Docker

Some slides from Martin Meyer

Agenda

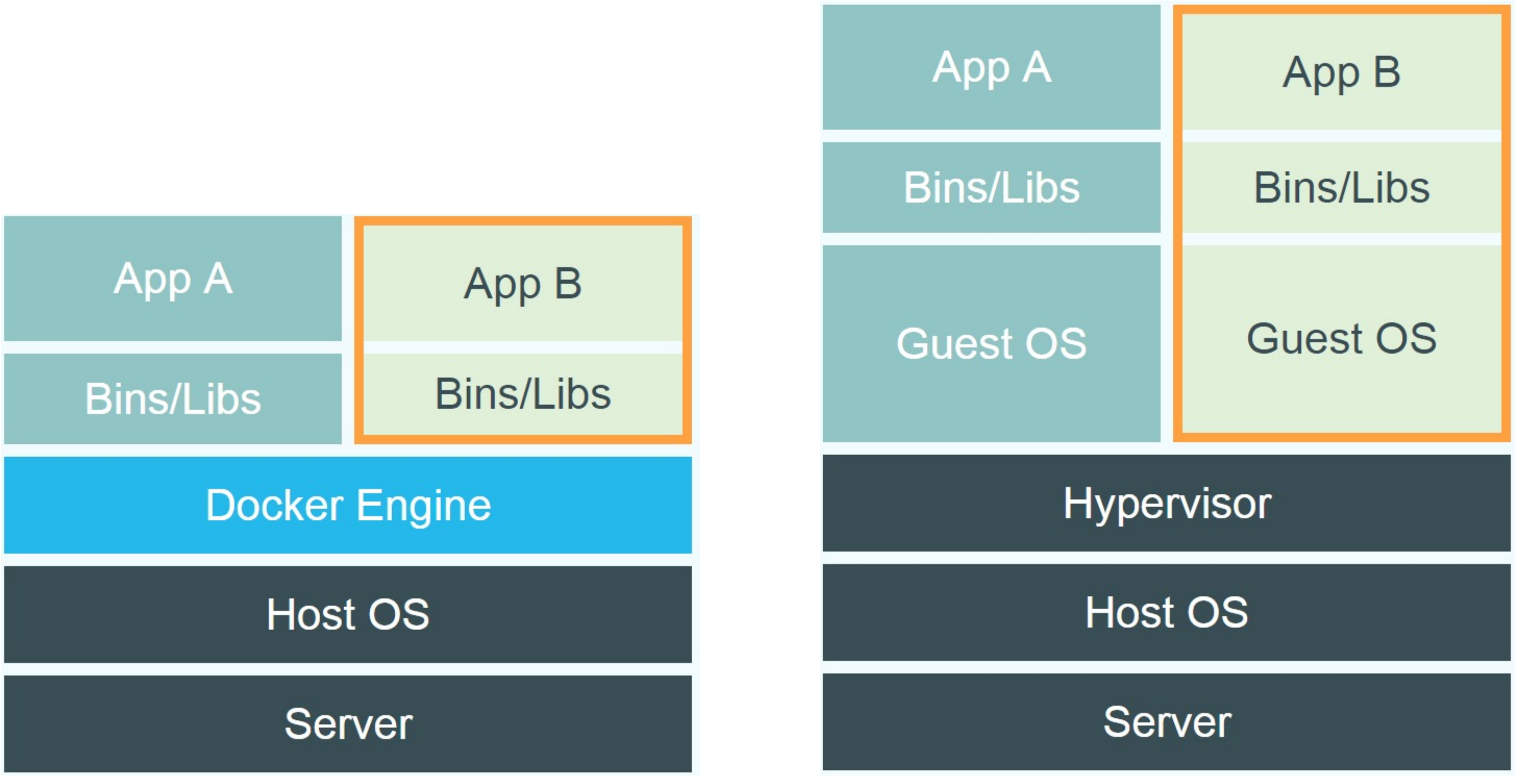
- What is Docker from an OS organization perspective?
  - Docker vs. Virtual Machine
  - History, Status, Run Platforms

