Containerization

Server OS / Operating Systems II

Contents

- Container architecture
- Microservices and Cloud-Native
- Orchestration
- Docker concepts
- Docker installation
- Docker commands
- Docker Compose

Container architecture

- **Applications/services**: crucial in the operation of organizations
- How to provision them
  - **Securely**
  - **Efficiently**

=> isolated from each other

=> only use the resources it needs

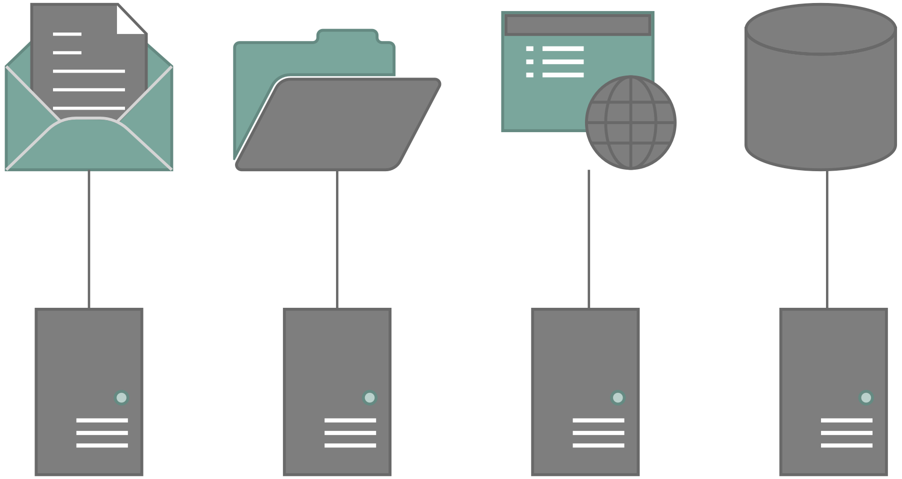
=> Scalable

Container architecture

- **Before 2000**:
  - Very often **1 application/service for each hardware server**
  - hardware very often overprovisioned

    => Waste of resources

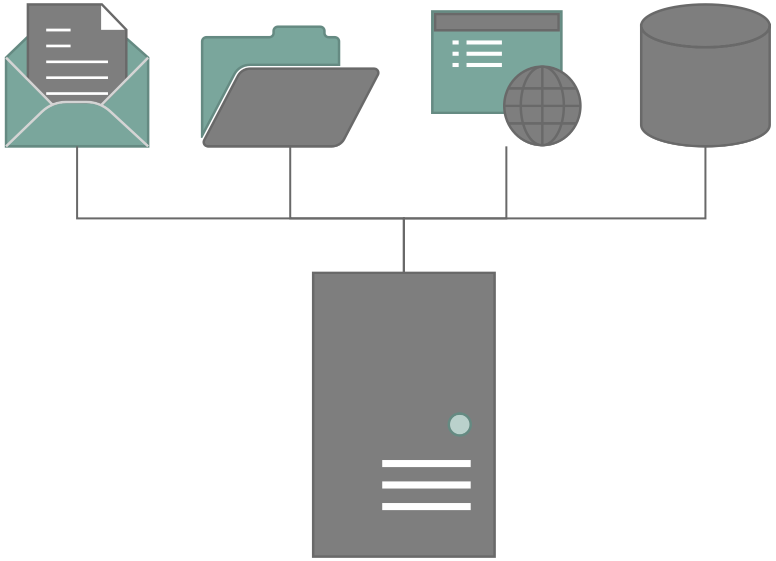
    - Hardware
    - Support
    - Uptime



Container architecture

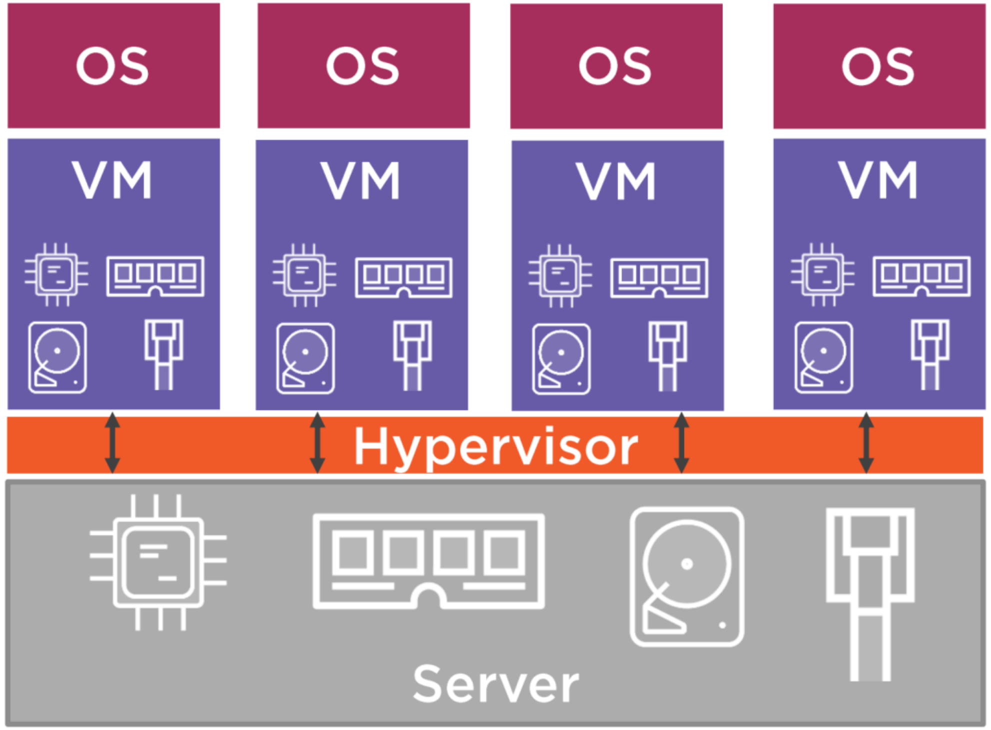
- **From 2000 onwards:**
  - **Virtualization** gets popular
  - Hardware:

