
# Introduction to containerisation

M.Borowski, CS PUT

# Motivation

- **Develpment**

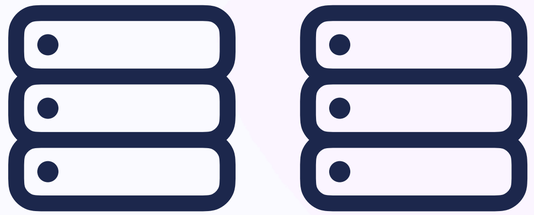
  „To get the development environment set up install Postgres, MongoDB, and run these 5 scripts. Oh wait, you are on Windows? Also change these configurations.”

„Run docker compose up”

- **Deployment**

  „To deploy the application, provision a server running Ubuntu, run this Ansible playbook to install the dependencies and configure the system, then copy the deployment binary and run it with these options.”

„Run this container image with these options”



# Containers

## What is containerisation?

- **Containerisation** is a technology that allows applications to be isolated with their dependencies (libraries, tools, configurations) in lightweight, portable units called containers**.**
- **Containers** operate in isolated environments, but share the kernel of the host operating system.
- **Purpose**: Simplify application deployment, ensure consistency of environments (dev, test, prod) and ease of portability between different systems.

# Containers

## What is containerisation?

- A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application

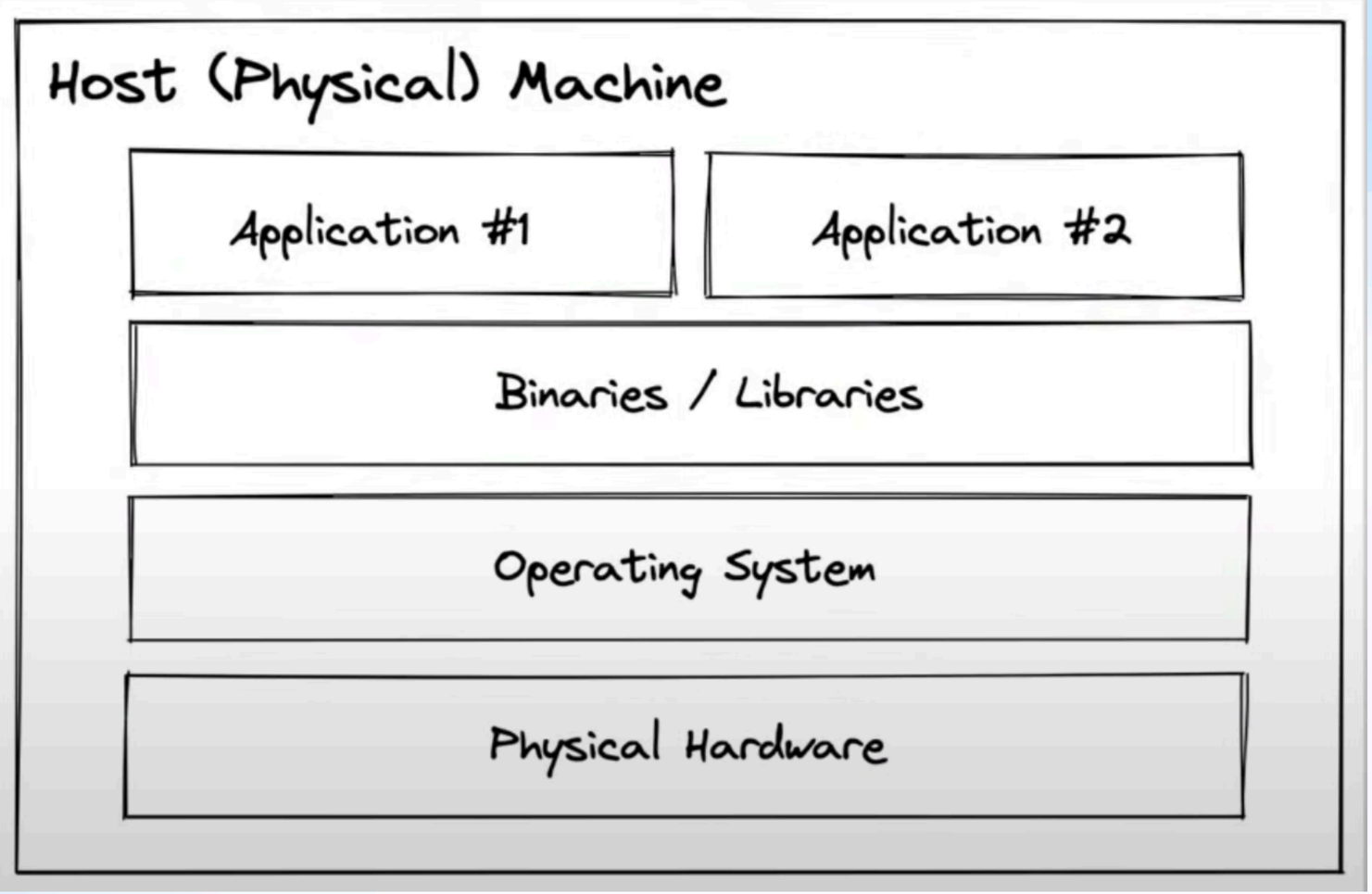
# Open Container Initiative (OCI)

