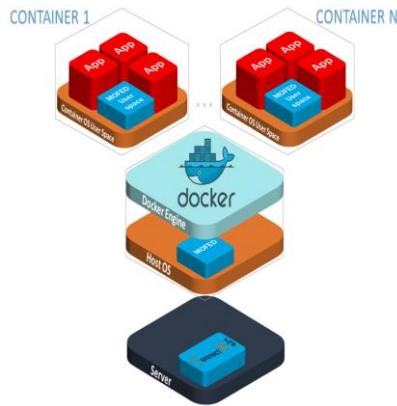


Docker and Containerization Basics

Package App + Dependencies +
Configurations as Containers



Technical Trainers

SoftUni Team



SoftUni



Software University

<https://about.softuni.bg>

Have a Question?



sli.do

#Dev-Ops

Table of Contents

1. Containerization
2. Docker
3. Docker CLI
4. File System and Volume





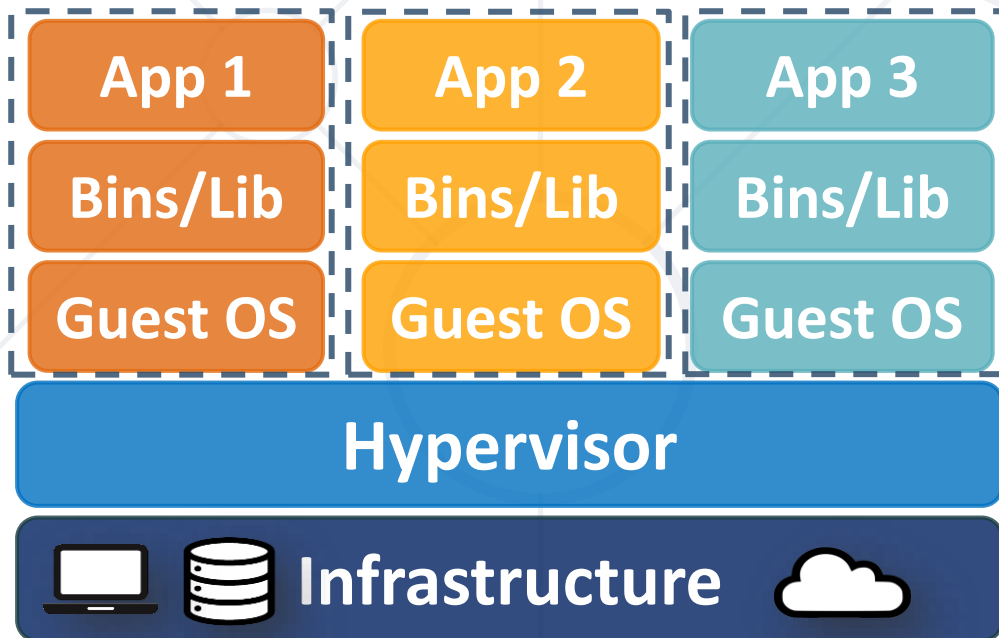
Containerization

Overview, VMs VS Containers, Advantages

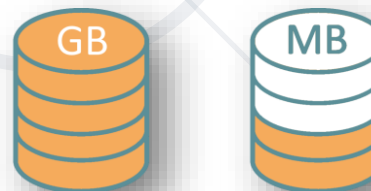
- **Containerization** == approach in which an **app** or **service** is packaged as a **container**
- **Image** == read-only template that contains a set of instructions for creating a container
 - It contains software, packaged with its dependencies and configuration
 - Designed to run in a virtual environment
- **Container** == a runnable instance of an image

VMs vs Containers

- **VMs** virtualize the hardware
- Complete isolation
- Complete OS installation. Requires more resources



Utilization

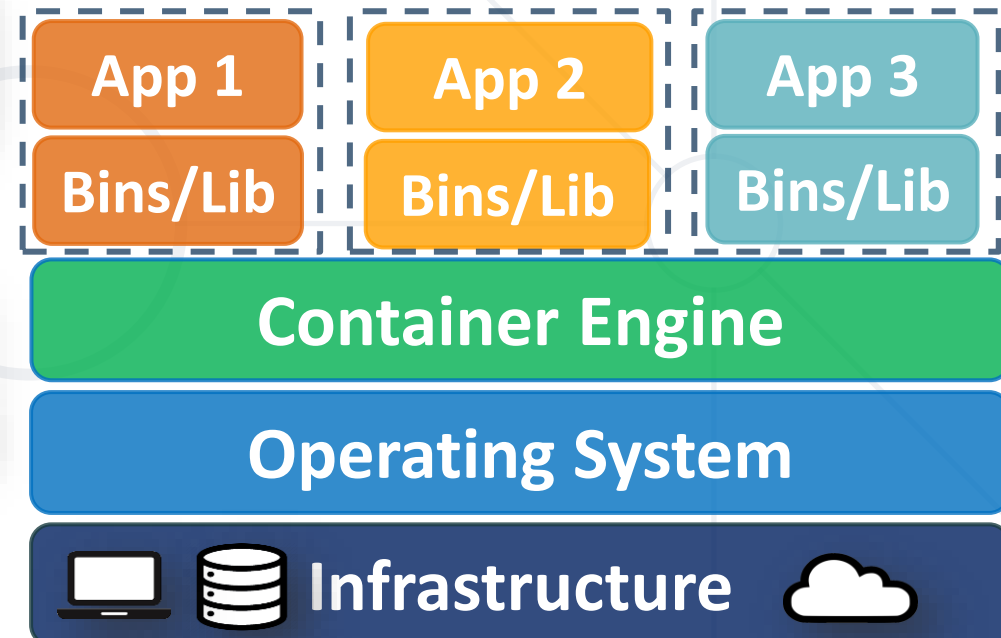


Size



Boot Up Time

- **Containers** virtualize the OS
- Lightweight isolation
- Shared kernel. Requires fewer resources



