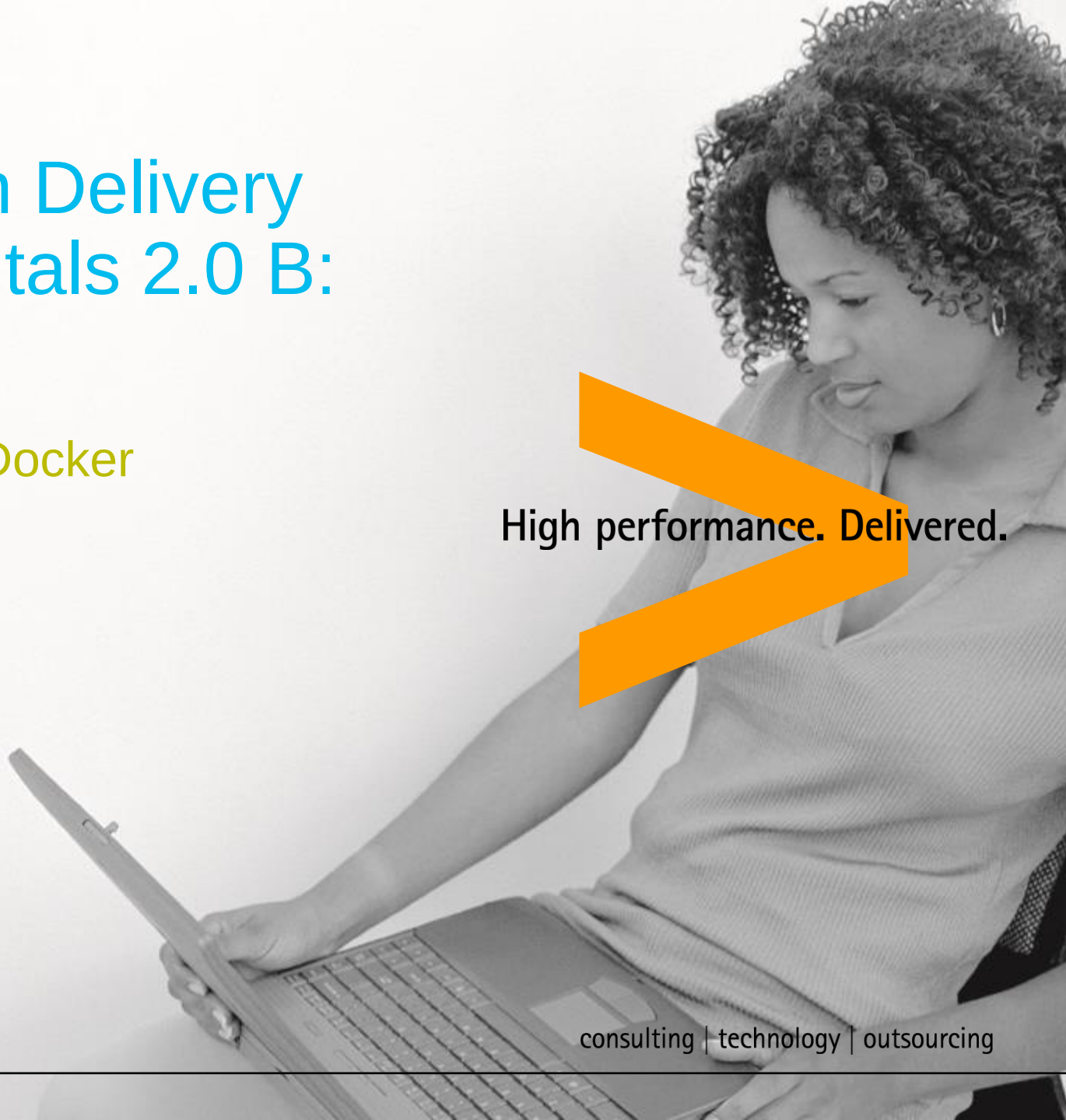
Application Delivery Fundamentals 2.0 B: Java

Introduction to Docker



consulting | technology | outsourcing

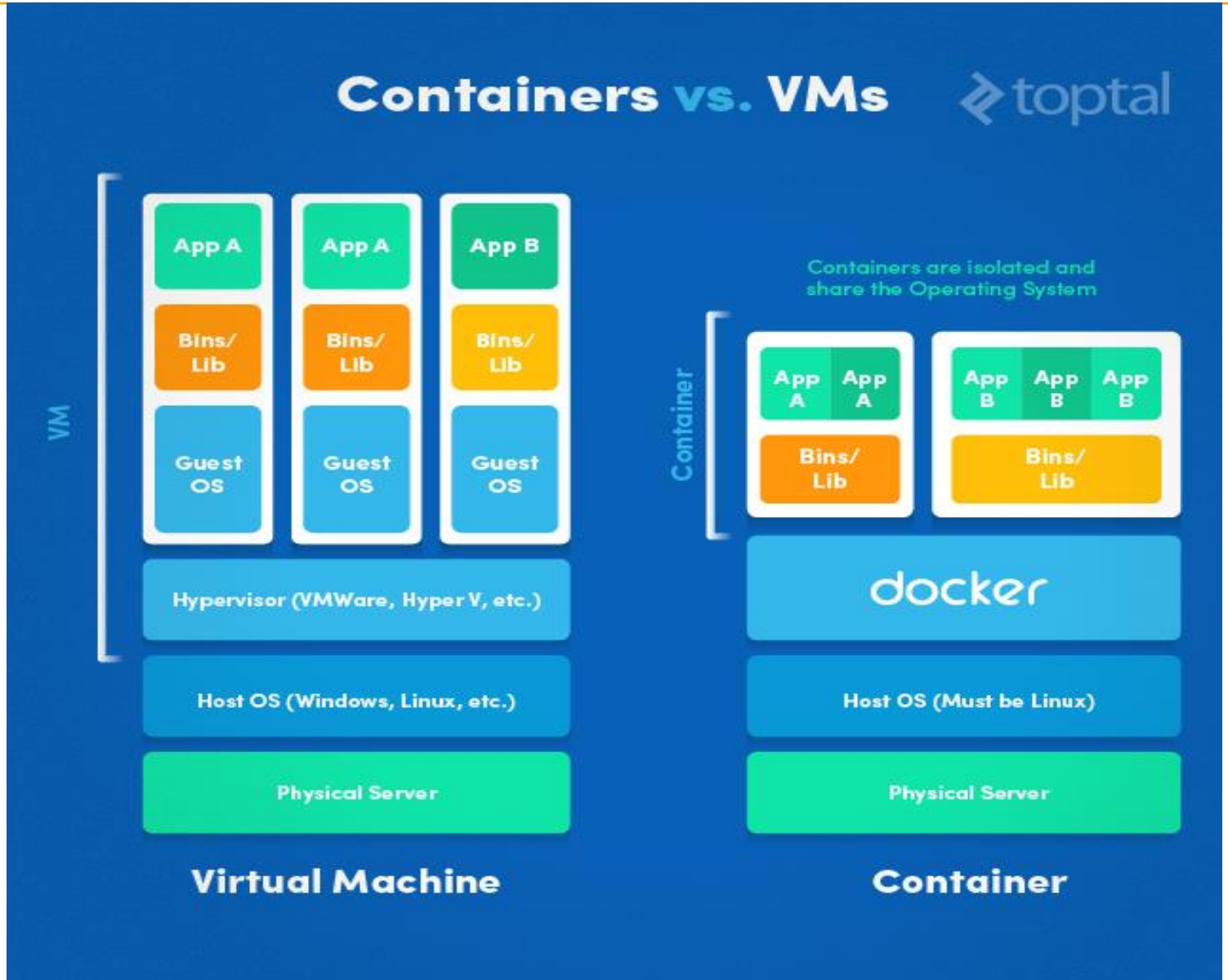
Goals

- Docker
- Docker Architecture
- Images
- Containers
- Volume
- Networks
- Docker Hub
- Docker Swarm

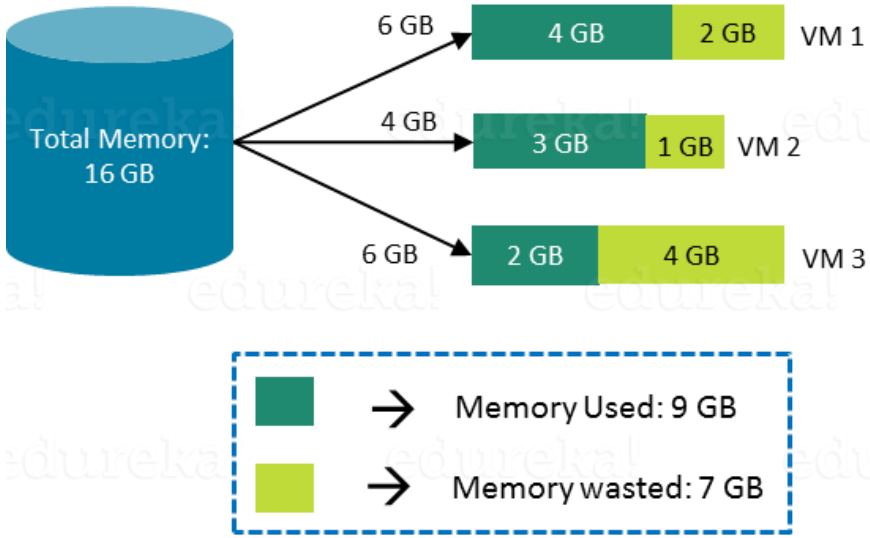
Docker

- Docker is an excellent tool for managing and deploying microservices.
- Each microservice can be further broken down into processes running in separate Docker containers, which can be specified with Dockerfiles and Docker Compose configuration files.
- Combined with a provisioning tool such as Kubernetes, each microservice can then be easily deployed, scaled, and collaborated on by a developer team.
- Specifying an environment in this way also makes it easy to link microservices together to form a larger application.

Docker


Containers vs. VMs
toptal
VM
App A
App A
App B
Bins/ Lib
Bins/ Lib
Bins/ Lib
Guest OS
Guest OS
Guest OS
Hypervisor (VMWare, Hyper V, etc.)
Host OS (Windows, Linux, etc.)
Physical Server
Virtual Machine
Containers are isolated and share the Operating System
Container
App A
App A
App B
App B
App B
Bins/ Lib
Bins/ Lib
docker
Host OS (Must be Linux)
Physical Server
Container

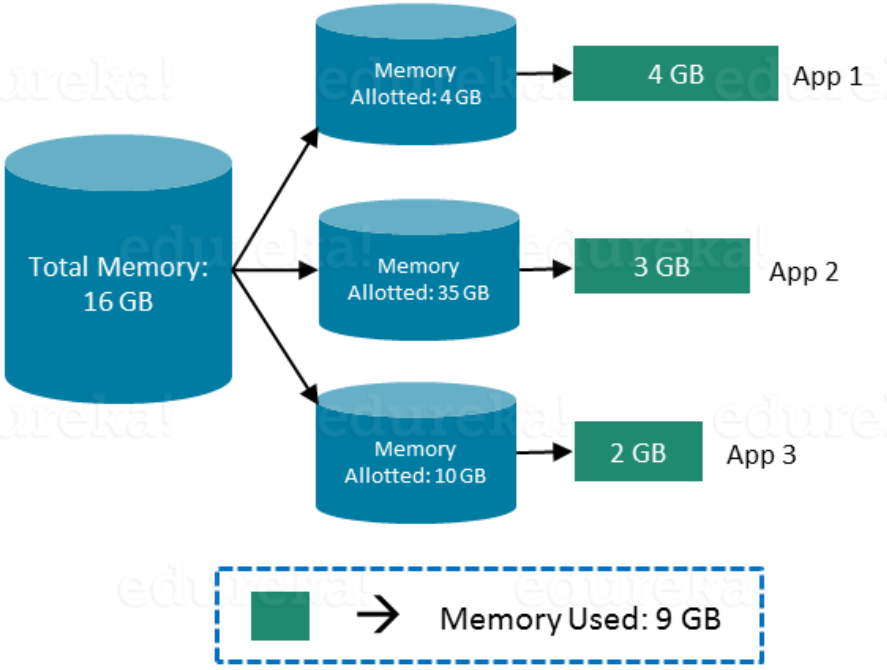
In case of Virtual Machines



7 Gb of Memory is blocked and
cannot be allotted to a new VM

In case of Docker

edureka!

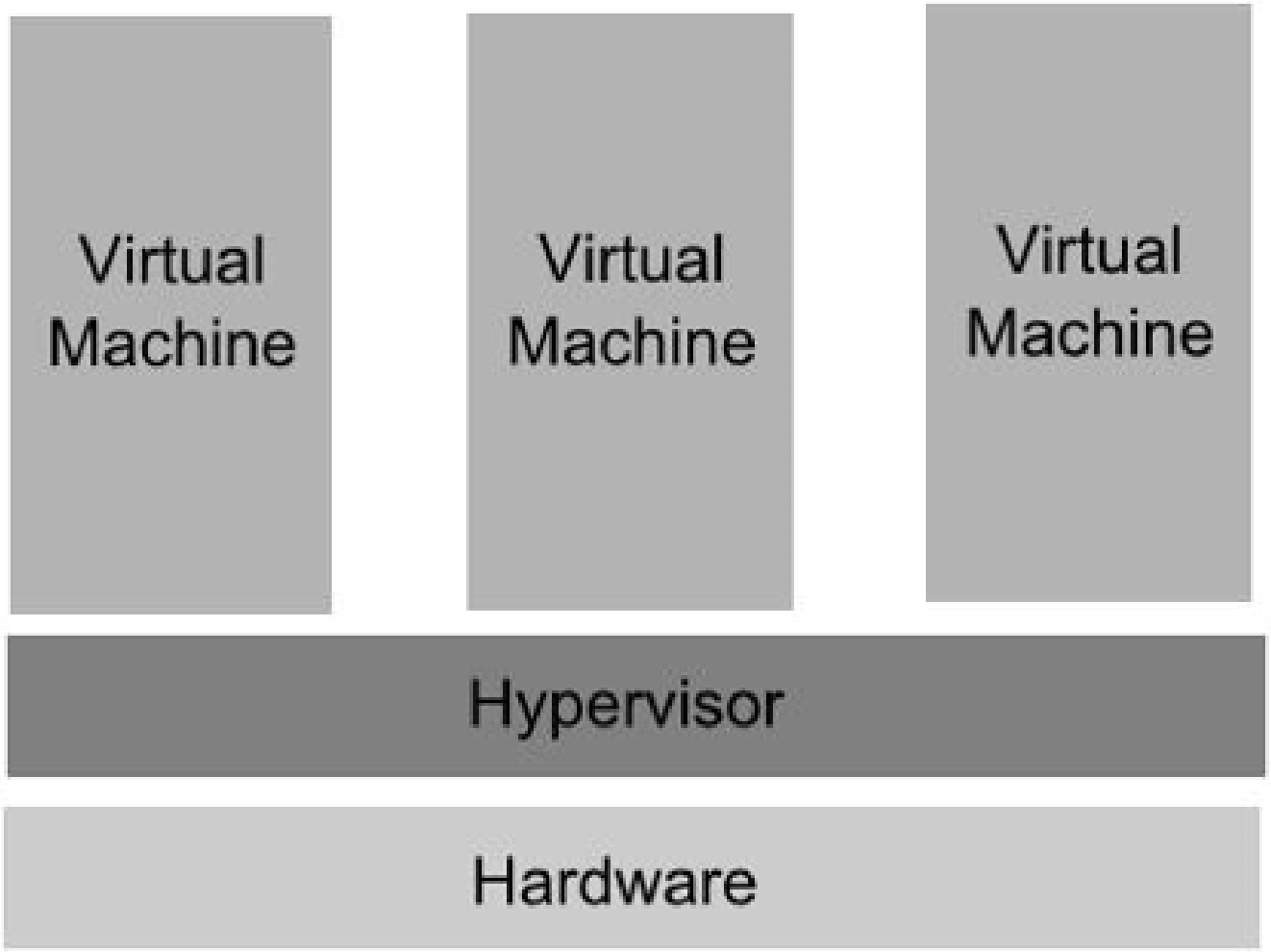


Only 9 GB memory utilized;
7 GB can be allotted to a new Container

Hypervisor

- A hypervisor is a software that makes virtualization possible.
- It is also called Virtual Machine Monitor.
- It divides the host system and allocates the resources to each divided virtual environment.

Hypervisor


Virtual Machine
Virtual Machine
Virtual Machine
Hypervisor
Hardware

Hypervisor

- Hyper-V will only run on processors which support hardware assisted virtualization.
- The x86 family of CPUs provide a range of protection levels also known as rings in which code can execute.
- Ring 0 has the highest level privilege and it is in this ring that the operating system kernel normally runs.
- Code executing in ring 0 is said to be running in system space, kernel mode or supervisor mode.
- All other code, such as applications running on the operating system, operate in less privileged rings, typically ring 3.