- Runtime Specification
- Image Specification
- Distribution Specification

- The **Open Container Initiative** is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes.

# Evolution of Virtualization

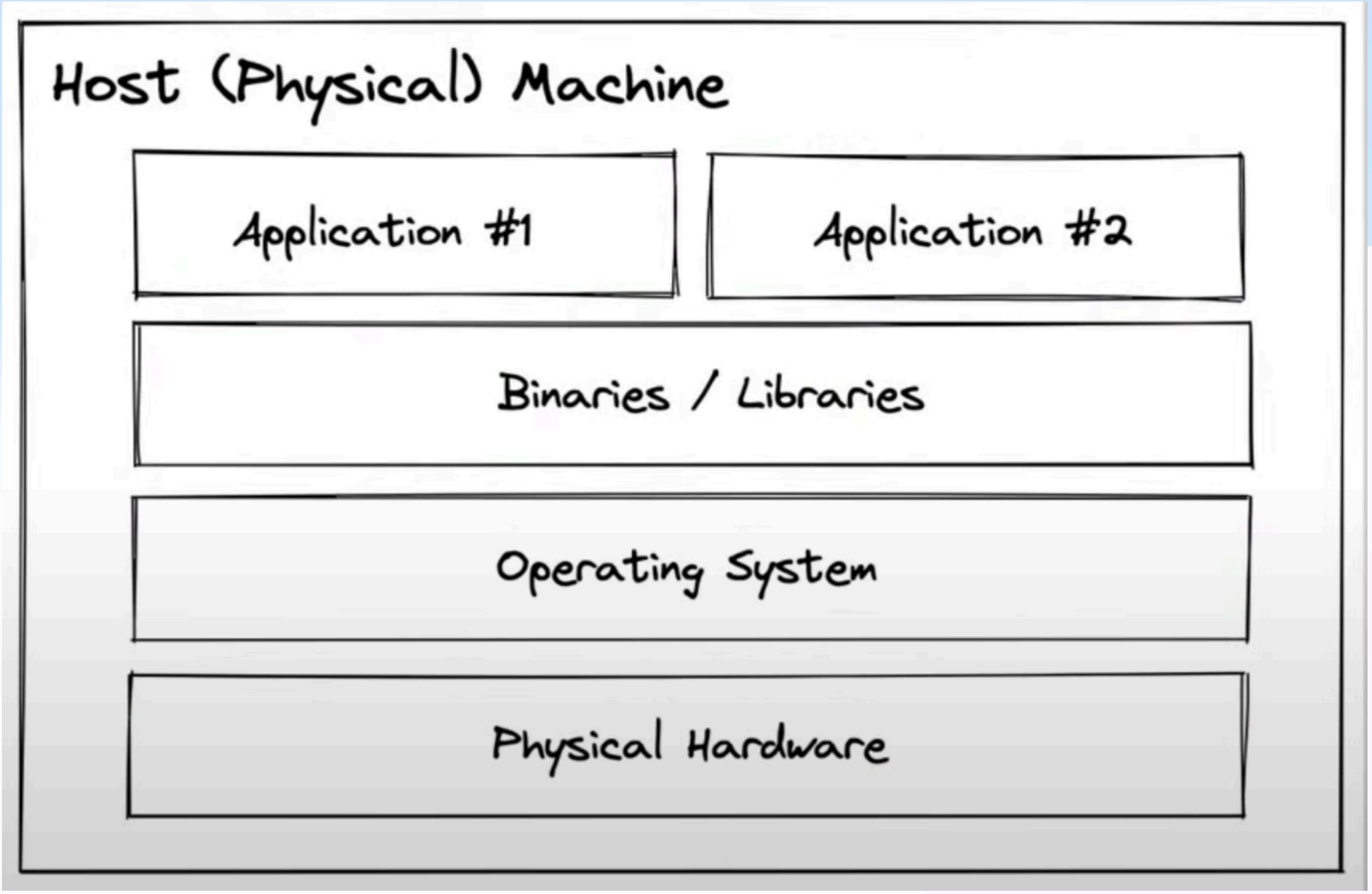
## Bare Metal



# Evolution of Virtualization

## Bare Metal

- Hellish dependency conflicts
- Low utilization efficiency
- Large blast radius
- Slow start up & shut down speed (minutes)
- Very slow provisioning & decommissioning (hours to days)



