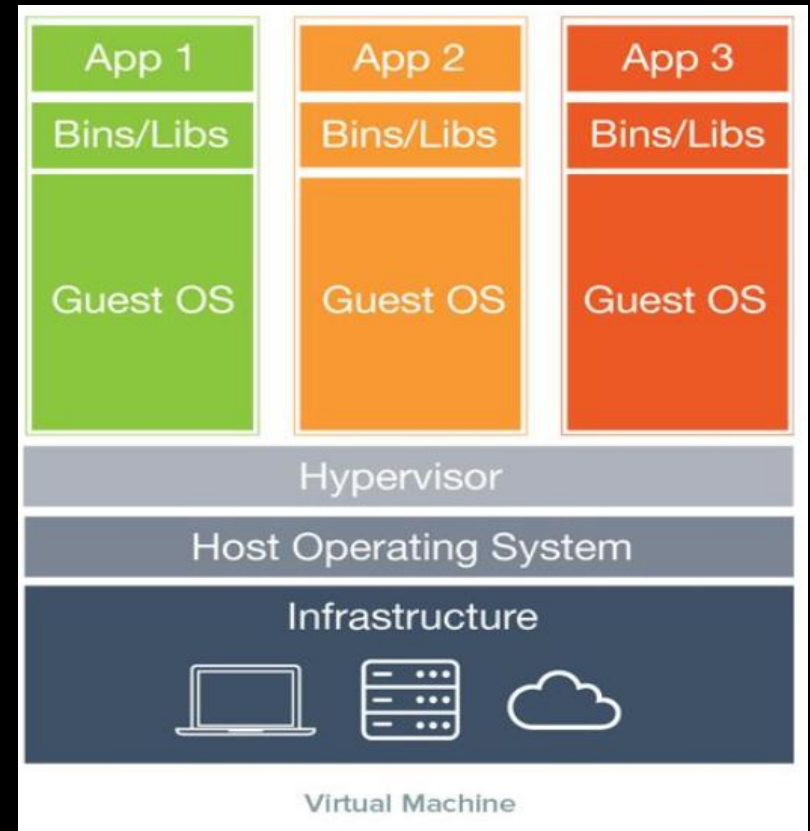


Containerization

Simplifying *Application Deployment* with Containers

Background - Virtualization.

- Virtualization is a process of creating an abstraction layer over hardware, allowing a single computer (host) to be divided into multiple virtual computers (guests). Each 'guests; uses part of the 'host' computer's hardware resources.
- Hypervisor s/w enables multiple guest OSs to run on top of the host OS.



Advantages of Virtualization

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



Advantages of Virtualization

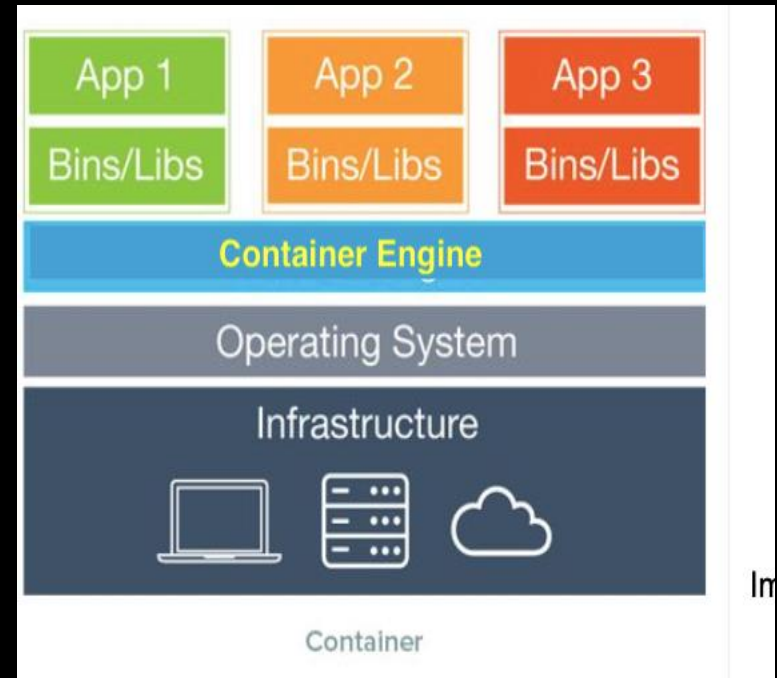
- Easier automation (Lower Operating Expenditure)
 - Simplified provisioning/administration of hardware and software.
- Improved Scalability.
- Greater h/w utilization.
- More flexibility.
 - Conserve power.
 - Free up unused physical resources