Hypervisor

- Under Hyper-V hypervisor virtualization a program known as a hypervisor runs directly on the hardware of the host system in ring 0.
- The task of this hypervisor is to handle tasks such CPU and memory resource allocation for the virtual machines in addition to providing interfaces for higher level administration and monitoring tools.

Hypervisor

- If the hypervisor is going to occupy ring 0 of the CPU, the kernels for any guest operating systems running on the system must run in less privileged CPU rings.
- Unfortunately, most operating system kernels are written explicitly to run in ring 0 for the simple reason that they need to perform tasks that are only available in that ring, such as the ability to execute privileged CPU instructions and directly manipulate memory.

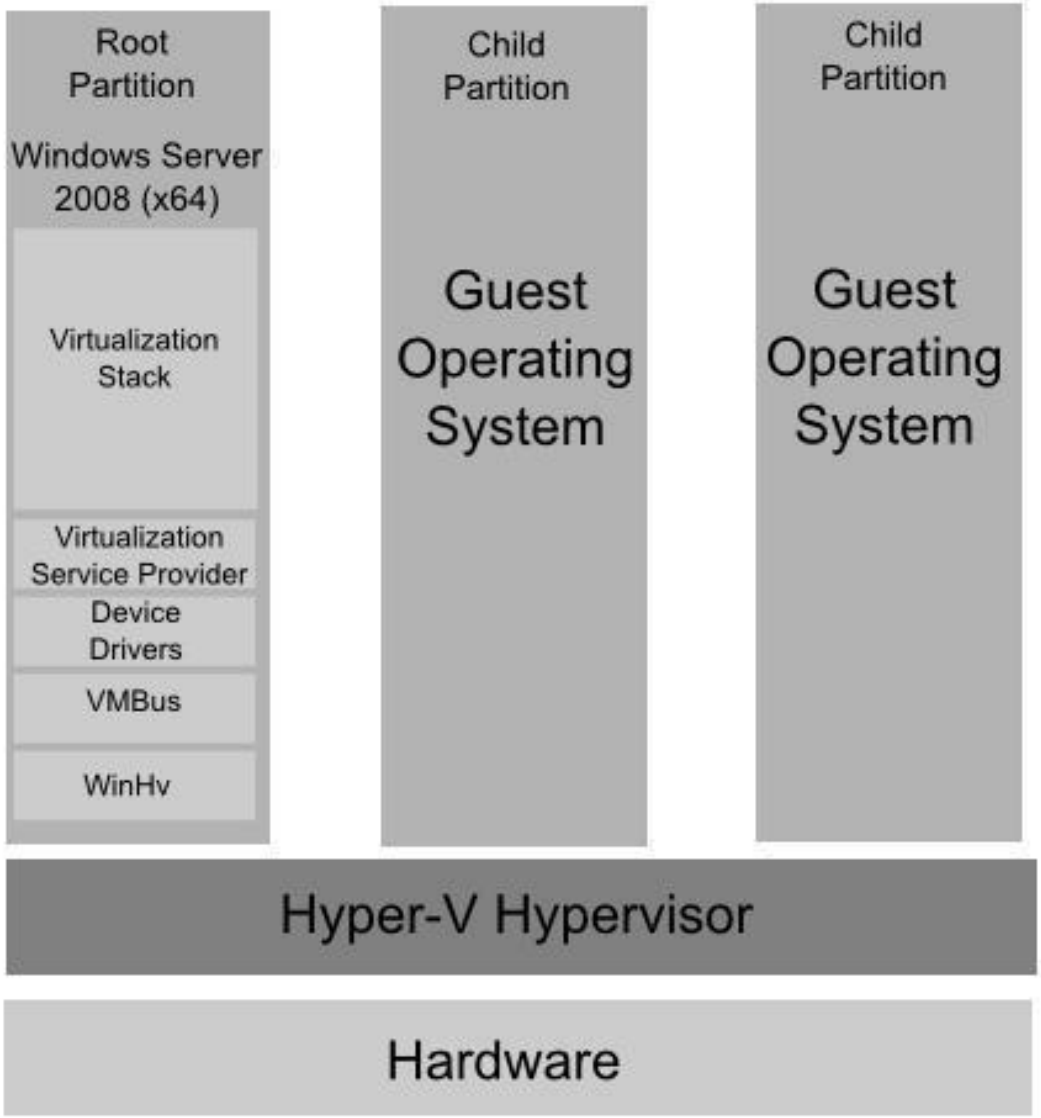
Hypervisor

- One solution to this problem is to modify the guest operating systems, replacing any privileged operations that will only run in ring 0 of the CPU with calls to the hypervisor (known as hypercalls).
- The hypervisor in turn performs the task on behalf of the guest system.

Hypervisor

- Another solution is to leverage the hardware assisted virtualization features of the latest generation of processors from both Intel and AMD.
- These technologies, known as Intel VT and AMD-V respectively, provide extensions necessary to run unmodified guest virtual machines.
- These new processors provide an additional privilege mode (referred to as ring -1) above ring 0 in which the hypervisor can operate, essentially leaving ring 0 available for unmodified guest operating systems.

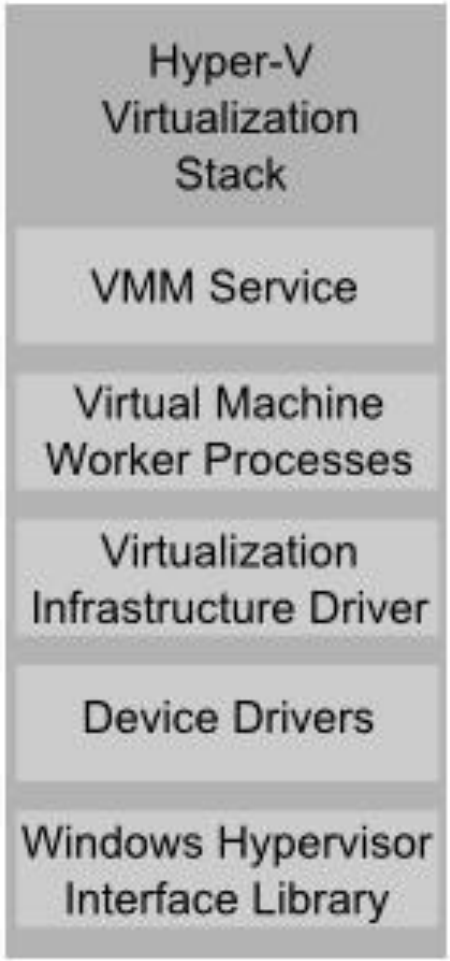
Hyper-V Root and Child Partitions



Hyper-V Root and Child Partitions

- The root partition is essentially a virtual machine which runs a copy of 64-bit Windows Server 2008 which, in turn, acts as a host for a number of special Hyper-V components.
- The root partition is responsible for providing the device drivers for the virtual machines running in the child partitions, managing the child partition lifecycles, power management and event logging.
- The root partition operating system also hosts the Virtualization Stack which is responsible for performing a wide range of virtualization functions.
- Child partitions host the virtual machines in which the guest operating systems run. Hyper-V supports both Hyper-V Aware (also referred to as enlightened) and Hyper-V Unaware guest operating systems.

The Virtualization Stack and Other Root Partition Components



The Virtualization Stack and Other Root Partition Components

Component	Description
Virtual Machine Management Service (VMM Service)	Manages the state of virtual machines running in the child partitions (active, offline, stopped etc) and controls the tasks that can be performed on a virtual machine based on current state (such as taking snapshots). Also manages the addition and removal of devices. When a virtual machine is started, the VMM Service is also responsible for creating a corresponding *Virtual Machine Worker Process*.
Virtual Machine Worker Process	Virtual Machine Worker Processes are started by the VMM Service when virtual machines are started. A Virtual Machine Worker Process (named vmwp.exe) is created for each Hyper-V virtual machine and is responsible for much of the management level interaction between the parent partition Windows Server 2008 system and the virtual machines in the child partitions. The duties of the Virtual Machine Worker Process include creating, configuring, running, pausing, resuming, saving, restoring and snapshotting the associated virtual machine. It also handles IRQs, memory and I/O port mapping through a *Virtual Motherboard* (VMB).

The Virtualization Stack and Other Root Partition Components

Component	Description
Virtual Devices	Virtual Devices are managed by the Virtual Motherboard (VMB). Virtual Motherboards are contained within the Virtual Machine Worker Processes, of which there is one for each virtual machine. Virtual Devices fall into two categories, *Core VDevs* and *Plug-in VDevs*. Core VDevs can either be *Emulated Devices* or *Synthetic Devices*.
Virtual Infrastructure Driver	Operates in kernel mode (i.e. in the privileged CPU ring) and provides partition, memory and processor management for the virtual machines running in the child partitions. The Virtual Infrastructure Driver (Vid.sys) also provides the conduit for the components higher up the Virtualization Stack to communicate with the hypervisor.

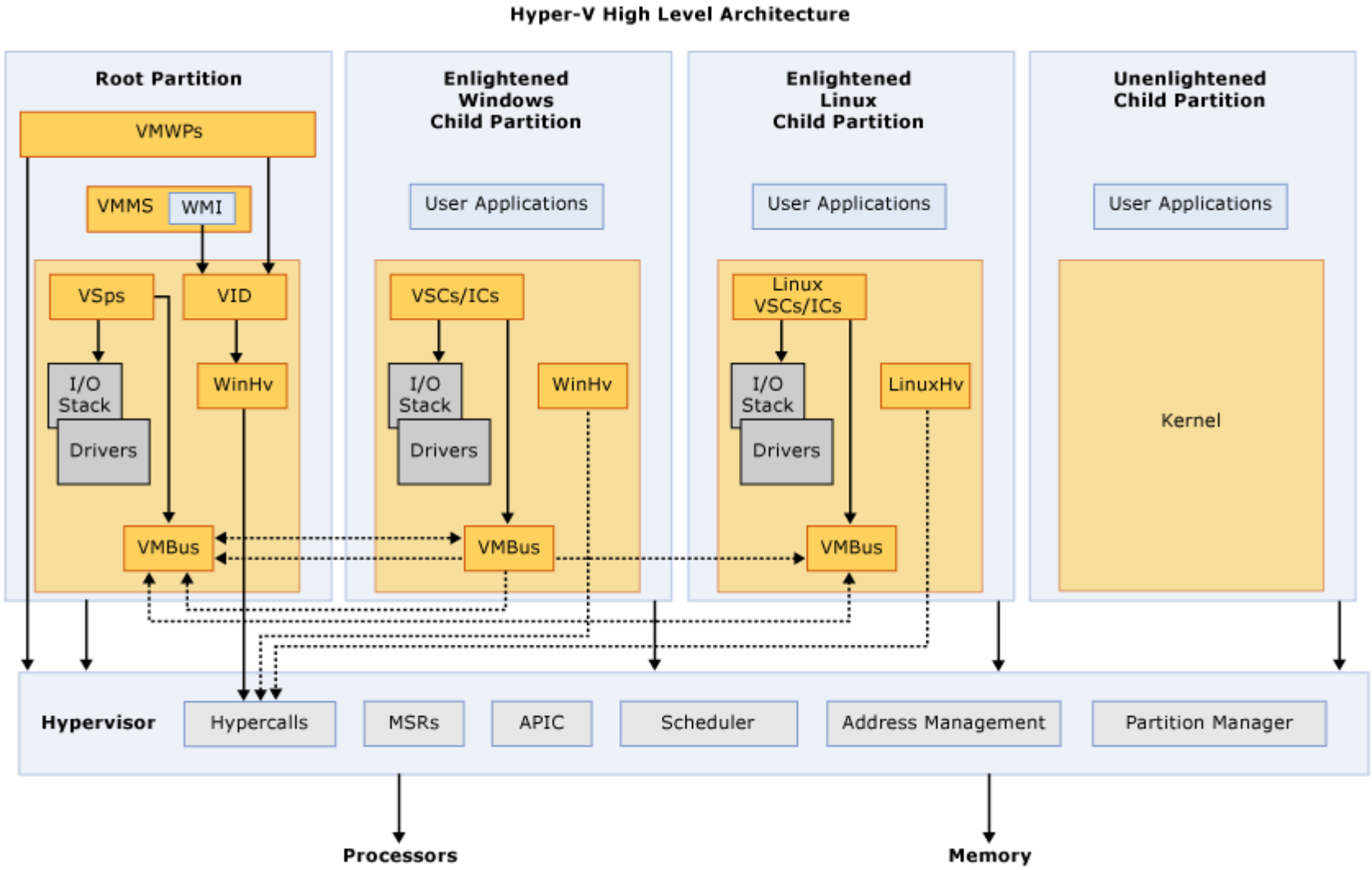
The Virtualization Stack and Other Root Partition Components

Component	Description
Windows Hypervisor Interface Library	A DLL (named WinHv.sys) located in the parent partition Windows Server 2008 instance and any guest operating systems which are *Hyper-V aware* (in other words modified specifically to operate in a Hyper-V child partition). Allows the operating system's drivers to access the hypervisor using standard Windows API calls instead of hypercalls.
VMBus	Part of Hyper-V Integration Services, the VMBus facilitates highly optimized communication between child partitions and the parent partition

The Virtualization Stack and Other Root Partition Components

Component	Description
Virtualization Service Providers	Resides in the parent partition and provides synthetic device support via the VMBus to Virtual Service Clients (VSCs) running in child partitions.
Virtualization Service Clients	Virtualization Service Clients are synthetic device instances that reside in child partitions. They communicate with the VSPs in the parent partition over the VMBus to fulfill the child partition's device access requests.

Hyper-V Architecture



Hyper-V Architecture

- APIC – Advanced Programmable Interrupt Controller – A device which allows priority levels to be assigned to its interrupt outputs.
- Child Partition – Partition that hosts a guest operating system - All access to physical memory and devices by a child partition is provided via the Virtual Machine Bus (VMBus) or the hypervisor.
- Hypercall – Interface for communication with the hypervisor - The hypercall interface accommodates access to the optimizations provided by the hypervisor.

Hyper-V Architecture

- Hypervisor – A layer of software that sits between the hardware and one or more operating systems. Its primary job is to provide isolated execution environments called partitions. The hypervisor controls and arbitrates access to the underlying hardware.
- IC – Integration component – Component that allows child partitions to communication with other partitions and the hypervisor.
- I/O stack – Input/output stack
- MSR – Memory Service Routine

Hyper-V Architecture

- Root Partition – Sometimes called parent partition. Manages machine-level functions such as device drivers, power management, and device hot addition/removal. The root (or parent) partition is the only partition that has direct access to physical memory and devices.
- VID – Virtualization Infrastructure Driver – Provides partition management services, virtual processor management services, and memory management services for partitions.

Hyper-V Architecture

- VMBus – Channel-based communication mechanism used for inter-partition communication and device enumeration on systems with multiple active virtualized partitions. The VMBus is installed with Hyper-V Integration Services.
- VMMS – Virtual Machine Management Service – Responsible for managing the state of all virtual machines in child partitions.
- VMWP – Virtual Machine Worker Process – A user mode component of the virtualization stack.
- The worker process provides virtual machine management services from the Windows Server 2008 instance in the parent partition to the guest operating systems in the child partitions.
- The Virtual Machine Management Service spawns a separate worker process for each running virtual machine.

Hyper-V Architecture

- VSC – Virtualization Service Client – A synthetic device instance that resides in a child partition.
- VSCs utilize hardware resources that are provided by Virtualization Service Providers (VSPs) in the parent partition.
- They communicate with the corresponding VSPs in the parent partition over the VMBus to satisfy a child partitions device I/O requests.
- VSP – Virtualization Service Provider – Resides in the root partition and provide synthetic device support to child partitions over the Virtual Machine Bus (VMBus).

Hyper-V Architecture

- WinHv – Windows Hypervisor Interface Library - WinHv is essentially a bridge between a partitioned operating system's drivers and the hypervisor which allows drivers to call the hypervisor using standard Windows calling conventions
- WMI – The Virtual Machine Management Service exposes a set of Windows Management Instrumentation (WMI)-based APIs for managing and controlling virtual machines.

hyper-v containers

- Windows Hyper-v container is a windows server container that runs in a VM.
- Every hyper-v container creates its own VM.
- This means that there is no kernel sharing between the different hyper-v containers.
- This is useful for cases where additional level of isolation is needed by customers who don’t like the traditional kernel sharing done by containers.
- The same Docker image and CLI can be used to manage hyper-v containers.
- Creation of hyper-v containers is specified as a runtime option.
- There is no difference when building or managing containers between windows server and hyper-v container.
- Startup times for hyper-v container is higher than windows native container since a new lightweight VM gets created each time.

hyper-v containers

- Creation of hyper-v containers is specified as a runtime option.
- There is no difference when building or managing containers between windows server and hyper-v container.
- Startup times for hyper-v container is higher than windows native container since a new lightweight VM gets created each time.

hyper-v containers

There are 2 modes of hyper-v container.

