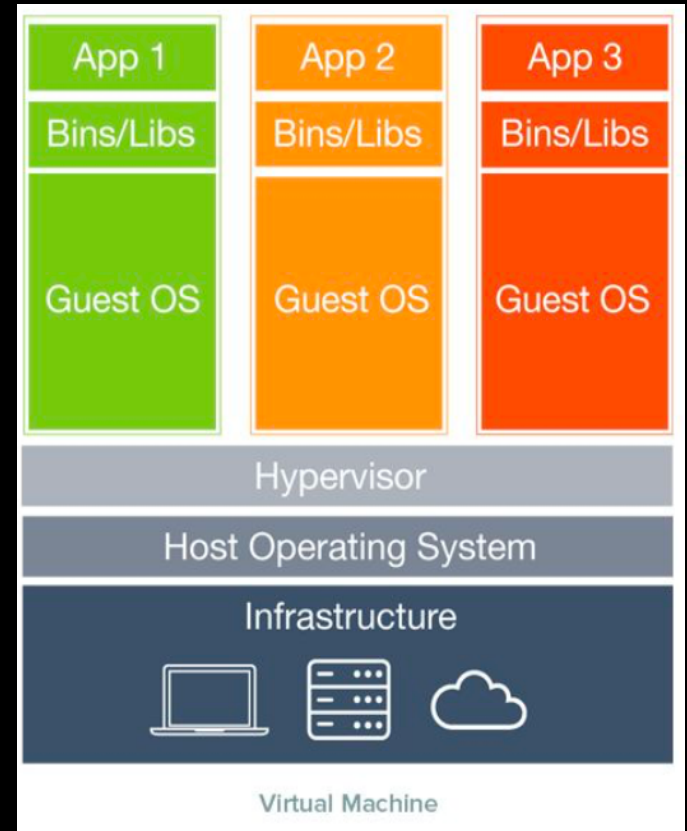


Containerization

Simplifying *Application Deployment* with Containers

Background - Virtualization.

- Virtualization is a process of creating an abstraction layer over hardware, allowing a single computer to be divided into multiple virtual computers. Each of those virtual computers (“guests”) uses part of the hardware resources of the main computer (a host)
- The software used to achieve this is a hypervisor. It runs on the host OS and enable multiple guest OSs to run on top of it, sharing the same physical resources managed by the host OS.



Advantages of Virtualization

- Minimize hardware costs (Capital Expenditure)
 - Multiple virtual servers on one hardware.
- Easily move VMs to other data centres.
 - Provide disaster recovery. Facilitate Hardware maintenance.
 - Follow the sun (active users) or follow the moon (cheap power).



Advantages of Virtualization

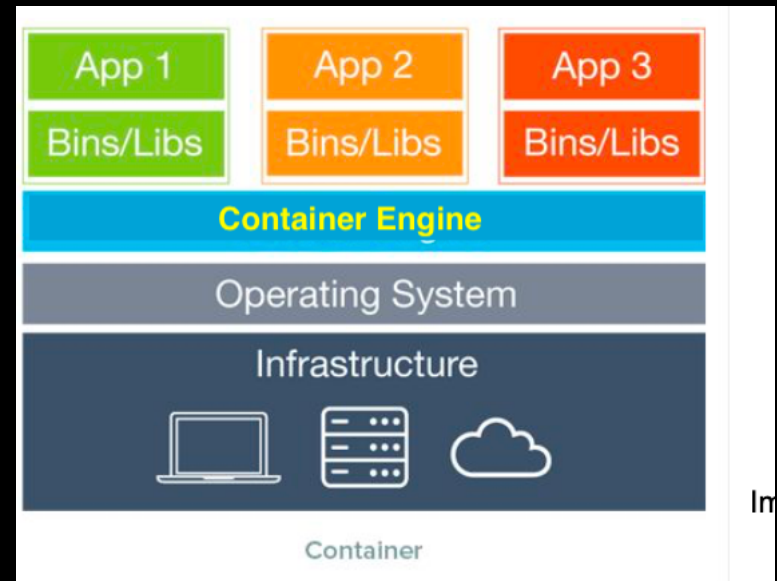
- Easier automation (Lower Operating Expenditure)
 - Simplified provisioning/administration of hardware and software.
- Scalability and Flexibility: Multiple operating systems.
- Consolidate idle workloads. Usage is bursty and asynchronous. Increase device utilization
- Conserve power
 - Free up unused physical resources

