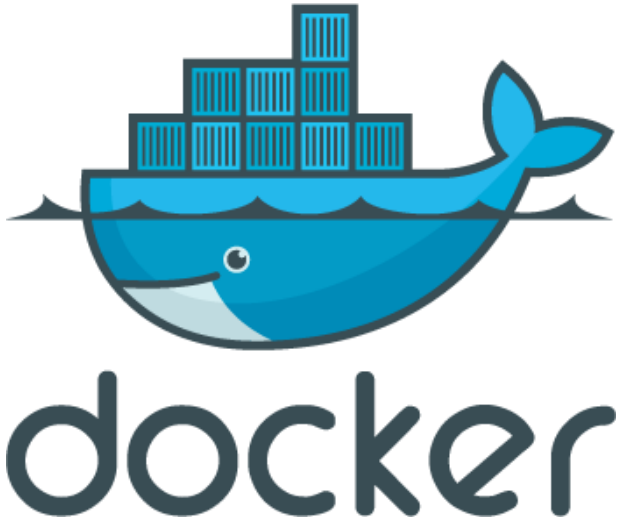
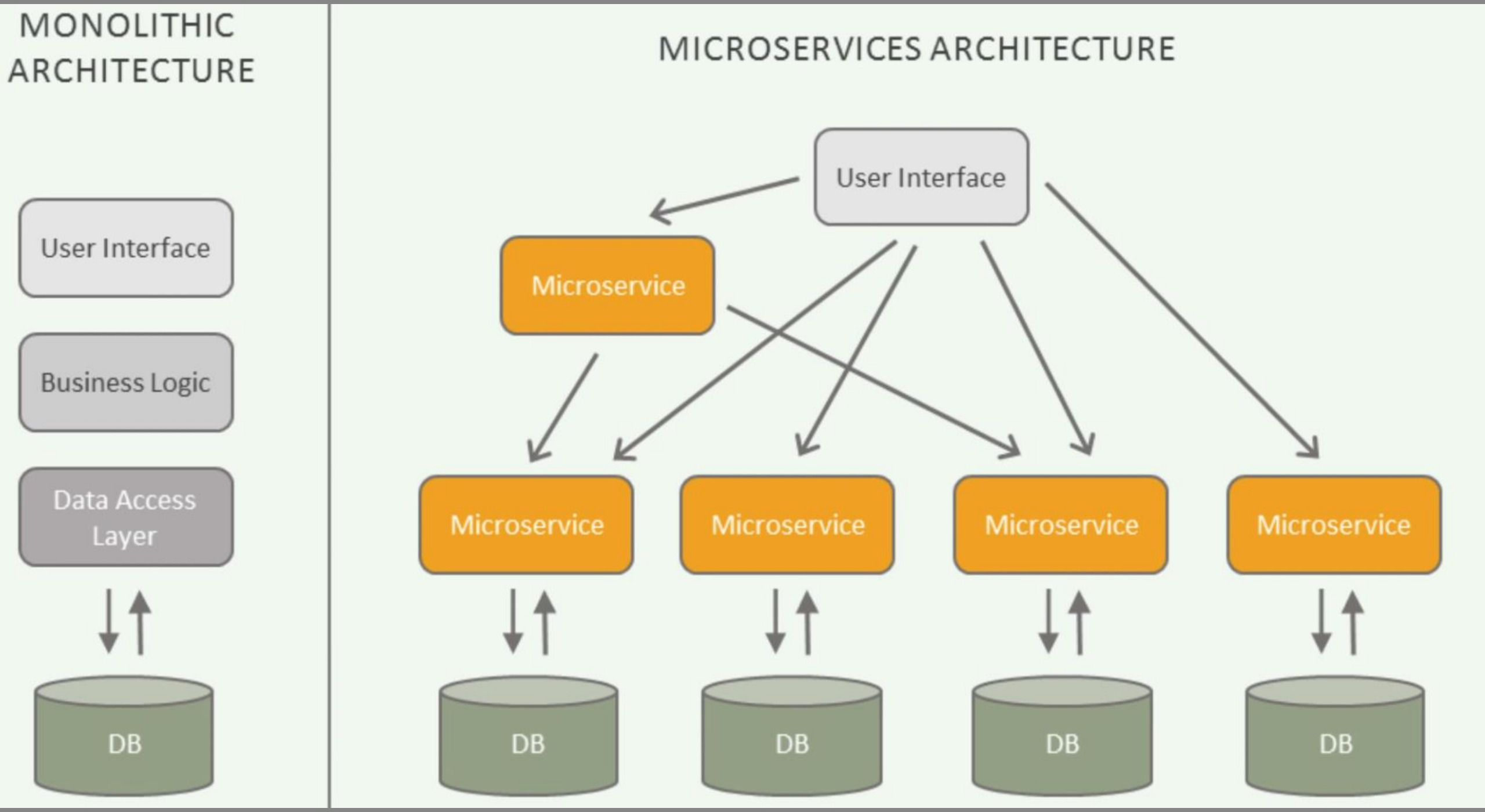