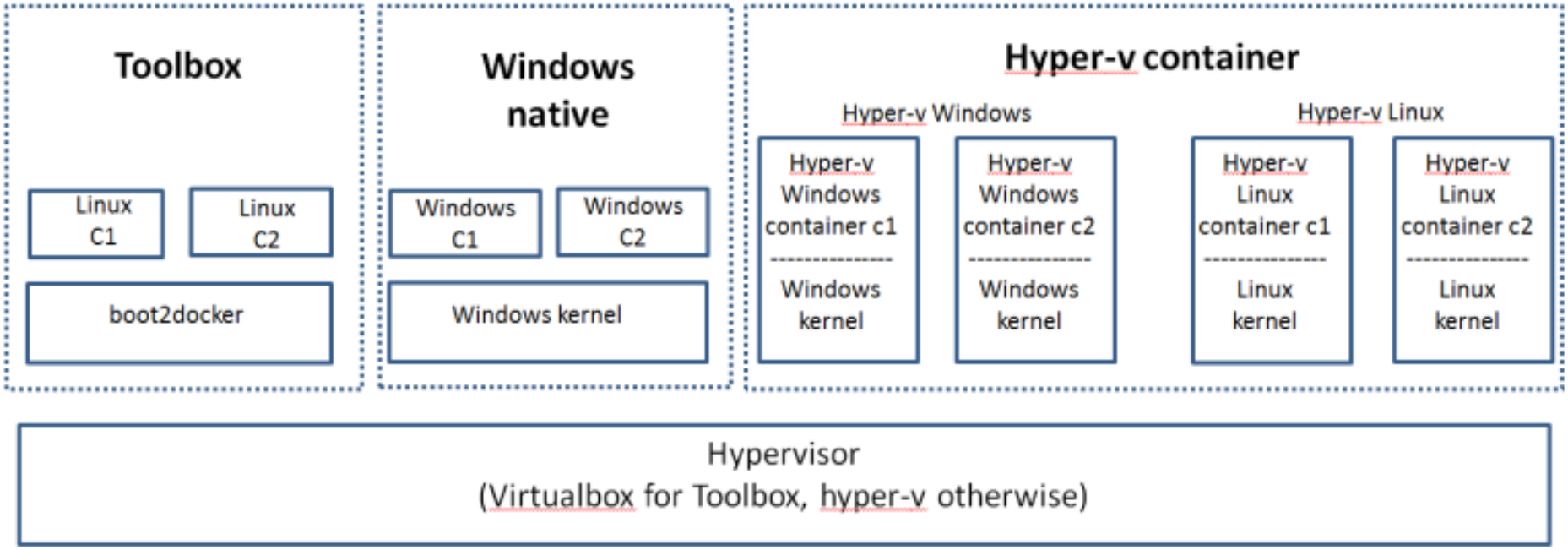
1. Windows hyper-v container – Here, hyper-v container runs on top of Windows kernel. Only Windows containers can be run in this mode.
2. Linux hyper-v container – Here, hyper-v container runs on top of Linux kernel. This mode was not available earlier and it was introduced as part of Dockercon 2017. Any Linux flavor can be used as the base kernel. Docker's Linuxkit project can be used to build the Linux kernel needed for the hyper-v container. Only Linux containers can be run in this mode.

hyper-v containers

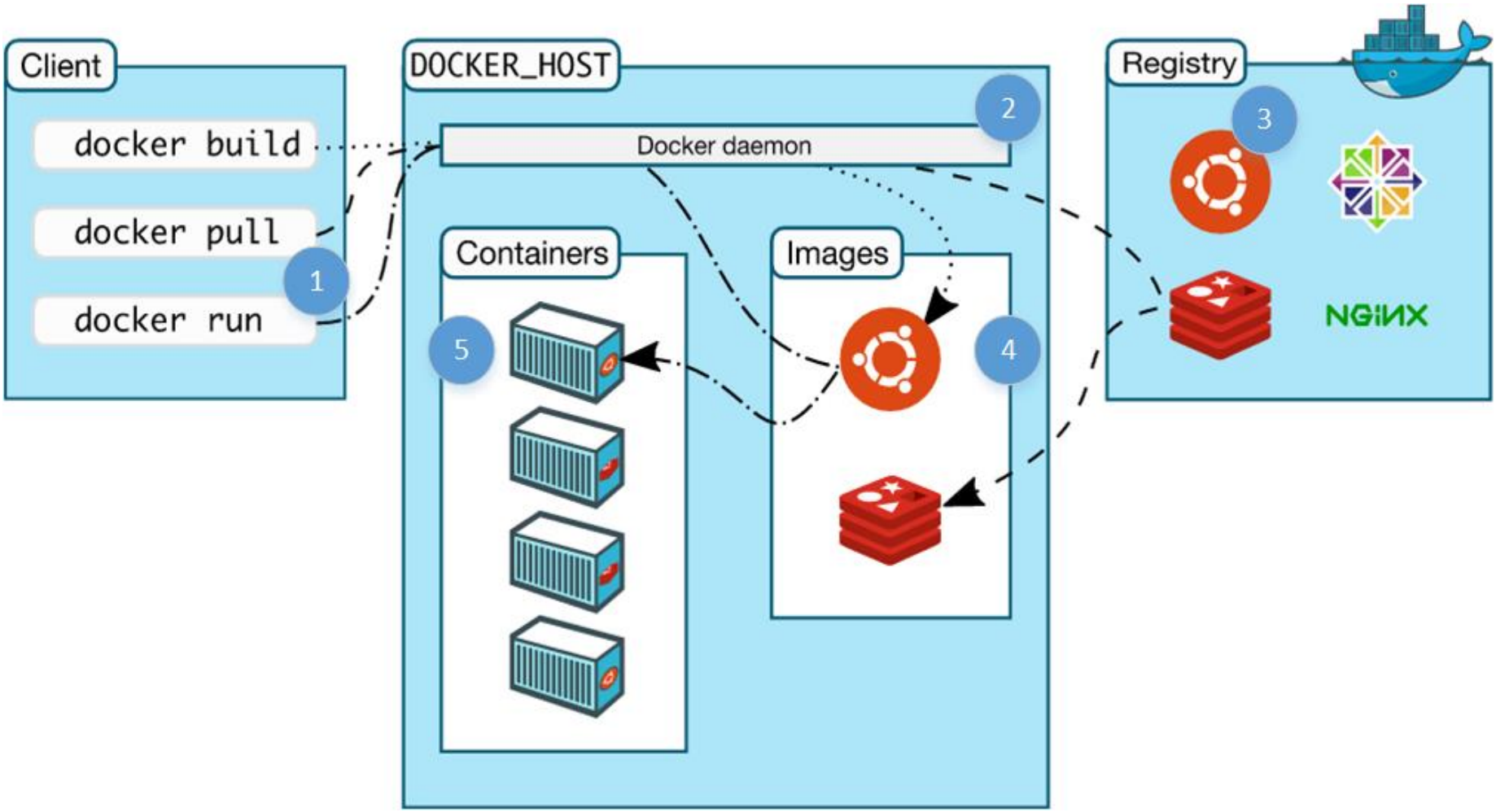
1. We cannot use Docker Toolbox and hyper-v containers at the same time. Virtualbox cannot run when “Docker for Windows” is installed.

2. https://nickjanetakis.com/blog/should-you-use-the-docker-toolbox-or-docker-for-mac-windows

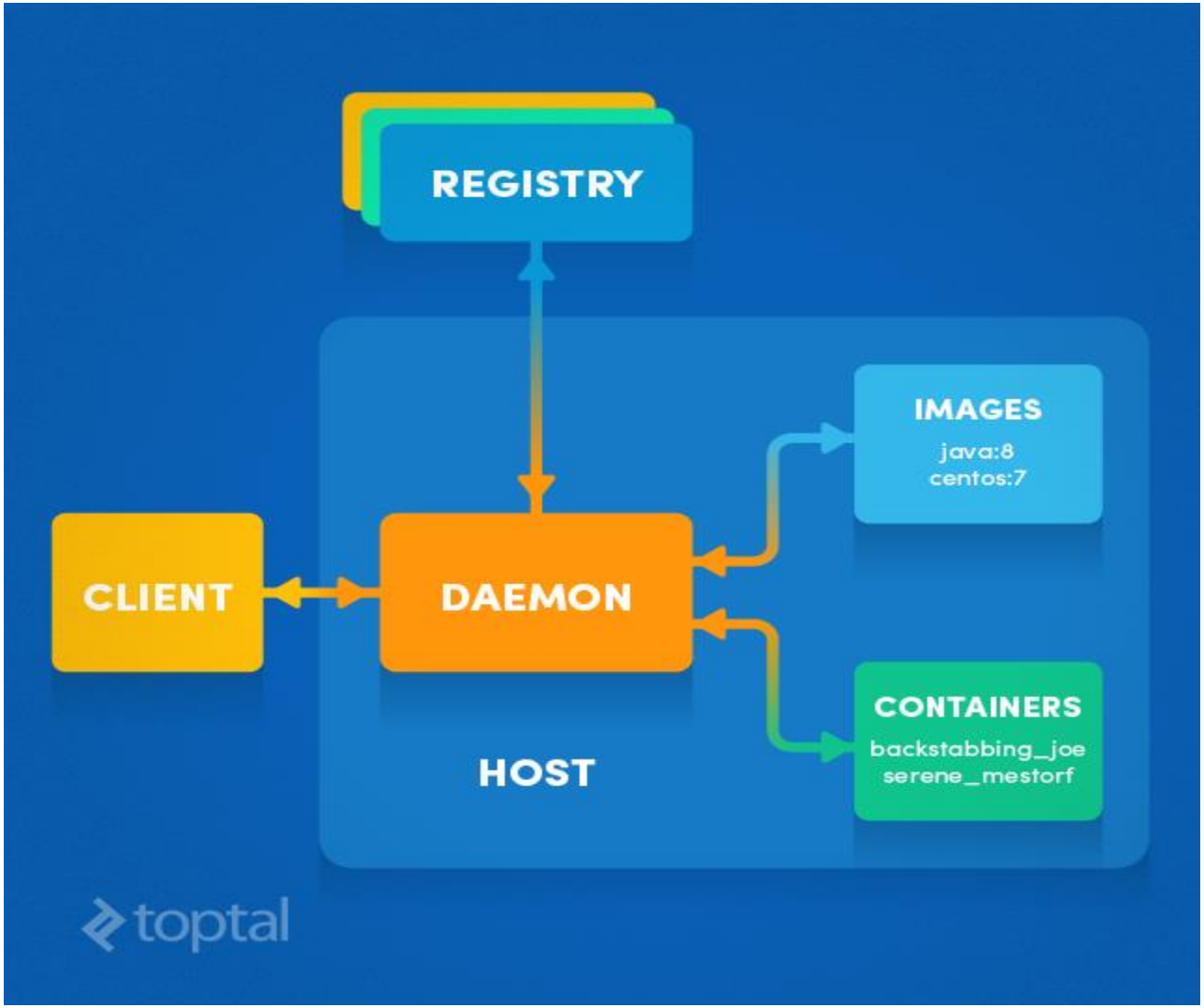
Docker


Toolbox
Linux C1
Linux C2
boot2docker
Windows native
Windows C1
Windows C2
Windows kernel
Hyper-v container
Hyper-v Windows
Hyper-v Windows container c1
Windows kernel
Hyper-v Windows container c2
Windows kernel
Hyper-v Linux
Hyper-v Linux container c1
Linux kernel
Hyper-v Linux container c2
Linux kernel
Hypervisor
(Virtualbox for Toolbox, hyper-v otherwise)

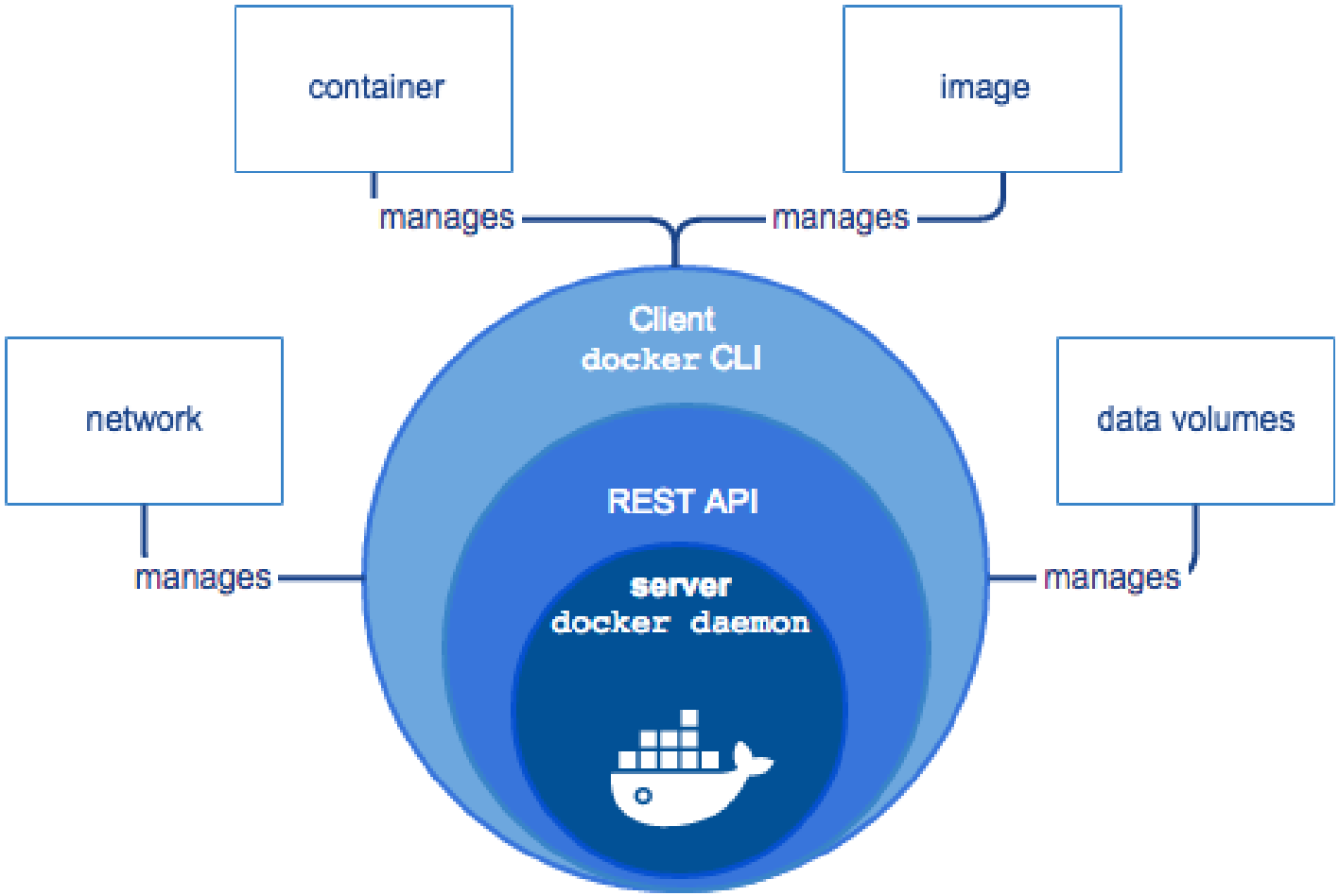
Docker Architecture



Docker Architecture



Docker Engine



Dockerfile, Docker Image And Docker Container

- A Docker Image is created by the sequence of commands written in a file called as Dockerfile.
- When this Dockerfile is executed using a docker command it results into a Docker Image with a name.
- When this Image is executed by “docker run” command it will by itself start whatever application or service it must start on its execution.

Image Demo

- https://learnk8s.io/spring-boot-kubernetes-guide


Build
Run
Docker File
Docker Image
Docker Container

Docker Network

- Docker implements networking in an application-driven manner and provides various options while maintaining enough abstraction for application developers.
- There are basically two types of networks available - the default Docker network and user-defined networks.
- By default, you get three different networks on the installation of Docker - none, bridge, and host.

Docker Network

- The none and host networks are part of the network stack in Docker.
- The bridge network automatically creates a gateway and IP subnet and all containers that belong to this network can talk to each other via IP addressing.
- This network is not commonly used as it does not scale well and has constraints in terms of network usability and service discovery.

Docker Network

- The other type of networks is user-defined networks. Administrators can configure multiple user-defined networks.
- There are three types:
- Bridge network: Similar to the default bridge network, a user-defined Bridge network differs in that there is no need for port forwarding for containers within the network to communicate with each other.
- The other difference is that it has full support for automatic network discovery.

