Mastering Docker Essentials - Hands-on

By Thinknyx Technologies LLP

Yogesh Raheja

Puppet for the Absolute Beginners - Hands-on - DevOps

Infrastructure Automation with OpenTofu – Hands-On DevOps

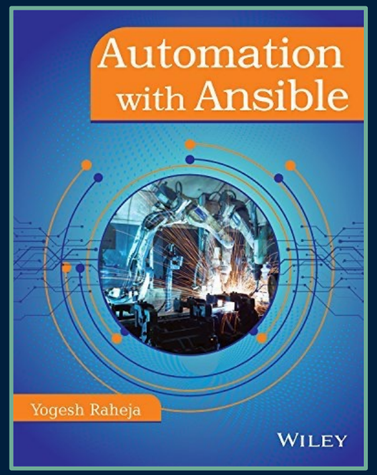
SaltStack for the Absolute Beginners - Practical DevOps

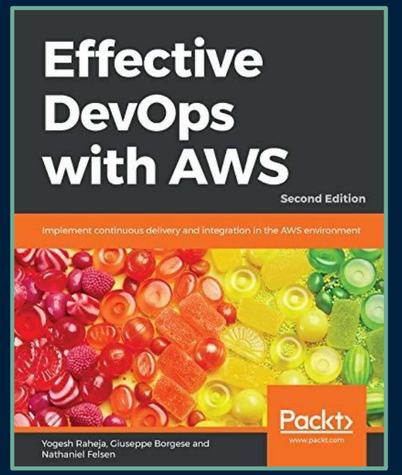
AI Ecosystem for the Absolute Beginners - Hands-On

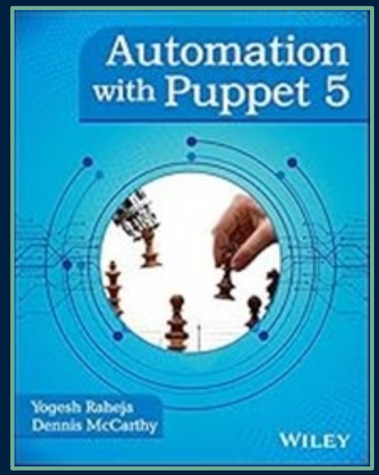
Mastering Prompt Engineering for GenAI - Practical Workshop

Generative AI Essentials - Practical Use Cases

Yogesh Raheja







Deepthi Narayan

Sangram Rath

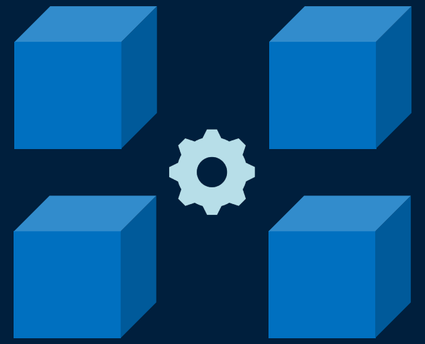
Course Workflow

Images

Containers

Volumes

Networks



Course Workflow

Lectures

Live Demonstrations

Assignments

Course Objective

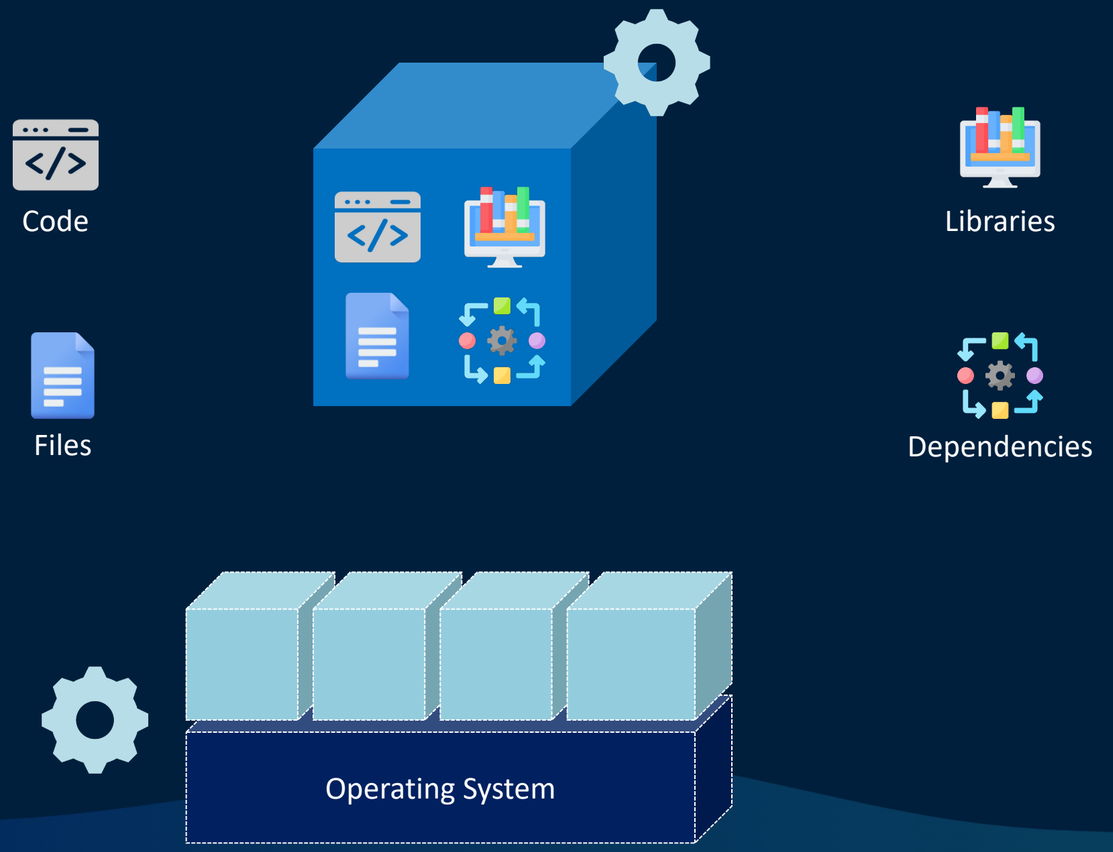
- Docker Introduction
- Setting Up Docker (Desktop/Engine)
- Container Image Management
- Container Lifecycle & Operations
- Understanding Dockerfile
- Docker Network Management
- Docker Volume Management
- Containerize a Python Application
- Multi-stage Build
- Docker Compose for Orchestration

Docker | Introduction to Docker

Introduction to Docker

- What is containerization?
- Containerization benefits
- Available container toolsets
- Introduction to Docker
- Docker architecture

Introduction to Containerization



Introduction to Containerization

Advantages of Containers

Environment consistency

Faster scalability

Development speed

Portability

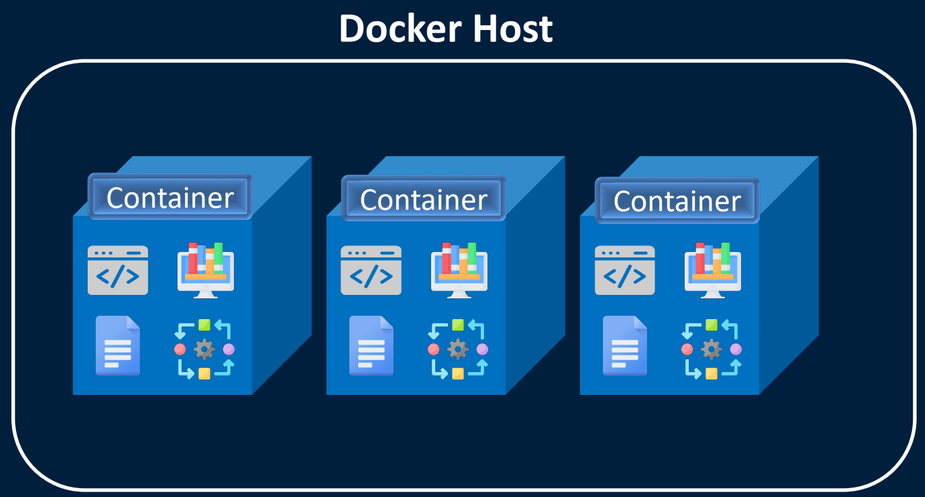
Resource efficiency

Agility

Isolation

Quick startup

Containerization Concepts



Containerization Concepts

- namespaces
- cgroups

Containerization Concepts

namespaces

namespaces

- ✔ Isolate and virtualize system resources
- ✔ Ensure each process accesses unique set of resources

Containerization Concepts

namespaces

Containerization Concepts

cgroups

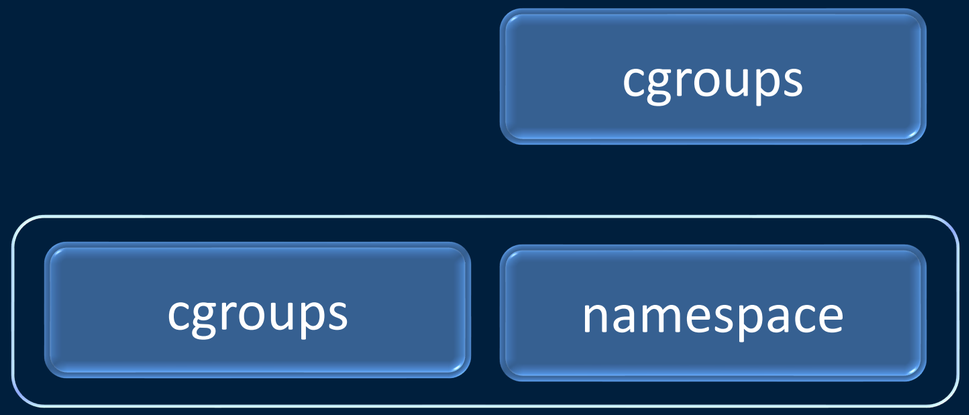
Control group

cgroups

- Isolates the resource usage such as CPU, memory, disk I/O, network etc

Containerization Concepts

cgroups



- ✔ Resource management
- ✔ Isolation and control for processes

Containerization Concepts

Eric Biederman

cgroups

namespace

Paul Menage & Rohit Seth

Available container toolsets

Image Build

Container Runtime

Container Orchestration

Available container toolsets

Image Build

Docker

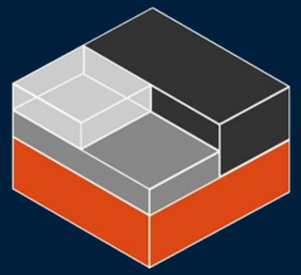
Buildah

Kaniko

Available container toolsets

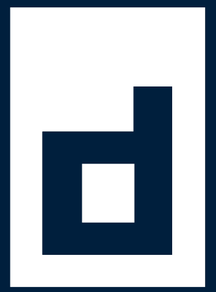
Container Runtime

Docker



LXC/LXD

rkt

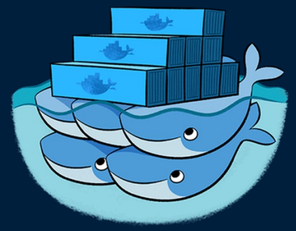


containerd

Available container toolsets

Container Orchestration

Kubernetes



Docker Swarm

Nomad

Available container toolsets

Container Orchestration

Amazon Elastic Container Service

Amazon Elastic Kubernetes Service

Azure Kubernetes Service

Google Kubernetes Engine

Docker at a Glance

- [D]co[illegible]pplications
- [illegible]ating code deployment to all environments

