22AIE305: CLOUD COMPUTING

Cargo Transport Pre-1960

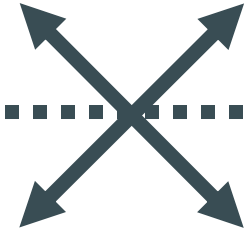
Multiplicity of Goods





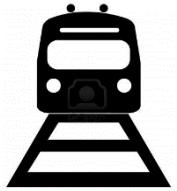


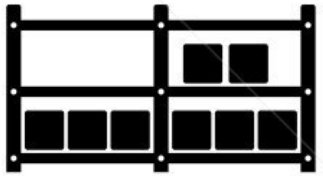
Do I worry about how goods interact (e.g. coffee beans next to spices)



Multipilicity of methods for transporting/storing






Can I transport quickly and smoothly (e.g. from boat to train to truck)

Solution: Intermodal Shipping Container

Multiplicity of Goods





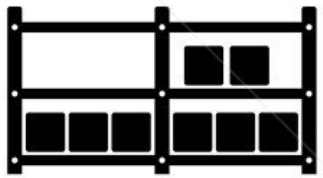


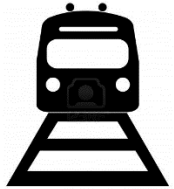
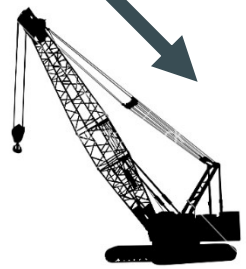
Do I worry about how goods interact (e.g. coffee beans next to spices)

A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.

…in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

Multiplicity of methods for transporting/storing

Can I transport quickly and smoothly (e.g. from boat to train to truck)

Docker is a shipping container system for code

Multiplicity of Stacks

Static website

User DB

Web frontend

Queue

Analytics DB

Do services and apps interact appropriately?

An engine that enables any payload to be encapsulated as a lightweight, portable, self-sufficient container…

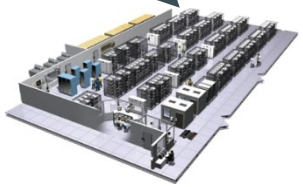
…that can be manipulated using standard operations and run consistently on virtually any hardware platform

Multiplicity of hardware environments

Development VM

QA server

Customer Data Center



Public Cloud

Production Cluster

Contributor's laptop

Can I migrate smoothly and quickly

- A container is a runtime instance of a container image (which is a lightweight, executable package that includes everything needed to run your code). It is the execution environment that runs the application or service defined by the container image.
- When a container is started, it becomes an isolated process on the host machine with its own filesystem, network interfaces, and other resources. Containers share the host operating system's kernel, making them more efficient and faster to start than virtual machines.
- A virtual machine (VM) is an **emulation of a physical computer**. Each VM runs a full operating system and has virtualized hardware, which makes them more resource-intensive and slower to start compared to containers.

CONTAINER REGISTRY

Docker Container is a lightweight software package that includes all the dependencies (frameworks, libraries, etc.) required to execute an application

The Container registry stores container images and allows you to associate an image with a repository. It is similar in function to the Windows Registry

You can choose whether to inherit permissions from a repository, or set granular permissions independently of a repository. You can also access public container images anonymously.

Namespaces

Namespaces are used for isolation of:

- filesystem - like `chroot` but more secure
- UTS (host and domain names)
- IPC (interprocess communication resources)
- PIDs (process ID number space)
- network stack (devices, addresses, routing, ports, etc.)
- users (user and group IDs)

Kernel Control Groups (cgroups)

cgroups partition sets of tasks into hierarchical groups

Allows control over system resources:

- resource limits (CPU, memory)
- bandwidth limits (block I/O)
- prioritization
- access control (devices)

Provides accounting/metrics

Allows management of tasks:

- suspend/resume

Docker

- **Docker is a platform as a service(*PaaS*) that allows developers to use OS-level virtualization to automate the deployment of applications with all its library dependencies inside lightweight and portable packages called containers.**
- **Terminologies**
- **Container: A standalone executable package of software that includes everything to run it.**
- **Docker Image: A read-only template that contains instructions for creating a container.**
- **Dockerfile: A file that contains a series of instructions on how to build a docker image.**

Docker run

- The docker run command creates and starts a new container, also pulling the image if it is needed to start the container.
- It is used when we want to create a container for the first time. Example of a docker run command is
- docker run -d --name <container-name>
- -d: run the container in the background and print the new container ID
- --name: assign the name container-name to the container

Step-by-step procedure for Docker run

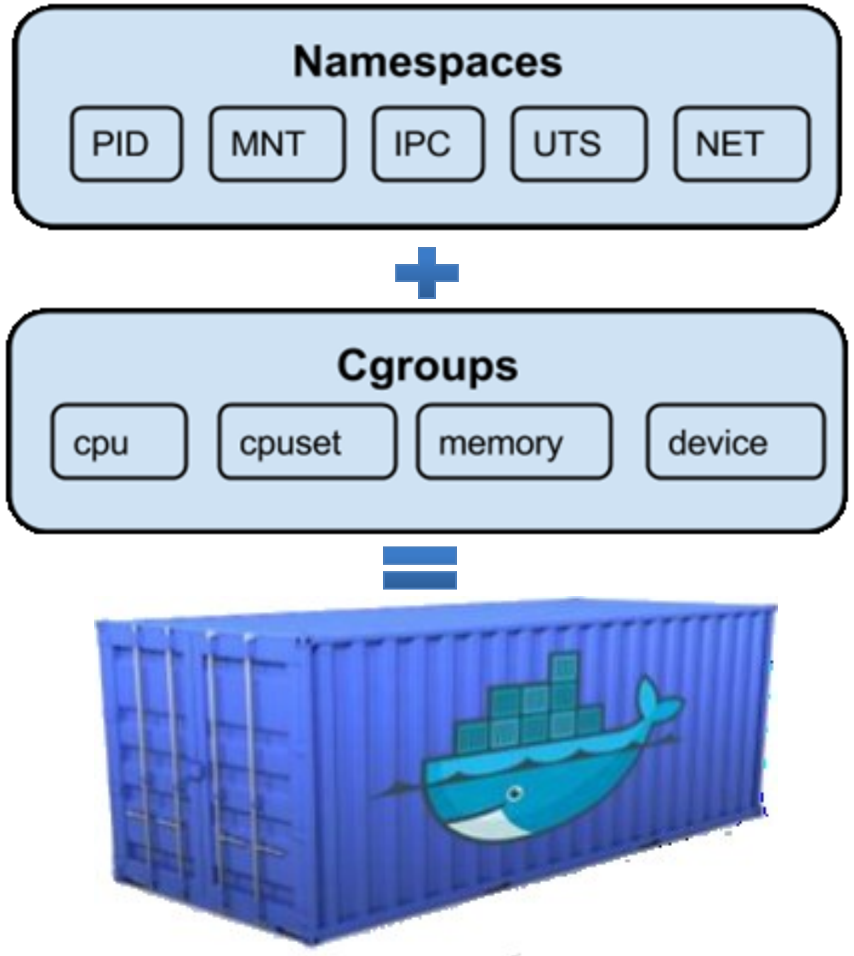
- Pull the image: If the specified image is not available, docker pulls it from the registry.
- Create container: a new container is created from the specified image.
- Assign resources: Compute resources like CPU, memory, volumes, and ports are assigned to the container.
- Start the container: The container is started, and the given command is executed.

