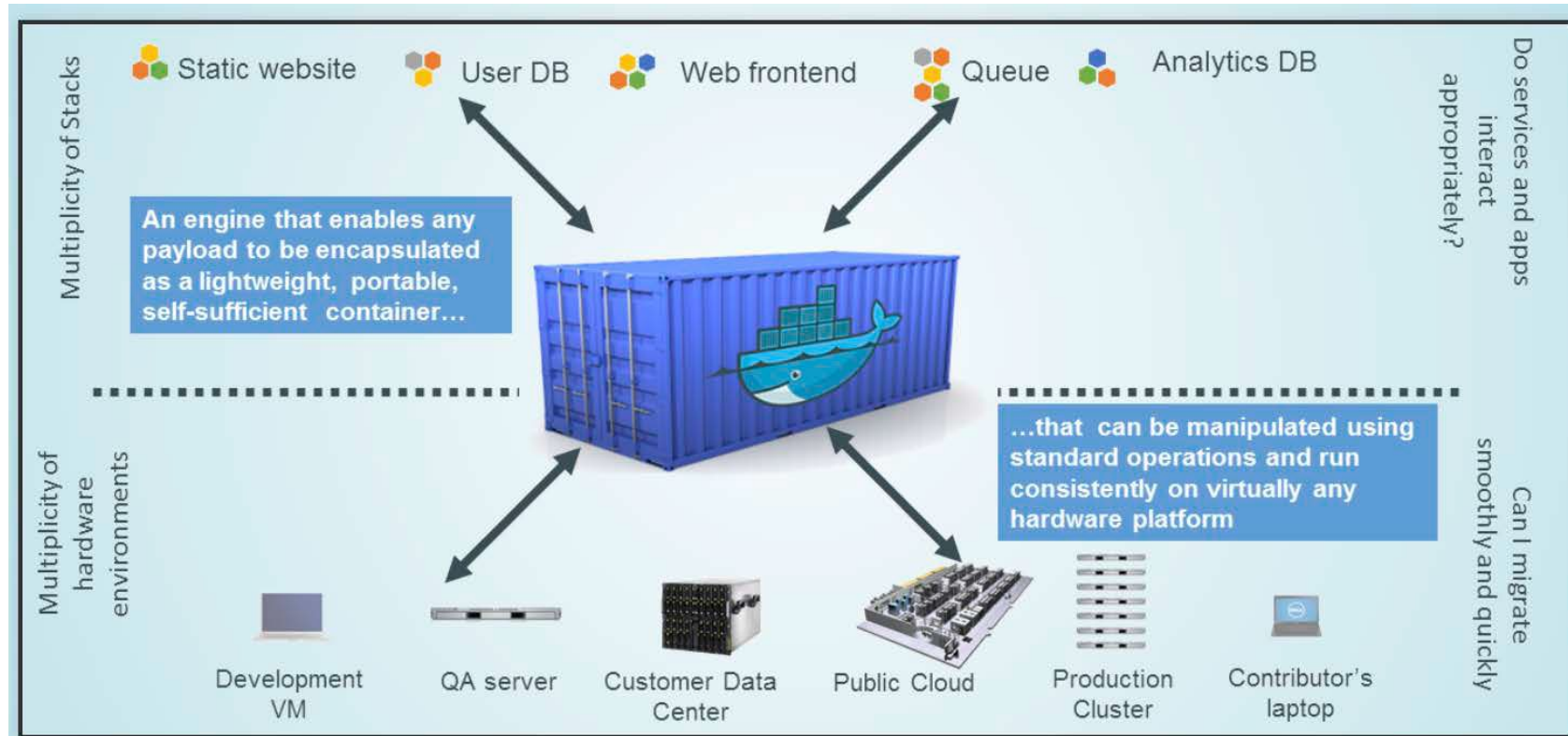
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Solution: Intermodal Shipping Container



