Object Oriented Containers: Applying SOLID Principles to Docker/Container design

Dec 2018

Nitin Bhide

Shipping Container Changed Transportation

Malcom McLean after spending hours in late 1937 queuing at a Jersey City pier to unload his truck, he realized that it would be quicker simply to hoist the entire truck body on boa

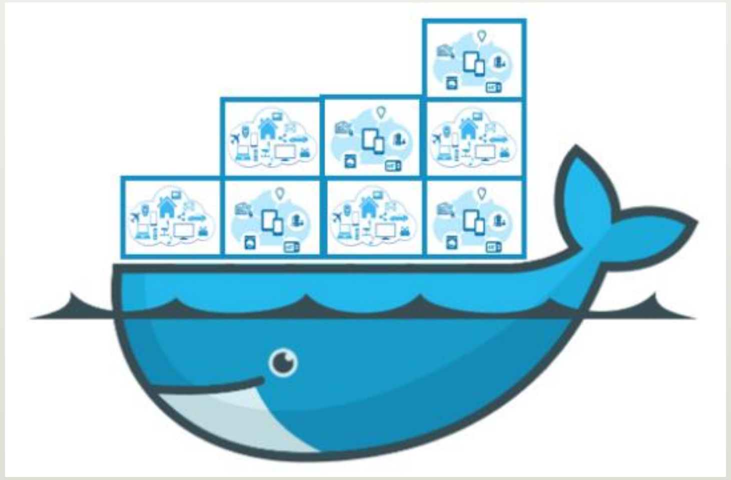
From this incident, came his decision eighteen years
buy a war-surplus tanker and equip it to carry 33-foot-
containers.

This insight started the silent resolution in transportati
‘Shipping Container’. Shipping Containers changed ‘scale’ of transportation operations.
By 2014, 20+ million containers in service.

Thinking Craftsman

Now Docker Containers are changing “Scale” of Software Operations

What is ‘Docker/Container’ ?



Docker performs operating-system-level virtualization, also known as "containerization". It was first released in 2013