Problems of Virtualization

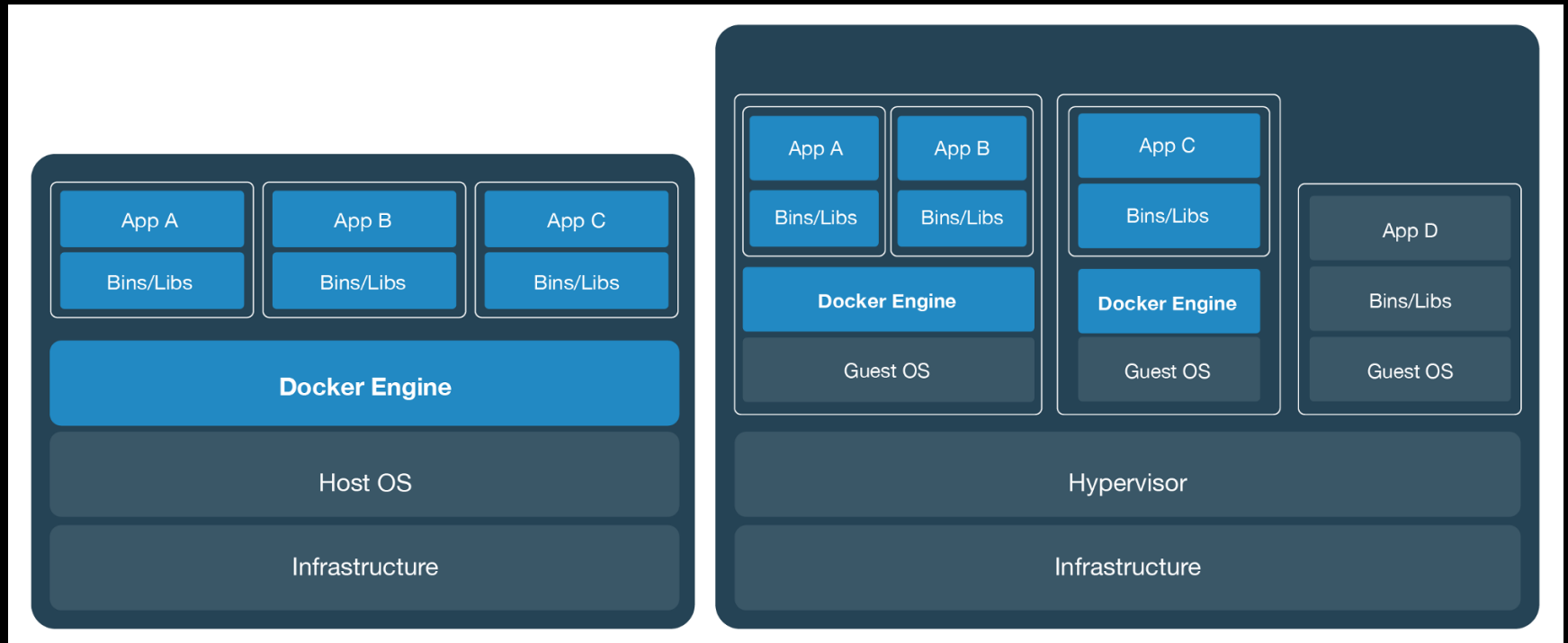
- Each VM requires a full operating system (OS).
Each OS:
 - Requires a license \Rightarrow Capital Expenditure.
 - Has its own compute and storage overhead.
 - Needs maintenance, updates \Rightarrow Operating Expenditure.
- VM Tax = added Capital Expenditure + Operating Expenditure.

Solution : Containers

- Containers:
 - Isolated environments to run apps/services (OS process isolation).
 - Portable, easy to build, quick to deploy.
 - Share the resources of an OS kernel.
 - Disposable / Ephemeral.



Containers Versus VMs - They're different, not mutually exclusive.



The Problem Containers Solve.

- Developers often face issues when working in different environments with varying configurations.
 - Development, test, production.
 - “But it works on my computer!”
- Containers provide a consistent and isolated environment, ensuring that your application runs the same way across different machines.

Benefits of Containerization

- Containerization offers several benefits for application development and deployment:
 - Portability: Application and its environments packaged in containers can run on any machine or cloud platform that supports containerization.
 - Scalability: you to scale your applications easily by running multiple containers across multiple machines.
 - Cost-effective – Containers are lightweight and use far less system resources.
 - Reproducibility: you can replicate and share your application's environment to ensure consistent results.

Containers.