Microservices

Landmark Technologies
mylandmarktech@gmail.com

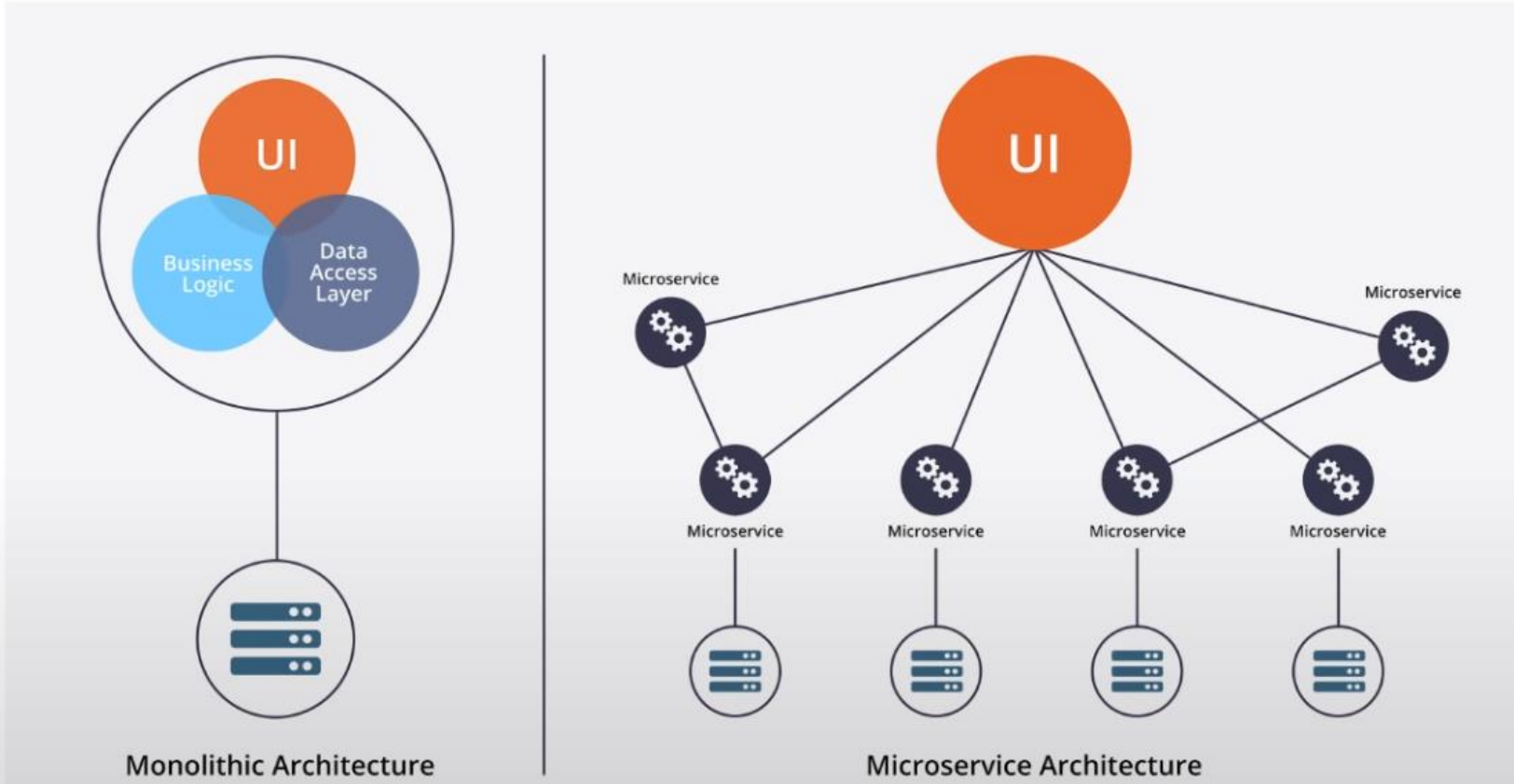
Microservices

- Separate business logic functions
- Instead of one big program, several smaller applications
- Communicate via well defined APIs - usually HTTP
- In demand


MONOLITHIC ARCHITECTURE
User Interface
Business Logic
Data Access Layer
DB
MICROSERVICES ARCHITECTURE
User Interface
Microservice
Microservice
Microservice
Microservice
Microservice
DB
DB
DB
DB

