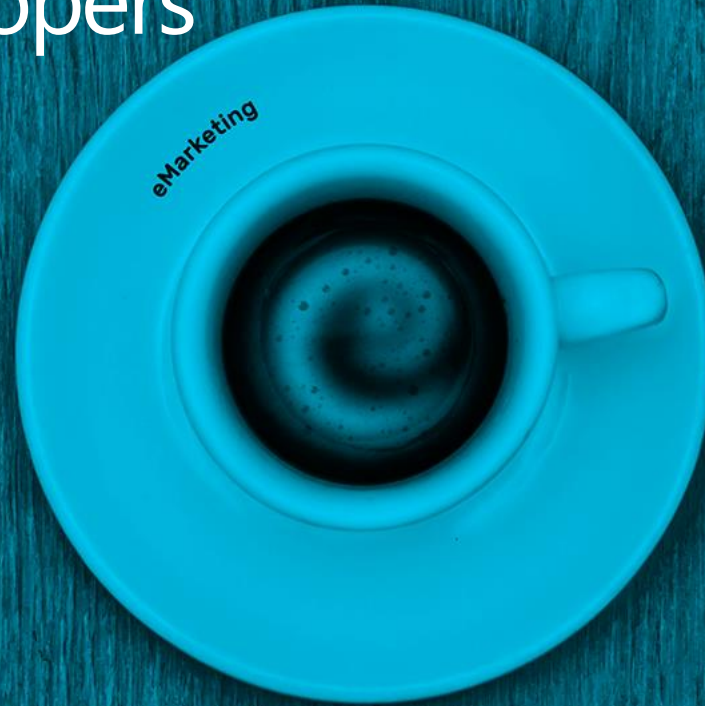




# Course: Docker for Web Developers

4/2021



One Stop Shop

eCommerce





# Who am I



# AGENDA DAY 1 - DOCKER

- What is docker?
- Containers VS. VMs
- Docker Architecture
- IMAGES
- Registries
- INSTALLING DOCKER
- Docker Tool
- Docker file
- Logging
- Volumes



# AGENDA DAY 2 - DOCKER

- network
- Docker compose
- Docker on visual studio



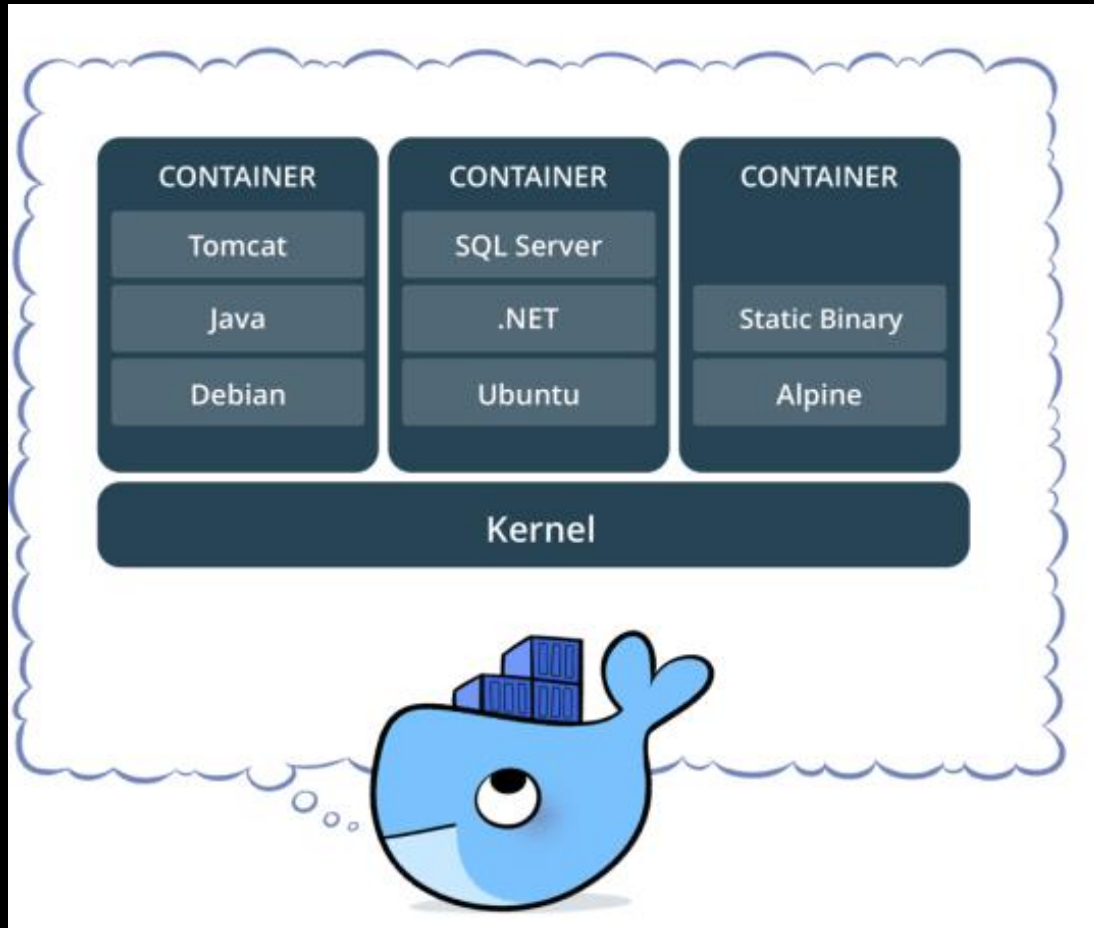
# QUESTIONS FOR YOU...

- What Do You Know About Docker?
- Who Used Docker For Development / QA / STG / PROD?
- Who Tried & Failed Implementing Docker

# WHAT IS DOCKER?

- ✓ Developed by DotCloud Inc. ( Currently Docker Inc.)
- ✓ open platform for developing, shipping, and running applications on containers
- ✓ Released it as open source 7+ years back
- ✓ written in the GO programming language
- ✓ Possible to set up in any OS, be it Windows, OSX, Linux - It work the same way
- ✓ Guaranteed to run the same way - Your development desktop, a bare-metal server, virtual machine, data center, or cloud

# WHAT IS DOCKER?



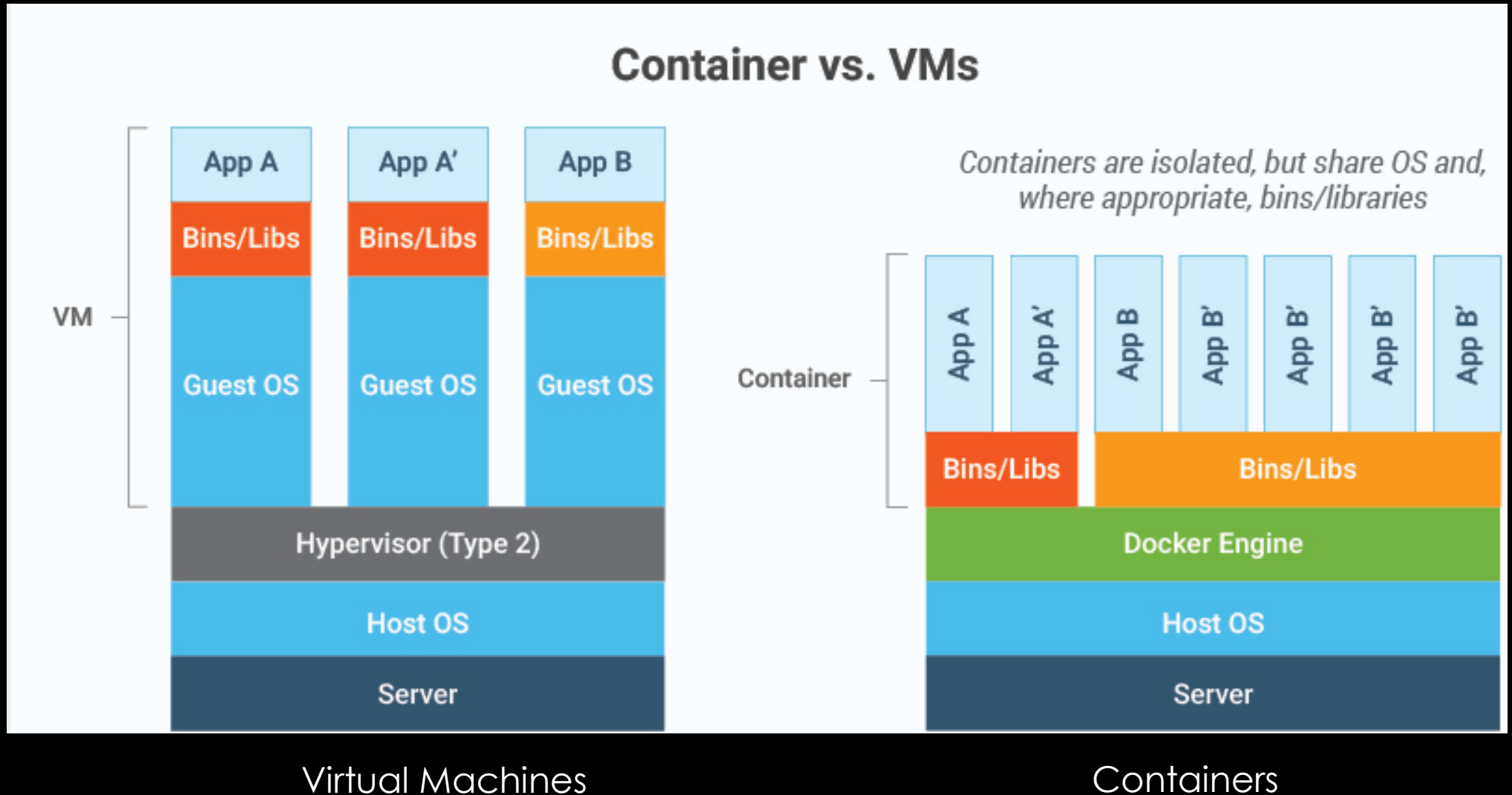
- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works for all major Linux distributions
- Containers native to win Server 2016

# WHAT IS DOCKER?

- **Flexible:** Even the most complex applications can be containerized.
- **Lightweight:** Containers leverage and share the host kernel.
- **Interchangeable:** You can deploy updates and upgrades on-the-fly.
- **Portable:** You can build locally, deploy to the cloud, and run anywhere.
- **Scalable:** You can increase and automatically distribute container replicas.
- **Stackable:** You can stack services vertically and on-the-fly.