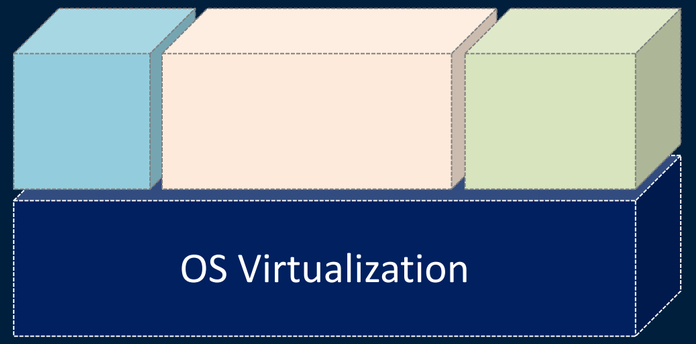
Build

Ship

Run

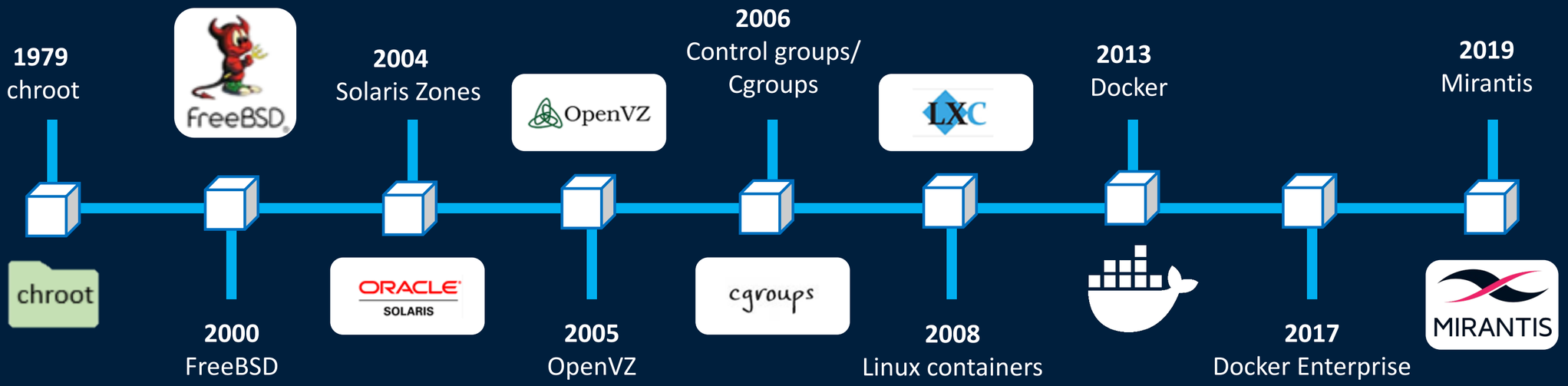
Docker at a Glance

History of Containers & Docker



Docker at a Glance

History of Containers & Docker



Docker at a Glance

Docker Products

Docker Desktop

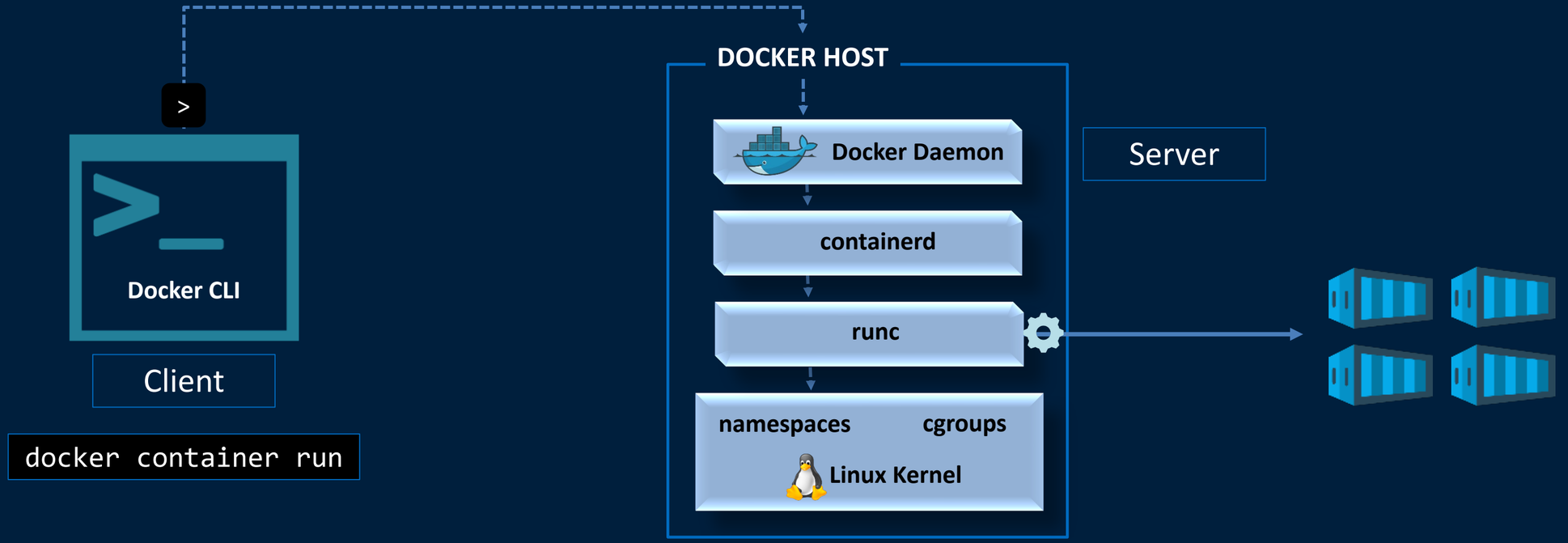
Docker Engine

Docker Hub

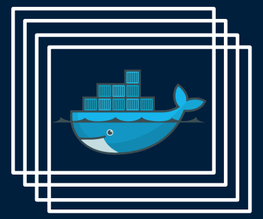
Docker Scout

Docker Build Cloud

Docker Architecture

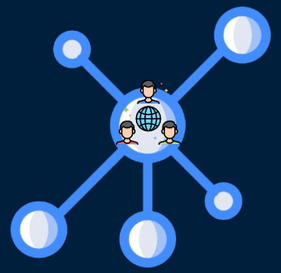


Docker Objects



Images

Containers



Networks

Volumes

Docker Objects

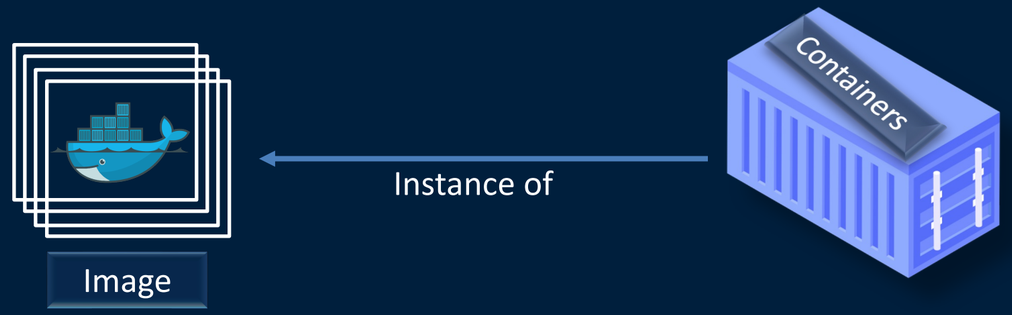
Images

Read-only templates

Set of instructions

Docker Objects

Containers



Lifecycle

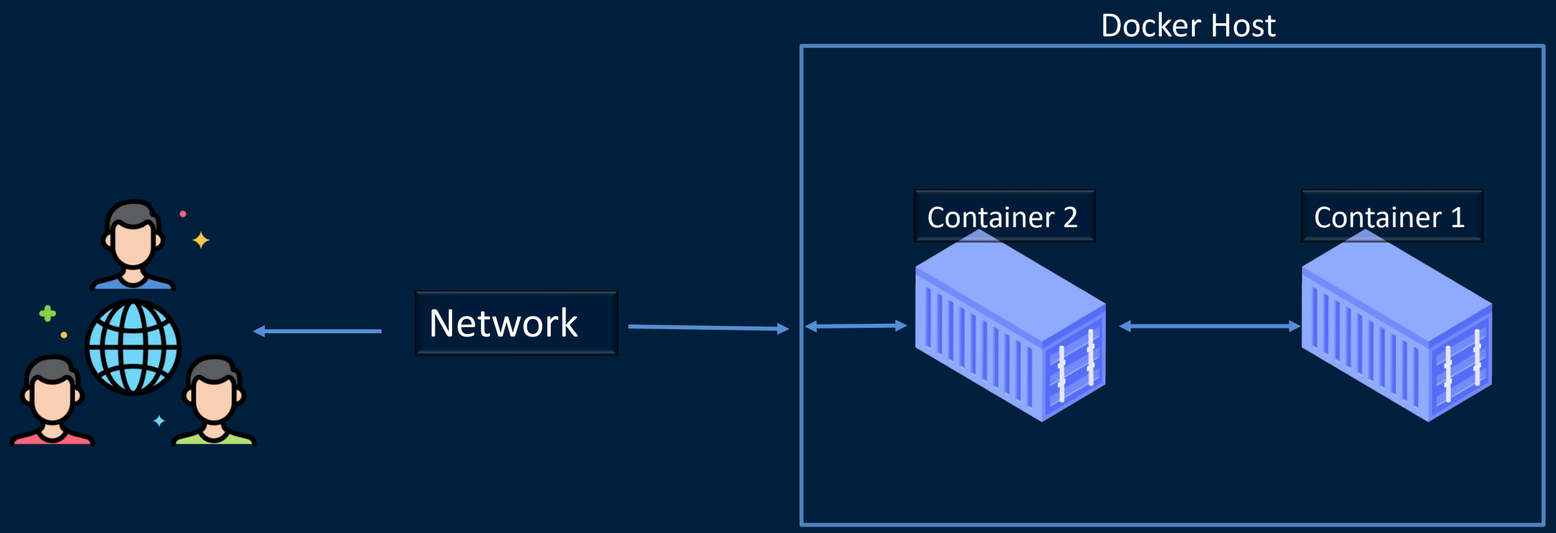
- Create
- Start
- Stop

Docker Objects

Networks

Default network

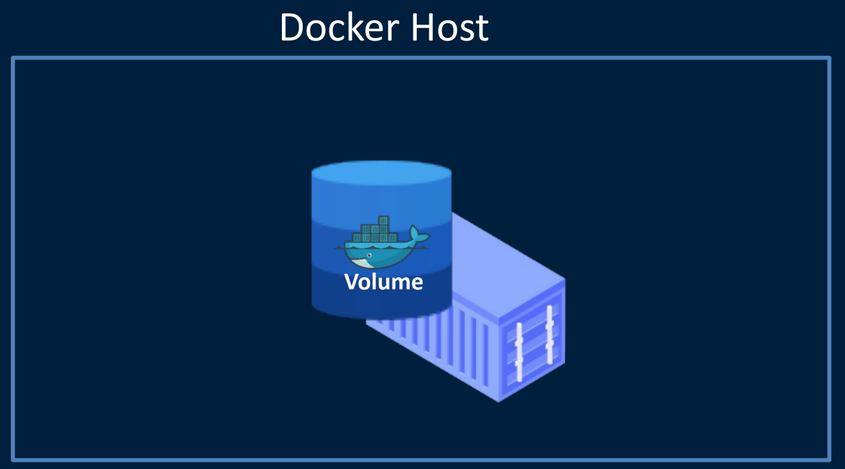
User-defined networks

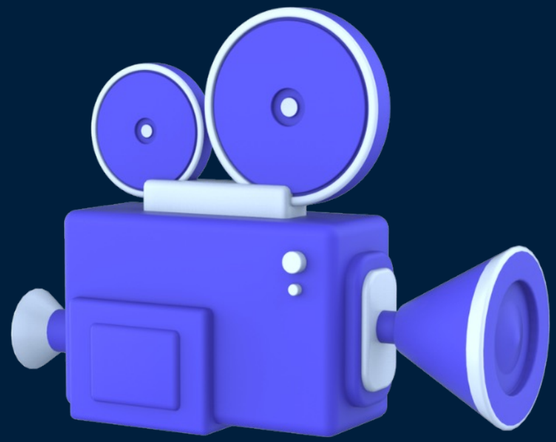


Docker Objects

Volumes

Ephemeral





Demonstration | Official Docker Documentation Walkthrough

Summary

- ✓ Containerization
- ✓ Docker

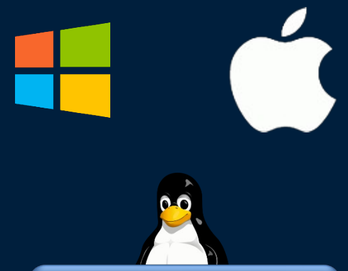
Docker | Getting Started with Docker

Getting Started with Docker