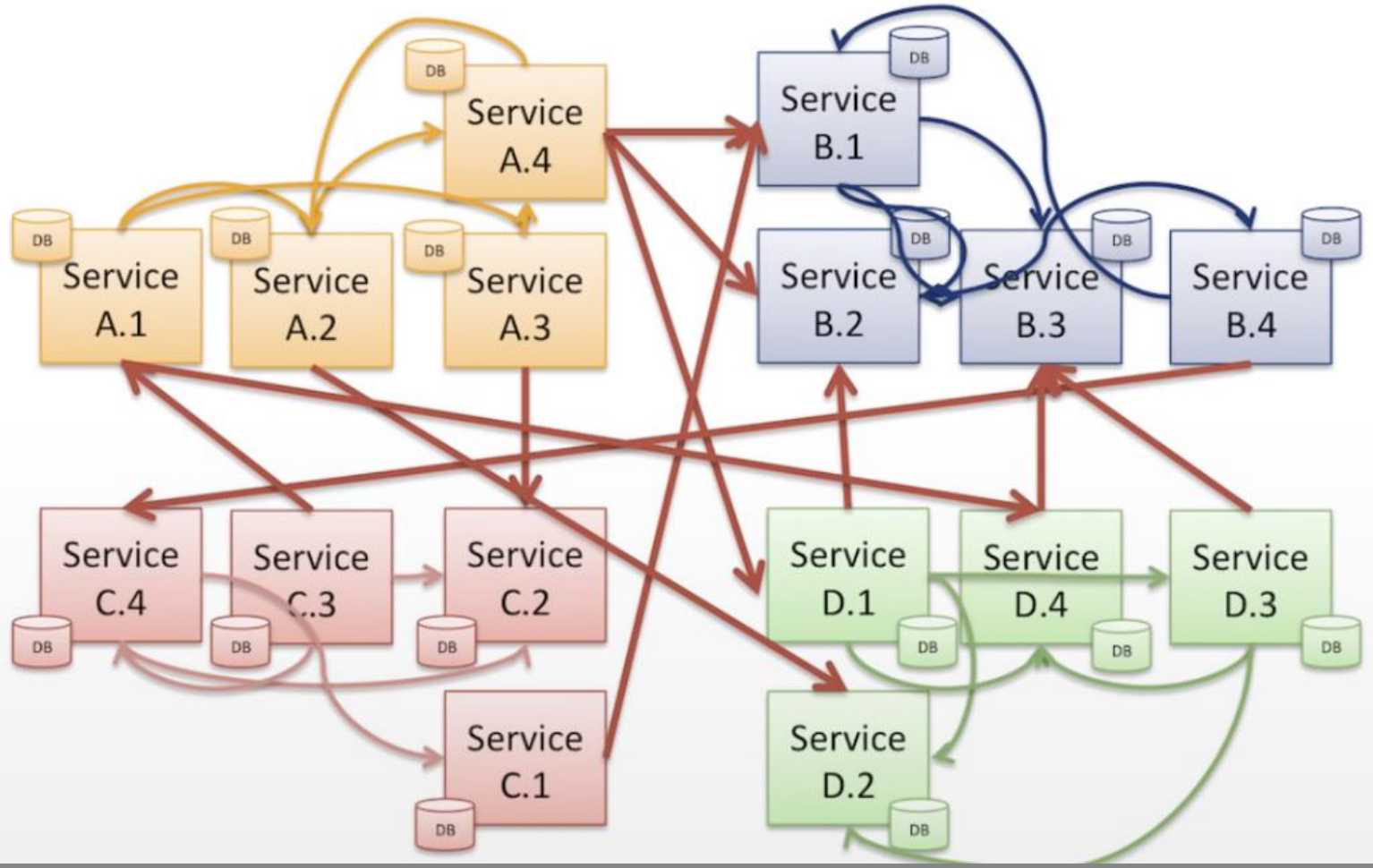
Microservices - Advantages

- Language independent
- Fast iterations
- Small teams
- Fault Isolation
- Pair well with containers
- SCALABLE
  - Big plus



Microservices - Disadvantages

- Complex networking
- Overhead
  - Databases
  - Servers



LANDMARK TECHNOLOGIES

LANDMARK
TECHNOLOGIES

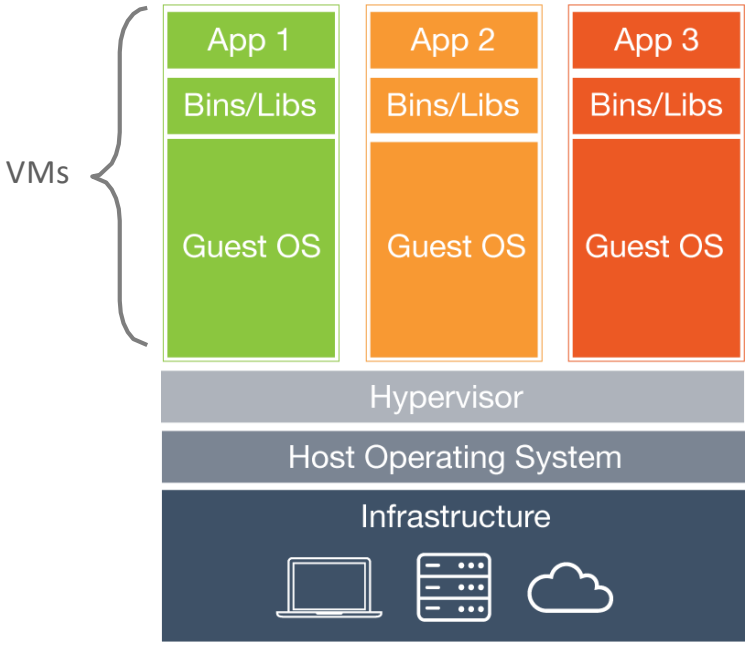
Program Agenda

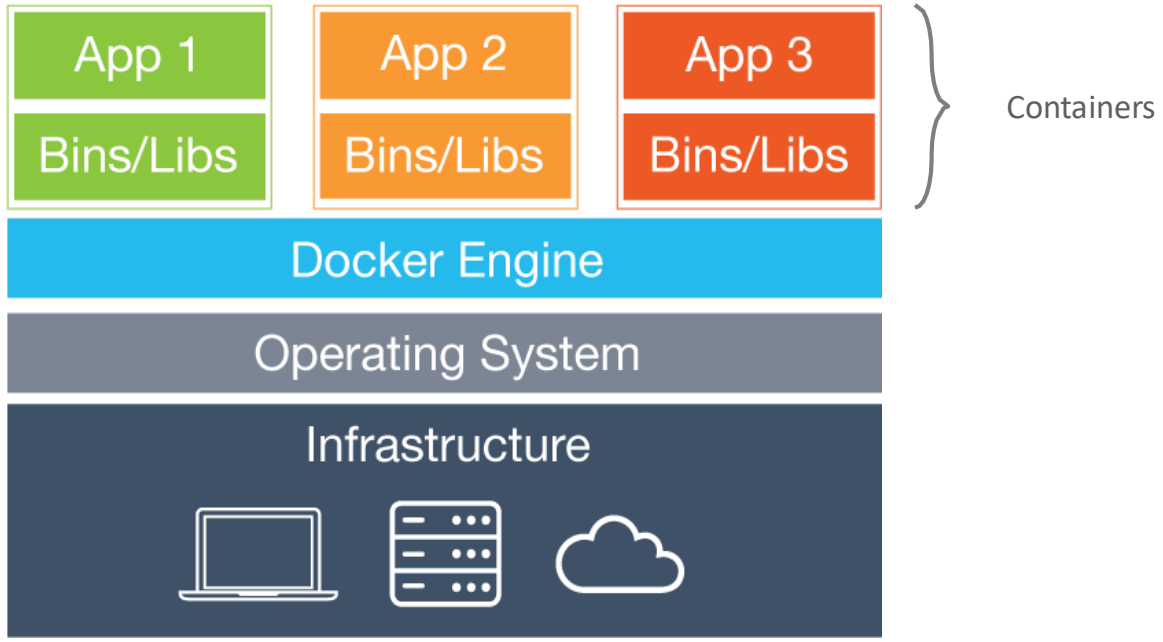
- Virtualization and Containerization
- Introduction
- Installation
- Virtual Machine and Docker
- Dockerfile, Docker Image and Docker Container
- Docker Commands
- Build the Dockerfile
- Create Docker Custom images and Containers, push to Docker Hub
- Docker Compose
- Docker Swarm