# Evolution of Virtualization

## Virtual Machines

Type 1

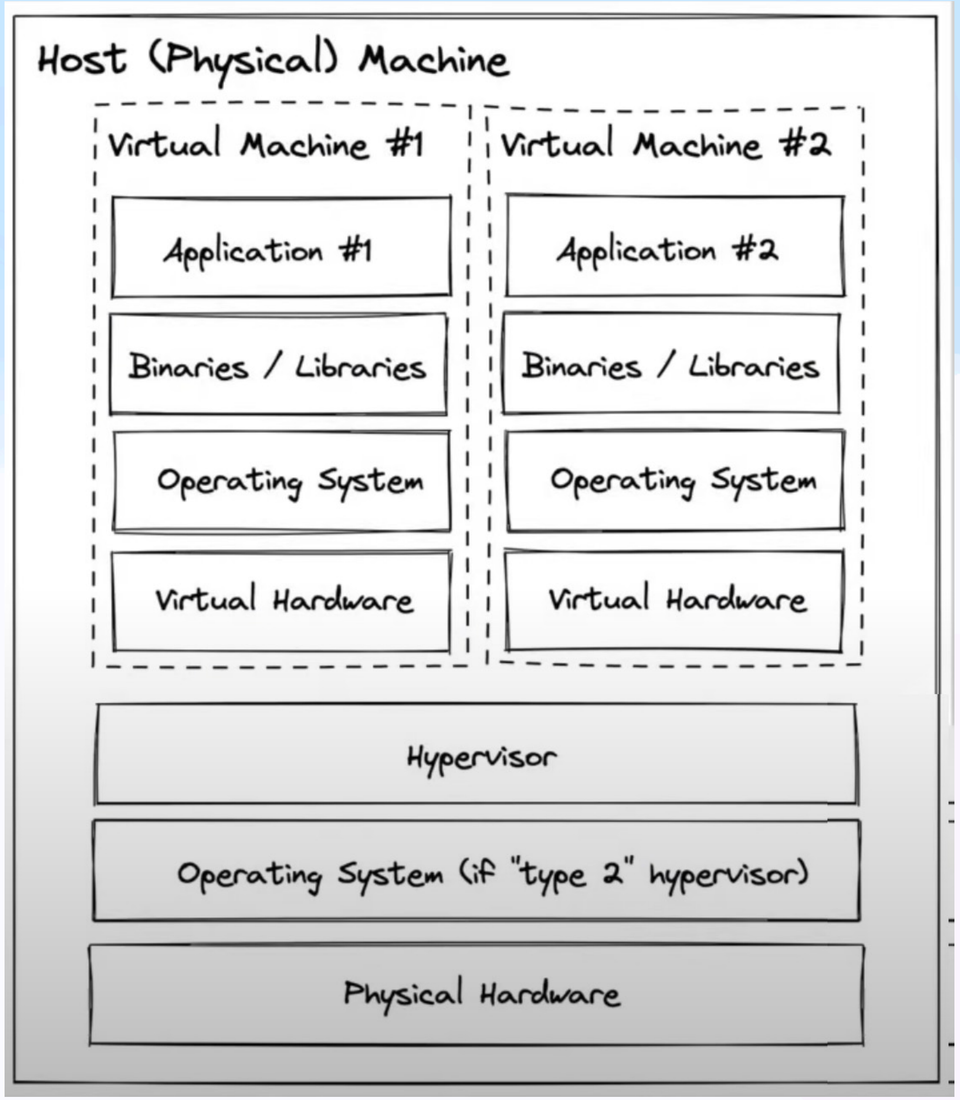
AWS Nitro System

vmware vSphere

Microsoft Hyper-V

Type 2