- Install Docker on Linux and Windows
- Introduction to Docker CLI

Demonstration

Overview of Docker Installation Methods

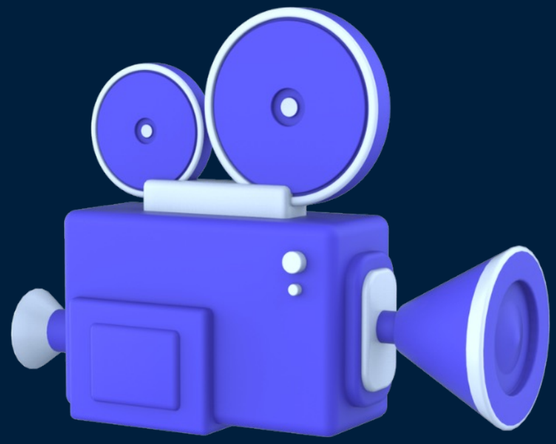


Docker Desktop

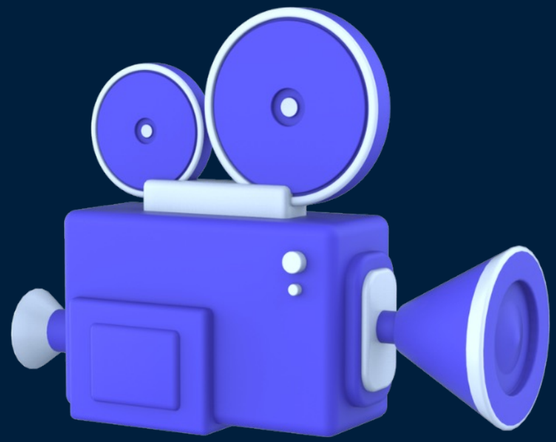
Docker Desktop

Docker Engine

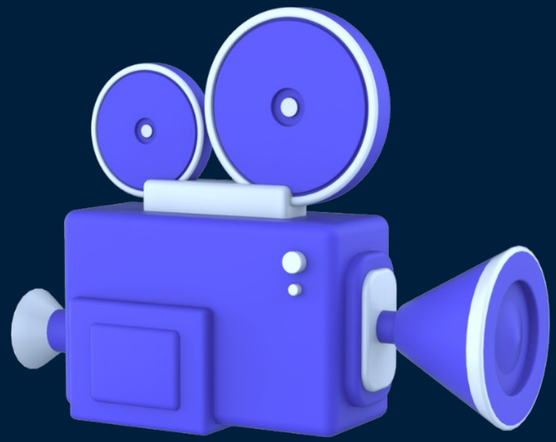
Docker CE (community edition)



Demonstration | Installing Docker on Linux



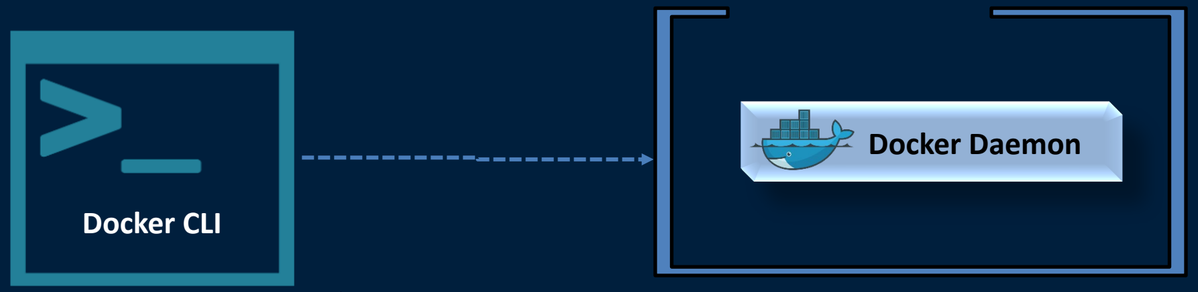
Demonstration | Installing Docker Desktop on Windows



Demonstration | Docker Playground

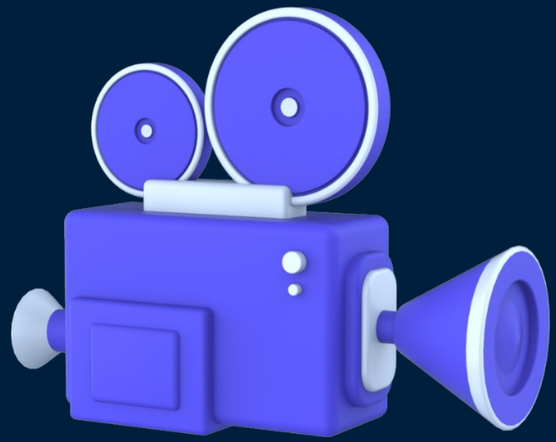
Introduction to Docker CLI

Standard installation process



- ✓ Checked for accuracy
- ✓ Converted to API calls
- ✓ Sent to docker daemon

docker <command> <sub-command> [options] <args>



Demonstration | Introduction to Docker Commands

Summary

- ✓ Different Installation Methods
- ✓ Demonstrations
- ✓ Docker CLI
- ✓ Docker Commands

Docker | Docker Images

Docker Images

- Images
- Managing images with Docker CLI
- Registry

Overview of Docker Images

Docker images = Container images
Read-only templates

Images and Containers

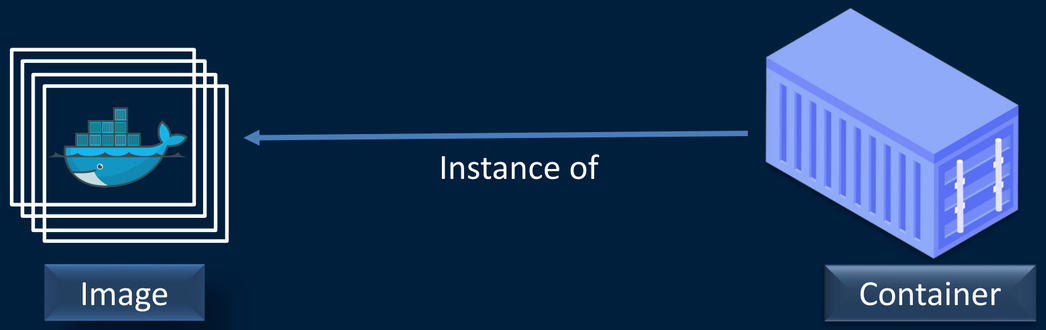
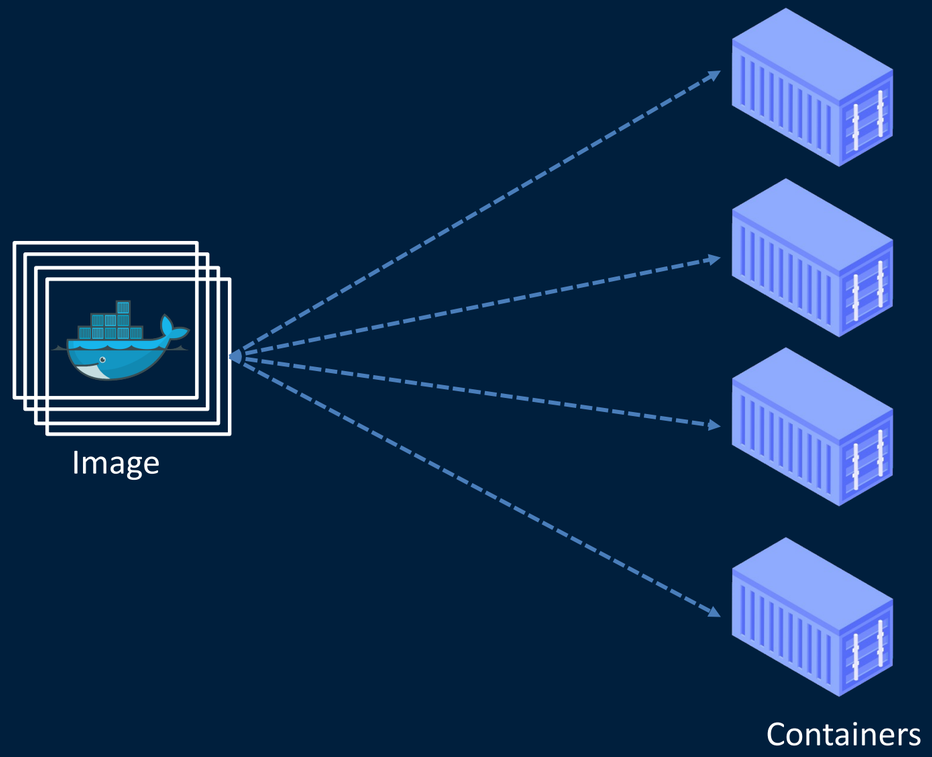