    shared between several Virtual Machines (VM's)
  - Less waste of resources



Container architecture

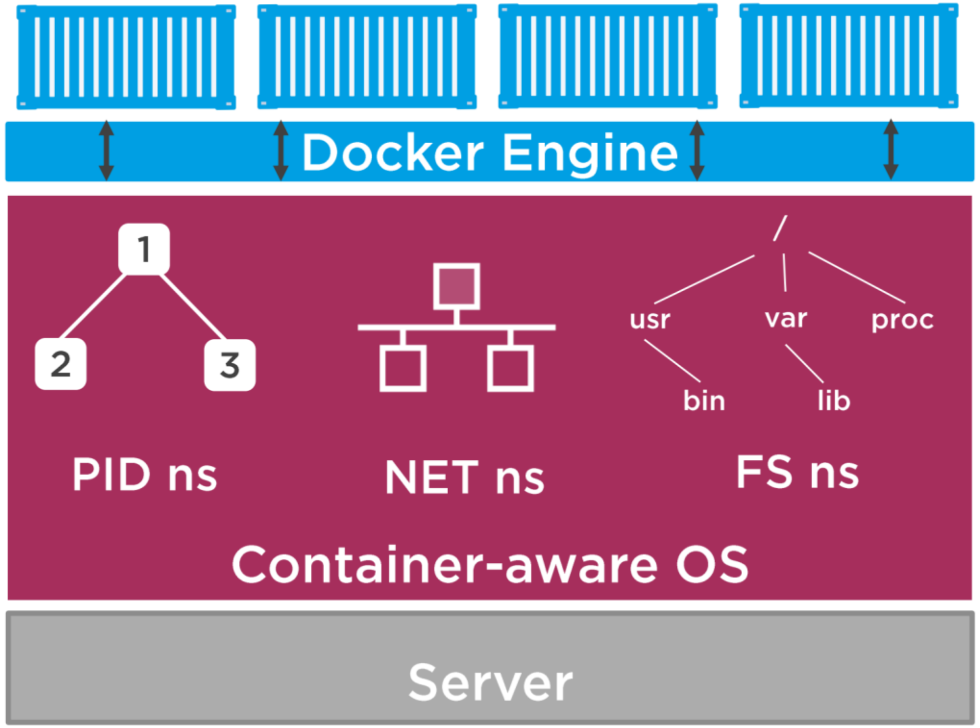
- **From 2000 onwards:**
  - **Every app:**
    - **its own VM**
      - its own virtual hardware
      - Its own virtual Operating System
    - Which requires resources for every VM
    - Often the same full OS in different VM's



https://app.pluralsight.com/library/courses/docker-getting-started/exercise-files

Container architecture

- **Containers**
  - **1 app per container**
  - Containers are isolated
  - each container **separate namespaces**:
    - filesystem
    - process tree
    - users and groups tree
    - network stack



https://app.pluralsight.com/library/courses/docker-getting-started/exercise-files

Container architecture

- **Containers**
  - act like a complete separate OS’s
  - **share the same OS kernel with the host computer**

=> Only one OS kernel necessary

=> less resources needed

Container architecture

- **Containers**
  - => **container OS kernel = host OS kernel**
    - Linux containers on Linux hosts
    - Windows containers on Windows hosts
    - VM’s can be used as host for mixed solutions
      - Linux container on Windows host with a Linux VM in Hyper-V or WSL

