Docker Container Fundamentals

Solutions Lab

Digital Networking

Sep 2019

NXP | SECURE CONNECTIONS FOR A SMARTER WORLD

AGENDA

- Introduction to Docker
- Docker Architecture
- Docker vs. Virtual Machine
- Docker Objects
  - Images
  - Containers
- Layers
- Docker Features
- Terminology
- Docker Command Examples
- References

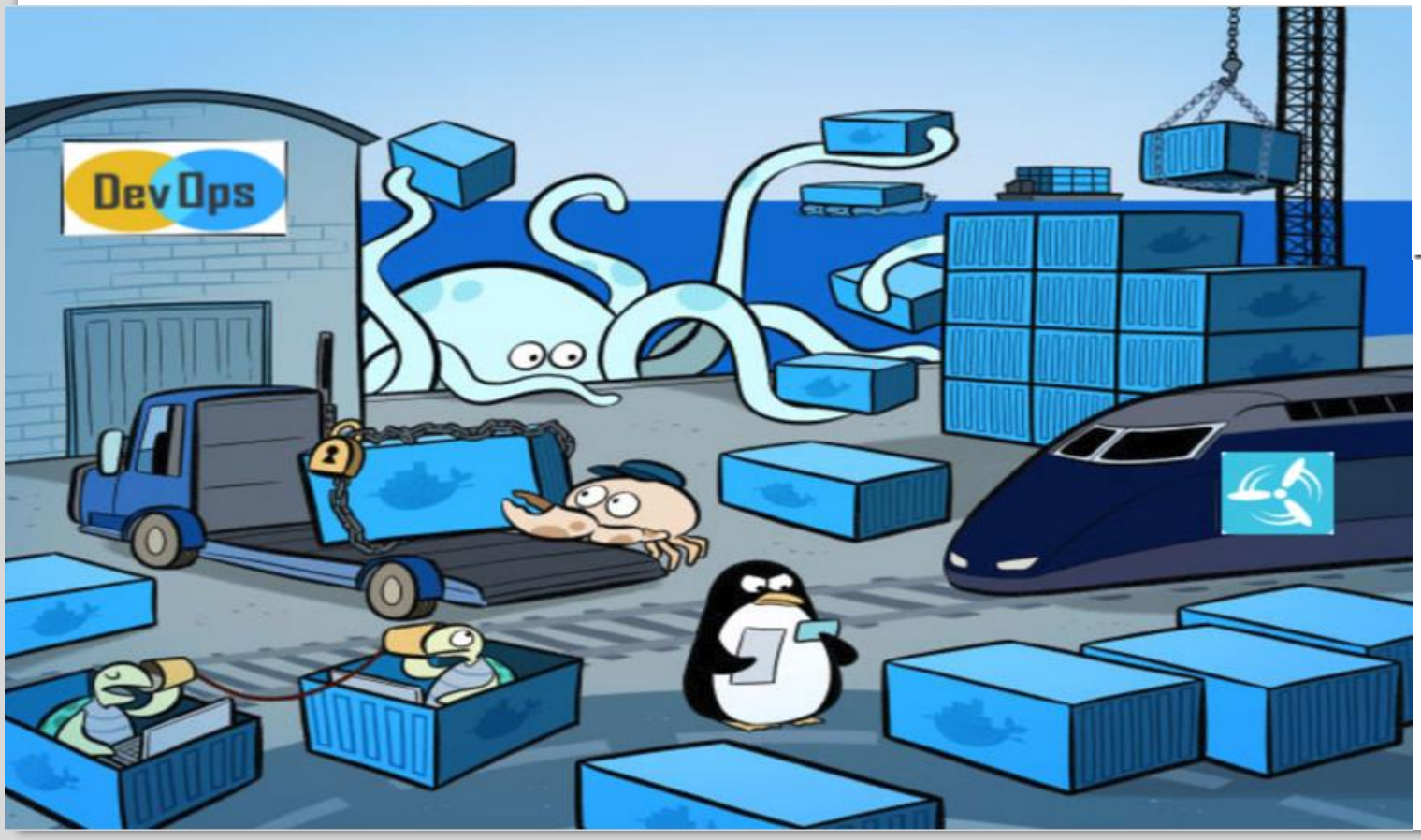
Prerequisites

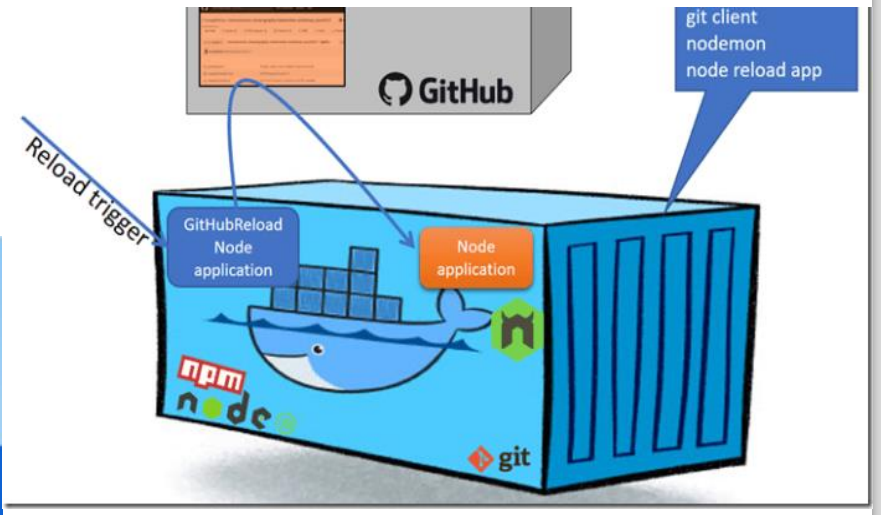
- There are no specific skills needed for this tutorial beyond:
  - A basic comfort with the command line
  - Using a text editor.

What is Docker?

- An open platform for developing, shipping, and running applications
- Separate applications from infrastructure so one can deliver software quickly
- Provides the ability to package and run an application in a loosely isolated environment called a container
- Docker provides tooling and a platform to manage the lifecycle of the containers:
  - Develop the application and its supporting components using containers.
  - The container becomes the unit for distributing and testing the application.
  - When a container is ready, deploy the application into the production environment, as a container. This works the same whether the production environment is a local data centre, a cloud provider, or a hybrid of the two.

What Docker Provides?