Docker is a Container System for code

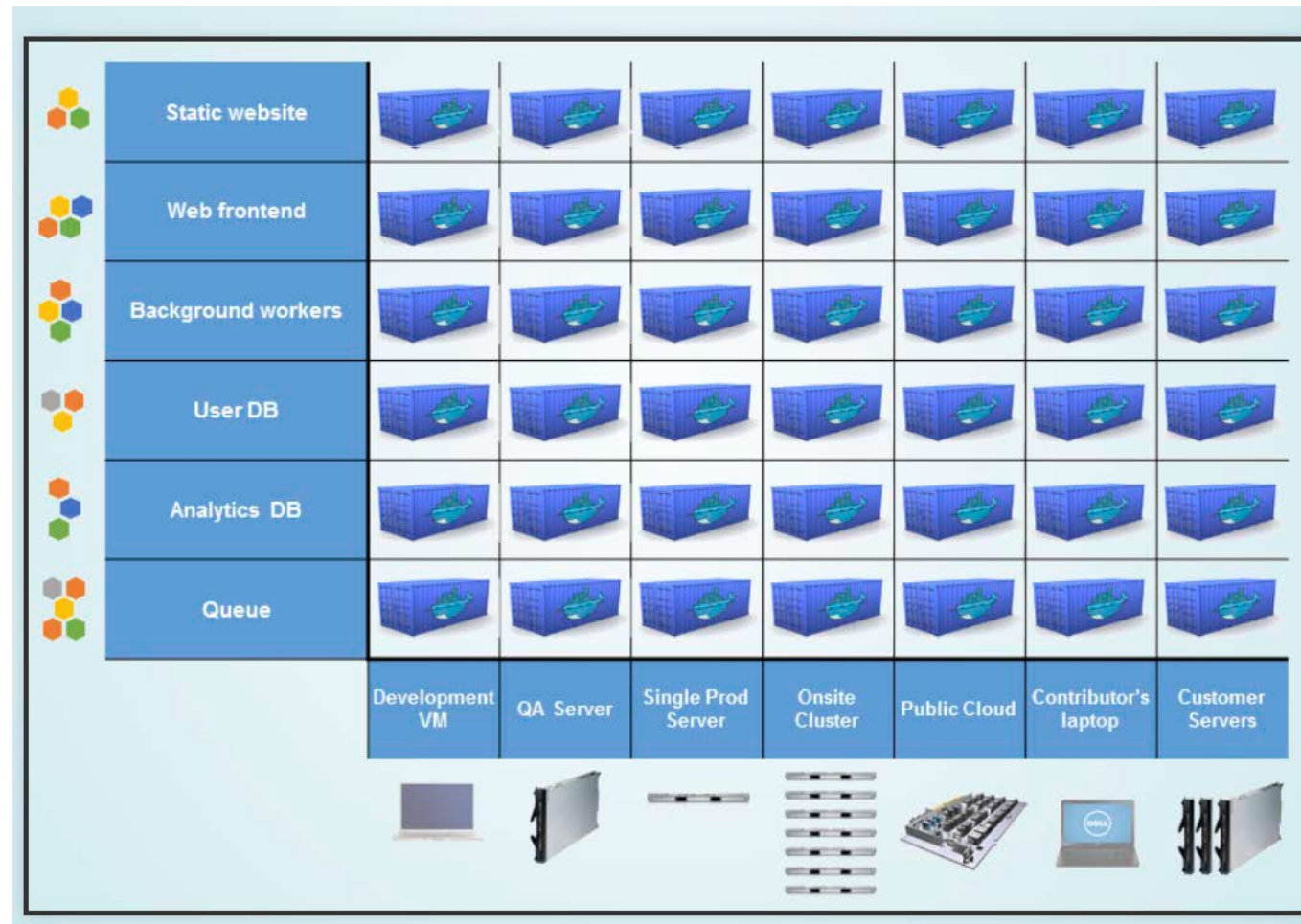
- Configure your docker container with the software you want and then just ship it to any hardware, it should just work!



Deploy to Multiple Environments

- To get complex software to work you need to make sure you have all the correct versions of each dependency.
- Using docker, you can effectively write a script which says exactly what sequences of software packages need to be installed in order to run the package.
- What is really cool is that the same script can be used in a Mac laptop also runs on a windows desktop and on linux virtual machine in Amazon Web Services?
- This will mean I can test everything on my laptop and then give it to Amazon Web Services, I am guaranteed it will work since docker will install the software on the container running in Amazon

Docker Eliminates the Matrix of Disaster



Why Developers Care?

- Build once ... (finally) run anywhere (means an x86 machine running a modern Linux distribution)
 - A clean, safe, hygienic, portable runtime environment for your app.
 - No worries about missing dependencies, packages and other pain points during subsequent deployments.
 - Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying.
 - Automate testing, integration, packaging... anything you can script
 - Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.
 - Cheap containers to deploy services. Instant replay and reset of image snapshots.

Why Administrators Care

- Configure once ... run anything
 - Make the entire lifecycle more efficient, consistent and repeatable
 - Most dockercontainers take less than a second to launch!
 - Eliminate inconsistencies between development, test, production and customer environments.
 - Support segregation of duties
 - Developers focuses on the applications running inside the containers
 - Administrators focuses on managing the containers.
 - Significantly improves the speed and reliability of continuous deployment and continuous integration systems.
 - Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with VMs.

Separation of Concerns

The Developers

- Worries about what's "inside" the container
 - code
 - Libraries
 - Package Manager
 - Apps
 - Data
- All Linux servers look the same

The Sys Admin

- Worries about what's "outside" the container
 - Logging
 - Remote access
 - Monitoring
 - Network config
- All containers start, stop, copy, attach, migrate, etc. the same way

Why are Docker Containers Lightweight?

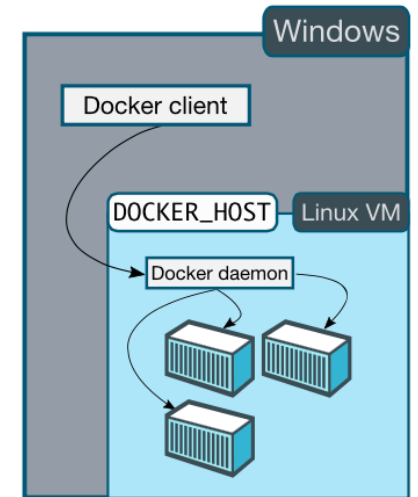
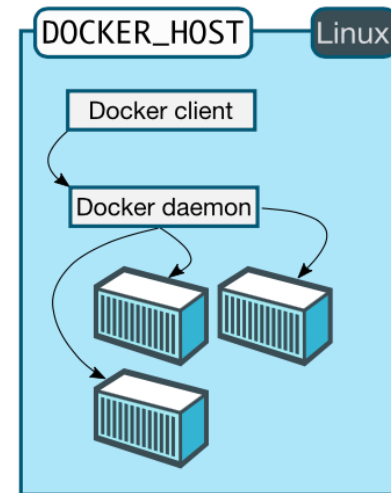
- Lets first look at what is bad about virtual machines
- You will need a new virtual machine (which can be say 10GB in size) whenever you want to do any of the following:
 - Run another instance of the application in a separate virtual machine
 - A slightly modified version of the application running in a separate virtual machine
- What if you use docker?
 - The original unmodified application takes less space since it does not require a new guest operating system
 - If we run a copy of the application in a separate container then it just shares all the libraries and code with the original copy of the application
 - If we run a modified version of the application docker only needs to save the difference between original container and the modified container.

