Container Orchestration using Docker & Kubernetes

By Mahendra Shinde

Educate | Advise | Implement | Manage

Agenda

Module 1: Introduction to Containers

Module 2: Container Architecture

Module 3: Docker Architecture

- HOL 1: Installing Docker CE on Windows & Linus Host

Module 4: Application modernization using docker

- Demo : Deploying TWO TIER application Using Docker compose
- HOL 2: Deploying TWO TIER application (DotNet Core + MS SQL) using Docker compose

Module 5: Understanding Orchestration

Module 1

Introduction to Containers

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.

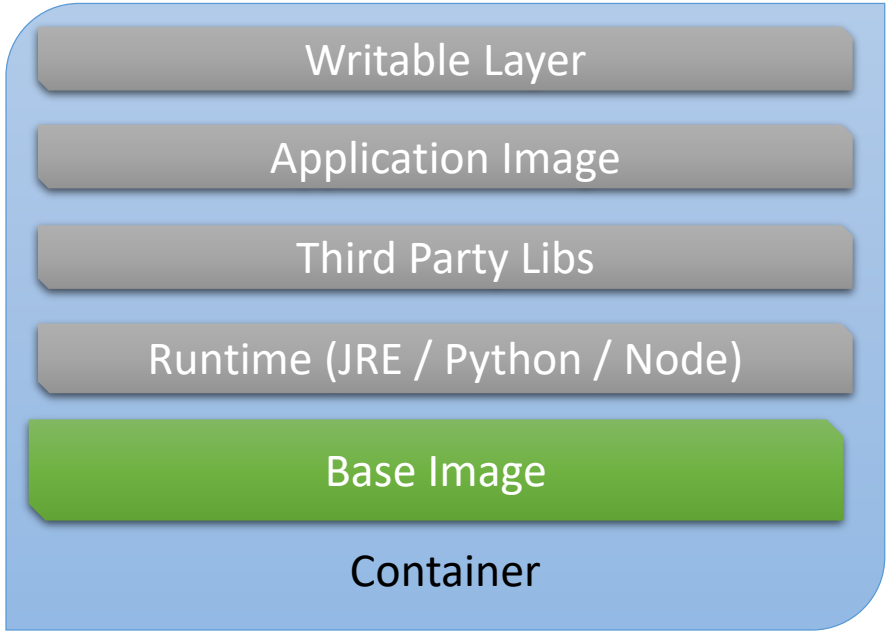
Application packaging strategies /challenges

- 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

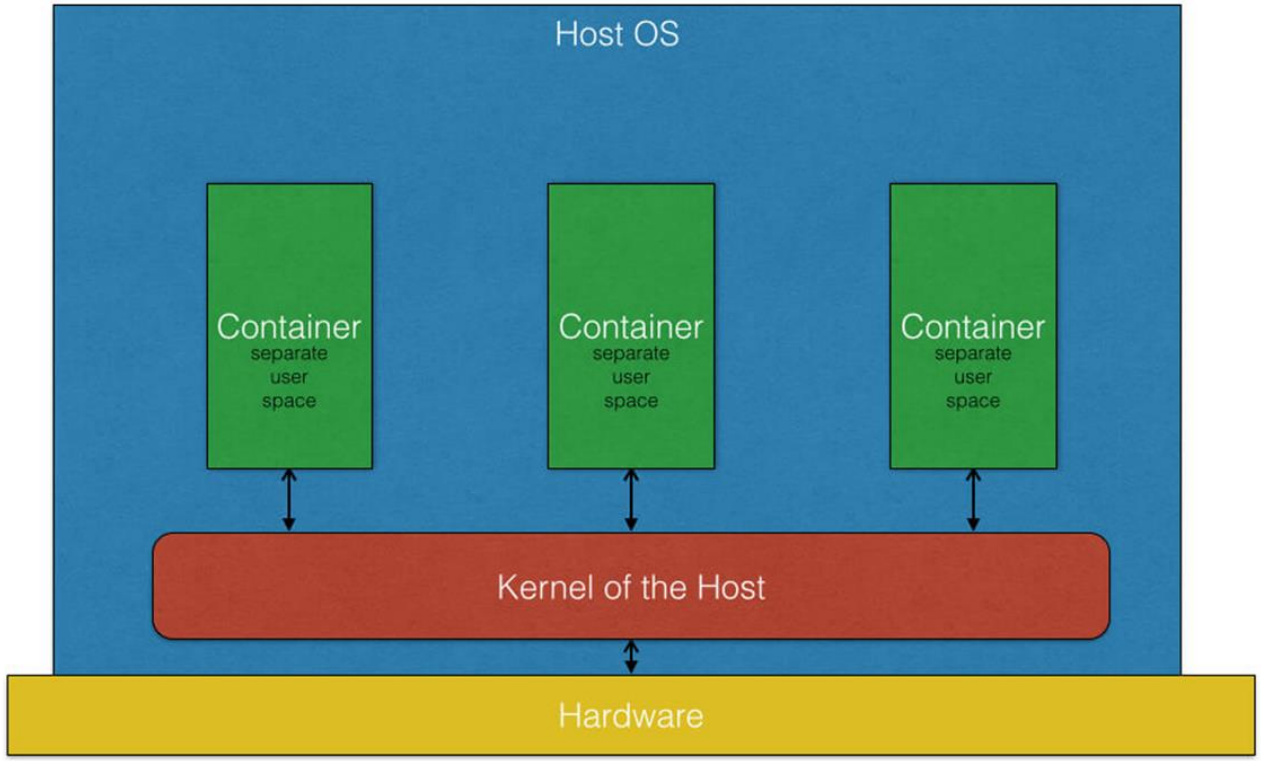
Containers as alternative

- 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.