- Containers have all the good properties of VMs:
 - Self-contained – Come complete with all files and data an app needs to run.
 - Scalable – Multiple copies can be run on the same machine or different machine.
 - Flexible, Portable – Same image can run on a PC, in a data center or in the cloud.
 - Isolation – For example, “Show Process” (ps on Linux) command in a container will show only the processes in the container.

VMs versus Containers.

- Criteria: Performance, Scalability, Security ...

Criteria	VM	Containers
Image Size	3X	X
Boot Time	>10s	~1s
Computer Overhead	>10%	<5%
Disk I/O Overhead	>50%	Negligible
Isolation	Good	Good
Security	Low-Medium	Medium-High

Containerization – Key terms.

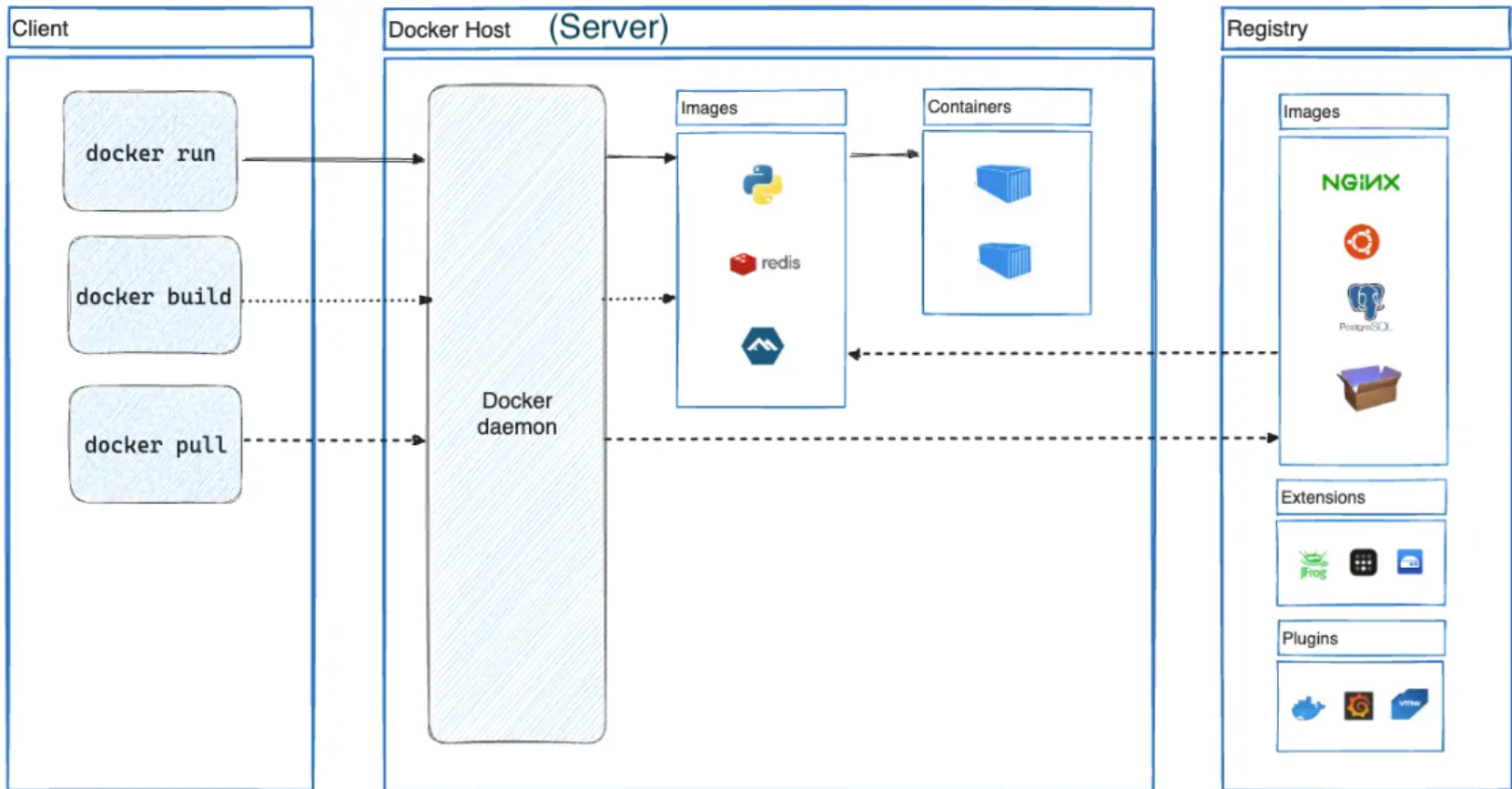
- Image: a standalone, executable package that includes everything needed to run a piece of software (application code, dependencies, system tools and libraries, configuration files).
 - The entire filesystem and metadata (e.g. environment variables) for an app at runtime.
 - An immutable template for a containers
- Container: The runtime instance of an image.
 - What the image becomes in memory when it's executed.
 - An OS process isolated from the rest of the system through abstractions created by the OS.
 - A container's filesystem comes from an image.