Image = class

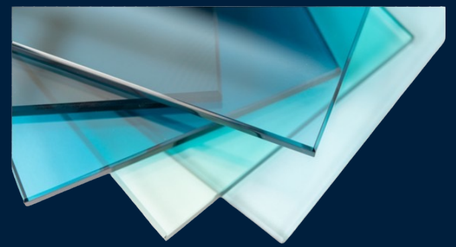
Container = object

Environment and Dependencies

Overview of Docker Images



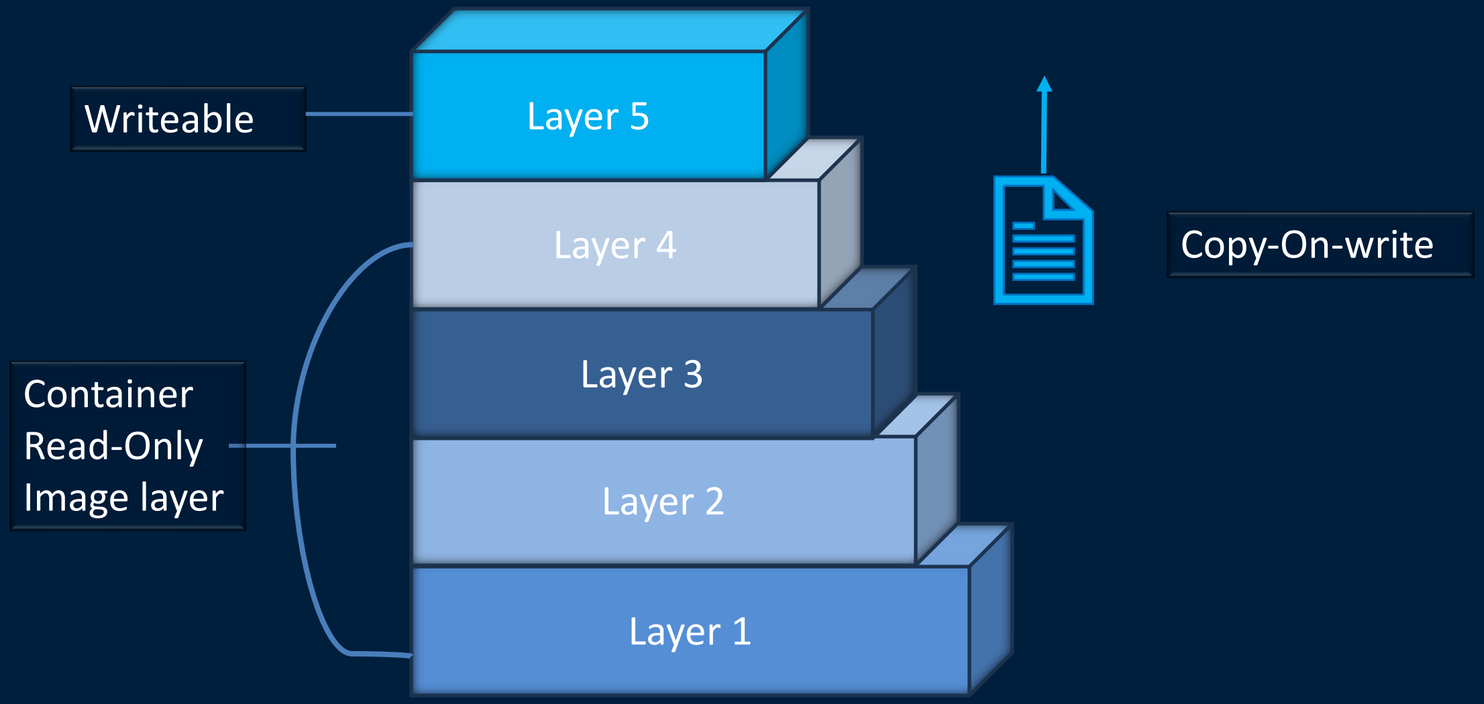
Overview of Docker Images



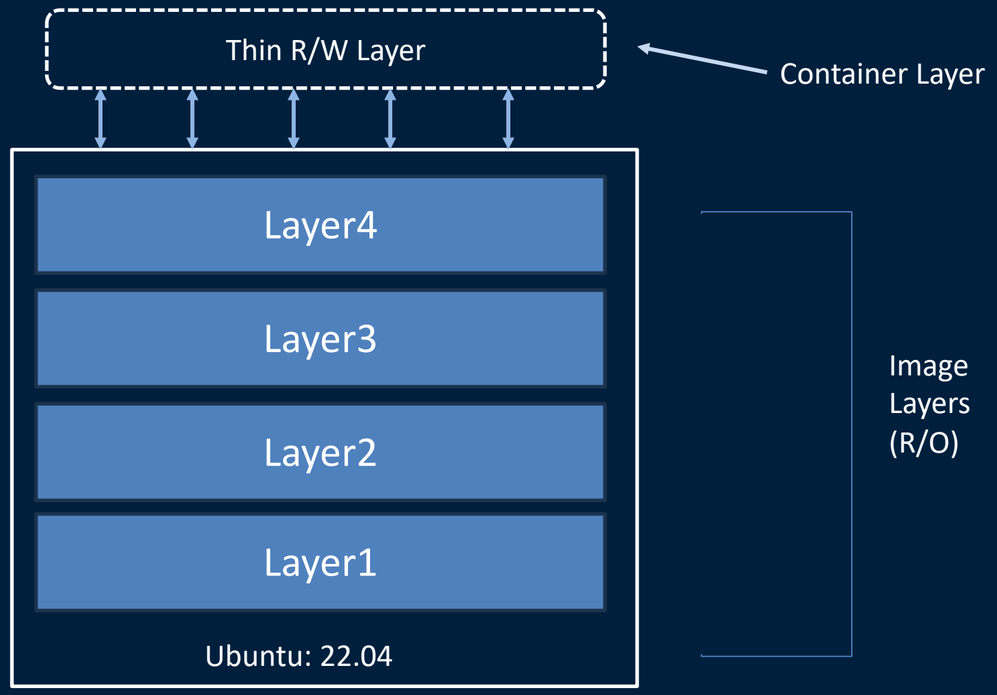
- Files
- Libraries
- Configurations

Overview of Docker Images

Union File System (OverlayFS)

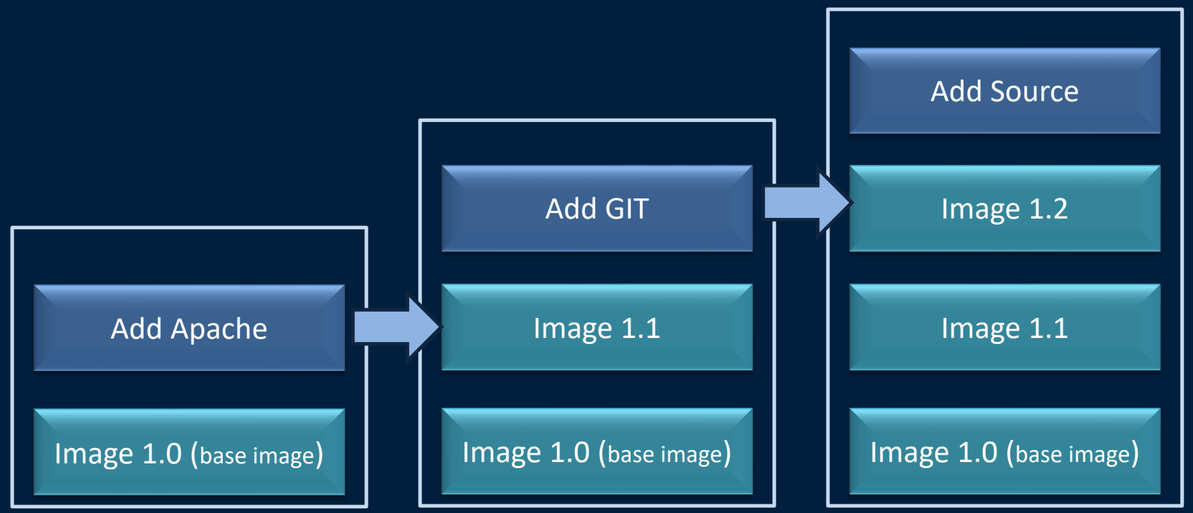


Overview of Docker Images



Overview of Docker Images

- ✓ Gradual modifications
- ✓ Reusability



Overview of Docker Images

Dockerfile

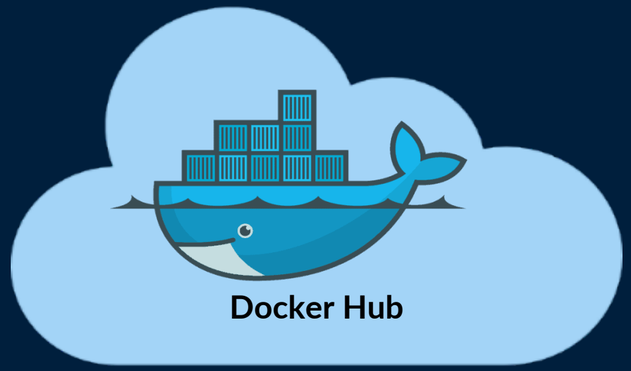
```
FROM centos:7
RUN yum -y update
RUN yum -y install httpd
RUN echo "Hello Docker" > /var/www/html/index.html
EXPOSE 8080
CMD ["httpd","-D","FOREGROUND"]
```

Overview of Docker Images

Registries

Dockerfile

```
FROM centos:7
RUN yum -y update
RUN yum -y install httpd
RUN echo "Hello Docker" > /var/www/html/index.html
EXPOSE 8080
CMD ["httpd","-D","FOREGROUND"]
```



Container Registry

Container Registry

- Store and share container images

Docker registry implementation

CLOUD NATIVE
COMPUTING FOUNDATION

Distribution

Registry (Docker Hub)

Public

Private

Docker Registry

- Harbor
- JFrog Artifactory
- Quay.io
- GitHub Packages

Registry (Docker Hub)

Image Repository

Version Control

Integration

Collaboration

Web Interface

Automated Builds

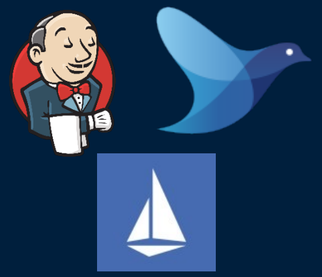
Registry (Docker Hub)

Types of Images

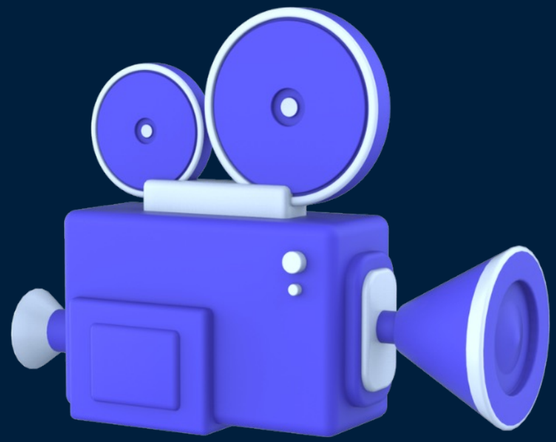
- Official

- Verified Publishers

- Sponsored OSS



- User
  - deepthianarayan/docker
  - thinknyx/jenkins



Demonstration | Docker Hub

Managing images with Docker CLI

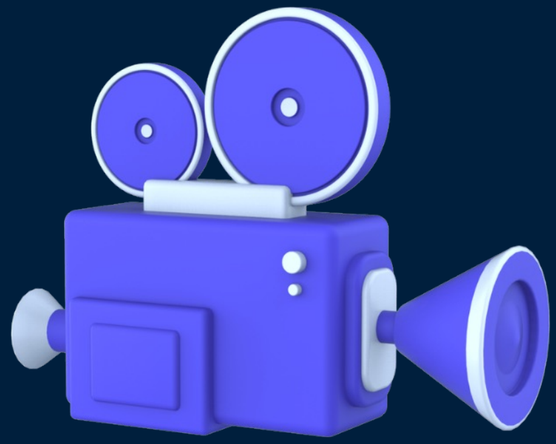
docker images <sub-command> [options]

docker images --help

```
Usage:  docker image COMMAND

Manage images

Commands:
  build       Build an image from a Dockerfile
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  inspect     Display detailed information on one or more images
  load        Load an image from a tar archive or STDIN
  ls          List images
  prune       Remove unused images
  pull        Download an image from a registry
  push        Upload an image to a registry
  rm          Remove one or more images
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

Run 'docker image COMMAND --help' for more information on a command.
```

Managing images with Docker CLI

Description	Command	Alternative command
Download an image	docker image pull <image_name>	docker pull <image_name>
List all the images	docker image ls	docker image list (or) docker images
Tag or rename an image	docker image tag <source_image> <target_image>	docker tag <source_image> <target_image>
Show the history of an image	docker image history <image_name>	docker history <image_name>
Display a detailed information of an image	docker image inspect <image_name>	docker inspect <image_name>
Remove an image	docker image rm <image_name>	docker image remove <image_name> (or) docker rmi <image_name>



