



Explore open source tools for Containerization

Envisioning Application Modernization with Containers
By Omprakash Pandey

Microsoft
Partner

Silver Learning
Gold Cloud Platform



Educate

Advise

Implement

Manage

Agenda

- Module 1: Introduction to Containers
- Module 2: Container Architecture
- Module 3: Docker Architecture
- Module 4: Application Modernization
- Module 5: Understanding Orchestration





Module 1

Introduction to Containers

Educate

Advise

Implement

Manage

Application packaging strategies

- Every Application has certain dependencies.
 - Libraries provided by Operating System
 - Libraries provided by runtime environment like Java, Python & Dot Net runtime
 - Server Runtime like Tomcat for Java, IIS for Asp.net, Apache httpd
 - Third Party Libraries
- Change in Dependencies affects Application.



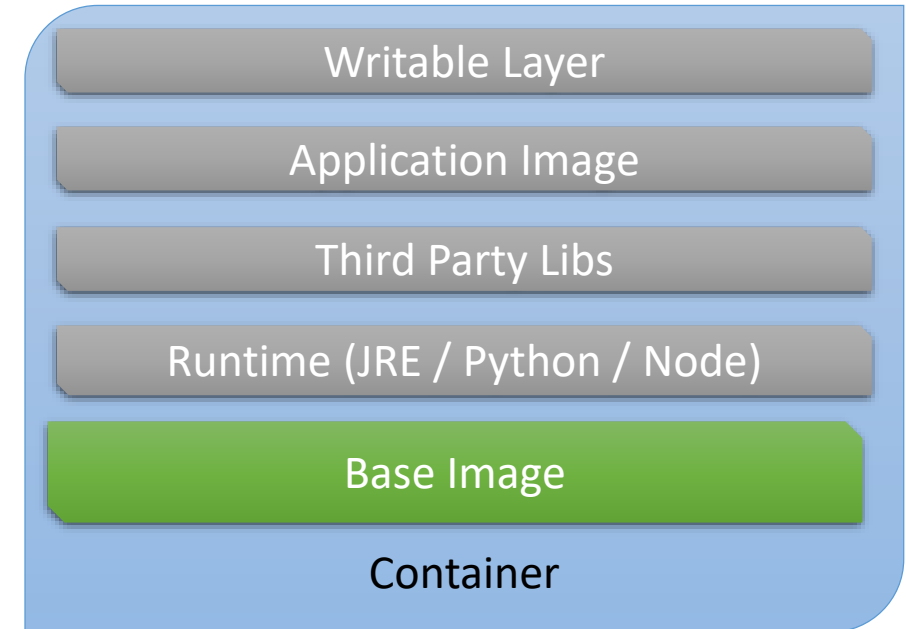
Application packaging strategies / challenges

- Application goes through following phases:
 - Development
 - Testing
 - Staging
 - Production
- Managing dependencies across all these environments could be challenging.
- Creating compatible dev-test environment may take considerable time.
- Question : Challenges in moving application between environments
- Question: List Existing solutions based on Virtualization



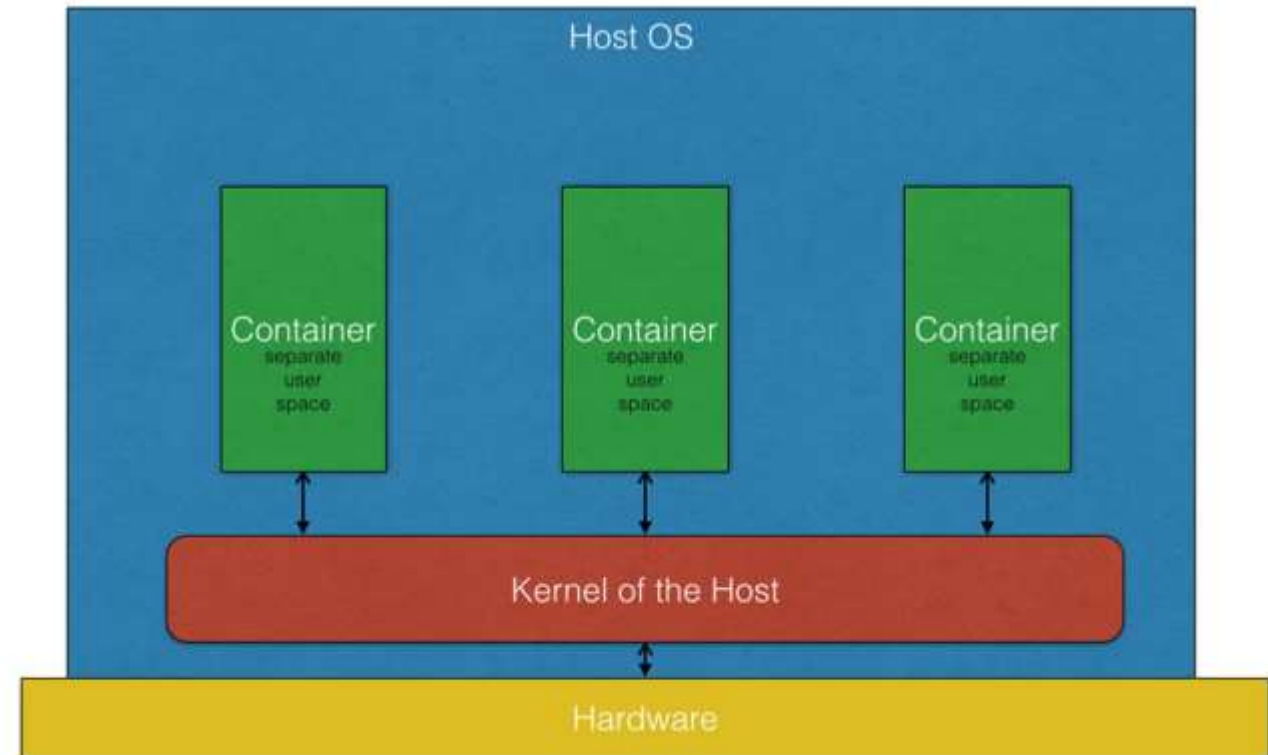
Containers as alternative

- A Typical container would have application & all its dependencies packed together.
- This makes container “decoupled” from targeted environment.
- Only restriction being Windows or Linux platform.