- Use cases

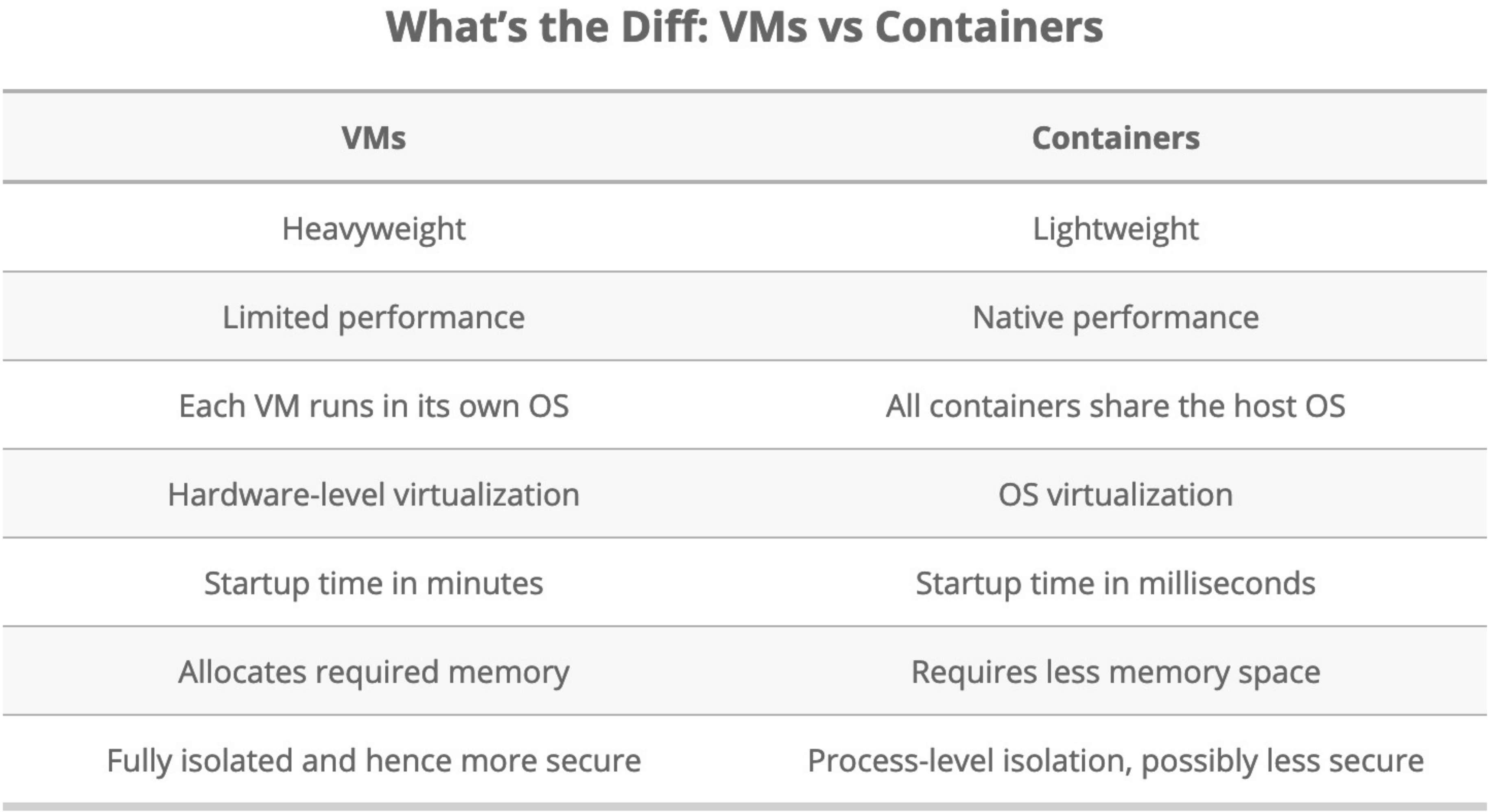
Containers

- Virtualize the OS, not the full machine
- Container sits on host OS kernel, and some shared binaries
  - These are read only
- Sharing OS resources significantly reduces footprint
  - Containers are lightweight – megabytes in size
    - Smaller snapshots, can have many more on one physical machine
  - VMs are an order of magnitude or more larger
    - Take longer to launch, etc…