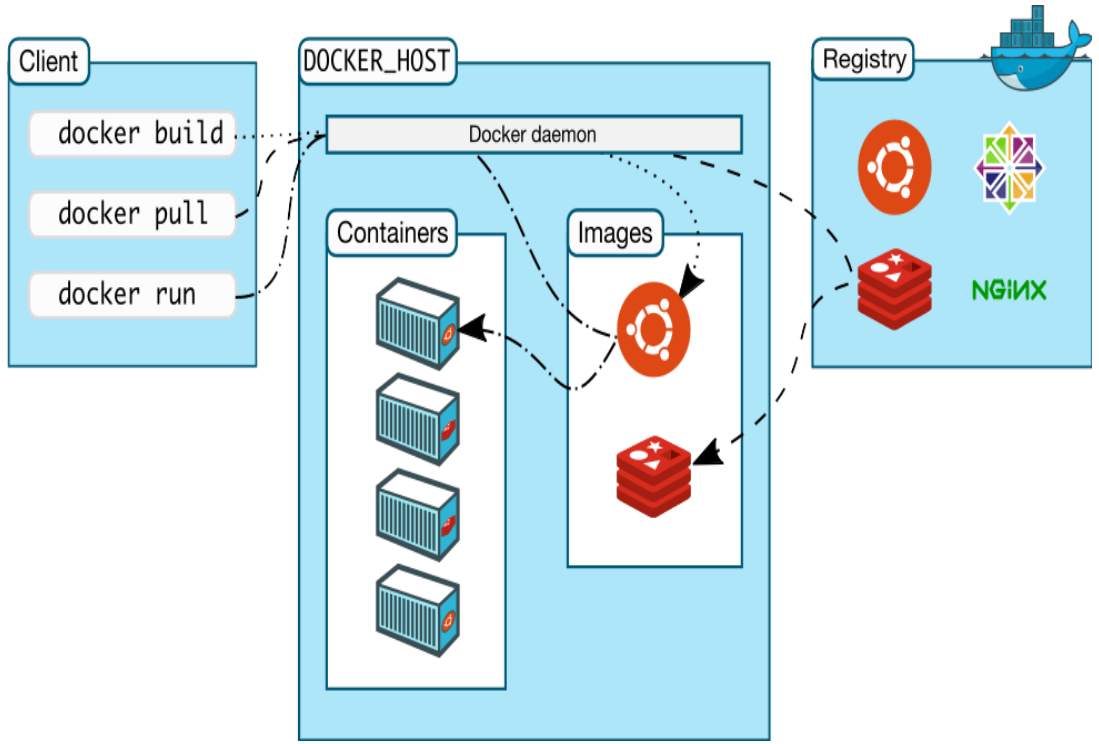




- Flexibility
- Lightweight
- Interchangeability
- Portability
- Scalability
- Stack ability

Docker Architecture

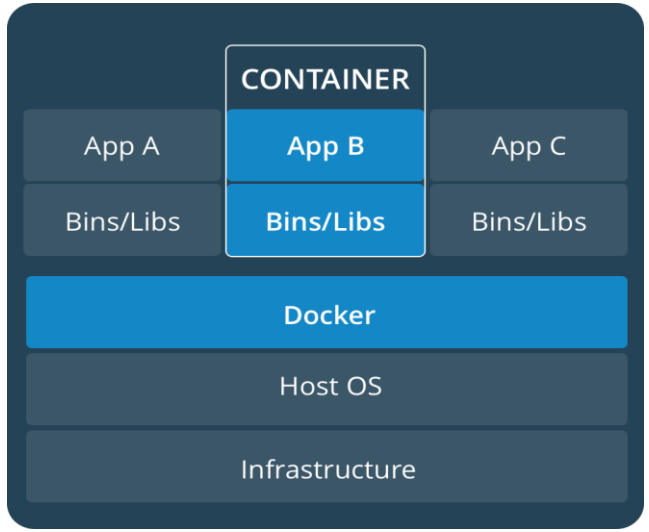
- **Client-Server Architecture**: The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers.
    - The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon.
    - The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.
- **Docker Daemon**(dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.
- **Docker Client**(docker) is the primary way that many Docker users interact with Docker. When *docker run* commands used, the client sends these commands to dockerd.

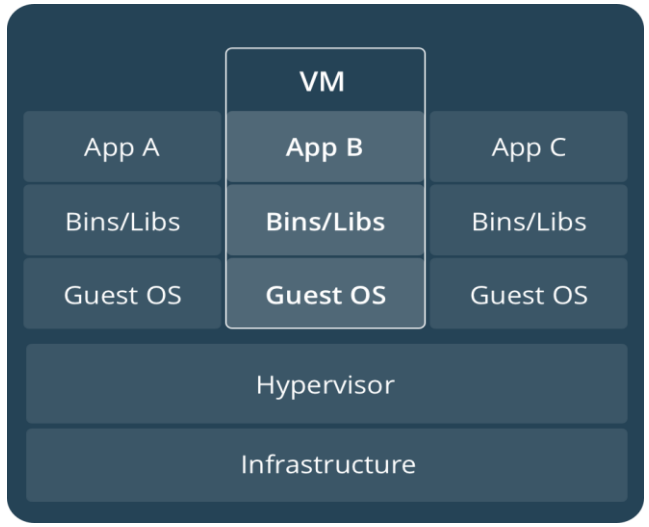


Container vs. Virtual Machine

- **A Container** runs natively on Linux and shares the kernel of the host machine with other containers.
  - Runs a discrete process, taking no more memory than any other executable, making it lightweight.
  - Sit on top of a physical server and its host OS - typically Linux or Windows.
  - Each container shares the host OS kernel and, usually, the binaries and libraries, too.