LANDMARK
TECHNOLOGIES

Basic Architecture and comparison to VM's

Virtual Machines vs. Containers





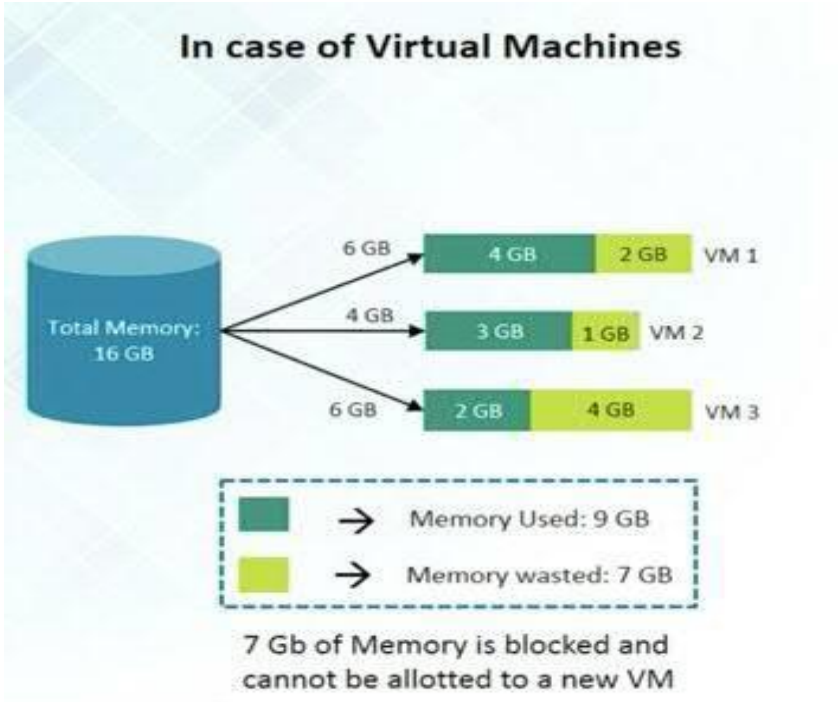
Virtual Machines

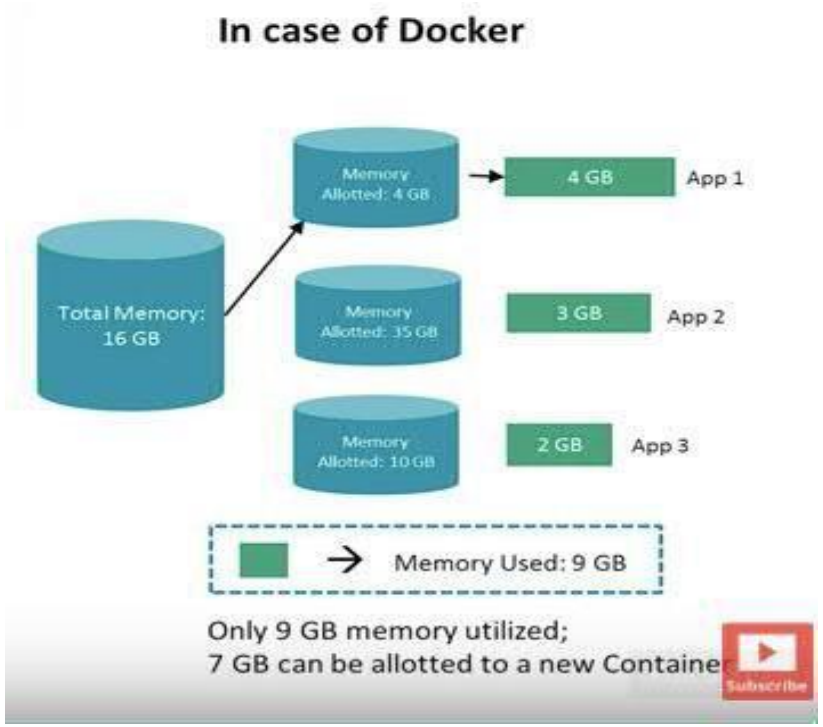
- Each virtual machine (VM) includes the app, the necessary binaries and libraries and an **entire guest operating system**

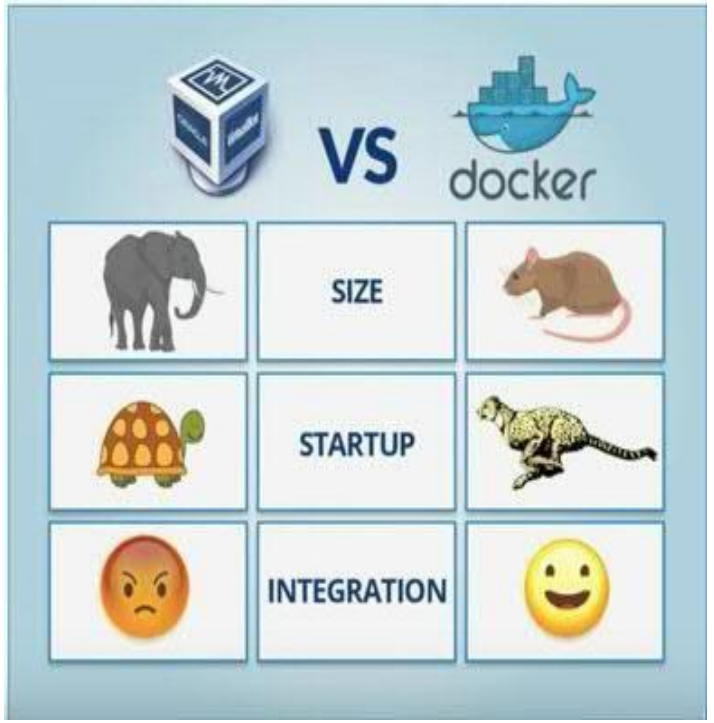
Containers

- Containers include the app & all of its dependencies, but **share the kernel** with other containers.
- Run as an isolated process in userspace on the host OS
- Not tied to any specific infrastructure – containers run on any computer, infrastructure and cloud.