Container architecture

- Containers have existed in the Linux world for decades
  - e.g. Google’s search engine
  - Complicated

docker

- **Containers made easy**:

  cgroups and namespaces into manageable containers

Container architecture

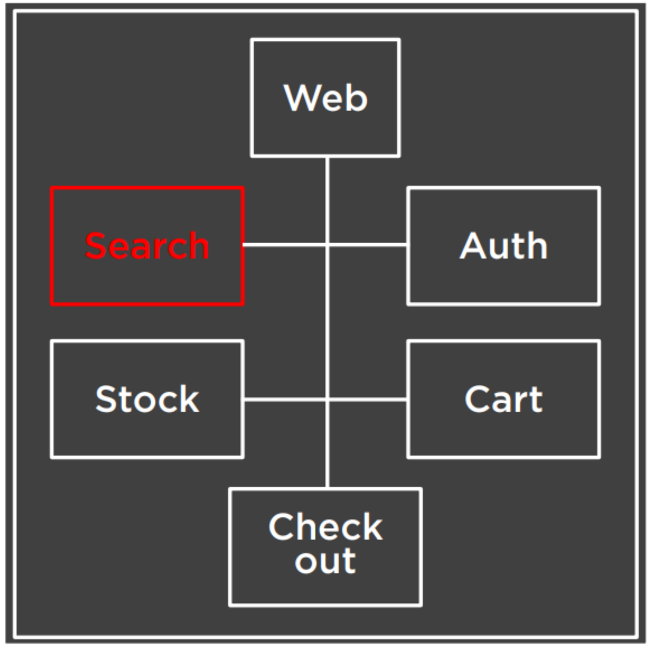
docker

- Controls individual containers
  - start and stop
- **Standard in containerization**
  - **Mainly in Linux**
  - also for Windows and Mac
  - Platform independent:
    - cloud, on-prem, hybrid, Linux, Mac, Windows
    - commands are the same

Microservices and Cloud-Native

- **Monolithic app architecture**
  - Large applications
  - all functionality integrated into one big binary
  - fix/upgrade of one component (e.g. search)?

    => Everything down!

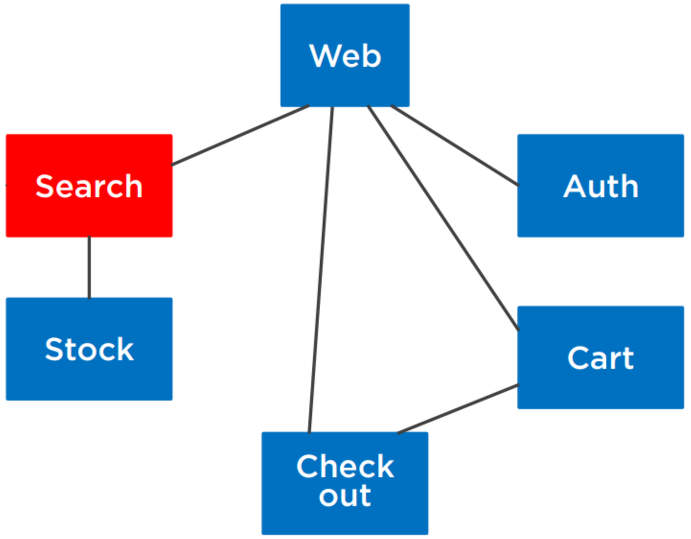


https://app.pluralsight.com/library/courses/docker-kubernetes-big-picture/exercise-files

Microservices and Cloud-Native

- **Microservices:**
  - All the separate components of an application => **split up**
  - fix/upgrade of one component (e.g. search)?

    => Only that component down!



https://app.pluralsight.com/library/courses/docker-kubernetes-big-picture/exercise-files

Microservices and Cloud-Native

- Containers are used as microservices
- **One app/process per container**
  - Scalable
  - Self-healing
  - Portable
  - Efficient in resource usage

Microservices and Cloud-Native

- **Docker itself**: written in a Microservices architecture
  - **Split up in different components:**
  - **Daemon (dockerd)**
    - Provides API for the client
    - Manages the actual containers
  - **CLI Client (docker)**
    - Allows interacting with the containers
      - e.g. start, stop
    - Connects to the daemon (local or remote) for the operations
    - Can be upgraded without stopping containers

Microservices and Cloud-Native

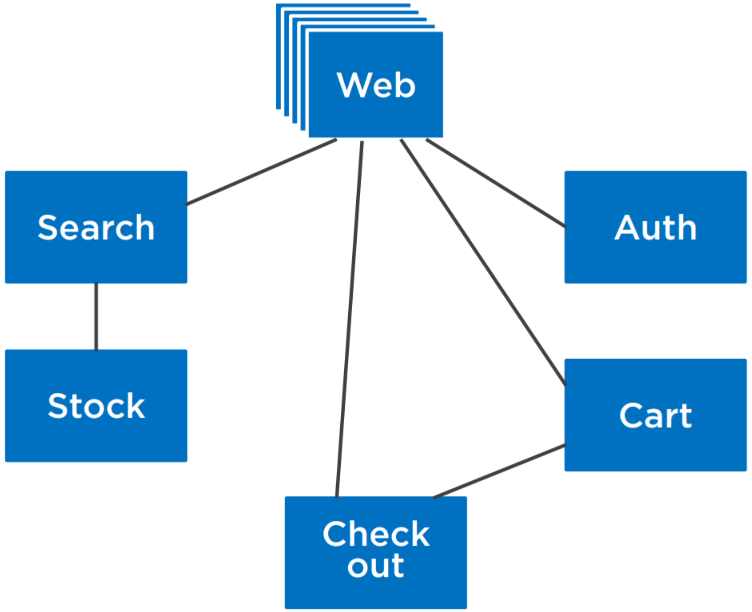
- **Cloud-Native apps:**
  - microservices optimized for the cloud
  - open source
  - containerized
  - dynamically orchestrated
  - microservices-oriented

https://www.cncf.io/about/faq/

Orchestration

- Microservices architecture can become **complex**…
  - Single app per container