SYNERGETICS
GET IT RIGHT

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

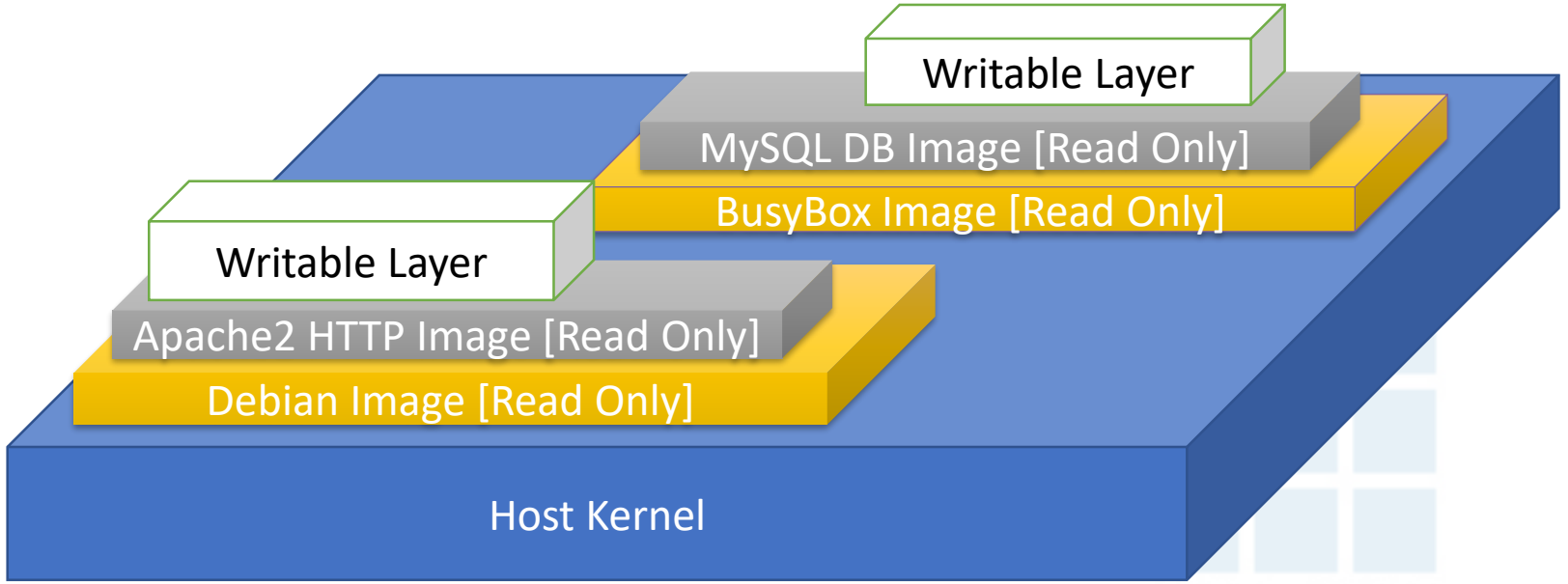
Module 2

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.