Docker Network

- Overlay network: An Overlay network is used when you need containers on separate hosts to be able to communicate with each other, as in the case of a distributed network.
- However, a caveat is that swarm mode must be enabled for a cluster of Docker engines, known as a swarm, to be able to join the same group.
- Macvlan network: When using Bridge and Overlay networks a bridge resides between the container and the host.
- A Macvlan network removes this bridge, providing the benefit of exposing container resources to external networks without dealing with port forwarding.
- This is realized by using MAC addresses instead of IP addresses.

Storage

- You can store data within the writable layer of a container but it requires a storage driver.
- Being non-persistent, it perishes whenever the container is not running.
- It is not easy to transfer this data.
- In terms of persistent storage, Docker offers four options:
- Data Volume, Data Volume Container, Directory Mounts, Storage Plugins

Storage

- There are storage plugins from various companies to automate the storage provisioning process. For example,
- HPE 3PAR
- EMC (ScaleIO, XtremIO, VMAX, Isilon)
- NetApp
- There are also plugins that support public cloud providers like:
- Azure File Storage
- Google Compute Platform.

Registries

- A Docker registry is a storage and distribution system for named Docker images.
- The same image might have multiple different versions, identified by their tags.
- A Docker registry is organized into Docker repositories , where a repository holds all the versions of a specific image.
- The registry allows Docker users to pull images locally, as well as push new images to the registry (given adequate access permissions when applicable).

Registries

- By default, the Docker engine interacts with DockerHub , Docker's public registry instance.
- However, it is possible to run on-premise the open-source Docker registry/distribution, as well as a commercially supported version called Docker Trusted Registry .

Docker Hub

- DockerHub is a hosted registry solution by Docker Inc.
- Besides public and private repositories, it also provides automated builds, organization accounts, and integration with source control solutions like Github and Bitbucket .

Docker Hub

- A public repository is accessible by anyone running Docker, and image names include the organization/user name.
- For example, docker pull jenkins/jenkins will pull the Jenkins CI server image with tag latest from the Jenkins organization.
- There are hundreds of thousands public images available.
- Private repositories restrict access to the repository creator or members of its organization.

Docker Hub

- DockerHub supports official repositories, which include images verified for security and best practices.
- These do not require an organization/user name, for example docker pull nginx will pull the latest image of the Nginx load balancer.
- DockerHub can perform automated image builds if the DockerHub repository is linked to a source control repository which contains a build context (Dockerfile and all any files in the same folder).
- A commit in the source repository will trigger a build in DockerHub.

Other Public Registries

- Other companies host paid online Docker registries for public use.
- Cloud providers like AWS and Google, who also offer container-hosting services, market the high availability of their registries.

Other Public Registries

- Amazon Elastic Container Registry (ECR) integrates with AWS Identity and Access Management (IAM) service for authentication. It supports only private repositories and does not provide automated image building.
- Google Container Registry (GCR) authentication is based on Google's Cloud Storage service permissions. It supports only private repositories and provides automated image builds via integration with Google Cloud Source Repositories, GitHub, and Bitbucket.

Other Public Registries

- Azure Container Registry (ACR) supports multi-region registries and authenticates with Active Directory. It supports only private repositories and does not provide automated image building.
- CoreOS Quay supports OAuth and LDAP authentication. It offers both (paid) private and (free) public repositories, automatic security scanning and automated image builds via integration with GitLab, GitHub, and Bitbucket.
- Private Docker Registry supports OAuth, LDAP and Active Directory authentication. It offers both private and public repositories, free up to 3 repositories (private or public).

Private Registries

- Use cases for running a private registry on-premise (internal to the organization) include:
- Distributing images inside an isolated network (not sending images over the Internet)
- Creating faster CI/CD pipelines (pulling and pushing images from internal network), including faster deployments to on-premise environments
- Deploying a new image over a large cluster of machines
- Tightly controlling where images are being stored

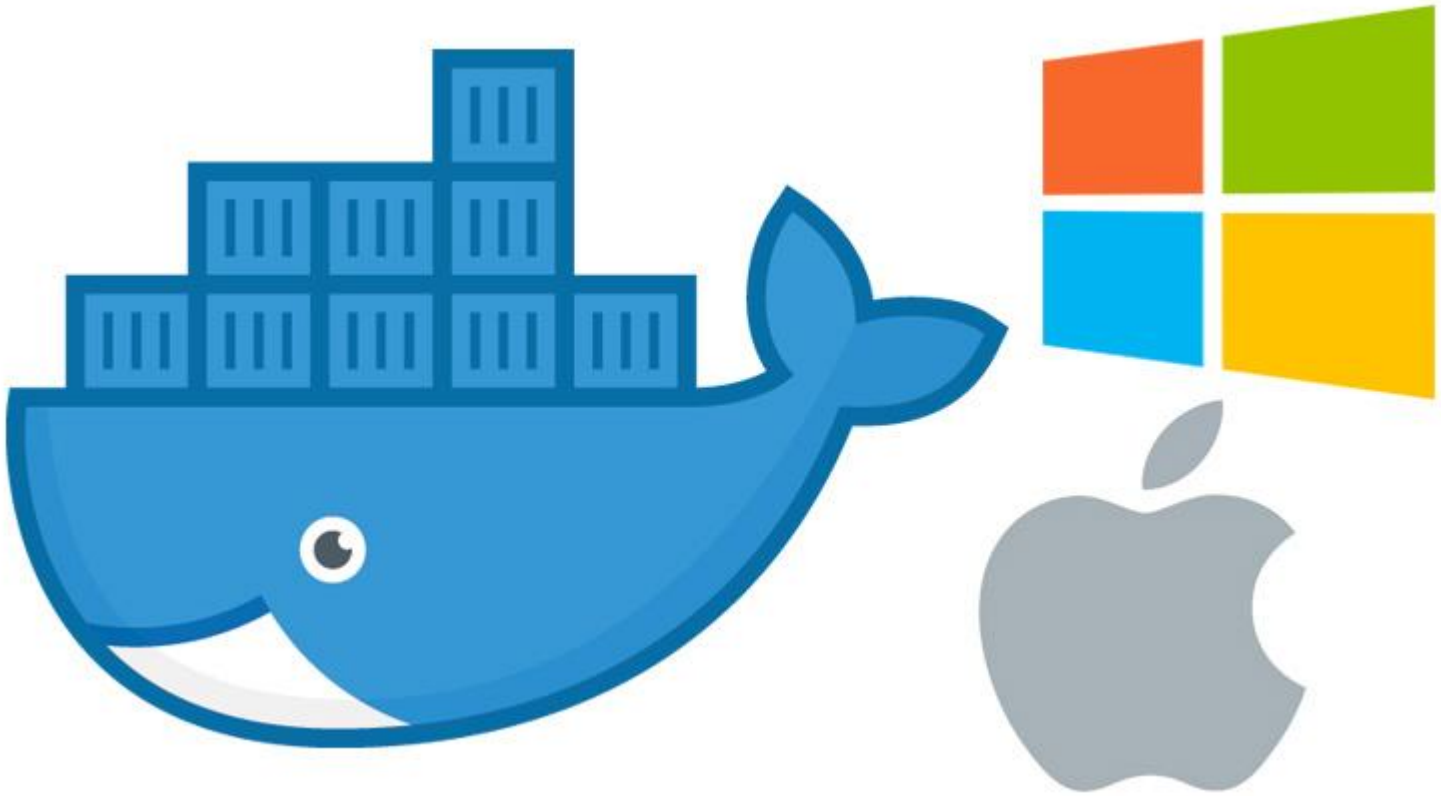
Private Registries

- Running a private registry system, especially when delivery to production depends on it, requires operational skills such as ensuring availability, logging and log processing, monitoring, and security.
- Strong understanding of http and overall network communications is also important.

Private Registries

- Docker Trusted Registry is Docker Inc's commercially supported version, providing high availability via replication, image auditing, signing and security scanning, integration with LDAP and Active Directory.
- Harbor is a VMWare open source offering which also provides high availability via replication, image auditing, integration with LDAP and Active Directory.
- GitLab Container Registry is tightly integrated with GitLab CI's workflow, with minimal setup.
- JFrog Artifactory for strong artifact management (not only Docker images but any artifact).

Should You Install Docker with the Docker Toolbox or Docker for Mac / Windows?



Should You Install Docker with the Docker Toolbox or Docker for Mac / Windows?

- If you're on MacOS or Windows you can install Docker with:
  - Docker for Mac / Windows (now known as Docker Desktop)
  - Docker Toolbox
  - Running your own Virtual Machine and installing Docker yourself

Should You Install Docker with the Docker Toolbox or Docker for Mac / Windows?

- If you're on MacOS or Windows you can install Docker with:
  - Docker for Mac / Windows (now known as Docker Desktop)
  - Docker Toolbox
  - Running your own Virtual Machine and installing Docker yourself

Docker for Mac / Docker for Windows (Docker Desktop)

- Pros
- Offers the most “native” experience, you can easily use any terminal you you want since Docker is effectively running on localhost from MacOS / Windows’ POV.
- Docker is heavily developing and polishing this solution.

Docker for Mac / Docker for Windows (Docker Desktop)

- **Cons**
- On certain MacOS hardware combos the volume performance can be a little slow.
- I can legit say there are not any “wow this sucks!” cons for Windows, it’s really solid.

Docker Toolbox

- Pros
- Offers an “out of the box” Docker experience if you have no other choice.

Docker Toolbox

- Cons
- You need to either use the Docker Quickstart Terminal, or configure your own terminal to connect to the Docker Daemon running a VM.

- Not a native solution, so you'll need to access your Docker Machine's IP address if you're developing web apps. Example: 192.168.99.100 instead of localhost.

Docker Toolbox

- Cons
- Unless you jump through hoops, your code needs to live in your Windows user directory such as C:\Users\eswari\src\myapp. Otherwise Docker won't be able to find it.

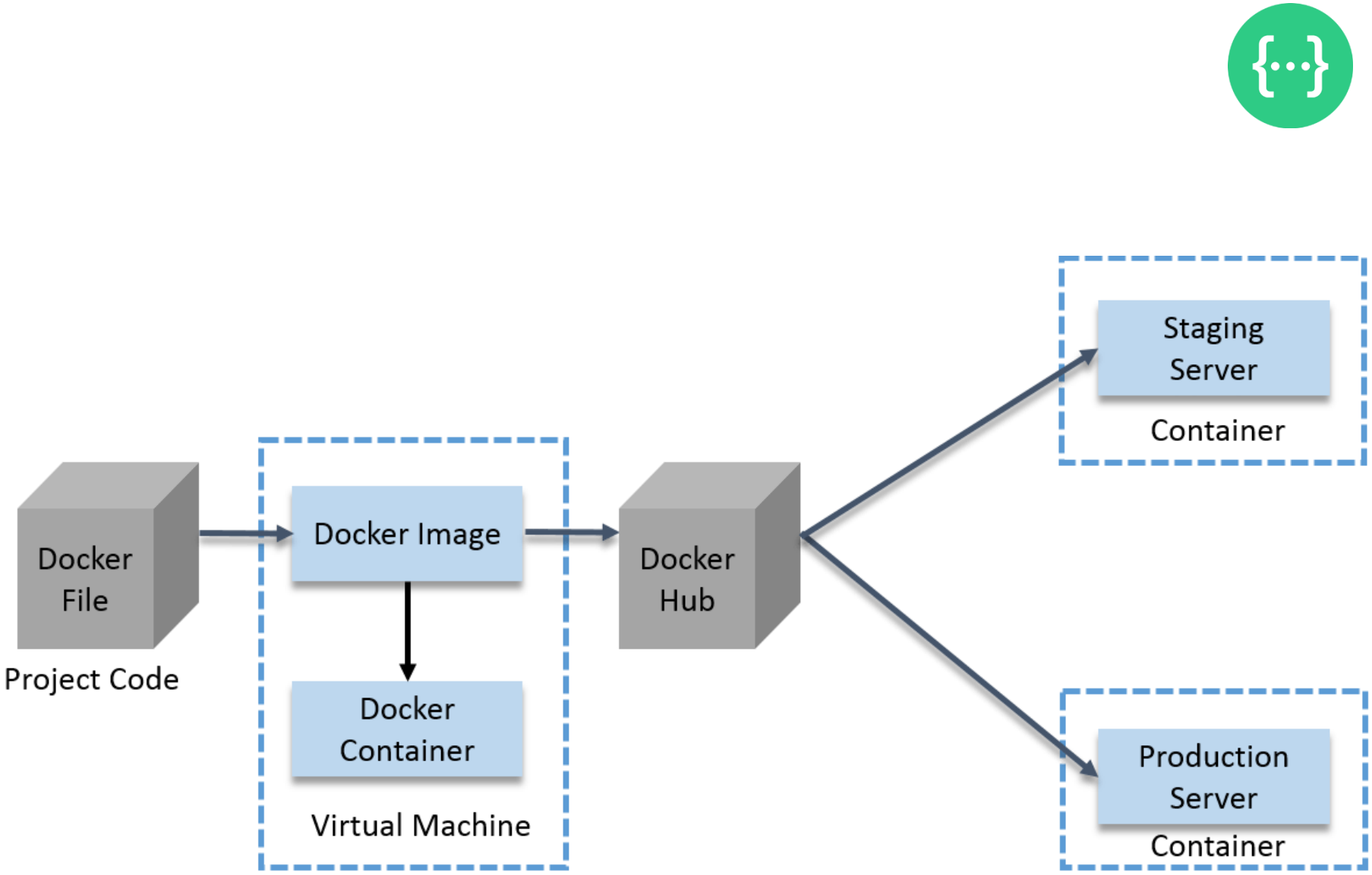
- Suffers from typical VirtualBox edge case bugs and mount performance issues.

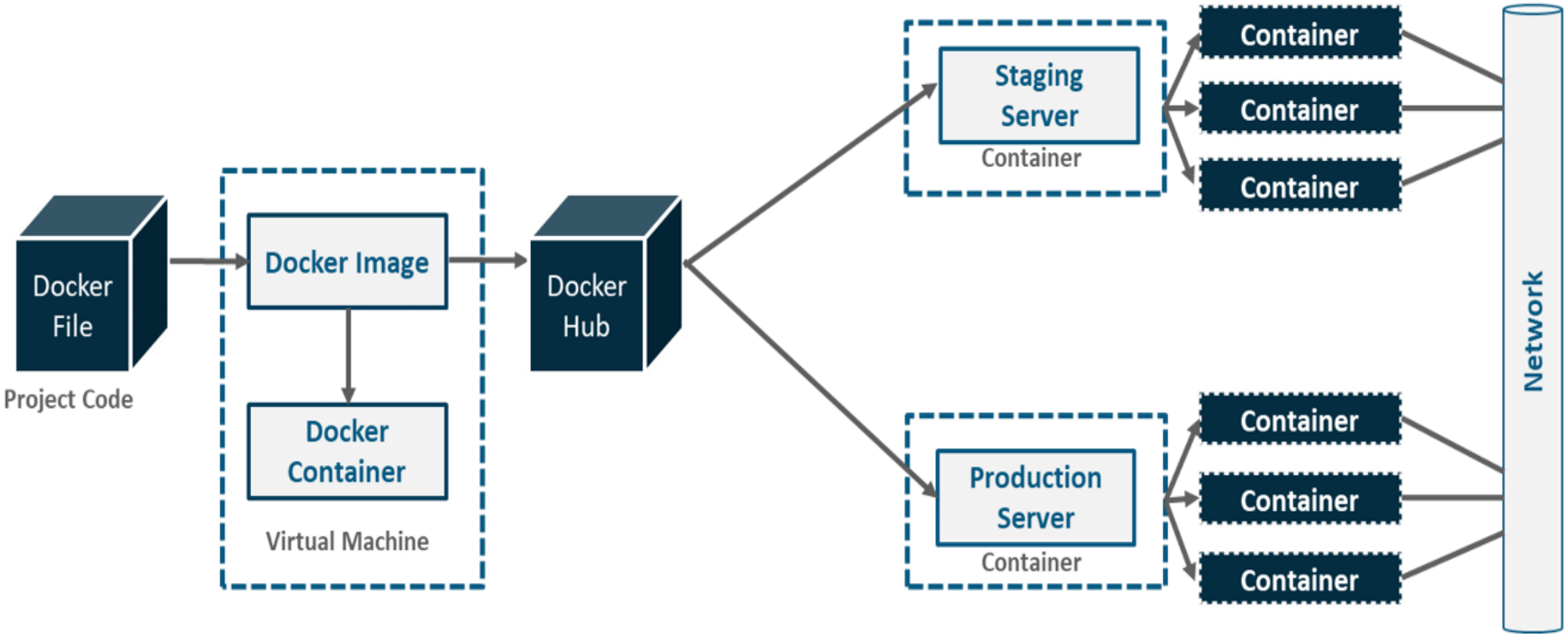
What is Docker

- What is Docker?
- At its heart, Docker is software which lets you create an *image* and then run instances of that image in a *container*.
- Docker maintain a vast repository of images, called the Docker Hub which you can use as starting points or as free storage for your own images.
- You can install Docker, choose an image you'd like to use, then run an instance of it in a container.