    => often requires **many containers for the full stack**
  - Often multiple instances of the same container
  - Often complex communication layout between containers



https://app.pluralsight.com/library/courses/docker-kubernetes-big-picture/exercise-files

Orchestration

- Microservices architecture can become **complex:**
- Containers
  - **dynamically** managed (according to load and failures)
  - often spread-out over **multiple hosts**
    - On-premises (on-prem), cloud or hybrid

Orchestration

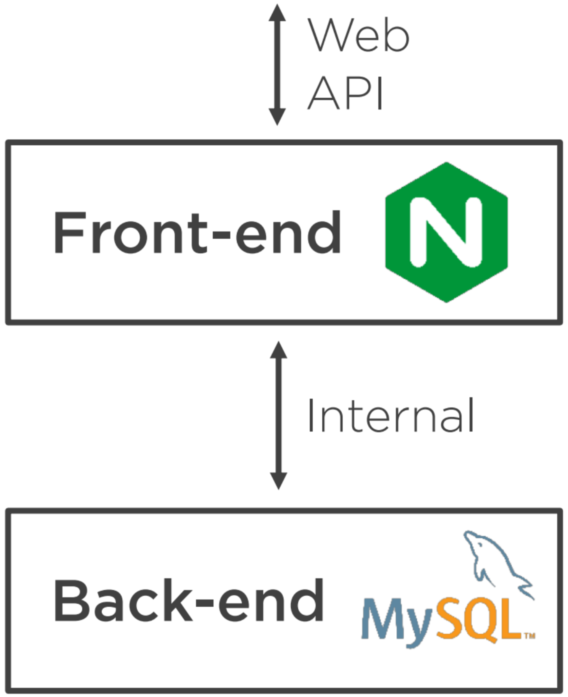
- Orchestration: **automation of management of container stacks**
- cfr orchestra:
    - every musician = container
    - conductor of the orchestra
        - starts and stops different groups
        - sets the tempo
        - manages the stack

Orchestration

- E.g. webstack:
- NginX as web front-end
- MySQL as backend
- Under normal load (desired state):

  2 instances of NginX container

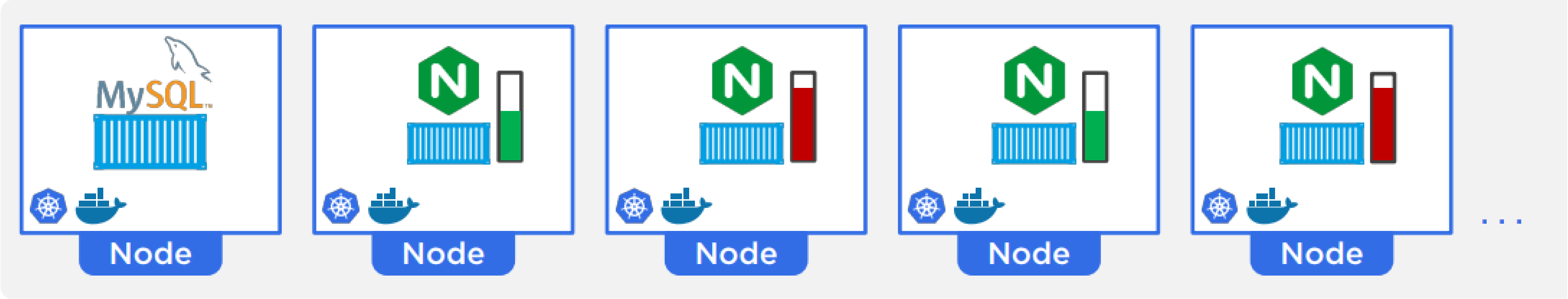
  (for load balancing)



https://app.pluralsight.com/library/courses/docker-kubernetes-big-picture/exercise-files

Orchestration

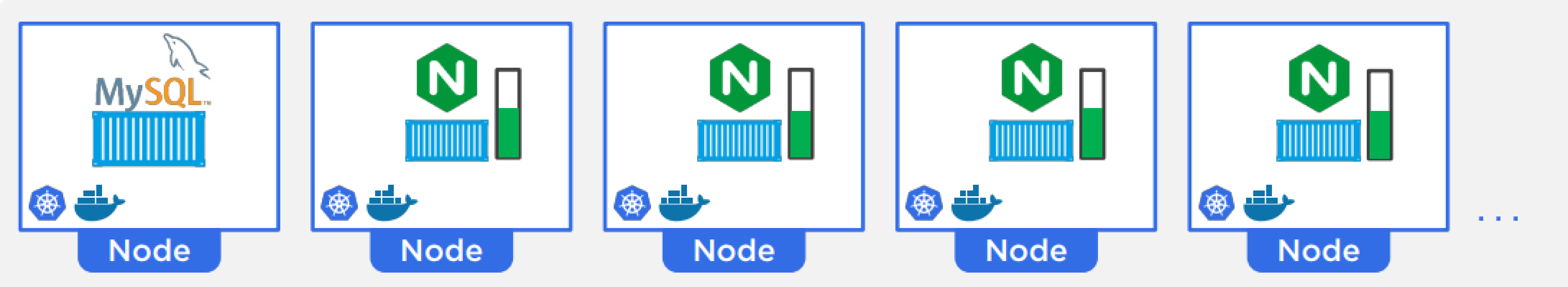
- If the 2 existing instances of NginX (3 and 5) get too much load:
  - 2 more instances (2 and 4)
  - Automatically added by the orchestrator



https://app.pluralsight.com/library/courses/docker-kubernetes-big-picture/exercise-files