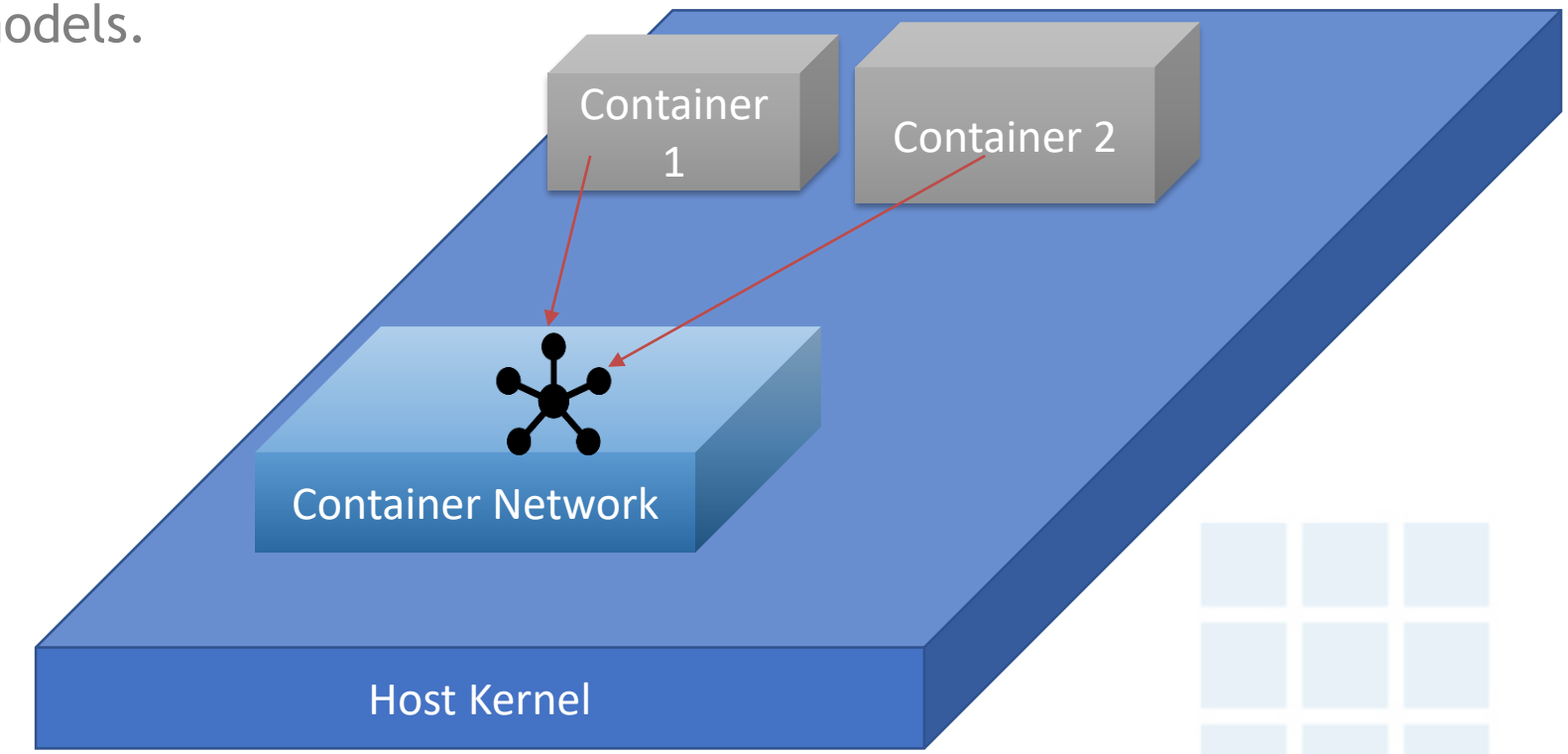
Image repositories

- 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 as 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 it’s 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”