Docker Terminology

- Docker images
 - Docker images contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container.
 - When you run the Docker image, it becomes one instance (or multiple instances) of the container.
- Docker containers
 - Docker containers are the live, running instances of Docker images.
- Docker Hub
 - Docker Hub is the public repository of Docker images that calls itself the “world’s largest library and community for container images.”
- Docker daemon
 - Docker daemon is a service running on your operating systems, such as Microsoft Windows or Apple MacOS or iOS. This service creates and manages your Docker images for you using the commands from the client, acting as the control centre of your Docker implementation.
- Docker registry
 - A Docker registry is a scalable open-source storage and distribution system for docker images.

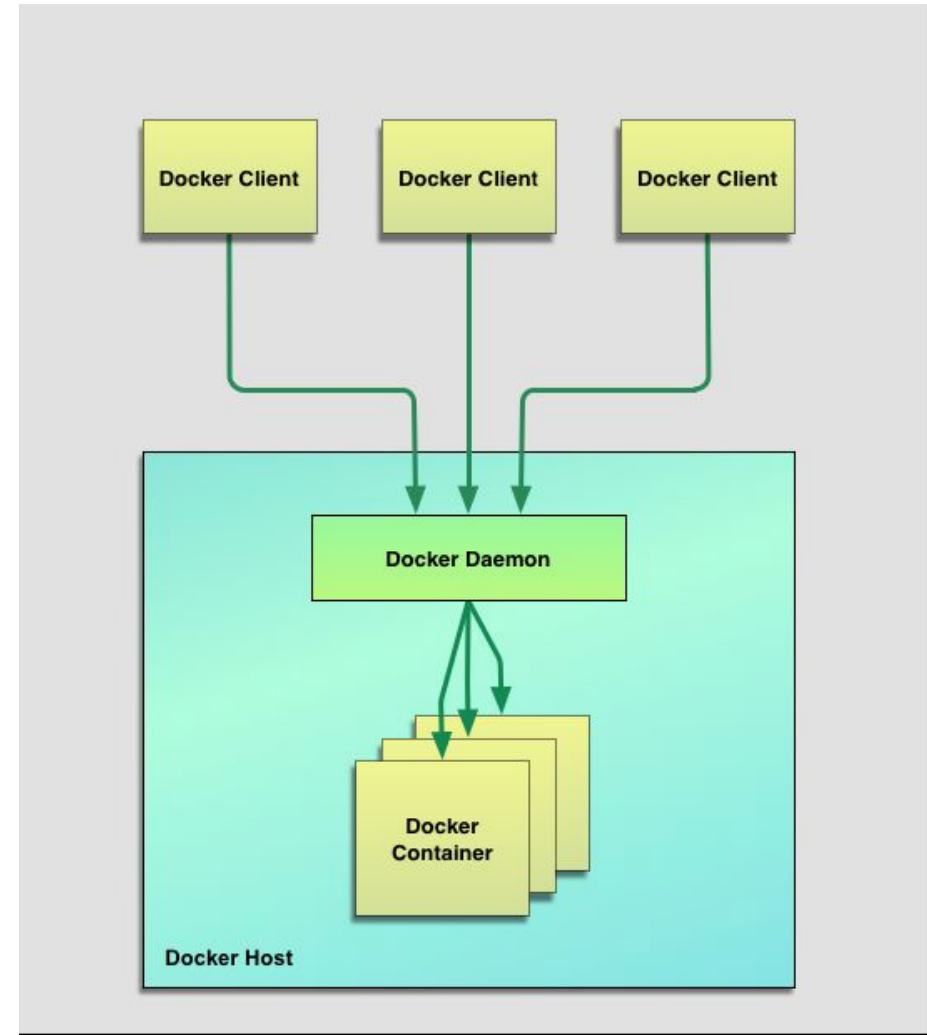
How Docker works?

- Docker runs Linux containers so how does it run in windows or Mac OS X?
- For both windows and Mac OS X Docker first installs a Linux virtual machine via VirtualBox(an open source virtual machine manager)
- Then it installs containers in the Linux virtual machine