# CONTAINERS VS. VMS



# VM VS DOCKER - SIMILARITY

Virtual Machines	Docker
Process in one VM can't see processes in other VMs	Process in one container can't see processes in other container
Each VM has its own root filesystem	Each container has its own root file system(Not Kernel)
Each VM gets its own virtual network adapter	Docker can get virtual network adapter. It can have separate IP and ports
VM is a running instance of physical files(.VMX and .VMDK)	Docker containers are running instances of Docker Image
Host OS can be different from guest OS	Host OS can be different from Container OS

# VM VS DOCKER - DIFFERENCE

Virtual Machines	Docker
Each VM runs its own OS	All containers share the same Kernel of the host
Boot up time is in minutes	Containers instantiate in seconds
VMs snapshots are used sparingly	Images are built incrementally on top of another like layers. Lots of images/snapshots
Not effective diffs. Not version controlled	Images can be diffed and can be version controlled. Dockerhub is like GITHUB
Cannot run more than couple of VMs on an average laptop	Can run many Docker containers in a laptop.
Only one VM can be started from one set of VMX and VMDK files	Multiple Docker containers can be started from one Docker image



# CONTAINERS VS. VMS



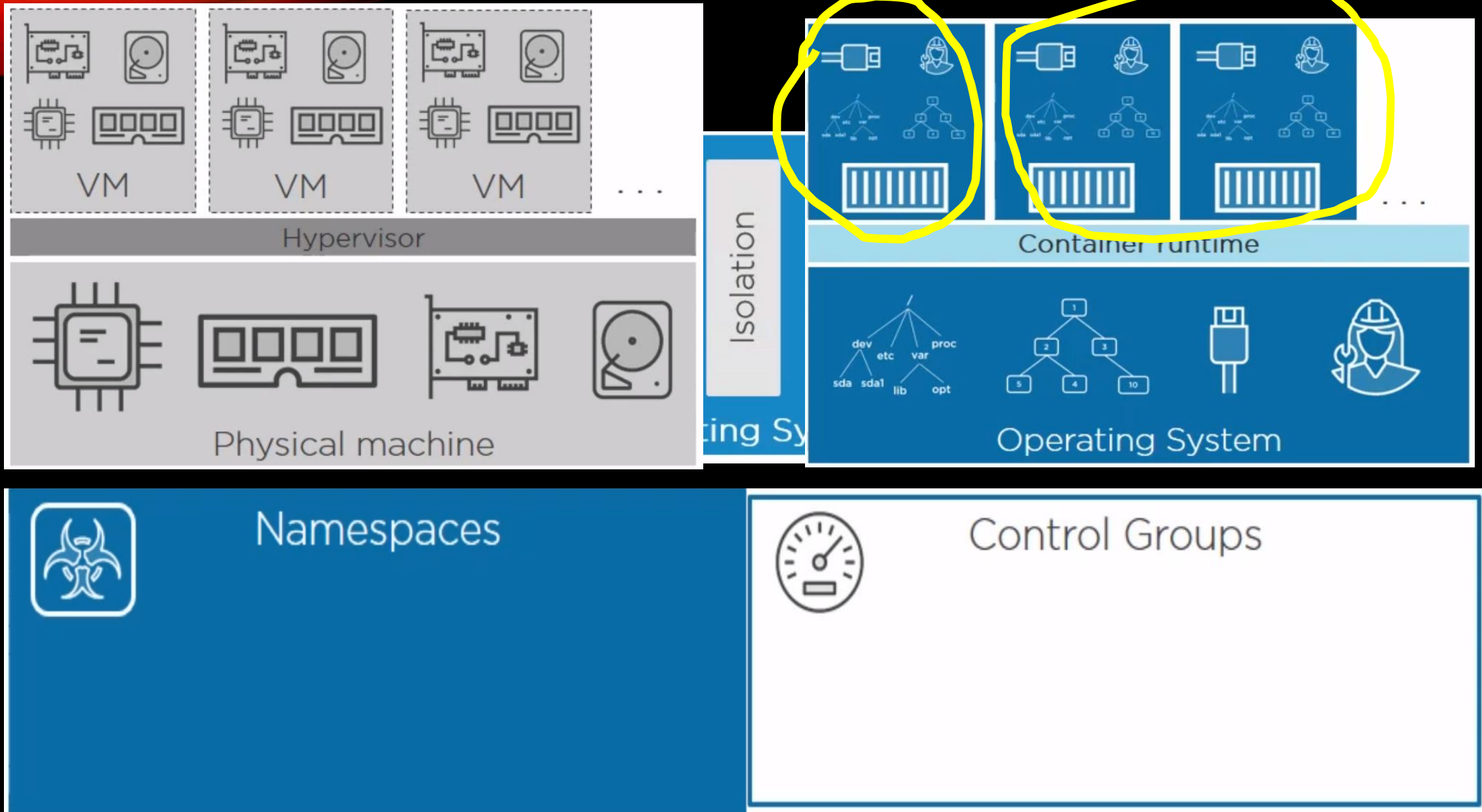
VMs



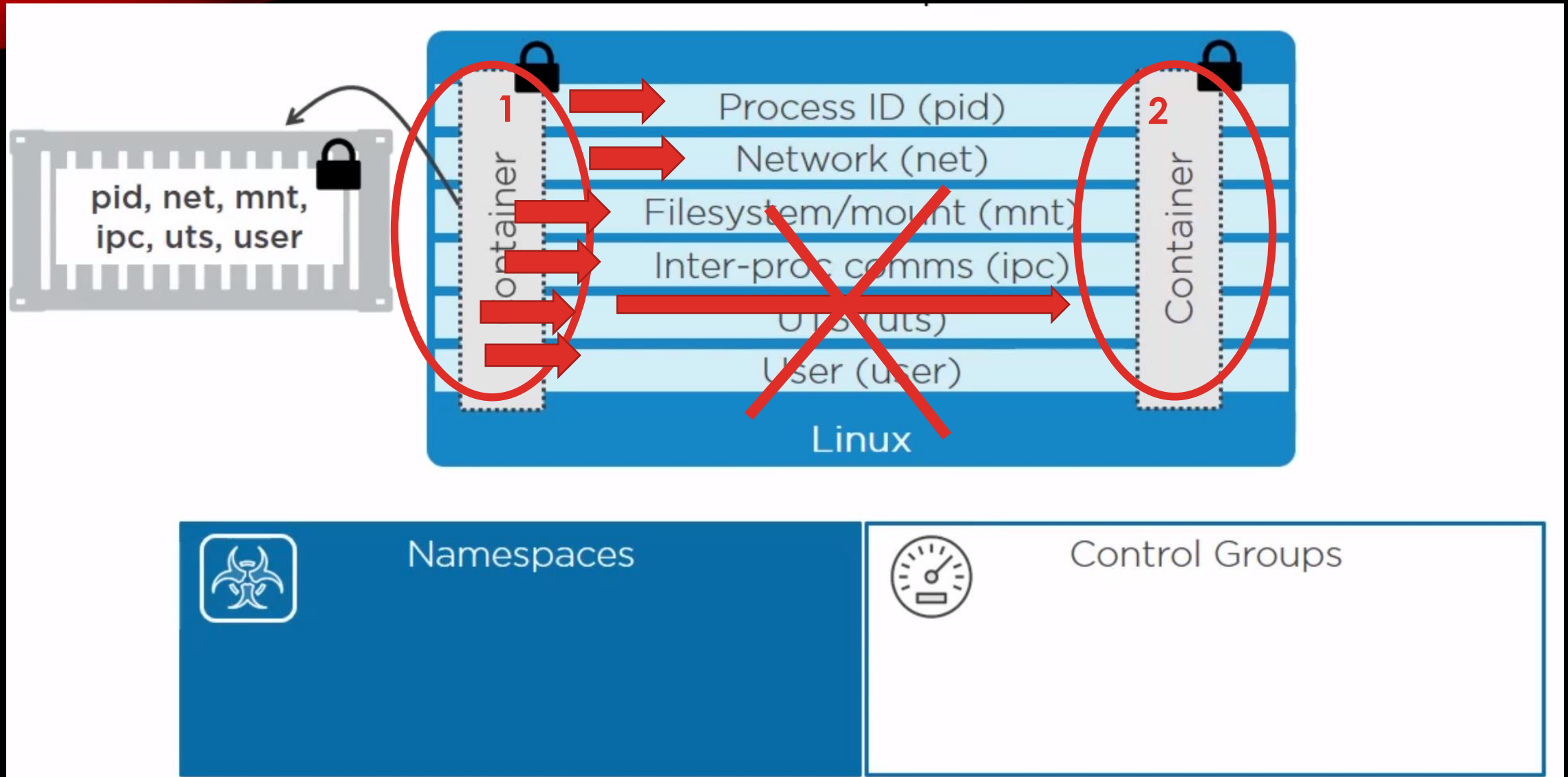
Containers



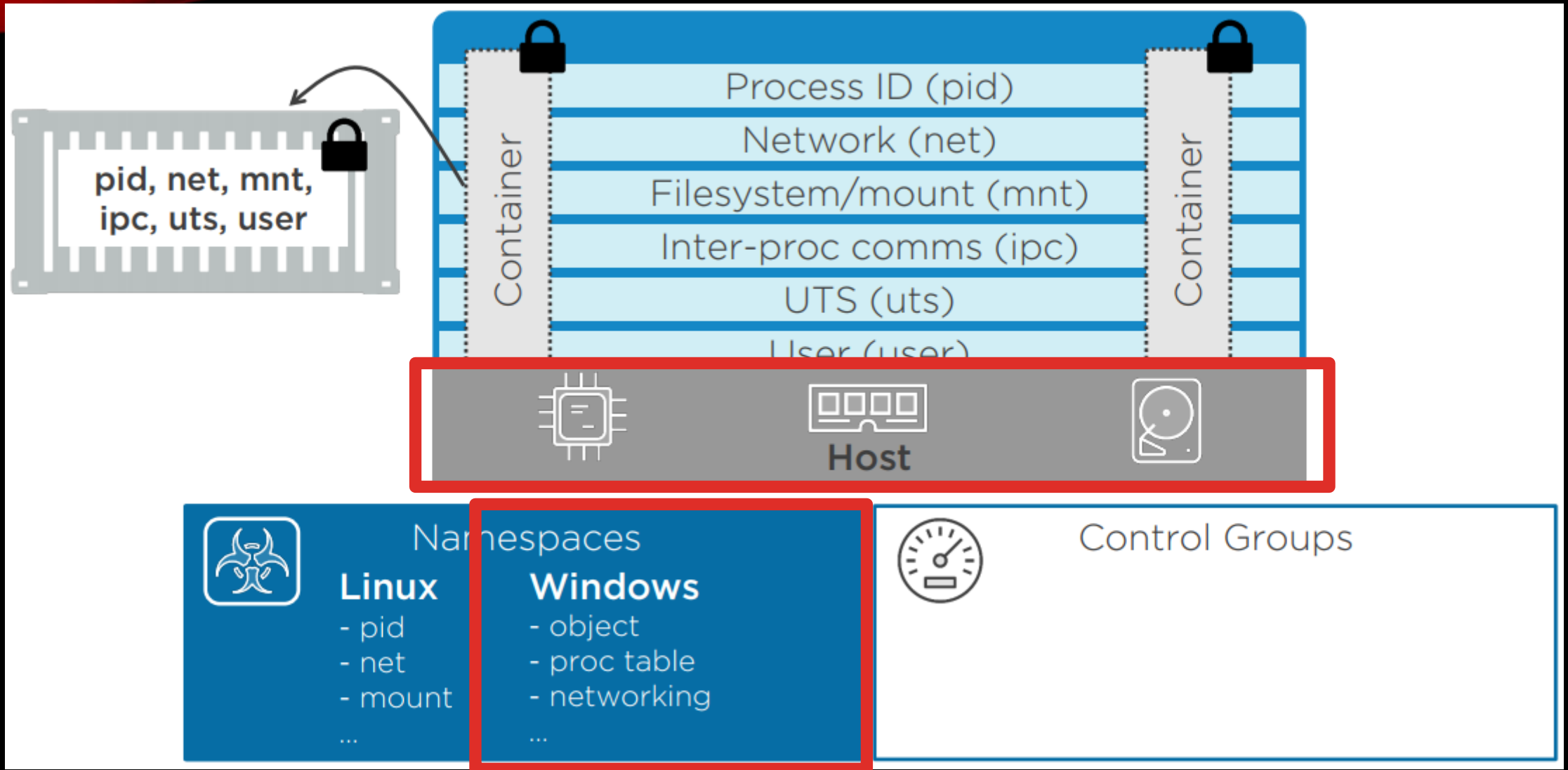
# DOCKER ARCHITECTURE - DOCKER ENGINE



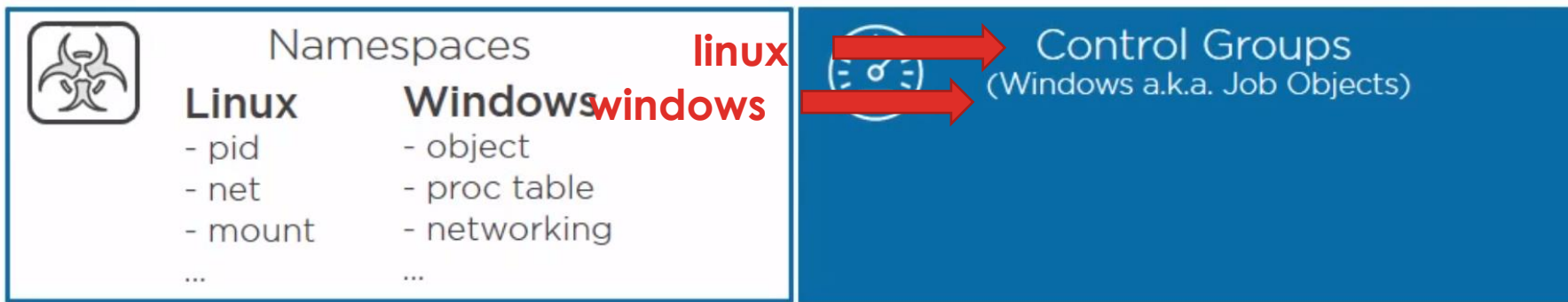
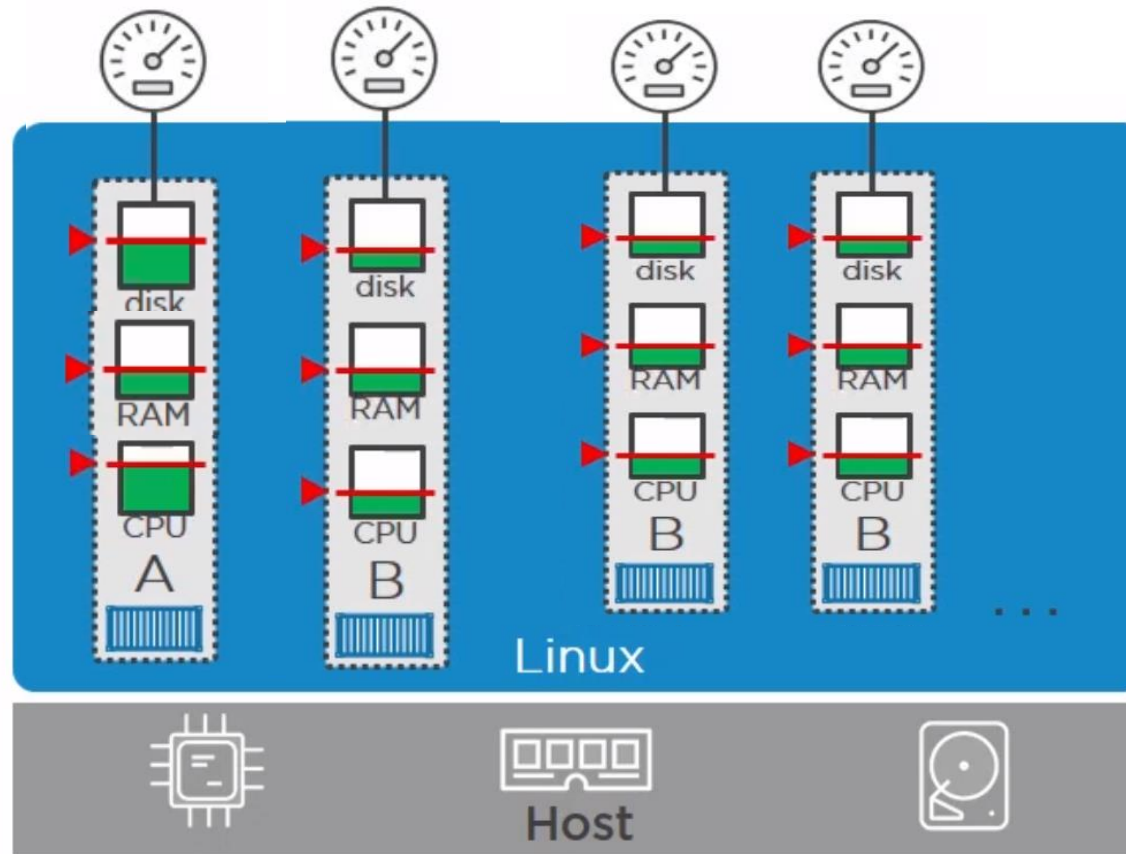
# DOCKER ARCHITECTURE - LINUX NAMESPACE