- Easily **deploy across environments** with little or no modification
- **Immutability**
 - Once a container is created, it **doesn't** change
 - To make a change, a new container must be created
 - Ensures **consistency** across **different environments**
- **Portability**
 - Depend of container runtime, not underlying infrastructure
 - Run on any machine that supports the container runtime

- A containerized app can be **tested** and **deployed** as a **unit** to the host OS
- **Resource-efficient**
 - Share the same OS kernel and isolate applications from each other
- **Scalability**
 - Can be easily scaled up or down
 - Orchestrated by special tools
 - More on that later



Docker

Docker Images, Containers, Software Development

Docker

- Docker == lightweight, open-source, secure containerization platform
- It simplifies building, shipping and running applications
 - On different environments
- Runs natively on Linux or Windows servers
- Runs on Windows or Mac development machines
- Relies on images and containers



- **Docker image** == blueprint for a container
 - A **read-only template**, used to create containers
 - If you want to change something, you should create a new image
 - Holds app/service/other software
 - **Framework, dependencies** and **code** are "described" here
- **Docker registry** == a repository for images

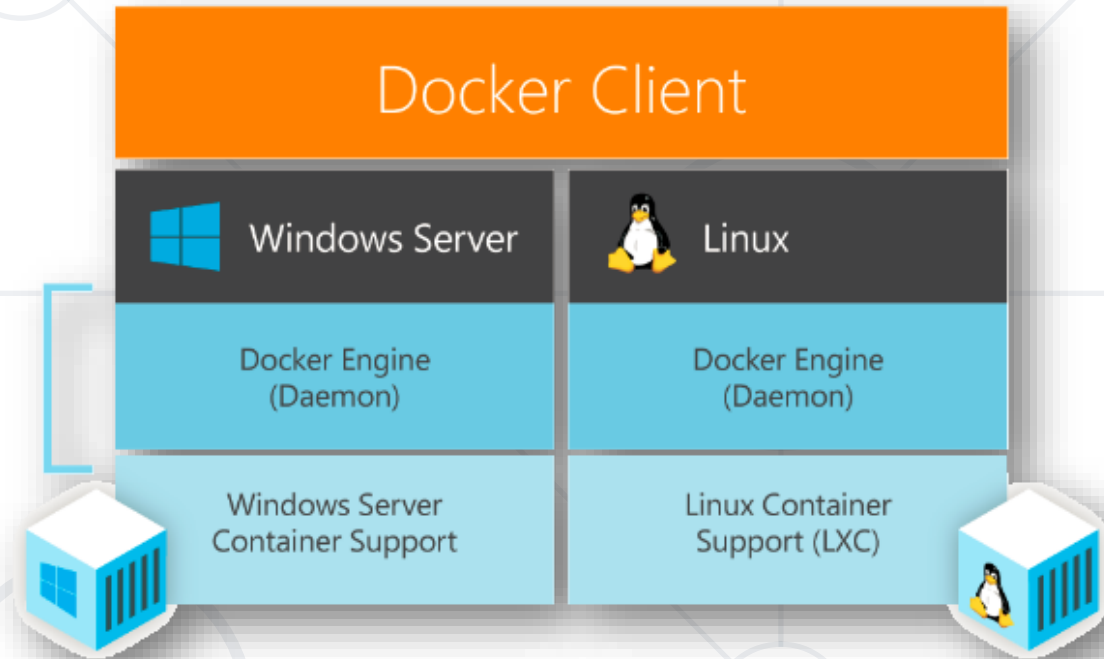
- Built **from the image**
 - Images become containers at **runtime**
- It is the actual **running environment** for your app
- **Isolated** and **secured**
- It can be started/stopped/deleted
- Different app components may reside in separate containers
 - Database, back-end, front-end, caching, messaging, etc.

Docker Desktop

- Out-of-the-box containerization software
- Runs on Windows or Mac development machines
- Includes **Docker Engine**, **CLI** and **Kubernetes**
- Complete Docker **development environment**
- Containerize any application
 - Build
 - Share
 - Run



- On Windows
 - Ability to switch between **Linux** and **Windows Server environments**
 - Typically runs Linux containers through **WSL2** technology (Windows Subsystem for Linux)
 - <https://docs.docker.com/desktop/install/windows-install>
- There are third-party solutions for Linux – DockStation, CairoDock, and more...