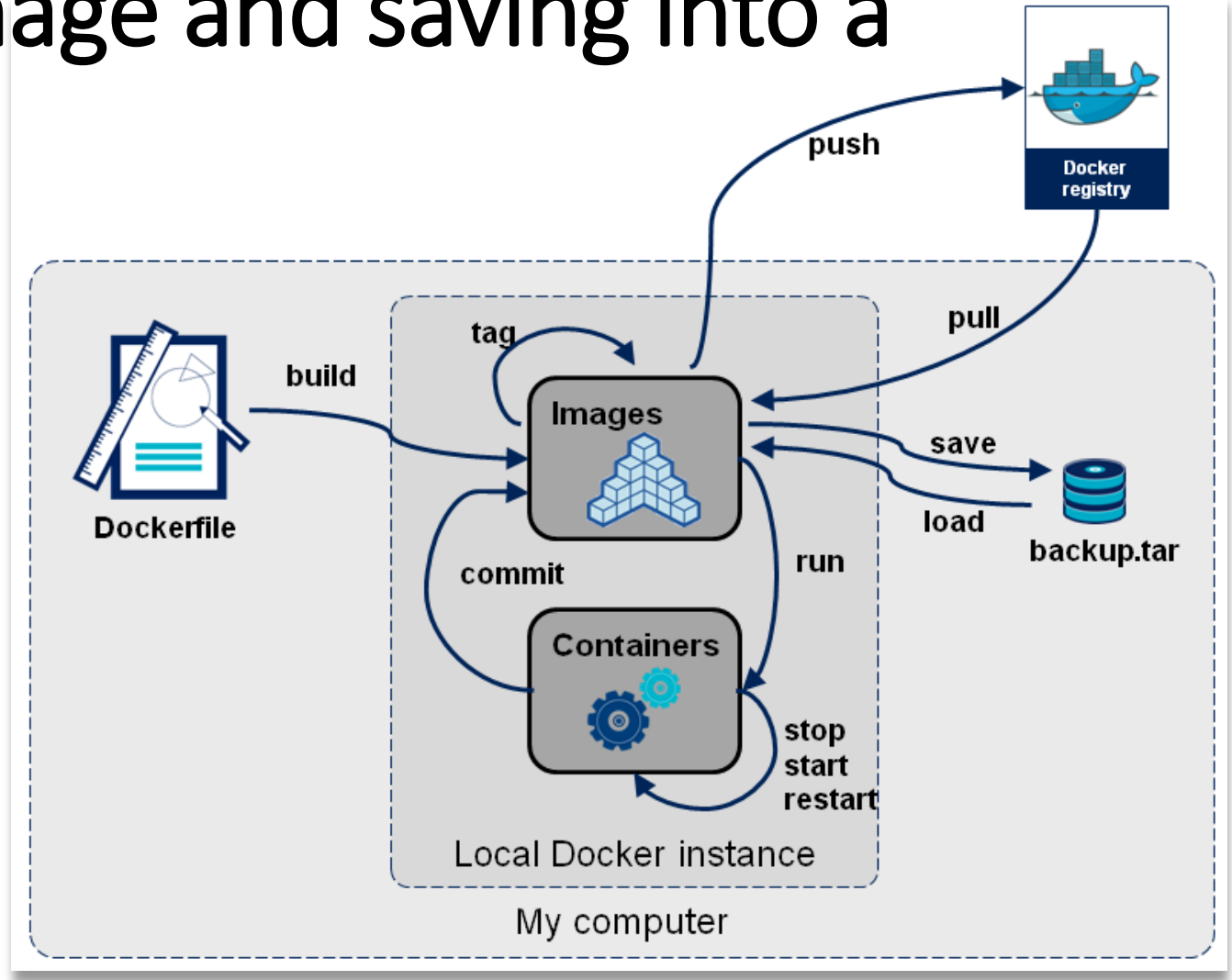
Docker Client and Server

- Docker is a client server application
- The Docker client talks to the Docker server or daemon (sometimes called Docker Engine), which in turn does all the work.
- Docker supports a full RESTful API to interact with the daemon
- You can run the Docker daemon and client on the same host or connect your local Docker client to a remote daemon running on another host.



Building a Docker image and saving into a Docker registry

- The Docker file is like a script that contains a list of instructions for building an Docker image
- The system builds an image by running containers and then committing the changes.
- Once the final image is created it is pushed (uploaded) into a Docker registry.



What does a DockerFile Look Like?

Start with the centos version 6 image from DockerHub

FROM centos:centos6

Enable Extra Packages for Enterprise Linux (EPEL) for CentOS

RUN yum install -y epel-release

Install Node.js and npm

RUN yum install -y nodejs npm

Install app dependencies

COPY package.json /src/package.json

RUN cd /src; npm install

Copy the source files from current directory into the src directory in the container

COPY . /src

Expose the HTTP port 8080 to the outside.

EXPOSE 8080

When the container runs it will run the command node with one argument of /src/index.js

CMD ["node", "/src/index.js"]

Docker File Commands

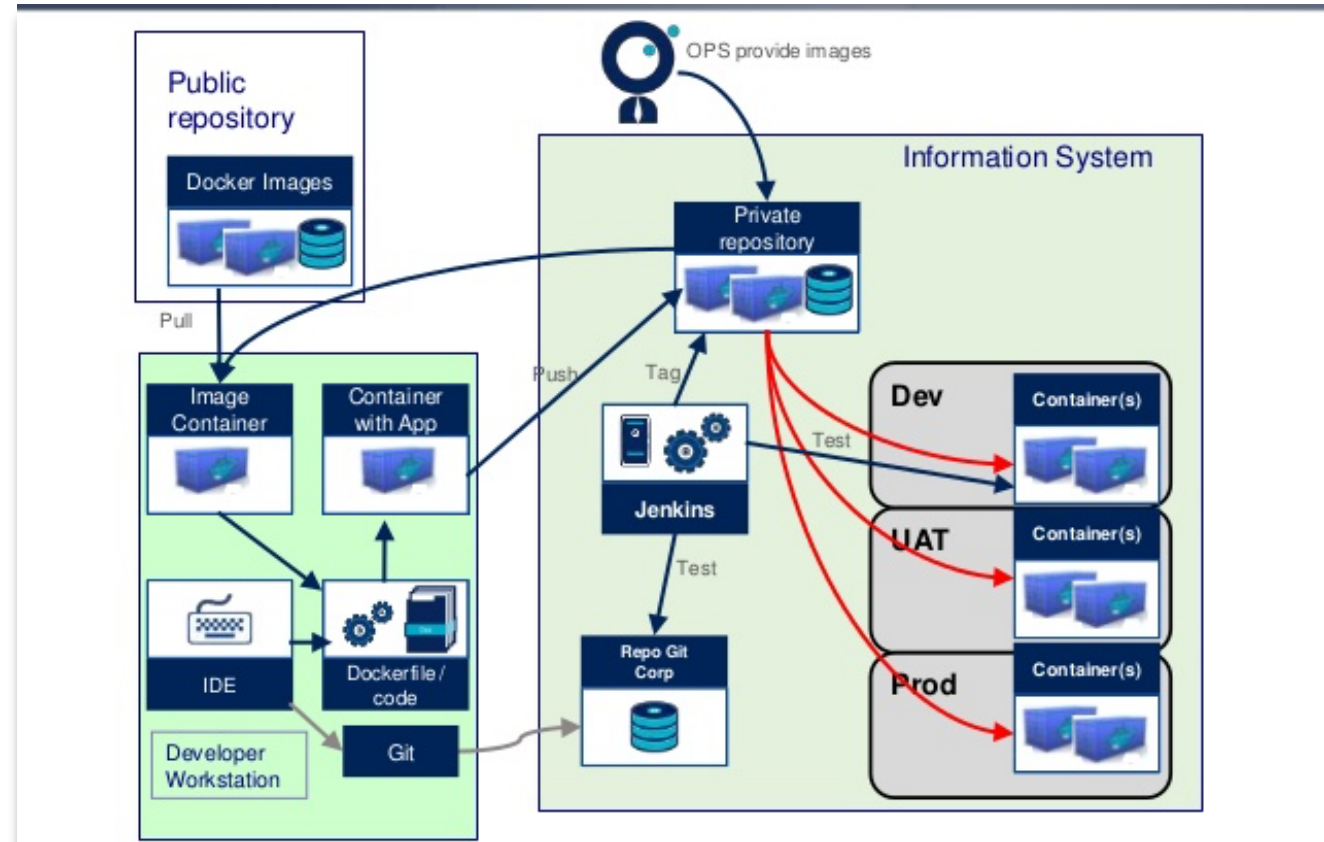
- There are many different commands that you can use in the DockerFile. These include:
 - ADD
 - COPY
 - ENV
 - EXPOSE
 - LABEL
 - USER
 - WORKDIR
 - VOLUME
 - STOPSIGNAL
- See the following reference to learn what they do
<https://docs.docker.com/engine/reference/builder/>