Dockerfile

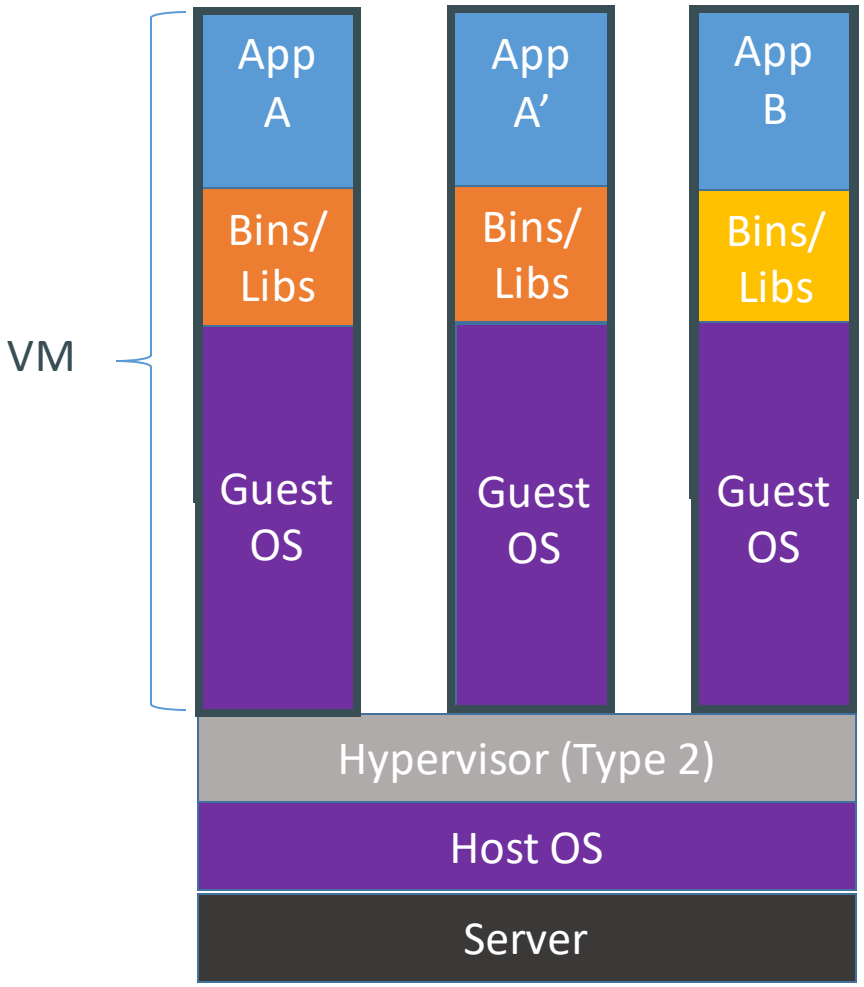
- Dockerfiles are configuration files that define how to build containers from images.
- It uses configuration tools, build tools, packages, etc.
- Example Dockerfile:

```
FROM ubuntu
RUN apt-get update
RUN apt-get upgrade -y
RUN apt-get install -y build-essential
```

