Cloud Computing Technologies

DAT515 - Fall 2024

Docker

Prof. Hein Meling

University
of Stavanger


docker
docker
docker
docker
docker
CONTAINER
KING
KING
docker
16:9
docker
docker
docker

What is Docker?

Docker

- Docker was introduced by Solomon Hykes in 2013
- Initially seen as yet another virtualization tool
- But Docker is a containerization technology
- Revolutionized development and deployment workflows
- Automating and streamlining the deployment process

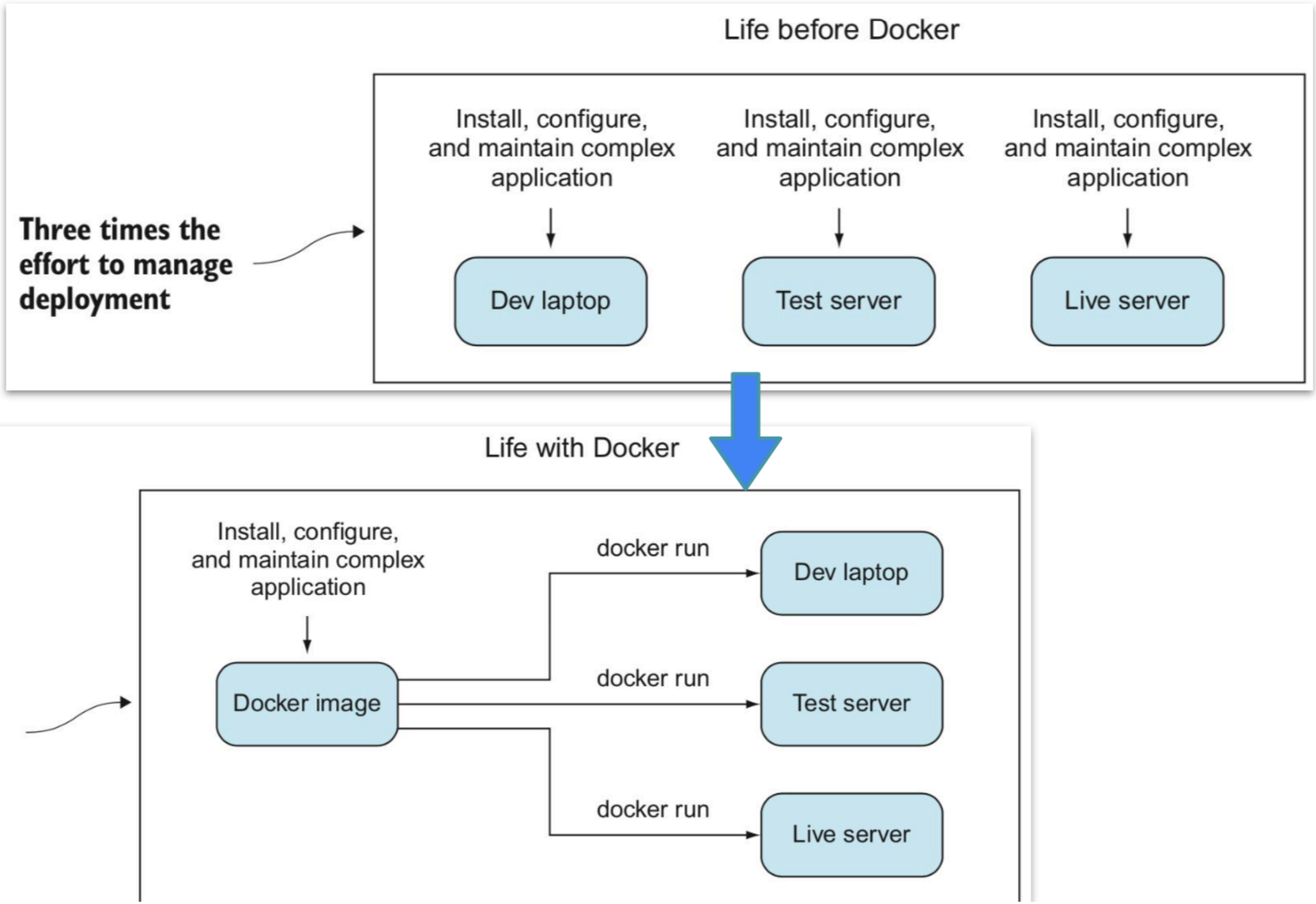
The word “Docker” comes from a British colloquialism meaning dock worker — somebody who loads and unloads ships.

Why is Docker Important?