# DOCKER ARCHITECTURE - NAMESPACE

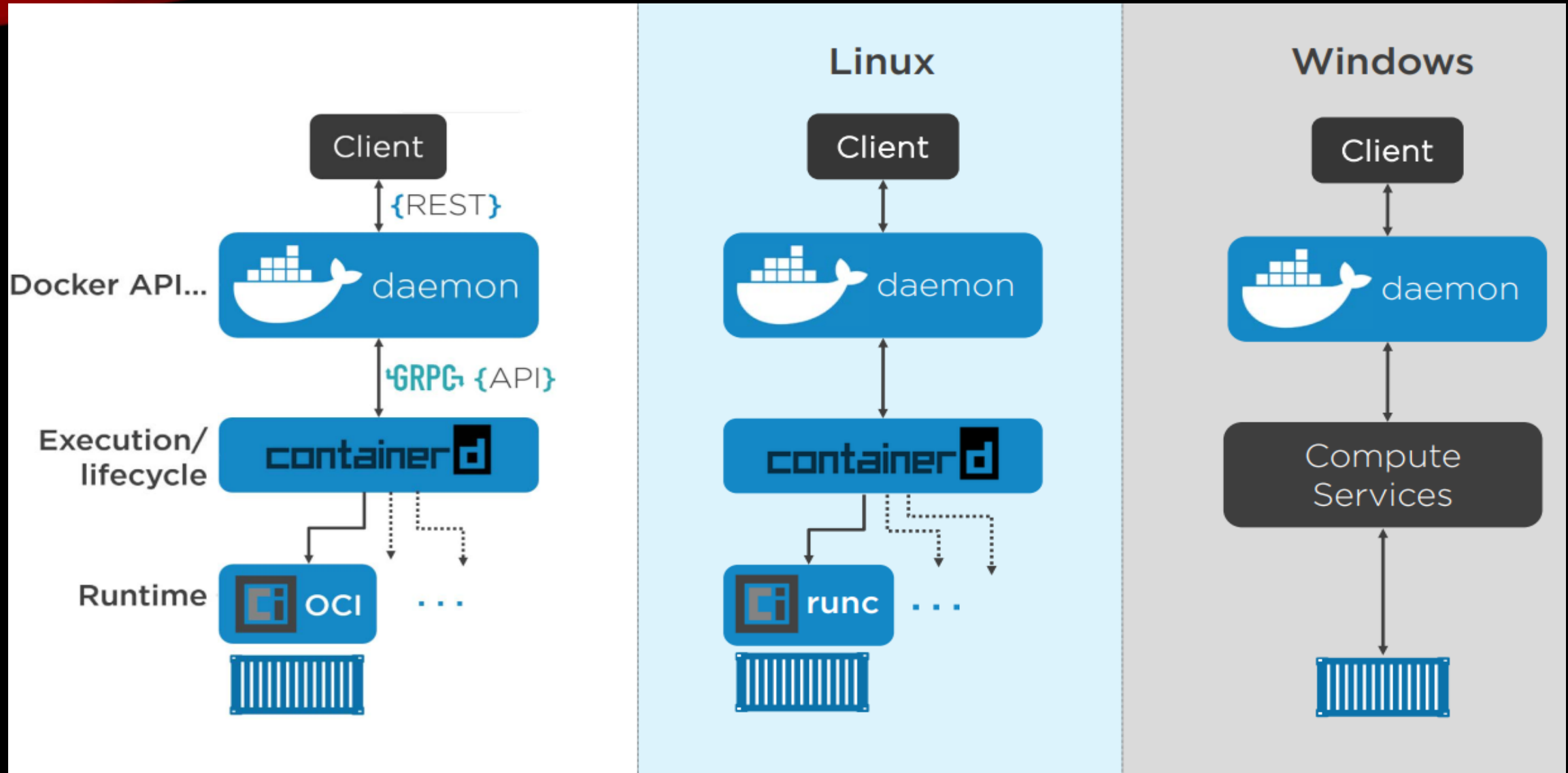


# DOCKER ARCHITECTURE - CONTROL GROUPS

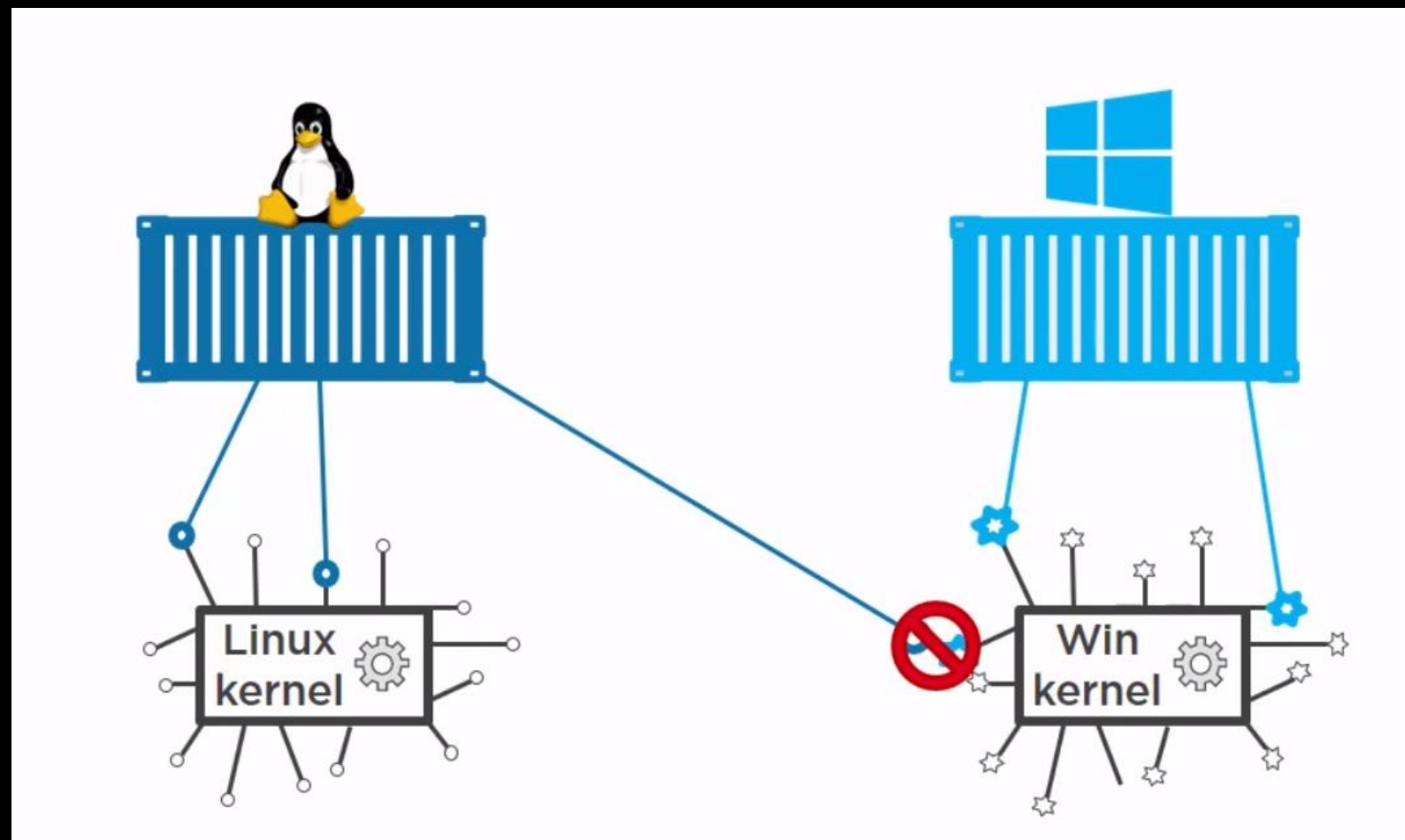




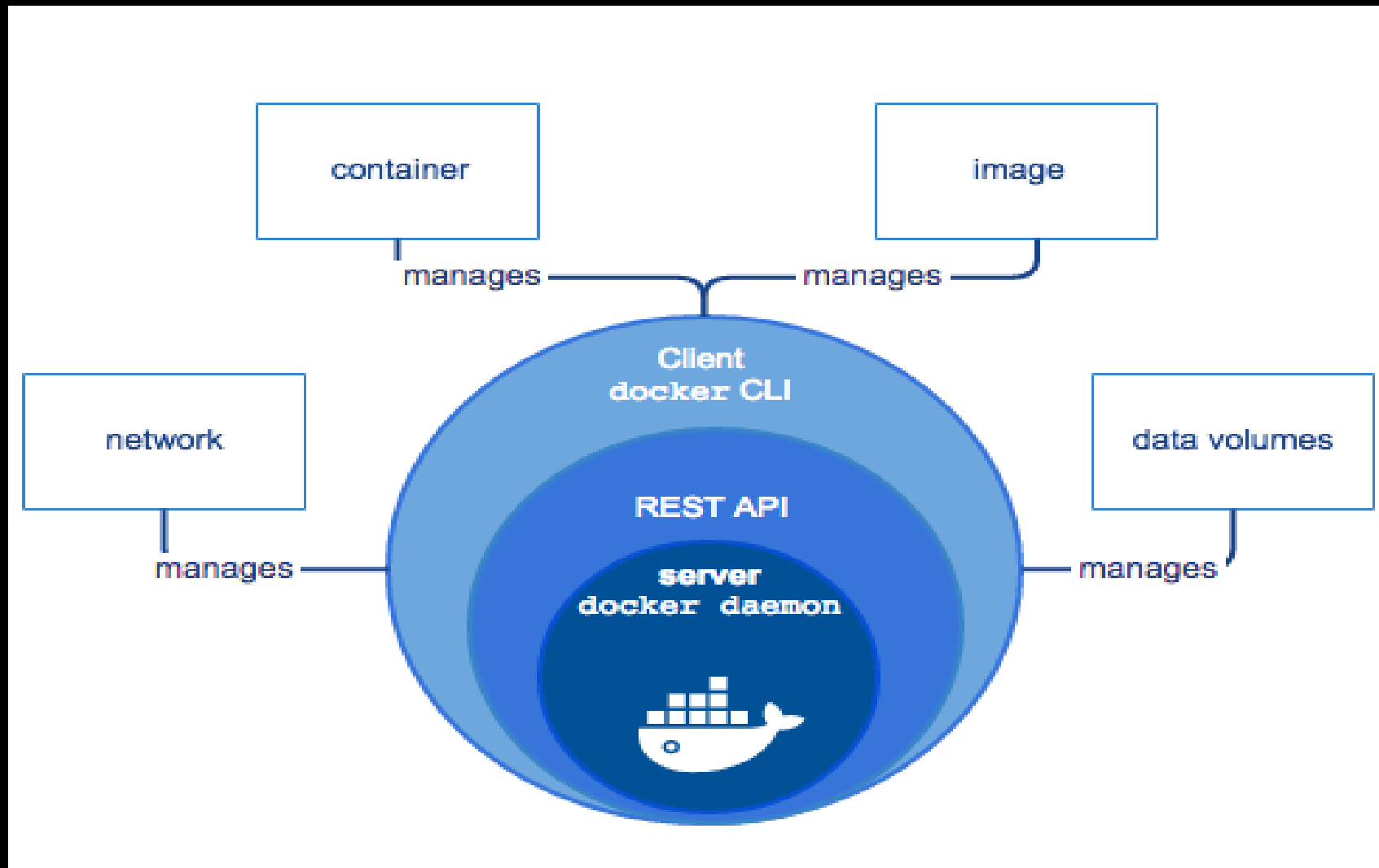
# DOCKER ARCHITECTURE - WINDOWS CONTAINERS



# DOCKER ARCHITECTURE - WINDOWS CONTAINERS



# DOCKER ARCHITECTURE –BIG PICTURE



# DOCKER ARCHITECTURE

Docker uses a client-server architecture.

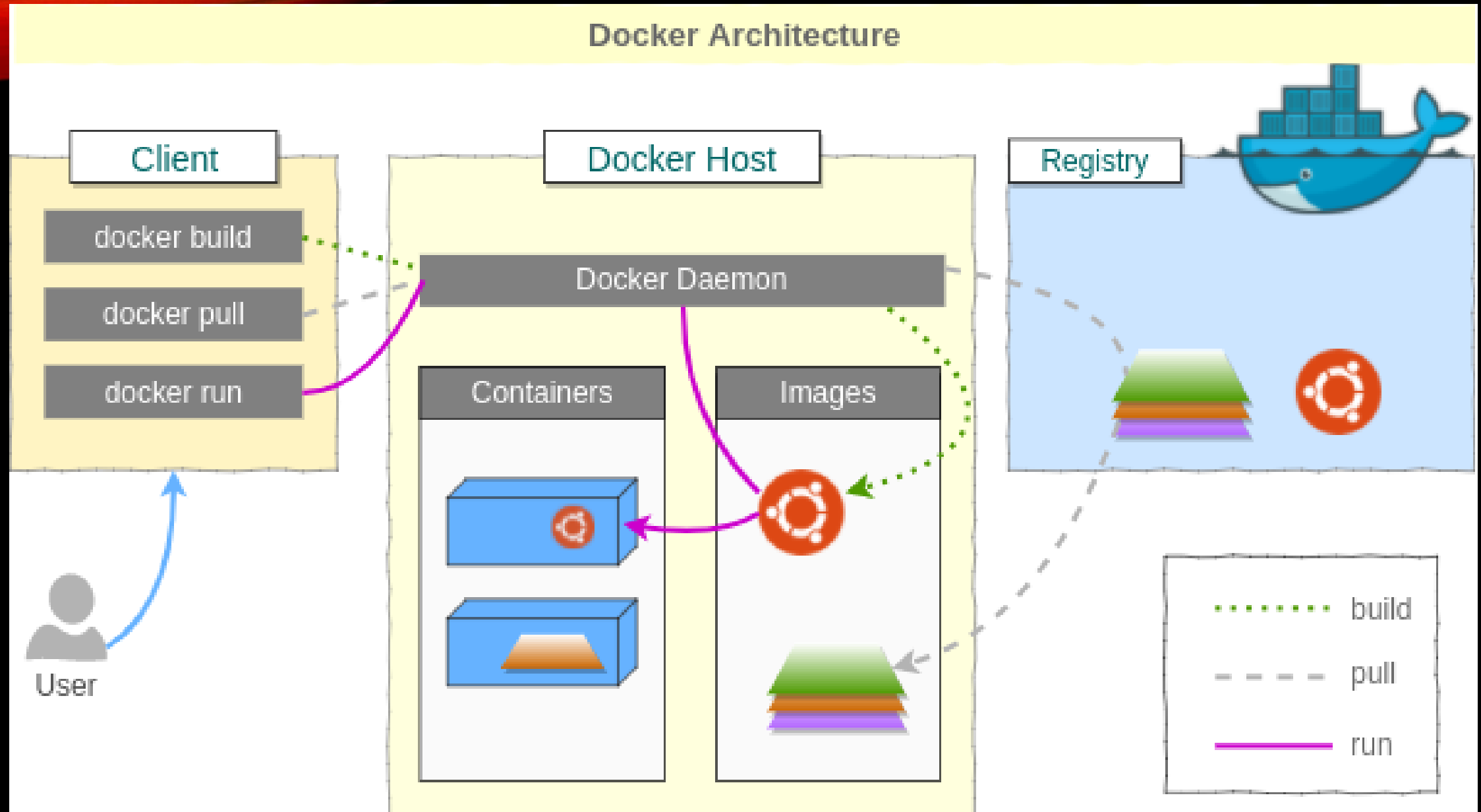
The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers.

The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon.

The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.

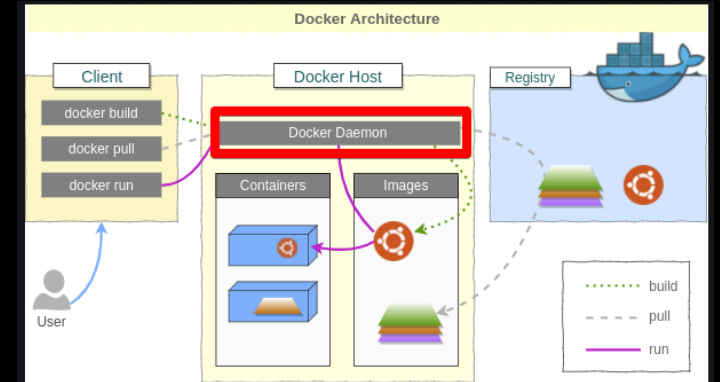


# DOCKER ARCHITECTURE



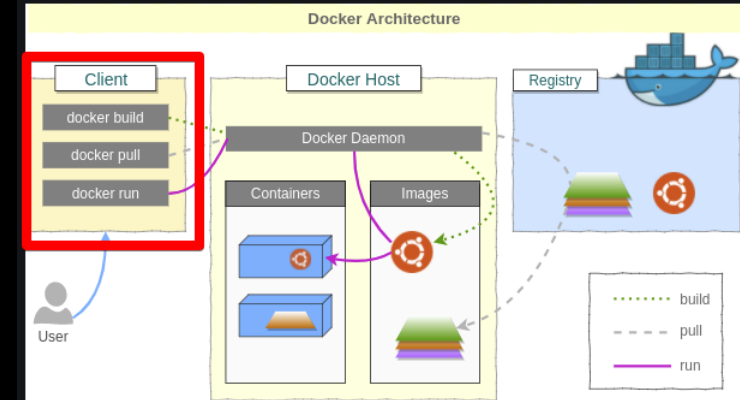
# DOCKER DAEMON

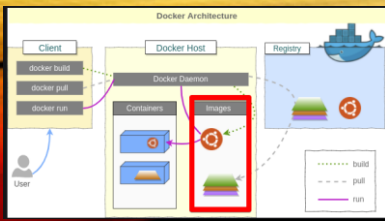