- **A Virtual machine** (VM) runs a full-blown “guest” operating system with virtual access to host resources through a hypervisor.
  - VMs provide an environment with more resources than most applications need.
  - The OS and their applications share hardware resources from a single host server, or from a pool of host servers.
  - Each VM requires its own underlying OS, and the hardware is virtualized.



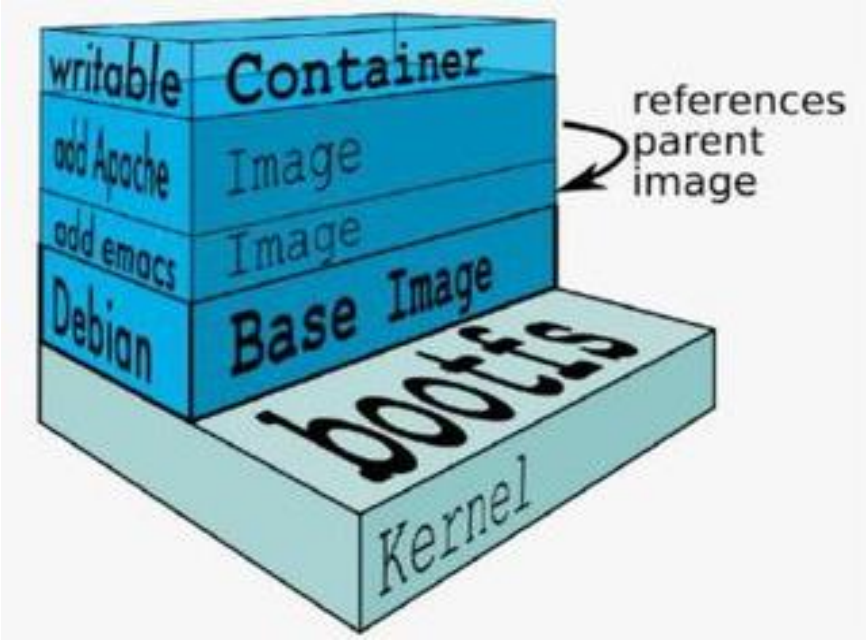


Docker Objects

When using Docker, it is creating and using images, containers, networks, volumes, plugins, and other objects.

What are IMAGES:

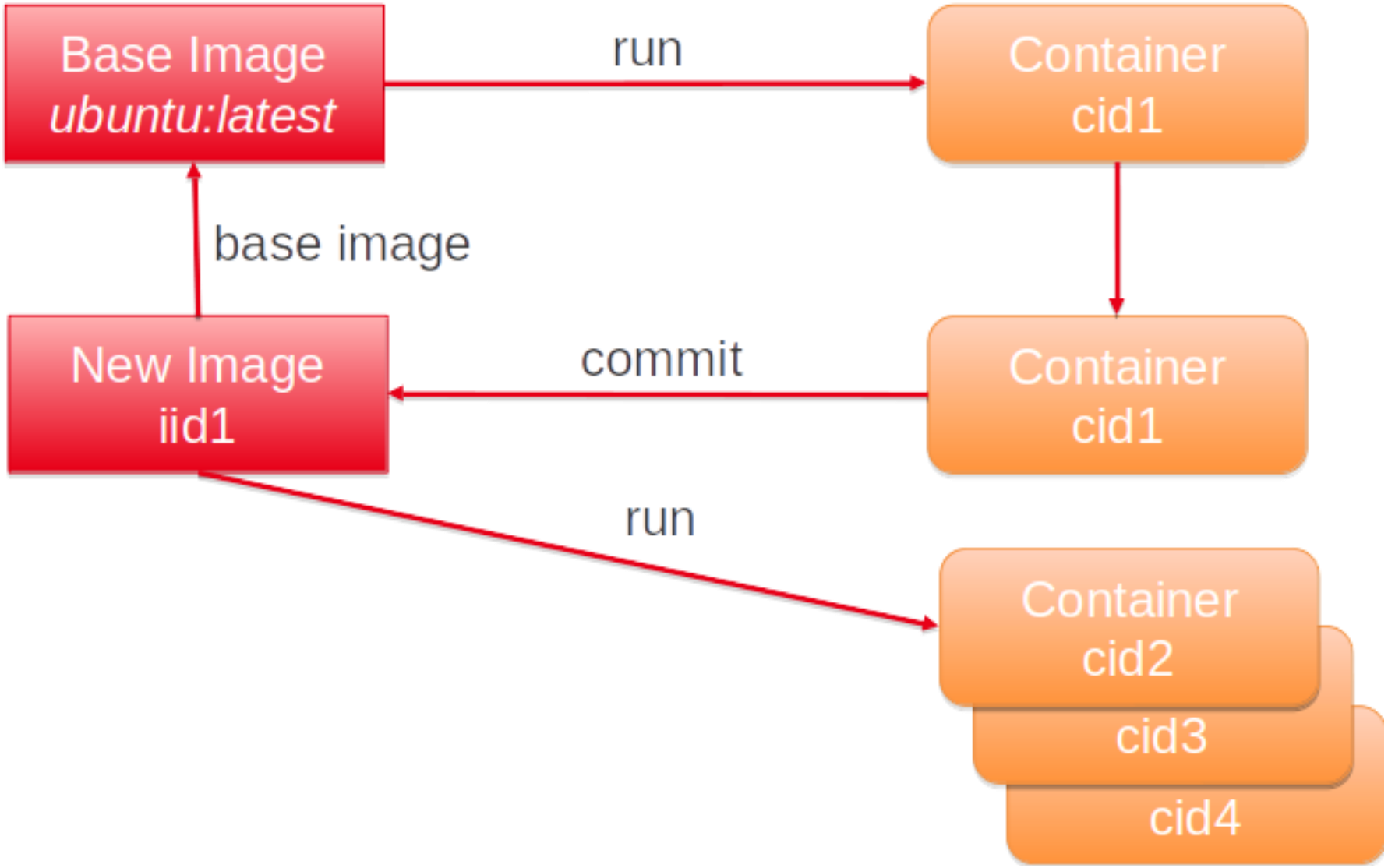
- An image is a collection of files and some meta data.
- Images are comprised of multiple layers, multiple layers referencing/based on another image.
- Each image contains software you want to run.
- Every image contains a base layer.
- Layers are read only.
- An image is an executable package that includes everything needed to run an application--the code, a runtime, libraries, environment variables, and configuration files.



Docker Objects

- **What are CONTAINERS:**
  - A container is a runnable instance of an image. One can create, start, stop, move, or delete a container using the Docker API or CLI.
  - You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.
  - A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.
  - A container is a runtime instance of an image--what the image becomes in memory when executed (that is, an image with state, or a user process). You can see a list of your running containers with the command, *docker ps*.

Image Vs Container



Docker's container ---the concept (and relation to our shipping container)

	Physical Containers	Docker
Content Agnostic	The same container can hold almost any type of cargo	Can encapsulate any payload and its dependencies
Hardware Agnostic	Standard shape and interface allow same container to move from ship to train to semi-truck to warehouse to crane without being modified or opened	Using operating system primitives (e.g. LXC) can run consistently on virtually any hardware—VMs, bare metal, openstack, public IAAS, etc.—without modification
Content Isolation and Interaction	No worry about anvils crushing bananas. Containers can be stacked and shipped together	Resource, network, and content isolation. Avoids dependency hell
Automation	Standard interfaces make it easy to automate loading, unloading, moving, etc.	Standard operations to run, start, stop, commit, search, etc. Perfect for devops: CI, CD, autoscaling, hybrid clouds
Highly efficient	No opening or modification, quick to move between waypoints	Lightweight, virtually no perf or start-up penalty, quick to move and manipulate
Separation of duties	Shipper worries about inside of box, carrier worries about outside of box	Developer worries about code. Ops worries about infrastructure.