Orchestration

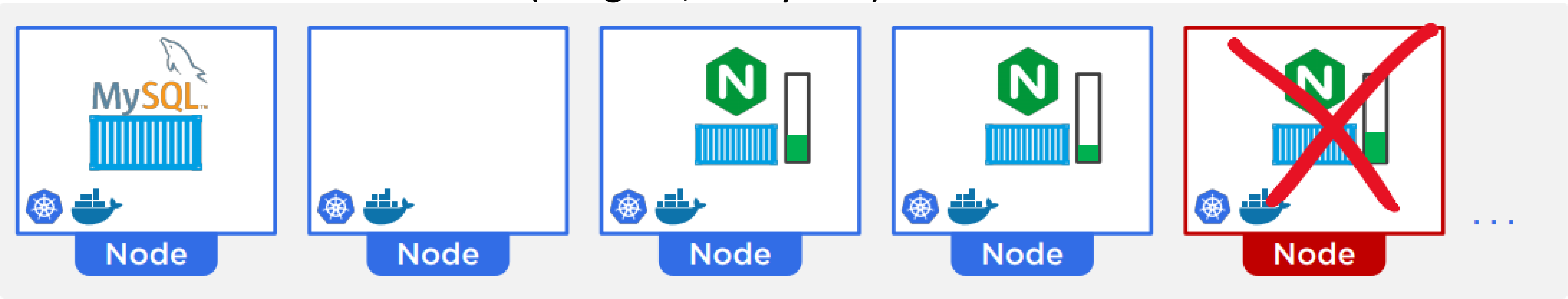
- Load = automatically balanced among all instances



https://app.pluralsight.com/library/courses/docker-kubernetes-big-picture/exercise-files

Orchestration

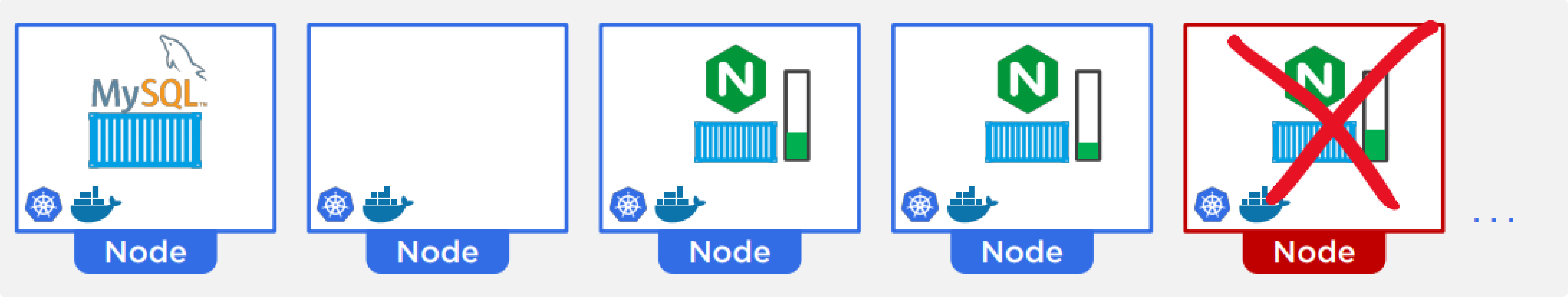
- If load is back to normal:
  - excessive instances removed
  - until the desired state (2 NginX, 1 MySQL) is reached.



https://app.pluralsight.com/library/courses/docker-kubernetes-big-picture/exercise-files

Orchestration

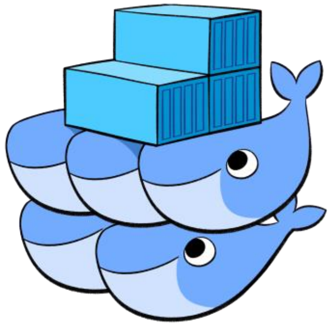
- If instance fails:
  - immediately picked up by the orchestrator
  - replacement instance started up



https://app.pluralsight.com/library/courses/docker-kubernetes-big-picture/exercise-files

Orchestration

- **Docker Swarm**: Docker’s own orchestration
  - **Easy to use**
  - All basic functionality included



Orchestration

- **Kubernetes (K8s)**: Google's orchestration
  - **Large amount of functionality**
  - Becoming the de facto **standard**
    - Integrated in all major Cloud Services
    - Integrated in many Server Class Operating Systems
    - Integrated in Docker

