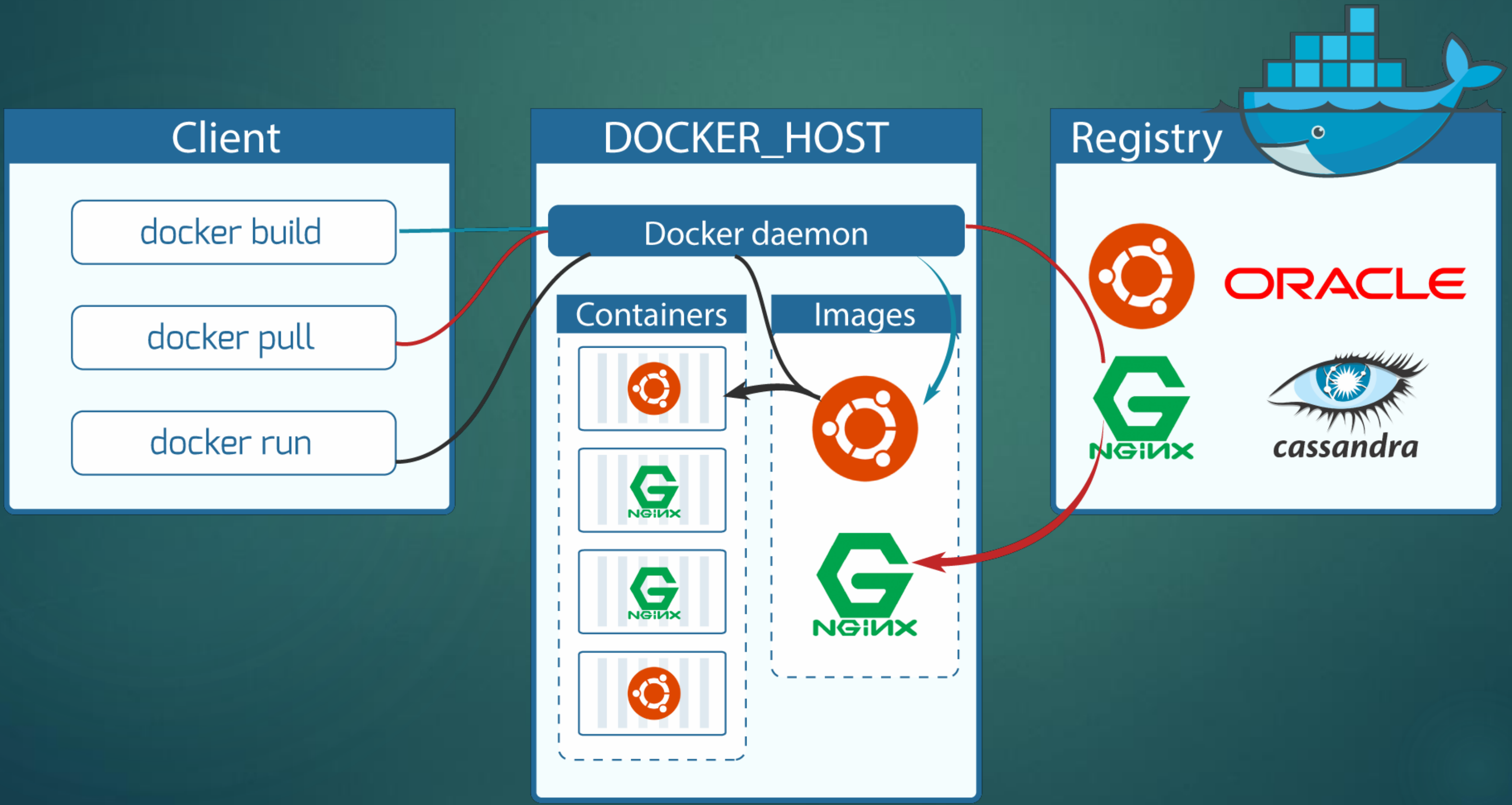
Docker Architecture and Components

Docker Architecture

DOCKER COMPONENTS



Docker Components

- **The Daemon** (dockerd) is responsible for distributing, building and running the containers. It listens to the requests by the client. It manages objects such as images, containers, networks and volumes.

- **The Client** (docker) is the docker user. A client can communicate with multiple docker daemons. Docker client communicates through **REST API** with the docker daemon which is available on the same machine or can be on a remote machine.

- **A Docker Registry** stores images of applications. Docker store allows you to buy and sell Docker images or distribute them for free. There are public registries available or you can even create a private registry.

