Docker & Containers

Vikram Krishnamurthy

Agenda

- ***History – cgroups, namespaces, overlayfs***
- ***What is/are Docker/Containers, what problems do they solve?***
- ***How do Containers differ from Hypervisor based virtualization?***
- ***What difference does Docker bring to Containers?***
- ***How does Docker work fundamentally?***
- ***Demo Time!!***

Resource Isolation

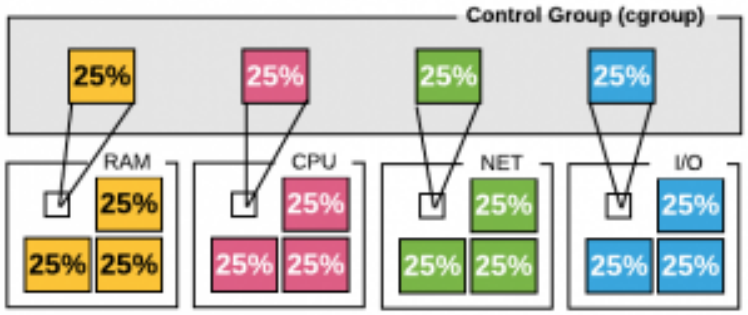
- ***Compute Resources:***
  - ***CPU – Consume and Release***
  - ***Memory – Consume and Release***
  - ***N/W – Consume and Release***
  - ***Disk/Secondary Storage – Consume and Hold***

Namespaces

- **Namespaces** are kernel enforced user space views.
- They are mechanisms to abstract, isolate, and limit the visibility that a group of processes has over various system entities such as process trees, network interfaces, user IDs, and filesystem mounts.
- Namespace Types:
  - Process ID – Allows a process to spin off a new process tree
  - Mount – confine set of mount points visible within a process namespace
  - UTS – isolating systems host and domain name
  - IPC – demarcate processes from using System V and POSIX message queries
  - User – allow a process to use unique UID and GID within and outside namespaces
  - Network – abstraction of network protocol services and interfaces
  - Cgroups – virtualizes content of /proc/self/cgroup file.

Cgroups

- Cgroups limit the amount of resources a process can consume (CPU, memory, network bandwidth, etc.). They were introduced in Linux kernel 2.6.24



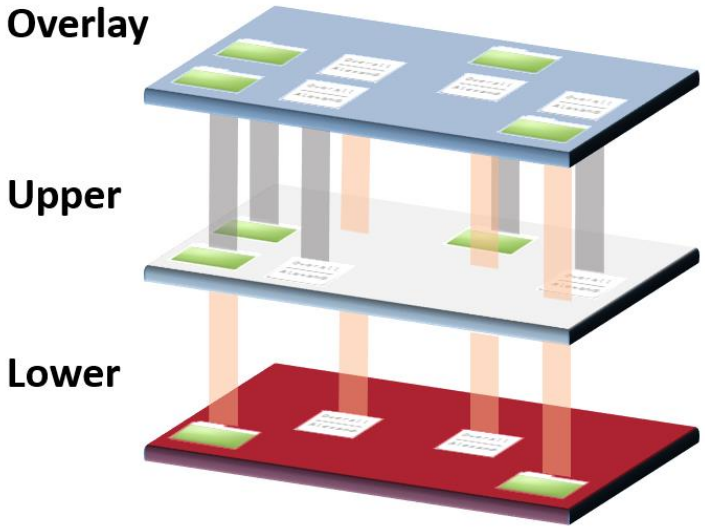
- Applying cgroups on namespaces allows isolation of processes into containers within a system, where resources can be managed distinctly.