- Name: dockerd
- listens for Docker API requests
- manages Docker objects such as images, containers, networks, and volumes.
- A daemon can also communicate with other daemons to manage Docker services.



# DOCKER CLIENT

- Name: from cli docker
- is the primary way that many Docker users interact with Docker.
- When you use commands such as docker run, the client sends these commands to docker daemon , which carries them out.
- The docker command uses the Docker API.
- The Docker client can communicate with more than one daemon.



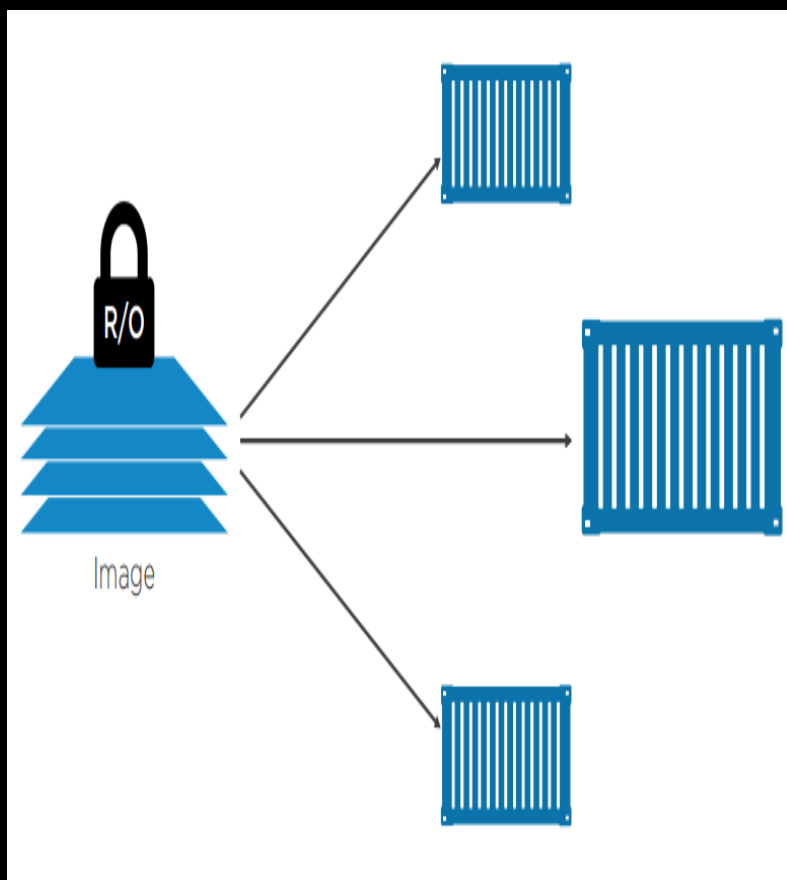


# DOCKER OBJECTS - IMAGES

- An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization.
- You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it.
- Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.