Docker Images

- Images are the building blocks of the Docker world.
- You launch your containers from images
- Images are the “build” part of Docker’s life cycle.
- Building an image involves executing a series of instruction.
 - For example
 - Add a file
 - Run a command
 - Open a port
- Consider images as the “source code” for your containers.
- They are highly portable and can be shared, stored, and updated.

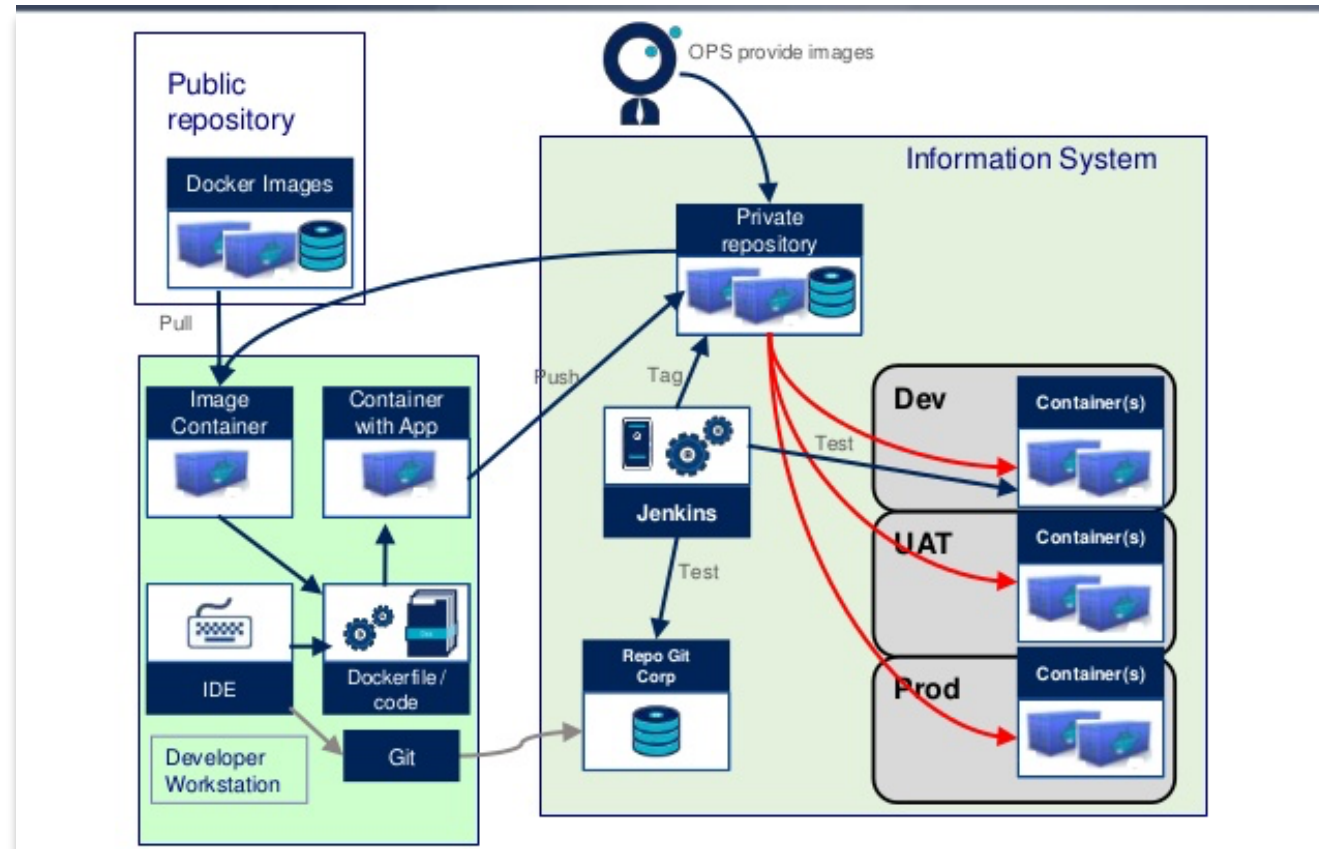
Docker Registries/Repositories

- Docker stores the images you build in registries (repositories)
- There is a public repository called Docker Hub
 - Contains more than 100, 000 pre-built images!
 - Contains images like Linux Ubuntu, Nodejs, PostgreSQL, MySQL, etc.
- For example to get MySQL just use the following command
 - `dockerpull mysql`