Docker vs. Virtual Machine



Source: https://www.docker.com/whatisdocker/

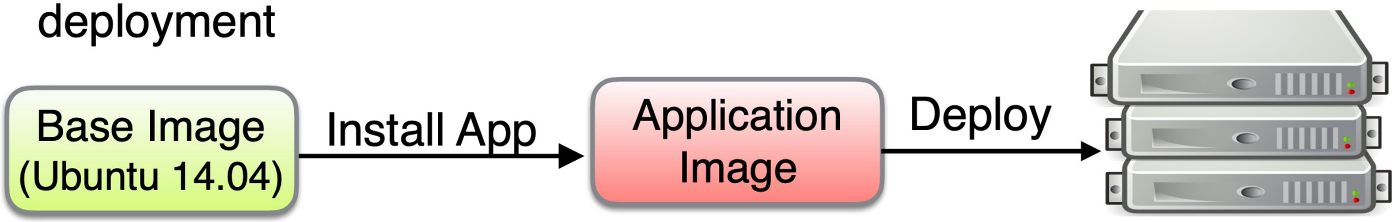


What's the Diff: VMs vs Containers

VMs	Containers
Heavyweight	Lightweight
Limited performance	Native performance
Each VM runs in its own OS	All containers share the host OS
Hardware-level virtualization	OS virtualization
Startup time in minutes	Startup time in milliseconds
Allocates required memory	Requires less memory space
Fully isolated and hence more secure	Process-level isolation, possibly less secure

Performance comparison

- Getting applications from development to production involves creating disk images
- Fast image creation enables rapid testing and continuous deployment

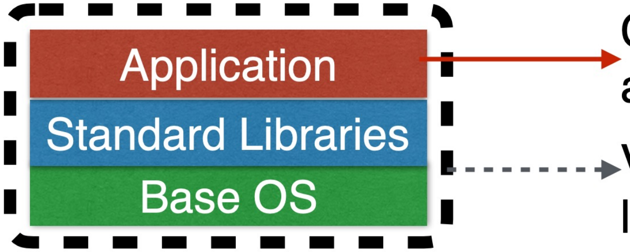


Base Image (Ubuntu 14.04)

Install App

Application Image

Deploy



Application

Standard Libraries

Base OS

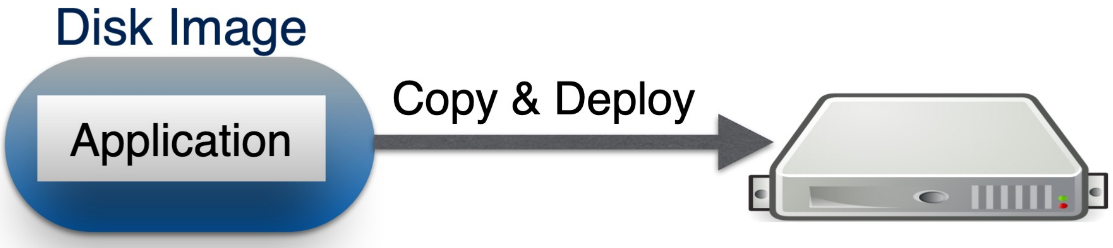
Container images have only application layer

VMs contain entire OS, and have larger images

Time (s)	VM (Vagrant)	Docker
MySQL	236	129
NodeJS	304	49

- Docker: 2-6x faster

Size comparison



Disk Image

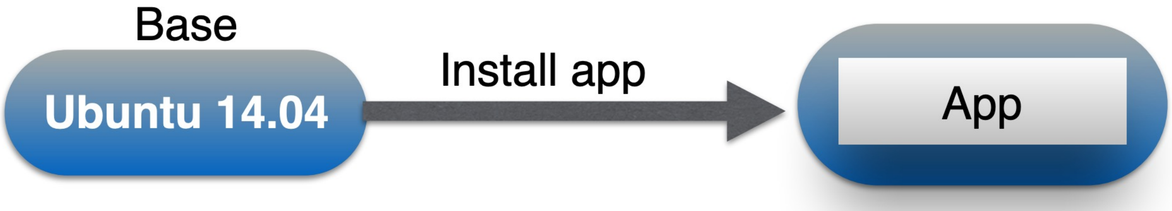
Application

Copy & Deploy

Image size	VM	LXC	Docker
MySQL	1.68 GB	0.4 GB	**112 KB**
NodeJS	2.05 GB	0.6 GB	**72 KB**

Docker: 2-6x smaller

- VMs contain entire OS, and have larger images
- Docker stores only differences (application layer)