Demonstration | Managing images with Docker CLI

Summary

- ✓ Container images
- ✓ Registry
- ✓ Manage images using Docker CLI

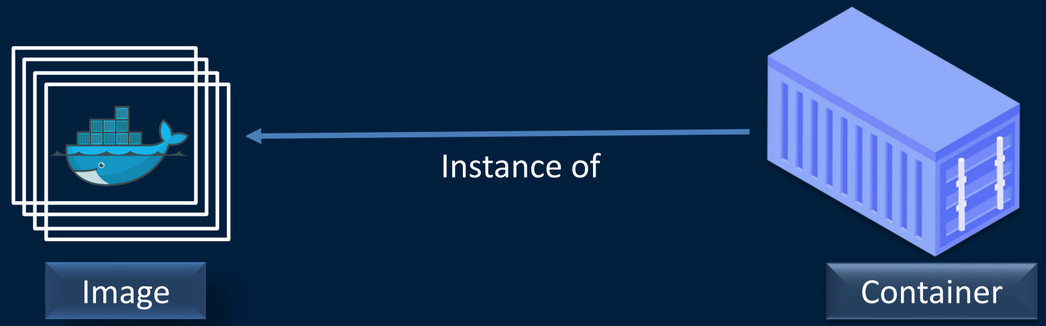
Docker | Docker Containers

Docker Containers

- Containers
- Container Lifecycle
- Manage containers using CLI
- Demonstrations on container commands

Overview of Docker Containers

- Code
- Libraries
- Runtime
- Configurations

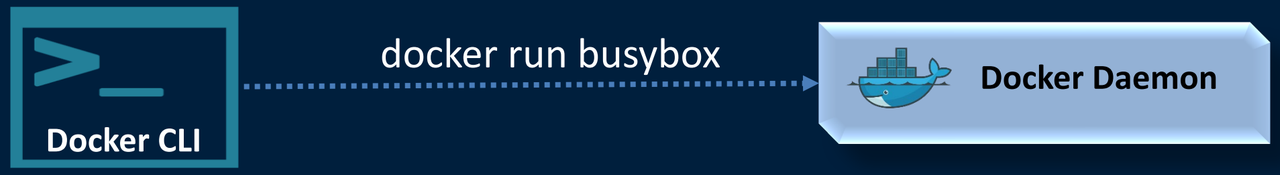


✓ Read-only templates

Overview of Docker Containers

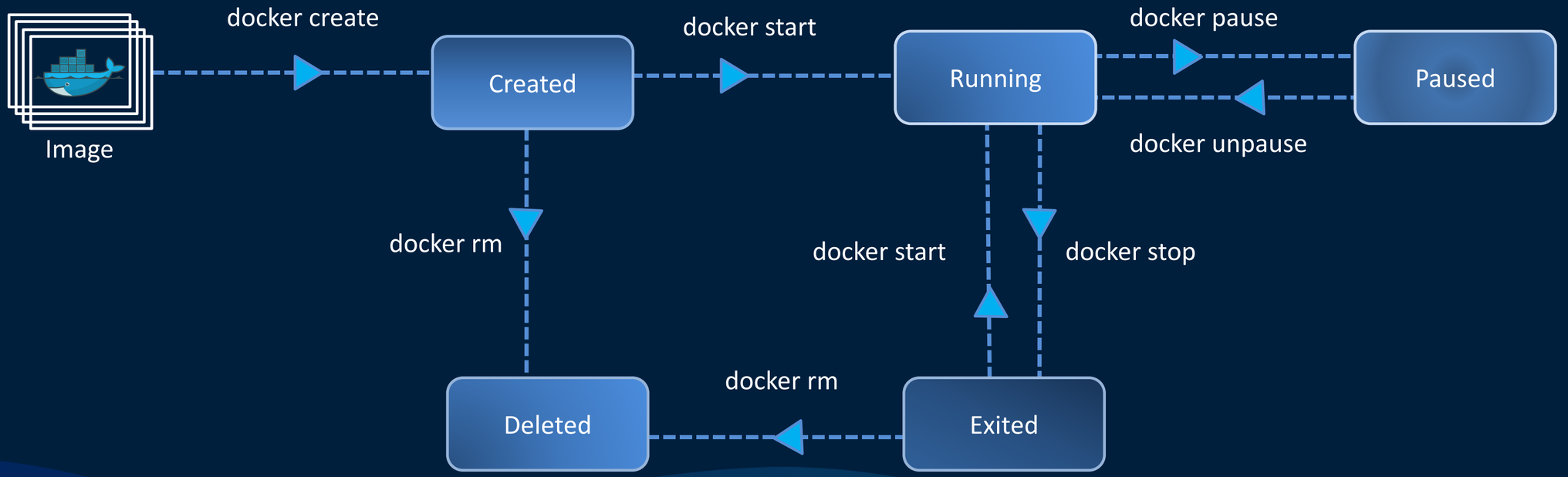
docker run <image_name>

docker run busybox



Overview of Docker Containers

```
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
1fa2a75ab62a   nginx     "/docker-entrypoint.…"   11 seconds ago   Up 9 seconds    80/tcp    dreamy_edison
```



Managing containers with Docker CLI

docker container <sub-command> [options]

docker container --help

```
Usage:  docker container COMMAND

Manage containers

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  exec        Execute a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Create and run a new container from an image
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit codes

Run 'docker container COMMAND --help' for more information on a command.
```

docker container run

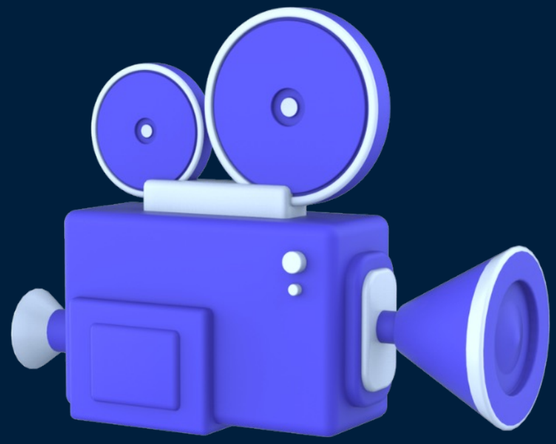
docker run

Managing containers with Docker CLI

Description	Command	Alternative command
Create and start a container	docker container create <image_name> docker container start <container_name>	docker create <image_name> docker start <container_name>
Run a container	docker container run <image_name>	docker run <image_name>
List all the containers	docker container ls -a	docker container list -a docker container ps -a docker ps -a
Stop a container	docker container stop <container_name>	docker stop <container_name>
Execute a command in a running container	docker container exec <container_name> [ARG]	docker exec <container_name> [ARG]
Rename a container	docker container rename <old_container_name> <new_container_name>	docker rename <old_container_name> <new_container_name>

Managing containers with Docker CLI

Description	Command	Alternative command
View the logs of a container	docker container logs <container_name>	docker logs <container_name>
View the detailed information about a container	docker container inspect <container_name>	docker inspect <container_name>
Remove a Container	docker container rm <container_name>	docker container remove <container_name> docker rm <container_name>



Demonstration | Managing containers with Docker CLI

Summary

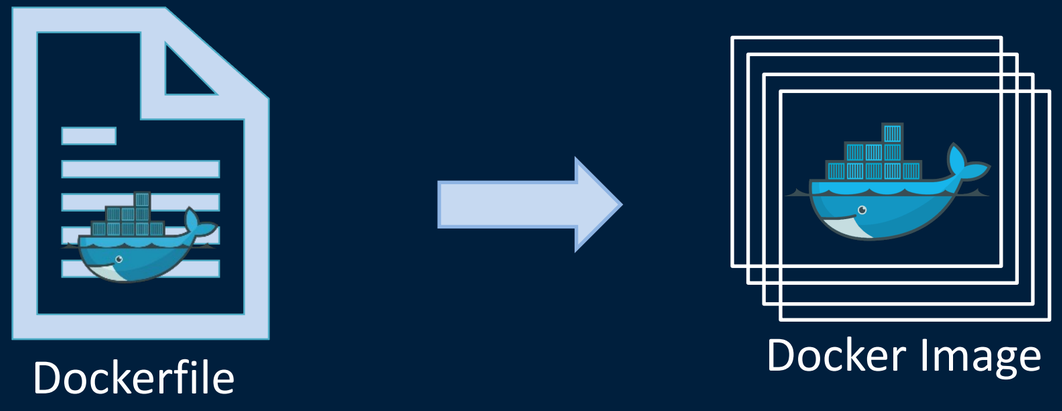
- ✓ Containers
- ✓ Container Lifecycle
- ✓ Manage containers using CLI

Docker | Building an image with a Dockerfile

Building an image with a Dockerfile

- Build an image using a Dockerfile
- Validate the Dockerfile
- Create a container using image

Getting Started with Dockerfile



- Script with instructions to construct the image

Getting Started with Dockerfile

Dockerfile

Steps to construct Docker image

- Setting up environment
- Installing dependencies
- Copying application files

Build process

Images are consistent, reproducible, and ready to deploy across any environment

Getting Started with Dockerfile

INSTRUCTION arguments

```
FROM busybox
```

Dockerfile

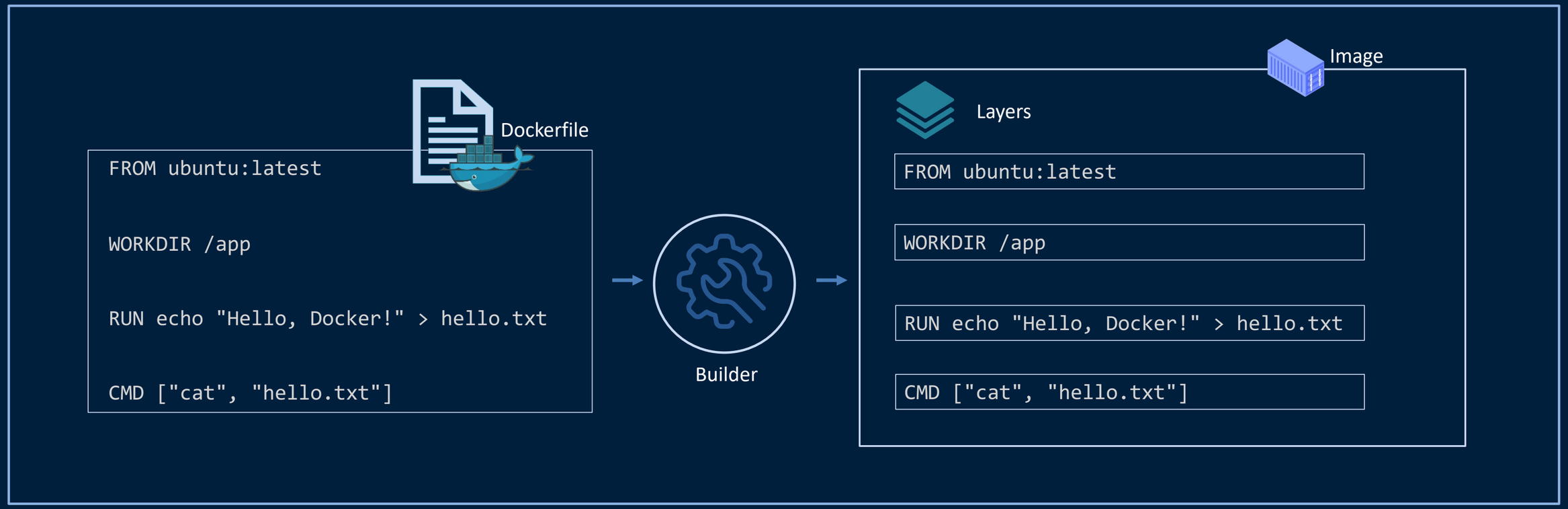
Instruction	Description
ADD	Add local or remote files and directories.
ARG	Use build-time variables.
CMD	Specify default commands.
COPY	Copy files and directories.
ENTRYPOINT	Specify default executable.
ENV	Set environment variables.
EXPOSE	Describe which ports your application is listening on.
FROM	Create a new build stage from a base image.
HEALTHCHECK	Check a container's health on startup.
LABEL	Add metadata to an image.
MAINTAINER	Specify the author of an image.
ONBUILD	Specify instructions for when the image is used in a build.
RUN	Execute build commands.
SHELL	Set the default shell of an image.
STOPSIGNAL	Specify the system call signal for exiting a container.
USER	Set user and group ID.
VOLUME	Create volume mounts.
WORKDIR	Change working directory.