Virtual Machines vs. Containers







LANDMARK TECHNOLOGIES

Virtual Machines vs. Containers

Features	Virtual Machines (VMs)	Application Containers	Application Container Benefits
Virtualization	In VMs, the virtualization occurs in server hardware.	In containers, the virtualization occurs at the OS level.	Faster time-to-market
Standardization	VMs are standardized systems that have capabilities similar to that of bare metal computers.	Containers are not standardized, as their kernel OS has varying degrees of complexity.	Makes applications more portable by isolating them from the underlying infrastructure
Resource management	Each VM has its own kernel OS, binary, and library.	Containers share the same host OS, resources and Kernel.	Lower overhead costs
Density	VMs require a few gigabytes (GB) of space that limits them into a single server.	Containers require a few megabytes (MB) of space that enables many containers to fit into a single server.	Lighter weight than VMs
Boot time	Minutes	Seconds	Easy to update and release newer versions of applications

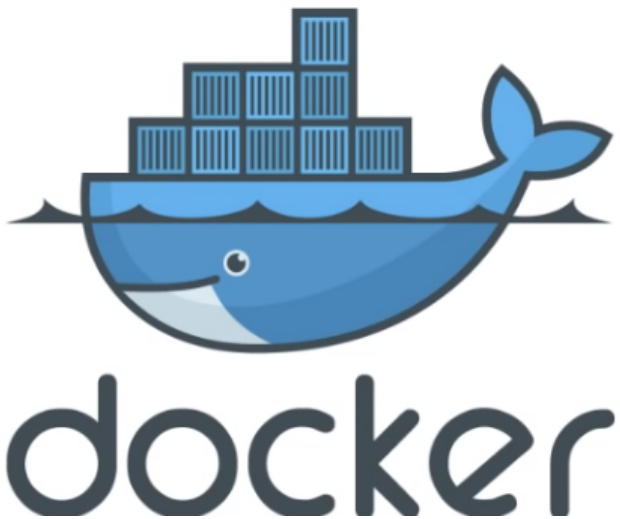
Docker Introduction

What is Docker?

- *Docker is a containerisation platform which packages your app and its all dependencies together in the form of containers. so as to ensure that your application works seamlessly in any environment be it dev or test or prod.*
- *Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications*
- *Docker is available in two editions: Community Edition (CE) and Enterprise Edition (EE).*
- *Docker Community Edition (CE) is ideal for developers and small teams looking to get started with Docker and experimenting with container-based apps.*
- *Docker Enterprise Edition (EE) is designed for enterprise development and IT teams who build, ship, and run business critical applications in production at scale.*

Docker

"Containers are a way to package software in a format that can run isolated on a shared operating system. Unlike VMs, containers do not bundle a full operating system - only libraries and settings required to make the software work are needed. This makes for efficient, lightweight, self-contained systems and guarantees that software will always run the same, regardless of where it's deployed."



LANDMARK
TECHNOLOGIES

Type	Containerization Tool
Vendor	Docker
Is Open Source?	Yes – Some extent
Operating system	Cross Platform
URL	https://www.docker.com/

Pre-Requisite
Download Docker and install from below url
https://www.docker.com/products/docker-toolbox

Register in https://hub.docker.com/

Advantages of Docker

- ***Rapid application deployment*** *– containers include the minimal run time requirements of the application, reducing their size and allowing them to be deployed quickly.*

- ***Portability across machines*** *– an application and all its dependencies can be bundled into a single container that is independent from the host version of Linux kernel, platform distribution, or deployment model. This container can be transferred to another machine that runs Docker, and executed there without compatibility issues.*

- ***Version control and component reuse*** *– you can track successive versions of a container, inspect differences, or roll-back to previous versions. Containers reuse components from the preceding layers(base images), which makes them noticeably lightweight.*

- ***Sharing*** *– you can use a remote repository to share your container with others. And it is also possible to configure your own private repository.*

- ***Lightweight footprint and minimal overhead*** *– Docker images are typically very small, which facilitates rapid delivery and reduces the time to deploy new application containers. Allowing for more efficient use of computing resources, both in terms of energy consumption and cost effectiveness.*

- ***Simplified maintenance*** *– Docker reduces effort and risk of problems with application dependencies. Multiple containerized apps on a single server don't mess up each other. If you update an app, you just build a new image, run fresh containers and don't have to worry about other ones on the machine breaking*

Advantages of Docker

- ***Self-sufficient***

  A Docker container contains everything it needs to run

  Minimal Base OS

  Libraries and frameworks

  Application code

  A Docker container should be able to run anywhere that Docker can run.

- ***Docker shines for microservices architecture.***

  NOTE: Containers aren't required to implement microservices, but they are perfectly suited to the microservices approach and to agile development processes generally.

Build & Deployment Process

To be more clear,
In 1990’S we had physical servers where apps are hosted and served.

In 2000’S we had Virtual Machines where apps are hosted and served ,running on top of the Physical servers. Virtualization started here.

In early 2013 the concept of Containerization picks up where we can package the application and its dependencies so that whatever a developer released will work regardless of which environment it runs under(DEV, QA & PROD).

Before Docker
Build

- A developer pushes a change to SCM tools.
- The CI sees that new code is available. It rebuilds the project, runs the tests and generates a application package(like jar/war/ear ..etc.)
- The CI saves application package as build artifacts into repositories.

Deployment

- Setup dependencies(Software's) and configuration files in App(Deployment) Servers(Physical/Virtual Machines).
- Download the application package
- Start the application

Build & Deployment Process

Cons:

- Packaging and deploying applications in multiple environments is a real and challenging job.
- Setup Environments before deploying an application.
- Applications may work in one environment which may not work in another environment. The reasons for this are varied; different operating system, different dependencies, different libraries, software versions.