Introduction to Docker

Docker.

- Docker is an open-source platform for creating and managing container-based environments.
- Docker simplifies the development and deployment process by providing a consistent environment across different machines.
- Facilitates sharing and distributing your applications with others.
- Written in Go language.
- Released in 2013.
- Linux-based, but solutions that work on Windows and Mac OS X now available, via virtualization.

Docker Architecture.



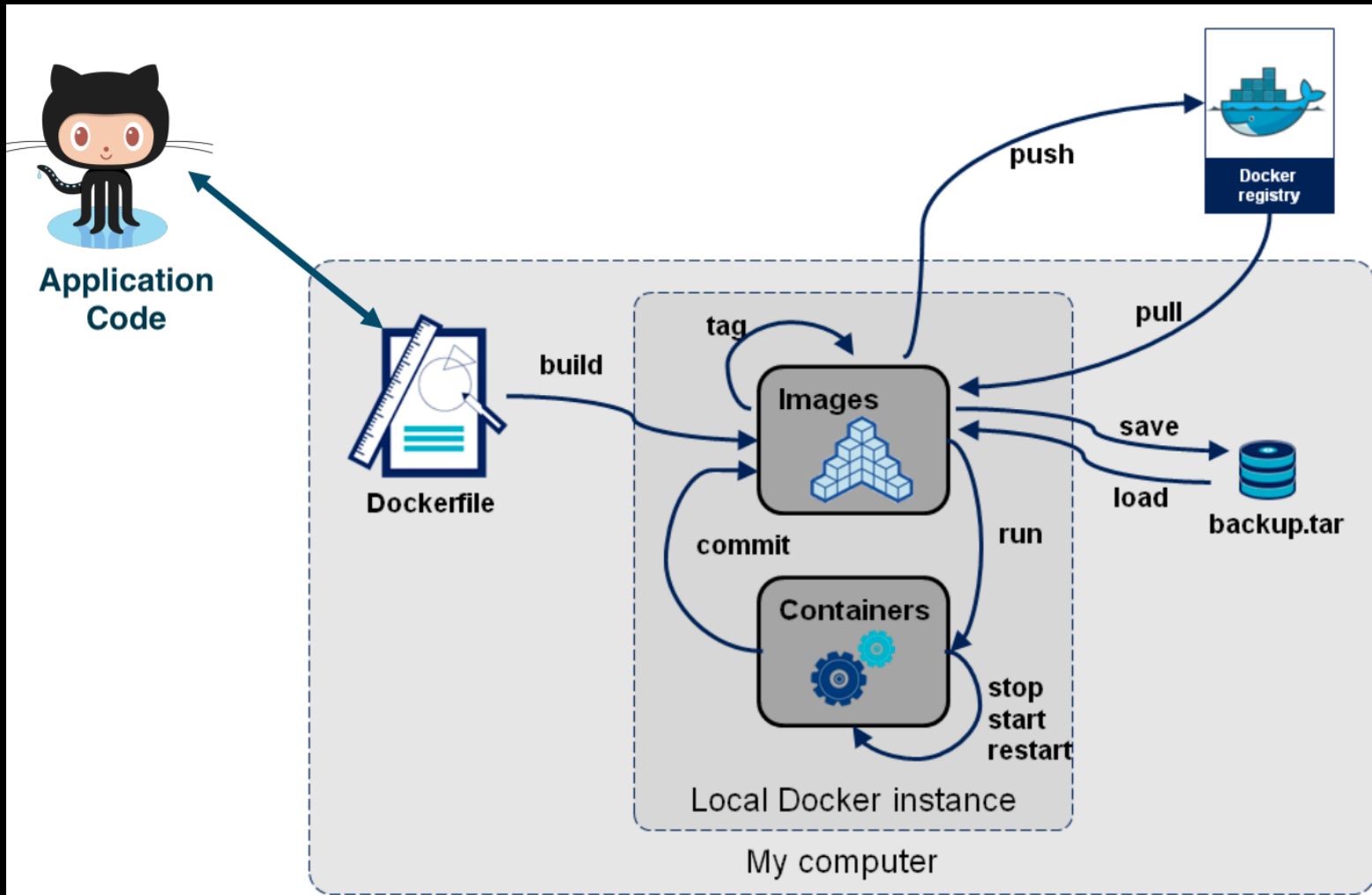
Docker Architecture.