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.

https://kubernetes.io/

Docker concepts

Docker Images

- Basis for every container
- ‘**template**’ for a container
- Read-only -> **immutable**
- **Build-time construct**
- “stopped” container
- Can be pulled from a registry with `docker pull` command

Docker concepts

Docker Images

- Contains:
  - **several stacked layers**
    - to build a unified filesystem
  - json manifest file
    - describes the image
    - how the layers should be combined together
  - e.g.
    - bottom layer for the OS files
    - next layer for the app files
    - third layer for updates

Docker concepts

Docker Images

- Layers are **locally stored**
  - in Linux: /var/lib/docker/[storage_driver_name]/diff
  - In Windows: C:\ProgramData\Docker\Windows Filter
- **extra writable layer**
  - added when a container is created from the image

Docker concepts

Docker Images

- Are stored in **registries**
- Need to have a copy in the **local registry** on the host
- If a local copy is not available

  => **automatically downloaded** (‘pulled’) from an image registry to the local registry

Docker concepts

Docker Images

- Docker registries

  Docker hub (hub.docker.com)

  - Default
  - Contains thousands of images for applications

Docker concepts

Docker Images

- Docker registries

  Docker hub (hub.docker.com)

  - Official images:
    - maintained by the developer of the app
    - Should be: stable, up-to-date, tested and well documented
    - Don’t need a separate namespace
    - e.g. `nginx`

Docker concepts

Docker Images

- Docker registries

  Docker hub (hub.docker.com)

  - Unofficial images: not from the official developer of the app
    - Require a separate namespace
    - e.g. `nginxdemos/hello`
  - Many different versions
    - `latest` version: most up-to-date stable version (usually)

Docker concepts

Docker Images

- Docker registries
  - Other public registries
    - Google
    - Amazon
    - Microsoft
    - …
  - Private registries
    - provided and maintained by your own organisation
    - privately created images

Docker concepts

Docker Images

- Image naming syntax: **registry/repository:tag**

  e.g. `docker.io/ubuntu:latest`

registry: name of the registry

- Default: docker.io (docker hub)
- If default

  => does not need to be mentioned

Docker concepts

Docker Images

- Image naming syntax: **registry/repository:tag**

  e.g. `docker.io/ubuntu:latest`

repository: separate space in a registry

Docker concepts

Docker Images

- Image naming syntax: **registry/repository:tag**

  e.g. `docker.io/ubuntu:latest`

tag: name of the image in the repository

- Default: `latest`
- If default

  => does not need to be mentioned
- `latest` is tagged as such manually by the repository maintainer

Docker concepts

Docker Images

- Are built for a **specific kernel**
  - Windows images are a lot bigger
  - necessary for apps that need a Windows kernel
  - e.g. Docker images for Powershell

Image	SIZE
Windows image for Powershell	5.35GB
Linux image for Powershell	339MB

Docker concepts

Docker Images

- Base images for building app image

  are focused on being very **lightweight**

  - Linux: **alpine**
  - Windows: **nano server**

Docker concepts

Docker containers

- Based on image
- **Runtime construct**: “running” instance of image
- Multiple instances of same image possible

Docker concepts

Docker containers

- Should be **ephemeral**:
  - only used for set period of time
  - changes need to be made in the image
  - replace container with a new one based on the new image
- **Stopping a container**:
  - the container is not removed but exited
  - Data persists in the container

Docker concepts

Docker containers

- Created to be lightweight
- Usually only one process running
  - ...in Linux containers
  - The Windows kernel needs more processes

    = > Windows containers too

Docker concepts
Docker containers

- Created to be lightweight

Example:

```
docker container run -it ubuntu bash
```

=> interactive ubuntu docker

```
vi
```

=> will not work

vi is not available in this image (lightweight)

```
top
```

=> only bash and top (forked from bash) are running

Docker concepts

Docker containers

- Created to be lightweight
  - Example:

  exit the terminal

  => the bash process is only running process

  => bash will be stopped

  => the container itself will also stop
  - To exit an interactive container without stopping

  => do `[CTRL+P+Q]`

Docker concepts

Docker containers

Processtable of a Powershell Windows Container

```
PS C:\> ps

NPM(K)   PM(M)     WS(M)    CPU(s)      Id  SI ProcessName
------   -----     -----    ------      --  -- -----------
     6    1.02      4.34      0.00    1168   1 CExecSvc
     5    0.91      2.19      0.00    1728   1 CompatTelRunner
     7    1.17      4.81      0.03    1244   1 conhost
    10    6.72      8.16      0.03    1736   1 conhost
    11    1.79      4.79      0.19     984   1 csrss
     5    0.82      2.62      0.02     616   1 fontdrvhost
     0    0.06      0.01      0.00       0   0 Idle
    22    4.21     12.37      0.14     520   1 lsass
    58   43.09     78.44      2.09    1264   1 pwsh
    11    2.17      5.96      0.08     500   1 services
     4    1.19      1.24      0.31     956   0 smss
    13    2.87      9.98      0.05     624   1 svchost
    16    2.41      7.96      0.08    1036   1 svchost
    25    6.09     18.20      0.11    1096   1 svchost
    15    3.04      8.76      0.05    1144   1 svchost
    18    7.53     13.60      0.09    1224   1 svchost
    34    5.45     16.94      1.20    1344   1 svchost
    22   11.73     24.00      0.78    1456   1 svchost
     7    1.34      5.60      0.02    1468   1 svchost
     8    1.56      6.17      0.03    1496   1 svchost
     0    0.16      0.13      0.66       4   0 System
    12    1.79      6.93      0.08     284   1 wininit
```