Docker using Cgroups

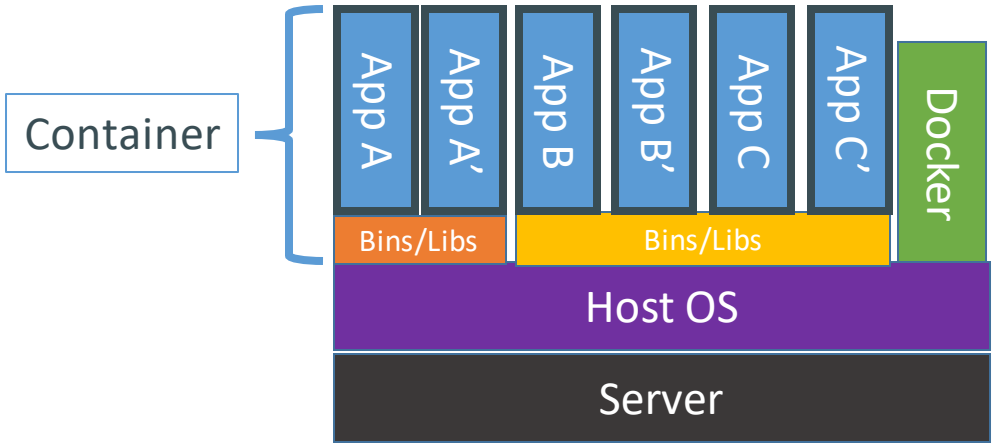


Containers vs. VMs



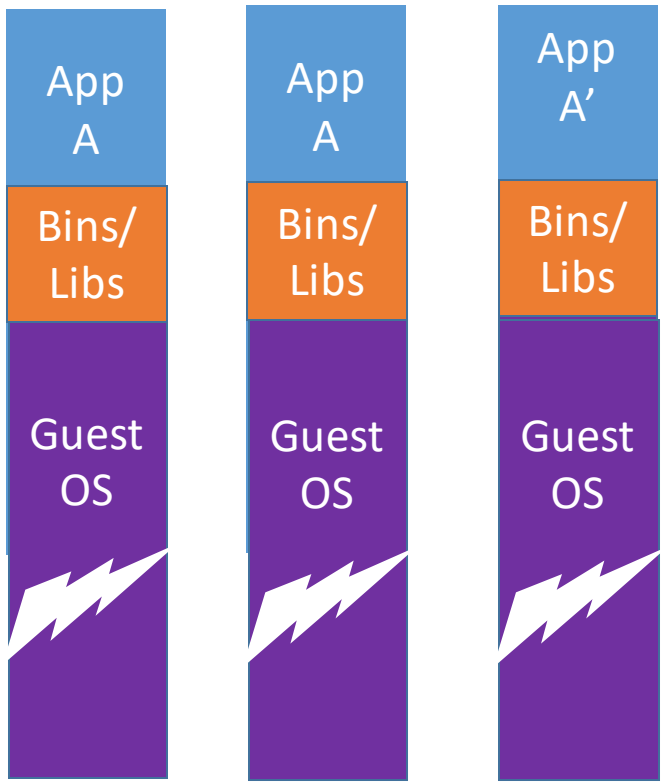
Containers are isolated, but share OS and, where appropriate, bins/libraries

…result is significantly faster deployment, much less overhead, easier migration, faster restart



Why are Docker containers lightweight?

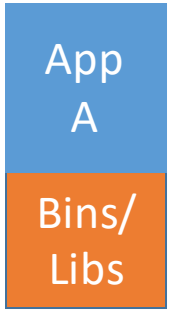
VMs



VMs

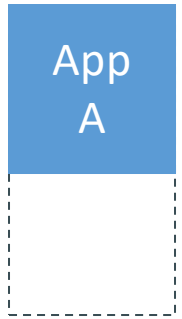
Every app, every copy of an app, and every slight modification of the app requires a new virtual server

Containers



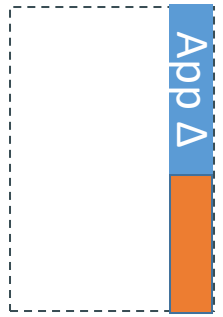
Original App

(No OS to take up space, resources, or require restart)