After Docker

Build

- A developer pushes a change to SCM tools.
- The CI sees that new code is available it. It rebuilds the project, runs the tests and generates an application package(jar/war/ear ..etc) & and also we will create docker image(application code(jar/ear/war..etc), dependencies (Software's) and configuration files).
- CI Will push docker image to the docker repo(Public Repo(Docker Hub) or Private Repo(Nexus/JFrog/DTR. Etc.)).

Deployment

- Download the Docker image(package).
- Start the docker image which will create a container where our application will be running.

Advantages:

Application works seamlessly in any environment be it dev or test or prod.

Monolithic VS Microservices Architecture

Monolithic Architecture

Monolithic application has single code base with multiple modules. Modules are divided as either for business features or technical features. It has single build system which build entire application and/or dependency. It also has single executable or deployable binary.

Microservices Architecture

Microservices are a software development technique that arranges an application as a collection of loosely coupled services. Which can be developed, deployed, and maintained independently. Each of these services is responsible for discrete task and can communicate with other services through simple APIs to solve a larger complex business problem.

LANDMARK
TECHNOLOGIES

Monolithic vs. Micro Services

The following table shows a side-by-side comparison of important criteria between monoliths (the old way of architecting applications) and microservices (the new way).

Category	Monolithic architecture	Microservices architecture
Architecture	Built as a single logical executable	Built as a suite of small services
Modularity	Based on language features	Based on business capabilities
Agility	Changes involve building and deploying a new version of the entire application	Changes can be applied to each service independently
Scaling	Entire application is scaled when only one part is the bottleneck	Each service is scaled independently when needed
Implementation	Typically, entirely developed in one programming language	Each service can be developed in a different programming language
Maintainability	Large code base is intimidating to new developers	Smaller code bases are easier to manage
Deployment	Complex deployments with maintenance windows and scheduled downtimes	Simple deployment as each service can be deployed individually, with minimal downtime

Docker Key Concepts

Terminology:

Image: A read-only snapshot of a container that is stored in Docker Hub or in a private repository; you use an image as a template for building containers.

Container: The standard unit where the application service is deployed or running.

Registry: A repository that stores, distributes, and shares container images; it is available in software-as-a-service (SaaS) or in an enterprise to deploy anywhere you that you choose. An example of a popular registry is Docker Hub.

Docker Engine: A program that creates, ships, and runs application containers; the engine runs on any physical or virtual machine or server locally, in a private or public cloud. The client communicates with the engine to run commands.

Dockerfile: File that has a simple and descriptive set of instructions to create an image.

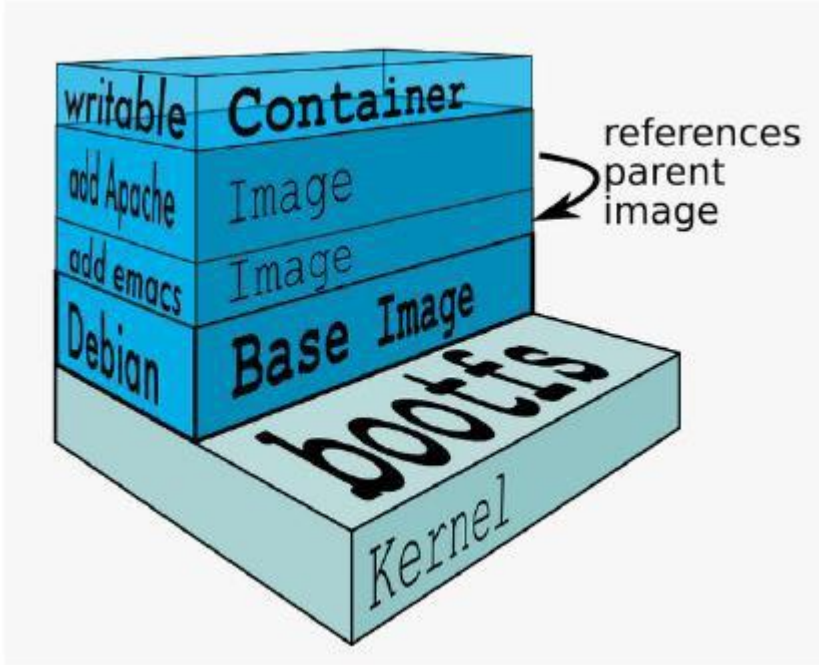
DockerHub: Public Registry which you can think of as an "app store for Docker images." Docker Hub has tens of thousands of public images created by the community that is readily available for use. It's incredibly easy to search for images that meet your needs, ready to pull down and use with little-to-no modification.
Link: https://hub.docker.com

Dockerfile

- Dockerfile is a file, it will describe the steps in order to create an image quickly.
- The Dockerfile uses a basic DSL (Domain Specific Language) with instructions for building Docker images.
- The Docker daemon runs the instructions in the Dockerfile are processed from the top down, so you should order them accordingly.
- The Dockerfile is the source code of the Docker Image.