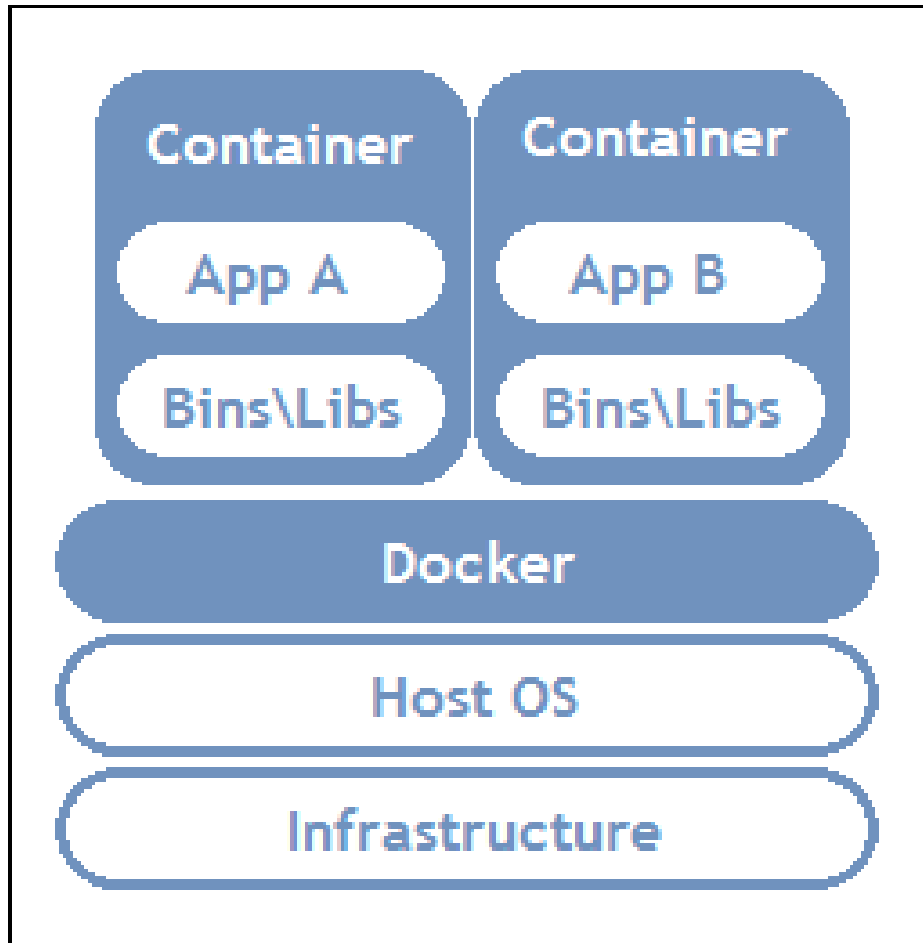


Benefits of Containers

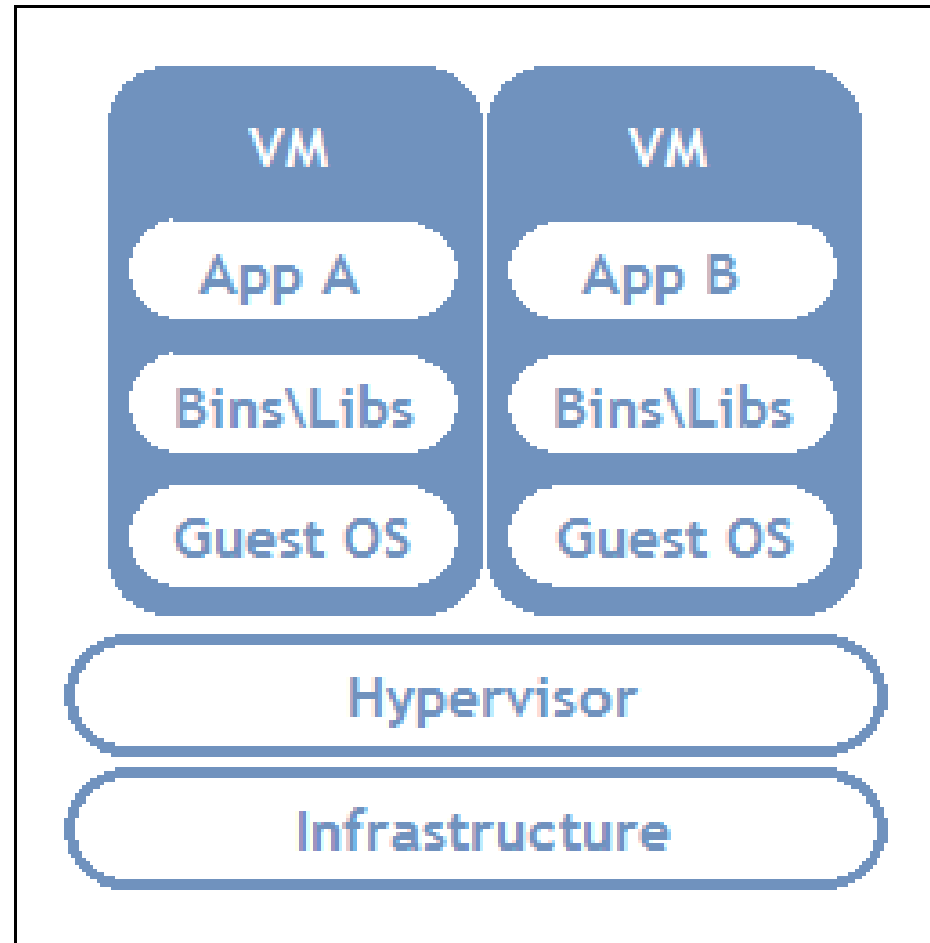
- Isolate application from underlying OS and Hardware.
- Lightweight, provides higher density than Virtual machines.
- Dev-Ops ready. Many CI/CD platforms support container deployments/build.
- Every container executes in a separate user space.



Container Based Implementation



Virtual Machine Implementation



Container vs Virtual machines

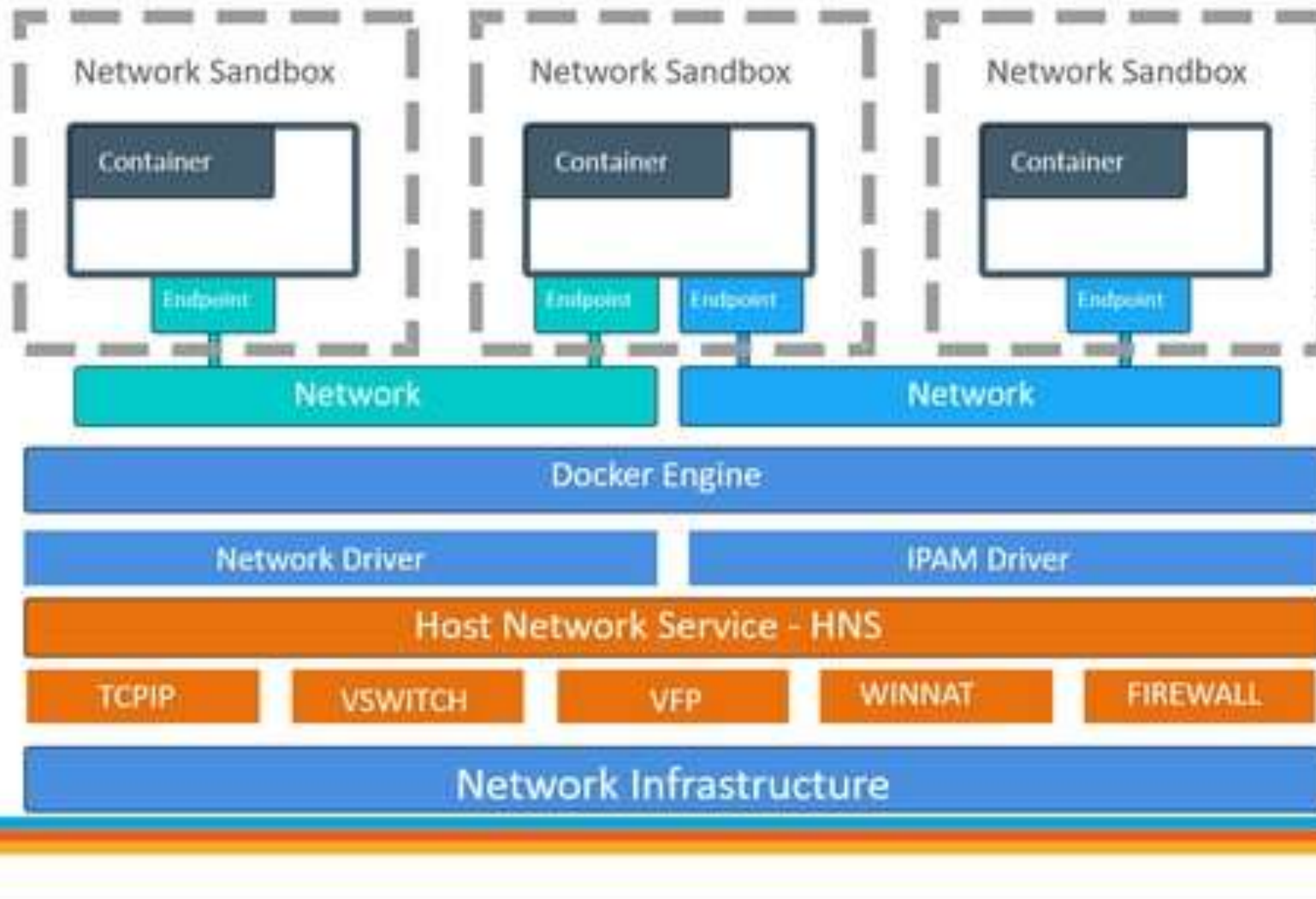
- Virtual machines provides hardware level virtualization.
- Virtual machines contains Operating system.
- Virtual machines use guest OS kernel.
- Host OS and Guest OS could be completely different.
- Booting up VM is equivalent of booting a physical machine.
- Containers provide software level isolation, no hardware virtualization.
- Containers use “base image” which provide basic OS level libraries but not the OS.
- Containers use host OS kernel.
- Due to dependence on Host OS Kernel, containers must be deployed on compatible host OS.
- Launching container is equivalent of launching an application.

Container platforms

- Docker from Docker Inc
- LXC from Linux
- LXD from Ubuntu
- RKT from CoreOS



Container Networking Model





Module 2

Container Architecture

Educate

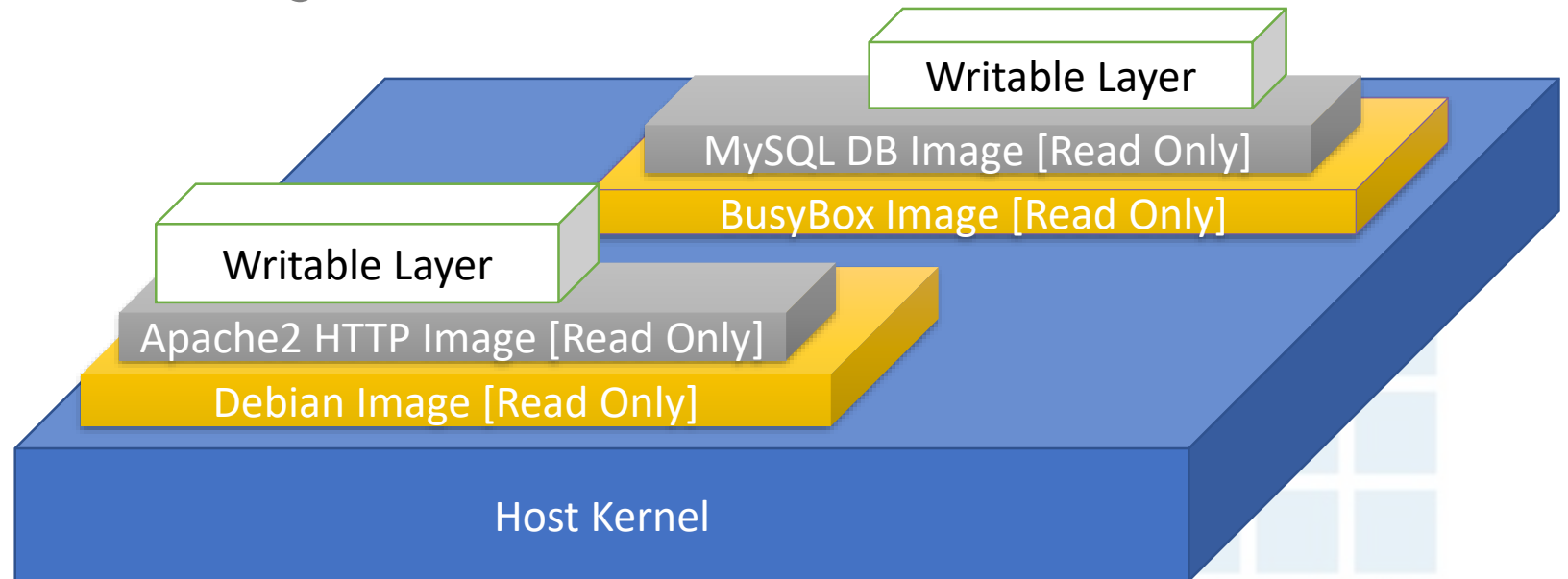
Advise

Implement

Manage

Container & Images

- Containers are running instances of image
- Image is a read-only filesystem made of “layers”
- Each “layer” is an image which may contain a specific application, library or runtime.
- Container has an extra “Writable” layer.
- The bottom-most image is “base” image.