Copy of App

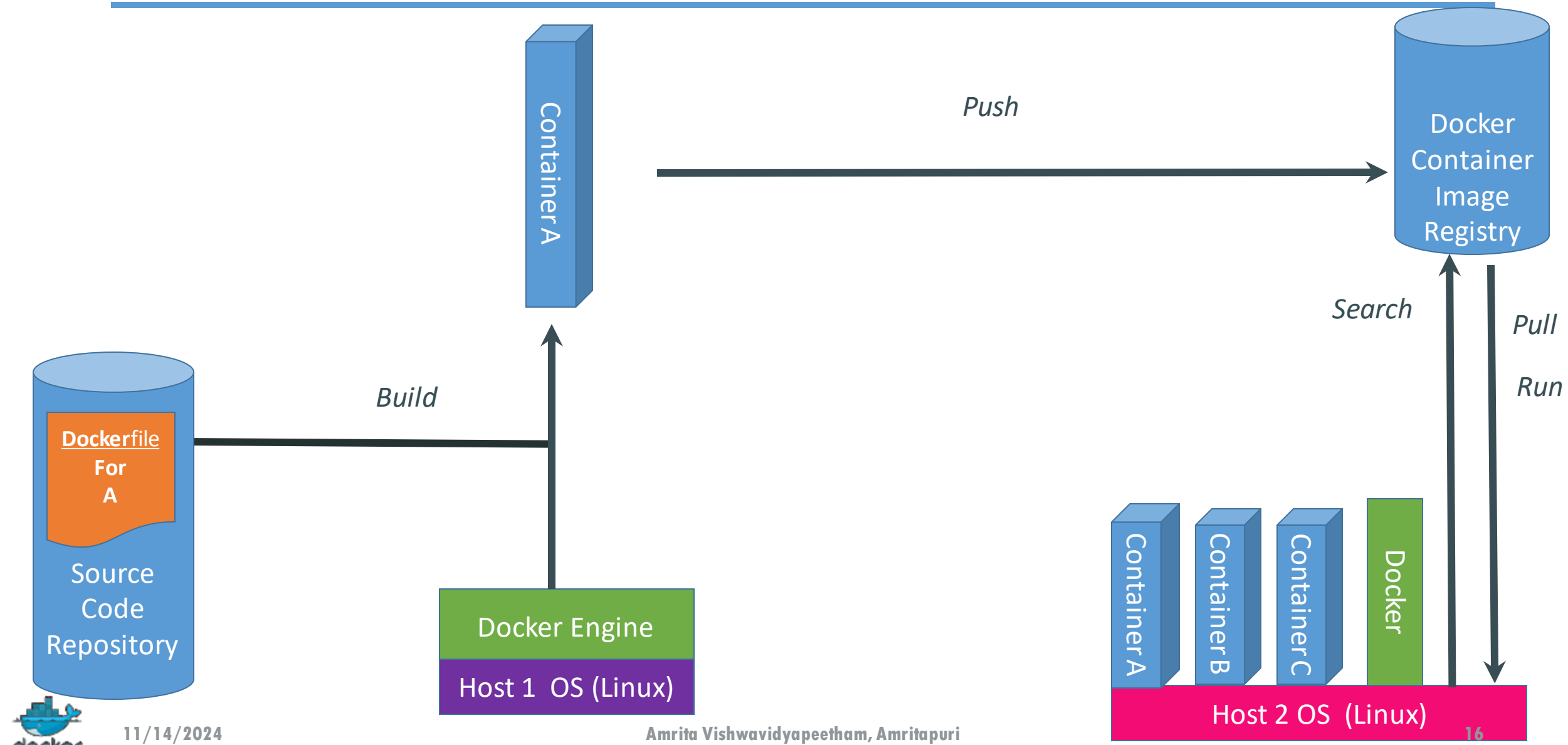
No OS. Can Share bins/libs



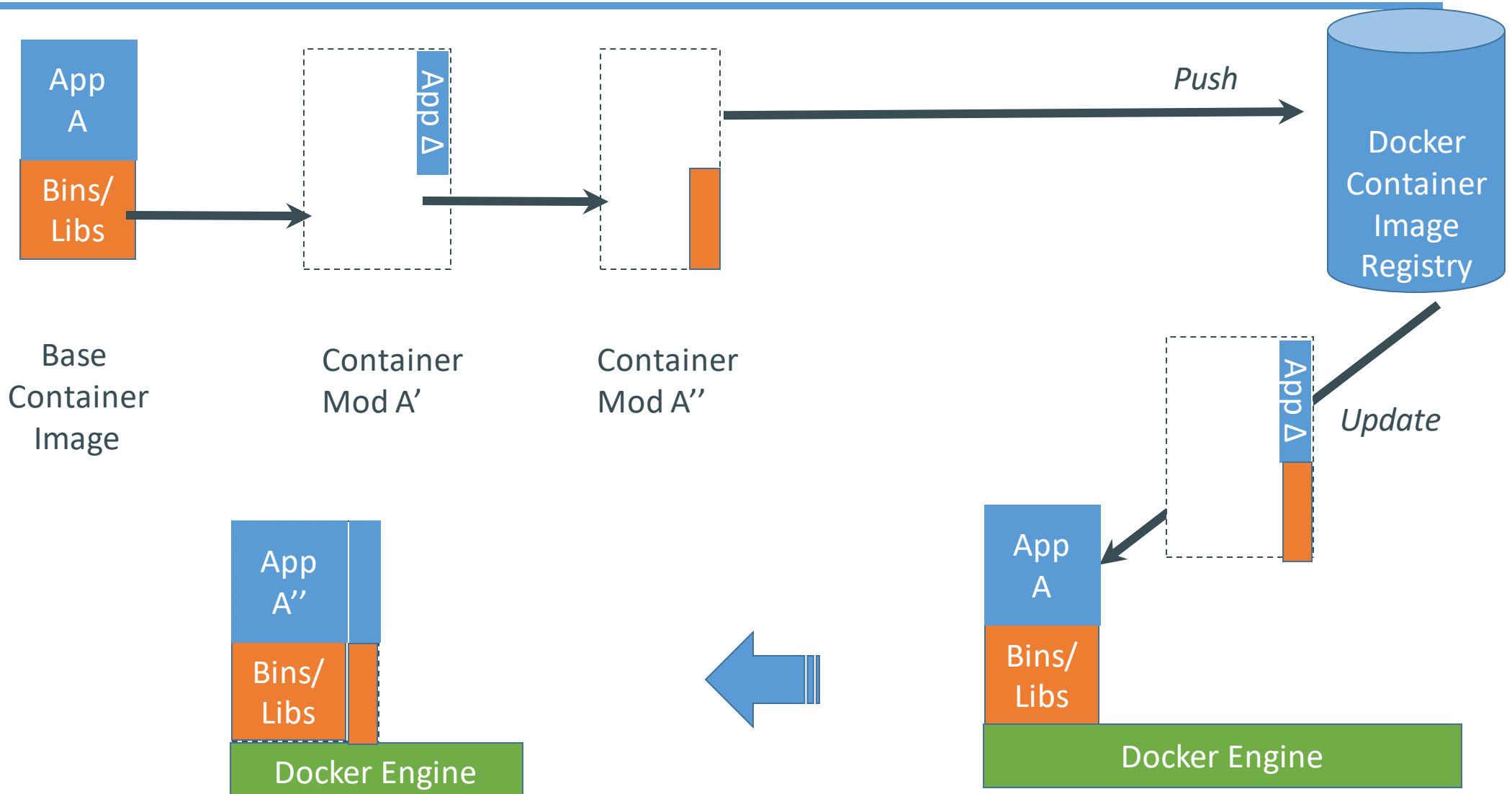
Modified App

Copy on write capabilities allow us to only save the diffs Between container A and container A'

What are the basics of the Docker system?



Changes and Updates



Ecosystem Support

- Operating systems
  - Virtually any distribution with a 2.6.32+ kernel
  - Red Hat/Docker collaboration to make work across RHEL 6.4+, Fedora, and other members of the family (2.6.32 +)
  - CoreOS—Small core OS purpose built with Docker
- OpenStack
  - Docker integration into NOVA (& compatibility with Glance, Horizon, etc.) accepted for Havana release
- Private PaaS
  - OpenShift
  - Solum (Rackspace, OpenStack)
  - Other TBA
- Public PaaS
  - Deis, Voxoz, Cocaine (Yandex), Baidu PaaS
- Public IaaS
  - Native support in Rackspace, Digital Ocean,+++
  - AMI (or equivalent) available for AWS & other
- DevOps Tools
  - Integrations with Chef, Puppet, Jenkins, Travis, Salt, Ansible +++
- Orchestration tools
  - Mesos, Heat, ++
  - Shipyard & others purpose built for Docker
- Applications
  - 1000's of Dockerized applications available at index.docker.io

Use Cases

- Ted Dziuba on the Use of Docker for Continuous Integration at Ebay Now
  - https://speakerdeck.com/teddziuba/docker-at-ebay
  - http://www.youtube.com/watch?feature=player_embedded&v=0Hi0W4gX--4

- Sasha Klizhentas on use of Docker at Mailgun/Rackspace
  - http://www.youtube.com/watch?feature=player_embedded&v=CMC3xdAo9RI

- Sebastien Pahl on use of Docker at CloudFlare
  - http://www.youtube.com/watch?feature=player_embedded&v=-Lj3jt_-3r0

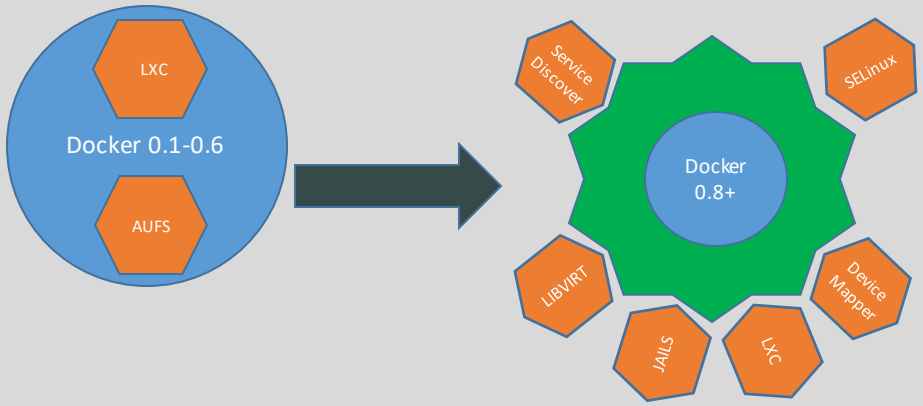
- Cambridge HealthCare
  - http://blog.howareyou.com/post/62157486858/continuous-delivery-with-docker-and-jenkins-part-i