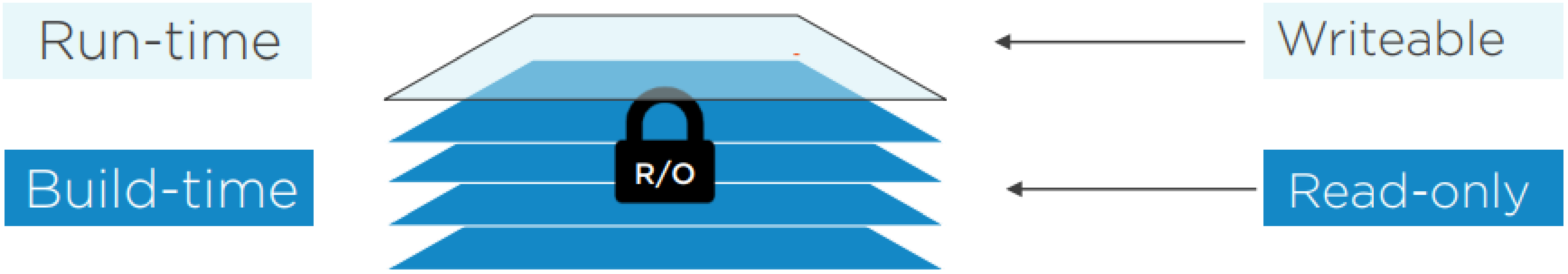
Processtable of a Powershell Linux Container

```
PS /> ps

PID   TTY     TIME       CMD
1     pts/0   00:00:00   pwsh
39    pts/0   00:00:00   ps
```

- Docker concepts

Docker images vs containers



https://app.pluralsight.com/library/courses/docker-deep-dive-update/exercise-files

Docker concepts

Building Docker images

- Similar to building an app outside without container
- Steps are defined in a **`Dockerfile`**
  - `FROM`:
    base image for new image
  - `WORKDIR`:
    directory in your image filesystem
    all actions should be taken here

Docker concepts

Building Docker images

- `COPY`

  copies files from the host to the image

  e.g. your application files
- `EXPOSE`

  documents which ports the application uses
- `RUN`

  run a command

  e.g. to build the application

Docker concepts

Building Docker images

- Additional metadata can be added to the Dockerfile
    - how to run the container based on this image
    - `CMD`

      default process to run in the container

Docker concepts

Building Docker images

- The actual image can be built from the `Dockerfile` with the command…
  - `docker image build -t [image_name:tag_name] [path_to_dir_with_all_necessary_files]`
  - e.g. `docker image build -t testapplication:1.0 /home/user/testapp`
- This image can be used to start a container

Docker concepts

Building Docker images

- Building small images
  - Use **small base images** (cfr `FROM`)
  - Use **multi-stage builds**
    - The first stage

      builds the app with build tools in a temporary image
    - Second stage

      copies the built app over to the final image without the build tools

Docker concepts

Persisting data: volumes

- More info: https://docs.docker.com/storage/volumes/
- Containers should be

  ephemeral
- **Where to keep data** after removing the container?

  => Volumes

Docker concepts

Persisting data: volumes

- **Volumes**
  - allow **mapping folders** from inside the container to the host
  - container is removed:

    data remains
  - can be shared among different containers
  - Can be on a remote host, cloud storage, SAN, NAS…
  - Will by default be created on the host:
    - `Linux: /var/lib/docker/volumes`

Docker concepts

Connecting to containers: port mapping

- The network stack of a container is isolated from the host
- How to access the container from the outside/host?

  => Port mapping

Docker concepts

Connecting to containers: port mapping

- Map a port from the host port to a container port
  - **Host port**:
    - Host will listen to traffic coming in on this port and forward it to the…
  - **Container port**:
    - the container will accept traffic on this port.
    - Should be the same port as the one stated with EXPOSE during build.
- Port mapping syntax:
  - [host_port]:[container_port]
  - e.g.: 80:80

Docker installation

- All the different procedures for installing Docker in different environments are available at https://docs.docker.com
- Docker Desktop
  - Ideal for testing and development
  - Windows

    Requires the Containers and Hyper-V features
      - VirtualBox will not work anymore
    - Also possible through Chocolatey
    - Can run from WSL2 too for linux container support
  - MAC OS

Docker installation

- Docker in Windows Server
  - Ideal for production environments
  - Through Powershell

```
Install-Module -Name DockerMsftProvider -Repository PSGallery
-Force
Install-Package -Name docker -ProviderName DockerMsftProvider
```