Difference between VM and Container

- Applications running in virtual machines, apart from the hypervisor, require a full instance of the operating system and any supporting libraries.
- Containers, on the other hand, share the operating system with the host.
- Hypervisor is comparable to the container engine (represented as Docker on the image) in a sense that it manages the lifecycle of the containers.
- The important difference is that the processes running inside the containers are just like the native processes on the host, and do not introduce any overheads associated with hypervisor execution.
- Additionally, applications can reuse the libraries and share the data between containers.

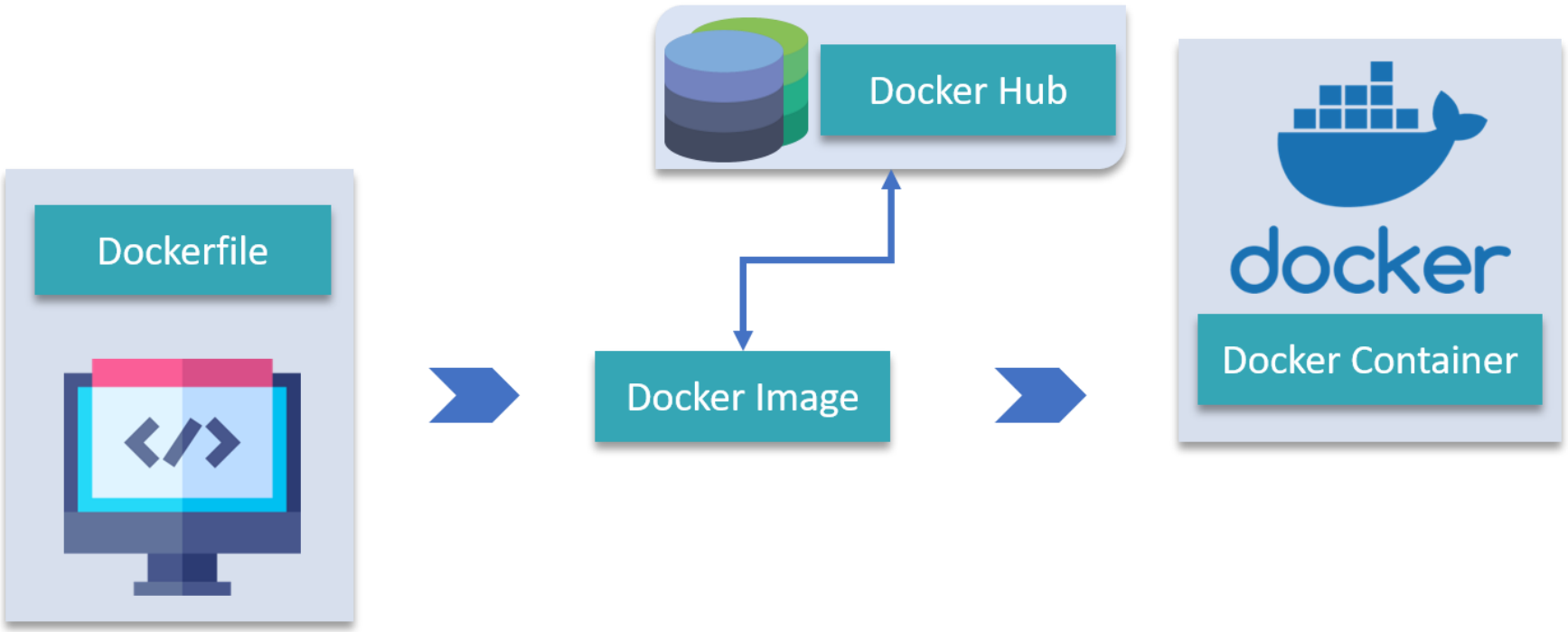

Docker
File
Project Code
Docker Image
Docker
Container
Virtual Machine
Docker
Hub
Staging
Server
Container
Production
Server
Container


Docker File
Project Code
Docker Image
Docker Container
Virtual Machine
Docker Hub
Staging Server
Container
Production Server
Container
Container
Container
Container
Container
Container
Network

Docker Hub:

- Docker Hub is like GitHub for Docker Images. It is basically a cloud registry where you can find Docker Images uploaded by different communities, also you can develop your own image and upload on Docker Hub, but first, you need to create an account on DockerHub.

Docker Hub


Dockerfile
Docker Hub
Docker Image
docker
Docker Container

Docker Architecture

- It consists of a Docker Engine which is a client-server application with three major components:
- A server which is a type of long-running program called a daemon process (the docker command).
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the docker command).
- The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI.

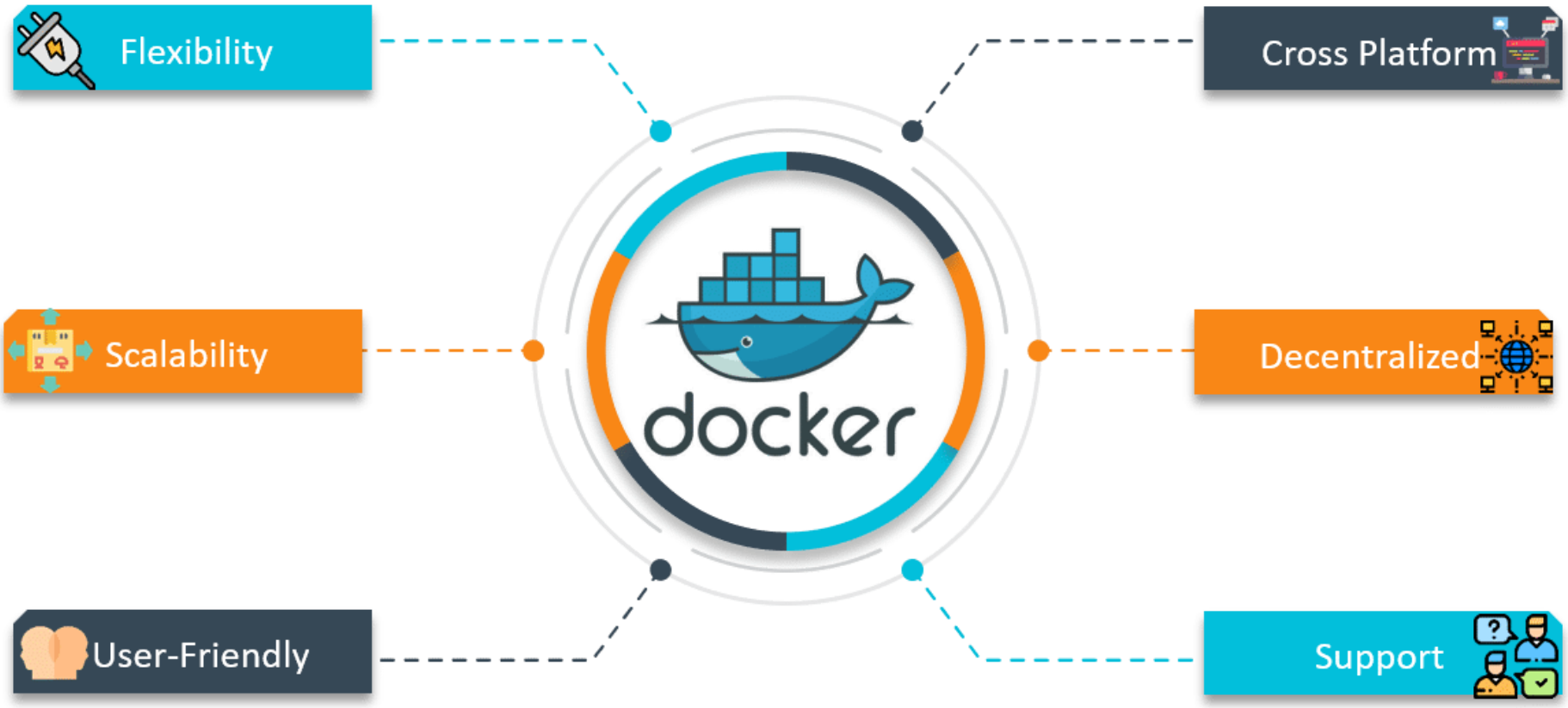
Docker Architecture

- By default, the main registry is the Docker Hub which hosts public and official images.
- Organizations can also host their private registries if they desire.
- Images can be downloaded from registries explicitly (docker pull imageName) or implicitly when starting a container. Once the image is downloaded it is cached locally.
- Containers are the instances of images - they are the living thing. There could be multiple containers running based on the same image.

Docker Architecture

- At the center, there is the Docker daemon responsible for creating, running, and monitoring containers.
- It also takes care of building and storing images.
- Finally, on the left-hand side there is a Docker client. It talks to the daemon via HTTP.
- Unix sockets are used when on the same machine, but remote management is possible via HTTP based API.

Goals of Docker Networking



Creating Test Database Server

- This is a great Docker use case.
- We might not want to run our production database in Docker (perhaps we'll just use Amazon RDS for example), but we can spin up a clean MySQL database in no time as a Docker container for development - leaving our development machine clean and keeping everything we do controlled and repeatable.

Docker Compose

- Docker Compose is basically used to run multiple Docker Containers as a single server. Let me give you an example:
- Suppose if I have an application which requires WordPress, Maria DB and PHP MyAdmin. I can create one file which would start both the containers as a service without the need to start each one separately. It is really useful especially if you have a microservice architecture.

Docker Container

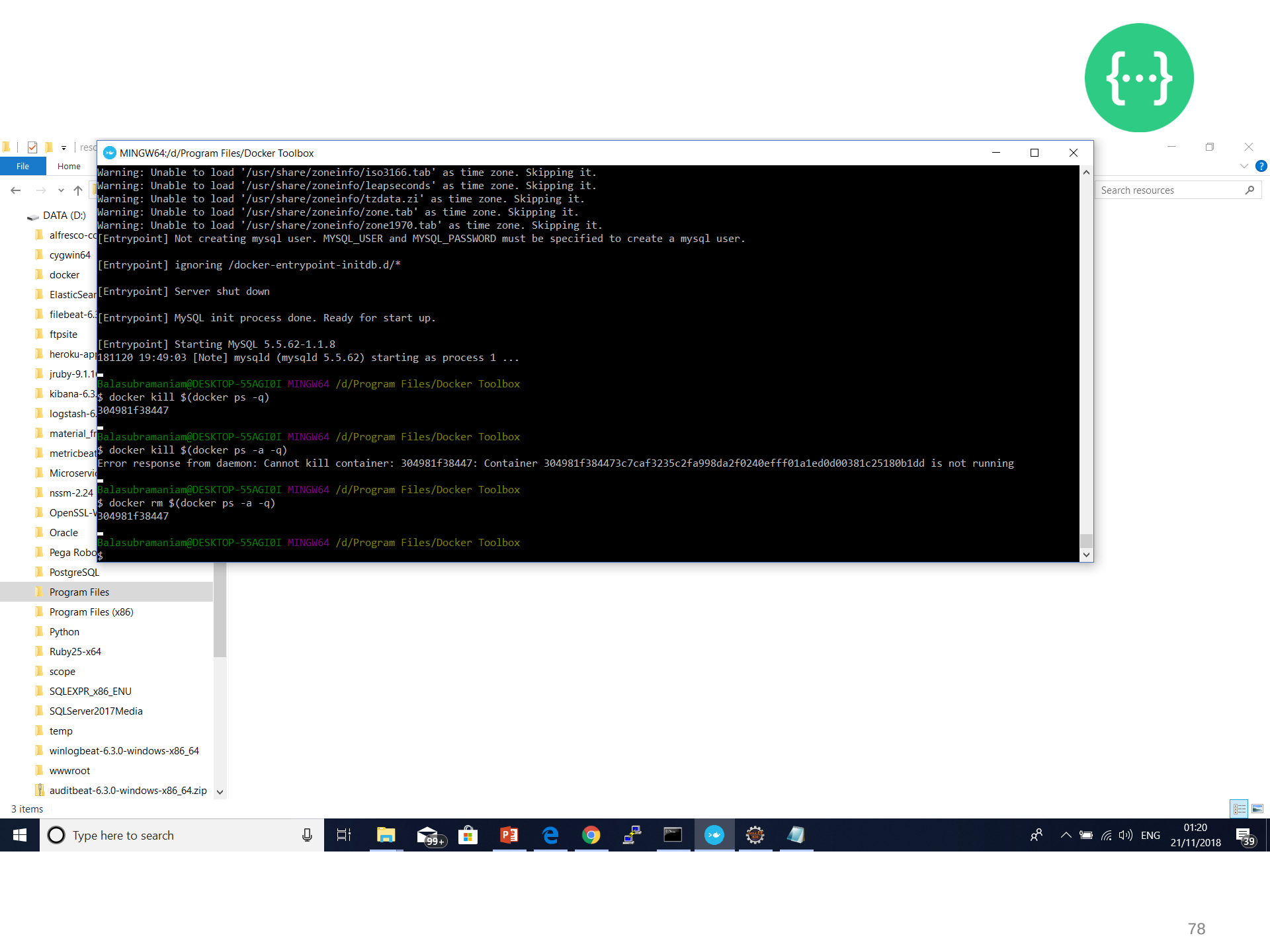
Column	Description
Container ID	The unique ID of the container. It is a SHA-256.
Image	The name of the container image from which this container is instantiated.
Status	The status of the container (created, restarting, running, removing, paused, exited, or dead).
Ports	The list of container ports that have been mapped to the host.
Names	The name assigned to this container (multiple names are possible).

Docker Client and Docker Engine

- **Docker Client** : This is the utility we use when we run any docker commands e.g. docker run (docker container run) , docker images , docker ps etc. It allows us to run these commands which a human can easily understand.
- **Docker Daemon/Engine**: This is the part which does rest of the magic and knows how to talk to the kernel, makes the system calls to create, operate and manage containers, which we as users of docker dont have to worry about.

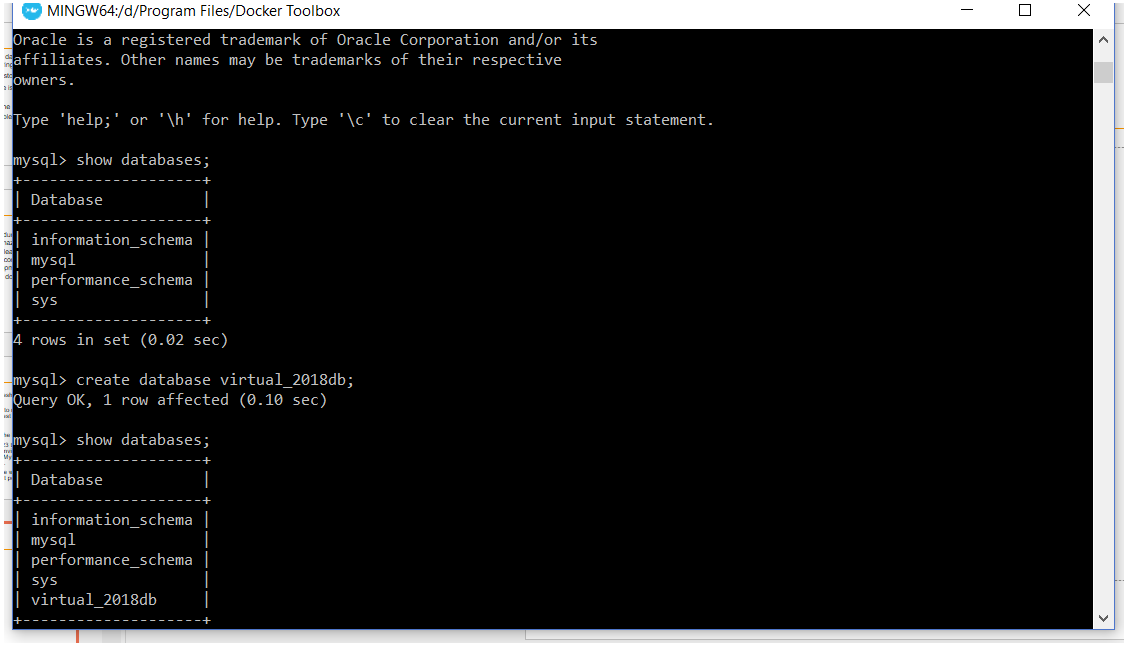
Creating Test Database Server

- docker run --name olddb -e MYSQL_ROOT_PASSWORD=vignesh –e MYSQL_DATABASE=virtusa_2018db –e MYSQL_USER=root -p 3306:3306 mysql/mysql-server:5.5
- docker run tells the engine we want to run an image (the image comes at the end, mysql:vlatest
- --name db names this container db.
- -d detach - i.e. run the container in the background.
- -e MYSQL_ROOT_PASSWORD=123 the -e is the flag that tells docker we want to provide an environment variable. The variable following it is what the MySQL image checks for setting the default root password.
- -p 3306:3306 tells the engine that we want to map the port 3306 from inside the container to out port 3306.