- Red Hat Openshift and Docker
  - https://www.openshift.com/blogs/technical-thoughts-on-openshift-and-docker

Use Cases—From Our Community

Use Case	Examples	Link
Clusters	Building a MongoDB cluster using docker	http://bit.ly/1acbjZf
	Production Quality MongoDB Setup with Docker	http://bit.ly/15CaiHb
	Wildfly cluster using Docker on Fedora	http://bit.ly/1bClX0O
Build your own PaaS	OpenSource PaaS built on Docker, Chef, and Heroku Buildpacks	http://deis.io
Web Based Environment for Instruction	JiffyLab – web based environment for the instruction, or lightweight use of, Python and UNIX shell	http://bit.ly/12oaj2K
Easy Application Deployment	Deploy Java Apps With Docker = Awesome	http://bit.ly/11BCvvu
	How to put your development environment on docker	http://bit.ly/1b4XtJ3
	Running Drupal on Docker	http://bit.ly/15MJS6B
	Installing Redis on Docker	http://bit.ly/16EWOKh
Create Secure Sandboxes	Docker makes creating secure sandboxes easier than ever	http://bit.ly/13mZGJH
Create your own SaaS	Memcached as a Service	http://bit.ly/11nL8vh
Automated Application Deployment	Multi-cloud Deployment with Docker	http://bit.ly/1bF3CN6
Continuous Integration and Deployment	Next Generation Continuous Integration & Deployment with dotCloud's Docker and Strider	http://bit.ly/ZwTfoy
	Testing Salt States Rapidly With Docker	http://bit.ly/1eFBtcm
Lightweight Desktop Virtualization	Docker Desktop: Your Desktop Over SSH Running Inside Of A Docker Container	http://bit.ly/14RYL6x

Docker Futures*



- Docker 0.7
  - Fedora compatibility
  - Reduce kernel dependencies
  - Device mapper
  - Container linking
- Docker 0.8
  - Shrink and stabilize Core
  - Provide stable, pluggable API
  - RHEL compatibility
  - Nested containers
  - Beam: Introspection API based on Redis
  - expand snapshot management features for data volumes
  - We will consider this “production ready”
- **Docker 27.2 is latest stable version**

Advanced topics

- Data
  - Today: Externally mounted volumes
    - Share volumes between containers
    - Share volume between a containers and underlying hosts
      - high-performance storage backend for your production database
      - making live development changes available to a container, etc.
    - Optional: specify memory limit for containers, CPU priority
    - Device mapper/ LVM snapshots in 0.7
  - Futures:
    - I/O limits
    - Container resource monitoring (CPU & memory usage)
    - Orchestration (linking & synchronization between containers)
    - Cluster orchestration (multi-host environment)
- Networking
  - Supported today:
    - UDP/TCP port allocation to containers
      - specify *which* public port to redirect. If you don't specify a public port, Docker will revert to allocating a random public port.
      - Docker uses IPtables/netfilter
    - IP allocation to containers
      - Docker uses virtual interfaces, network bridge,
  - Futures:
    - See Pipework (Upstream) : **Software-Defined Networking for Linux Containers** (https://github.com/jpetazzo/pipework)
    - Certain pipework concepts will move from upstream to part of core Docker
    - Additional capabilities come with libvirt support in 0.8-0.9 timeframe

C/S ARCHITECTURE

Docker uses client-server architecture.

A Docker client talks with the docker daemon which helps to build, run, and distribute the docker containers.

The Docker client runs with the daemon on the same system or we can connect the Docker client with the Docker daemon remotely.

With the help of REST API over a socket or a network, the docker client and daemon interact with each other.

DOCKER NETWORKING

Docker networking provides complete isolation for docker containers.

A user can link a docker container to many networks.

It requires very less OS instances to run the workload.

data can be stored within the writable layer of the container but it requires a storage driver.

Storage driver controls and manages the images and containers on our docker host.

TYPES OF DOCKER STORAGE

Data Volumes: Data Volumes can be mounted directly into the filesystem of the container (essentially directories or files on the Docker Host file system)

Volume Container: In order to maintain the state of the containers (data) produced by the running container, Docker volumes file systems are mounted on Docker containers. independent container life cycle, the volumes are stored on the host. This makes it simple for users to exchange file systems among containers and backup data.

Directory Mounts: A host directory that is mounted as a volume in your container might be specified.

Storage Plugins: Docker volume plugins enable us to integrate the Docker containers with external volumes like Amazon EBS so that we can maintain the state of the container.

- Docker's architecture allows for efficient container management and deployment.
- The docker command uses the Docker API.
- A Docker client can communicate with multiple daemons.
- When a docker client runs any docker command on the docker terminal then the terminal sends instructions to the daemon.
- The main objective of the docker client is to provide a way to direct the pull of images from the docker registry and run them on the docker host. The common commands which are used by clients are **docker build, docker pull,** and **docker run.**