Problems of Virtualization

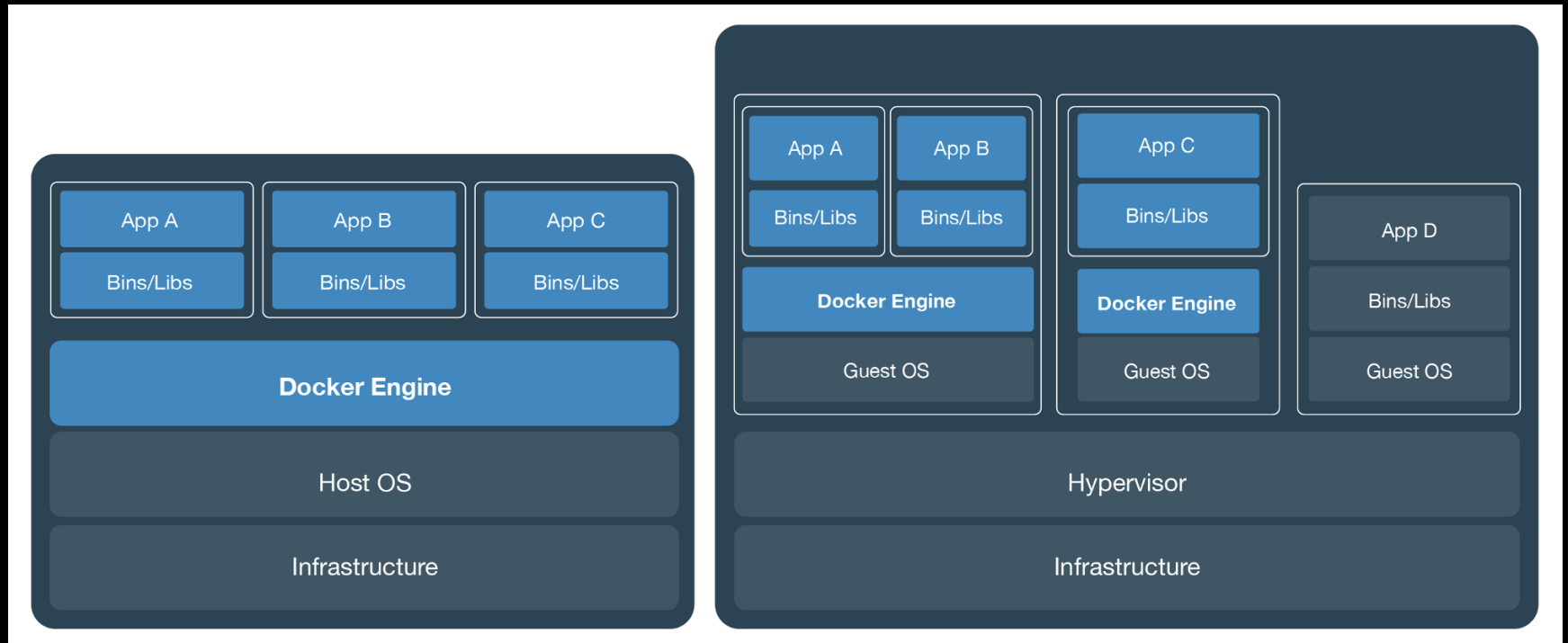
- Each VM contains a full operating system (OS). Each OS:
 - Requires a license \Rightarrow Capital Expenditure.
 - Has its own compute and storage overhead.
 - Needs maintenance/updates \Rightarrow Operating Expenditure.