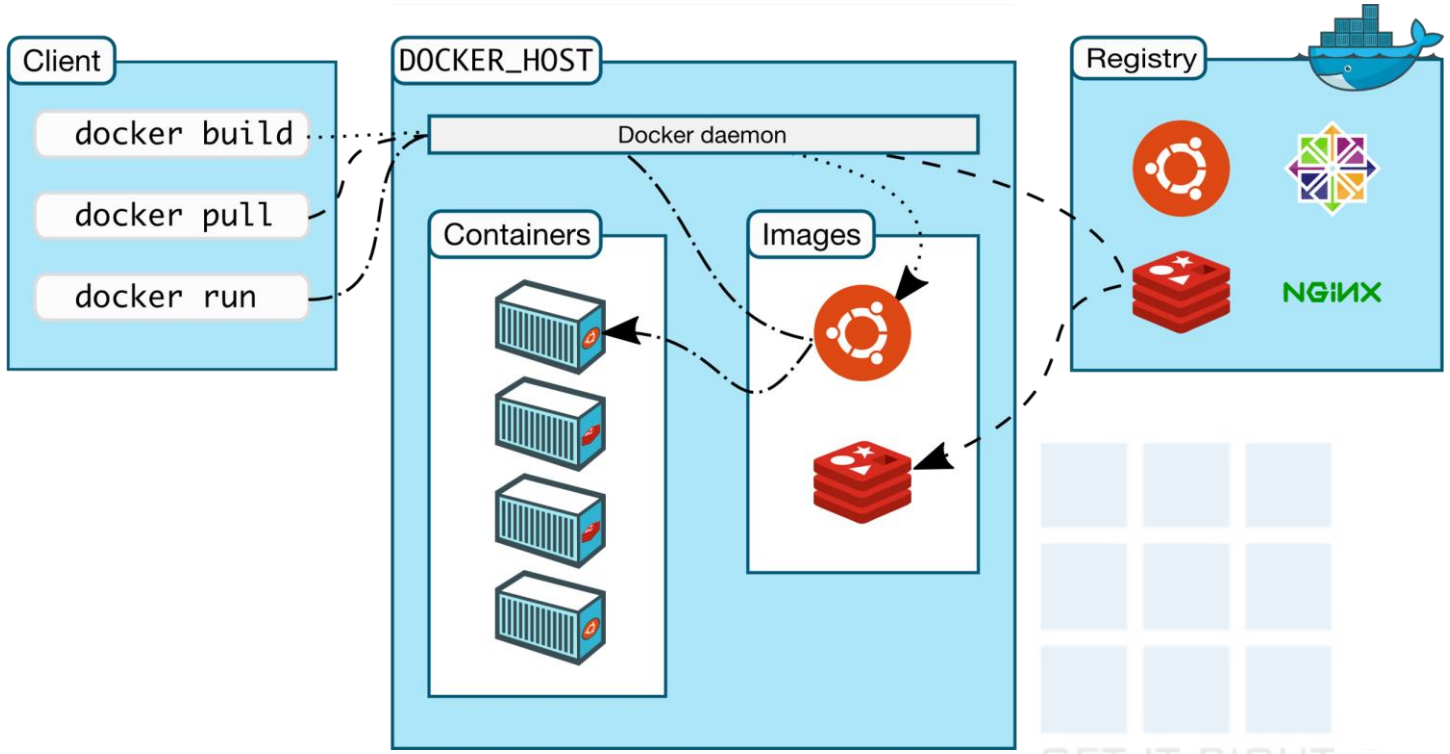
Docker Editions

- Community edition (Docker CE)
  - Ideal for developers and small teams experimenting with containers.
- Enterprise Edition (Docker EE)
  - Enterprise development & IT team for business critic 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				✓

Docker components

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



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.

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.

Hands On LAB

- HOL 1: Installing Docker CE on Windows Server 2016
  - Pre-requisite:
    - Azure subscription
    - Basic Windows administration w/ powershell.
    - Windows server 2016 VM on Azure
    - Remote Desktop client (built in windows) for accessing windows VM.

Hands On LAB

- HOL 2: Installing Docker CE on Ubuntu Server 18.04 LTS
  - Pre-requisite:
    - Azure subscription
    - Basic Linux administration using bash shell.
    - Ubuntu Server 18.04 LTS VM on Azure
    - SSH Client (PuTTY or Git Bash or Ubuntu Bash)