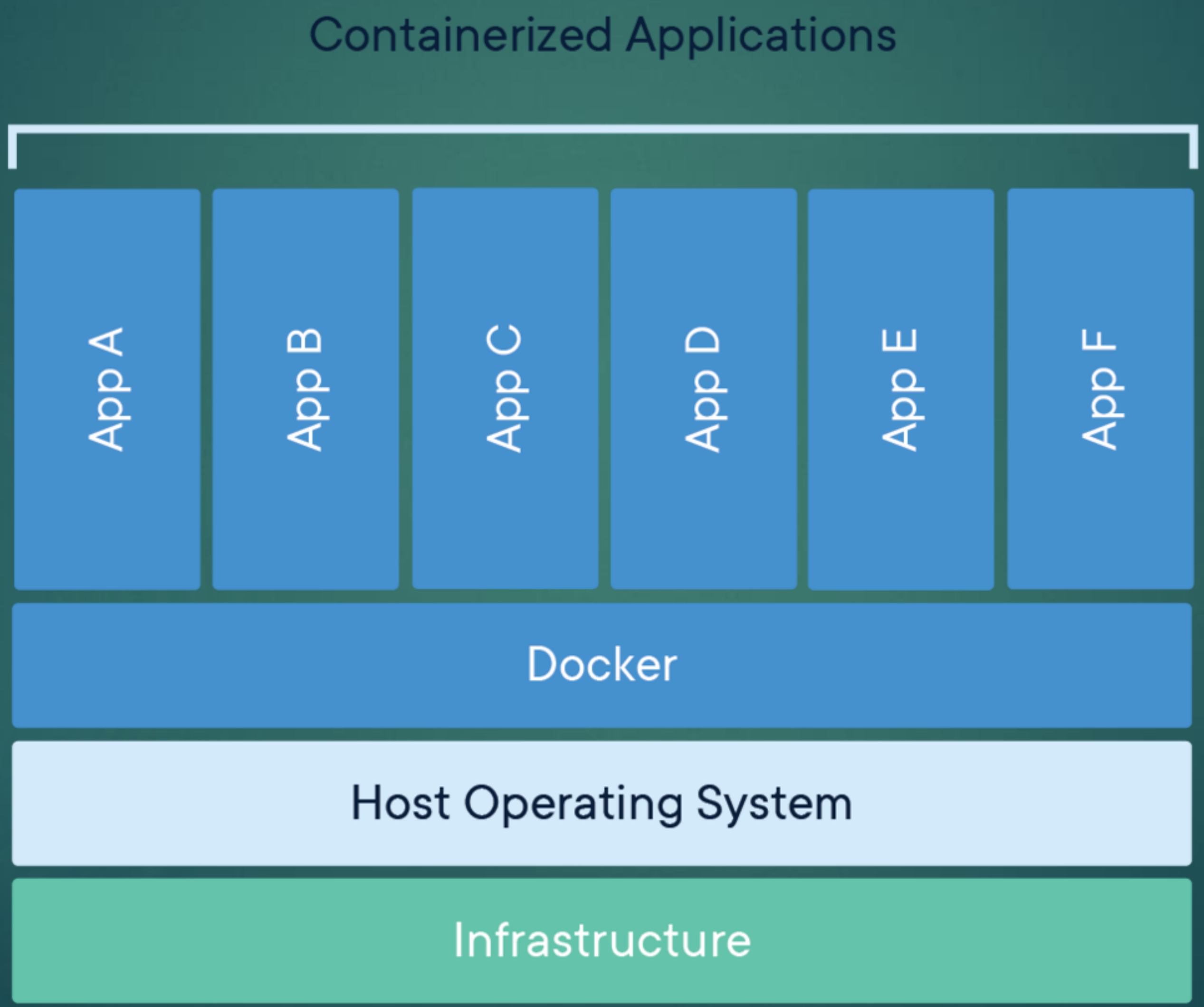
- Docker Host:
- Provides a complete environment to execute and run applications.
- Comprises of Docker Daemon , Images, Containers ,Networks and Storage

Images

- Images are a core part of the Docker experience as they enable collaboration between developers in a way
- Read-only binary template used to build containers.
- They contain metadata that describe the container's capabilities and needs.
- It is used to store and ship applications.
- An image can be used on its own to build a container or customized to add additional elements to extend the current configuration.
- Container images can be shared across teams within an enterprise using a private container registry, or shared with the world using a public registry like Docker Hub.

Eg: Hello World , Apline , Centos etc..

Containers


Containerized Applications
App A
App B
App C
App D
App E
App F
Docker
Host Operating System
Infrastructure

Containers

- A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

- Containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

- Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs

Thank you