# DOCKER OBJECTS - IMAGES



```
C:\> Administrator: C:\Windows\System32\cmd.exe

Z:\> docker image pull redis
Using default tag: latest
latest: Pulling from library/redis
8ec398bc0356: Pull complete
da01136793fa: Pull complete
cf1486a2c0b8: Pull complete
a44f7da98d9e: Pull complete
c677fde73875: Pull complete
727f8da63ac2: Pull complete
Digest: sha256:90d44d431229683cadd75274e6fcb22c3e0396d149a8f8b7da9925021ee75c30
Status: Downloaded newer image for redis:latest
docker.io/library/redis:latest

Z:\> docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
redis	latest	9b188f5fb1e6	2 weeks ago	98.2MB

# DOCKER OBJECTS - IMAGES

## Image Manifest

```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
  "config": {
    "mediaType": "application/vnd.docker.container.image.v1+json",
    "size": 7813,
    "digest": "sha256:b5b25c5f9e994348a98115408093aaa861712696415d9bca9f432a537bc3f"
  },
  "layers": [
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 32654,
      "digest": "sha256:c692413e4c0f99ca6895a6481747baa33ee08896d9652f4e05ed7fc33"
    },
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 30733,
      "digest": "sha256:5c3a084a5f5dc327416d9b421c4370ca5b128f4bf398f635a2ebdab"
    },
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 73189,
      "digest": "sha256:ec458955958665577941c89412c1a9665f763694ac3da7f21284862a0671"
    },
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 73189,
      "digest": "sha256:ec458955958665577941c89412c1a9665f763694ac3da7f21284862a0671"
    }
  ]
}
```

Files & objects



Files & objects



Files & objects

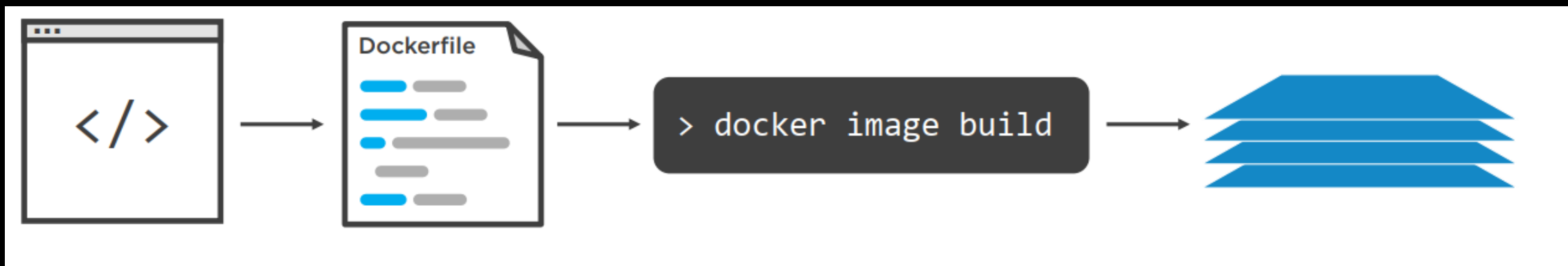


Files & objects



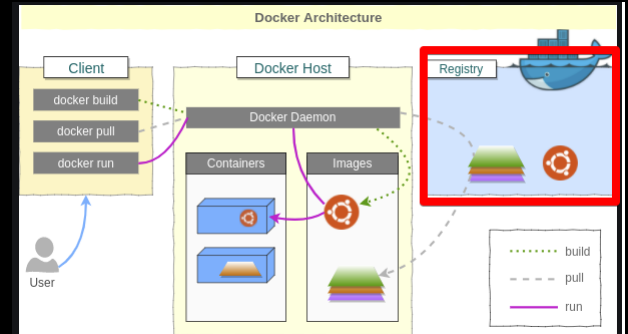
Layers

# DOCKER OBJECTS - IMAGES



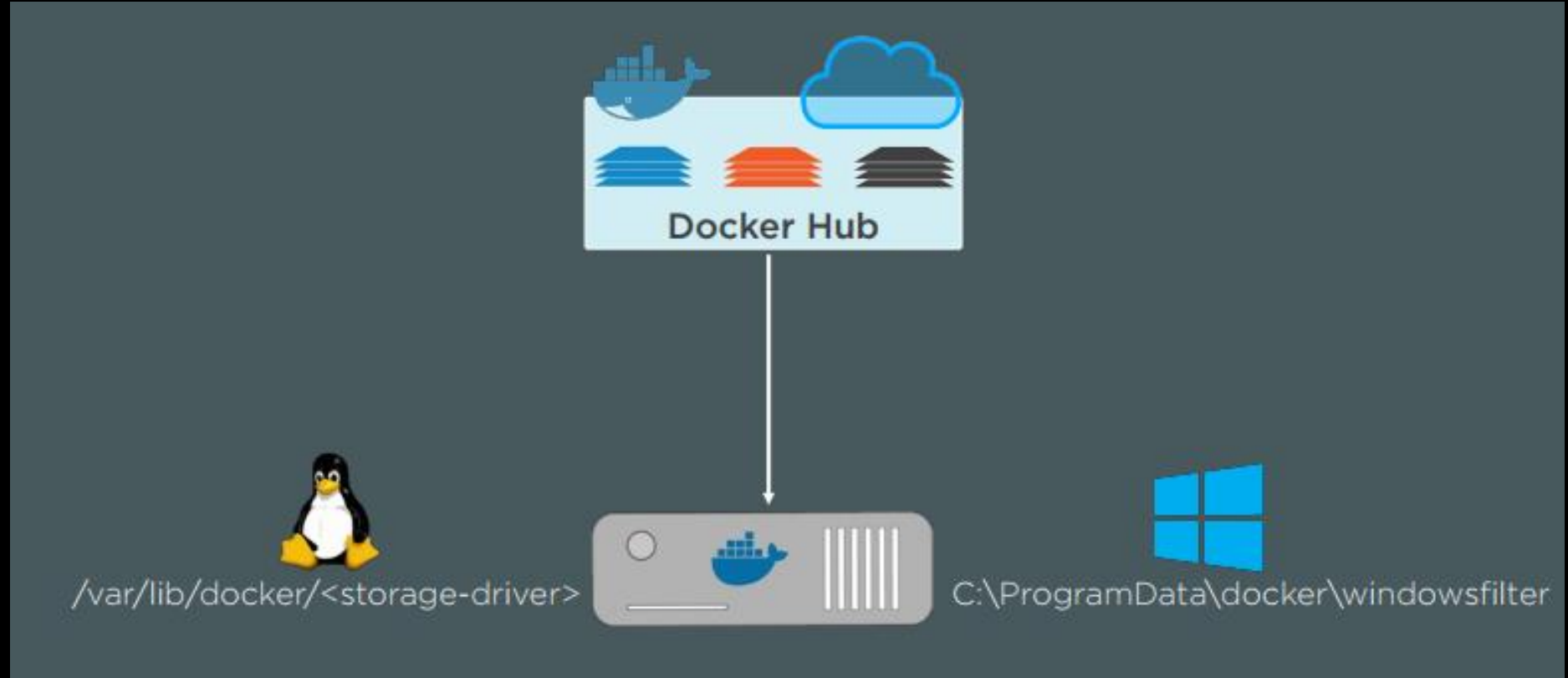
# DOCKER REGISTRIES

- stores Docker images.
- Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default.
- You can even run your own private registry.
- When you use the docker pull or docker run commands, the required images are pulled from your configured registry.
- When you use the docker push command, your image is pushed to your configured registry.





# DOCKER REGISTRIES



# DOCKER REGISTRIES

The screenshot shows the Docker Hub interface with a search for 'redis'. The top navigation bar includes the Docker Hub logo, a search bar with 'redis' entered, and links for Explore, Repositories, Organizations, Get Help, and a user profile for 'shalomlevi'. Below the navigation bar, there are tabs for DOCKER EE, DOCKER CE, CONTAINERS (which is selected), and PLUGINS. The main content area displays search results for 'redis'. On the left, there are filters for Docker Certified, Verified Publisher, and Official Images. The search results show 1 - 25 of 11,675 results for 'redis'. The first result is 'redis' by 'redis', which is an official image. It has over 10 million downloads and 7.7K stars. The description states that Redis is an open source key-value store that functions as a data structure server. Below the description, there are tags for Container, Linux, Windows, x86-64, 386, IBM Z, ARM, ARM 64, PowerPC 64 LE, and Databases. The second result is 'rediscommander/redis-commander' by 'rediscommander', which has over 5 million downloads and 33 stars.

dockerhub

Explore Repositories Organizations Get Help shalomlevi

DOCKER EE DOCKER CE CONTAINERS PLUGINS

Filters 1 - 25 of 11,675 results for **redis**. [Clear search](#) Most Popular

Docker Certified *i*

☐ ☒ Docker Certified

Images

☐ Verified Publisher *i*  
Docker Certified And Verified Publisher Content

☐ Official Images *i*  
Official Images Published By Docker

Categories *i*

**redis**  
Updated an hour ago

Redis is an open source key-value store that functions as a data structure server.

Container Linux Windows x86-64 386 IBM Z ARM ARM 64 PowerPC 64 LE Databases

**rediscommander/redis-commander**  
By [rediscommander](#) • Updated a month ago

10M+ 7.7K  
Downloads Stars

5M+ 33  
Downloads Stars

# DOCKER REGISTRIES

REGISTRY

REPO

IMAGE (TAG)

`docker.io/redis/latest`

`docker.io/nginx/1.13.5`

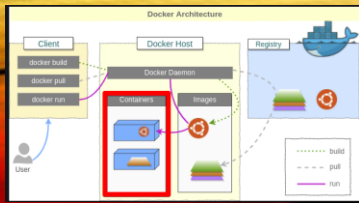
**`docker.io`** / `<repo>` / **`latest`**  
(default) (default)



# IMAGE BEST PRACTICE

- use official images
- images small
- for external images not use latest tag,  
use explicit version





# DOCKER OBJECTS - CONTAINERS

- A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.
- By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.
- A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

# INSTALLING DOCKER

- <https://docs.docker.com/docker-for-windows/install/>

# CHECK FUNCTIONALITY

```
Z:\>docker ps
```

CONTAINER ID	IMAGE	COMMAND

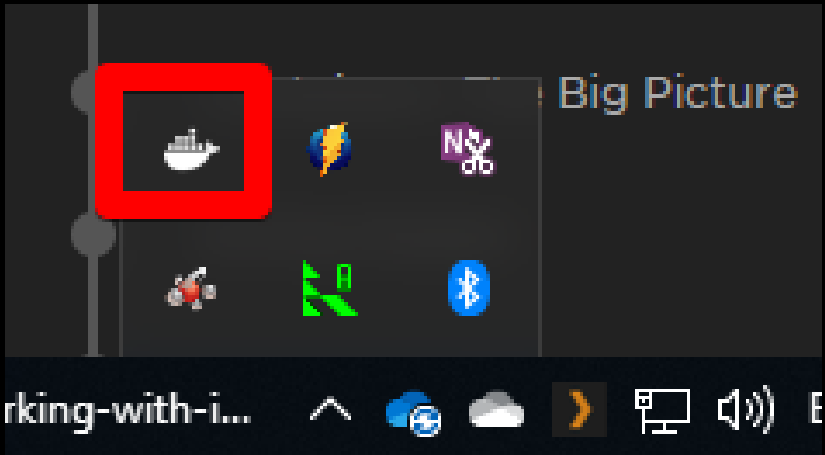
```
Z:\>
```

```
C:\Users\Administrator>docker version
```

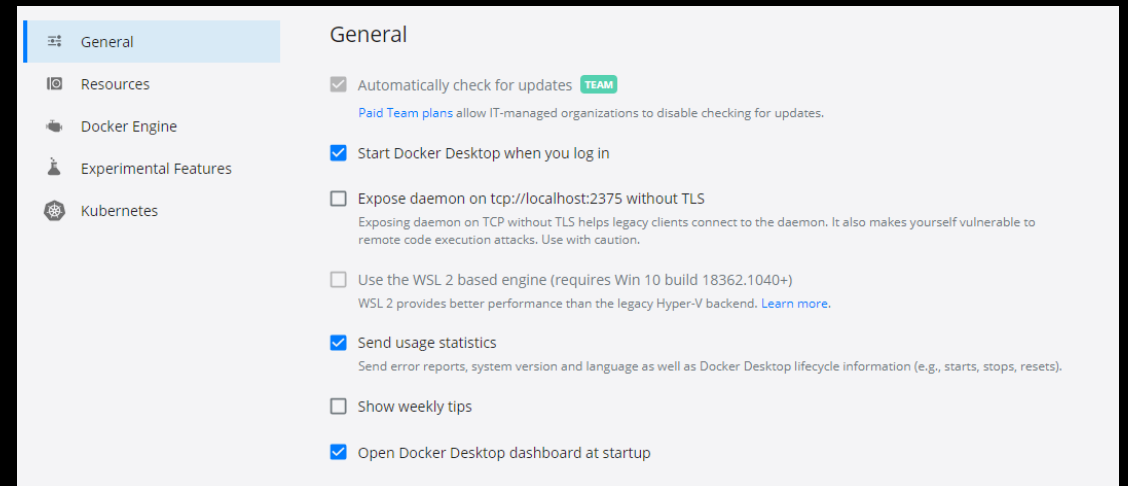
```
Client: Docker Engine - Community
Version:           19.03.5
API version:       1.40
Go version:        go1.12.12
Git commit:        633a0ea
Built:             Wed Nov 13 07:22:37 2019
OS/Arch:           windows/amd64
Experimental:      false