SYNERGETICS
GET IT RIGHT

Module 04

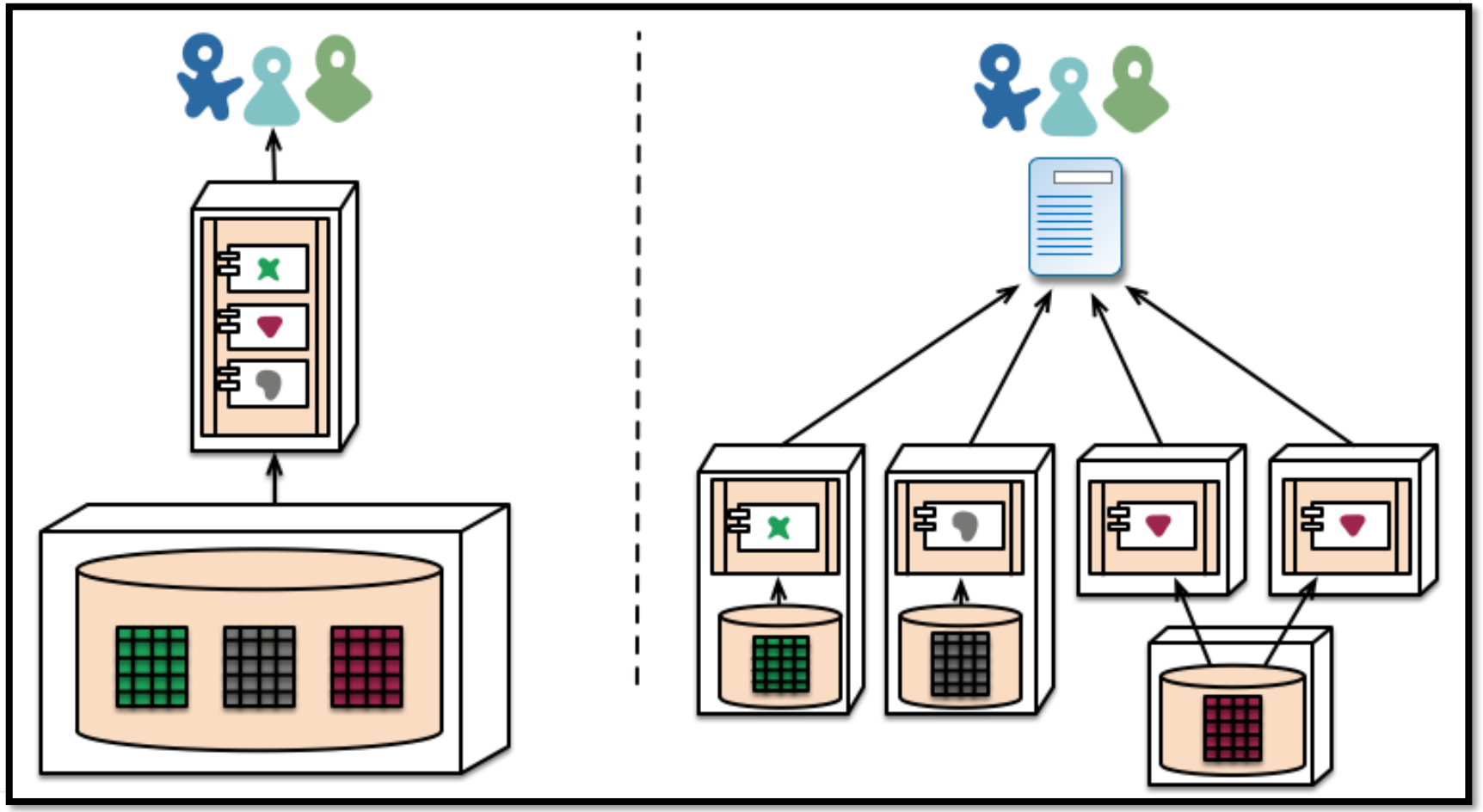
Application modernization

Educate | Advise | Implement | Manage

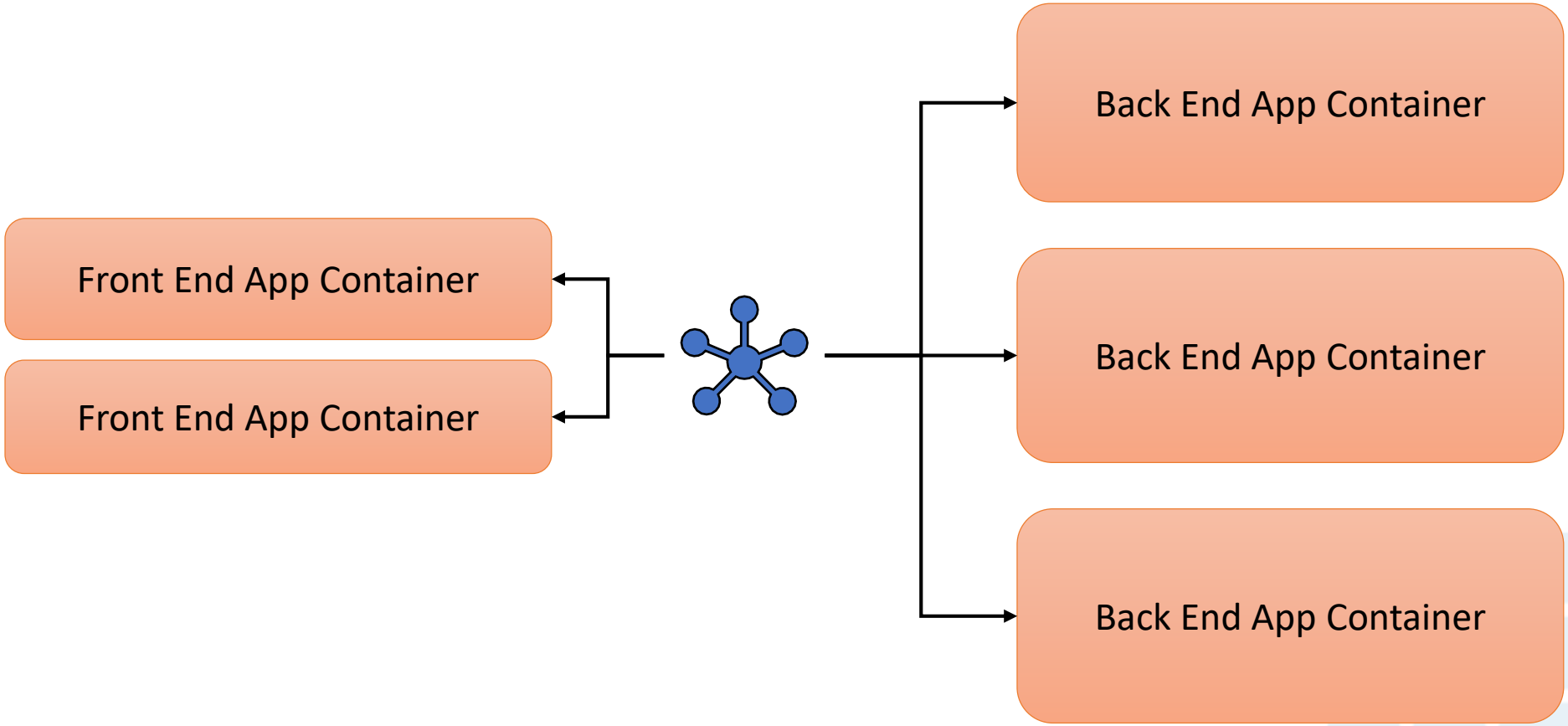
Application types

- 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



Dockerfile

SYNERGETICS
GET IT RIGHT

- 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

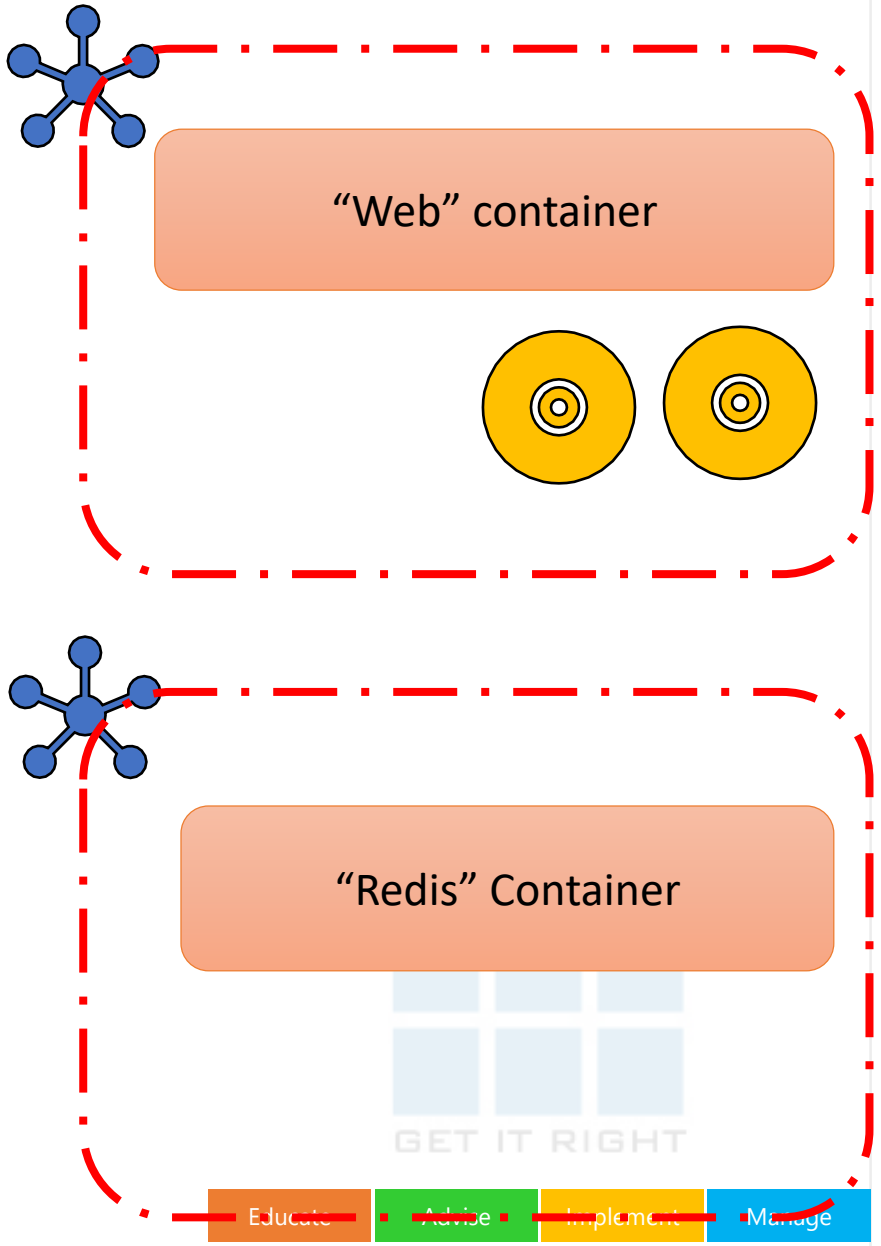
```
1  # Using base image of PHP-7 installed on "apache htttd"
2  # This base image is derived from "Ubuntu"
3  FROM php:7.0-apache
4
5  # Install any updates and clean the cache immediately
6  RUN apt-get update && \
7      apt-get clean
8
9  # copy contents of local directory "files" into
10 # deployment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

Docker compose

- A Tool for defining and deploying multi-container application.
- Compose is a THREE STEP process
  - Define the application environment using Dockerfile
  - Define the services that make up your application using docker-compose.yml file
  - Run docker-compose up command to quickly run entire application in isolated environment.
- Provides re-producible deployment artefacts.
- Only recreates container that have changed.
  - Upon “restart” containers are spawned from “cached” versions if no change detected.
- Supports environment variables
- Supports “secrets”
- *Traditionally* used for Single host deployment.

Docker Compose

```
version: '3'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```



Docker Compose vs Dockerfile

SYNERGETICS
GET IT RIGHT

```yaml
version: '3'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Dockerfile ▸ ...

```
1  # Using base image of PHP-7 installed on "apache htttd"
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

Docker-compose reference

Section	Description
Services:	Define one or more service [YAML syntax]
Build:	Provide relative path to directory which contains “Dockerfile”
Image:	Provide image name using syntax: [repository-name]/[image-name]:[tag] For public images on docker-hub [repository-name] is dropped.
Volumes:	Define volumes to be mounted
Networks:	Define a private network for application.
Labels:	Additional meta-data

Demos

- Demo 1: Deploying an application back-end and front-end as single application.
  - Demo 1-A Using MySQL with Wordpress
  - Demo 1-B Using MS-SQL with Asp.net core

Hands On

- HOL 2] Deploying front-end application (ASP.NET core) and Back-end (MS SQL) using single docker-compose.
  - Use docker-compose to define entire application
  - Use existing VM created in HOL-1