MINGW64:/d/Program Files/Docker Toolbox
Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/leapseconds' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/tzdata.zi' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone.tab' as time zone. Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as time zone. Skipping it.
[Entrypoint] Not creating mysql user. MYSQL_USER and MYSQL_PASSWORD must be specified to create a mysql user.
[Entrypoint] ignoring /docker-entrypoint-initdb.d/*
[Entrypoint] Server shut down
[Entrypoint] MySQL init process done. Ready for start up.
[Entrypoint] Starting MySQL 5.5.62-1.1.8
181120 19:49:03 [Note] mysqld (mysqld 5.5.62) starting as process 1 ...
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker kill $(docker ps -q)
304981f38447
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker kill $(docker ps -a -q)
Error response from daemon: Cannot kill container: 304981f38447: Container 304981f384473c7caf3235c2fa998da2f0240efff01a1ed0d00381c25180b1dd is not running
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker rm $(docker ps -a -q)
304981f38447
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$
DATA (D:)
Program Files
Program Files (x86)
Python
Ruby25-x64
scope
SQLEXPR_x86_ENU
SQLServer2017Media
temp
winlogbeat-6.3.0-windows-x86_64
wwwroot
auditbeat-6.3.0-windows-x86_64.zip
3 items
Search resources
Type here to search
ENG
01:20
21/11/2018

Creating Test Database Server

- Docker ps
- docker exec -it olddb /bin/bash
- mysql –u root –p



Springboot Docker

```
<plugin>
    <groupId>com.spotify</groupId>
    <artifactId>docker-maven-plugin</artifactId>
    <version>VERSION GOES HERE</version>
    <configuration>
      <imageName>example</imageName>
      <baseImage>java</baseImage>
      <entryPoint>["java", "-jar", "/${project.build.finalName}.jar"]</entryPoint>
      <!-- copy the service's jar file from target into the root directory of the image -->
      <resources>
        <resource>
          <targetPath>/</targetPath>
          <directory>${project.build.directory}</directory>
          <include>${project.build.finalName}.jar</include>
        </resource>
      </resources>
    </configuration>
  </plugin>
```

Best practices for writing Dockerfiles

- Docker builds images automatically by reading the instructions from a Dockerfile -- a text file that contains all commands, in order, needed to build a given image.
- A Dockerfile adheres to a specific format and set of instructions which you can find at Dockerfile reference.
- A Docker image consists of read-only layers each of which represents a Dockerfile instruction.
- The layers are stacked and each one is a delta of the changes from the previous layer.

Command	Purpose
FROM	To specify the parent image.
WORKDIR	To set the working directory for any commands that follow in the Dockerfile.
RUN	To install any applications and packages required for your container.
COPY	To copy over files or directories from a specific location.
ADD	As COPY, but also able to handle remote URLs and unpack compressed files.
ENTRYPOINT	Command that will always be executed when the container starts. If not specified, the default is /bin/sh -c
CMD	Arguments passed to the entrypoint. If ENTRYPOINT is not set (defaults to /bin/sh -c), the CMD will be the commands the container executes.
EXPOSE	To define which port through which to access your container application.
LABEL	To add metadata to the image.

Best practices for writing Dockerfiles

- FROM ubuntu:18.04
- COPY . /app
- RUN make /app
- CMD python /app/app.py
- FROM creates a layer from the ubuntu:18.04 Docker image.
- COPY adds files from your Docker client’s current directory.
- RUN builds your application with make.
- CMD specifies what command to run within the container.

Best practices for writing Dockerfiles

- https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Springboot Docker

- Spring boot Project Build

- Goal package docker:build
- mvn clean install dockerfile:build
- Docker folder created and docker image name example deployed in docker machine
- Check docker images
- Docker ps

Springboot Docker

- Docker run –t –name sampleapp –link v1db –p 8080:8080 example:latest

- Example:latest --- image name
- --name – container reference
- --link – db ref in container

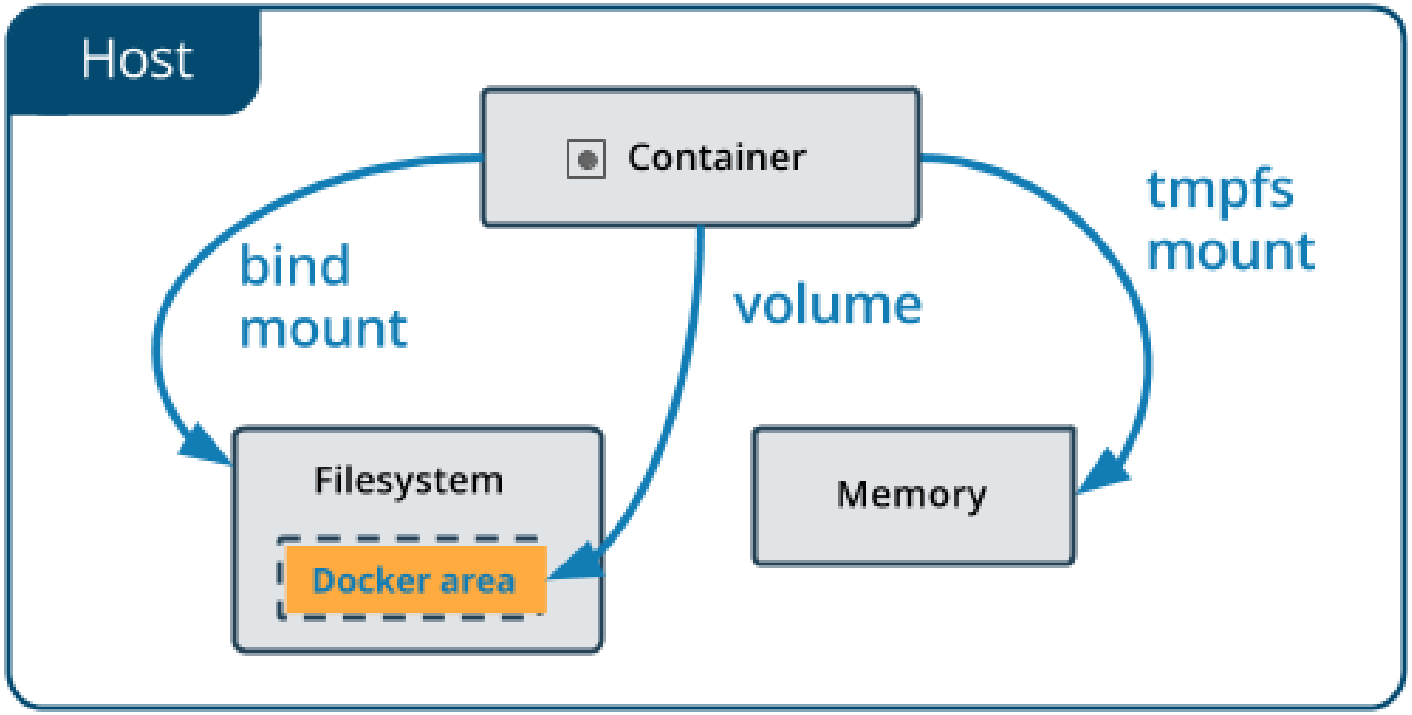
docker run -h 192.168.99.100 -p 7070:7070 -t my-repo/example --name my-repo-image:latest

Docker Volume

- Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.
- While bind mounts are dependent on the directory structure of the host machine, volumes are completely managed by Docker.

Docker Volume

- Volumes have several advantages over bind mounts:
  - Volumes are easier to back up or migrate than bind mounts.
  - You can manage volumes using Docker CLI commands or the Docker API.
  - Volumes work on both Linux and Windows containers.
  - Volumes can be more safely shared among multiple containers.
  - Volume drivers let you store volumes on remote hosts or cloud providers, to encrypt the contents of volumes, or to add other functionality.
  - New volumes can have their content pre-populated by a container


Host
Container
bind
mount
volume
tmpfs
mount
Filesystem
Docker area
Memory

Manage Data in Docker

- There are 2 ways in which you can manage data in Docker:
  - Data volumes
  - Data volume containers

Manage Data in Docker

- A data volume is a specially designed directory in the container.
- It is initialized when the container is created.
- By default, it is not deleted when the container is stopped.
- It is not even garbage collected when there is no container referencing the volume.
- The data volumes are independently updated.
- Data volumes can be shared across containers too. They could be mounted in read-only mode too.

Manage Data in Docker

- Mounting a Data volume
- docker container run -it -v/udata --tty ubuntu /bin/bash
- cd udata
- touch file1.txt
- exit
- docker container restart 2eec01eb7368
- docker attach 2eec01eb7368exit
- docker container rm 2eec01eb7368
- docker volume ls

Manage Data in Docker

- After removing container also the data volume is still present on the host.
- This is a dangling or ghost volume and could remain there on your machine consuming space.
- Do remember to clean up if you want.
- Alternatively, there is also a -v option while removing the container

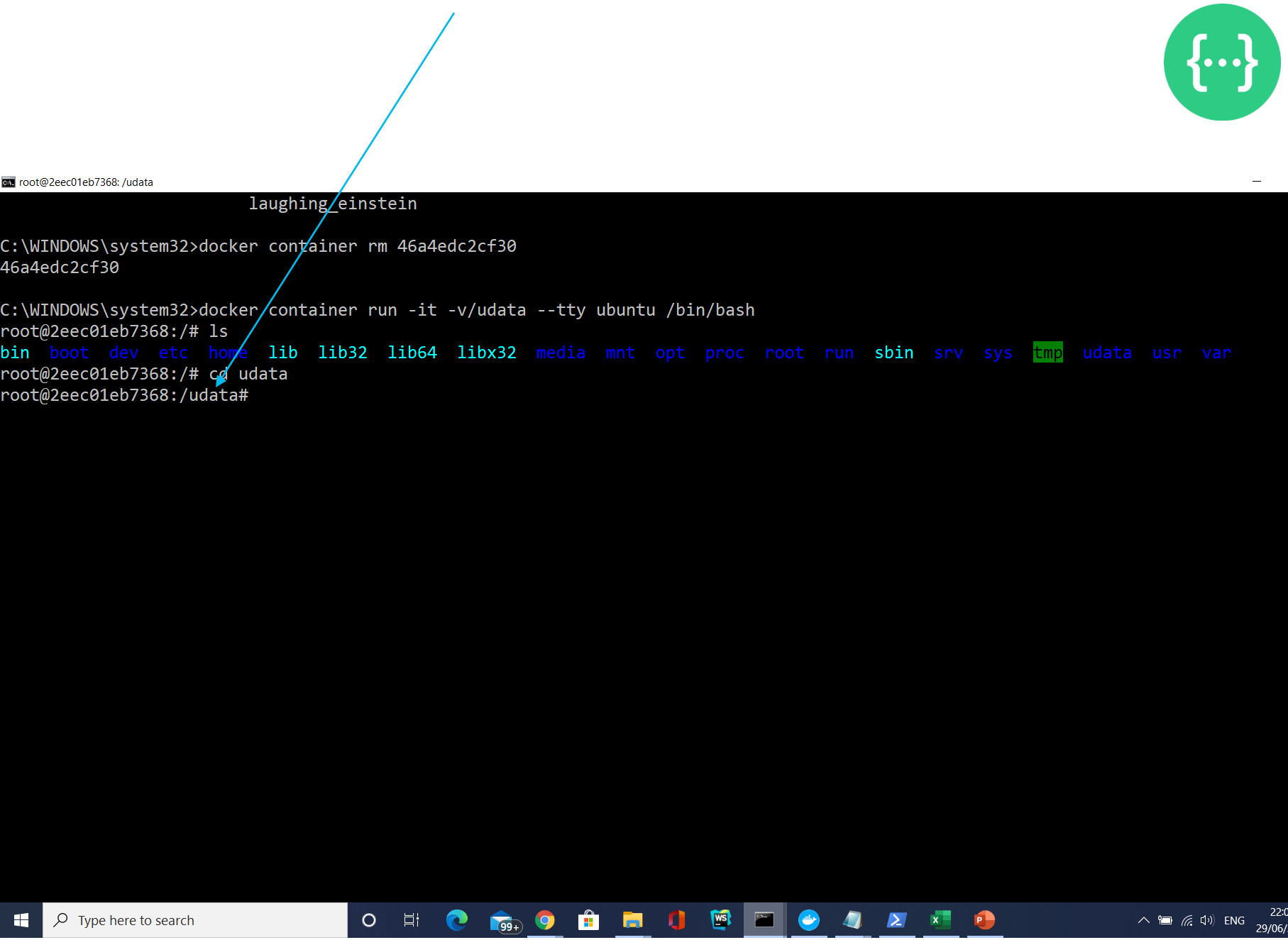
root@46a4edc2cf30: /

```
Error response from daemon: You cannot remove a running container e26b6d0c5de3bc889381c20b2d169ce8a61bf701dfe46aea127e0e03d314fb0e. Sto
p the container before attempting removal or force remove

C:\WINDOWS\system32>docker container stop e26b6d0c5de3
e26b6d0c5de3

C:\WINDOWS\system32>docker container rm e26b6d0c5de3
e26b6d0c5de3

C:\WINDOWS\system32>docker container run -it --tty ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
a4a2a29f9ba4: Pull complete
127c9761dcba: Pull complete
d13bf203e905: Pull complete
4039240d2e0b: Pull complete
Digest: sha256:35c4a2c15539c6c1e4e5fa4e554dac323ad0107d8eb5c582d6ff386b383b7dce
Status: Downloaded newer image for ubuntu:latest
root@46a4edc2cf30:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@46a4edc2cf30:/#
```

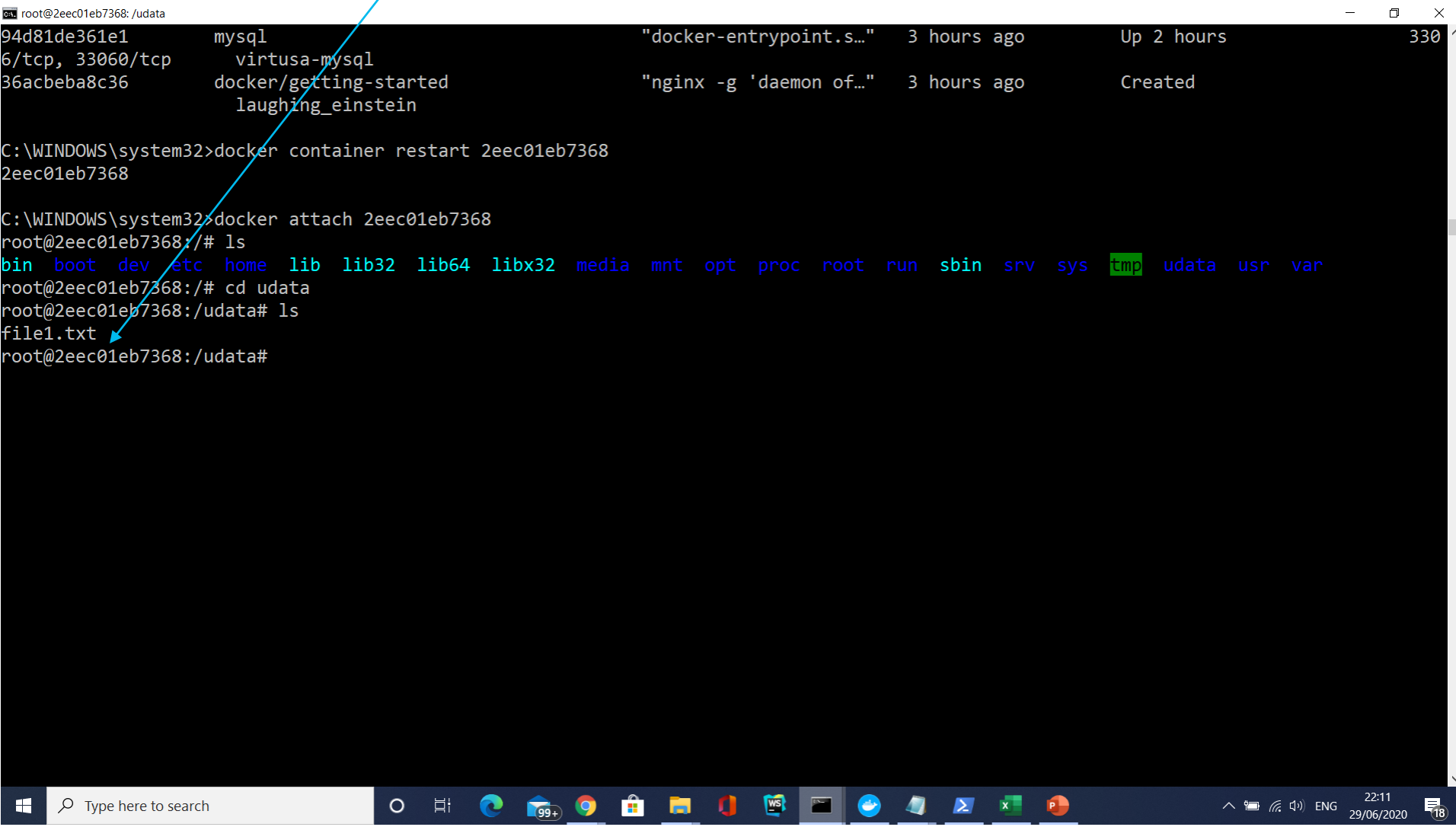

root@2eec01eb7368: /udata
laughing_einstein
C:\WINDOWS\system32>docker container rm 46a4edc2cf30
46a4edc2cf30
C:\WINDOWS\system32>docker container run -it -v/udata --tty ubuntu /bin/bash
root@2eec01eb7368:/# ls
bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp udata usr var
root@2eec01eb7368:/# cd udata
root@2eec01eb7368:/udata#
Type here to search
ENG
22:05
29/06/2020