Docker Hub

- Docker Hub == cloud-based **image repository** (registry)
- Used for easy **finding** and **sharing images**
- Supports **public and private repositories**
- Automated builds and webhooks
- For every tool we use in Docker, it is recommended that we **read its documentation** first
 - As sometimes we need to perform configurations to work with the tool

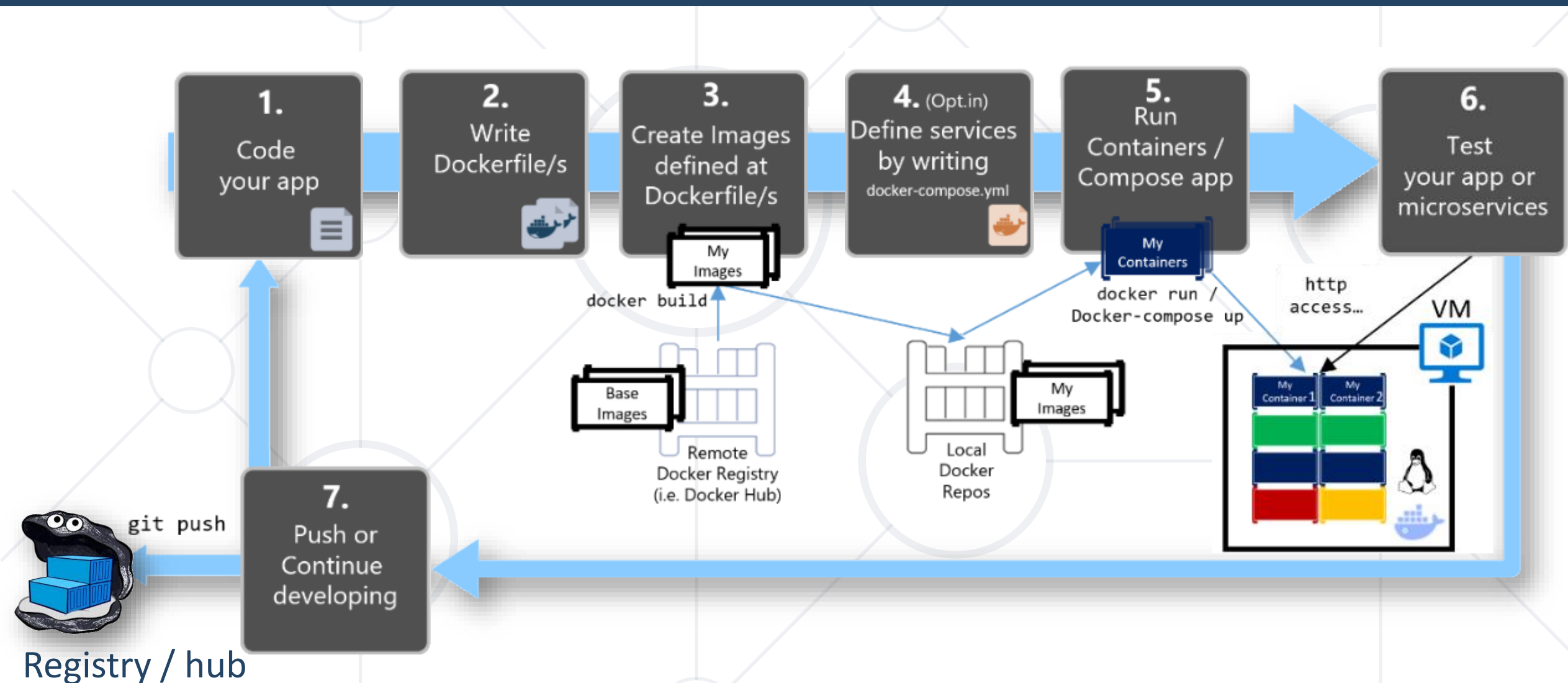


Docker Compose

- Some apps combine multiple components
 - e.g., WordPress requires Linux + NGINX + PHP + MySQL
 - Each component may run in a separate Docker container
- To run **multiple connected containers**, we use **Docker Compose**