OverlayFS

- Union file systems are a creative solution to allow a virtual merge of multiple folders, while keeping their actual contents separate. The Overlay file system (OverlayFS) is one example of these, though it is more of a mounting mechanism than a file system.
- OverlayFS allows you to overlay

  the contents (both files and directories)

  of one directory onto another
- *Lower* directory is Read only, *Upper* directory is RW

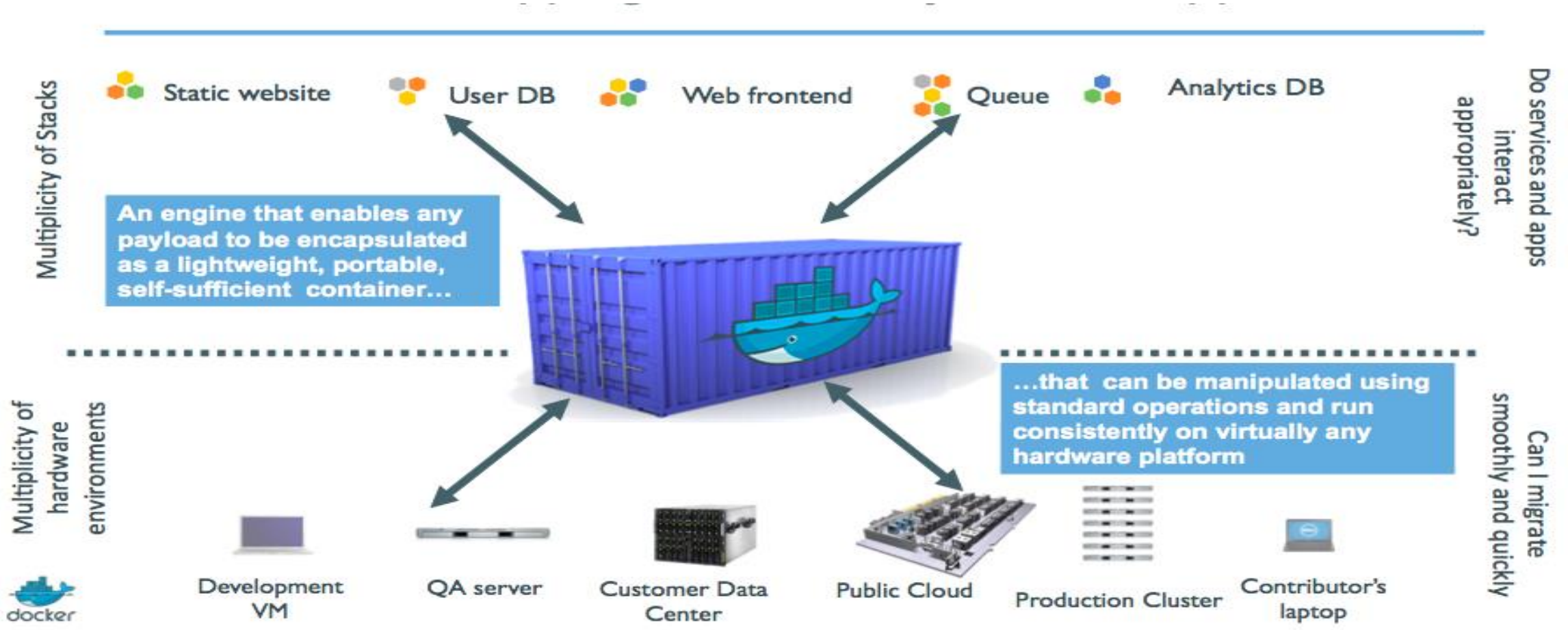


The Challenge – The Matrix From Hell