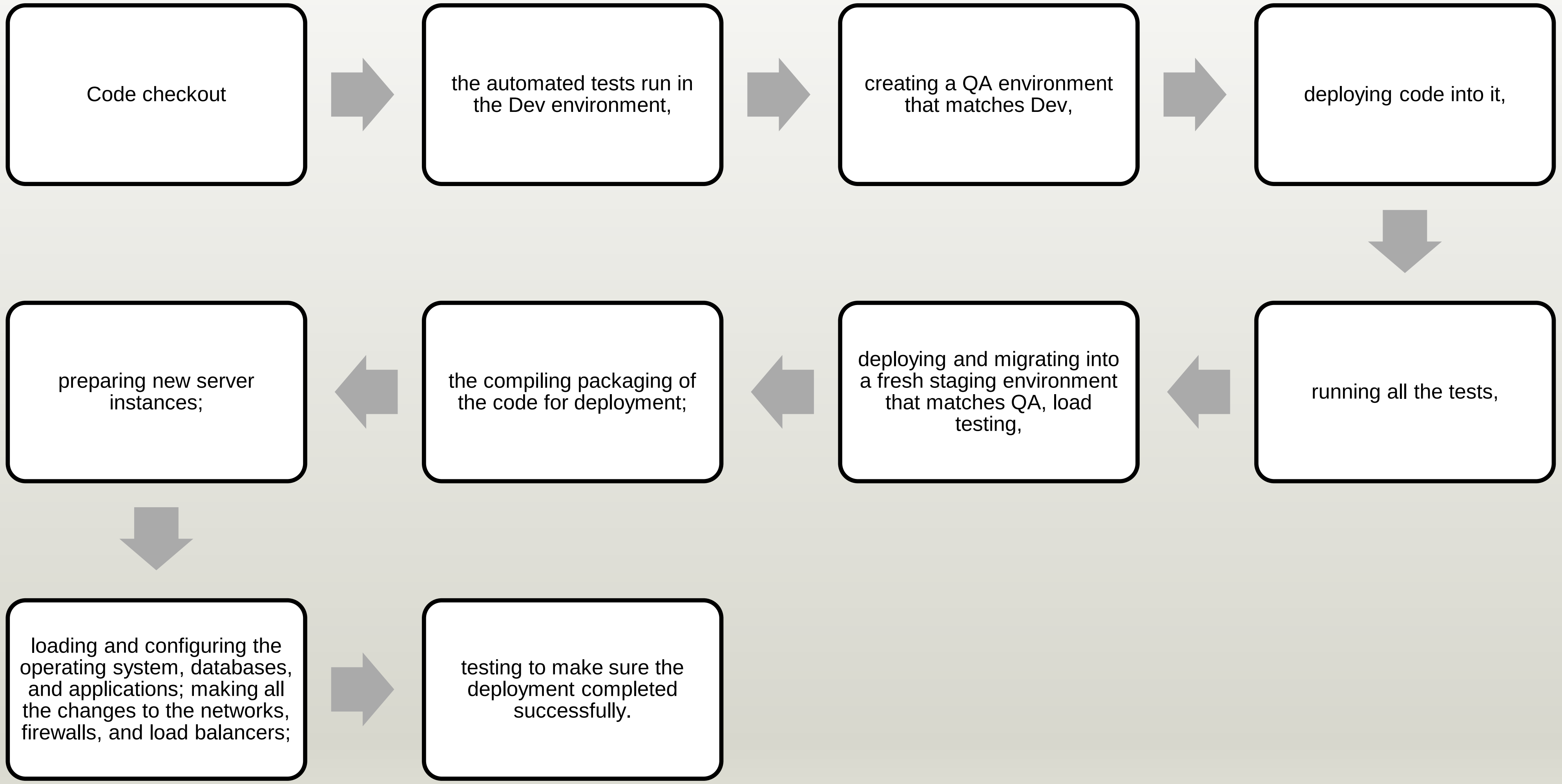
Docker is used to run software packages called "containers".

Containers are isolated from each other and bundle their own application, tools, libraries and configuration files;

Thinking Craftsman

What Problems Does the Docker/Container Solve ?

Lets look at typical cloud deployment scenario


Code checkout
the automated tests run in the Dev environment,
creating a QA environment that matches Dev,
deploying code into it,
running all the tests,
deploying and migrating into a fresh staging environment that matches QA, load testing,
the compiling packaging of the code for deployment;
preparing new server instances;
loading and configuring the operating system, databases, and applications; making all the changes to the networks, firewalls, and load balancers;
testing to make sure the deployment completed successfully.