Module 05

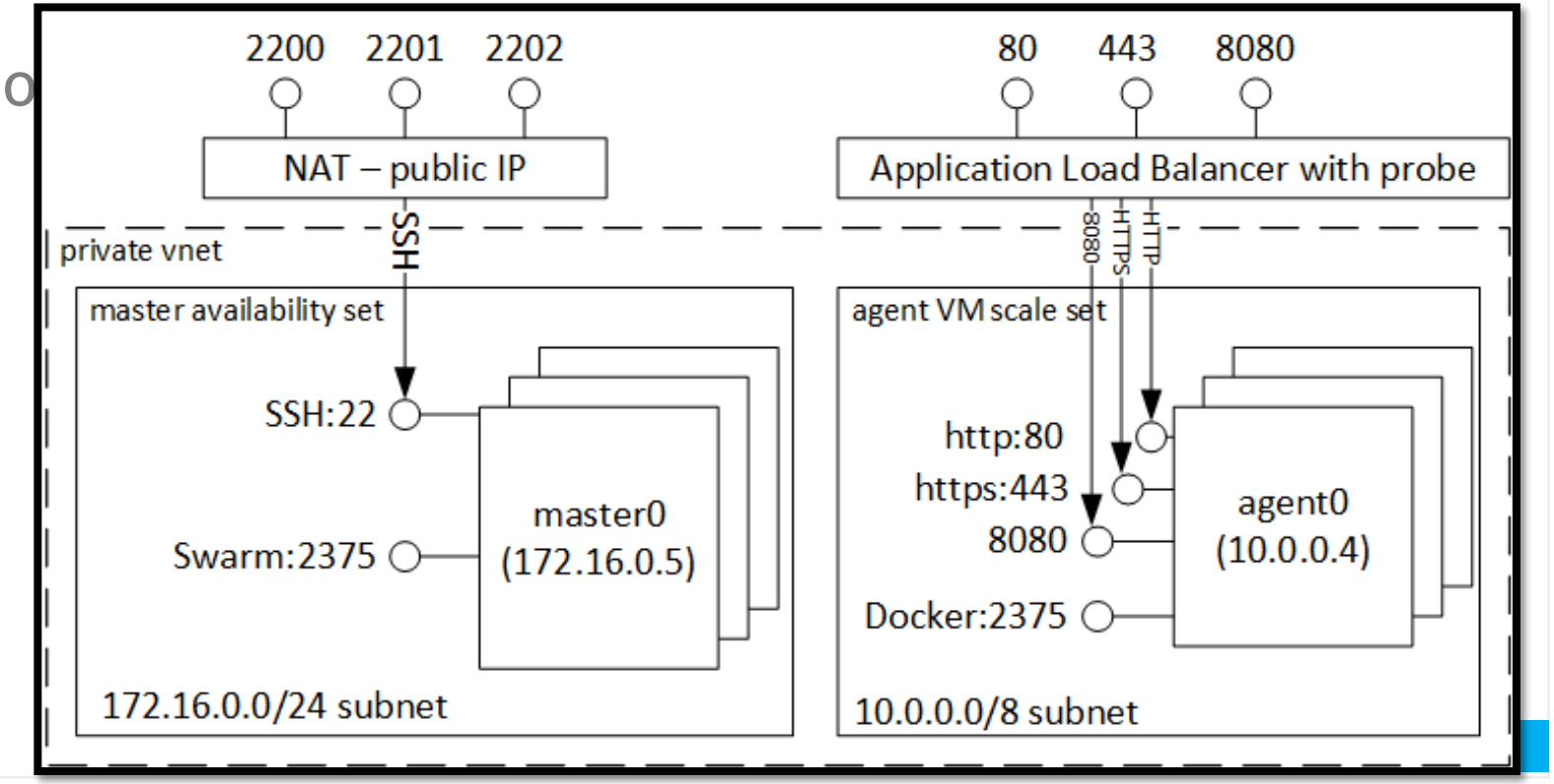
Understanding Orchestration

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)