Getting Started with Dockerfile

```
FROM ubuntu:latest

WORKDIR /app

RUN echo "Hello, Docker!" > hello.txt

CMD ["cat", "hello.txt"]
```

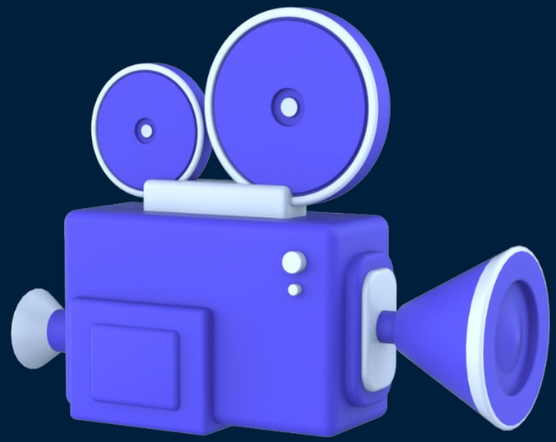
- Dockerfile instructions are executed sequentially
- Each instruction translates to an image layer

Getting Started with Dockerfile

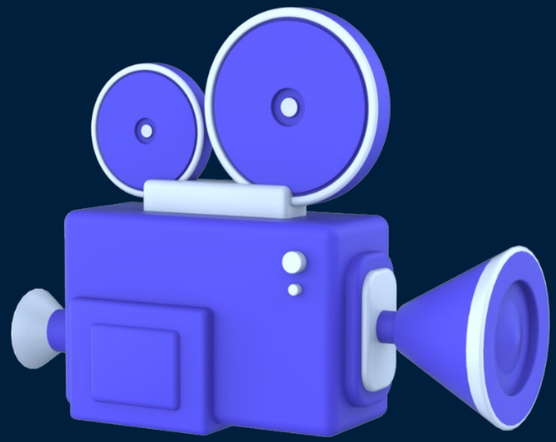


Getting Started with Dockerfile

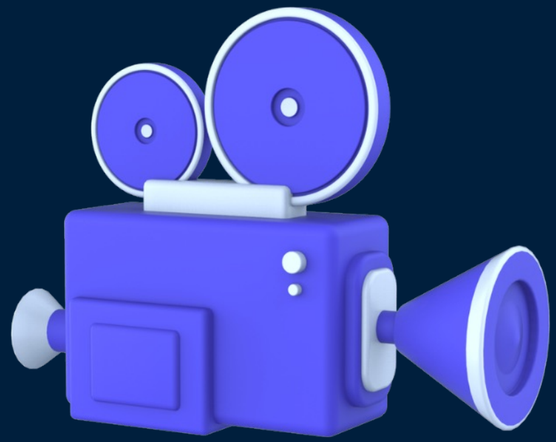
Description	Command	Alternative command
Build an image from a Dockerfile	docker image build -t <image_name> .	docker build (or) docker buildx build (or) docker builder build



Demonstration | Creating a Dockerfile



Demonstration | Validating Dockerfile and building image



Demonstration | Running a Container from our image

Summary

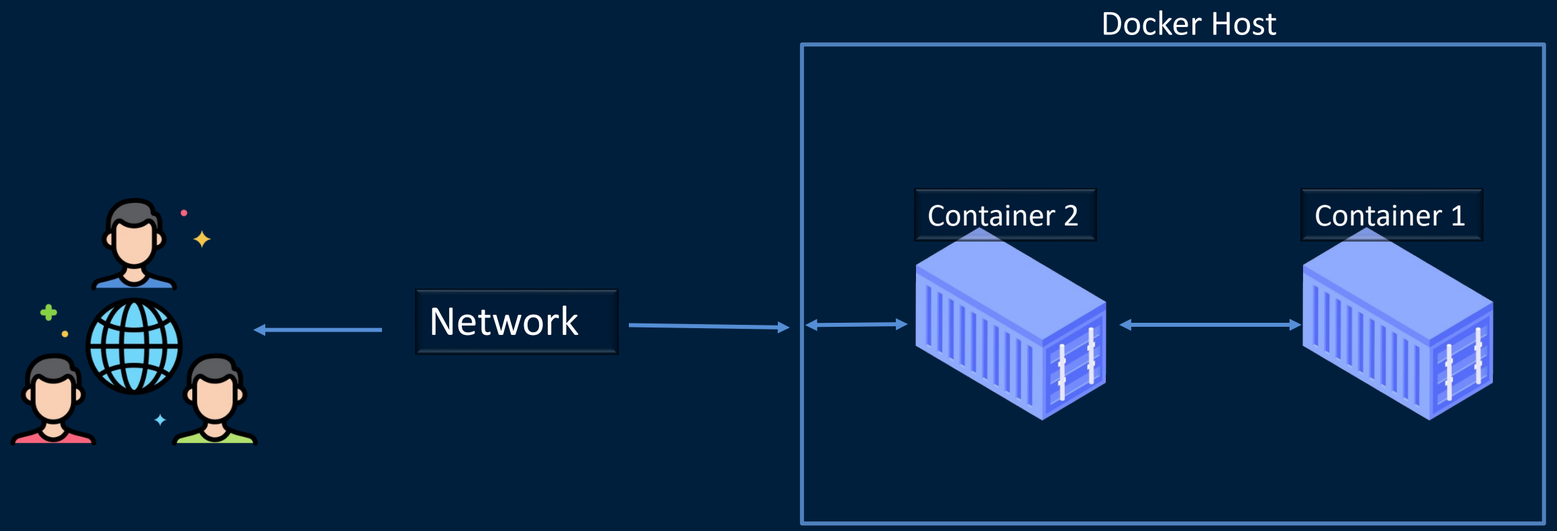
- ✓ Dockerfile instructions
- ✓ Created Dockerfile
- ✓ Validated and built image from Dockerfile
- ✓ Docker layering mechanism
- ✓ Created container using the image

Docker | Docker Networking

Docker Networking

- Container networking
- How networking works?
- Default network drivers
- User-defined networks
- Docker network commands

Overview of Docker Networking



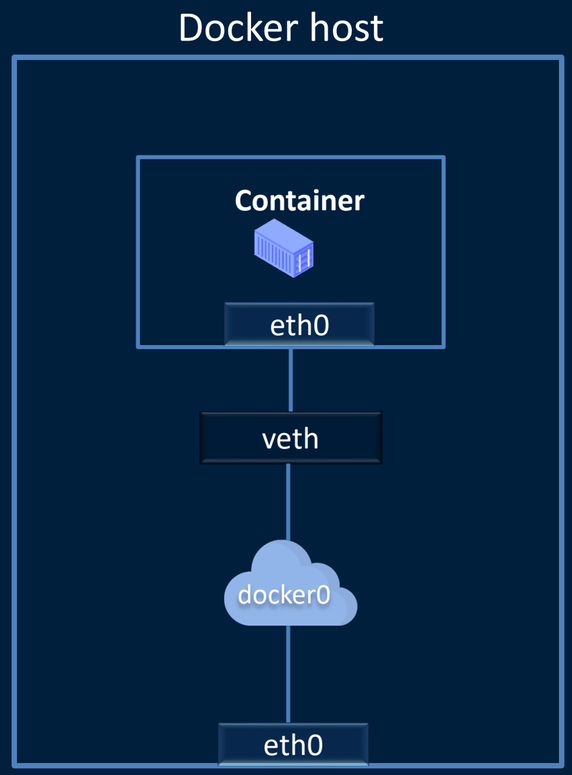
- IP address
- Gateway
- Routing table
- DNS

```
docker container run -d --name=thinknyxcon -p 8081:80 nginx:latest
```

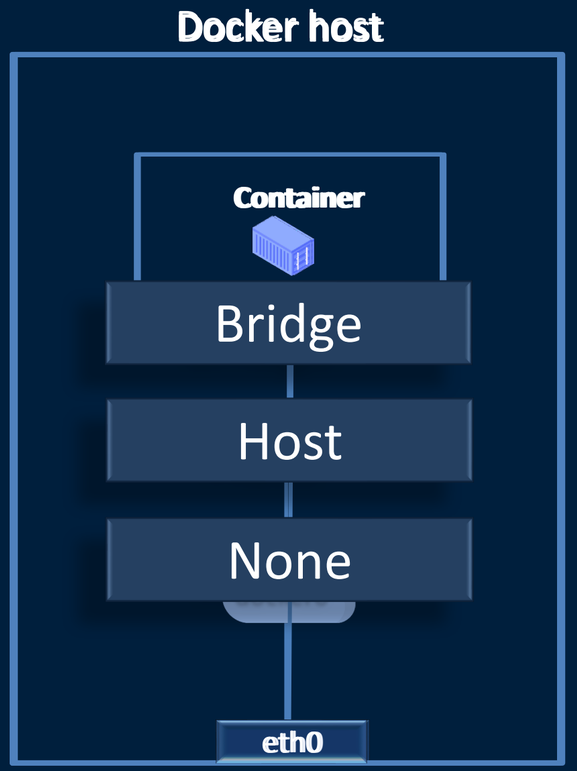
Overview of Docker Networking

- bridge
- host
- overlay
- macvlan
- ipvlan
- none
- User-defined networks

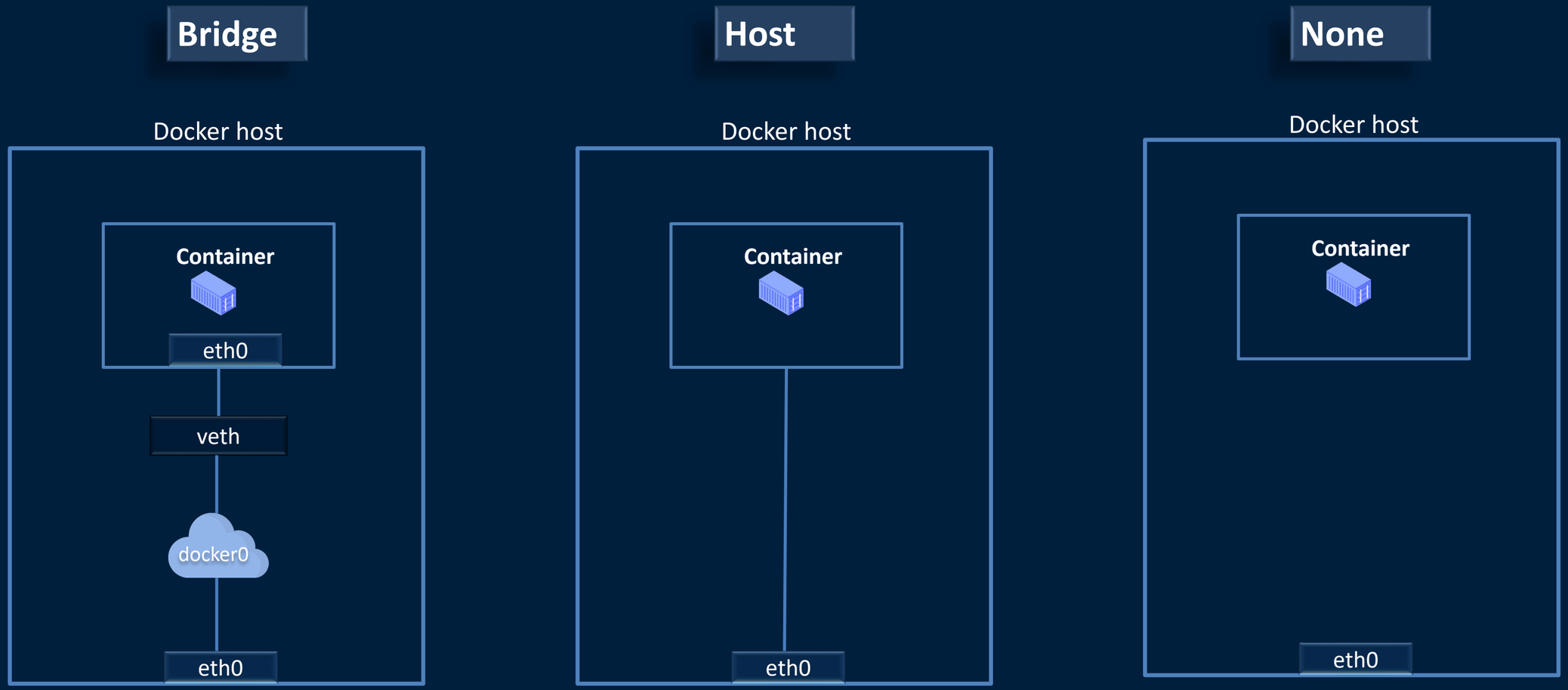
Docker Networking Essentials



Docker Networking Essentials



Docker Networking Essentials



Docker Networking Essentials

overlay

- Multiple Docker daemons to connect
- Allow Swarm services and containers to communicate