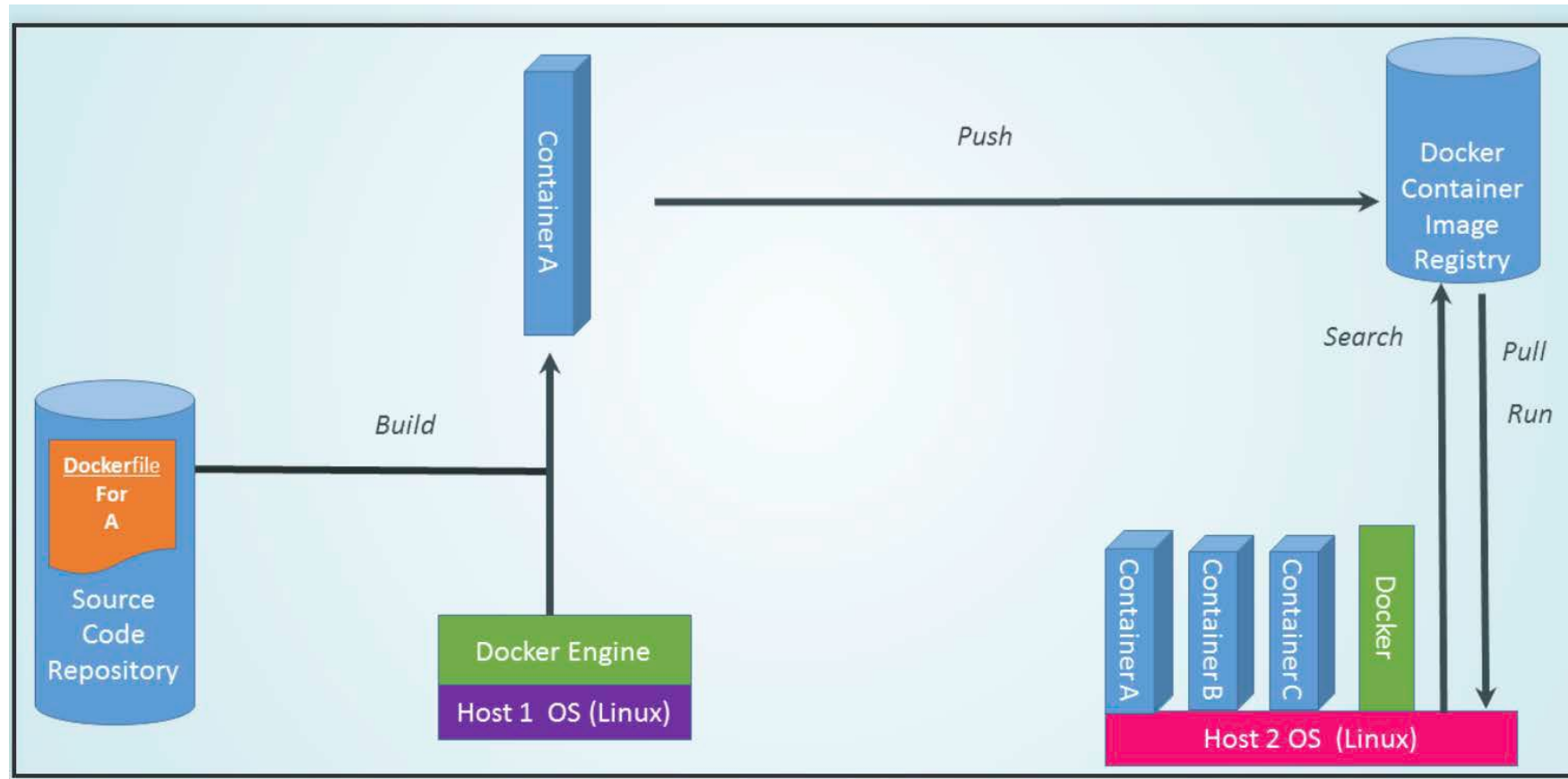


Docker Registries/Repositories

- You can also create a private repository, where you store your private Docker images.
 - In the labs we will show you how to do this via Amazon's EC2 Container Registry (ECR)
 - <https://aws.amazon.com/ecr/>
- You can then pull the Docker image into the different environments, such as development (Dev), testing (UAT) and production (Prod)



What are the basics of a Docker System?



The above shows creating a Docker image on Host 1 operating system and then pushing it into a registry and then pulling the image into Host 2 operating system.

Containers

- Containers are launched from images and can contain one or more running processes
- Images are like the building or packing aspect of Docker
- Containers are the running or execution aspect of Docker
- A Docker container is:
 - An image format
 - A set of standard operations
 - An execution environment
- Docker doesn't care about the contents of the container when performing actions.
 - For example whether a container is a web server, a database or an application server. Each container is loaded the same as any other container.

Docker encourages service and micro services oriented architecture

- Docker recommends that each container run a single application or process
- This promotes a distributed application model where an application or service is represented by a series of inter-connected containers
- For example the database can be one service running in one container whereas the web server can be a different service running on a different container running on a different machine.
- This makes it easy to scale your application!