Now Imagine that

you have to support

500+ developers and QA Enginers

to deploy their changes on ***100+ servers*** and

you are expecting at least ***10 such deploys every day***

Docker helps you deploy at SCALE

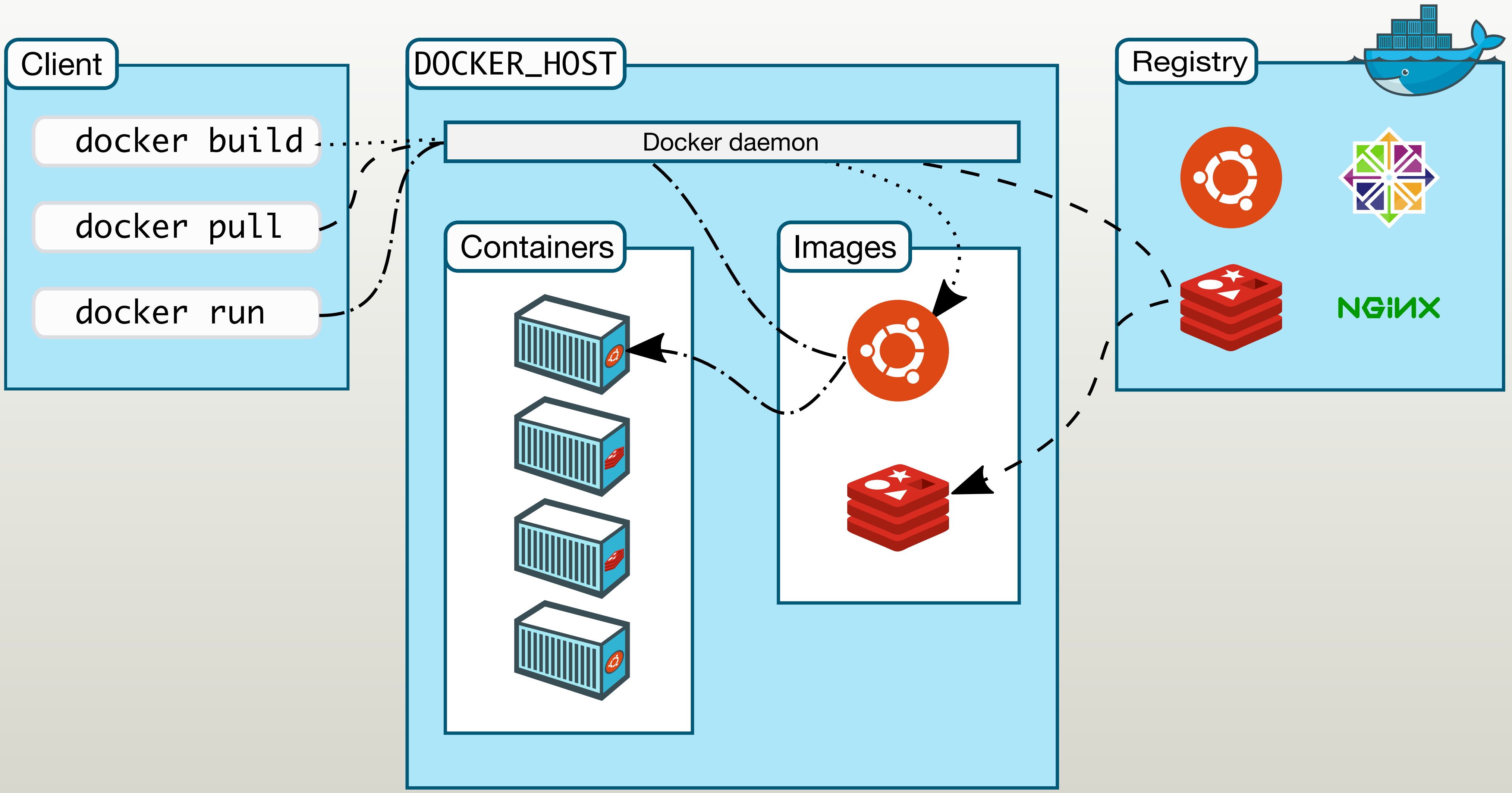
Docker allows you to ‘bundle’ specific version of OS, third party libraries, your own application, tools, and configuration files, network and storage in a ‘isolated container’

You have to just copy the container image on a computer and instantiate the container instance from the image

Now the deploying your application on 100 servers is just copy the container on each server.