Development Workflow for Docker Apps






Docker CLI

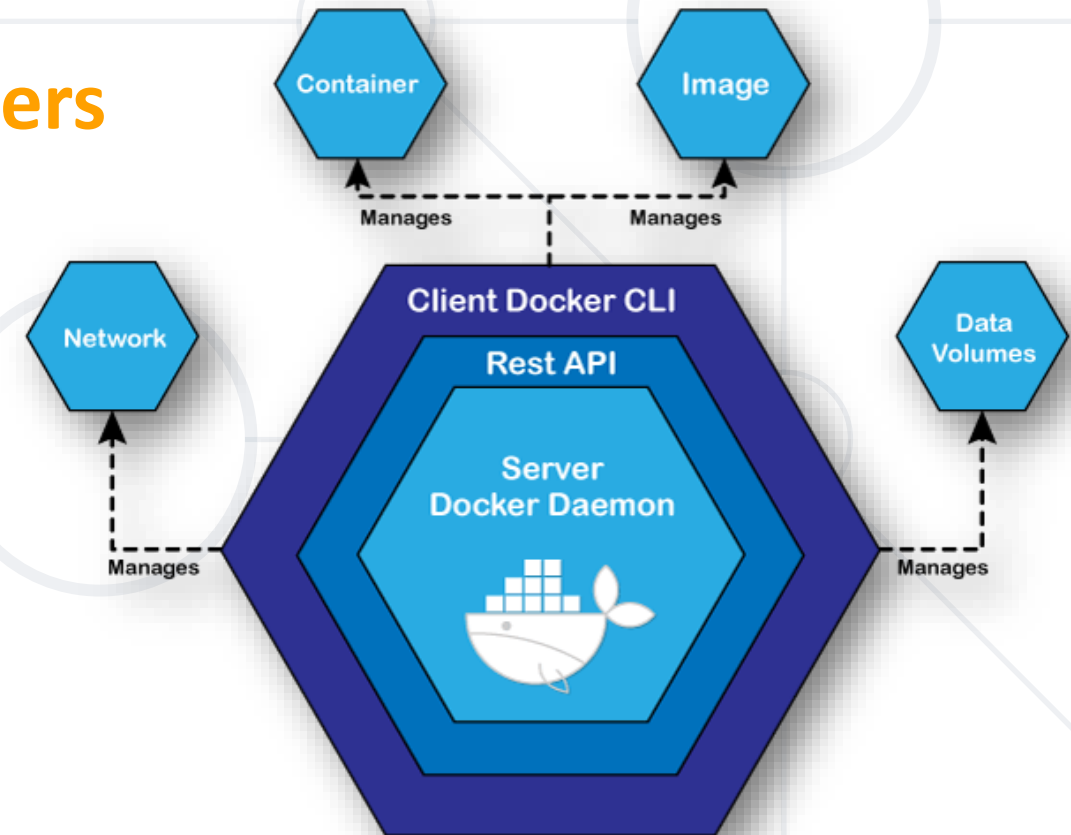
Command Line Tool to Talk to the Docker Daemon

Docker CLI

- **Docker CLI** allows working with the **Docker Engine**
 - Build and manage **images**
 - Run and manage **containers**
- Example commands



```
docker pull [image]
docker run [image]
docker images
docker ps
docker logs [container]
```