Read-Only vs Writable layers

- Read-only layers are shared by multiple containers / images.
- Writable layer allows read and write operations
- Every container has ONE writable layer.
- Every layer has unique ID and meta-data.

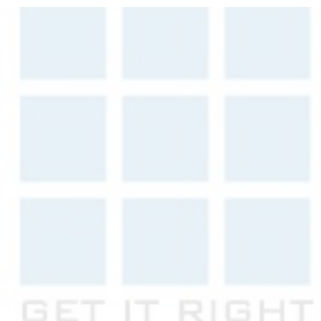
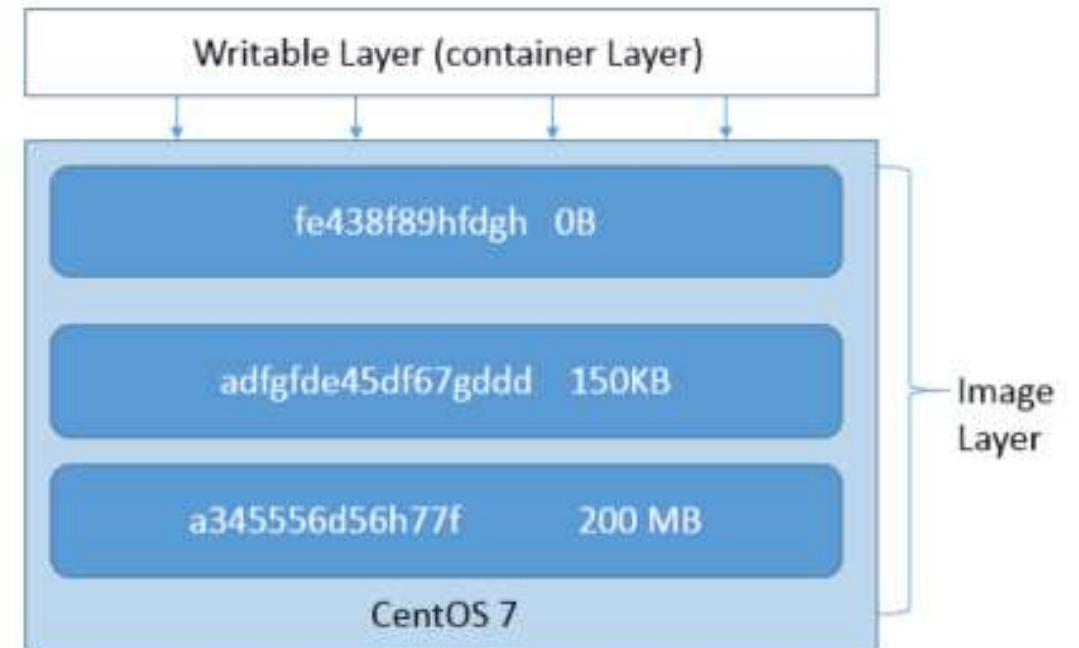
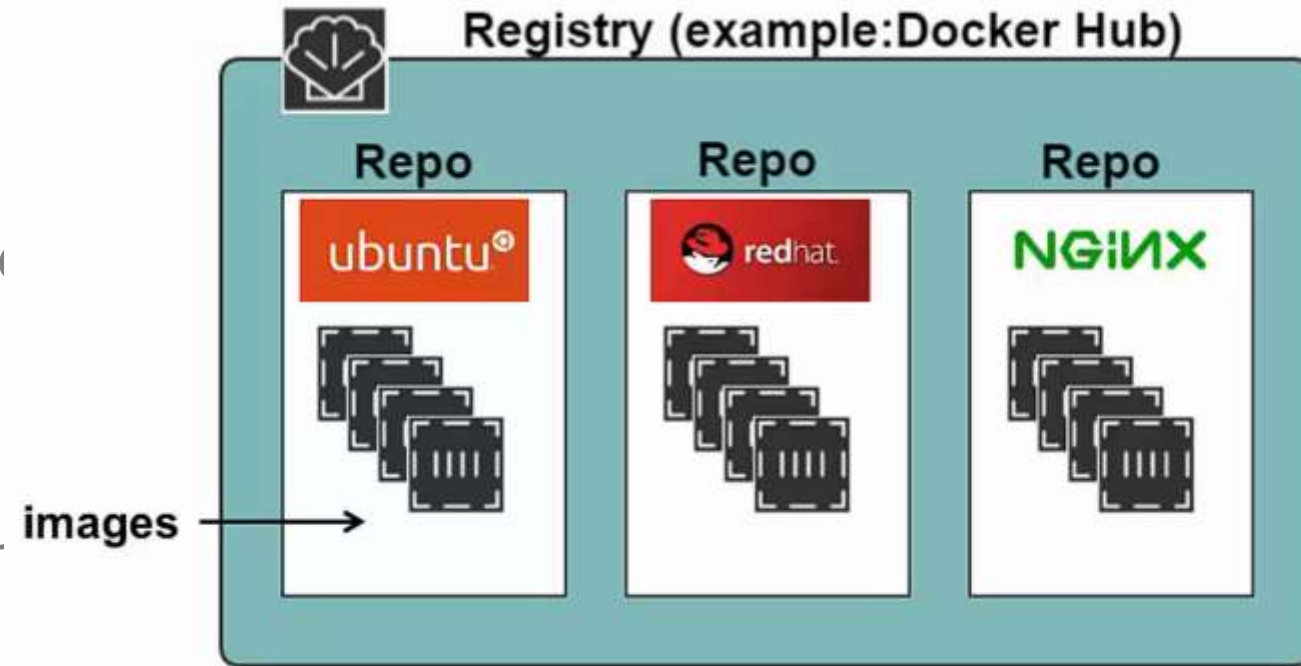


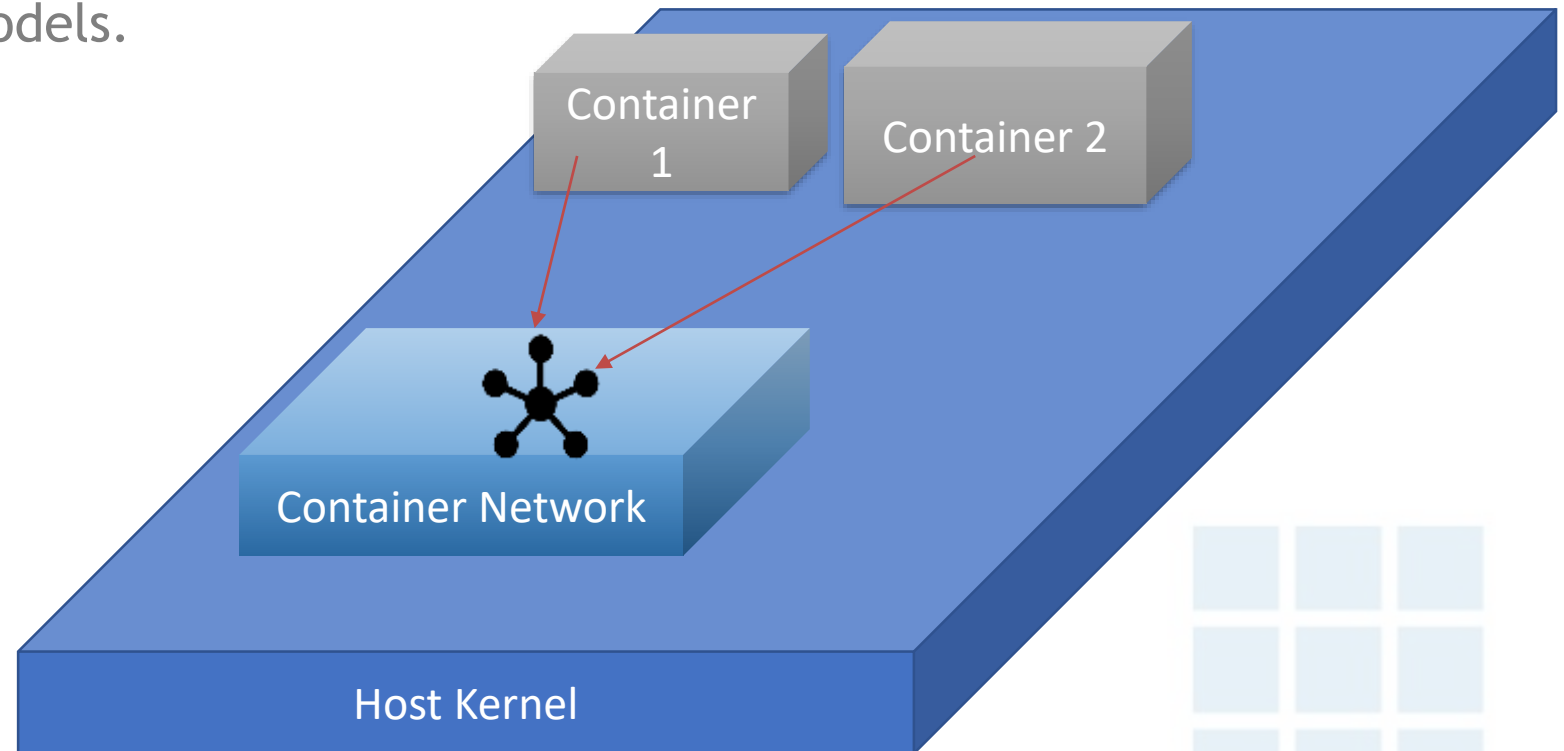
Image repositories

- An Image repository is persistent storage for container images
- A private repository can be accessed by selected users.
- A public repository can be accessed by any one.
- A Developer may “Push” application container image.
- An operations team may “Pull” the same image for deployment.
- Many cloud platforms do provide “Hosted” container repositories.
- Host repositories are managed by vendor.