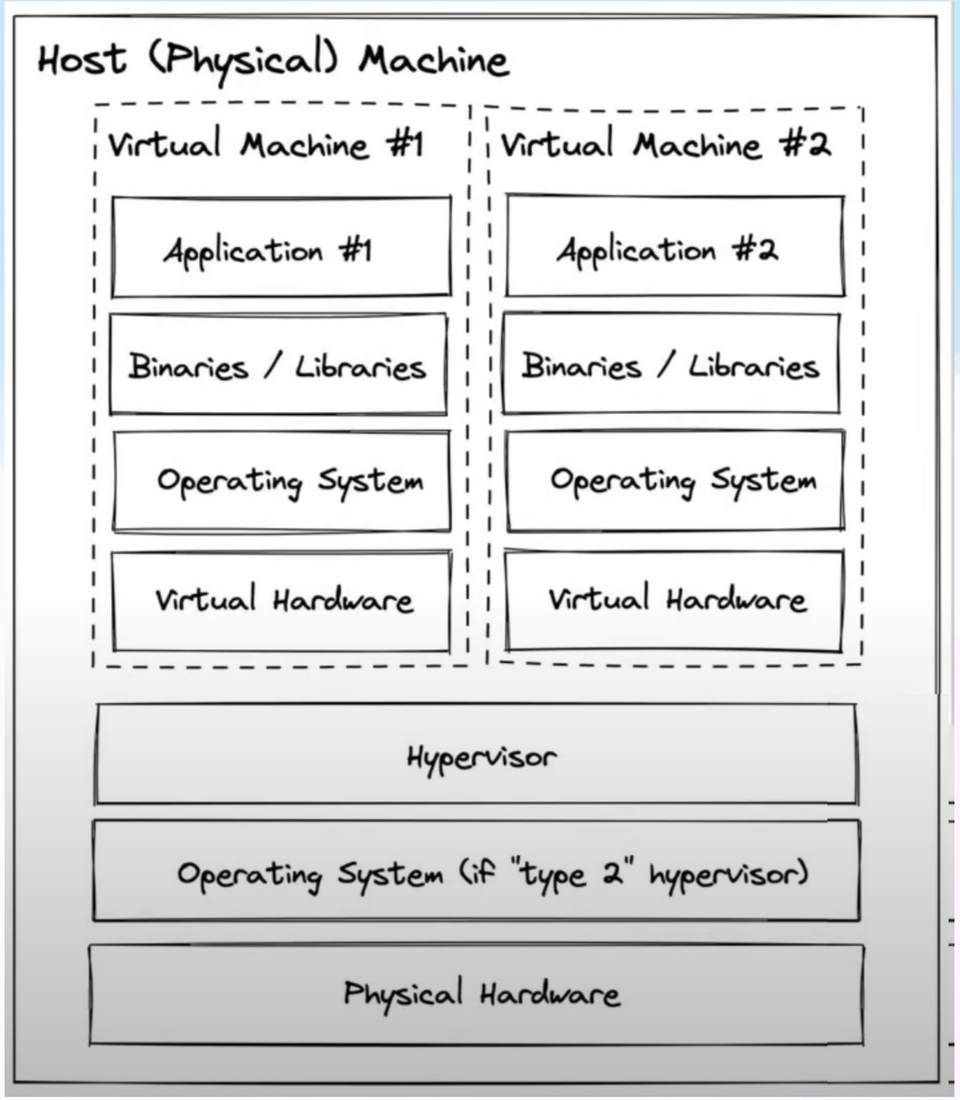
VirtualBox



# Evolution of Virtualization

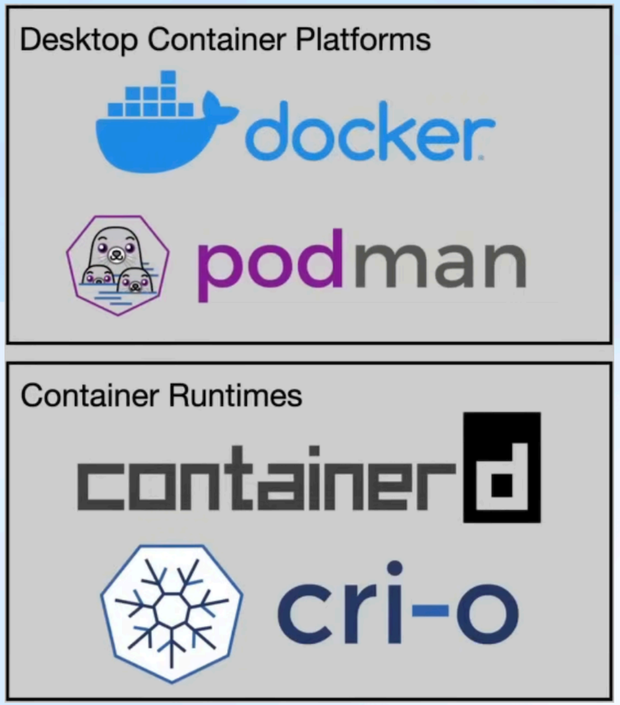
## Virtual Machines

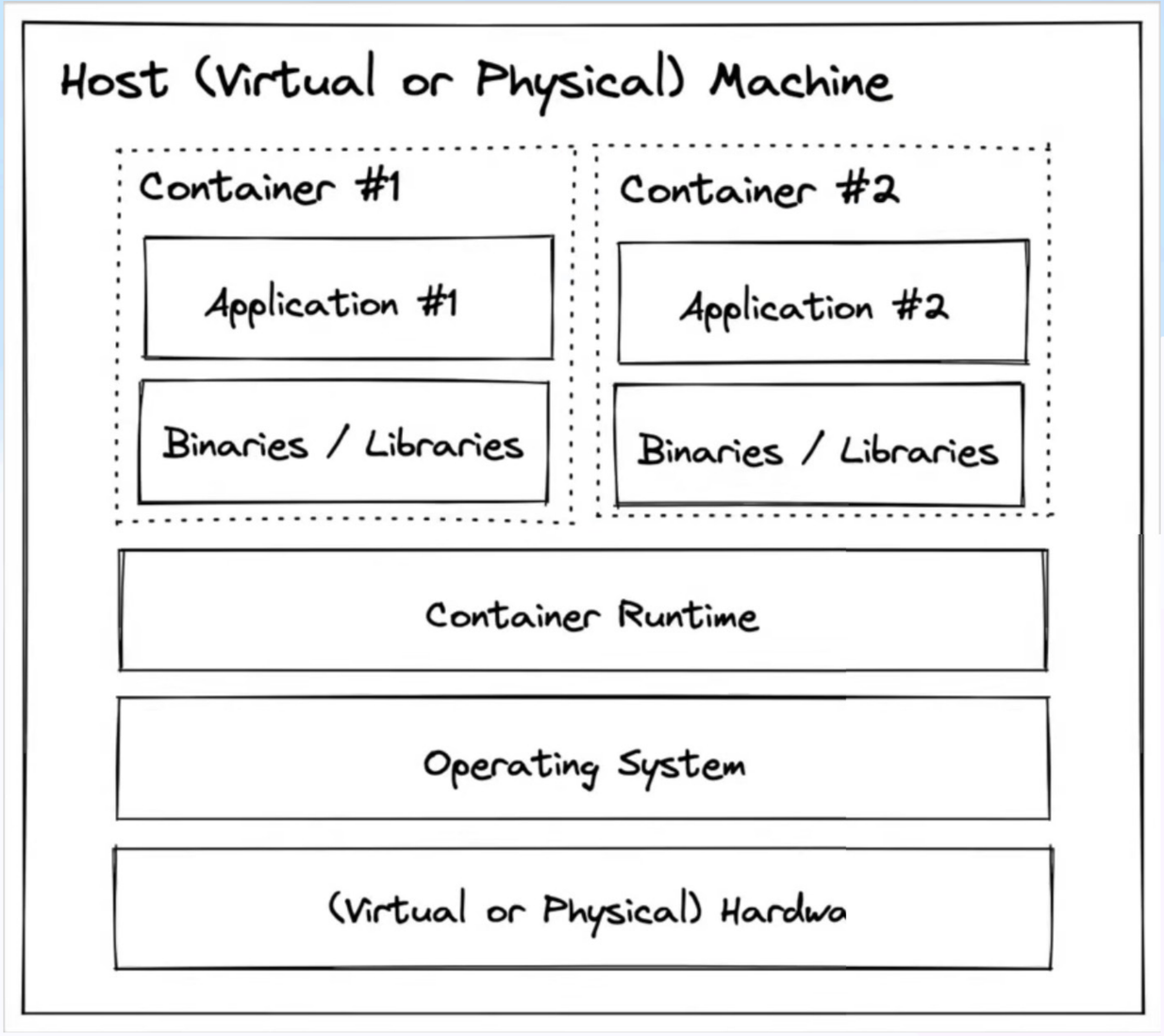
- No dependency conflicts
- Better utilization efficiency
- Small blast radius
- Faster startup and shutdown (minutes)
- Faster provisioning & decommissioning (minutes)