TYPES OF DOCKER NETWORKS

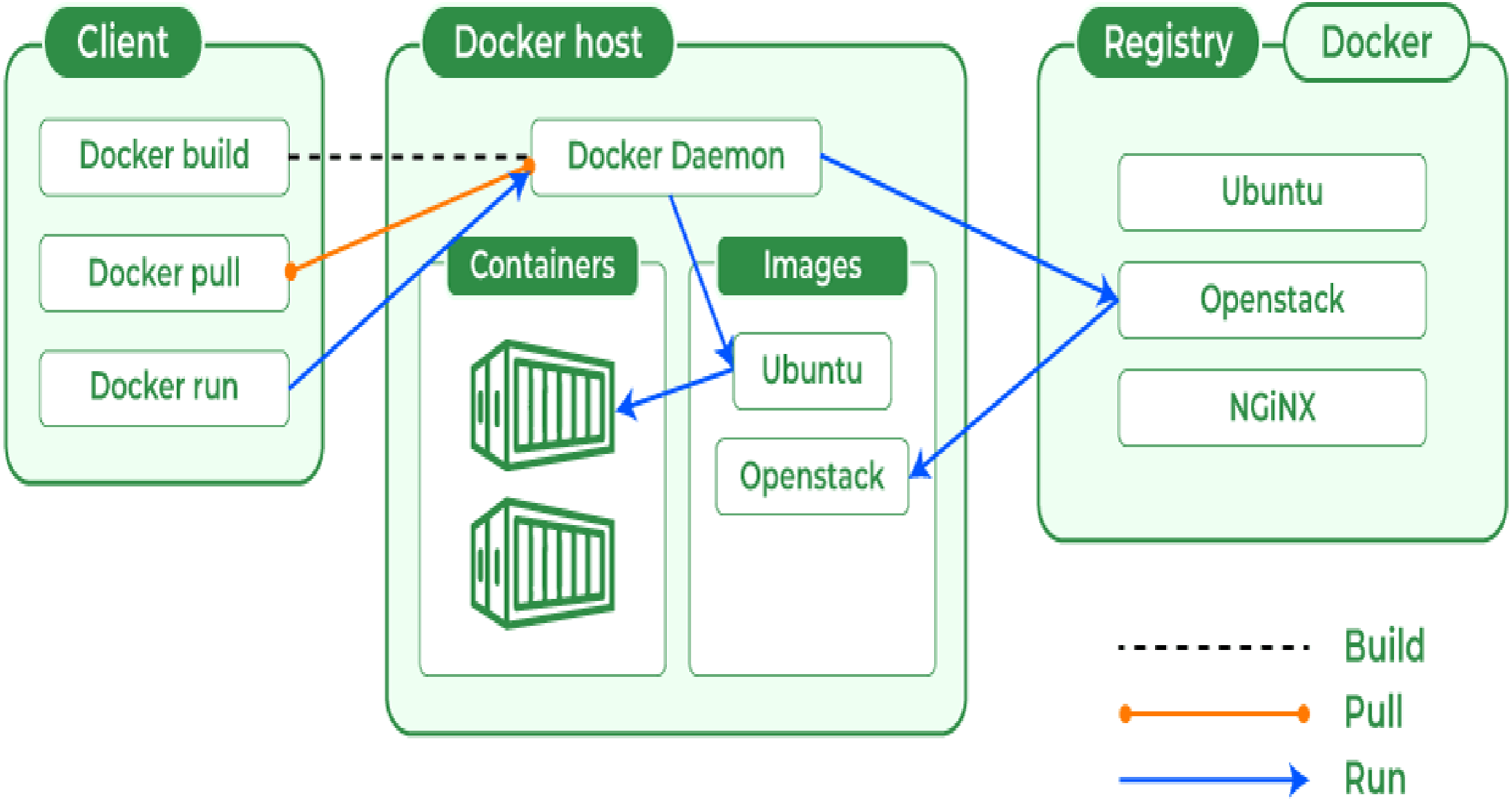
Bridge: It is the default network driver. Use this when different containers communicate with the same docker host.

Host: When you don't need any isolation between the container and host then it is used.

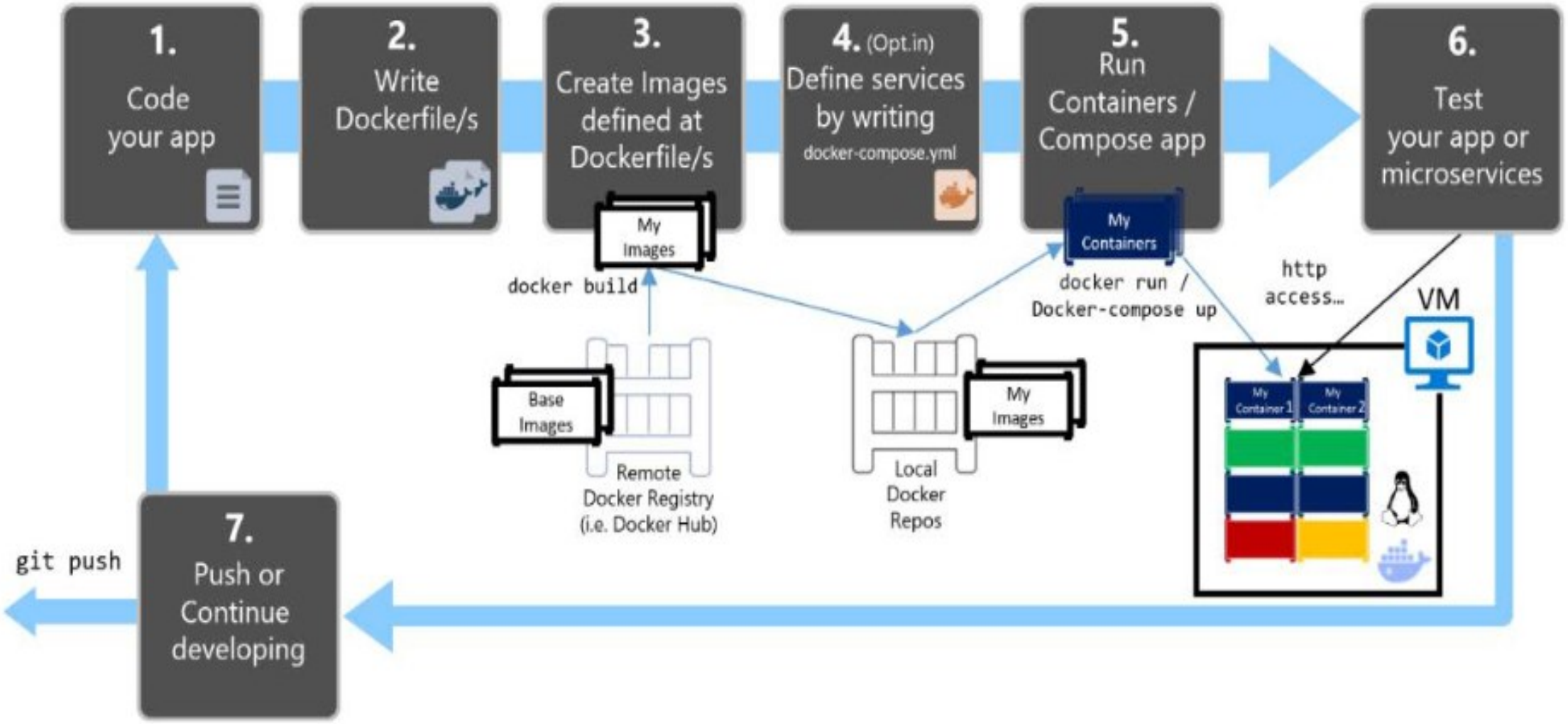
Overlay: For communication with each other, it will enable the swarm services.