- Docker Daemon.
 - Creates, ships and runs Docker containers that are deployable on a physical or virtual machine, hosted locally, in a datacentre or cloud service provider.
 - aka Docker Engine, Docker Runtime.
 - Provides an API for user interaction.
- Docker Client – the primary way a user interacts with the daemon.
- Registry.
 - Cloud or server based storage and distribution service for Docker images.
 - Docker Hub (default), AWS ECR, GitHub, GitLab.

Docker – Basics.

- Basic workflow:
 1. Create an image locally from a Dockerfile (set of instructions/directives).
 2. Instantiate container from the image
 3. Push (i.e. upload) the image to a remote registry, e.g. Docker Hub.
 4. Pull (i.e. download) the image on a target machine and run a container from the image.
- Basic commands:
 - `docker build` – create image
 - `docker push` – upload
 - `docker pull` – download
 - `docker run` – create container instance from image.
 - `docker ps` – list running containers
 - `docker stop/start` – stop/start container

Docker – Basics.



Docker – Command line Interface (CLI).

```
$ docker run httpd          (Apache Web Server image)
```

- What happens?
 1. Check if httpd image is available locally? No
 2. Download httpd image from Docker Hub registry and store in the local registry.
 3. Create a new container from the image
 - Allocate system resources – CPU, memory, storage
 - Assign IP address to container from the subnet of Docker's default virtual network.
 4. Container executes the image's start-up command, i.e. start the Apache web server.

Docker CLI – Image handling.

- Download an image from the registry:
`$ docker pull <image>`
- Upload an image to the registry:
`$ docker push <image>`
- Image names have the form:
 `[registry/][user/]name[:tag]`
 - Default registry is `registry-1.docker.io` (Docker Hub)
 - Default user is `library` (Official images)
 - default tag is `latest`.`$ docker pull`
 `registry-1.docker.io/library/httpd:latest`

Docker CLI – Image handling..

- List images in local registry:
\$ docker images
- Tagging – Give an image an alternative name (alias):
\$ docker tag <oldname> <newname>
e.g.
\$ docker tag
 registry-1.docker.io/library/httpd:latest
 mywebserver:1.0
\$ docker run mywebserver:1.0 ls /
- Delete an image locally:
\$ docker rmi <image>
– Stop and delete its containers first, if any

Docker CLI – Run.

- Start a new container
\$ docker run <imagename>
- Most used options
 - --name Give the container a symbolic name
 - -d Run container in background mode
 - -p Publish a container's port(s) to the host
 - -e Set environment variables in the container
 - -v Bind/mount a host volume to the container
 - --restart="no" Restart policy (no, on-failure[:max-retry], always)
 - --rm Automatically remove the container when it exits. Useful when experimenting.
 - -t Allocate a pseudo-TTY

Docker CLI – Run.

- The run command (contd.)
 - Publish port 80 from container as port 8080 on host: `-p 8080:80`
 - Mount local directory `/html` as directory `/usr/share/nginx/html` in the container:
 - `-v /html:/usr/share/nginx/html`
 - “Mount” just means “make available”.
 - `/usr/share/nginx/html` is where nginx web server expects HTML files.
- Example

```
$ docker run -d -p 8080:80
    --name nginx-server nginx:1.15.8-alpine
[NGINX is another popular web server and load balancer]
```

Docker CLI – Container handling.

- List containers
 - \$ docker ps # The running containers
 - \$ docker ps -a # All containers(includes exited ones)
- Stop running containers:
 - \$ docker stop <container..>
- Start stopped containers:
- \$ docker
- Remove containers:
 - \$ docker rm <container..>

Docker CLI – Interaction and Debugging.

- The exec command:
 - Run a command in an existing container,
\$ docker exec <containername/ID> <command>
e.g start a shell inside a container
\$ docker exec -it <container> /bin/bash
 - Some images only support /bin/sh
- See the logs (stdout) of a container.
\$ docker logs -f <container>
- Show Metadata of a container
\$ docker inspect <container-name/ID>