# Evolution of Virtualization

## Containers



# Differences between containers and virtual machines (VMs)

| Containers | VMs |
| --- | --- |
| They share the kernel of the host operating system. | Each VM has its own operating system kernel. |
| They are light and fast (they start up in milliseconds). | They are heavier and slower (they run in seconds/ minutes). |
| They require fewer resources (CPU, RAM, disk). | They require more resources. |
| Isolation at process level. | System-wide isolation. |

# Advantages of containerisation

- Portability: Containers work the same on any environment (local, cloud, server).
- Better resource utilisation: Containers are lightweight, so you can run more of them on the same hardware.
- Speed: Containers start up in milliseconds.
- Isolation: Applications in containers do not affect each other.
- Consistency of environments: No 'it works on my computer' issues.

# Disadvantages of containerisation

- Kernel sharing: If the host system's kernel has a vulnerability, this can affect the security of containers.
- Less isolation than in VM: Containers are not completely isolated from the host system.
- Management complexity: With a large number of containers, orchestration tools (e.g. Kubernetes) are needed.

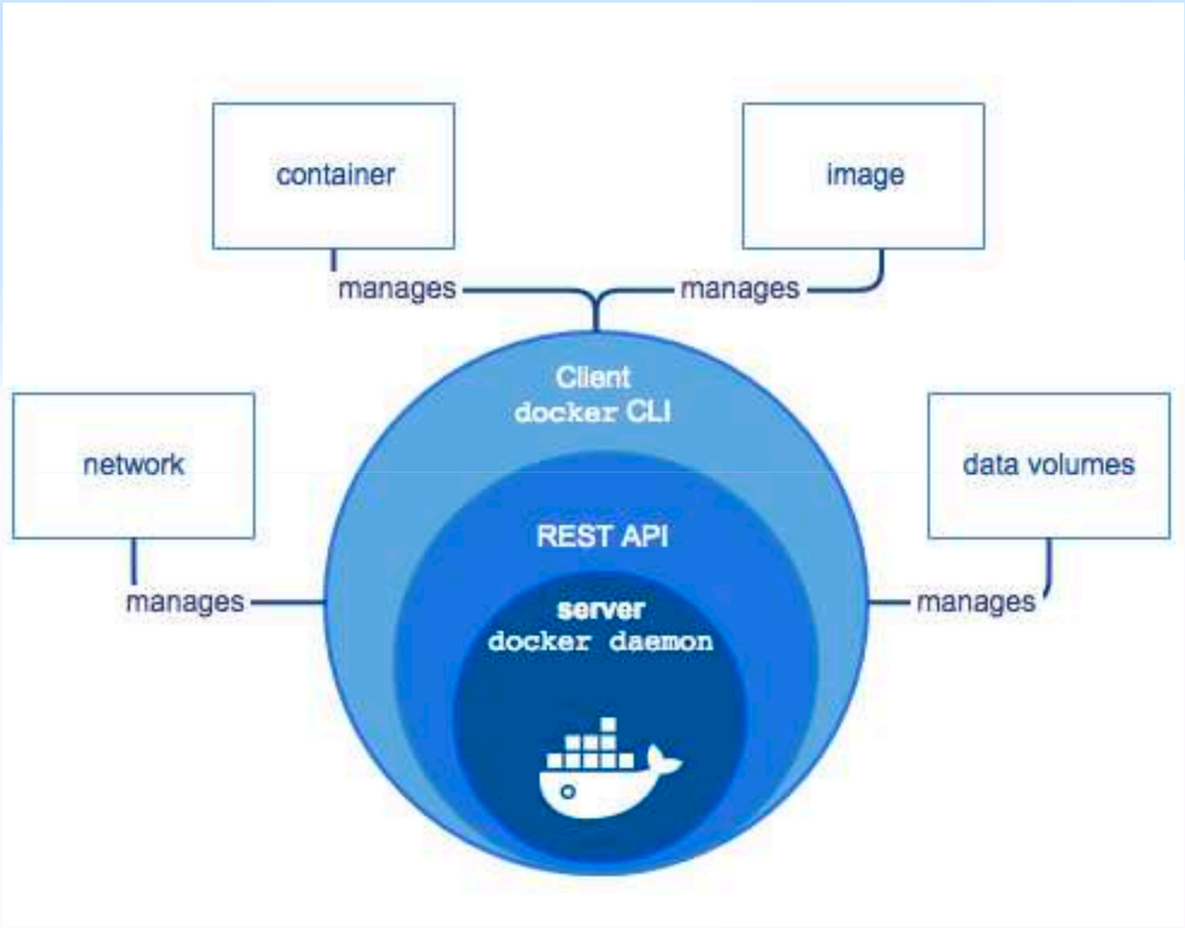
# Examples of the use of containerisation