None: It disables all networking.

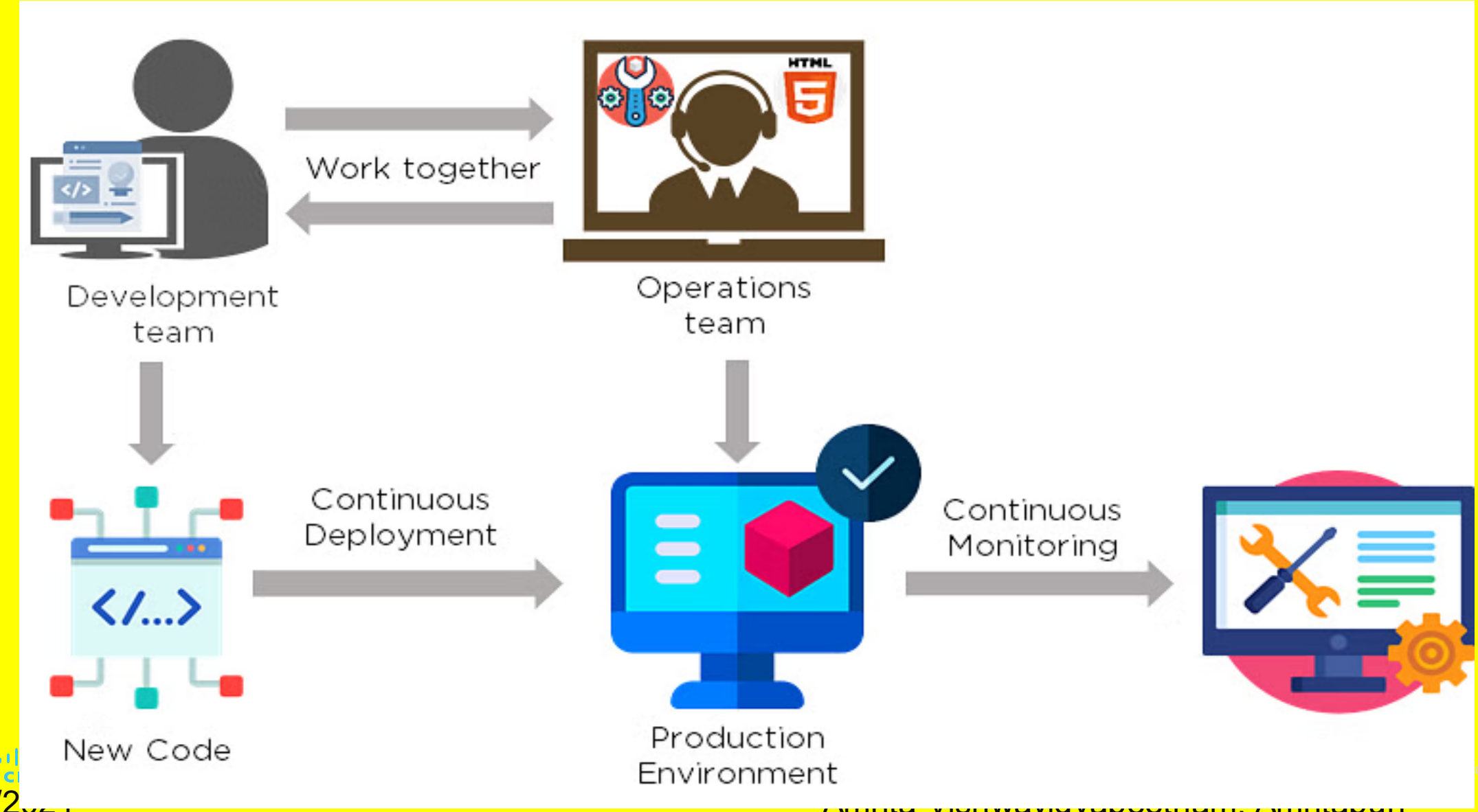
macvlan: This network assigns MAC(Media Access control) address to containers which look like a physical address.


Client
Docker build
Docker pull
Docker run
Docker host
Docker Daemon
Containers
Images
Ubuntu
Openstack
Registry
Docker
Ubuntu
Openstack
NGiNX
Build
Pull
Run