(Of course, this is a really really simplified version)

Docker Architecture



Refer - https://docs.docker.com/engine/docker-overview/#docker-architecture

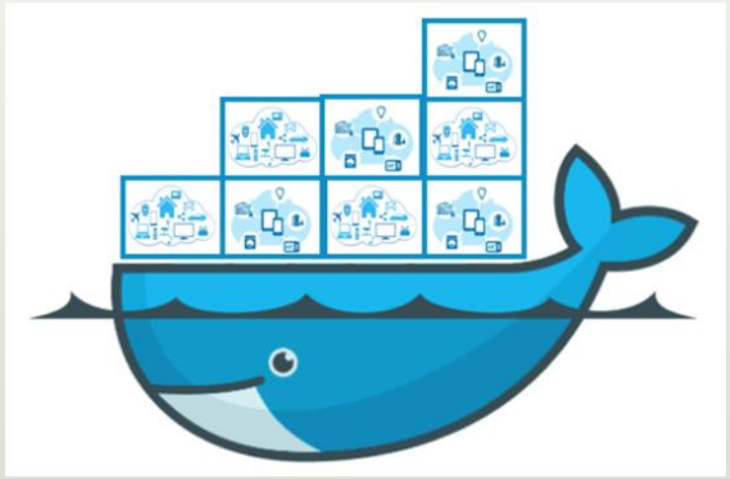
What is Docker (Container) ?

Containers can communicate with each other through well-defined channels.

All containers are run by a single operating system kernel and are thus more lightweight than virtual machines.

Containers are created from "images" that specify their precise contents.

Images are often created by combining and modifying standard images downloaded from public repositories



Problems that Containers are Solving

Managing *Complexity*

Managing *Scale*

Containers NOT really needed for 3-4 people development team deploying on 1 or 2 servers

Thinking Craftsman

Problems that OO Programming Solves

Managing *Complexity*

Managing *Scale*

We don’t use ‘classes’ for small simple shell scripts

OO Design help in developing large scale/complex software

Containers and OO Programming is about managing
‘Systems of made up of Objects’

In case Docker,
these are ‘individual containers’

In case of OO Programming,
these are ‘classes and modules’

OO Design Vs Container Design

OO Principles Solve 'Scale/Complexity' in Software Development	Docker/Containers try to Solve 'Scale/Complexity' in Software Deployment
OO Concepts are Developed in 1960s and 1970s	Containers are new. Docker is released in 2013
SOLID Design Principles are articulated by Robert Martin in 2000	Docker Best Practices are still being established
OO Best Practices are well established and with lot of theoretical and experimental background	Must less theoretical and experimental background

Thinking Craftsman

Can We apply OO Design Principles to Docker Design ??

Will it result in 'better' Container Design ?

For example

Docker

Google Recommends Following Best Practice as ‘Highly Important’

Package a single app per container

- *Because a container is designed to have the same lifecycle as the app it hosts, each of your containers should c[illegible]tain only one app.*

I think the reason “containers are designed to have same lifecycle as the app” is to ‘enforce’ the SRP

OO Programming

First principle in SOLID design principles

Single Responsibility Principle

- *A class should have one, and only one, reason to change.*

Lets See what we can learn with other OO Principles and apply it Docker

Thinking Craftsman

OO Principle - Encapsulation

Parnas in his Seminal Paper *"On the criteria to be used in decomposing systems into modules"* **(published 1972)** says