Container Networking

- Containers can communicate through a container network.
- Container platform like docker have
Multiple network models.



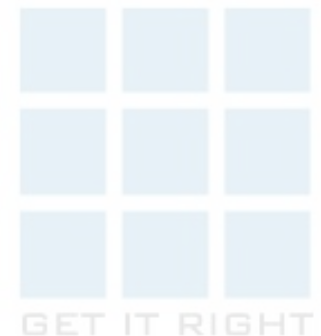
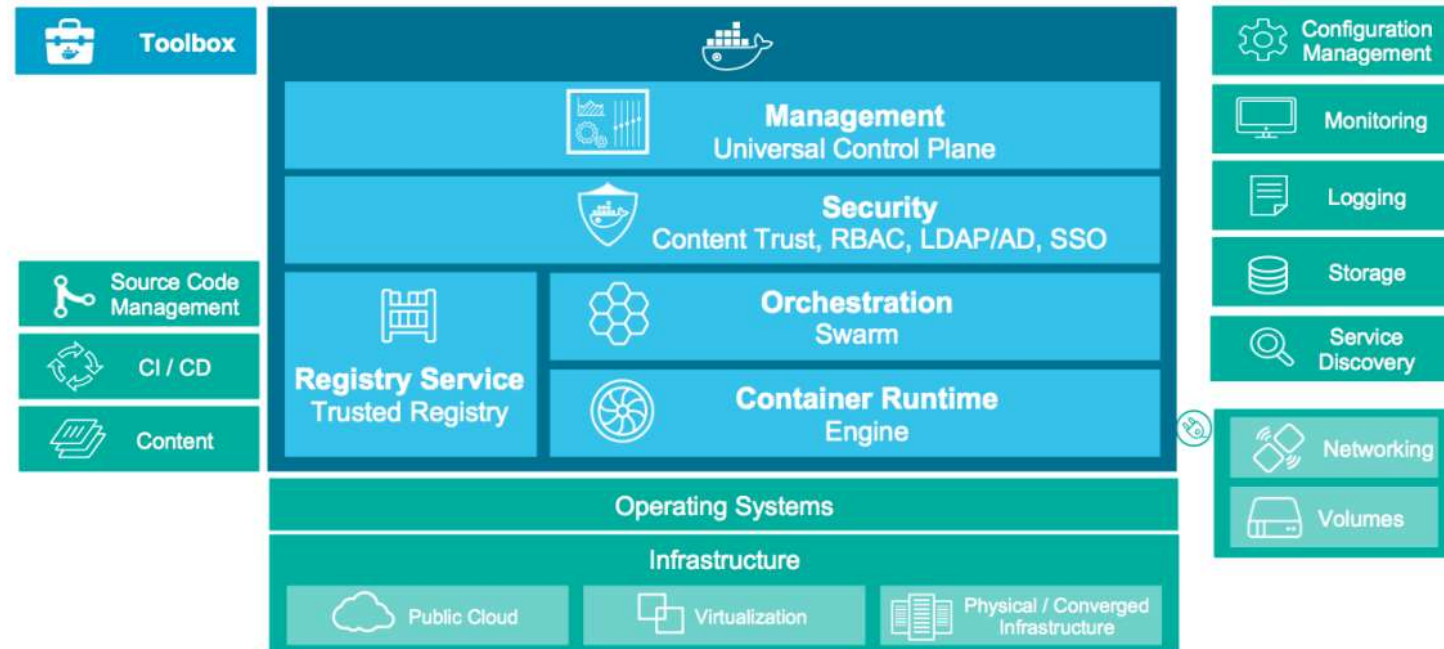


Module 03

Docker Architecture

Docker As a Container platform

- An open platform for developing, shipping, and running applications.
- Started its life as a PaaS provider by name dotCloud.
- Since its inception, working with linux container.
- The container platform was named “Docker”
- In 2013, got “rebranded” as Docker Inc.
- A container platform based on “runc” and “libcontainer”



Docker Editions

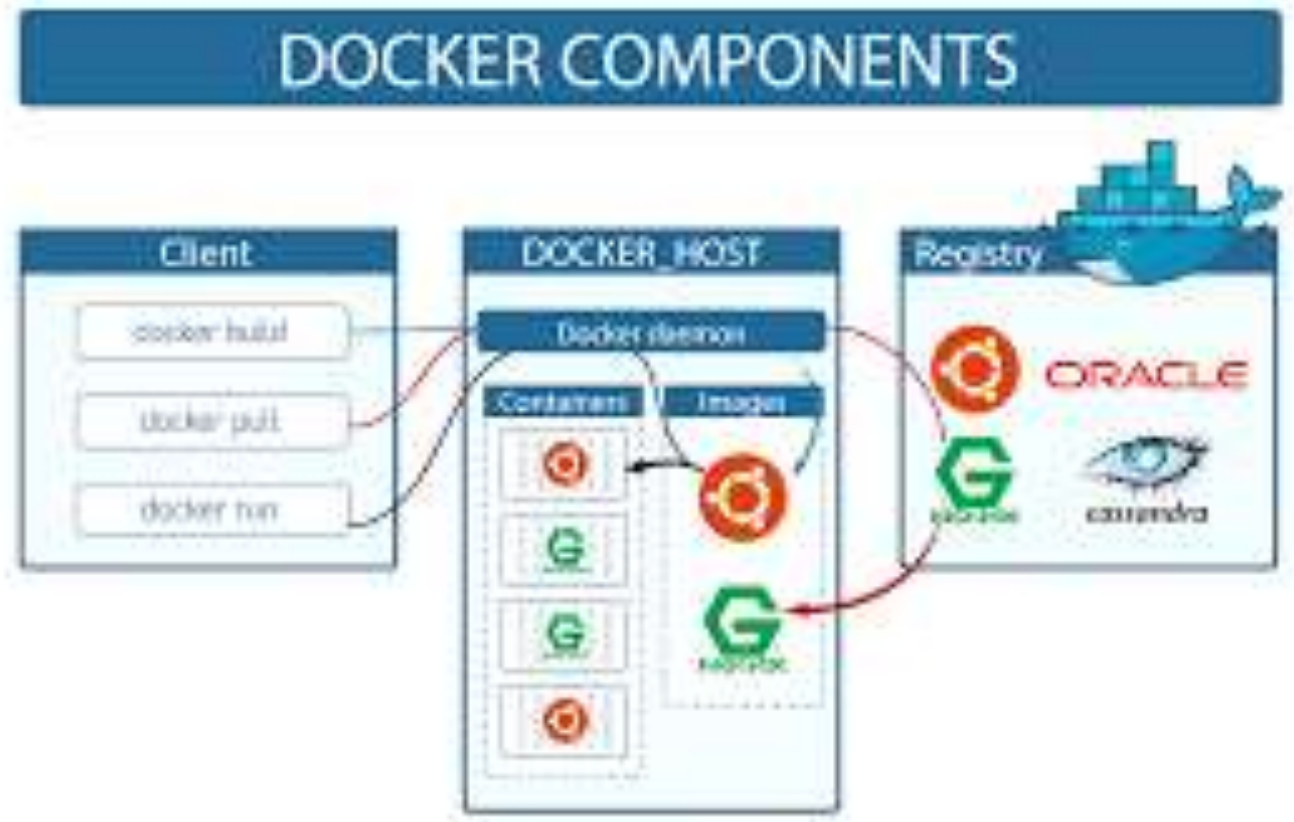
- Community edition (Docker CE)
 - Ideal for developers and small teams experimenting with containers.
- Enterprise Edition (Docker EE)
 - Enterprise development & IT team for business critical applications at production scale.