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

Help From Elsewhere - Shipping Containers

Applied to IT World - Application Containers



Containers vs Processes vs VMs

Process

- Isolate address space
- No isolation for files or networks
- Lightweight

Virtual Machine

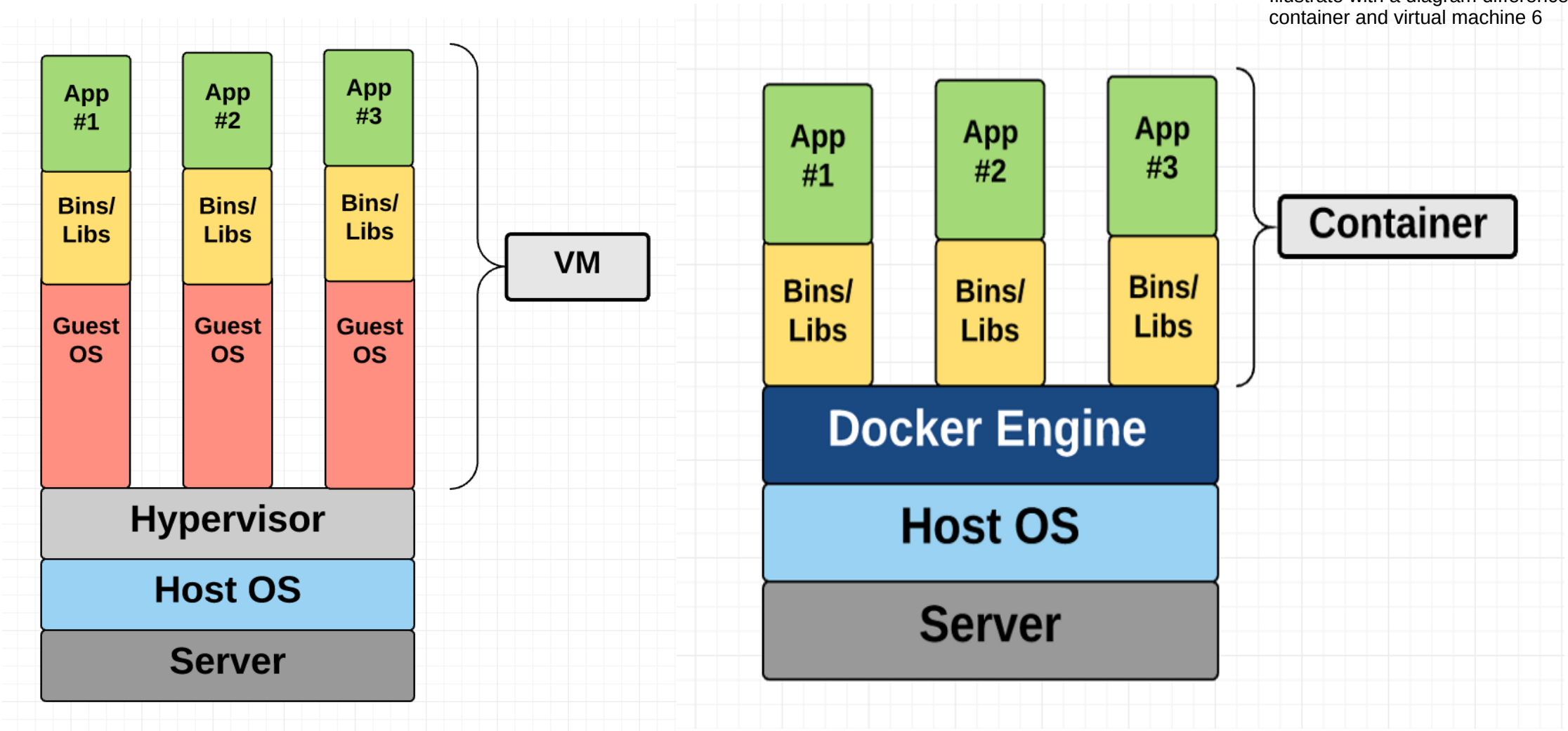
- Isolate address space
- isolate files and networks
- Heavyweight

Container

- Isolate address space
- isolate files and networks
- Lightweight

Containers vs Virtual Machines?