Administrator: Command Prompt


```
                "LowerDir": "/var/lib/docker/overlay2/b79f152c6131e7d3c11ec0c3b019497c8a4f17ac12411a0a49719b21508c35a1-init/diff:/var/l
ib/docker/overlay2/3eab3aebc8c580252e330fe3efa59b8d92c1339cb4dbf46e8fc374dfef63f569/diff:/var/lib/docker/overlay2/3872c83edf6b5c30265e2
1691152c9126a2ffee424afc81202f02f07b1819210/diff:/var/lib/docker/overlay2/25d3f8faffb92bef36d1b620a6ababbd344c1cb58f41fb565620be058a0dc
f8e/diff:/var/lib/docker/overlay2/e4ebde681507d7b624d11dae676fbca8cb273d91e18e55a3d511a5b4fd0cd9ac/diff",
                "MergedDir": "/var/lib/docker/overlay2/b79f152c6131e7d3c11ec0c3b019497c8a4f17ac12411a0a49719b21508c35a1/merged",
                "UpperDir": "/var/lib/docker/overlay2/b79f152c6131e7d3c11ec0c3b019497c8a4f17ac12411a0a49719b21508c35a1/diff",
                "WorkDir": "/var/lib/docker/overlay2/b79f152c6131e7d3c11ec0c3b019497c8a4f17ac12411a0a49719b21508c35a1/work"
            },
            "Name": "overlay2"
        },
        "Mounts": [
            {
                "Type": "volume",
                "Name": "7b3c1be1134c62d876416f35818c1545d2a96c4aff4443b1457ebb0b08f840f3",
                "Source": "/var/lib/docker/volumes/7b3c1be1134c62d876416f35818c1545d2a96c4aff4443b1457ebb0b08f840f3/_data",
                "Destination": "/udata",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
        "Config": {
            "Hostname": "2eec01eb7368",
            "Domainname": "",
            "User": "",
            "AttachStdin": true,
            "AttachStdout": true,
            "AttachStderr": true,
            "Tty": true,
            "OpenStdin": true,
            "StdinOnce": true,
            "Env": [
```


Type here to search

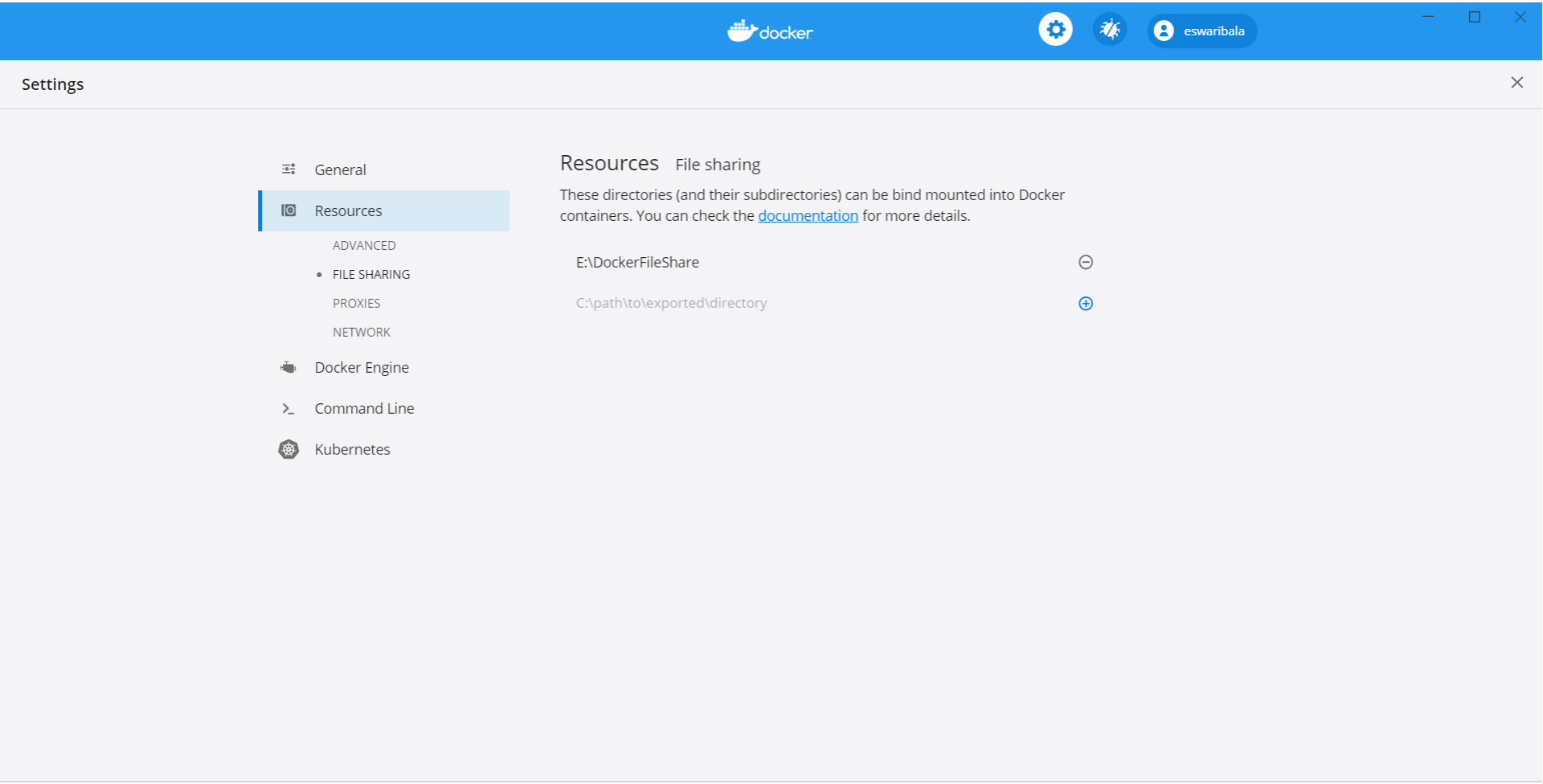
ENG 22:08 29/06/2020


root@2eec01eb7368: /udata
94d81de361e1 mysql "docker-entrypoint.s…" 3 hours ago Up 2 hours 330
6/tcp, 33060/tcp virtusa-mysql
36acbeba8c36 docker/getting-started "nginx -g 'daemon of…" 3 hours ago Created
laughing_einstein
C:\WINDOWS\system32>docker container restart 2eec01eb7368
2eec01eb7368
C:\WINDOWS\system32>docker attach 2eec01eb7368
root@2eec01eb7368:/# ls
bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp udata usr var
root@2eec01eb7368:/# cd udata
root@2eec01eb7368:/udata# ls
file1.txt
root@2eec01eb7368:/udata#
Type here to search
ENG
22:11
29/06/2020

Manage Data in Docker

- The sequence of steps are as follows:
- We restarted the container (container1)
- We attached to the running container (container1)
- We did a ls and found that our volume /data is still there.
- We did a cd into the /data volume and did a ls there. And our file is still present.

Mounting a Host Directory as a Data volume



docker container run -it -v e:/DockerFileShare/project/web01:/mnt/test ubuntu /bin/bash

Data volume containers

- Creating a Data volume container is very useful if you want to share data between containers or you want to use the data from non-persistent containers.
- The process is really two step:
  - You first create a Data volume container
  - Create another container and mount the volume from the container created in Step 1.

Data volume containers

- Creating a Data volume container is very useful if you want to share data between containers or you want to use the data from non-persistent containers.
- The process is really two step:
  - You first create a Data volume container
  - Create another container and mount the volume from the container created in Step 1.

Data volume containers

Administrator: Command Prompt - docker run -it -v /data --name container1 busybox

```
local               4e7fc43d326fc162732551862b8ec241772a468b891ee93e716c870af082b228
local               7b3c1be1134c62d876416f35818c1545d2a96c4aff4443b1457ebb0b08f840f3
local               030cf2d5467d3086b88f8d6e1d613a56426f16c9e398407b1082f168b3b01e4e
local               5834bcb24dcc7c574889dde2c5091956a1915c11a11baad89bf166a3bd00d9dd
local               a6176c52cd63f2ce3a787f61f7374bd36b5810406d4ae2d55bedcb2cd493dcb2

C:\WINDOWS\system32>docker run -it -v /data --name container1 busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
76df9210b28c: Pull complete
Digest: sha256:95cf004f559831017cdf4628aaf1bb30133677be8702a8c5f2994629f637a209
Status: Downloaded newer image for busybox:latest
/ # cd data
/data # touch file1.txt
/data # touch file2.txt
/data # ls
file1.txt  file2.txt
/data #
```

Data volume containers

Administrator: Command Prompt - docker run -it -v /data --name container1 busybox

```
local               4e7fc43d326fc162732551862b8ec241772a468b891ee93e716c870af082b228
local               7b3c1be1134c62d876416f35818c1545d2a96c4aff4443b1457ebb0b08f840f3
local               030cf2d5467d3086b88f8d6e1d613a56426f16c9e398407b1082f168b3b01e4e
local               5834bcb24dcc7c574889dde2c5091956a1915c11a11baad89bf166a3bd00d9dd
local               a6176c52cd63f2ce3a787f61f7374bd36b5810406d4ae2d55bedcb2cd493dcb2

C:\WINDOWS\system32>docker run -it -v /data --name container1 busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
76df9210b28c: Pull complete
Digest: sha256:95cf004f559831017cdf4628aaf1bb30133677be8702a8c5f2994629f637a209
Status: Downloaded newer image for busybox:latest
/ # cd data
/data # touch file1.txt
/data # touch file2.txt
/data # ls
file1.txt  file2.txt
/data #
```

Docker network

Administrator: Command Prompt

```
76df9210b28c: Pull complete
Digest: sha256:95cf004f559831017cdf4628aaf1bb30133677be8702a8c5f2994629f637a209
Status: Downloaded newer image for busybox:latest
/ # cd data
/data # touch file1.txt
/data # touch file2.txt
/data # ls
file1.txt  file2.txt
/data # exit

C:\WINDOWS\system32>docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
5cc9ae61fdda        bridge              bridge              local
3857a50b891b        host                host                local
e43a4f132e84        none                null                local

C:\WINDOWS\system32>
```

Docker network

- docker network inspect bridge
- Docker automatically creates a subnet and gateway for the bridge network, and docker run automatically adds containers to it. I
- If you have containers running on your network, docker network inspect displays networking information for your containers.

Docker network

- Any containers on the same network may communicate with one another via IP addresses.
- Docker does not support automatic service discovery on bridge.
- You must connect containers with the --link option in your docker run command.

Docker network

- The Docker bridge supports port mappings and docker run --link allowing communications between containers on the docker0 network.
- However, these error-prone techniques require unnecessary complexity.
- It’s better to define your own networks instead.

Docker network

- None
- This offers a container-specific network stack that lacks a network interface.
- This container only has a local loopback interface (i.e., no external network interface).

Docker network

- Host
- This enables a container to attach to your host's network (meaning the configuration inside the container matches the configuration outside the container).

Defining your own networks

- You can create multiple networks with Docker and add containers to one or more networks.
- Containers can communicate within networks but not across networks.
- A container with attachments to multiple networks can connect with all of the containers on all of those networks.
- This lets you build a “hub” of sorts to connect to multiple networks and separate concerns.

Defining your own networks

- Bridge networks (similar to the default docker0 network) offer the easiest solution to creating your own Docker network.
- While similar, you do not simply clone the default0 network, so you get some new features and lose some old ones.
- Follow along below to create your own my_isolated_bridge_network and run your Postgres container my_mysql_db on that network:

Defining your own networks

- docker network create --driver bridge my_isolated_bridge_network
- docker network inspect my_isolated_bridge_network
- docker network ls
- docker run --net=my_isolated_bridge_network --name=my_psql_db postgres
- docker run --network=my_isolated_bridge_network --name=my_app hello-world

Defining your own networks

Administrator: Command Prompt


```
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "fce4adc125341807e7b334cb2908436a2a5b9596aa6b08749203f9c6e18d6dbb": {
                "Name": "competent_darwin",
                "EndpointID": "ca6a9d07770e8226967cf6c601d327cfe19b7b15396a09229f0e540f83f066e1",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]

C:\WINDOWS\system32>
```

Creating an overlay network

- If you want native multi-host networking, you need to create an overlay network.
- These networks require a valid key-value store service, such as Consul, Etcd, or ZooKeeper.
- You must install and configure your key-value store service before creating your network.
- Your Docker hosts (you can use multiple hosts with overlay networks) must communicate with the service you choose.
- Each host needs to run Docker.
- You can provision the hosts with Docker Machine.

Creating an overlay network

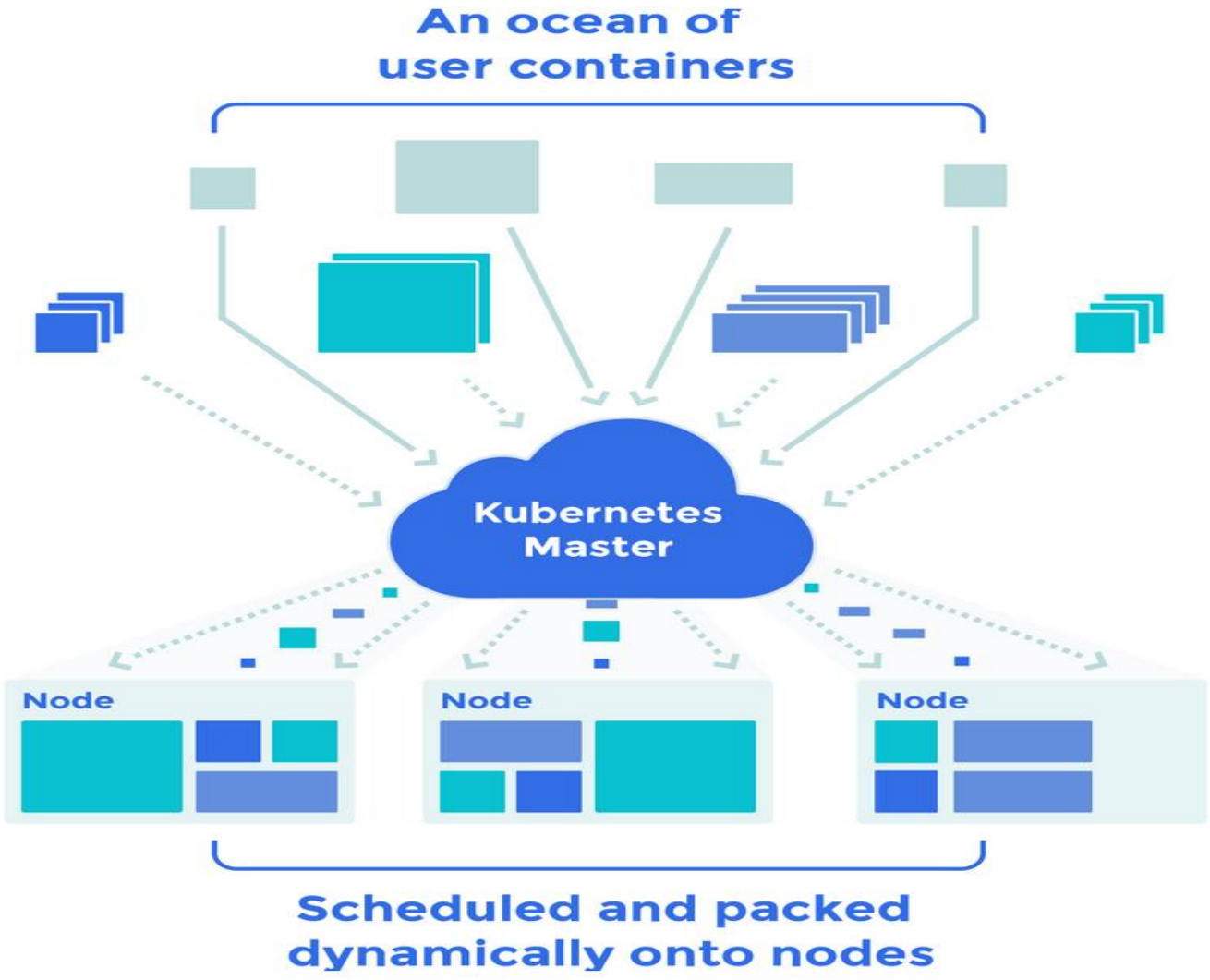
- Create the overlay network in a similar manner to the bridge network (network name my_multi_host_network):
- docker network create --driver overlay my_multi_host_network
- Launch containers on each host; make sure you specify the network name:
- docker run -itd -net=my_multi_host_network my_python_app