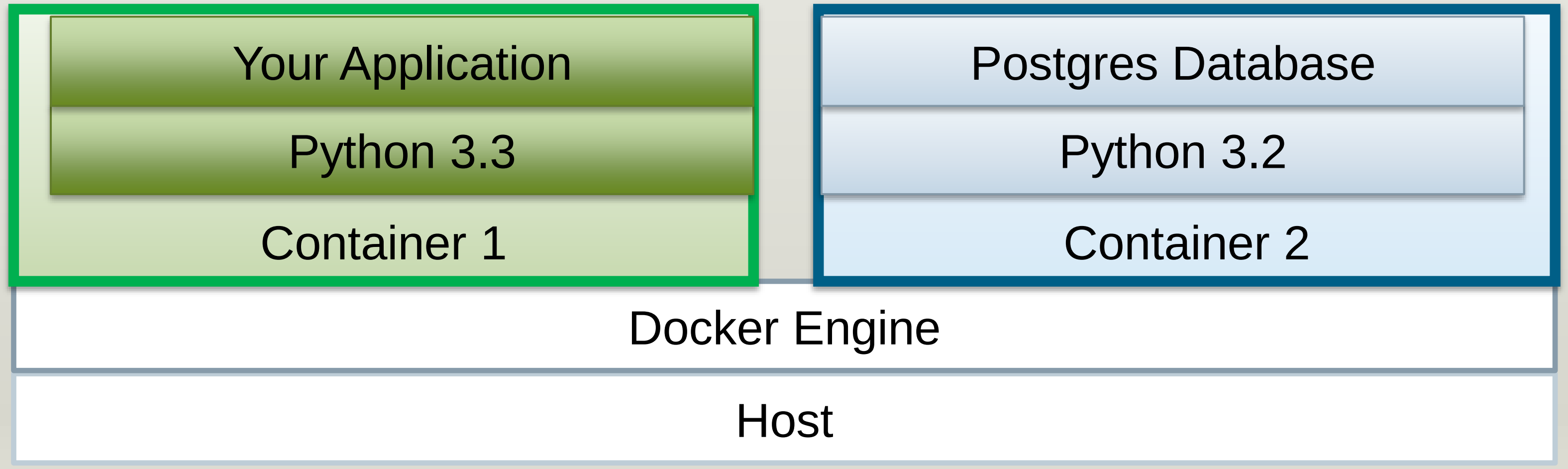
> *We have tried to demonstrate by these examples **that it is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart**. We propose instead that **one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others.***

Encapsulation is about Hiding Potential Change and NOT about Hiding Data

Encapsulation – with Containers

Docker Container is about **“Encapsulating”** the entire packaging and dependencies required for running a specific application

For example,
Your application requires Python 3.4 and your database server also uses python but requires Python 3.3.



OO Principle – Open/Closed Principle

“Software entities (classes, modules, functions, etc.) should be ***open for extension, but closed for modification”***

Adding new functionality will involve no change in the existing code.

The new functionality will be implemented as new class or new method

Open/Closed Principle – with Docker

Docker image is ‘immutable’ (closed for modification)

- once built, it’s unchangeable, and if you want to make changes, you’ll get a new image as a result.
- Basic design decision of Docker

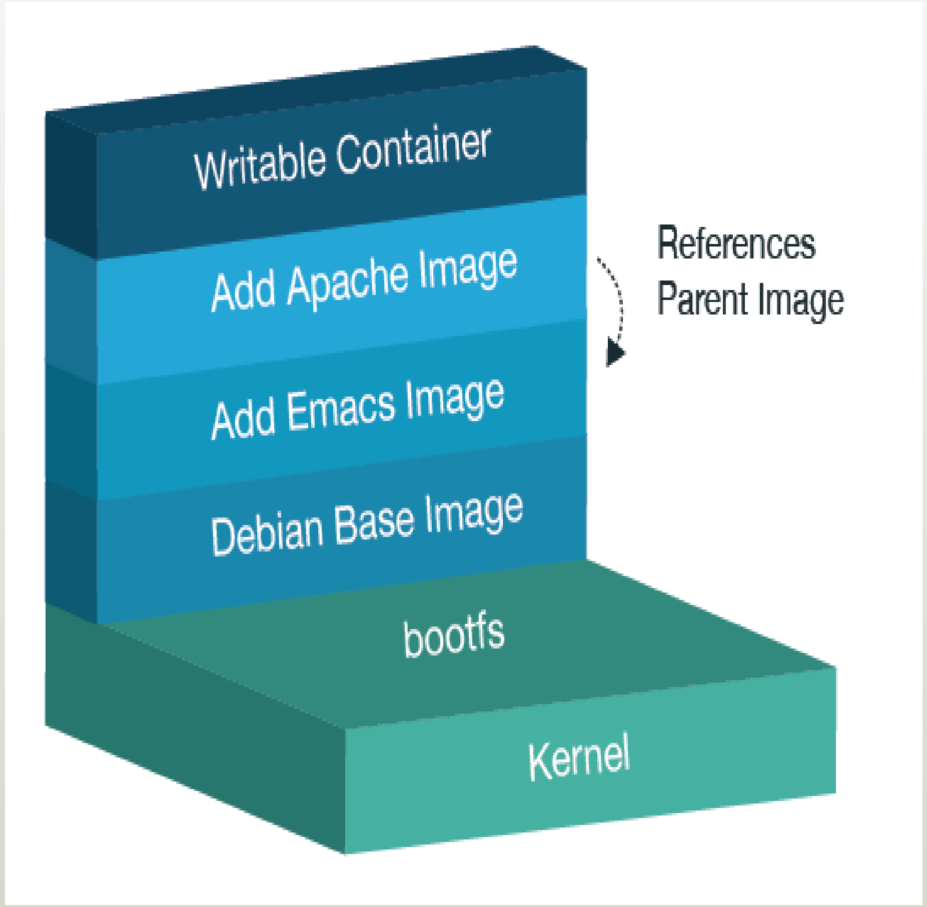
- **Benefits**
  - Immutability lowers rollback times because you can probably keep the old version running for a little longer and switch traffic in case of problems.
  - Brings in Predictability and Portability