Docker Encourages service and micro services oriented architecture

- Note you can run more than one application within a single container, although this is not recommended in general.
- Using docker, we can quickly build an application server, a message bus, a utility appliance, a continuous integration test bed for an application, or one of a thousand other possible applications, services, and tools.
- How to create multiple docker containers at once?
 - Use docker compose

Create a docker-compose.yml configuration file

```
web:
  build: .
  command: python app.py
  ports:
    - "5000:5000"
  volumes:
    - ./code
  links:
    - mysql
mysql :
  image: mysql
```

Web server

MySQL
Database

To start the containers just type in the following:

docker-compose up

Checking running containers

```
[AKIRA@^_^:compose]$ docker-compose ps
```

Name	Command	State	Ports
compose_redis_1	/entrypoint.sh redis-server	Up	6379/tcp
compose_web_1	python app.py	Up	0.0.0.0:5000->5000/tcp

Stop containers

```
[AKIRA@^_^:compose]$ docker-compose stop
```

```
Stopping compose_web_1... done
```

```
Stopping compose_redis_1... done
```

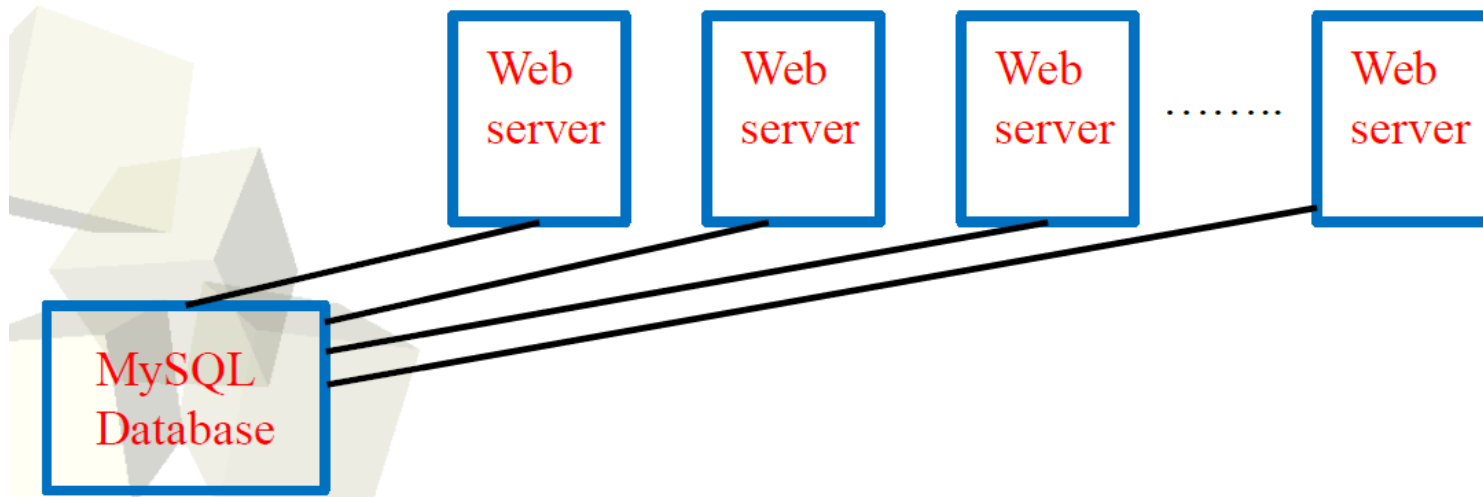
Remove containers

```
$ docker-compose rm
```

<https://www.penflip.com/akira.ohio/appcatalyst-hands-on-lab-en/blob/master/docker-compose.txt>

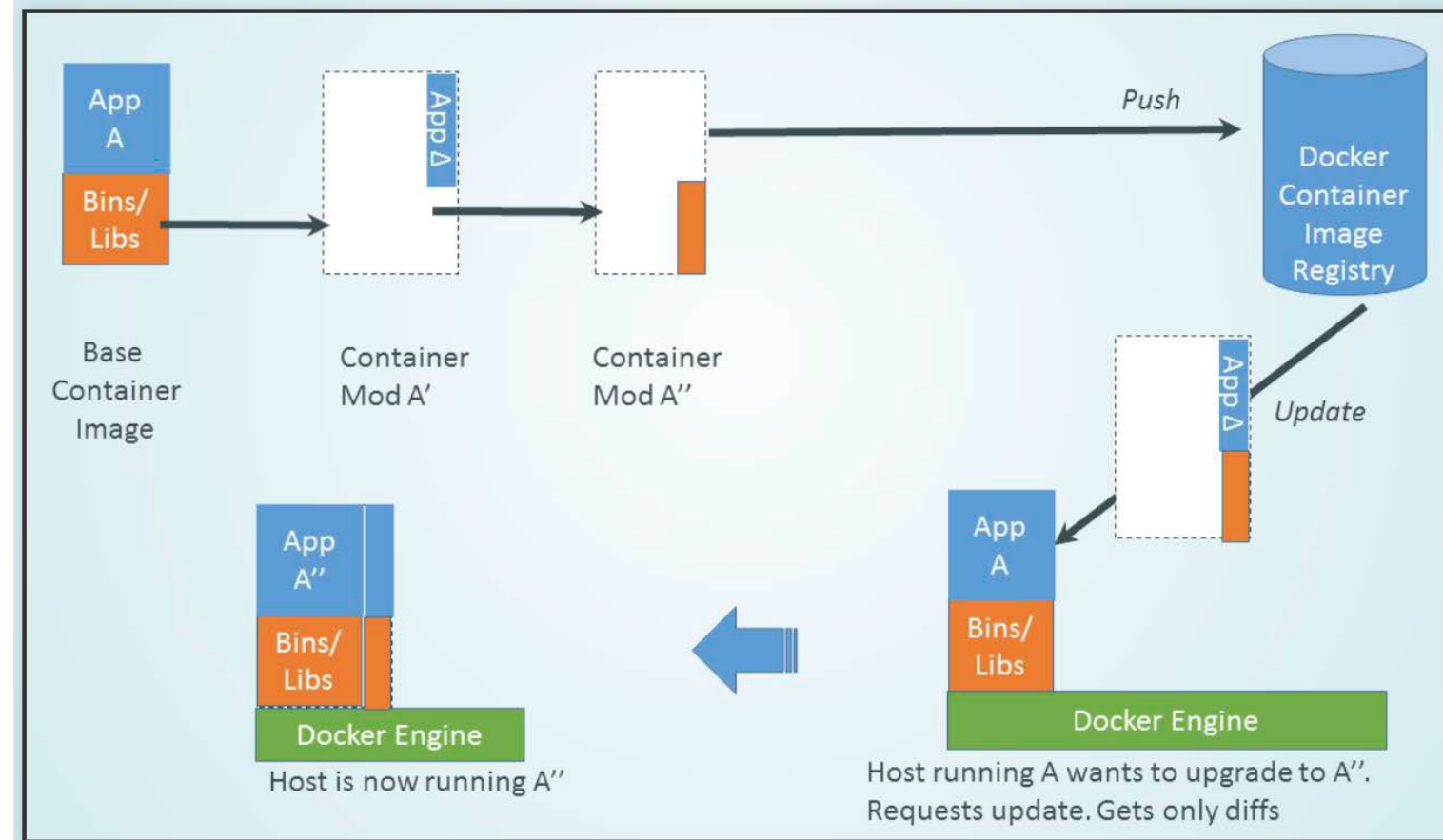
Scale Out

```
[AKIRA@^_^:compose]$ docker-compose scale web=10  
Creating and starting 3... done  
Creating and starting 4... done  
Creating and starting 5... done  
Creating and starting 6... done  
Creating and starting 7... done  
Creating and starting 8... done  
Creating and starting 9... done  
Creating and starting 10... done
```