Getting Started with Artifactory as a Docker Registry

- Login to your repository use the following command with your Artifactory Cloud credentials.
- docker login ${server-name}-{repo-name}.jfrog.io
- Pull an image using the following command.
- docker pull ${server-name}-{repo-name}.jfrog.io/<image name>
- Push an image by first tagging it and then using the push command.
- docker tag <image name> ${server-name}-{repo-name}.jfrog.io/<image name>
- docker push ${server-name}-{repo-name}.jfrog.io/<image name>

Getting Started with Artifactory as a Docker Registry

- In this example, the Artifactory Cloud server is named **acme**.
- Start by creating a virtual Docker repository called dockerv2-virtual.
- Pull the hello-world image
- docker pull hello-world
- Login to repository dockerv2-virtual
- docker login acme-dockerv2-virtual.jfrog.io
- Tag the hello-world image
- docker tag hello-world acme-dockerv2-virtual.jfrog.io/hello-world
- Push the tagged hello-world image to dockerv2-virtual
- docker push acme-dockerv2-virtual.jfrog.io/hello-world


An ocean of
user containers
Kubernetes
Master
Node
Node
Node
Scheduled and packed
dynamically onto nodes

Docker Swarm

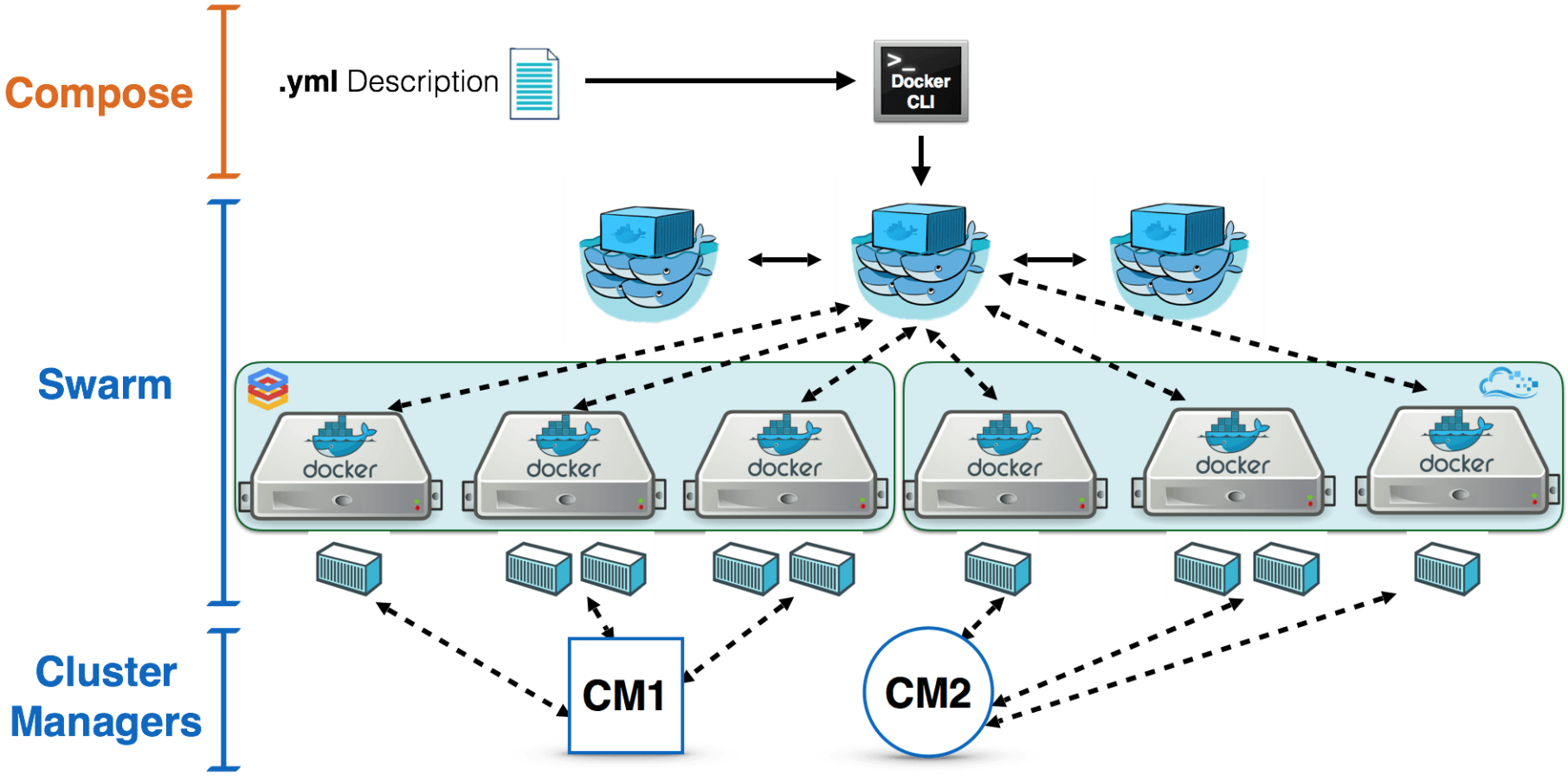
- Docker Swarm is Docker's native feature to support clustering of Docker machines.
- This enables multiple machines running Docker Engine to participate in a cluster, called Swarm.
- The Docker engines contributing to a Swarm are said to be running in Swarm mode.
- Machines enter into the Swarm mode by either initializing a new swarm or by joining an existing swarm.
- To the end user the swarm would seem like a single machine.
- A Docker engine participating in a swarm is called a node

Docker Swarm

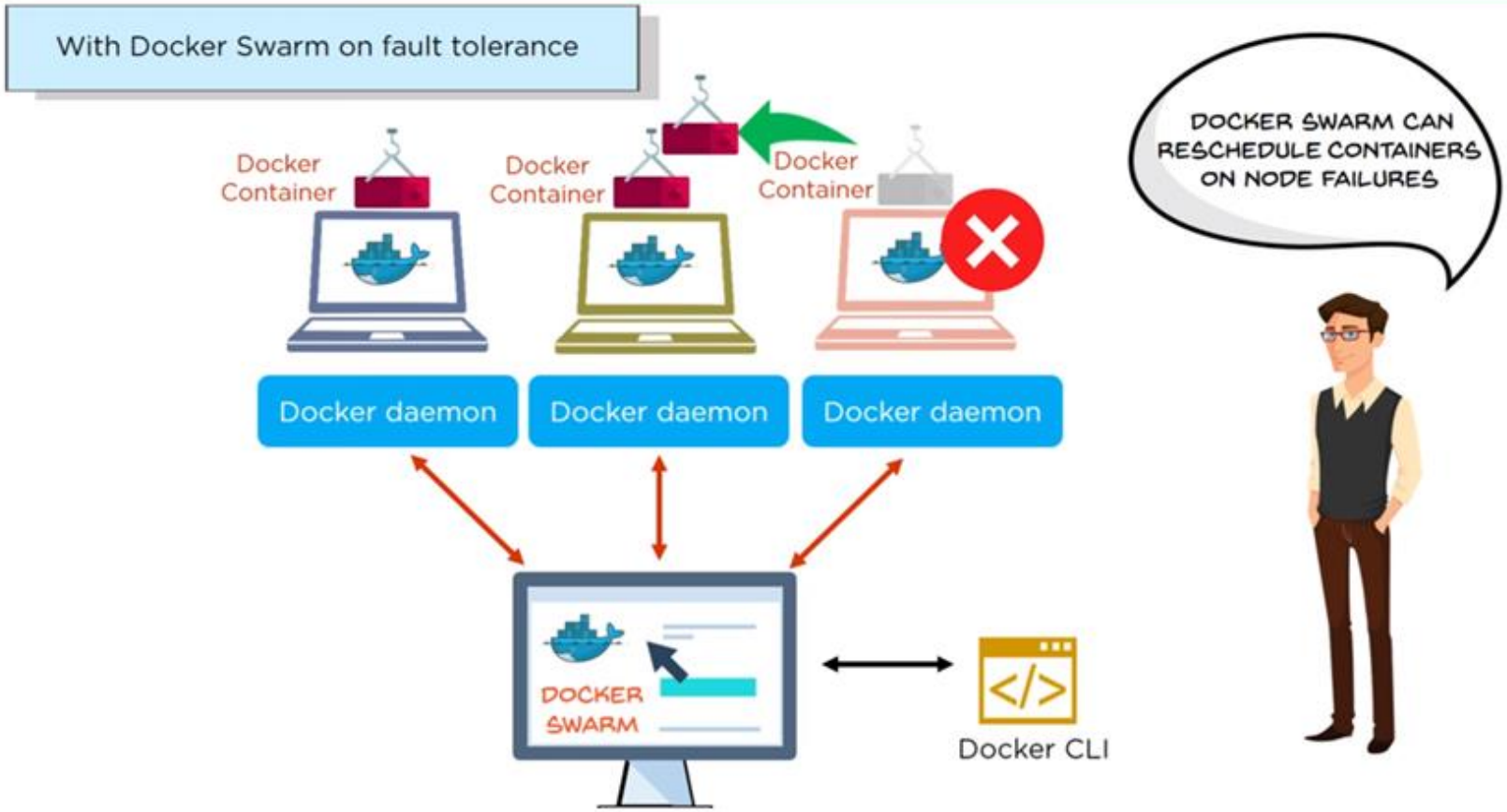
- . A node can either be a manager node or a worker node.
- The manager node performs cluster management and orchestration while the worker nodes perform tasks allocated by the manager.
- A manager node itself, unless configured otherwise, is also be a worker node.
- The central entity in the Docker Swarm infrastructure is called a service.
- A Docker swarm executes services. The user submits a service to the manager node to deploy and execute.
- A service is made up of many tasks. A task is the most basic work unit in a Swarm.
- A task is allocated to each worker node b the manager node.

Docker Swarm

- The Docker ecosystem consists of tools from development to production deployment frameworks.
- In that list, docker swarm fits into cluster management.
- A mix of docker-compose, swarm, overlay network and a good service discovery tool such as etcd or consul can be used for managing a cluster of Docker containers.
- Docker swarm is still maturing in terms of functionalities when compared to other open-source container cluster management tools.
- Considering the vast docker contributors, it won't be so long for docker swarm to have all the best functionalities other tools possess.
- Docker has documented a good production plan for using docker swarm in production.


Compose
.yml Description
Docker CLI
Swarm
docker
docker
docker
docker
docker
docker
Cluster Managers
CM1
CM2

What is Docker Swarm?



Features of Docker Swarm

Decentralized access

High security

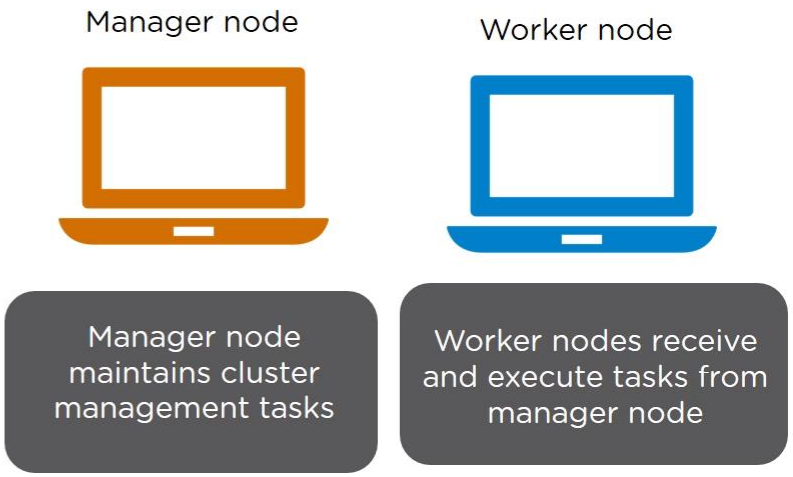
Auto load balancing

High scalability

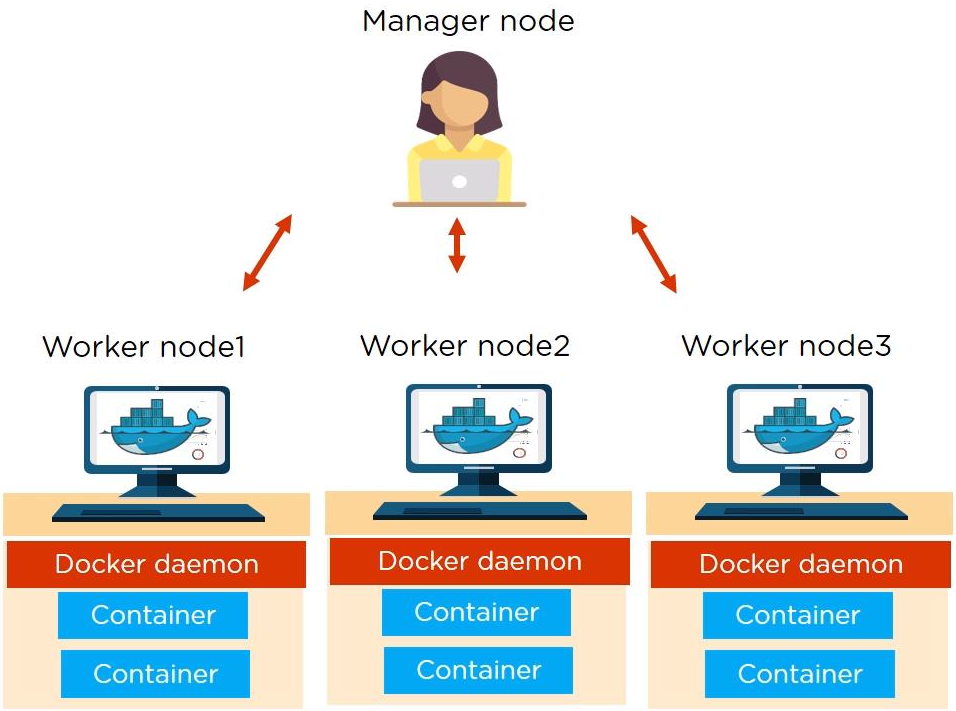
Roll-back a task

Docker Swarm

- In Swarm, containers are launched using services
- A service is a group of containers of the same image
- Services enables to scale your application
- Before you can deploy a service in Docker Swarm, you must have at least one node deployed
- There are two types of nodes in Docker Swarm



Architecture of Docker Swarm



Docker Swarm Steps

Command Prompt

```
C:\Users\Balasubramaniam>docker swarm init
Swarm initialized: current node (79assnkjjut36pv3blv7tb1lz) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-47gh210k487vjtj5diybgi1l3mmkxt6qxbz6m725yod5z8poq7-3idk04u6gonz4elxrf0vde644 192.168.65.3:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.


C:\Users\Balasubramaniam>
```

Command Prompt

```
C:\Users\Balasubramaniam> docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS      ENGINE VERSION
aw1gnnexmdexbwncvlj72c723 *   docker-desktop      Ready               Active              Leader              19.03.13

C:\Users\Balasubramaniam>
```

```
Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service create --name eurekaswarm -p 8761:8761 eureka-app:latest
image eureka-app:latest could not be accessed on a registry to record
its digest. Each node will access eureka-app:latest independently,
possibly leading to different nodes running different
versions of the image.

0uw58z9g5vbjl13zzzjop10k9
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged

Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
0uw58z9g5vbj        eurekaswarm         replicated          1/1                 eureka-app:latest   *:8761->8761/tcp

Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service scale eurekaswarm=3
eurekaswarm scaled to 3
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged

Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
$ docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE               PORTS
0uw58z9g5vbj        eurekaswarm         replicated          3/3                 eureka-app:latest   *:8761->8761/tcp

Balasubramaniam@DESKTOP-55AGI0I MINGW64 /d/Program Files/Docker Toolbox
```

Questions

Module Summary

- In this module we discussed
  - Overview of Maven
  - Maven archetypes
  - Maven life cycle phases
  - The pom.xml file
  - Creation of Java projects using Maven
  - Creation of war files