Thinking Craftsman

Open/Closed Principle – with Docker

Docker image is Multi Layered i.e. inherited (Open for extension)

- You can create new docker images from a parent image
- It allows you create 'predictable' deployment as the 'base image' can be common across various pieces of deployment
- Developer, QA Engineer and Production images can be 'derived' from common base image with just different settings/environment variables



OO Principle – Liskov Substitution & Inheritance

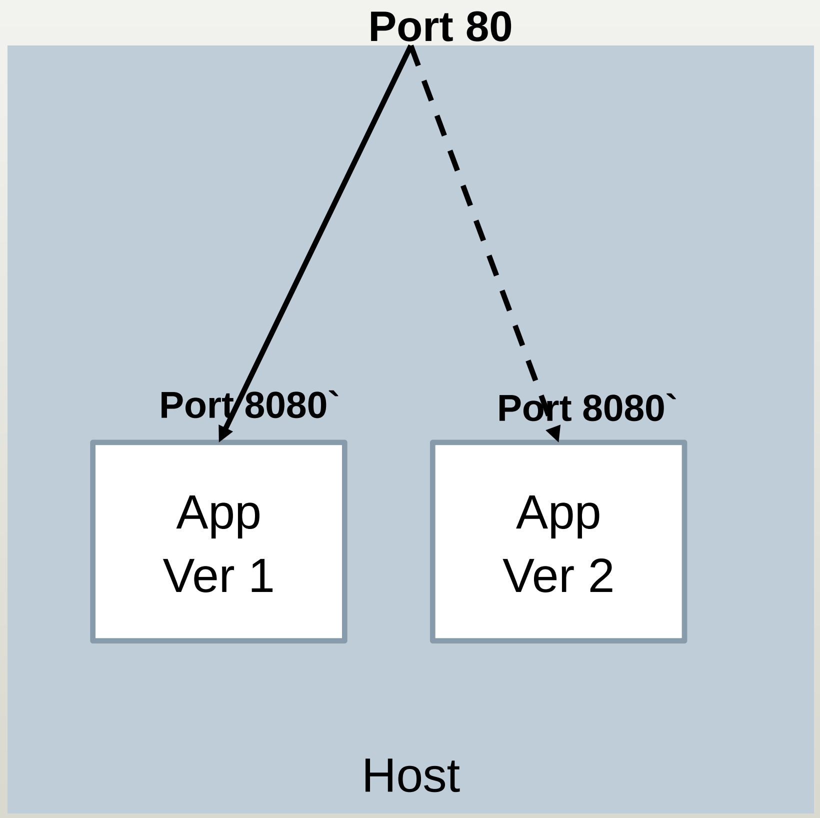
In class hierarchies,
subclasses should be substitutable for their base classes

- All code operating with a pointer or reference to the base class should behave the same when made to work with an object of derived class
- Inheriting classes should not perform any actions that will invalidate the assumptions made by the base class
- The derived class should not expect anything more & must not guarantee anything less than that of the base class.