Solution: Containers

- Containers:
 - Isolated environments to run apps/services (OS process isolation).
 - Portable.
 - Share the resources of a host OS kernel.
 - Disposable / Ephemeral.



Containers Versus VMs - Different, but not mutually exclusive.



Benefits of Containerization

- Portability: Application and its environments packaged in containers can run on any machine or cloud platform.
 - “... but it works on my computer!”
 - Dev, Test, Prod
- Scalability: scale an application by running multiple containers across multiple machines.
- Cost-effective – Containers are lightweight and use far less system resources than VMs.
- Self-contained - all system files and libraries an app needs are inside the container.
- Isolation – a container owns its own OS processes.

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 + libraries + configuration files).
 - The entire filesystem and metadata (e.g. environment variables) to run an app.
 - An immutable template for a containers.
- Container: The runtime instance of an image.
 - An OS process isolated from the rest of the system through abstractions created by the OS.
 - A container's filesystem comes from an image.

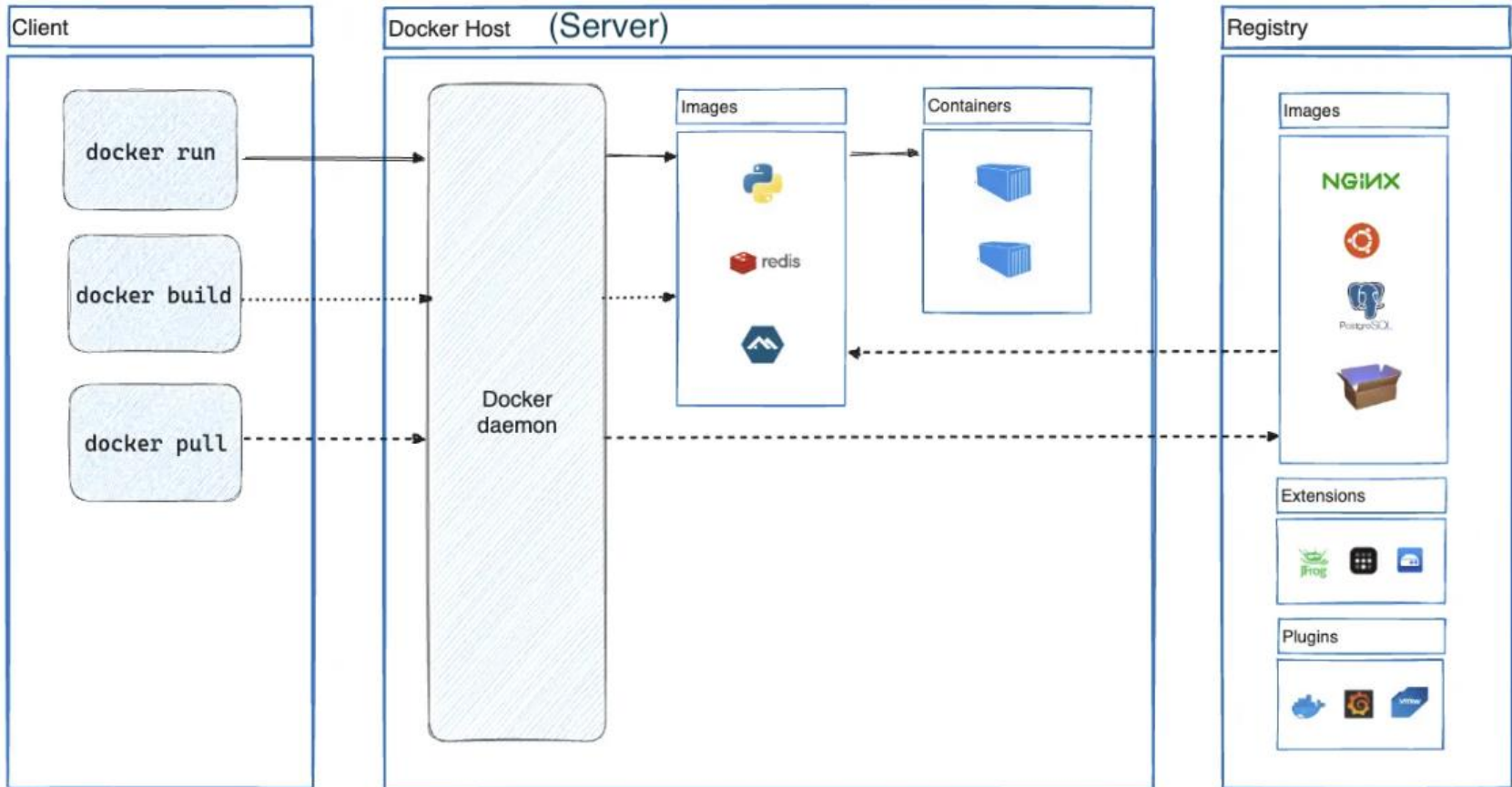


The Docker Container Engine

Docker.

- An open-source platform for creating and managing container-based environments.
- Simplifies app development and deployment by providing a consistent experience across different machines.
- Written in Go language; Released in 2013; Linux-based.
- Docker Desktop – Docker for Windows and Mac OS X; uses virtualization internally.

Docker Architecture.



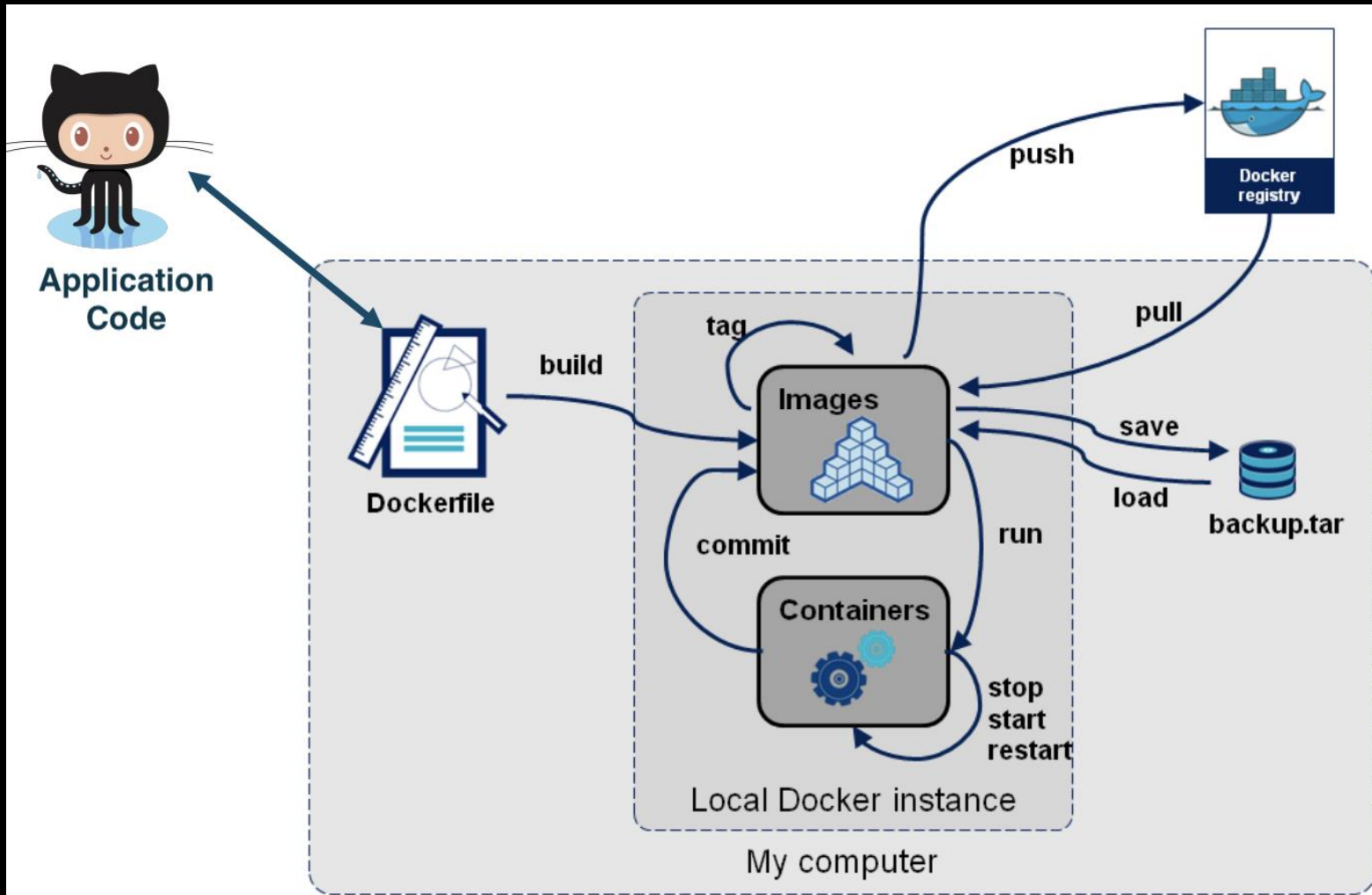
Docker Architecture.