macvlan

- Look like real devices
- Ideal for migrating from VMs
- When containers need unique MAC addresses

ipvlan

- Full control over both IPv4 and IPv6 addressing
- When there are limits on MAC addresses

Docker Networking Essentials

```
root@ip-172-31-25-140:~# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
8da72d8b21e4   bridge    bridge    local
64da84f0a68f   host      host      local
ff8257df8370   none      null      local
```

```
root@ip-172-31-25-140:~# docker network create -d ipvlan \
  --subnet=192.168.1.0/24 \
  --gateway=192.168.1.1 \
  -o ipvlan_mode=l2 \
  ipvlan_network
33238cd53d7625c1490581b173306cc1c0071fa1b8372d633d5e0dc968d93183
root@ip-172-31-25-140:~# docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
8da72d8b21e4   bridge            bridge    local
64da84f0a68f   host              host      local
33238cd53d76   ipvlan_network    ipvlan    local
ff8257df8370   none              null      local
root@ip-172-31-25-140:~#
```

Managing networks with Docker CLI

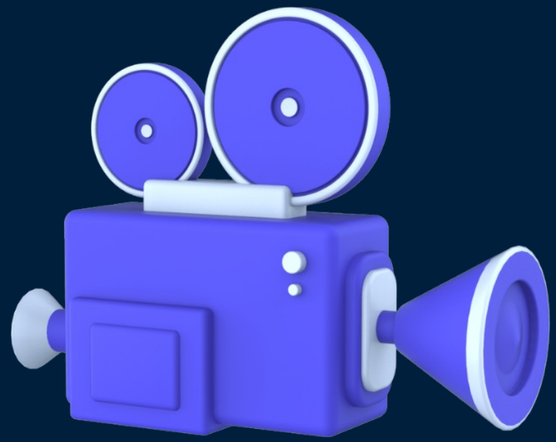
docker network <sub-command> [options]

docker network --help

```
Usage:  docker network COMMAND

Manage networks

Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.
```

Managing networks with Docker CLI

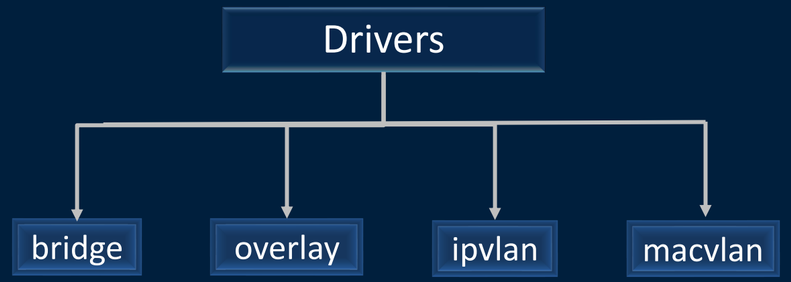
Description	Command	Alternative command
List all the networks	docker network ls	docker network list
Create a network	docker network create <network_name>	-
Display detailed information of a network	docker network inspect <network_name>	-
Remove a network	docker network rm <network_name>	docker network remove <network_name>
Remove all unused networks	docker network prune	-



Demonstration | Managing networks with Docker CLI

User-defined Networks

- Create beyond the default networks
- Advanced networking capabilities
- Help define custom network configurations
- Multiple containers can use the network



works only when docker hosts are in swarm mode

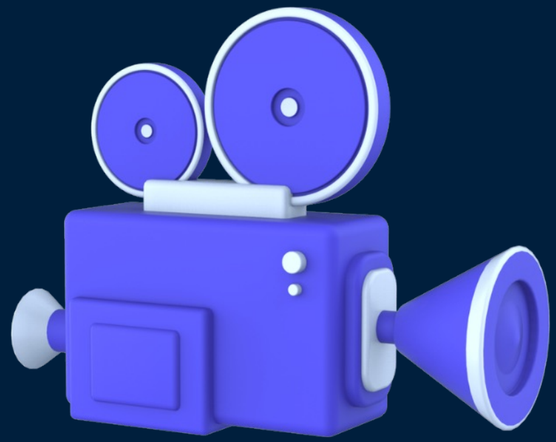
```
docker network create -d bridge thinknyxnet
```

User-defined Networks

	User-defined	Bridge
DNS Resolution	Name or Alias	IP addresses
Isolation	Better isolation	Risk of unrelated services/containers communicating
Configuration	Using docker network create	Requires a restart of Docker

```
docker container run -d --name=thinknyxcon --network=thinknyxnet -p 8085:80 nginx
```

- overlay
- ipvlan
- macvlan



Demonstration | User-defined Networks

Summary

- ✓ Fundamental concepts of networking
- ✓ Different types of networks
- ✓ Demonstrations on how networking works

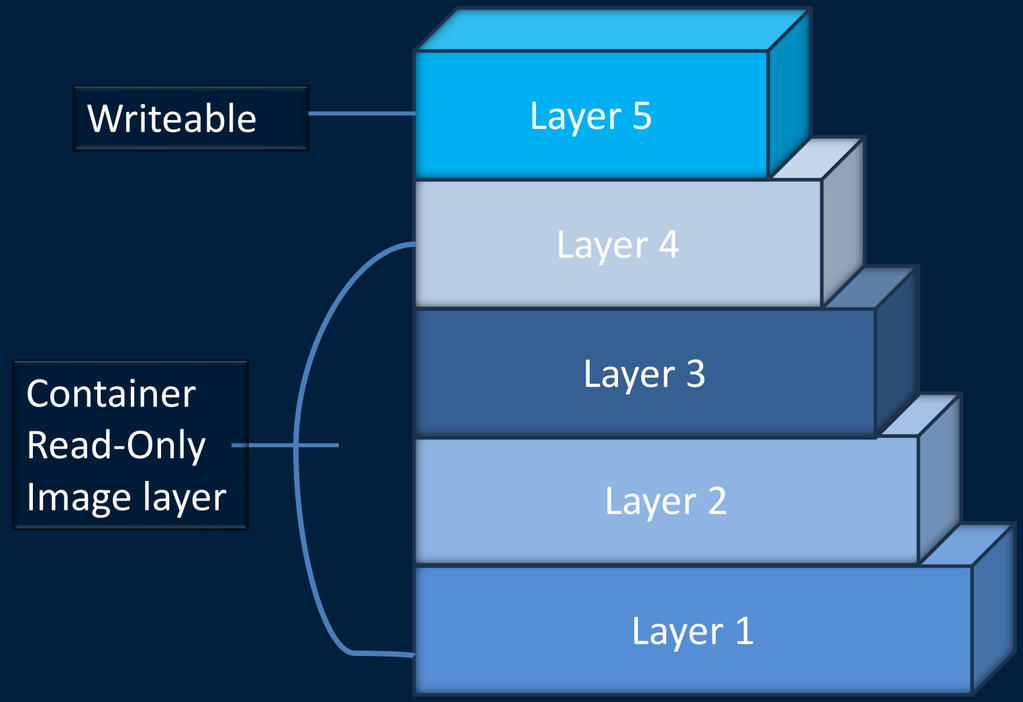
Docker | Docker Volumes

Docker Volumes

- How volumes work
- Types of volumes
- Manage and mount using CLI
- Hands-on examples

Data Storage in Docker containers

Ephemeral and disposable



Data Storage in Docker containers

Data can be made permanent → Adding that data to the image

- ✖ Not ideal and practical solution
- ✖ Increase in the size of the image
- ✖ Tightly coupled
- ✖ Performance issues

- ❑ Volumes
- ❑ Bind mounts

- ✔ Independent of container lifecycle
- ✔ No increase in the size of the image

Data Storage in Docker containers

❑ Volumes

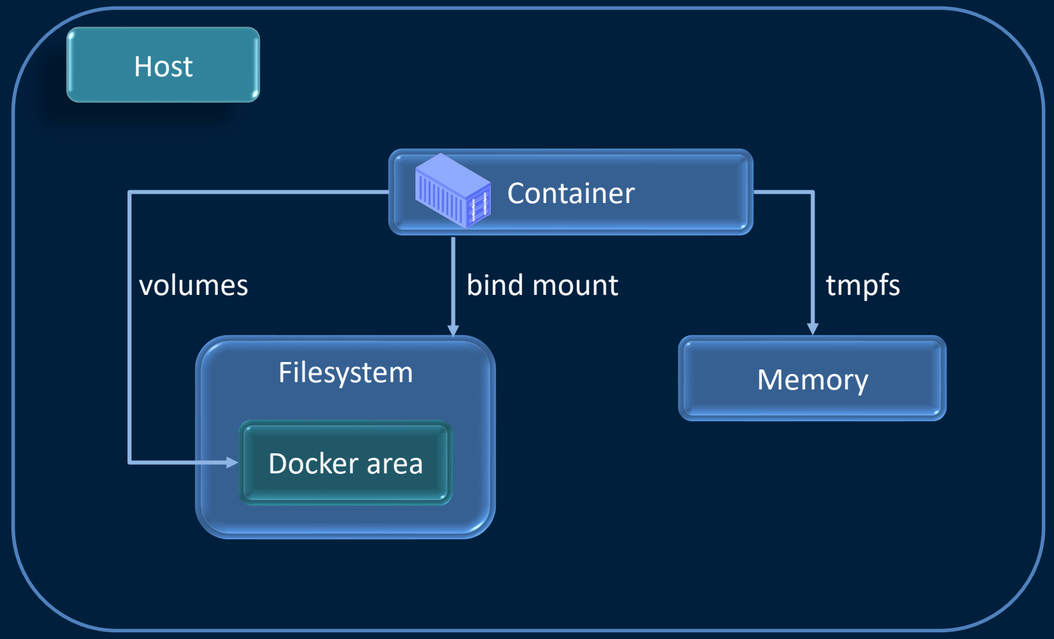
Database files or application logs
Backup, restore and migrate
Long-term data management

❑ Bind mounts

Map a directory or file from the host's filesystem to container
✔ Flexible and portable
✘ Careful configuration
✔ Dev environments