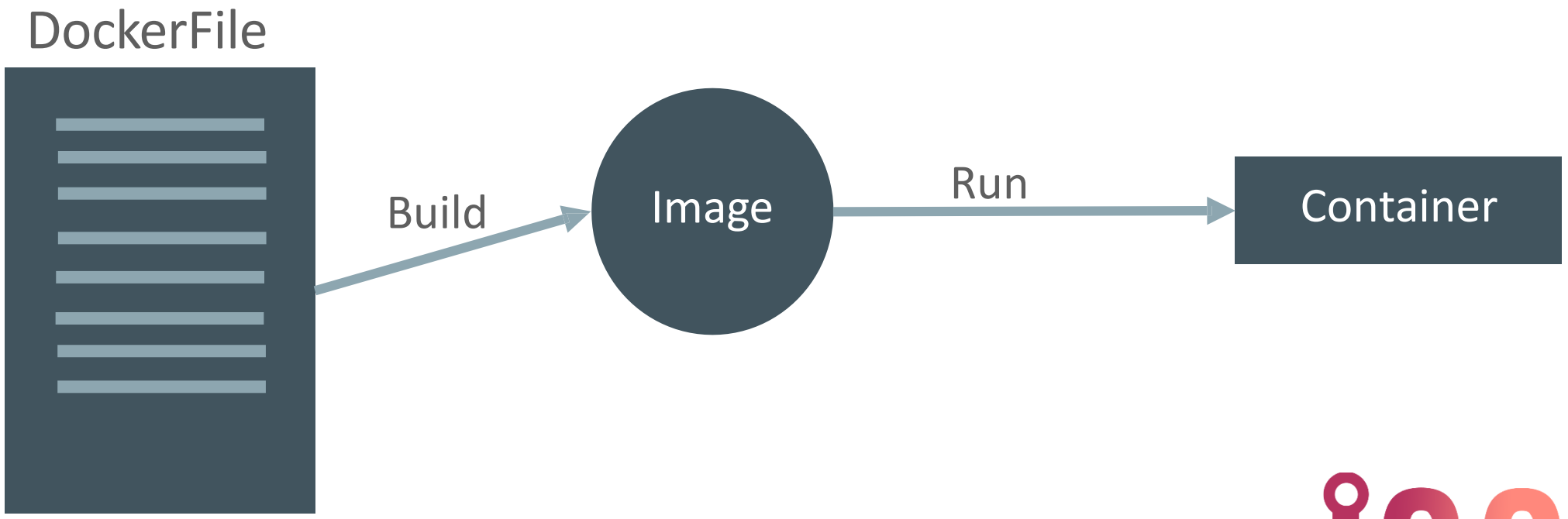
Docker Images

- An **image** is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software, including the code, a runtime, libraries, environment variables, and config files.



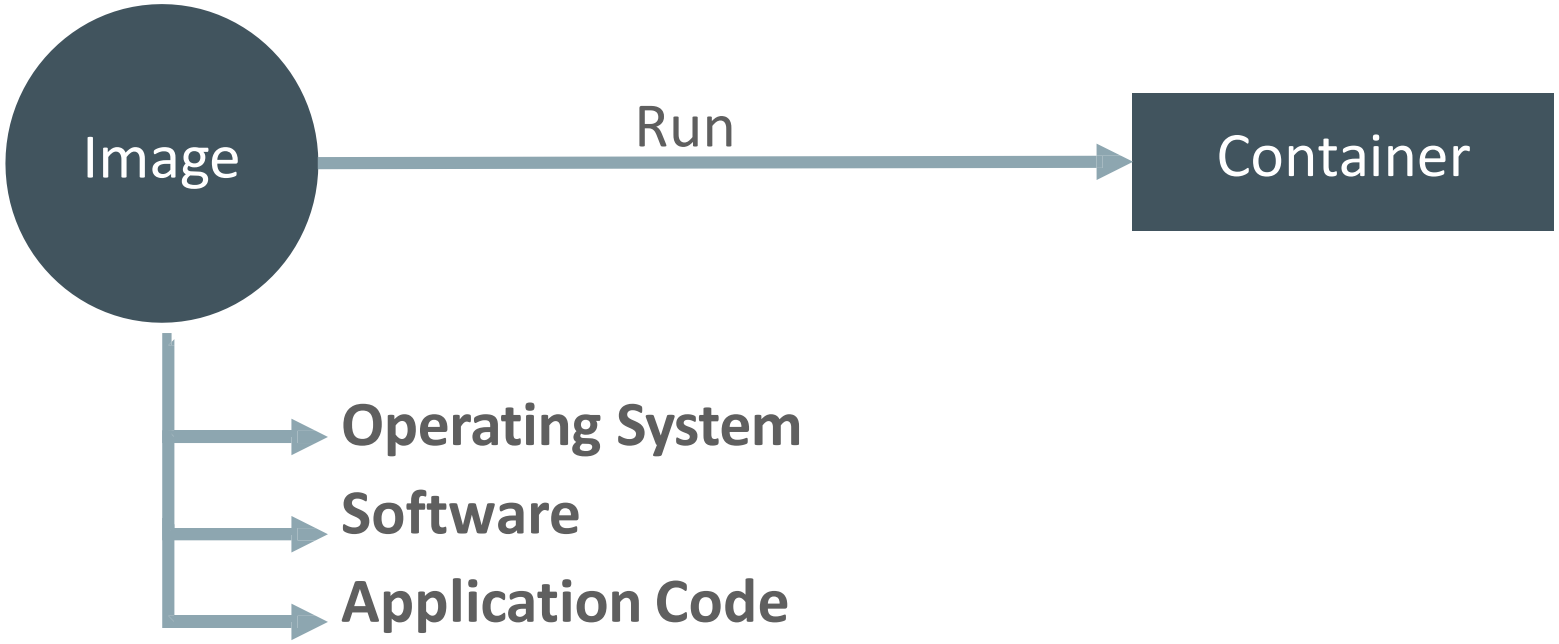
LANDMARK
TECHNOLOGIES

Docker Flow:



LANDMARK
TECHNOLOGIES

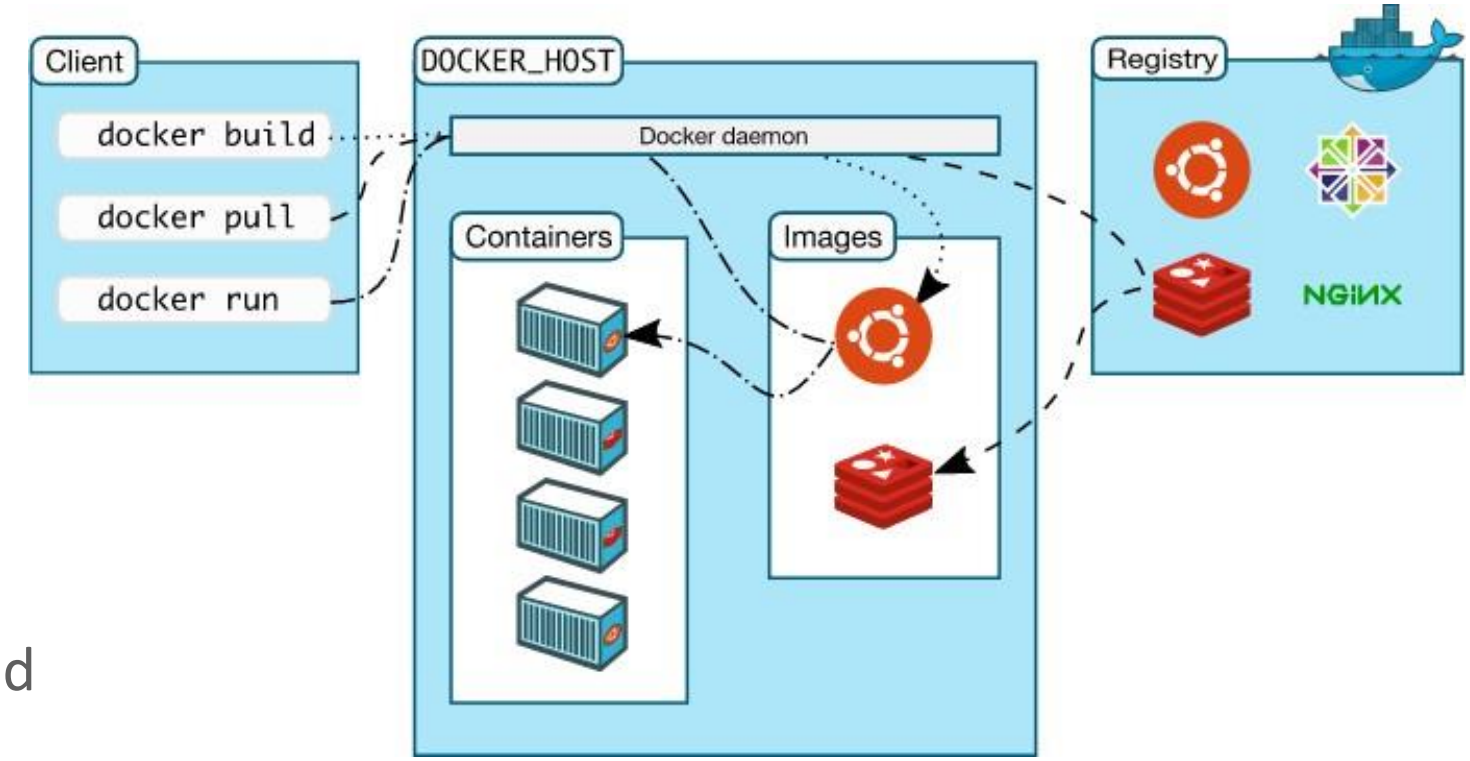
Docker Flow:



LANDMARK
TECHNOLOGIES

Docker Architecture

- Docker client – Command Line Interface (CLI) for interfacing with the Docker
- Dockerfile – Text file of Docker instructions used to assemble a Docker Image
- Image – Hierarchies of files built from a Dockerfile, the file used as input to the docker build command
- Container – Running instance of an Image using the docker run command
- Registry – Image repository



LANDMARK TECHNOLOGIES

Dockerfile ……

```
1  FROM ubuntu:12.04
2
3  RUN apt-get update
4
5  # Make it easy to install PPA sources
6  RUN apt-get install -y python-software-properties
7
8  # Install Oracle's Java (Recommended for Hadoop)
9  # Auto-accept the license
10 RUN add-apt-repository -y ppa:webupd8team/java
11 RUN apt-get update
12 RUN echo oracle-java7-installer shared/accepted-oracle-license-v1-1 select true | sudo /usr/bin/debconf-set-selections
13 RUN apt-get -y install oracle-java7-installer
14 ENV JAVA_HOME /usr/lib/jvm/java-7-oracle
```

- Like a Makefile (shell script with keywords)
- Extends from a Base Image
- Results in a new Docker Image
- Imperative, not Declarative
- A Docker file lists the steps needed to build an images
- docker build is used to run a Docker file
- Can define default command for docker run, ports to expose, etc

LANDMARK TECHNOLOGIES

Dockerfile – Text file (recipe) used to create Docker images

Example Hello World Dockerfile

```
FROM nginx:1.10.1-alpine
Add  index.html  /usr/share/nginx/html/index.html
# Override the nginx start from the base container
COPY start.sh /start.sh
RUN  chmod  +x  /start.sh
ENTRYPOINT ["/start.sh"]
```

Docker build image CLI example

```
$ docker build -t helloworld:1.0 .
```

NOTE: The "." references Dockerfile in local directory

Docker CLI – Common / useful commands

- docker build : build docker image from Dockerfile
- docker run : run docker image
- docker logs : show log data for a running or stopped container
- docker ps : list running docker containers (analogous to ps)
- docker ps –a : list all containers including not running
- docker images : list all images on the local volume
- docker rm : remove/delete a container | docker rmi : remove/delete an image
- docker tag : name a docker image
- docker login : login to registry
- docker push/pull : push or pull volumes to/from Docker Registries
- docker inspect : return container run time configuration parameter metadata

Why Containers?

Developers care because:

- Quickly create ready-to-run packaged applications, low cost deployment and replay
- Automate testing, integration, packaging
- Reduce / eliminate platform compatibility issues (“It works in dev!”)
- Support next gen applications (microservices)

IT cares because:

- Improve **speed** and frequency of releases, reliability of deployments
- Makes app lifecycle efficient, consistent and repeatable – configure once, run many times
- Eliminate environment inconsistencies between development, test, production
- Improve production application resiliency and scale out / in on demand

Questions ?

Thank You