Live Demo

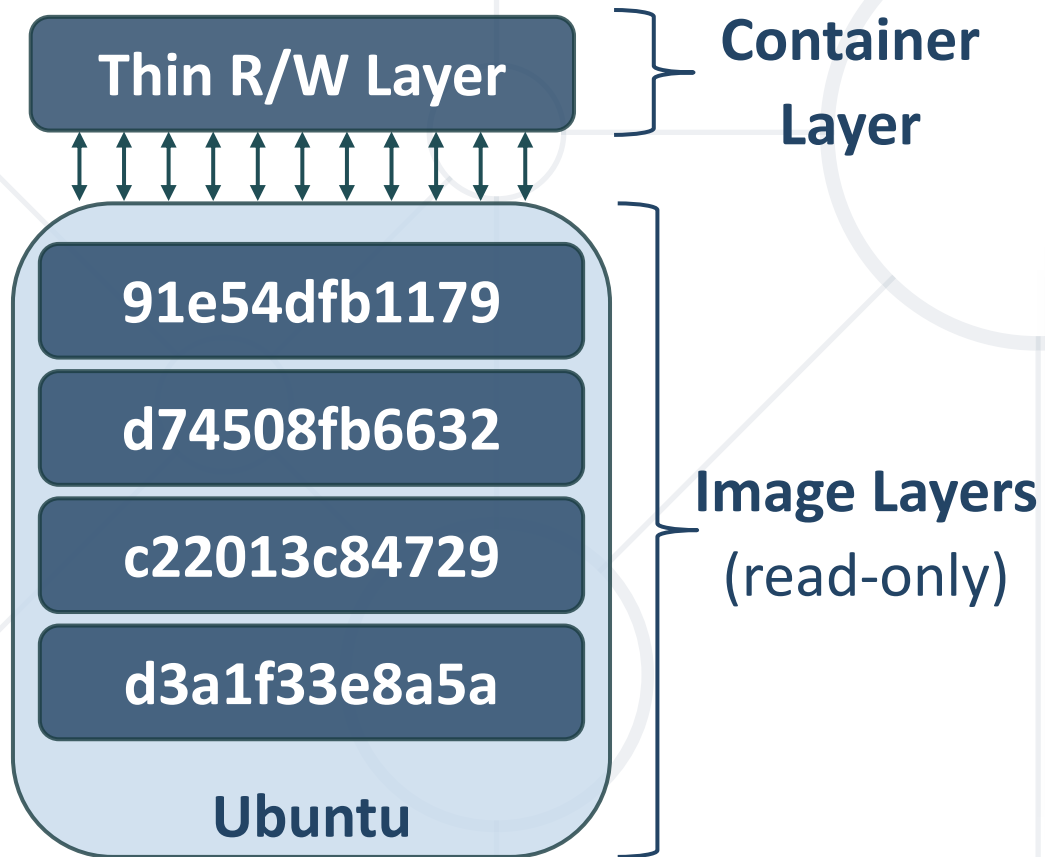
NGINX Server Container



File System and Volume

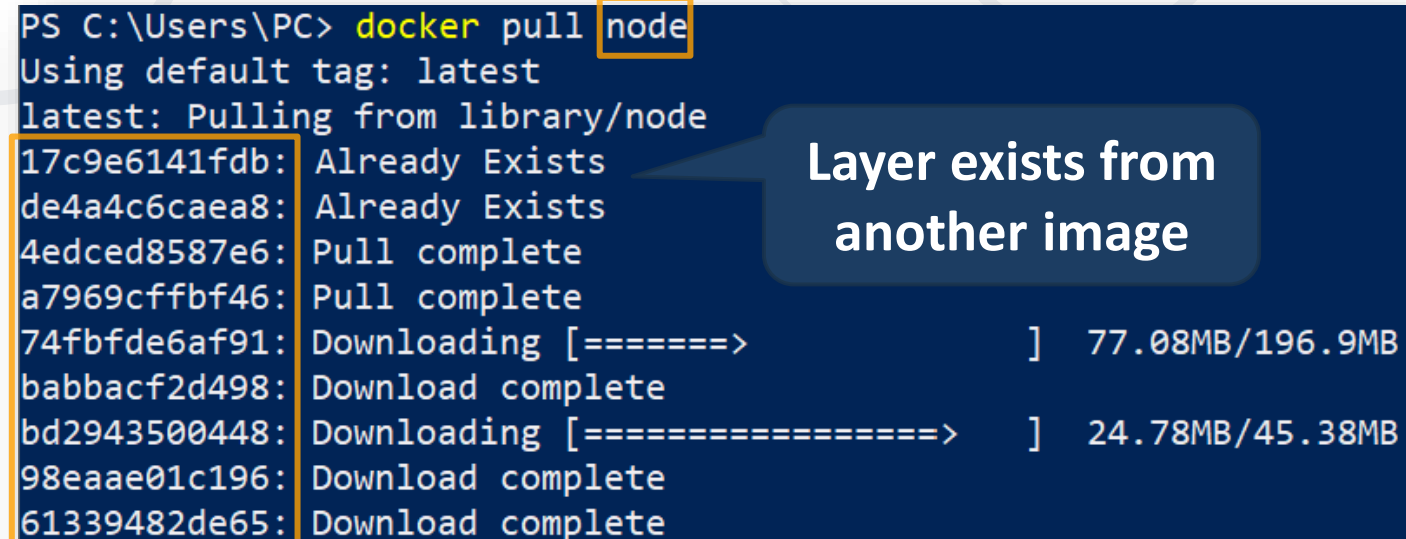
Data in Docker Containers

- Each **image** has **file system layers**, which are read-only and isolated



Container, based on Ubuntu

- Image layers are **reused** in different images

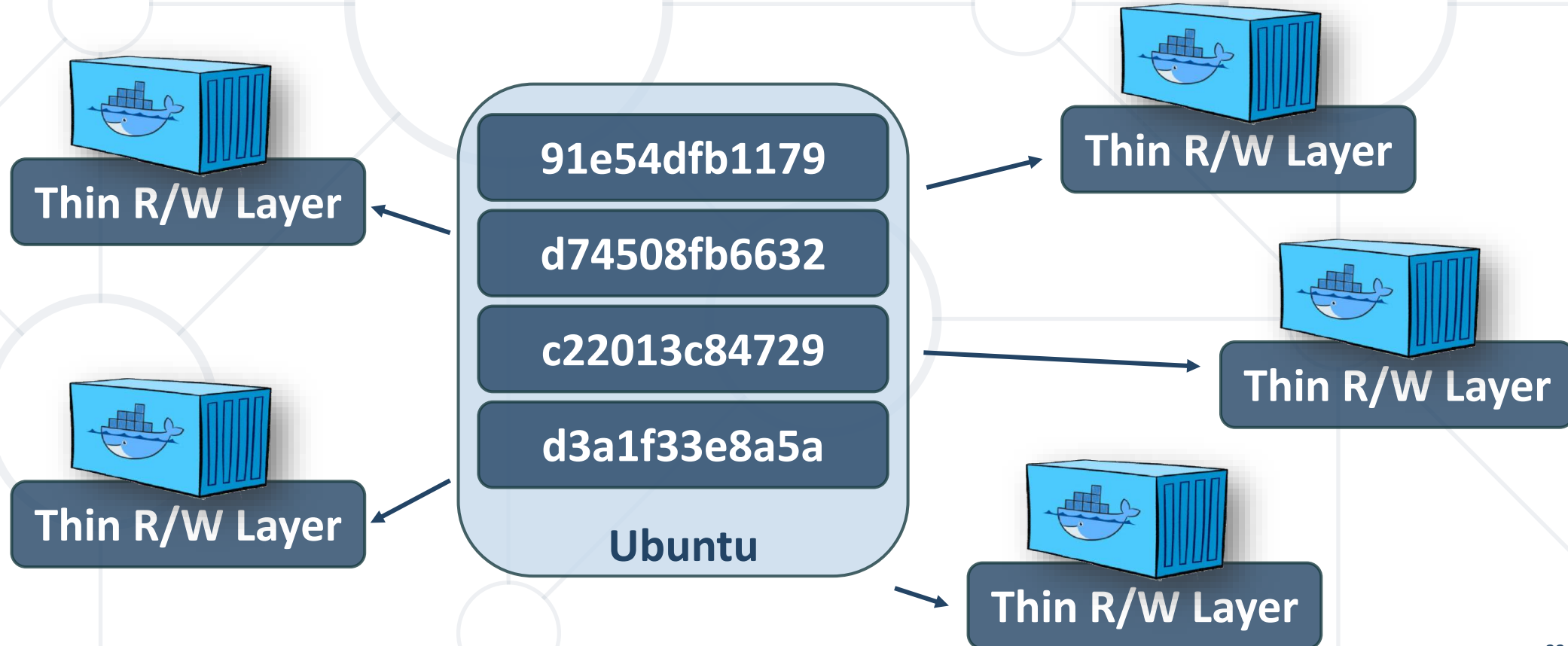


A terminal window shows the command `docker pull node` being executed. The output indicates that the 'latest' tag is being pulled from the library. The first two layers, 17c9e6141fdb and de4a4c6caea8, are marked as 'Already Exists'. The subsequent layers are being pulled or downloaded. A callout box points to the 'Already Exists' status of the first two layers, stating 'Layer exists from another image'.

```
PS C:\Users\PC> docker pull node
Using default tag: latest
latest: Pulling from library/node
17c9e6141fdb: Already Exists
de4a4c6caea8: Already Exists
4edced8587e6: Pull complete