Capabilities	Community Edition	Enterprise Edition Basic	Enterprise Edition Standard	Enterprise Edition Advanced
Container engine and built in orchestration, networking, security	✓	✓	✓	✓
Certified infrastructure, plugins and ISV containers		✓	✓	✓
Image management			✓	✓
Container app management			✓	✓
Image security scanning				✓

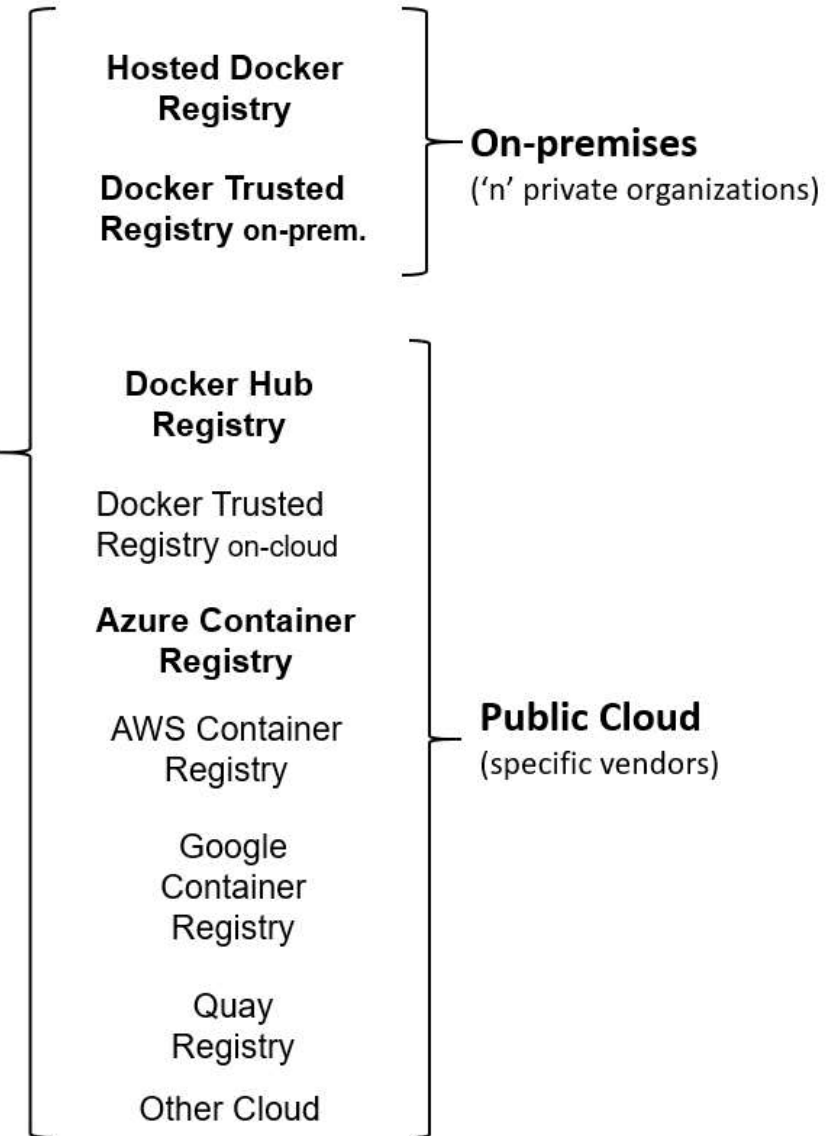
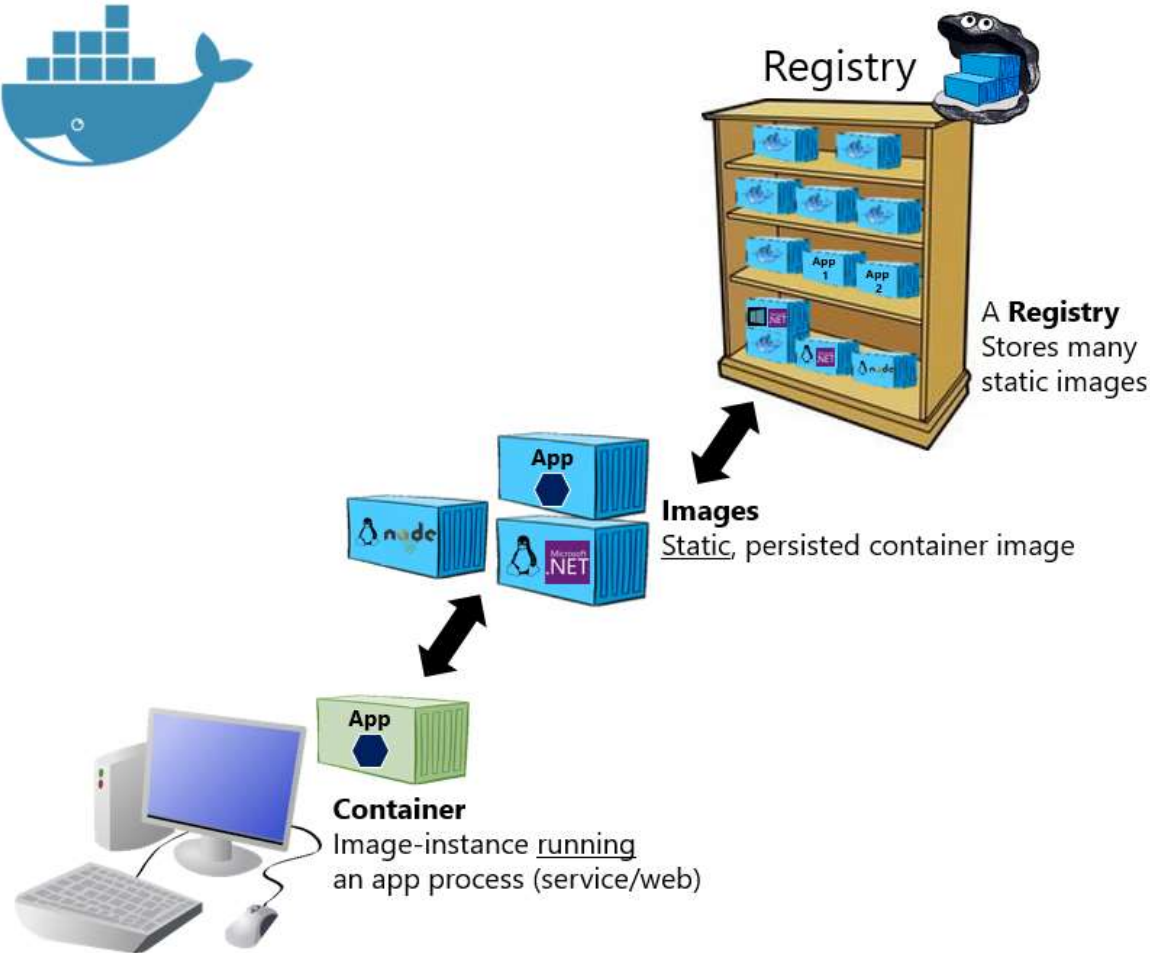
[Educate](#)[Advise](#)[Implement](#)[Manage](#)

Docker components

- Docker engine (docker daemon)
- Docker container images
- Docker containers
- Docker CLI (client)
- Docker (REST) API
- Docker Registry



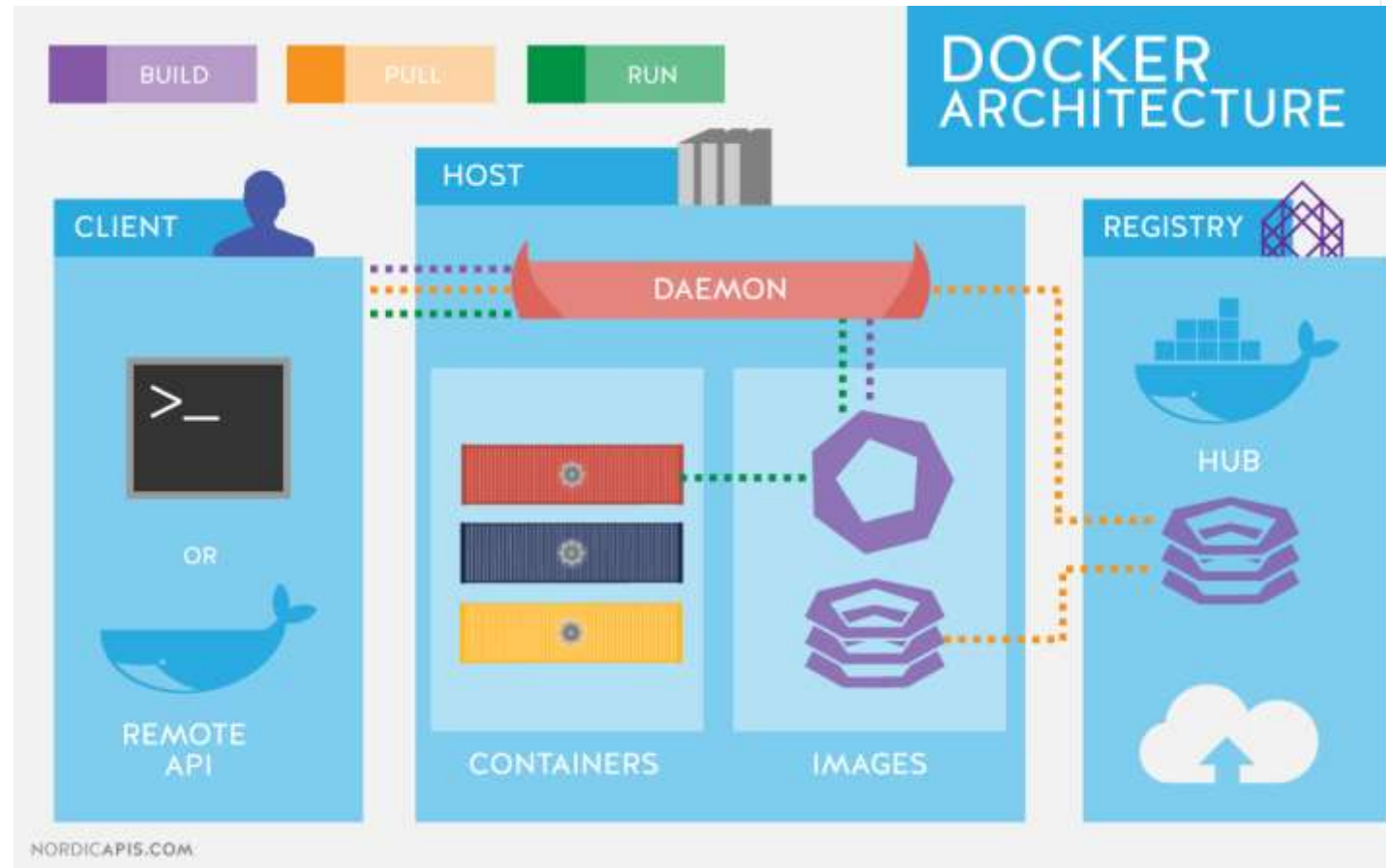
Basic taxonomy in Docker



IT

Docker components

- Batteries included but “removable” (“replaceable”)
- Built in security
 - Linux hosts can use AppArmor or SELinux
 - Namespaces to restrict access to host system resources.