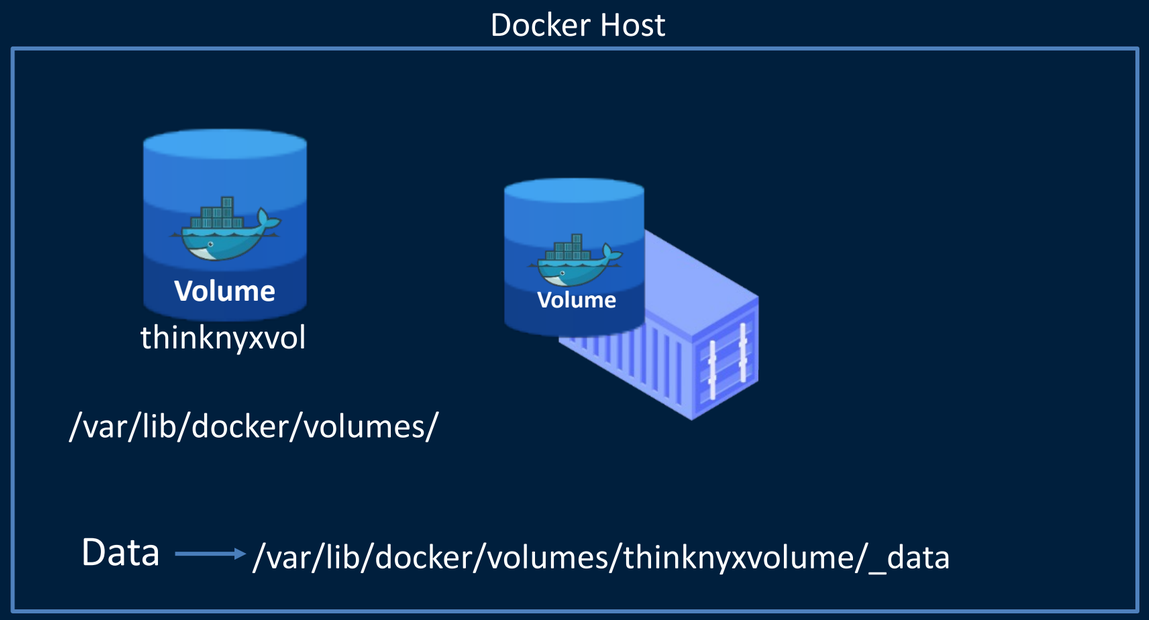
❑ tmpfs

tmpfs for Linux and named pipes for windows
✔ in-memory solutions
✔ Fast
✘ Temporary

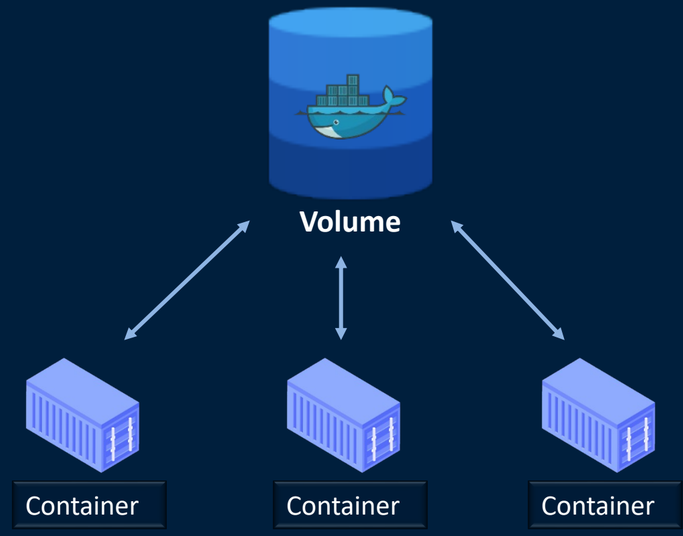


Overview of Docker Volumes

Persist data beyond container lifecycle ✔



Default Location = /var/lib/docker/overlay2/



Overview of Docker Volumes

- Named Volume
  - User provides specific name
  - Unique within the host
  - Easy to identify
  - Shared with multiple containers

- Anonymous Volume
  - Do not have a name
  - Created by Docker automatically
  - Randomly generated unique name and ID
  - Created when container is created
  - Manually mount to share across multiple containers
  - Persist after container is removed unless --rm flag is used

```
docker volume create thinknyxvol
```

```
docker container run -v thinknyxvol:/usr/share/nginx/html nginx
```

```
docker container run -v /usr/share/nginx/html nginx
```

Managing volumes with Docker CLI

docker volume <sub-command> [options]

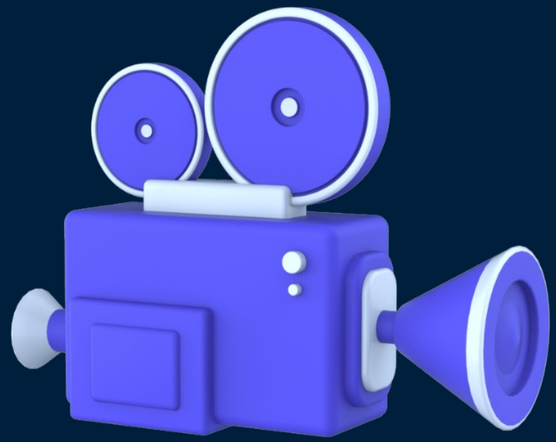
docker volume create

docker volume --help

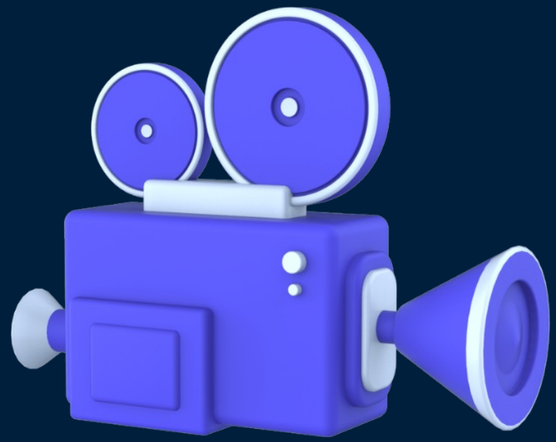
```
Usage:  docker volume COMMAND

Manage volumes

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove unused local volumes
  rm          Remove one or more volumes
  update      Update a volume (cluster volumes only)

Run 'docker volume COMMAND --help' for more information on a command.
```

Managing volumes with Docker CLI

Description	Command	Alternative command
List all volumes	docker volume ls	docker volume list
Create a volume	docker volume create <volume_name>	-
View detailed information about a volume	docker volume inspect <volume_name>	-
Mount a named volume	docker container run --mount source=<volume_name>,target=<container_path> <image_name>	docker container run -v <volume_name>:<container_path> <image_name>
Mount an anonymous volume	docker container run --mount target=<container_path> <image_name>	docker container run -v <container_path> <image_name>
Remove a volume	docker volume rm <volume_name>	docker volume remove <volume_name>



Demonstration | Managing volumes with Docker CLI



Demonstration | Persisting data with Docker volumes

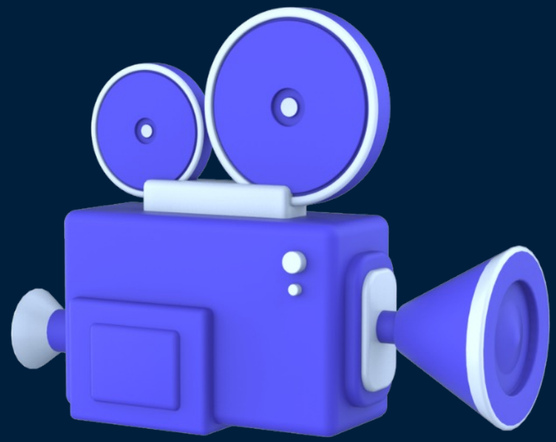
Summary

- ✓ Storage options in Docker
- ✓ Docker volumes
- ✓ Docker volume types
- ✓ Create, mount and delete volumes
- ✓ Practical demonstrations:
  1. Managing volumes using CLI
  2. How to persist data using volumes

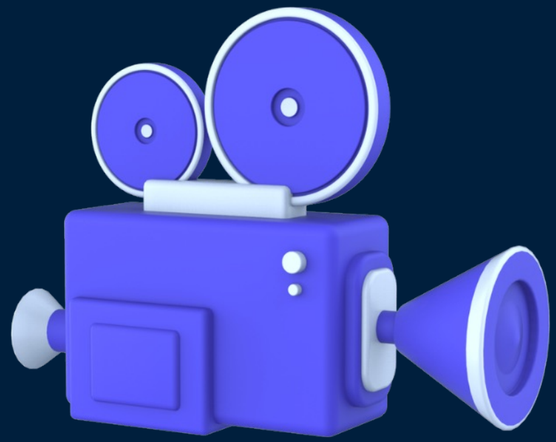
Docker | Containerizing Applications

Containerizing Applications

- How to containerize a python application
- Create a Dockerfile
- Publish image to Docker Hub
- Pull and run the image
- Benefits of multi-stage builds



Demonstration | Creating Dockerfile for the app



Demonstration | Multi-stage Builds

Single and Multi-stage builds

Single-stage

```
FROM base_image
WORKDIR /app
COPY . .
RUN build_commands
CMD ["run_command"]
```

- ❌ Large
- ❌ Inefficient
- ❌ Possibly insecure
- ❌ Complex and error-prone

Multi-stage

```
# Build Stage
FROM base_image AS buildstage
WORKDIR /app
COPY . .
RUN build_commands

# Final Stage
FROM base_image AS finalstage
WORKDIR /app
COPY --from=buildstage /app/output .
CMD ["run_command"]
```

Stage1

Stage2

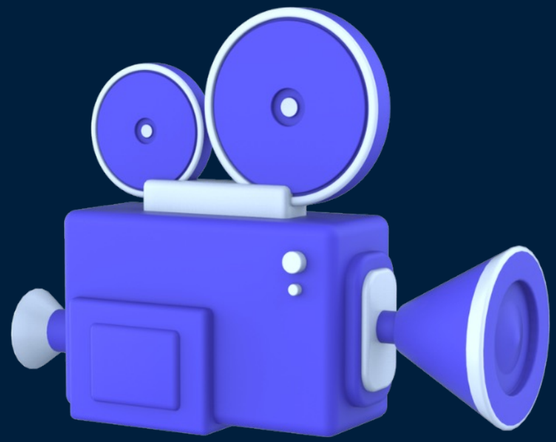
Build

- Application is compiled
- Dependencies are installed
- Necessary build tools are used

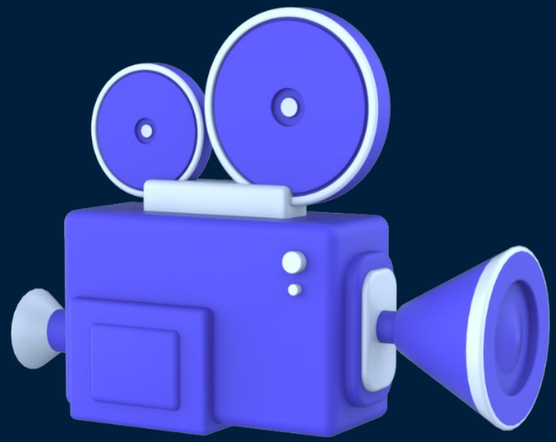
Runtime

- Necessary artifacts are included

- ✔ Smaller
- ✔ More secure
- ✔ Efficient



Demonstration | Publishing to a Registry



Demonstration | Real time application deployment

Summary

Entire Use Case

- ✓ Containerized the Python application
- ✓ Created a Dockerfile
- ✓ Optimized the image further
- ✓ Pushed image to Docker Hub
- ✓ Pulled and run the image