Docker installation

- Docker in Linux
  - Suitable for

    production

    development

    testing environments
  - Procedure depends on Linux distribution
  - https://docs.docker.com
  - If available, install using the repositories

Docker commands

```
docker
```

- Displays a quick overview of available docker options and parameters

```
docker version
```

- Displays the installed version of docker

```
docker info
```

- Displays general information of your docker installation

Docker commands

```
docker [image] pull image_name
```

- Pulls the latest image from the registry
- Based on image_name

```
docker [image] pull image_name[:tag]
```

- Pulls an image from the registry
- Based on image_name
- Optional tag referring to version of the image

https://docs.docker.com/engine/reference/commandline/pull/

Docker commands

```
docker [container] run image_name
```

- Runs a container
- Based on `image_name`
- If the image is not present:

  => image downloaded (pulled) from the registry

https://docs.docker.com/engine/reference/commandline/run/

Docker commands

```
docker [container] run [options] image_name
```

OPTIONS

- **`-d`** : **detached**

  container runs in the background

- **`-i -t`** : **interactive** and **TTY -pseudoterminal**

  gives a cli terminal into the container

- **`-p port_host:port_container`** :

  maps the internal `port_container` **to the** `port_host`

  makes internal port available from the host

https://docs.docker.com/engine/reference/commandline/run/

Docker commands

```
docker [container] run [options] image_name
```

OPTIONS

- **`-e environment variable`**

  sets environment variable

- **`--name container_name`**

  names the newly started container with name `container_name`

- **`--mount source=volume_name,target=path_to_targetfolder`**
- **`-v path_to_hostfolder:path_to_containerfolder`**

  mounts the volume on the host with name `volume_name`
  to the folder inside the container `path_to_target_folder`

https://docs.docker.com/engine/reference/commandline/run/

Docker commands

`docker image`

- Manage images

- **`ls` : list**

  List all the images available locally on the host

  Same as `docker images` command

- **`rm image_name` : remove**

  Remove image with name `image_name`

  Same as docker rmi image_name

https://docs.docker.com/engine/reference/commandline/image/

Docker commands

`docker image`

- **`build -t image_name:image_tag` path_to_build_files**:

  build a new image with name **`image_name`** and tag **`image_tag`**

  needs `path_to_build_files` pointing to the build files, including the `Dockerfile`

- **`prune`**

  Remove all dangling images

- **`prune -a`**

  Remove all unused images

https://docs.docker.com/engine/reference/commandline/image/

Docker commands

`docker image`

- **`$(docker images -q)`** = all images

  Can be used with `rm`

  e.g. docker image `rm $(docker images -q)`

  removes all images

https://docs.docker.com/engine/reference/commandline/image/

Docker commands

`docker container`

- Manage containers

- **`ls`** **: list**

  List all the running containers on the host.

  Same as `docker ps` command

- **`ls -a`** **: list all**

  List all the containers on the host.

https://docs.docker.com/engine/reference/commandline/container/

Docker commands

`docker container`

- **`rm container_name/id`: remove**

  Remove container with name or id `container_name/id`

  Same as `docker rm container_name/id`

- **`exec command` : execute**

  Run the command `command` inside the container (must be running)

  Can be used with the option `-it` to run the command `command` interactively in the container

https://docs.docker.com/engine/reference/commandline/container/

Docker commands

`docker container`

- **`logs container_name/id`**

  shows the logs for the container with name or id `container_name/id`

- **`top container_name/id`**

  shows the running processes int the container with name or id `container_name/id`

  same as `top` in a regular linux environment

- **`prune`**

  removes all stopped containers

https://docs.docker.com/engine/reference/commandline/container/

Docker commands

`docker volume`

- **`create volume_name`**

  creates a volume with name `volume_name`

- **`ls`**

  list all volumes on the host

- **`rm volume_name`**

  removes volume with name `volume_name`

https://docs.docker.com/engine/reference/commandline/volume_create/

Docker commands

`docker system`

- **`prune`** : removes all…

  all stopped containers

  all networks not used by at least one container

  all dangling images

  all dangling build cache

- **`prune -a`** : removes all…

  all stopped containers

  all networks not used by at least one container

  all images without at least one container associated to them

  all build cache

https://docs.docker.com/engine/reference/commandline/system/

Docker-Compose

Working with commands:

- Practical for individual containers
- Less so for a stack of containers with plenty of additional parameters

⇒**Docker-Compose**

- Tool for defining and running multi-container Docker applications
- Defined in 1 yaml file
- Needs to be installed: https://docs.docker.com/compose/install/

Docker-Compose

docker-compose.yaml example:

```
services:
•     traefik:
•         image: traefik:alpine
•         container_name: traefik
•         command: --api --docker
•         volumes:
•             - /var/run/docker.sock:/var/run/docker.sock
•             - /dl/config/traefik/traefik.toml:/etc/traefik/traefik.toml
•         ports:
•             - "80:80"
•         environment:
•             - PGID=1000
•             - TZ=Europe/Brussels
•         restart: always
```

Docker-Compose

Start:

V1: docker-compose up -d

V2: docker compose up -d

Stop:

V1: docker-compose down -d

V2: docker compose up -d