Illustrate with a diagram differences between container and virtual machine 6



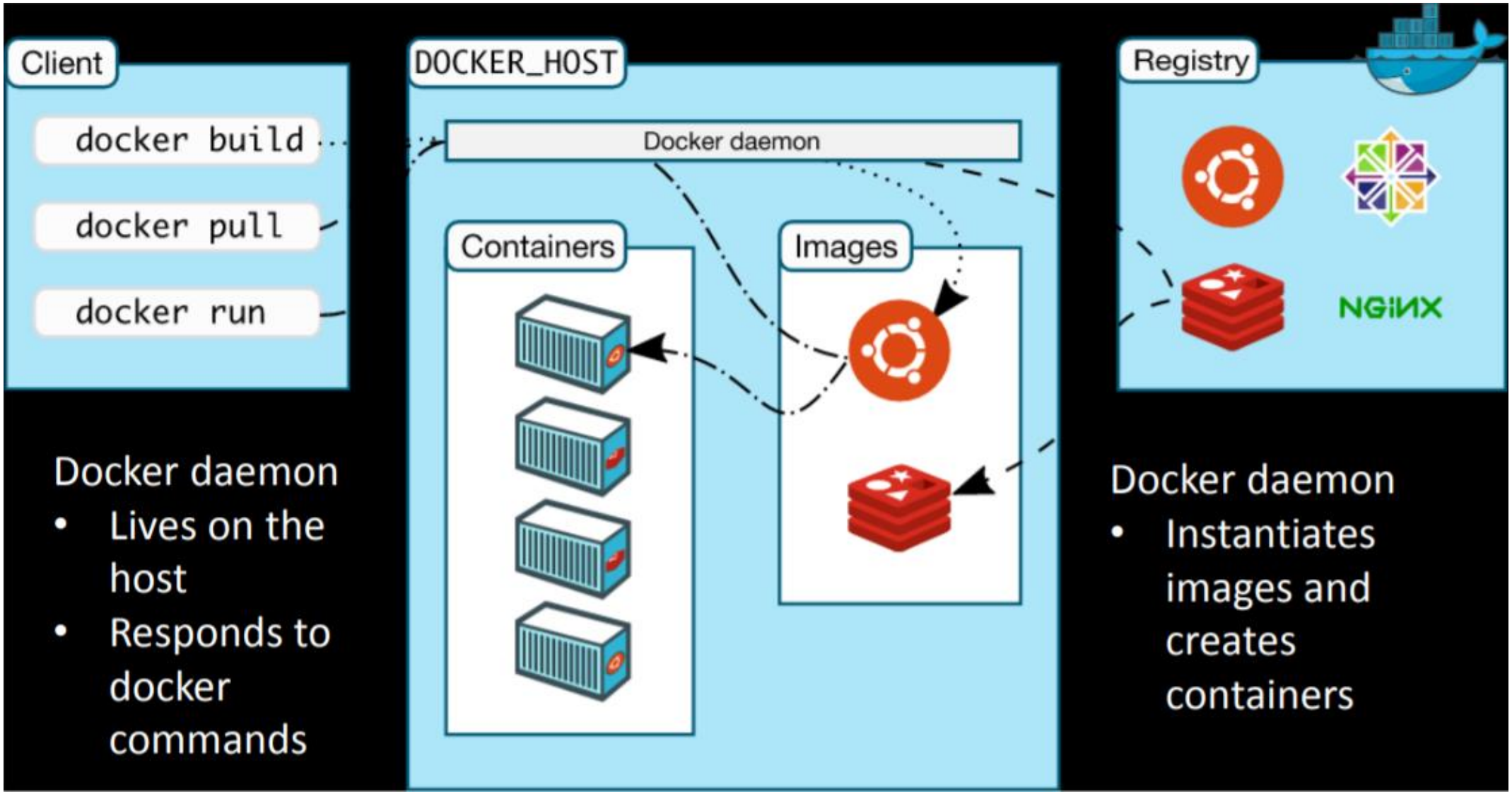
Hypervisors virtualize hardware, Containers virtualize OS!!

Why Should We Care about Containers?

- Build once, run *almost* anywhere
- A clean, safe, portable runtime environment for the app.
- Eliminate worries about missing dependencies, packages or compatibility between different platforms.
- Run each app in its own isolated container, so various versions of libraries and app can be run without conflicts.
- Automate testing, integration, packaging
- A VM without the overhead of a VM.