Base

Ubuntu 14.04

Install app

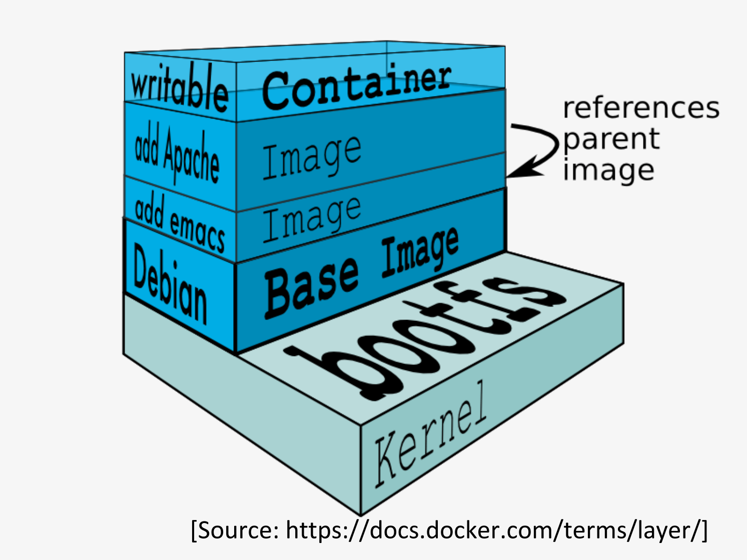
App

Additional discussion points

- Name spaces, and unix jail
- VMs include a separate OS image, adding complexity to all stages of development lifecycle
  - Limits portability between clouds and data centers
- Performance isolation:
  - Unix cgroups can provide isolation for CPU, memory, I/O and network