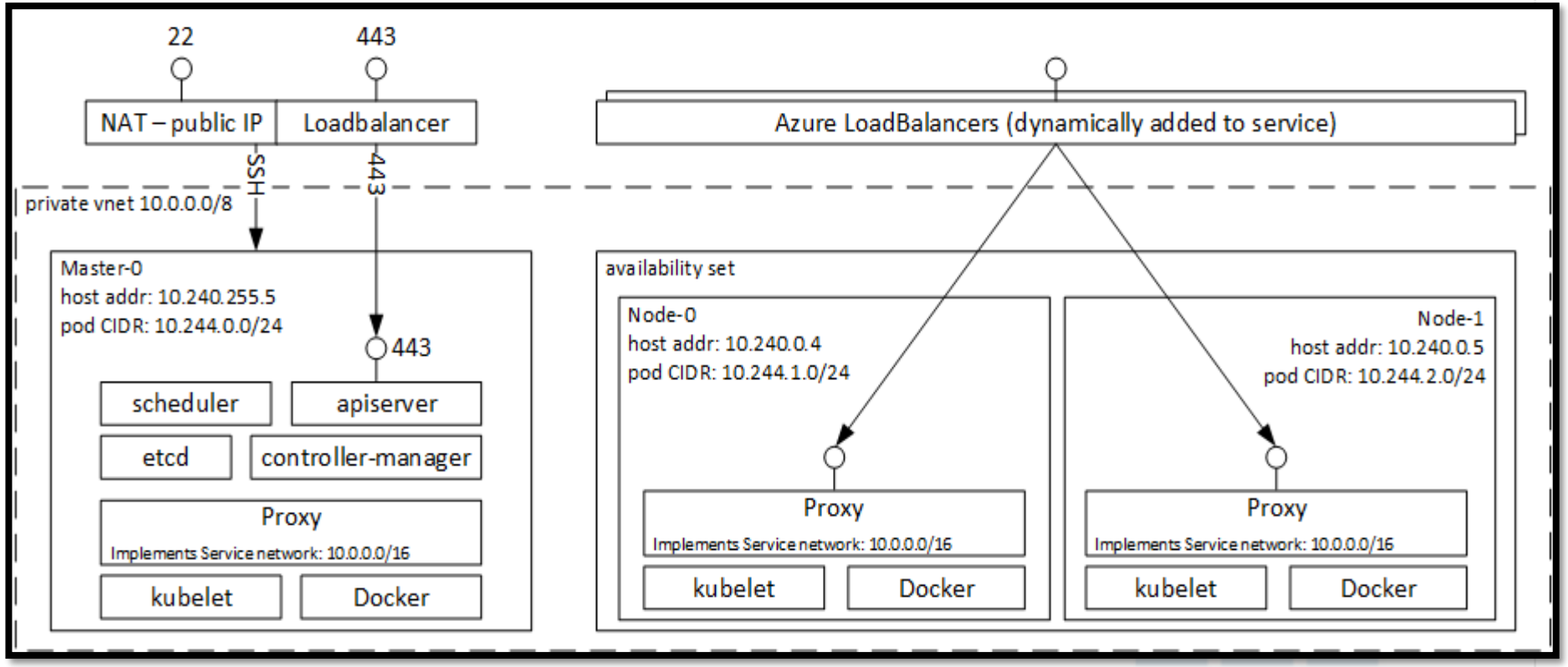
Docker Swarm

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



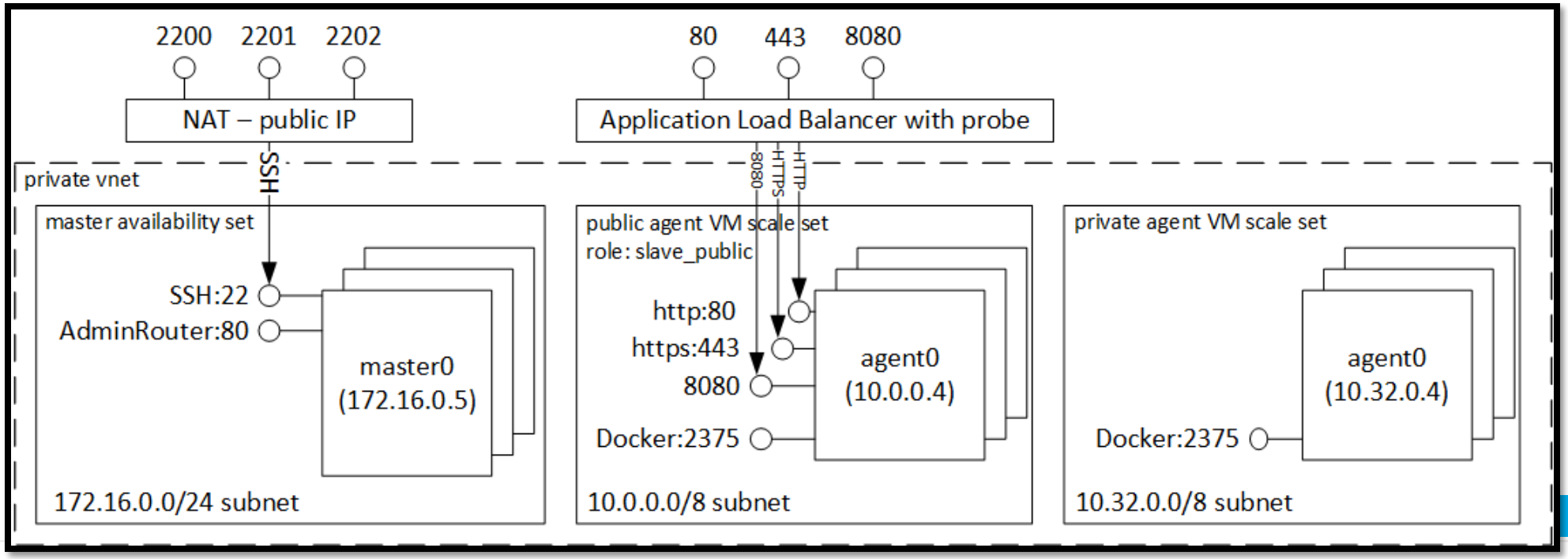
Kubernetes

- 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