a7969cffbf46: Pull complete
74fbfde6af91: Downloading [=====>] 77.08MB/196.9MB
babbacf2d498: Download complete
bd2943500448: Downloading [=====>] 24.78MB/45.38MB
98eaae01c196: Download complete
61339482de65: Download complete
```

Image layers

- **Images** share **layers**
 - Therefore they load faster once you have them



- Each container is **isolated** and has its **own writable file system**

- By default, file system is deleted after you delete the container

- Which is not very suitable for persistence operations

Delete old container and create a new one

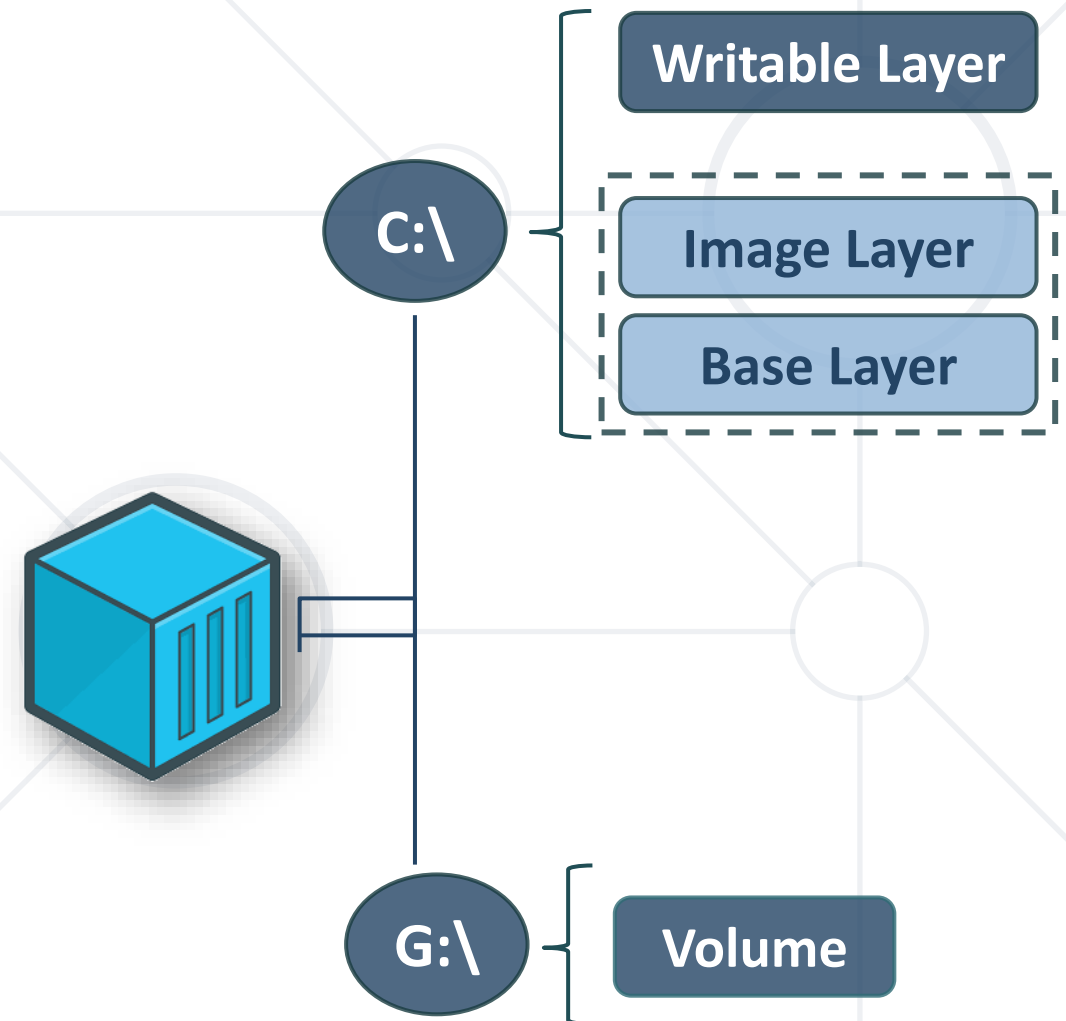
```
PS C:\Users\PC> docker exec -it code_it_up /bin/sh
/ # touch test.txt
/ # ls
bin          dev          media        srv
docker-      dev          mnt          sys
docker-      docker-entrypoint.d  opt          tmp
etc          docker-entrypoint.sh  proc         usr
home         etc          root         var
lib          home        run
            lib        sbin
```