- Development: Consistent environments for developers.
- Testing: Isolated test environments.
- Production: Deployment of microservices in the cloud.
- CI/CD: Automating build and deployment processes.

# Structure of the Docker ecosystem

## Docker ecosystem - key components

- Docker is a comprehensive containerisation management tool with several main components:
- Images
- Containers
- Volumes
- Networks
- Docker Hub
- Dockerfile
- Docker Compose
- These components work together to enable the creation, launch and management of applications in containers.



# Images

## Docker images - the basis of containers

- What is an image?

  An image is a ready-made template containing the application, its dependencies, libraries and configuration.
  Images are immutable - once created, they cannot be modified.
  Possibility of using one image as a base for creating another image.

- How are images created?

  They are created using a Dockerfile, which contains instructions for building the image.
  Example:
  **docker build -t my-application**

- Where are the images stored?

  Locally on disk (in the Docker cache).
  In registries/hubs, such as Docker Hub, where they can be shared and downloaded.

- Examples of images:

  - **nginx** - web server
  - **python:3.9** - an image with Python 3.9 installed.

# Containers

## Containers - running instances of images

- What is a container?

  A container is a running instance of a Docker image.
  Containers are isolated from each other and from the host system, but share the system kernel.
  Possibility of running multiple containers on the same computer (from the same or different images).

- How do I run a container?

  Command:
  **docker run <image_name>**

  Example:
  **docker run -d nginx** (runs the container with the Nginx image in the background).

- Container lifecycle:
  - Start (**docker run**).
  - Stopping (**docker stop**).
  - Deletion (**docker rm**).

- Usage examples:

  Running a web application.
  Testing an application in an isolated environment.

# Volumes

## Volumes - data management in Docker

- What are volumes?

  Volumes are a mechanism to permanently store data in Docker.
  Data in volumes is independent of the container lifecycle.
  Possibility of mapping between the system directory and the directory seen by the running image as seen by the running Docker image (container);
  ensuring persistence of data (persistence), e.g. written in the container to the database

- Why use volumes?

  To preserve data even after a container is deleted.
  To share data between multiple containers.

- How to create volumes?

  Command:
  **docker volume create <volume_name>**

  To mount a volume to a container:
  **docker run -v <volume_name>:<path_in_container>**

- Usage example:

  - Storing database data (e.g. PostgreSQL) in a volume.

# Networks

## Networks in Docker - communication between containers

- What are Docker networks?

  A mechanism that enables communication between containers.
  Each container can be connected to one or more networks.

- Types of networks:

  Bridge - the default network for containers on a single host.
  Host - containers share a network with a host.
  Overlay - network for containers distributed across multiple hosts (e.g. in Docker Swarm).

- How do you create networks?

  Command:
  **docker network create <network name>**

  Connecting a container to a network:
  **docker run --network <network name>**

- Usage example:

  - A web application communicating with a database via the Docker network.

# Docker Hub

## Docker Hub - image repository

- What is the Docker Hub?

  A public repository for Docker images.
  You can find ready-made images of popular applications (e.g. Nginx, MySQL, Python) there.