Educate

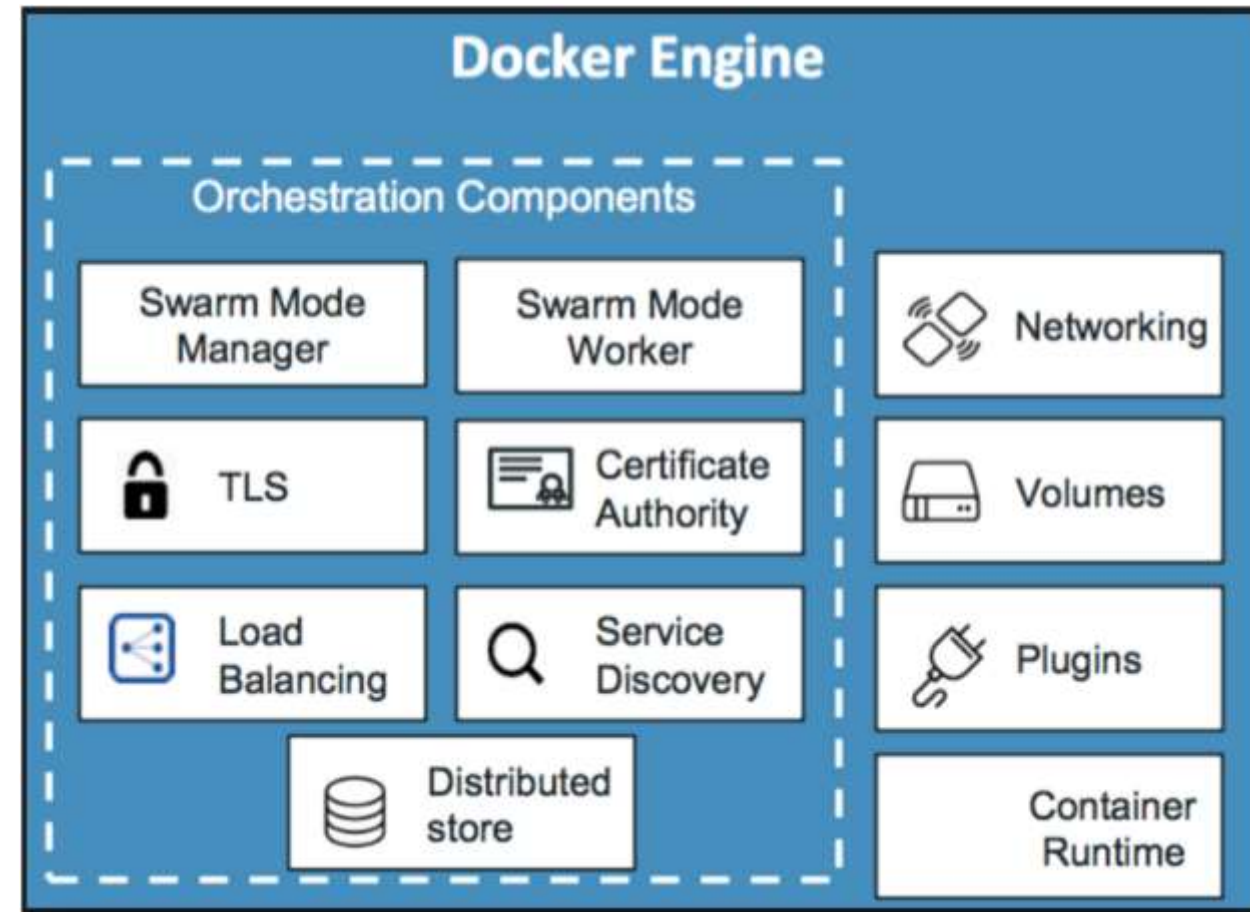
Advise

Implement

Manage

Docker on Windows & Linux hosts

- Docker being only container platform supporting both Windows and Linux.
- Windows based container supported only on Windows 10 PRO and Windows Server 2016.
- Docker installation on windows uses different components than linux host.
- Images built for linux cannot be deployed on windows hosts.
- Windows do have a trick: docker-machine and moby project to run linux containers through a lightweight VM.