Server: Docker Engine - Community
Engine:
Version:          19.03.5
API version:      1.40 (minimum version 1.12)
Go version:       go1.12.12
```

# DOCKER TOOL





settings





# DOCKER TOOL - SETTINGS

 General


 Resources


ADVANCED


- FILE SHARING

PROXIES

NETWORK

 Docker Engine

 Experimental Features

 Kubernetes

## Resources

### File sharing

These directories (and their subdirectories) can be bind mounted into Docker containers. You can check the [documentation](#) for more details.

C:\logs	⊖
C:\Users\Administrator\.nuget\packages	⊖
C:\Users\Administrator\AppData\Roaming	⊖
C:\Users\Administrator\AppData\Roaming	⊖
C:\Users\Administrator\source\repos\Wel	⊖
C:\Users\Administrator\vsdbg\vs2017u5	⊖
X:\	⊖

# DOCKER TOOL - SETTINGS

☰ General

🖼️ Resources

• ADVANCED

FILE SHARING

PROXIES

NETWORK

🐳 Docker Engine

🧪 Experimental Features

🌀 Kubernetes

Resources Advanced


CPUs: 2


Memory: 4.00 GB

Swap: 1 GB

Disk image size: 64 GB (43.6 GB used)

# DOCKER TOOL - SETTINGS

 General


 Resources


ADVANCED


FILE SHARING

PROXIES

• NETWORK

 Docker Engine

 Experimental Features

 Kubernetes

Resources Network

Configure the way Docker containers interact with the network

Docker subnet  
192.168.65.0/28  
default: 192.168.65.0/28

DNS Server

☐ Manual DNS configuration

DNS  
8.8.8.8

# DOCKER TOOL - SETTINGS



General



Resources



Docker Engine



Experimental Features



Kubernetes

## Kubernetes

v1.19.7



Enable Kubernetes

Start a Kubernetes single-node cluster when starting Docker Desktop.



Deploy Docker Stacks to Kubernetes by default

Make Kubernetes the default orchestrator for "docker stack" commands (changes "~/.docker/config.json")



Show system containers (advanced)

Show Kubernetes internal containers when using Docker commands.

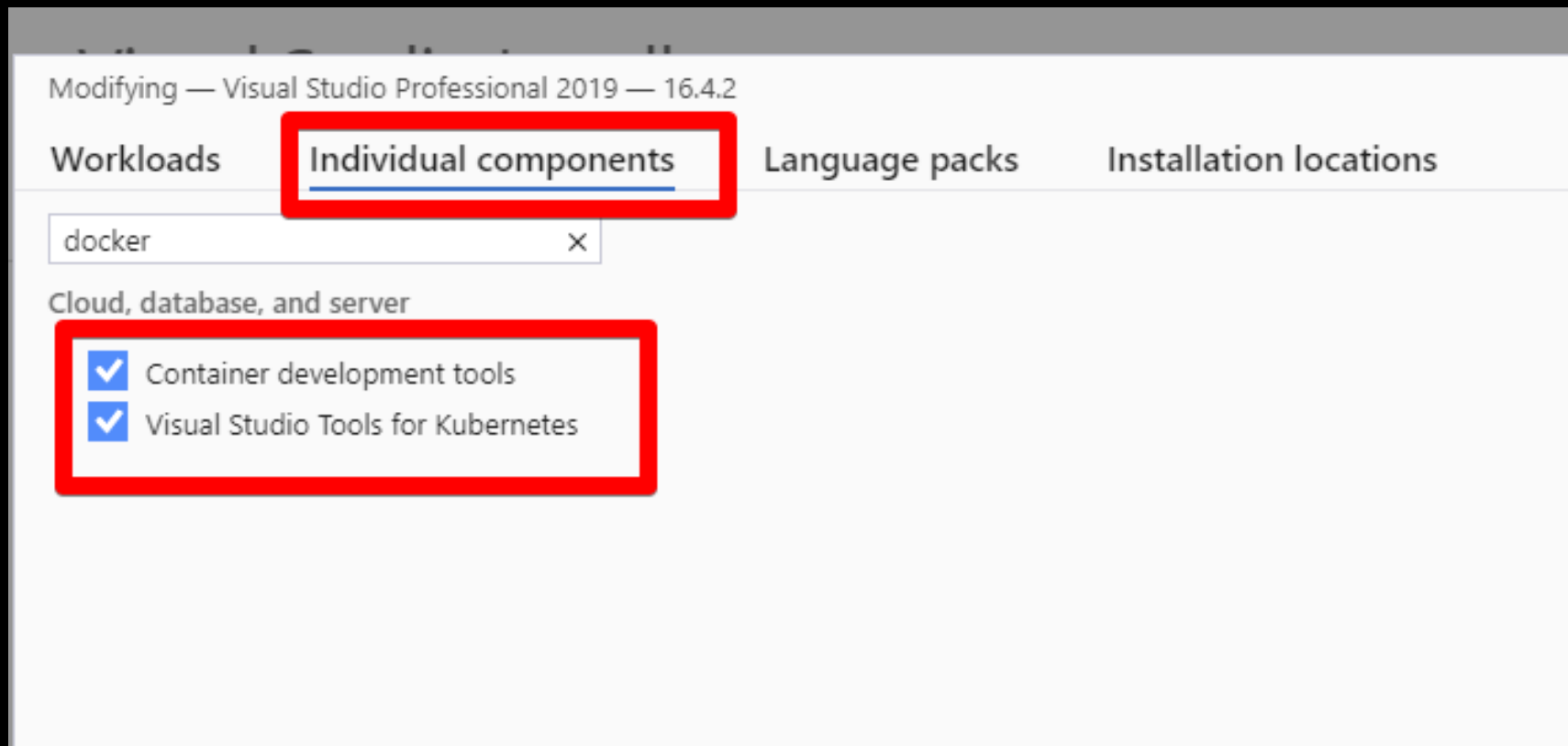
Reset Kubernetes Cluster

All stacks and Kubernetes resources will be deleted.



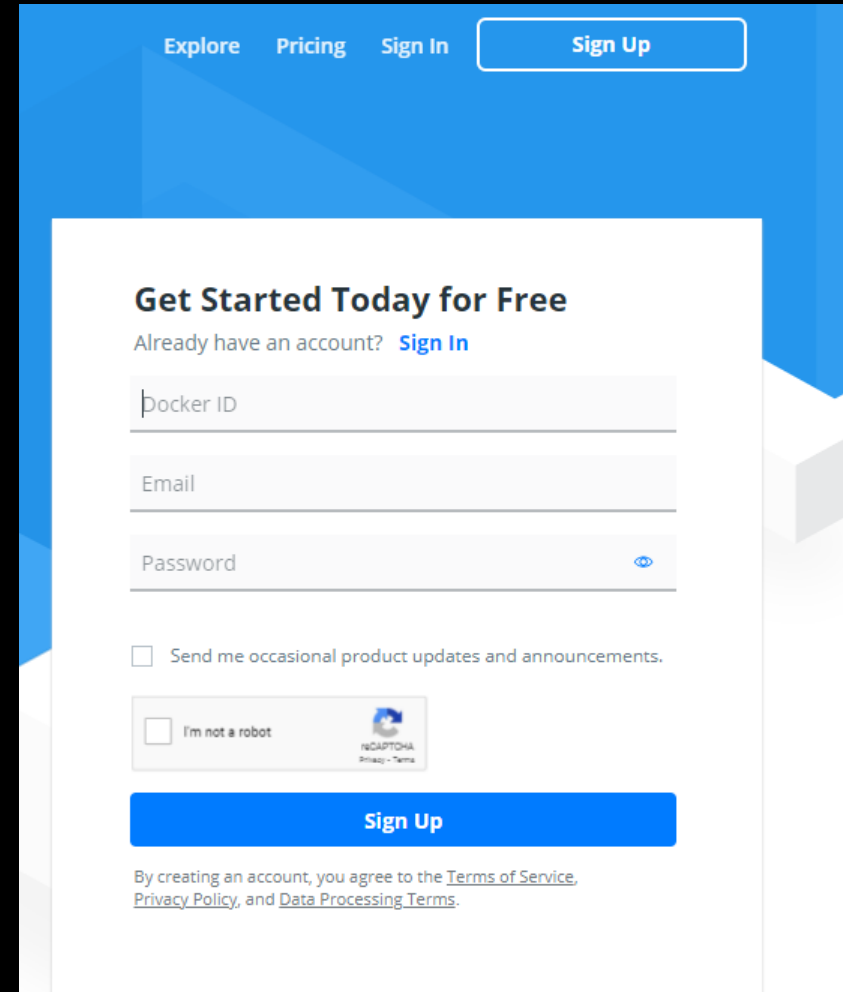
# DOCKER TOOL - VISUAL STUDIO

Open Microsoft visual studio installer



# REGISTRY - DOCKER HUB

- <https://hub.docker.com/>



The screenshot shows the Docker Hub sign-up page. At the top, there is a navigation bar with links for 'Explore', 'Pricing', 'Sign In', and a 'Sign Up' button. The main content area is titled 'Get Started Today for Free'. Below this, there is a link for 'Already have an account? Sign In'. The sign-up form consists of three input fields: 'Docker ID', 'Email', and 'Password'. Below the 'Password' field, there is a checkbox for 'Send me occasional product updates and announcements.' and a checkbox for 'I'm not a robot' with a CAPTCHA icon. At the bottom of the form is a large blue 'Sign Up' button. Below the button, there is a disclaimer: 'By creating an account, you agree to the Terms of Service, Privacy Policy, and Data Processing Terms.'

Explore Pricing Sign In Sign Up

## Get Started Today for Free


Already have an account? [Sign In](#)

Docker ID

Email

Password

☐ Send me occasional product updates and announcements.

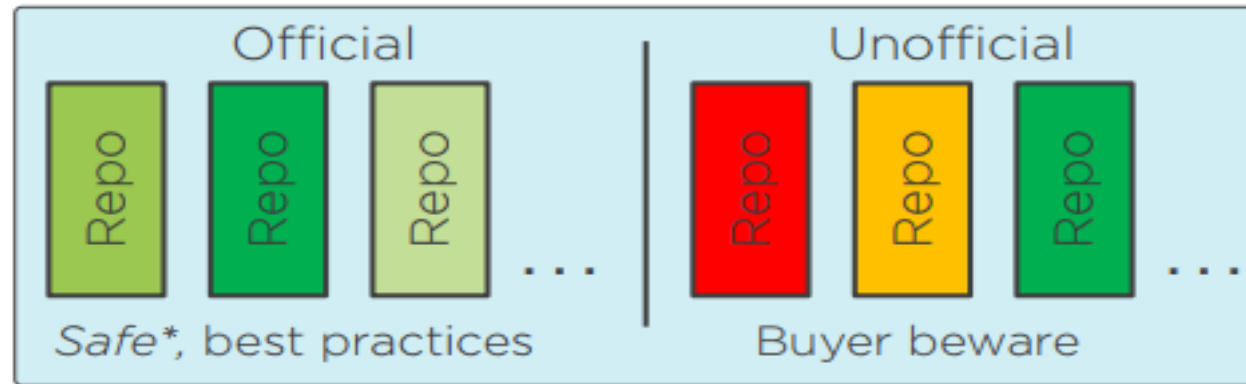
☐ I'm not a robot 

[Terms of Service](#) [Privacy Policy](#) [Data Processing Terms](#)

Sign Up

By creating an account, you agree to the [Terms of Service](#), [Privacy Policy](#), and [Data Processing Terms](#).

# REGISTRY - DOCKER HUB



Docker Hub