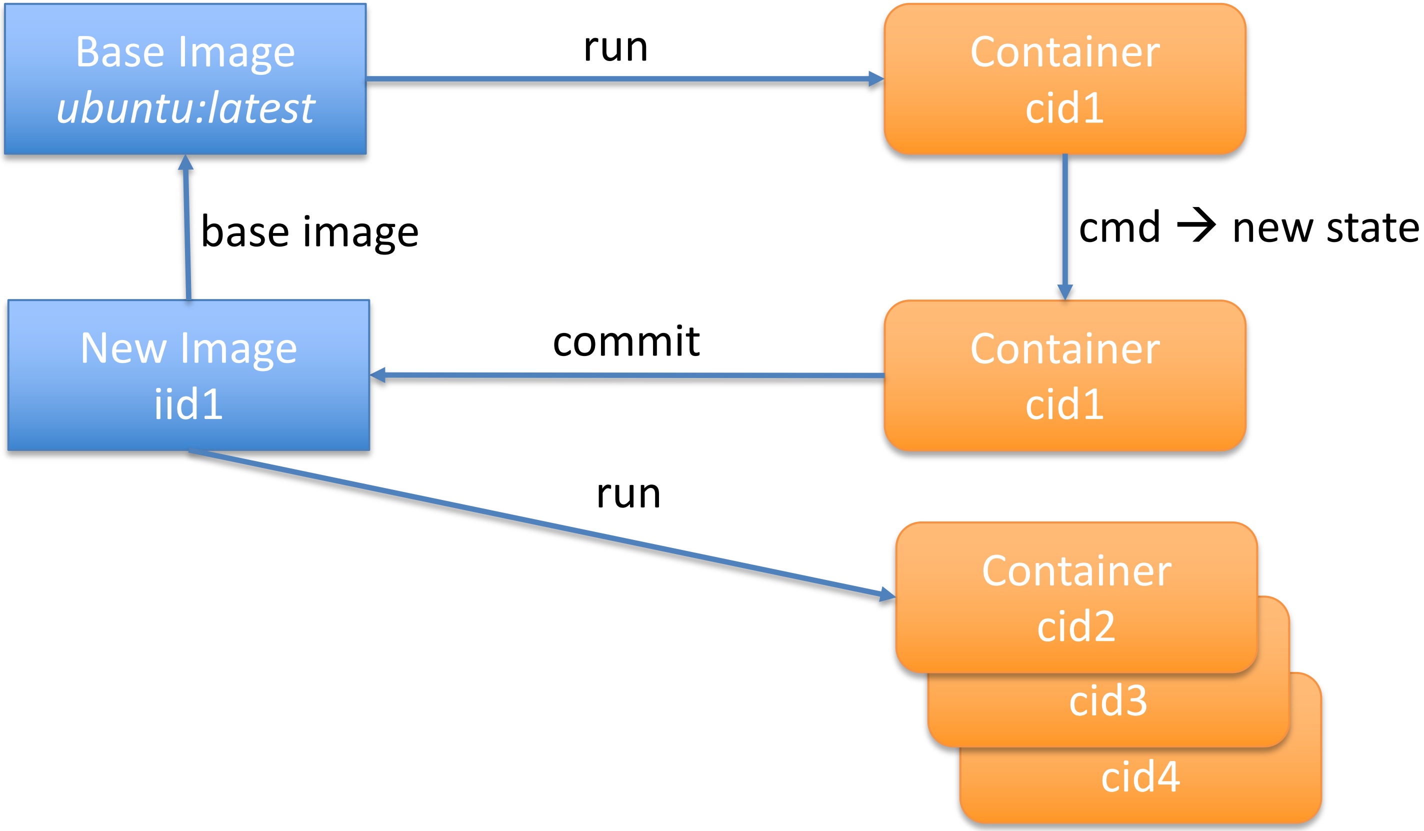
Docker Technology

- libvirt: Platform Virtualization
- LXC (LinuX Containers): Multiple isolated Linux systems (containers) on a single host
- Layered File System



[Source: https://docs.docker.com/terms/layer/]

Image vs. Container



Base Image
ubuntu:latest

run

Container
cid1

cmd → new state

Container
cid1

commit

New Image
iid1

base image

run

Container
cid2

cid3

cid4

Dockerfile

- Create images automatically using a build script: «Dockerfile»
- Can be versioned in a version control system like Git along with all dependencies
- Docker Hub can automatically build images based on dockerfiles on Github

Docker Use Cases

- Development Environment
- Environments for Integration Tests
- Quick evaluation of software
- Microservices
- Multi-Tenancy
- Unified execution environment (dev → test → prod (local, VM, cloud, ...)

Use-case: scientific reproducibility

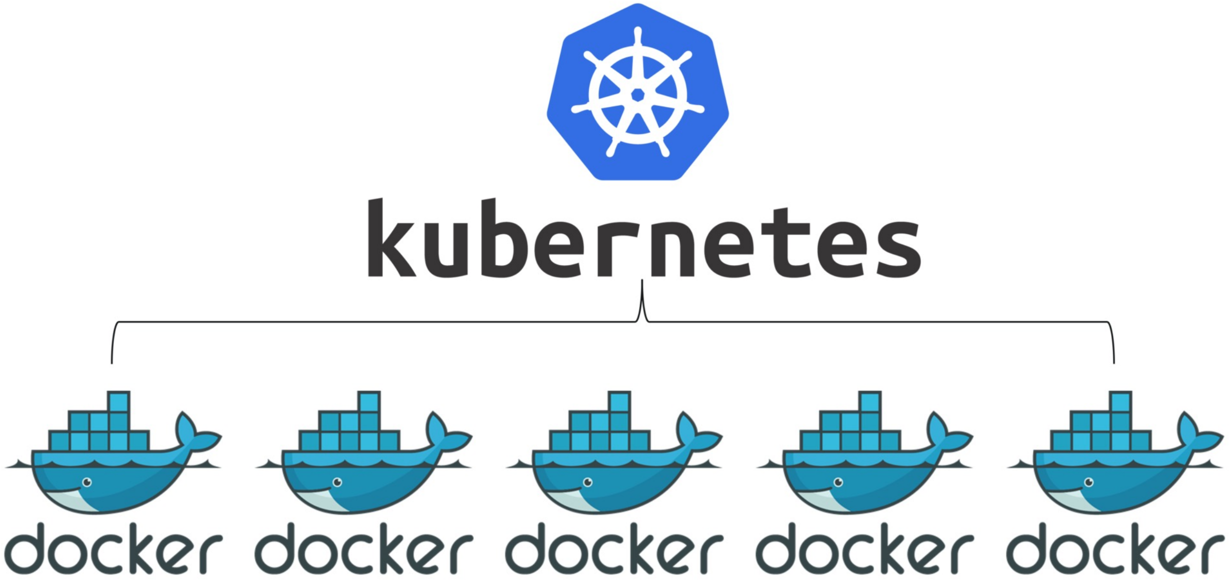
- Dependency hell
  - Less than 50% of software could be built or installed
  - Difficult to reproduce computational environment
- Imprecise documentation
  - Difficult to figure out how to install
- Code rot
  - Software dependencies change, affecting results
- Barriers to adoption and reuse
  - Difficulty to coordinate build tools/package managers