LSP & Inheritance – with Docker

Common Docker Use cases are possible ONLY if you are following LSP

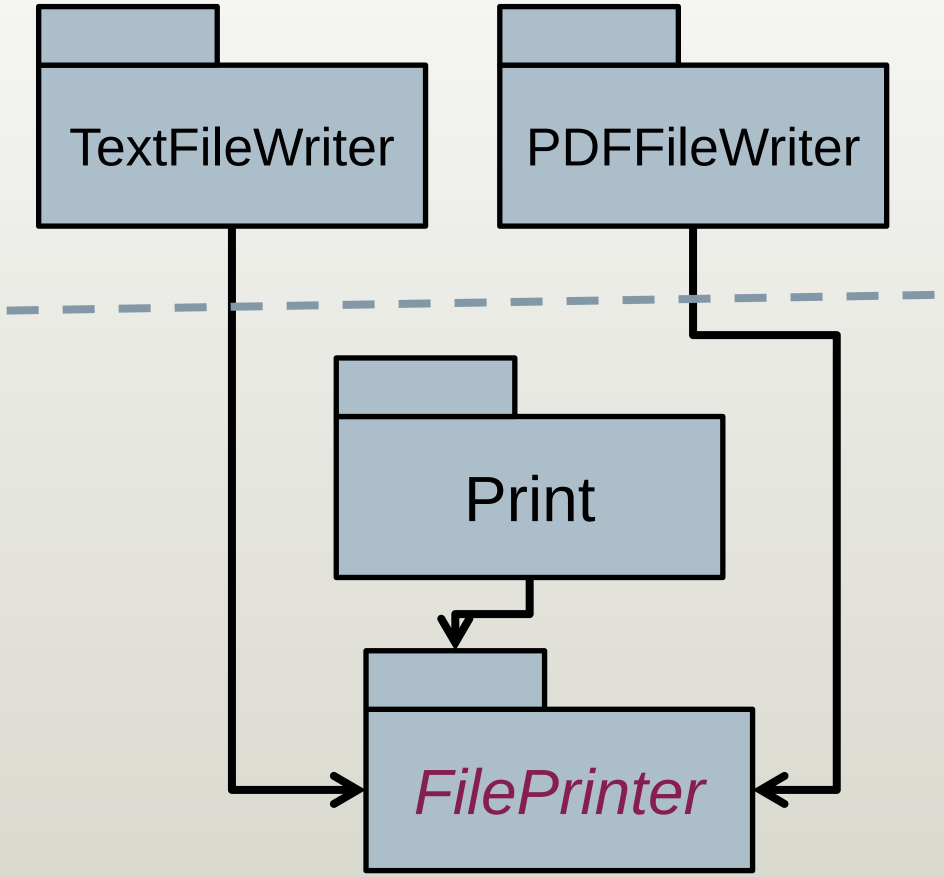
- Deployment and Rollback
- Fast Deployment by Swapping Containers
- A/B Testing



OO Principle – Dependency Inversion

- High-level modules should not depend on low-level modules. Both should depend on abstractions.

- Abstractions should not depend on details. Details should depend on abstractions.