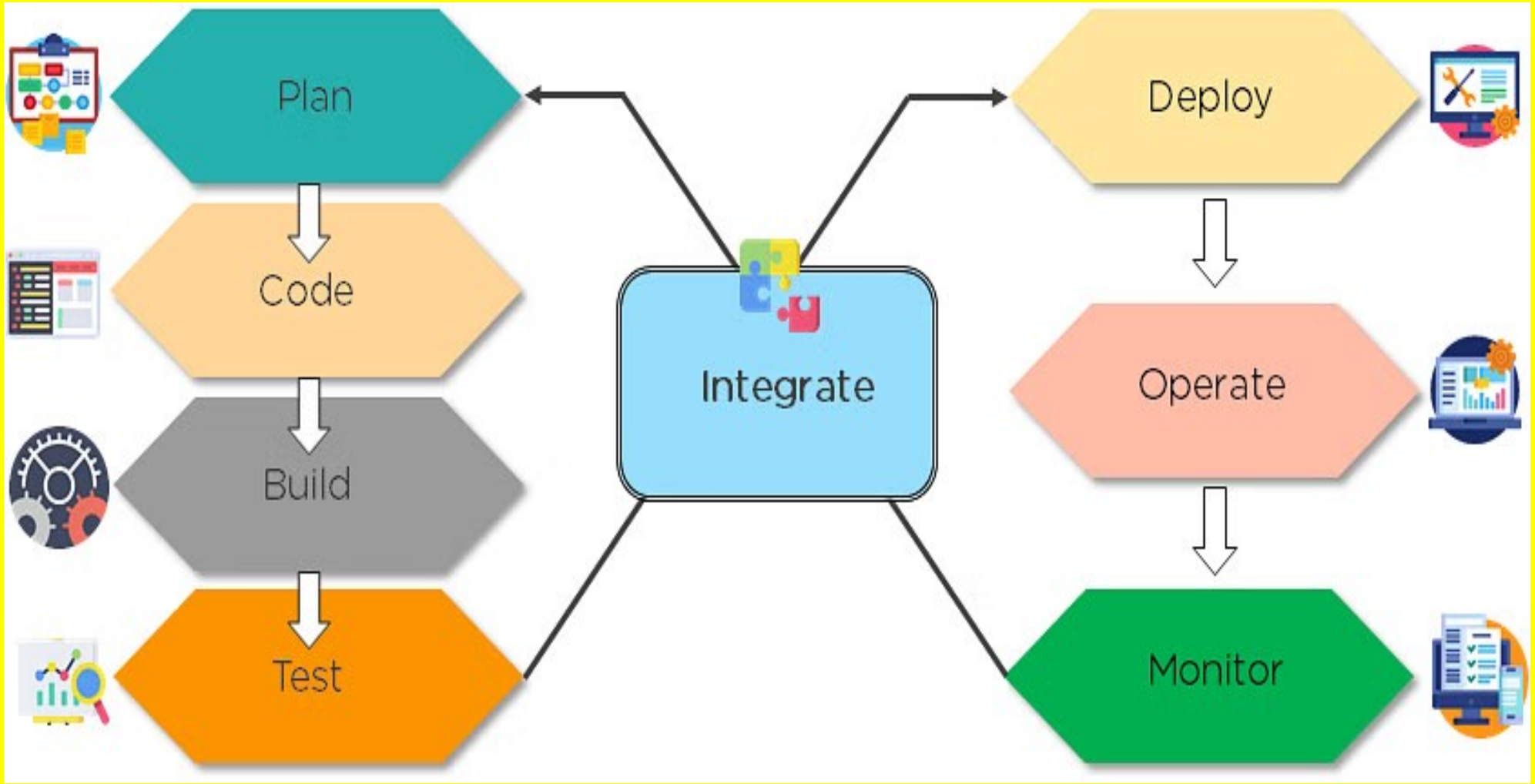
Inner-Loop development workflow for Docker apps

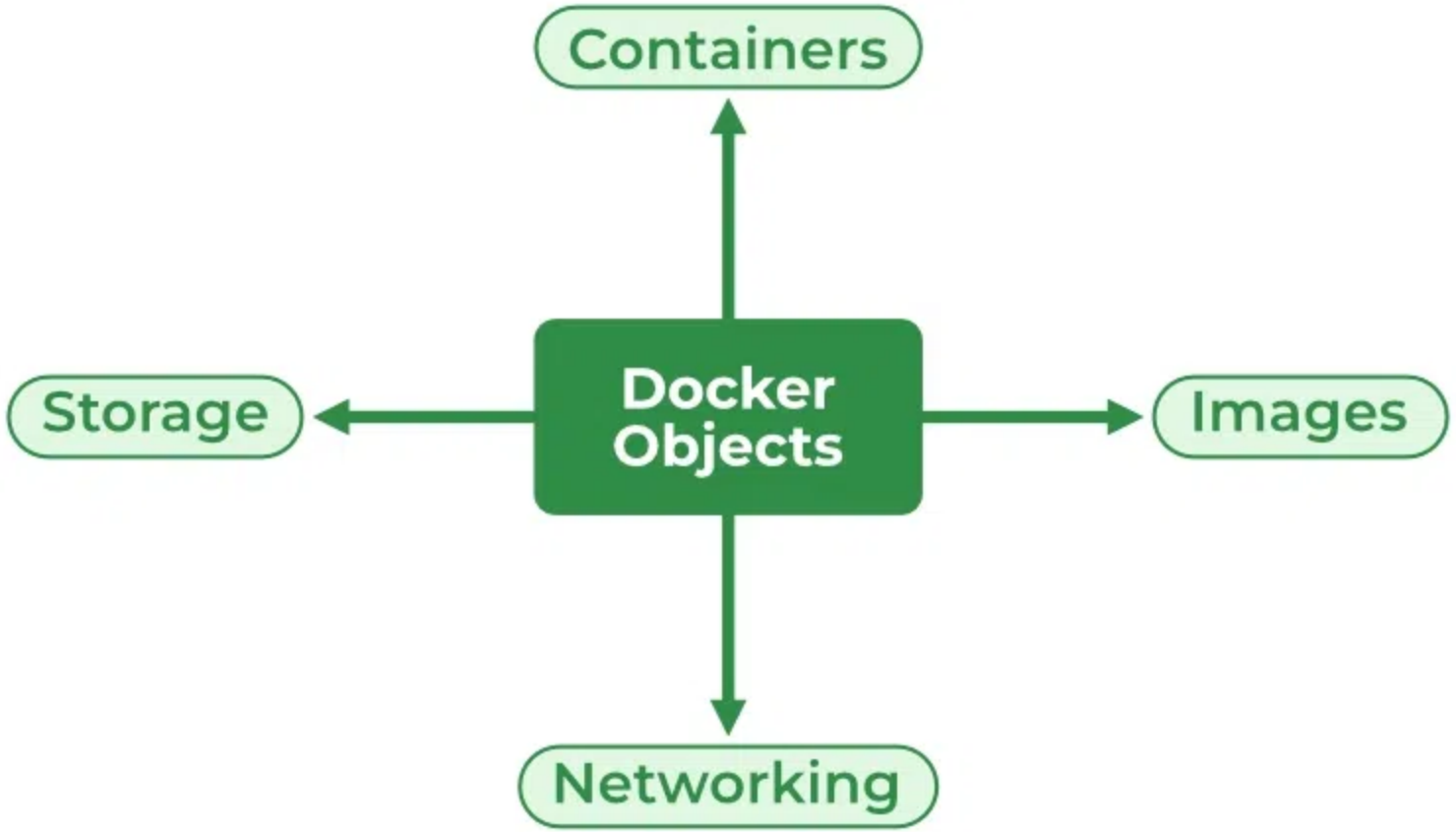


DevOps: development and operations teams work together



Phases in DevOps




Containers
Storage
Docker Objects
Images
Networking

Images

- The collection of binaries, libraries, and other files needed to run an application

Container

- A running application with its associated resources (i.e. the image)
- Short lived and usually stateless