Containers & Container Services

Cloud Computing

Brenden west

Contents

Learning Outcomes

- Overview of containers
- Using Docker
- Using AWS container services

Reading

- Cloud Computing: Concepts, Technology, Security, and Architecture - Ch. 6 & Appendix B.1
- https://docs.docker.com/get-started/overview/
- https://docs.docker.com/guides/get-started/
- AWS Cloud Foundations - Module 6, Sec. 6 - Container Services
- AWS Cloud Developing - Module 8 - Container Services

Intro to Containers

- **Containerization** - virtualization technology for running applications without need to deploy a virtual server for each
- **Container** - virtual, isolated environment that provides only resources required for the program it hosts
- **Image** - predefined template used to create containers
- **Pod** - logical host of one or more containers with shared stored, network resources, & configuration
- **Host** - OS environment in which a container is deployed. Can support multiple containers or pods.
- **Host cluster** - multiple hosts (**nodes**) combined as a pool of available processing resources. Created in advance of usage.

Containerization Benefits

- Can create an isolated, solution-specific environment with minimal infrastructure needs
- Can be scaled rapidly & effectively
- Can be resilient and regenerated in response to failure
- Easier than virtual servers to create & deploy
- Can be ported across hosting environments w/o changes
- Allows specification & version tracking of app code & dependencies

Container Engine

- Creates and runs one or more containers
- Can read images from and publish to a **container registry**
- Mediates container access to OS kernel functions
- Able to interact with kernels from different OS's
- Can run on physical or virtual servers
- Can run on **Type 1** (physical server has no OS) or **Type 2** environments
- Abstracts resources from underlying server OS
- Exposes **management plane** - GUI & command-line tools
- Carries out **control plane** functions automatically or in response to commands

Container Networks

- Containers on same host can communicate with each other via **host network**
- Related containers & engineers can communicate across hosts via an **overlay network**
- Admins can configure networks to control access to external resources
- A deployed container receives a network (IP) address for each network it participates in
- Containers in the same pod share a network address & are identified through different network ports

Container Deployment

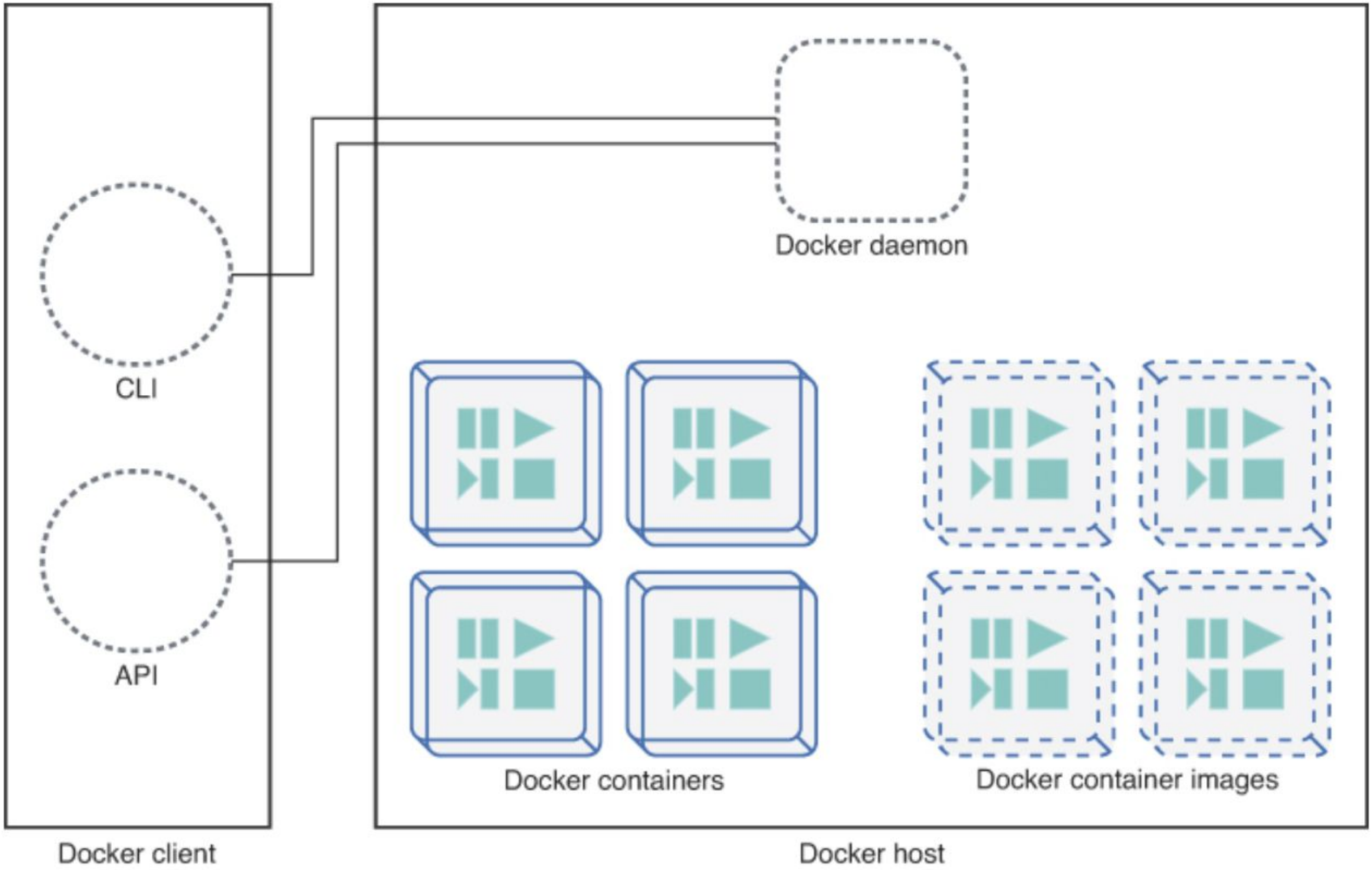
- A **package manager** coordinates the initial deployment of containers based on predefined workflow logic
- A **container orchestrator** automates the deployment, scaling, and management of containerized applications in a distributed computing environment

Container Images

- **Base** (or **partial**) images are templates for container images
- **Customized** images are created by the container image to create actual, deployed containers. Can be saved as new base images.
- Defined by a **build file**, a human-editable configuration file
- Content organized into **layers**, each corresponding to a build-file statement
- Created images are **immutable** & build-file contents can’t be modified
- Each image has a unique auto-generated key
- Includes non-kernel parts of an OS required by the hosted application

Docker

- **Server** - a host running a Docker containerization engine (**daemon**)
- **Daemon** - mediates interaction with containers on a host through REST-based APIs
- **Client** - tools (e.g. CLI, API) to enable interaction with a Docker server to deploy & manage containers
- **Registry** - Image repository used by host to deploy containers. Can be private, but Docker provides a public repository of standard images - https://hub.docker.com/_/registry
- **Dockerfile -** a text-based file containing a script of instructions that Docker uses to build a container image


Docker daemon
CLI
API
Docker containers
Docker container images
Docker client
Docker host