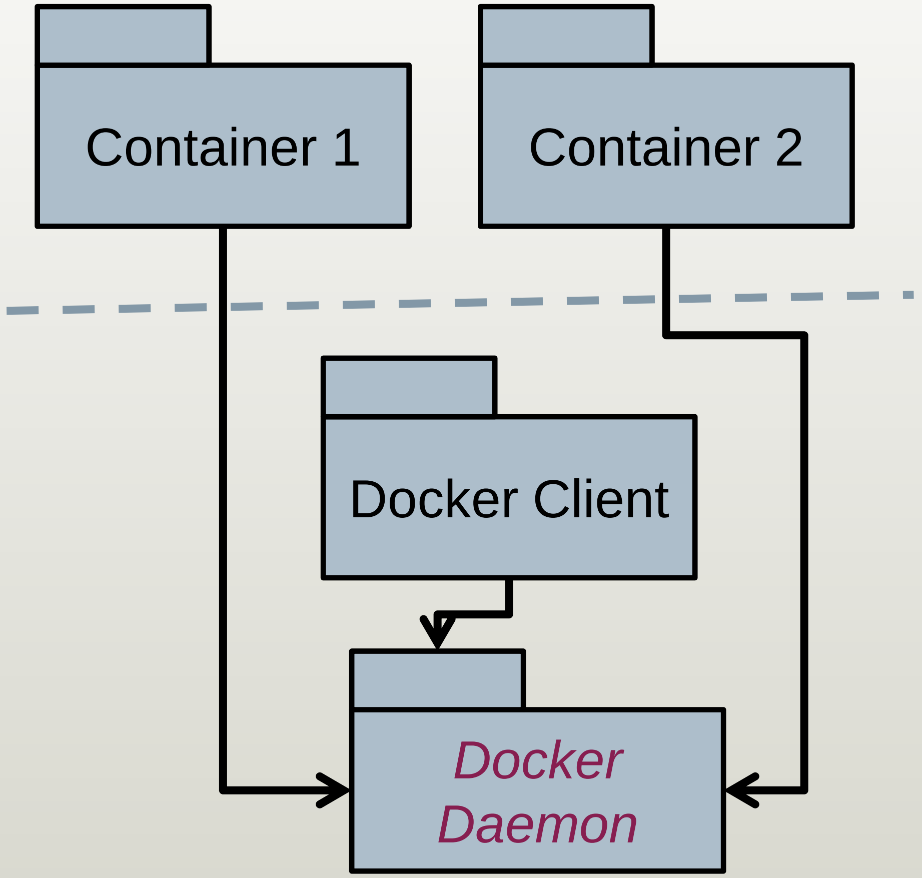
Dependency Inversion : With Docker

Docker Design Decision :

- Docker Clients CANNOT directly communicate with Containers
- Docker clients will communicate to Docker Daemon and Daemon will communicate with Containers
- This way both ‘Containers’ and ‘Clients’ depend on ‘Docker Daemon’ and NOT directly on each other

Best Practices

- Containers should NOT directly depend on each other.
- Containers should communicate using service registry



Dependency Inversion : with Docker

From OpenShift Docker Guideline

Use Services for Inter-image Communication
For cases where your image needs to communicate with a service provided by another image, such as a web front end image that needs to access a database image to store and retrieve data, your image should consume an OpenShift service. Services provide a static endpoint for access which does not change as containers are stopped, started, or moved.

Docker instance themselves do not provide this capability, OpenShift is providing it. You may have to use something 'ZooKeeper' or other Service Registration and Discovery layer

Food for Thought

Thinking Craftsman

Can we map to Docker/containers to Other OO Ideas ?

- **Other OO Principles**
  - Acyclic Dependency Principle
  - The Stable Dependencies Principle
  - The Stable Abstractions Principle
  - The Release Reuse Equivalency Principle
- **Guideline like “Prefer composition over inheritance”**
- **Design Patterns e.g. Proxy and Mediator**
- **Refactoring monolithic deployments to modular deployments**

Thinking Craftsman

Object Oriented Docker/Containers

Nitin Bhide
Thinking Craftsman/Passionate Programmer

Website : http://thinkingcraftsman.in

E-mail: nitinbhide@thinkingcraftsman.in