Use case: scientific reproducibility

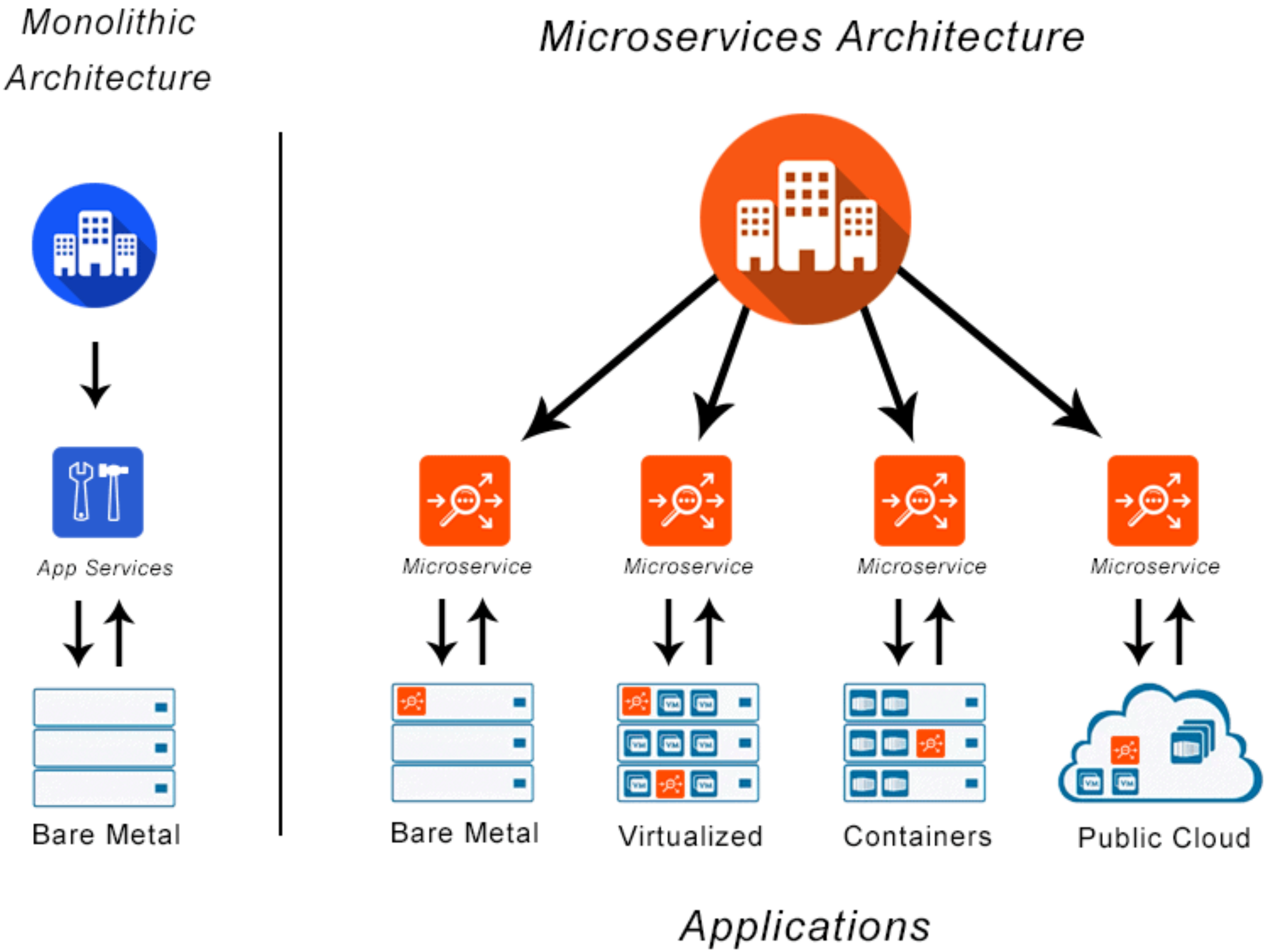
- Dependency hell
  - Docker! Container includes everything!
- Imprecise documentation
  - Dockerfiles keep record of dependencies
- Code rot
  - versioning
- Barriers to adoption and reuse
  - Argues that docker provides features to help with that