Docker Basic Architecture

With a neat diagram explain the architecture of dockers

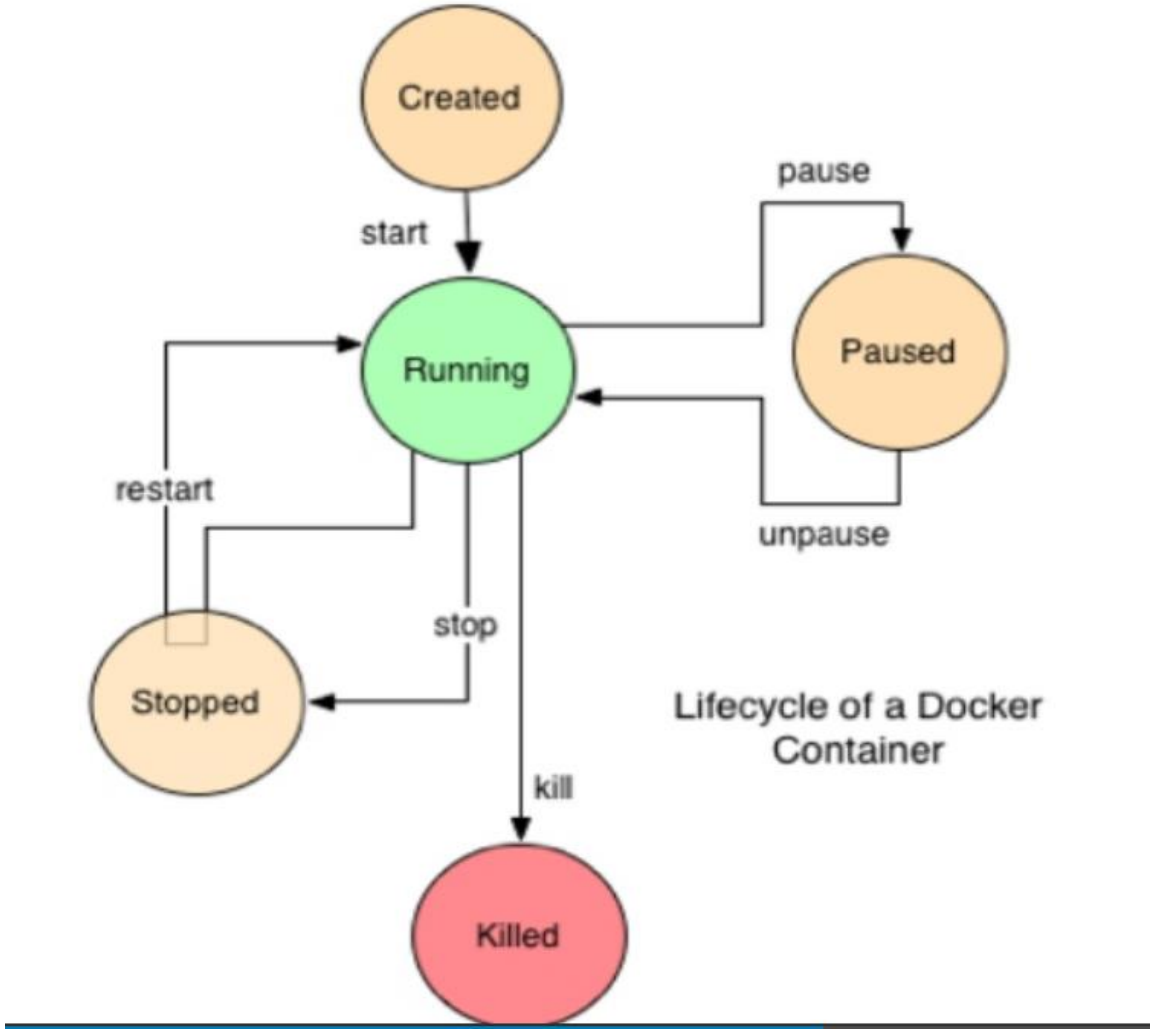


Docker – Important Terminology

Define the following terms:
i) Image ii) Container iii) Dockerfile iv)Docker Client v) Docker Daemon/Engine

- Image: Persisted snapshot of application runtime and dependent binaries/libraries that can be «run»
- Container: Live running instance of a Docker «image»
- Dockerfile: A text document with commands to build Docker "image".
- Docker Client : The utility that runs docker commands – *docker run*, *docker ps, docker build* etc.
- Docker Daemon/Engine: The server part that talks to the kernel, makes the system calls to create, operate and manage containers.