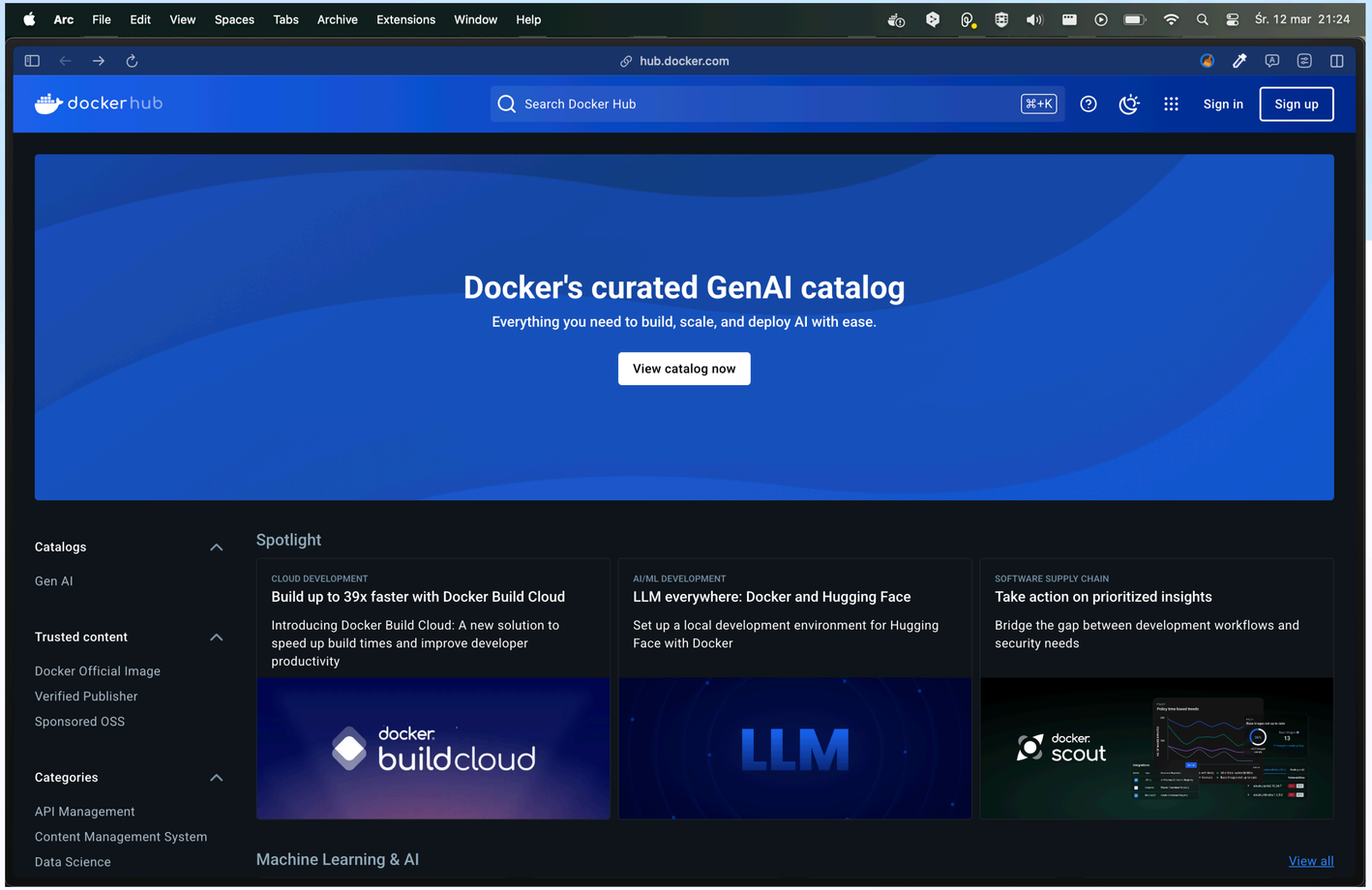
- How to use the Docker Hub?

  Downloading images:
  **docker pull <image_name>**

- Uploading your own images:
  **docker push <image_name>**

- Examples:

  - **docker pull nginx** - downloads an Nginx image.
  - **docker push my-app** - uploads own image to Docker Hub.



# Dockerfile

## Dockerfile - a recipe for an image

- What is a Dockerfile?

  A text file containing instructions for building a Docker image.
  Each instruction creates a new image layer.

- Key instructions:
  - **FROM** - specifies the underlying image.
  - **RUN** - executes commands during the build.
  - **COPY** - copies files from the host to the image.
  - **CMD** - specifies the default command to run in the container.

Dockerfile

```
FROM python:3.9
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

# Docker Compose

## Docker Compose - managing multiple containers

- What is Docker Compose?

  A tool for defining and running multi-container applications.

  It uses the *docker-compose.yml* file for configuration.

yaml

```yaml
version: "3"
services:
  web:
    image: nginx
    ports:
      - "80:80"
  db:
    image: postgres
    volumes:
      - db_data:/var/lib/postgresql/data
volumes:
  db_data:
```