Layers

- A Docker container is structured in terms of "layers".
- Process of building image
  - Start with base image.
  - Load software desired.
  - Commit base image + software to form new image.
  - New image can then be base from more software.

- **Exploiting Layers:**

  - When an image is updated, only update new layers.
  - Unchanged layers do not need to be updated.
  - Consequently, less software is transferred, and an update is faster.

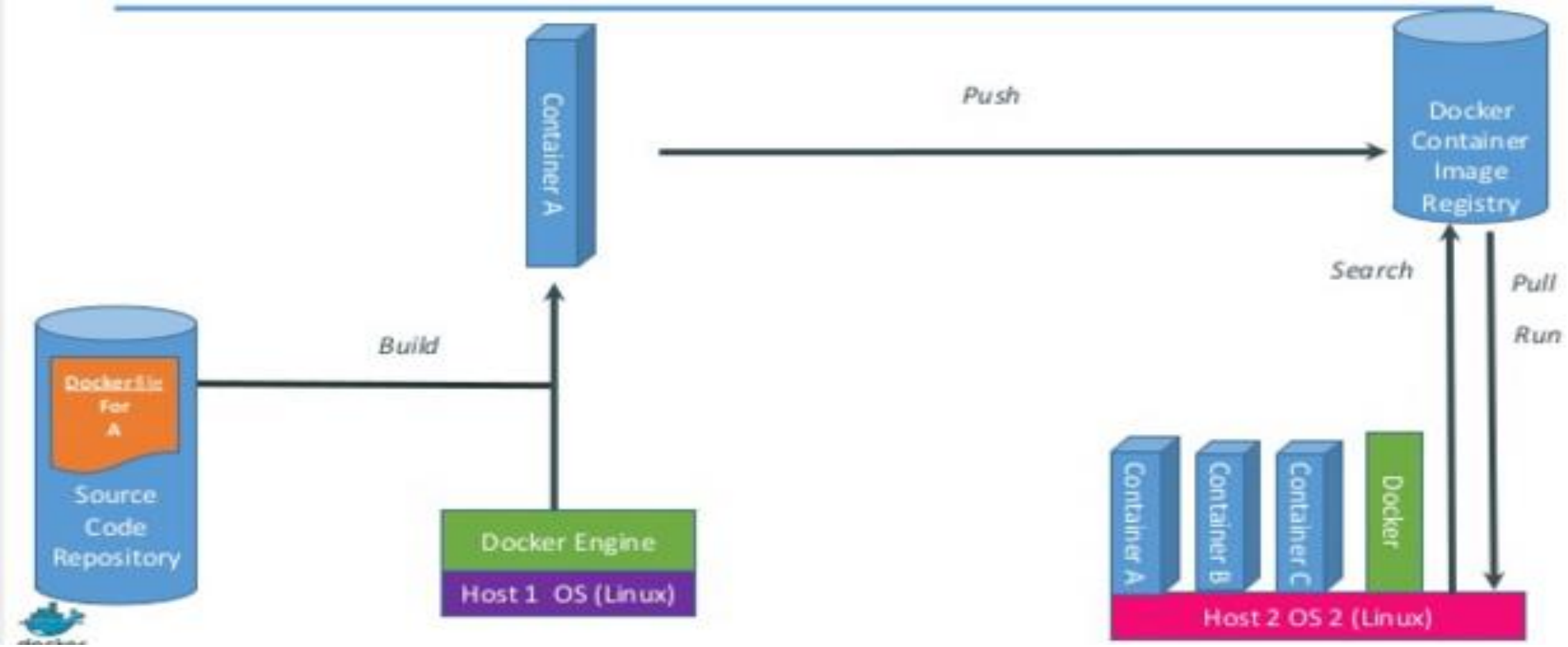
Terminology

- Terminology - Image
  - Persisted snapshot that can be run
    - images :- List all local images.
    - run :- Create a container from an image and execute a command in it.
    - tag :- tag an image
    - pull :- Download image from repo.
    - rmi :- Delete a local image.
      - This will remove intermediate images if no longer used.
- Terminology – Container
  - Runnable instance of an image
    - ps :- List all running containers
    - ps -a :- List all containers (incl. stopped)
    - top :- Display processes of a container
    - start :- Start a stopped container
    - stop :- Stop a running container
    - pause :- Pause all processes within a container
    - rm :- Delete a container
    - commit :- Create an image from a container