Module 04

Application modernization

Educate

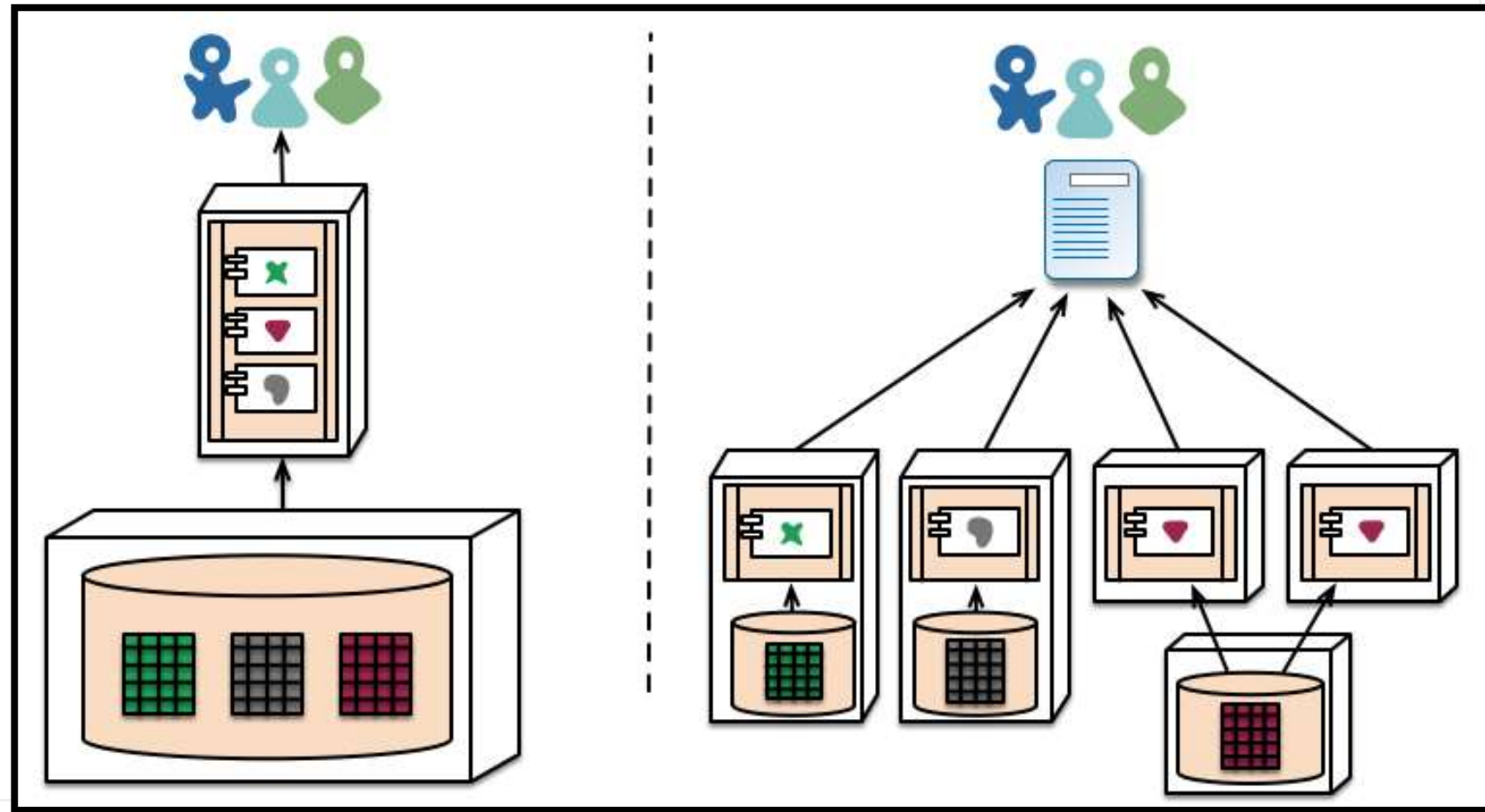
Advise

Implement

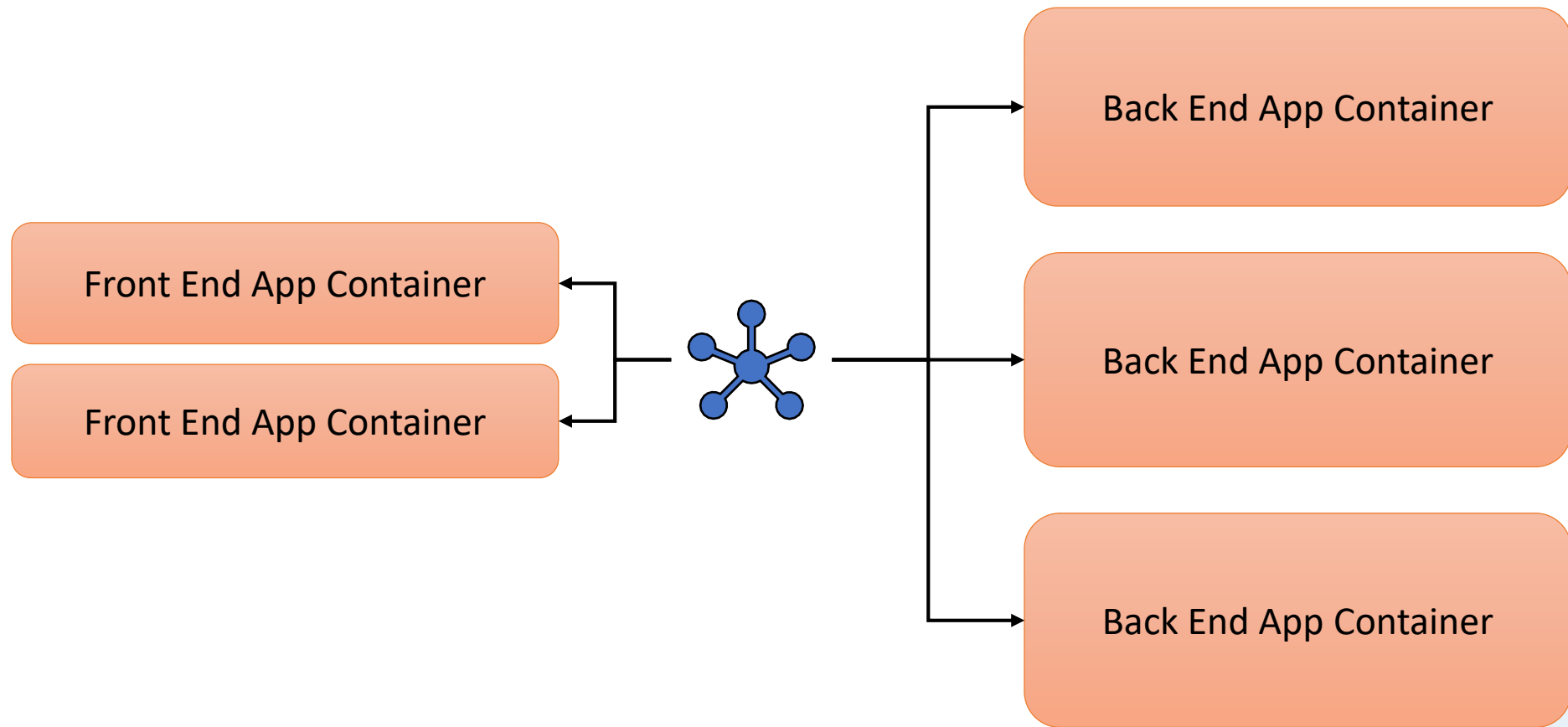
Manage

Application types

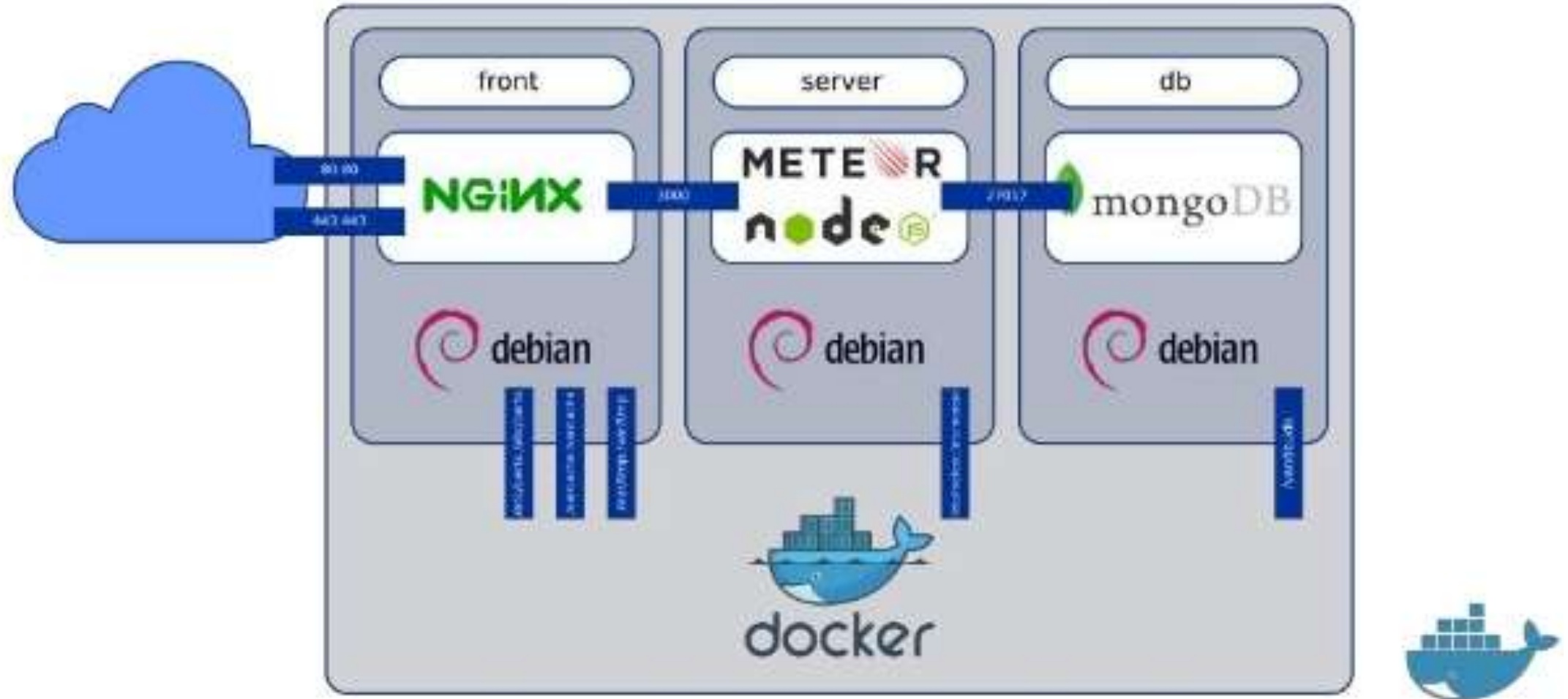
- A multi-layered monolith application
- A distributed application / SOA
- Micro-services
- Containers do support all of them!
- Monolith = Single large Container for App
+ DB Container
- Distributed =
Container for each Service / Module



A typical TWO TIER Application



Life after Docker



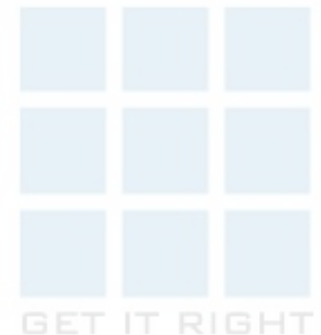
Dockerfile

- Automates the “Build” process of a container.
- Integrates with all popular CI/CD tools.
- Simple language to define entire build workflow
- Basic system administration commands needed.
 - Windows : prefer powershell
 - Linux: prefer bash
- Can be placed in SCM

```
1 | # Using base image of PHP-7 installed on "apache httpd"
2 | # This base image is derived from "Ubuntu"
3 | FROM php:7.0-apache
4 |
5 | # Install any updates and clean the cache immediately
6 | RUN apt-get update && \
7 |     apt-get clean
8 |
9 | # copy contents of local directory "files" into
10 | # deployment directory in container
11 | COPY files /var/www/html/
12 |
```

Dockerfile reference

Command	Description
FROM	Initializes a new build and define “base” image to be used for building.
RUN	A Step to be executed in build process. Can be used for installing necessary package.
CMD	Provide DEFAULT “startup” action for container. Must be used only ONCE.
LABEL	Provides additional META-DATA like Author name
EXPOSE	A Hint for port used by container, can be overridden at run-time.
ENV	Define environment variables.
COPY	Copy files from host system to container. NOTE : optional feature --chown for linux containers





Module 05

Understanding Orchestration

Educate

Advise

Implement

Manage

Orchestration

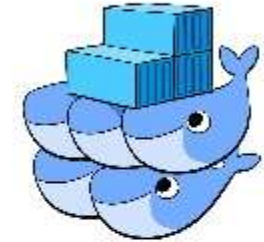
- Need for Orchestration
 - Ephemeral computing
 - Desired State
 - Cluster management
- Container Orchestration platforms
 - Docker Swarm (from Docker Inc)
 - Kubernetes (from Google)
 - DC/OS (based on Apache Mesos)
 - Nomad (from HashiCorp)

Container Orchestration Tools



MESOS

Marathon (Mesosphere)



Docker Swarm



Nomad (HashiCorp)