```
$ docker image push
```

```
$ docker image pull
```

```
$ docker image inspect
```

```
$ docker image rm
```

# BASIC DOCKER COMMANDS

- **Pulling Docker Image**

```
$ docker pull mcr.microsoft.com/dotnet/core/samples
```

- 

- **Listing out Docker Images**

```
$ docker image ls
```

- 

- **Running Docker Containers**

```
$ docker run -itd --rm -p 8000:80 --name aspnetcore_sample mcr.microsoft.com/dotnet/core/samples:aspnetapp
```

```
$ docker run --rm -it mcr.microsoft.com/dotnet/core/runtime:3.1 dotnet --list-runtimes
```

```
$ docker run --rm -it mcr.microsoft.com/dotnet/core/runtime:3.1 sh
```

- **Run shell on running container**

```
docker container exec -it 6a25541cb9d8 sh
```

- 

- **Stopping the container**

```
$ docker container stop mcr.microsoft.com/dotnet/core/samples (or <container id>)
```

- 

- **remove container**

```
$ docker rm -f mcr.microsoft.com/dotnet/core/samples (or <container id>)
```

- **Docker image build**

```
docker image build -t shalom-test .
```

- **Docker containers running**

```
docker ps
```

# DOCKER RUN

- **Running Docker Containers**
- `docker run [OPTIONS] IMAGE [COMMAND] [ARG...]`

```
$ docker run -itd --rm -p 8000:80 --name aspnetcore_sample  
mcr.microsoft.com/dotnet/core/samples:aspnetapp
```

-l	--interactive	Keep STDIN open even if not attached
-t	--tty	Allocate a pseudo-TTY
-d	--detach	Run container in background and print container ID



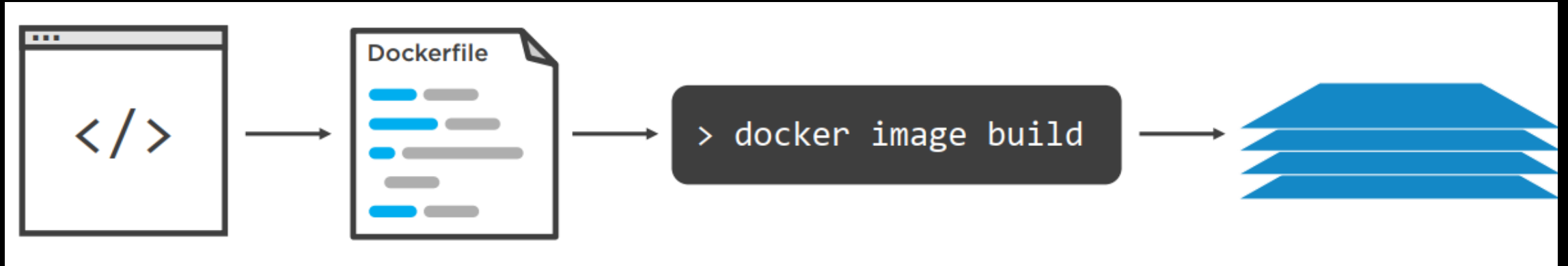
# תרגיל 1

1. צור משתמש ב docker hub
2. התקן סביבת docker על מחשב המעבדה.
3. הפעל תצורת Linux , ניתן להיעזר בפקודה docker info כדי לבדוק באיזה תצורה הסביבה עובדת
4. בדוק שסביבת ה docker תקינה
5. חפש ב docker hub את ה image "alpine"
6. הרץ את alpine כ container שיפעיל את הפקודה "ls /bin", מה קיבלת?
7. בדוק איזה images מותקנים.
8. הפעל את הפקודה הבאה:

```
docker run --itd -- rm -p ppatenpsa:selpmas/eroc/tentod/moc.ftosorcim.rcm elpmas_eroc tenpsa eman --8000 : 80
```

9. בדוק את היישום בדפדפן בפורט 8000
  10. בדוק איזה container רצים כרגע
  11. מחק את ה container.
  12. בנה image מהפקודה הבאה:
- ```
docker image build -t shalom-node https://github.com/shaloml/node-docker-sample.git
```
12. הרץ את ה image שנוצר בפורט 888
  13. בדוק את היישום בדפדפן בפורט 888

# DOCKER FILE



## Name of file: Dockerfile

## DOCKER FILE

|                                                  |   |                                                     |
|--------------------------------------------------|---|-----------------------------------------------------|
| Base image <b>layer-1</b>                        | → | <code>FROM alpine</code>                            |
| Good practice to list maintainer                 | → | <code>LABEL maintainer="shaloml@gmail.com"</code>   |
| execute command and create <b>layer-2</b>        | → | <code>RUN apk add --update nodejs nodejs-npm</code> |
| copy code into image as new <b>layer-3</b>       | → | <code>COPY . /src</code>                            |
| Some instructions add metadata instead of layers | → | <code>WORKDIR /src</code>                           |
| execute command and create <b>layer-4</b>        | → | <code>RUN npm install</code>                        |
| Expose port from image                           | → | <code>EXPOSE 8080</code>                            |
| default app for image/container                  | → | <code>ENTRYPOINT ["node", "./app.js"]</code>        |

# \$docker image history [image-name] LAYER

| IMAGE        | CREATED       | CREATED BY                                    | SIZE   |
|--------------|---------------|-----------------------------------------------|--------|
| ad50ba62c029 | 3 minutes ago | /bin/sh -c #(nop) ENTRYPOINT ["node" "./a...  | 0B     |
| c55fbce13408 | 3 minutes ago | /bin/sh -c #(nop) EXPOSE 8080/tcp             | 0B     |
| 3e958d9a9321 | 3 minutes ago | /bin/sh -c npm install                        | 18.4MB |
| 91d611eaa0b9 | 3 minutes ago | /bin/sh -c #(nop) WORKDIR /src                | 0B     |
| d1e85b579cc6 | 3 minutes ago | /bin/sh -c #(nop) COPY dir:441bd548f7cd704... | 21.4kB |
| 89188337f5ca | 3 minutes ago | /bin/sh -c apk add --update nodejs nodejs-npm | 32.9MB |
| ff978f010983 | 3 minutes ago | /bin/sh -c #(nop) LABEL maintainer=nigelp...  | 0B     |
| 76da55c8019d | 13 days ago   | /bin/sh -c #(nop) CMD ["/bin/sh"]             | 0B     |
| <missing>    | 13 days ago   | /bin/sh -c #(nop) ADD file:4583e12bf5caec4... | 3.97MB |

## \$docker image inspect [image-name]

```
"RootFS": {
  "Type": "layers",
  "Layers": [
    "sha256:5bef08742407efd622d243692b79ba0055383bbce12900324f75e56f589aedb0",
    "sha256:b698ddd3252db48462ee329955afe3bc410d0020fff1fc8587dd073f62c3dcf0",
    "sha256:d11ba22daaa3952cd43f56c16f2d0fe4f90cbe62b3e2caca92ef856538b6cce9",
    "sha256:f0a7ce91571fbc9631b9cd9bef8548861acaee77f447d87b21de18861f96862d"
  ]
}
```

# MULTI STAGE BUILDS

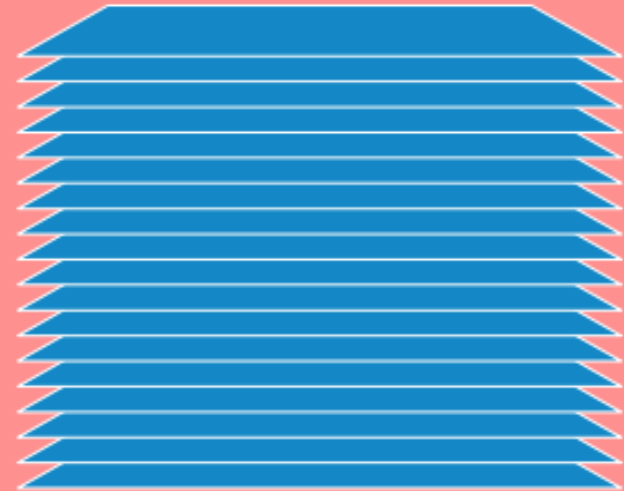


**Small Images**



Minimal OS  
and packages

**Bloated Images**



Too much OS  
Too many packages



# MULTI STAGE BUILDS



[Stage 0] `FROM mcr.microsoft.com/dotnet/core/aspnet:3.1-buster-slim AS base`  
`WORKDIR /app`  
`EXPOSE 80`  
`EXPOSE 443`



[Stage 1] `FROM mcr.microsoft.com/dotnet/core/sdk:3.1-buster AS build`  
`WORKDIR /src`  
`COPY ["WebApplication6.csproj", ""]`  
`RUN dotnet restore "./WebApplication6.csproj"`  
`COPY . .`  
`WORKDIR "/src/."`  
`RUN dotnet build "WebApplication6.csproj" -c Release -o /app/build`



[Stage 2] `FROM build AS publish`  
`RUN dotnet publish "WebApplication6.csproj" -c Release -o /app/publish`



[Stage 3] `FROM base AS final`  
`WORKDIR /app`  
`COPY --from=publish /app/publish .`  
`ENTRYPOINT ["dotnet", "WebApplication6.dll"]`

# MULTI STAGE BUILDS