Changes and Updates

- As we can see at the top we just need to push the updated part (orange part) of the image being run by the container to the Docker Registry. Then the remote site can just apply that update to its previous copy of the image (blue part).



List Docker Images

```
bash-3.2$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
fortunecookie_web	latest	539113d4e5a7	23 hours ago	653.1 MB
<none>	<none>	fff695d3e071	24 hours ago	653.1 MB
<none>	<none>	40a43bf23bc0	5 days ago	653.1 MB
<none>	<none>	62ad1f3f7c67	5 days ago	653.1 MB
<none>	<none>	c16cc2ea52eb	5 days ago	653.1 MB
<none>	<none>	69c868d9a9de	5 days ago	653.1 MB
<none>	<none>	e12161004072	5 days ago	653.1 MB
<none>	<none>	675cd58df58b	5 days ago	642.3 MB
<none>	<none>	25df1a2dd1ad	5 days ago	653.1 MB

- The command `docker image` lists the details of the images
 - Note the virtual size is the size of the image incorporating all layers. However, many of the images share layers hence the actually total size occupied by all layers is much smaller than the total virtual size of all the layers.

```
bash-3.2$ docker history fortunecookie_web
```

IMAGE	CREATED	CREATED BY	SIZE
539113d4e5a7	23 hours ago	/bin/sh -c #(nop) CMD ["nodemon" "-L" "-x" "n	0 B
90b9a846f1ac	23 hours ago	/bin/sh -c apt-get update && apt-get install	3.13 MB
42f903782e6a	23 hours ago	/bin/sh -c #(nop) ENV PATH=/usr/local/lib/nod	0 B
0f10d03e9201	23 hours ago	/bin/sh -c #(nop) ENV NODE_PATH=/usr/local/li	0 B
a0ca61d865fb	23 hours ago	/bin/sh -c npm install -g	7.663 MB
4b72be3a36e8	23 hours ago	/bin/sh -c #(nop) COPY file:951e3bd4ab4ef1734	127 B
35721a9acf9c	23 hours ago	/bin/sh -c #(nop) EXPOSE 3000/tcp	0 B
944036b7d4df	23 hours ago	/bin/sh -c #(nop) COPY dir:1553191a1fb5811a02	19.23 kB
8e9fb7054b6b	6 days ago	/bin/sh -c #(nop) WORKDIR /app	0 B

- The command `docker history <repository name or image ID>` shows you all the layers installed for that image.
- Note above `fortunecookie_web` image only had a few small layers added (Created 23 hours ago) on top of the an image that was created 6 days ago.

Other Docker Commands

- `docker pull ubuntu:12.04`
- This command pulls Ubuntu 12.04 image from the **public (from Docker Hub)** ubuntu repository
 - The `ubuntu` part is the name of the image and the `12.04` is the name of the tag.
- `docker pull zhen/puppetmaster`
- This pulls the puppetmaster repository from the **user (from Docker Hub)** repository with username zhen
- `docker run -t -i --name new_container ubuntu:12.04 /bin/bash`
- This launches a container from the ubuntu:12.04 image, which is an Ubuntu 12.04 operating system
- `docker run -t -i --name web -v /src/webapp:opt/webapp python app.py`
- The above mounts the host directory `/src/webapp` to the container directory
 - `/opt/webapp` in the container.
- `docker search puppet`
 - searches for a public available image on Docker Hub
- `docker build .`
- This will build a docker image using the Dockerfile inside the current directory.

Why we need

Virtual Machine

- You need desktop-based applications where GUI is the most important aspect
- You need total isolation from host
- You need specific OS for your work
- You want to reserve the hardware for the virtualisation

Containers

- You want service-based applications without hassle of installation of the OS and Apps
- You want to have easy multiplexing the services
- You want to share hardware resources to all available services

We don't need

Virtual Machine

- Host resources are limited
- The applications are service-based without desktop GUI
- Total isolation is not required
- Shareable hardware resources between services and host OS

Container

- You need to run/install a desktop-based application where you need to interact with the GUI
- Your machine has no OS in it.
 - AWS, GCP or Azure offer virtualisation as VM with built-in OS
- You need a complete isolation environment from host OS

Conclusion

- Virtual machines are really useful for cloud computing.
- Container technologies like docker make software deployment into different environments even easier.
- Containers are much more light weight compared to virtual machines.