Use case 2: Kubernetes/Microservices

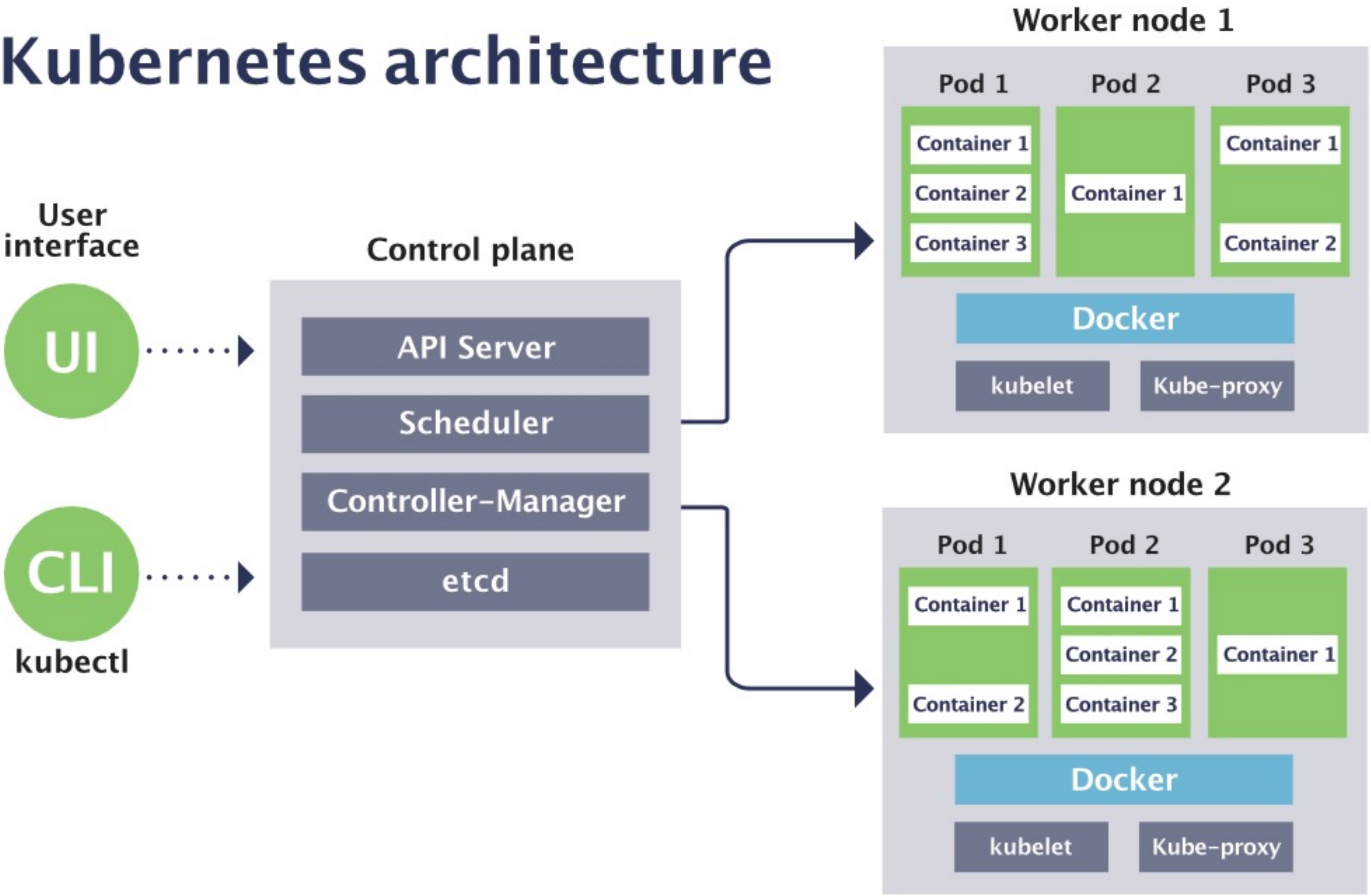
- How to use containers to provide services on the cloud?
- Rapid ramp up enables *micro-services*



What are microservices?



Kubernetes architecture



User interface

UI

CLI

kubectl

Control plane

API Server

Scheduler

Controller-Manager

etcd

Worker node 1

Pod 1

Container 1

Container 2

Container 3

Pod 2

Container 1

Pod 3

Container 1

Container 2

Docker

kubelet

Kube-proxy

Worker node 2

Pod 1

Container 1

Container 2

Pod 2

Container 1

Container 2

Container 3

Pod 3

Container 1

Docker

kubelet

Kube-proxy