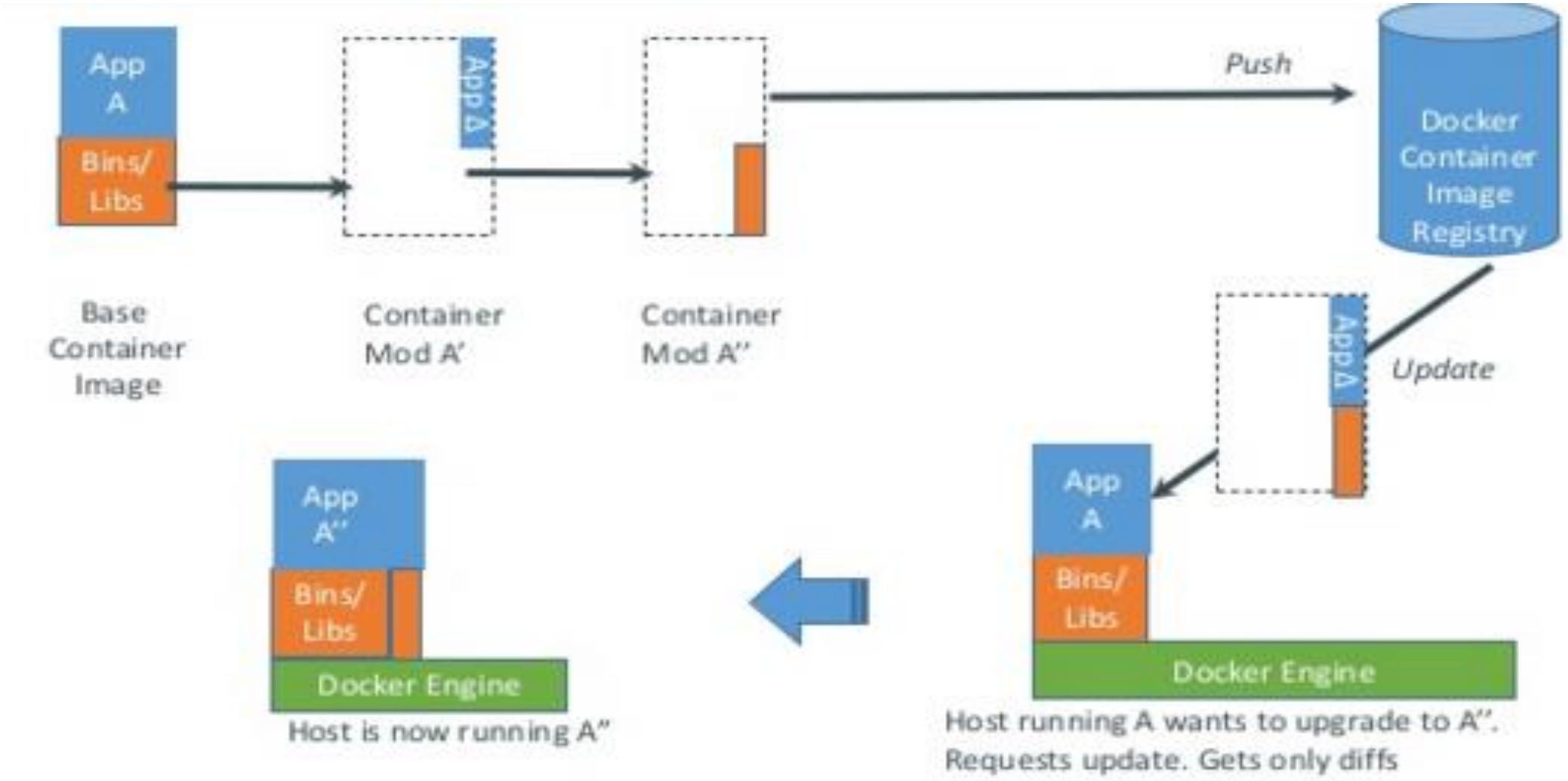
Some Docker Command Examples

- Mount Volume
  - $ docker run –it –v /home/user:/root <docker name or id>
- Publish Port
  - $ docker run –it –p 8080:80 <docker name or id>
- With Root permission
  - $ docker run –it –privileged <docker name or id>
- In background
  - $ docker run –itd <docker name or id>
- Execute Docker Container
  - $ docker exec –it <container id> /bin/bash
- Provide Internet to the docker
  - $ docker run –it –net=host <docker name or id>

How Does Docker Works



Docker Updates/Changes



Docker Hello World Examples

Playing with Busybox

- We are going to run a Busybox container on our system and get a taste of the docker run command.

- Let's get started
  - $ docker pull busybox

- After pulling the image from the **Docker registry** it gets saves in the system. One can use the docker images command to see a list of all images on your system.

  - $ docker images

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
busybox	latest	c51f86c28340	4 weeks ago	1.109 MB

- Run the downloaded image
  - $ docker run busybox

Docker Hello World Examples (Contd..)

•Wait, nothing happened! Is that a bug?

•Well, no. Behind the scenes, a lot of stuff happened. When you call run, the Docker client finds the image, loads up the container and then runs a command in that container. As we didn't provide a command, so the container booted up, ran an empty command and then exited.

•**$ docker run busybox echo "hello from busybox"**

hello from busybox

- If you've noticed, all of that happened quickly. This Proves **containers are fast!**