```
C:\Users\Administrator\Source\Repos\WebApplication6>docker images
```

| REPOSITORY                            | TAG             | IMAGE ID     | CREATED       | SIZE   |
|---------------------------------------|-----------------|--------------|---------------|--------|
| shalom-net1                           | latest          | 19243d266418 | 7 minutes ago | 208MB  |
| <none>                                | <none>          | c23a5796ea9a | 7 minutes ago | 694MB  |
| alpine                                | latest          | e7d92cdc71fe | 5 days ago    | 5.59MB |
| mcr.microsoft.com/dotnet/core/samples | aspnetapp       | dedaa3c9ce5f | 8 days ago    | 212MB  |
| mcr.microsoft.com/dotnet/core/sdk     | 3.1-buster      | 2fe8fe202baf | 9 days ago    | 689MB  |
| mcr.microsoft.com/dotnet/core/aspnet  | 3.1-buster-slim | 5b704ff3cb6b | 9 days ago    | 207MB  |
| mcr.microsoft.com/dotnet/core/runtime | 3.1             | a708cda756ab | 9 days ago    | 190MB  |

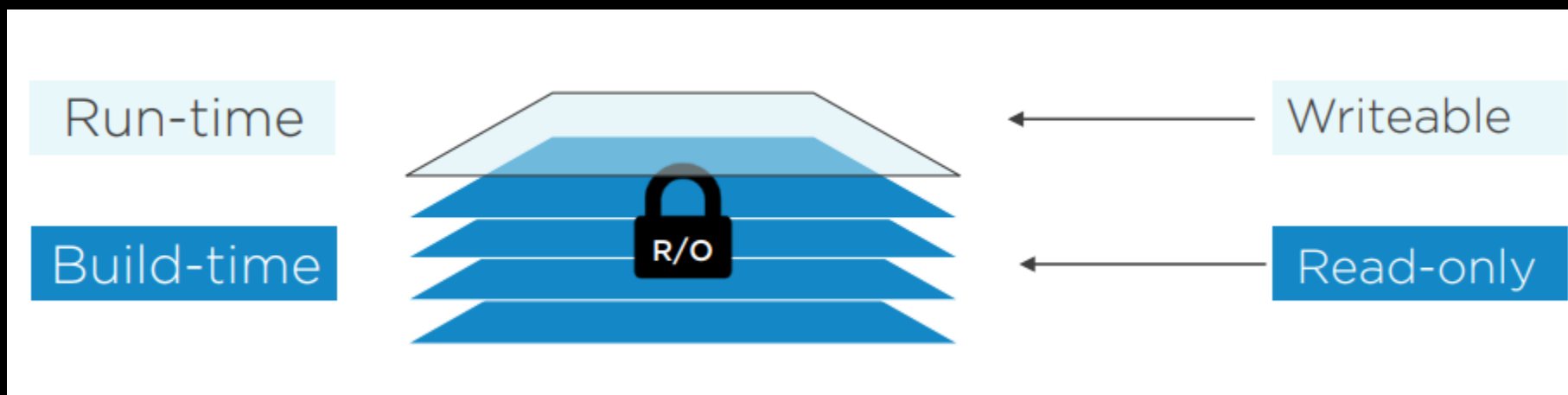
# CONTAINER



Build-time



Run-time



# CONTAINER BASIC COMMAND

- Show container running

**\$Docker ps / docker container ls**

- Show all containers

**\$Docker ps -a**

- Delete container

**\$Docker rm -f [container Id / container name]**

- Delete container Start

**\$Docker container [container Id / container name] start**

- container exec

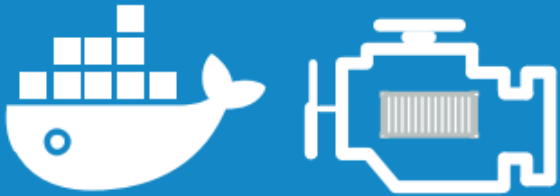
**\$Docker container exec -it [container Id / container name] sh**

- Delete all ( only on power shell)

**\$docker container rm \$(docker container ls -aq) -f**

# LOGGING

## Engine/daemon



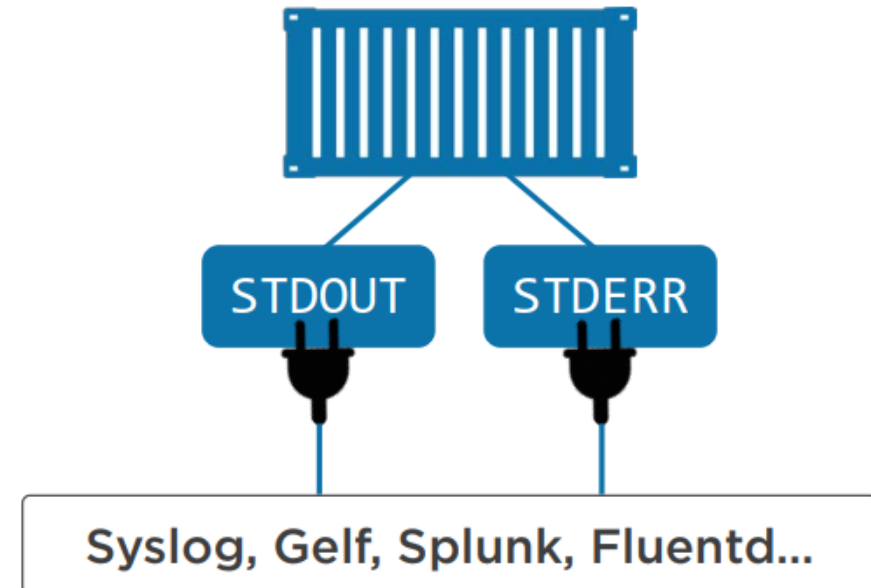
### Linux

- systemd:  
`journalctl -u docker.service`
- Non-systemd:  
Try `/var/log/messages`

### Windows:

`~/AppData/Local/Docker`

## Container/App



Set default logging driver in `daemon.json`

Override per-container with  
`--log-driver --log-opts`

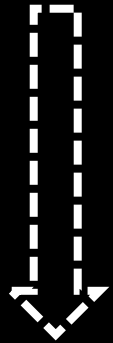
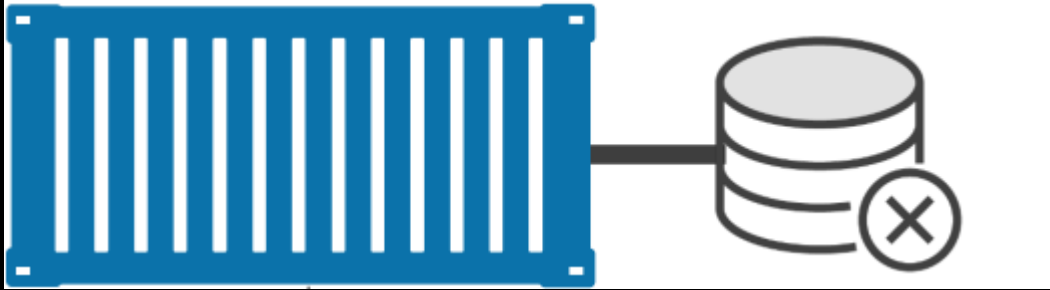
Inspect logs with `docker logs <container>`  
*- Doesn't work with all drivers*



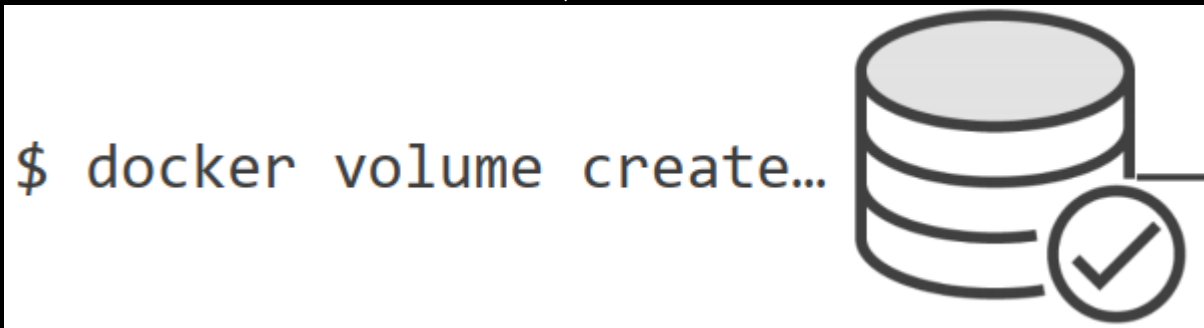
## תרגיל 2

1. צור פרויקט מסוג `api.net core`
1. צור את הפרויקט עם תמיכה ב `docker`
2. פתח את `Dockerfile` ונסה להבין את השורות.
3. הרץ את הפרויקט דרך `visual studio` לוודא שהקוד תקין
5. פתח `CLI`
  1. בנה `image` בשם `my-first-container`
  6. בדוק שנוצר `image`
  7. הראה שכבות `image` באמצעות `history`
  8. הפעל `inspect` על ה `image`, נסה להבין מה אתה רואה.

Not persistent



persistent



Out side of container



# VOLUMES

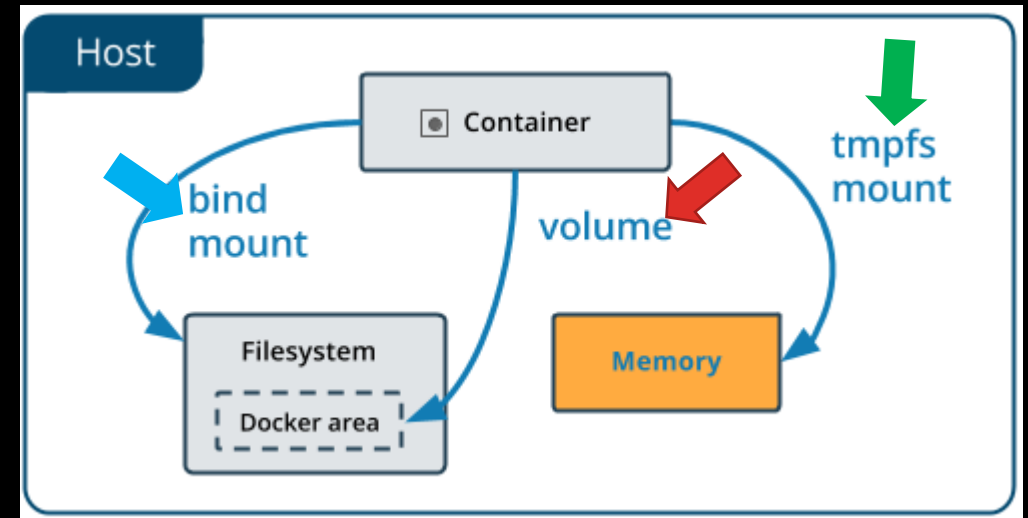


# USE VOLUMES

- 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.

# VOLUMES

- An easy way to visualize the difference among **volumes**, **bind mounts**, and **tmpfs** mounts is to think about where the data lives on the Docker host.
- **Volumes** are stored in a part of the host filesystem which is managed by Docker (`/var/lib/docker/volumes/` on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.
- **Bind mounts** may be stored anywhere on the host system
- **tmpfs** mounts are stored in the host system's memory only



# VOLUMES

## RUN

```
docker run -dti --name alpine1 --mount target=/app alpine ash
```

## INSPECT

```
docker inspect alpine1
```

```
{,
  "Mounts": [
    {
      "Type": "volume",
      "Name": "e2c79e12b3f8b90888688da603cca4c2297ee96f2b914a601378e5d944f17214",
      "Source": "/var/lib/docker/volumes/e2c79e12b3f8b90888688da603cca4c2297ee96f2b914a601378e5d944f17214/_data",
      "Destination": "/app",
      "Driver": "local",
      "Mode": "z",
      "RW": true,
      "Propagation": ""
    }
  ],
}
```

## STOP AND DELETE CONTAINER

```
docker stop alpine1 && docker rm alpine1
```



# VOLUMES

Creating a VOLUME managed by docker FS and share it with multiple containers

## RUN

```
docker volume create fs_shared
```

## LIST VOLUMES

```
docker volume ls
```

```
local          fs_shared
```

## RUN AND MOUNT

```
docker run --rm -tdi --name alpine1 --mount source=fs_shared,target=/app alpine ash
```

```
docker run --rm -tdi --name alpine2 --mount source=fs_shared,target=/app alpine ash
```

```
docker run --rm -tdi --name alpine3 --mount source=fs_shared,target=/app alpine ash
```

<https://docs.docker.com/storage/volumes/>

# VOLUMES

- The `-v` and `--mount` examples below produce the same result. You can't run them both unless you remove the devtest container and the myvol2 volume after running the first one.

`-v`

```
docker run --rm -tdi -v "$(pwd)"/source:/app [image] [CMD]  
docker run -d --name devtest -v myvol2:/app nginx:latest
```

`--mount`

```
docker run --rm -tdi -v "$(pwd)"/source:/app [image] [CMD]  
docker run -d --name devtest --mount source=myvol2,target=/app \ nginx:latest
```

# VOLUMES

WINDOWS MAP VOLUME BIND

**-v**

```
docker run --rm -tdi -v C:/folder/name:/data [image] [CMD]
```

**example**

```
docker run --rm -it --name alpine1-windows -v c:/temp:/app alpine sh
```

# VOLUMES

## BIND MOUNTS USING -V OR --MOUNT ?

- all options for volumes are available for both --mount and -v flags.
- When using volumes with services, only --mount is supported.
- **What should I use?**
  - What you want to use comes mostly down to either preference or your management. If you want to keep everything in the "docker area" (/var/lib/docker) you can use volumes. If you want to keep your own directory-structure, you can use binds.
  - **Docker recommends the use of volumes over the use of binds**, as volumes are created and managed by docker and binds have a lot more potential of failure (also due to layer 8 problems).
  - If you use binds and want to transfer your containers/applications on another host, you have to rebuild your directory-structure, where as volumes are more uniform on every host.

# MANAGE VOLUMES

`$docker volume [commands]`

`Ls`

List volumes

`Create`

`docker volume create [volume name]`

`Info`

`docker volume inspect [volume name]`

`Remove`

`Docker volume rm [volume name]`

On Linux volume store: `/var/lib/docker/volumes`

# תרגיל 3

1. ייצר volume חדש בשם my\_shared
2. בצע ls לכל ה volumes , בדוק שנוצר my\_shared
3. הרץ שני container מסוג alpine עם CLI ( הוספת sh בסוף הפקודה )  
- test-vol1 ו- test-vol2  
קשר את my\_shared ל /app לכל אחד מה containers
4. ב test-vol1  
- כנס לתיקיית app באמצעות הפקודה: cd app  
- הרץ את הפקודה: Echo hello > test.txt
5. בדוק שנוצר קובץ באמצעות הפקודה ls
6. עבור ל test-vol2  
- כנס לתיקיית app באמצעות הפקודה: cd app  
- בדוק שהקובץ קיים גם /app באמצעות הפקודה: ls
7. הרץ container חדש מסוג alpine עם CLI, מפה כונן c:\temp ל /app
8. ייצר קובץ בתיקיית ה WINDOWS ובדוק שהקובץ קיים ב container.