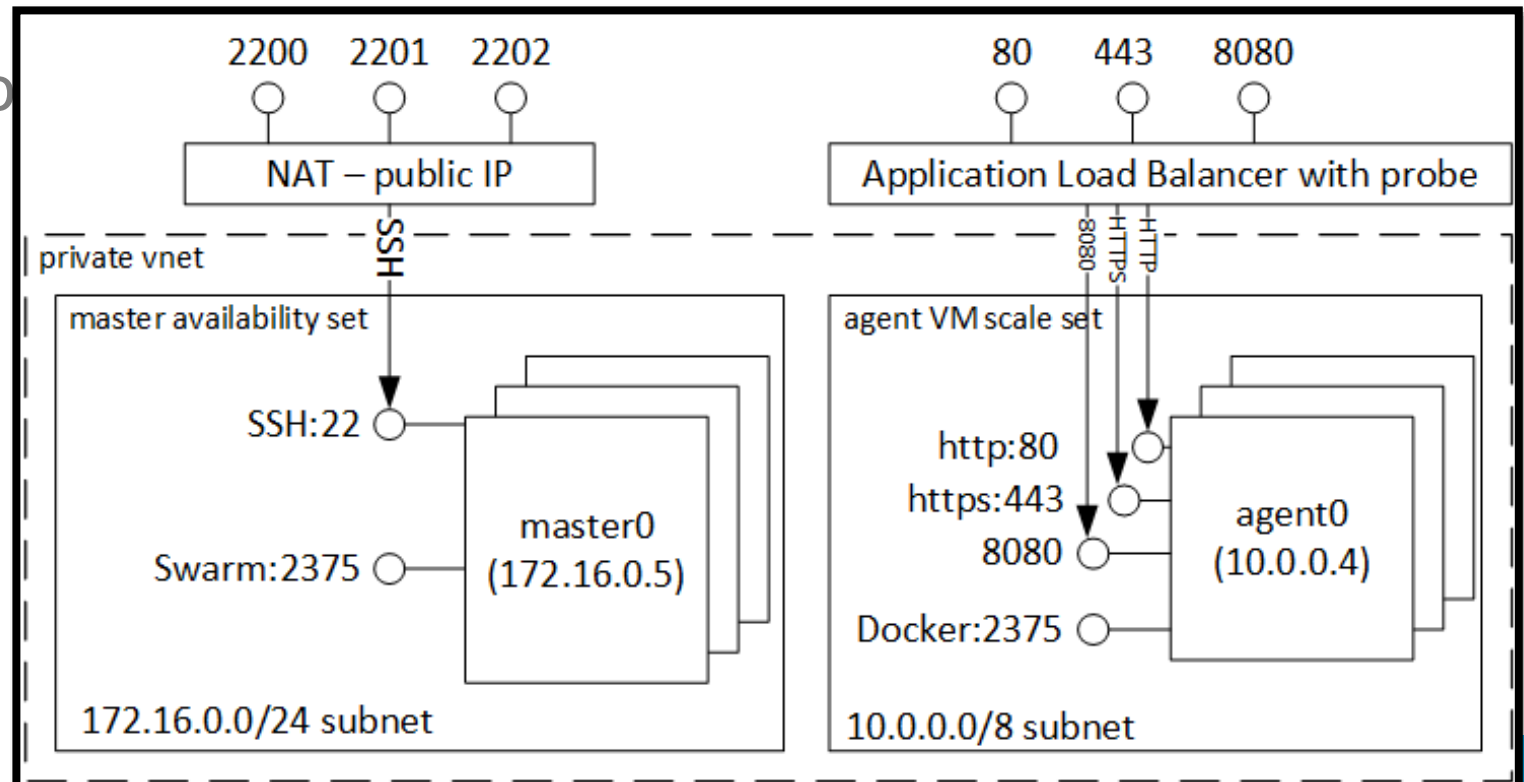


Kubernetes

GET IT RIGHT

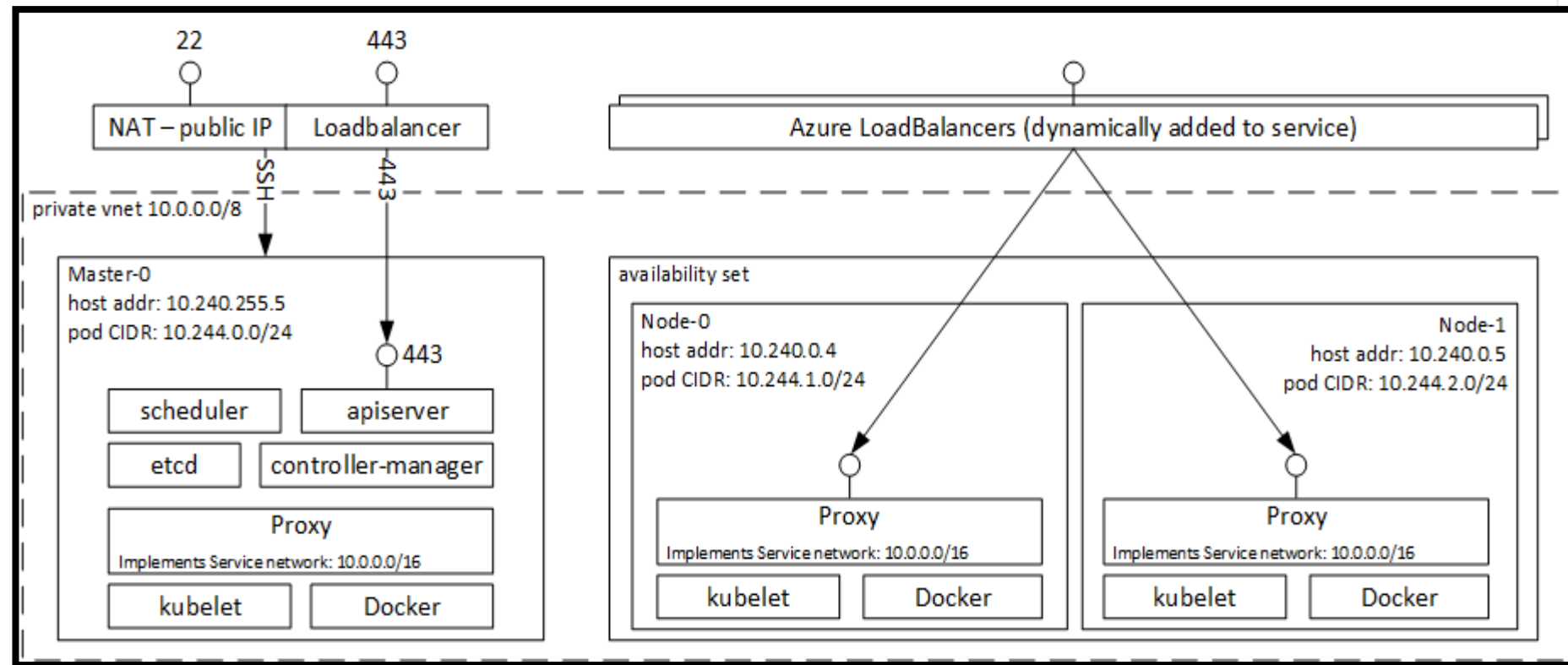
Docker Swarm

- Cluster management *integrated with Docker* engine
- Decentralized design (for cluster management)
- Declarative service model (ref: docker-compose)
- Scaling
- Desired state reconciliation
- Multi-host networking
- Service discovery
- Load balancing



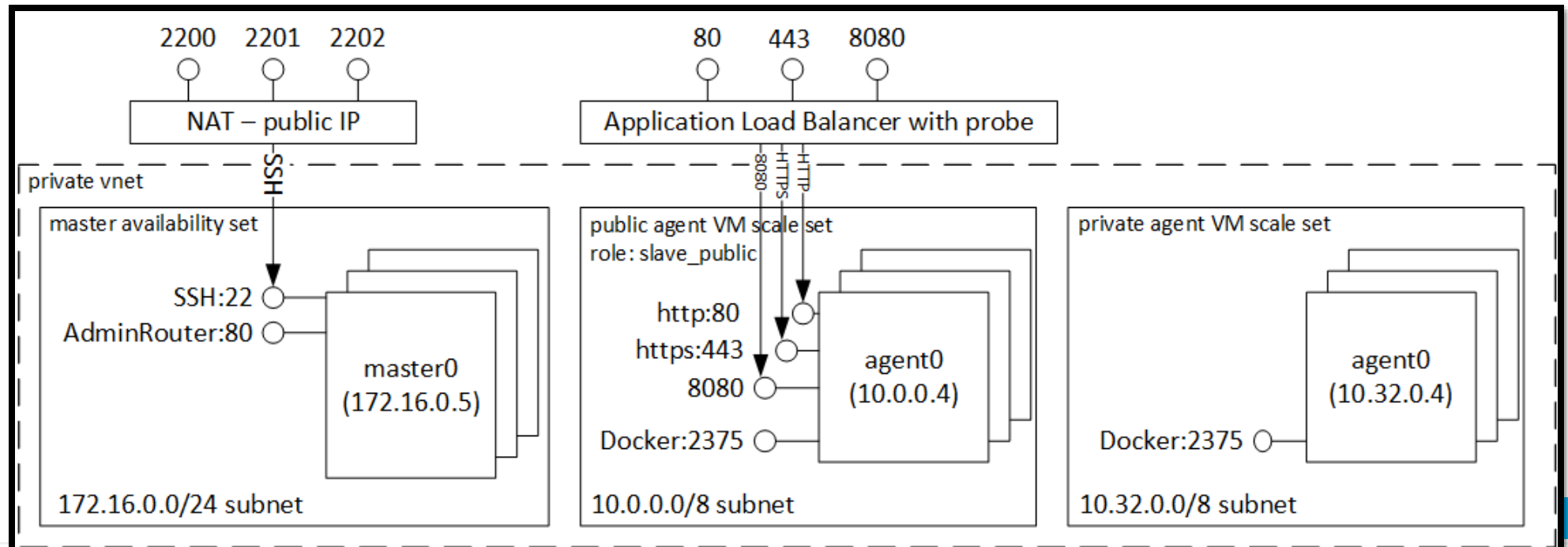
Kubernetes

- Cluster managed from google with 10+ years of internal use by google.
- Open source, run anywhere
- Service discovery
- Load balancing
- Self healing
- Storage orchestration
- Scaling



DC/OS

- Distributed OS based on Apache Mesos
- Treat cluster as a single machine
- High resource utilization
- Container orchestration using “Marathon”
- Cloud agnostic installation





SYNERGETICS
— GET IT RIGHT —

Q/A

Educate

Advise

Implement

Manage