code_it_up
7fbae24f31a3

code_it_up
774cdfc8a290

test.txt file
is missing

- To persist data, use **volumes**
 - Special type of **directory on the host**
 - Mapped to the real file system
 - Can be shared and reused among containers
 - Image updates won't affect volumes
 - Persisted even after the container is deleted
 - You have full control over them

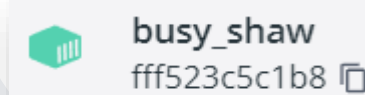


Attach Local Folder as Volume

- Attach **local folder as volume** to a container

```
docker run -p 5001:80 -d -v c:\users:/app nginxdemos/hello
```

```
PS C:\Users\PC> docker run -p 5001:80 -d -v c:\users:/app nginxdemos/hello  
fff523c5c1b81e457a53d51ee5afa963553c8523766846f906002053a695d157
```



- Examine mapped container's **/app** folder

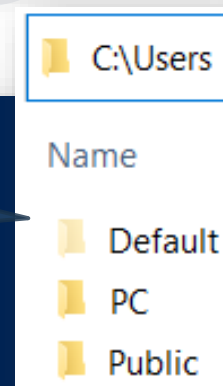
```
PS C:\Users\PC> docker exec -it busy_shaw /bin/sh
```

```
/ # cd /app
```

```
/app # ls -al
```

```
total 4  
dr-xr-xr-x   1 root    root    4096 Nov  5  2021 .  
drwxr-xr-x   1 root    root    4096 Dec 14  08:50 ..  
lrwxrwxrwx   1 root    root      23 Dec  7  2019 All Users -> /mnt/host/c/ProgramData  
dr-xr-xr-x   1 root    root    4096 Nov  6  2021 Default  
lrwxrwxrwx   1 root    root     25 Dec  7  2019 Default User -> /mnt/host/c/Users/Default  
drwxrwxrwx   1 root    root    4096 Dec 12 12:09 PC  
drwxrwxrwx   1 root    root    4096 Nov  5  2021 Public  
-r-xr-xr-x   1 root    root    174 Dec  7  2019 desktop.ini
```

/app has files
from **c:\users**



Creating and Using Volumes

- **Create** a volume

```
docker volume create myvolume
```

```
PS C:\Users\PC> docker volume create myvolume  
myvolume
```

- **List** all volumes

```
docker volume ls
```

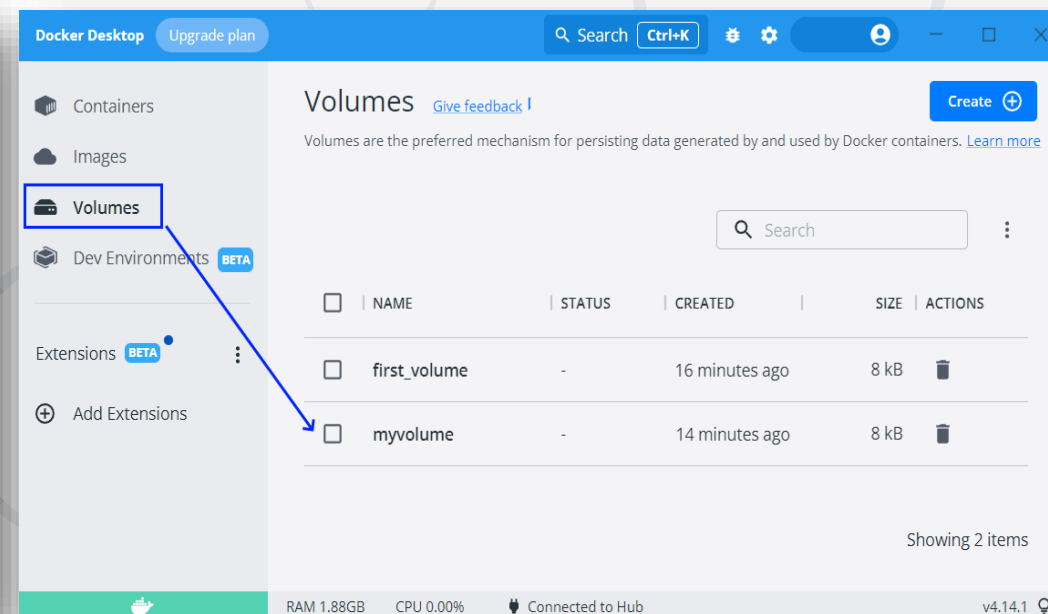
```
PS C:\Users\PC> docker volume ls  
DRIVER      VOLUME NAME  
local      first_volume  
local      myvolume
```