- Lets see the docker ps command. The docker ps command shows you all containers that are currently running.
  - $ docker ps
  - **CONTAINER ID IMAGE COMMAND CREATED STATUS**

- To run more than one command with docker, create a script and pass it with docker run command
  - $ docker run -it busybox sh

    / # ls

    bin dev etc home proc root sys tmp usr var

    / # uptime

    05:45:21 up 5:58, 0 users, load average: 0.00, 0.01, 0.04

Docker Hello World Examples (Contd..)

- Remove the existing Docker
  - **docker rm 305297d7a235 ff0a5c3750b9**
    305297d7a235
    Ff0a5c3750b9

  - Or **$ docker rm $(docker ps -a -q -f status=exited)**

  - Or **$ docker container prune**

    WARNING! This will remove all stopped containers.
    Are you sure you want to continue? [y/N] y
    Deleted Containers:
    4a7f7eebae0f63178aff7eb0aa39f0627a203ab2df258c1a00b456cf20063
    f98f9c2aa1eaf727e4ec9c0283bcaa4762fbdba7f26191f26c97f64090360
    Total reclaimed space: 212 B

Use Of Docker

- Open platform for developers and system administrators to build and test applications
- Development Environment
- Environments for Integration Tests
- Quick evaluation of software
- Microservices
- Multi-Tenancy
- Unified execution environment (dev test prod (local, VM, cloud, ...)

References

- For Docker Architecture:
  - https://docs.docker.com/engine/docker-overview/
- For Docker:
  - https://opensource.com/resources/what-docker
- For Docker Objects:
  - https://docs.docker.com/engine/docker-overview/
- For Docker Example:
  - https://docker-curriculum.com/#hello-world

SECURE CONNECTIONS
FOR A SMARTER WORLD

www.nxp.com