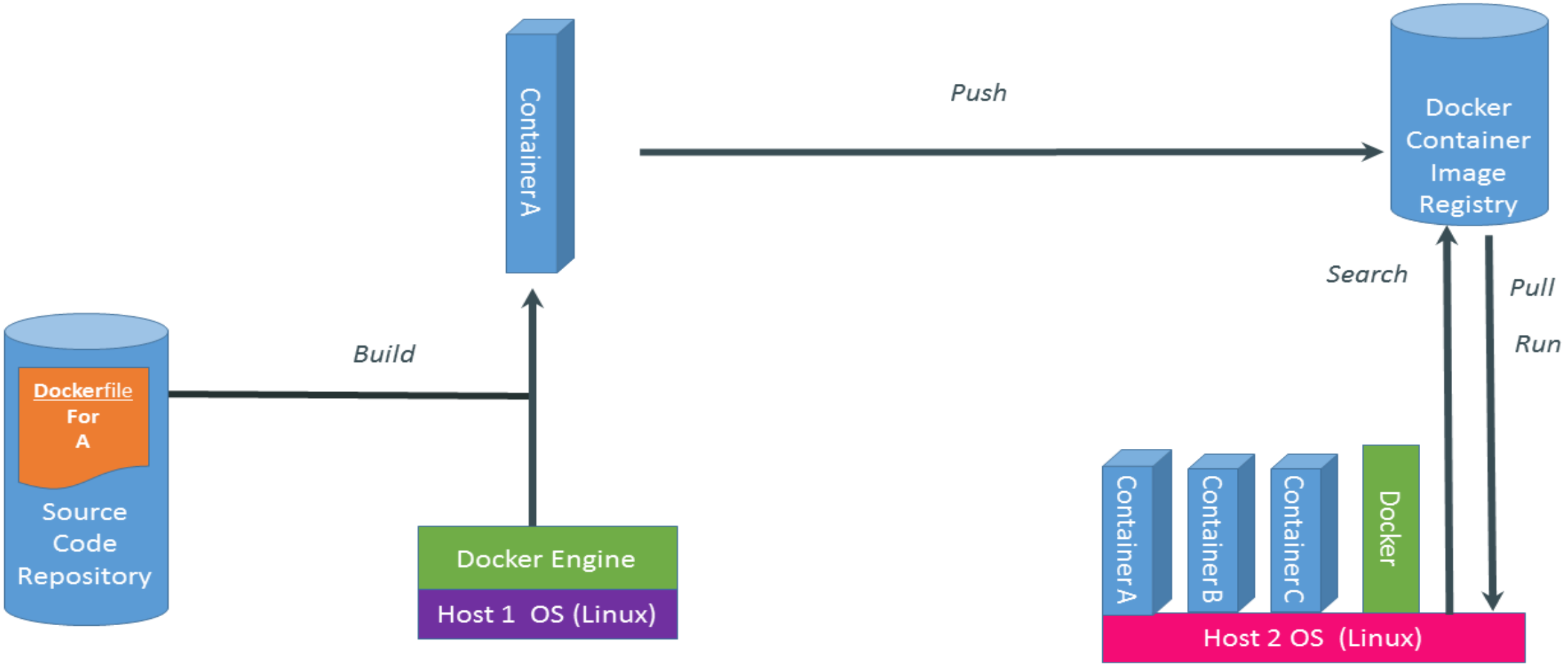
Docker Container Life Cycle



With a neat diagram describe Docker Container Life Cycle.