Creating and Using Volumes

- **Inspect** volume

```
docker volume inspect myvolume
```

```
PS C:\Users\PC> docker volume inspect myvolume
[
  {
    "CreatedAt": "2022-12-14T08:14:20Z",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/myvolume/_data",
    "Name": "myvolume",
    "Options": {},
    "Scope": "local"
  }
]
```



Creating and Using Volumes

- **Mount volume** to container

```
docker run -p 5000:80 -d -v myvolume:/myapp nginxdemos/hello
```

```
PS C:\Users\PC> docker run -p 5000:80 -d -v myvolume:/myapp nginxdemos/hello
061e1027c3830bb7321485258e9b1573967be98d998a241c5dfbc1bb30b923f4
```

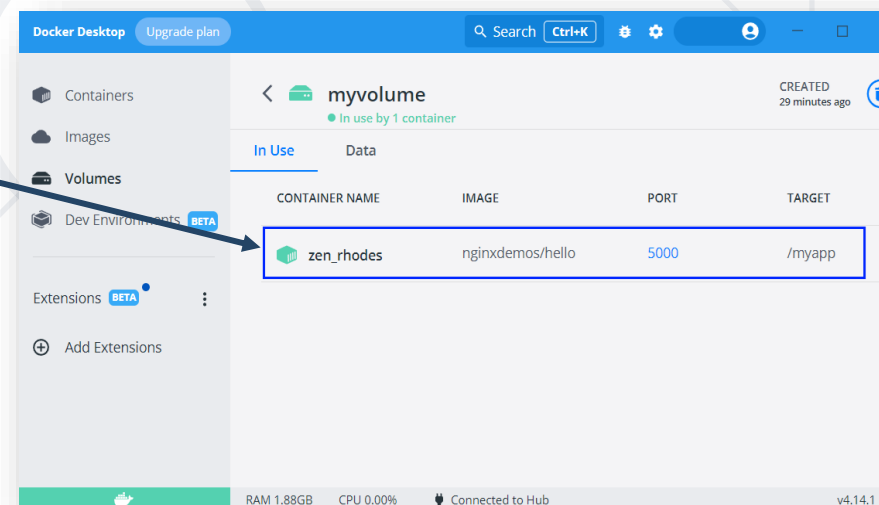


zen_rhodes

061e1027c383

- Create a file in the **/myapp** folder

```
PS C:\Users\PC> docker exec -it zen_rhodes /bin/sh
/ # cd /myapp
/myapp # touch test.txt
/myapp # ls
test.txt
```



- **Remove** volume
 - A volume that is in use cannot be removed
 - You can remove multiple volumes simultaneously

```
docker volume rm myvolume
```

```
PS C:\Users\PC> docker volume rm myvolume  
myvolume
```

Should not be in use



Live Demo

Vue.js App in a Container



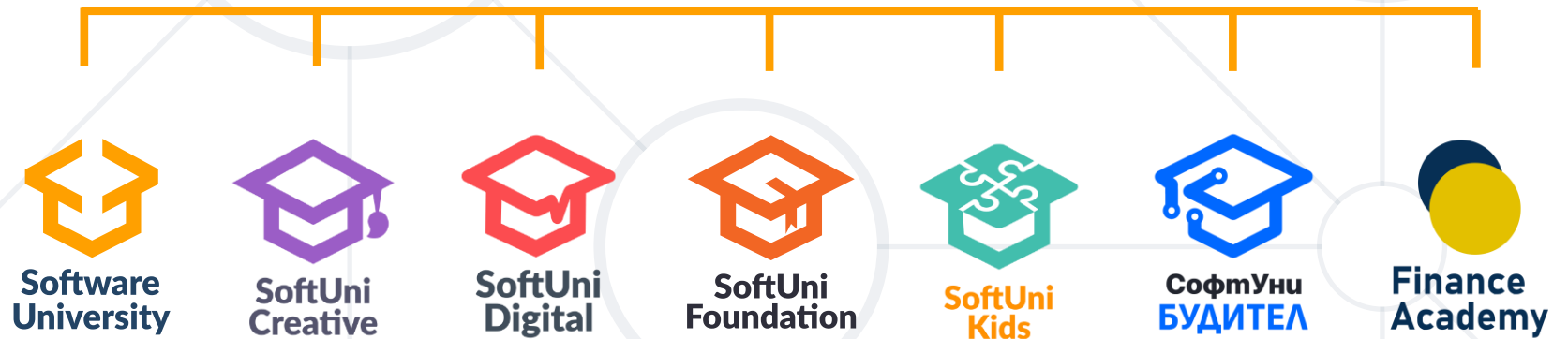
Live Demo

Docker Container with MongoDB

- With **Docker** we can create and manage **images**, **containers**, **volumes**, etc.
 - **Image** == read-only template with instructions for creating a Docker container
 - **Container** == a runnable instance of an image
 - **Volumes** == the preferred mechanism for persisting data
- We can **run apps in containers**
- We can also have a working **database in a container**



Questions?



SoftUni Diamond Partners



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, about.softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