- Docker Daemon.
 - Creates and runs containers.
 - Works on a physical or a virtual machine hosted locally, in a datacentre or cloud provider.
 - aka Docker Engine, Docker Runtime.
 - Provides an API for client interaction.
- Docker Client – allow users interact with the daemon.
- Registry.
 - Cloud-based storage and distribution service for Docker images.
 - E.g. Docker Hub (default), AWS ECR, GitHub.

Docker – Basics.

- Basic workflow:
 1. Develop your application code.
 2. Write a Dockerfile to describe the steps (instructions) to create an image of your app.
 3. Instantiate a container from the image.
 4. Push (upload) the image to a remote registry, e.g. Docker Hub.
 5. Pull (download) an 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?
 2. If not, download image from Docker Hub and store in the local registry.
 3. Create a new container from the image.
 - System resources allocated – 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 a remote registry:
\$ docker pull <image>
- Upload an image to a remote registry:
\$ docker push <image>
- Format for image fully qualified name:
[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 stored in your local registry:
\$ docker images
- Tagging - Give an image an alternative name (alias):
\$ docker tag <original_name> <alias>
- Delete an image locally:
\$ docker rmi <image>
 - Must stop and delete any related containers first

Docker CLI – Run.

- Start a new container

`$ docker run [options] <image-name>`

- 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.
- `--rm` Automatically remove the container when it exits. Useful when experimenting.
- `-it` Opens an interactive terminal session.
- `--restart="no"` Restart policy (no, on-failure[:max-retry], always)

Docker CLI – Run (Contd.).

- Command options examples:
 - Publish the container's port 80 to port 8080 on the host: *-p 8080:80*
 - Mount the host's */web/html* directory to */usr/share/nginx/html* in the container:

-v /web/html:/usr/share/nginx/html

- Full Example:

```
$ docker run -d -p 8080:80 --name nginx-server \
```

```
  -v /web/html:/usr/share/nginx/html
```

```
  nginx:1.15.8
```

[NGINX is popular web server and load balancer]

Docker CLI – Container handling.

- List containers:
\$ docker ps # Only running containers
\$ docker ps -a # Running and exited ones.
- Stop a running container:
\$ docker stop <container-name>
- Start stopped containers:
\$ docker start <container-name>
- Remove container:
\$ docker rm <container-name>

Docker CLI – Interaction and Debugging.

- The exec command:
 - Execute a command inside a running container,
\$ docker exec <container-name/ID> <command>
e.g start a shell inside a container
\$ docker exec -it <container> /bin/bash
 - Some images only support /bin/sh
- Show the logs (stdout) of a container:
\$ docker logs -f <container>
 - -f streams the output.
- Show the metadata of a container
\$ docker inspect <container-name/ID>


Named Volumes in Compose.

```
services:
  frontend:
    image: node:lts
    volumes:
      - myapp:/home/node/app
```

```
volumes:
```

```
  myapp:
```

Create new volume



```
services:
  frontend:
    image: node:lts
    volumes:
      - myapp:/home/node/app
```

```
volumes:
```

```
  myapp:
```

```
    external: true
```

Use an existing volume