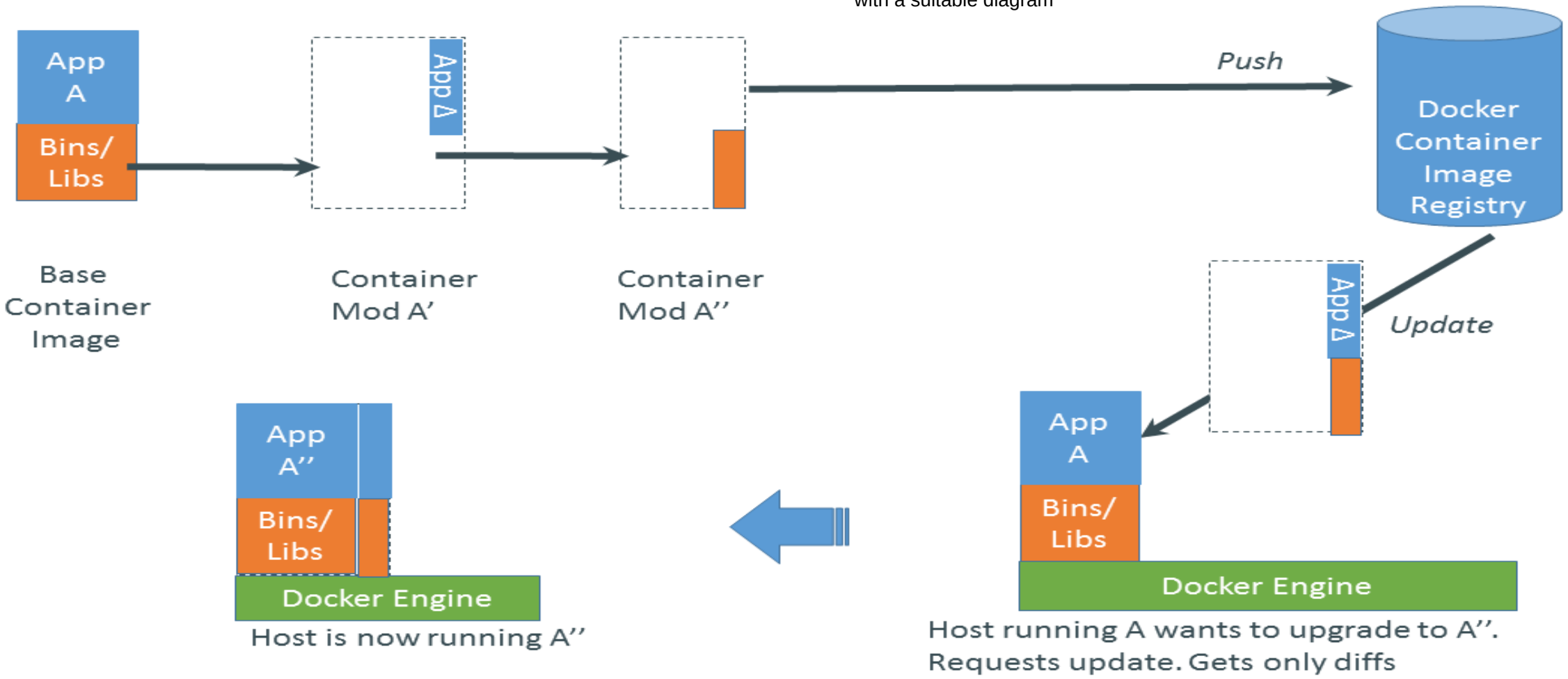
Docker Workflow - Basics

Illustrate basic workflow of the docker with a suitable diagram



Docker Workflow – App Updates / Changes

Illustrate workflow of the docker with App Updates / Changes with a suitable diagram



Docker – Examples

- Basics
  - docker –version/version
  - docker pull
  - docker images
- Container lifecycle
  - docker run
  - docker ps
  - docker exec
  - docker pause/stop/kill/rm
  - docker unpause/restart

Thank You.