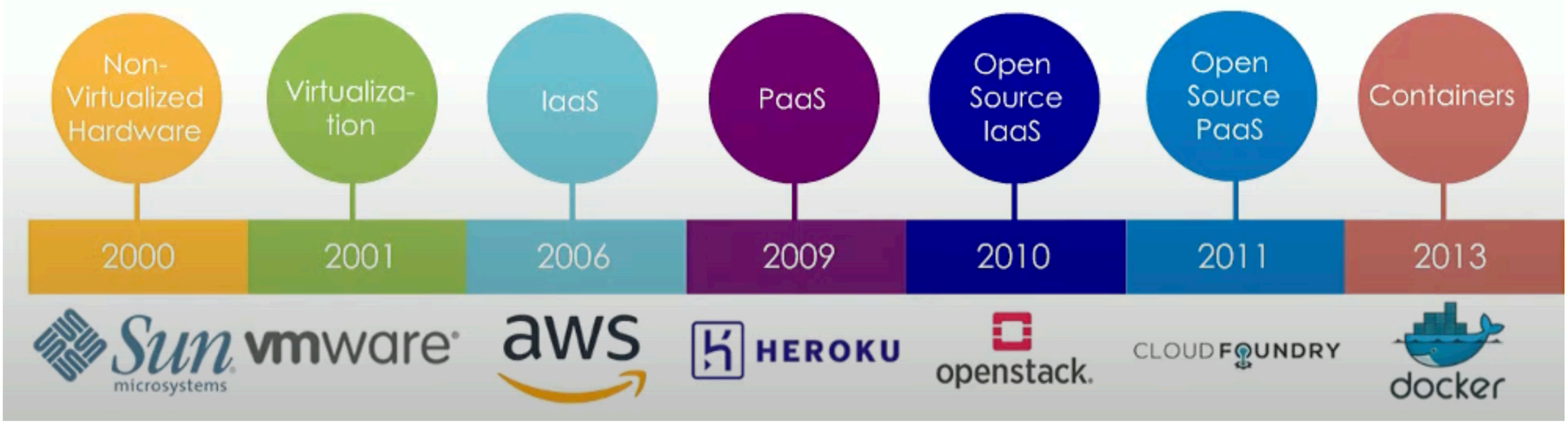
- Docker provides an abstraction to simplify development and deployment
- Cross platform and open source
  - Runs on Windows, macOS, and Linux…
  - Runs on several processor architectures (amd64, arm64, …)
- Community and company adoption
  - Amazon, Microsoft, and Google

Life with Docker

Docker



Another View of History



Docker has become synonymous with Containers

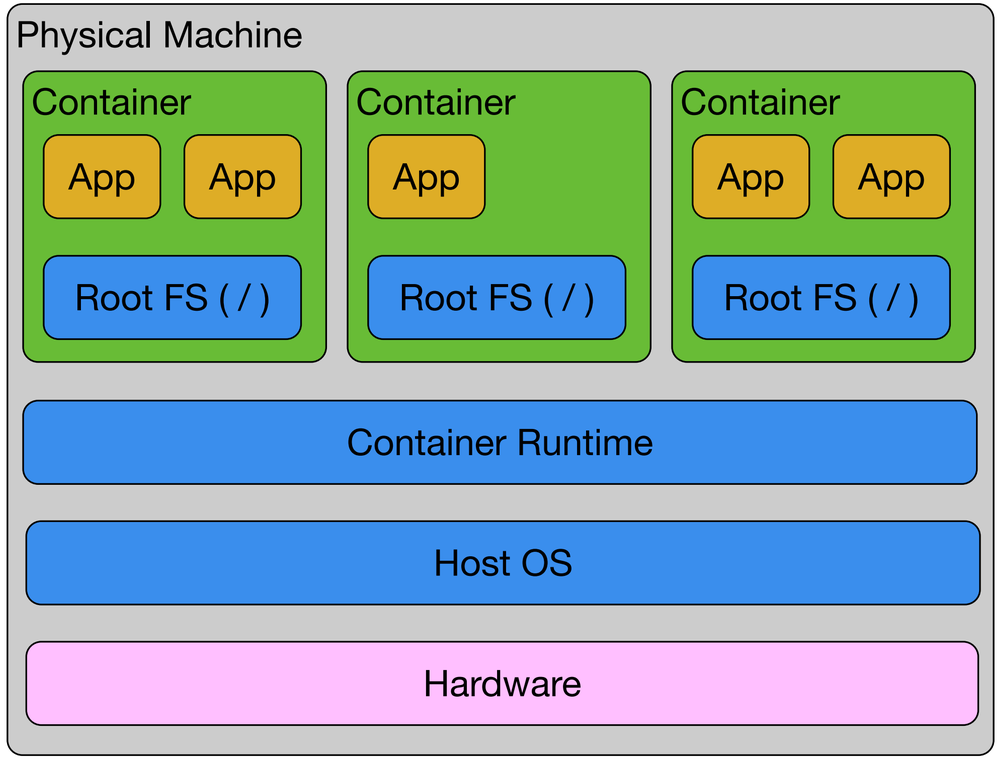
What is Docker, Technically?

Containers Recap

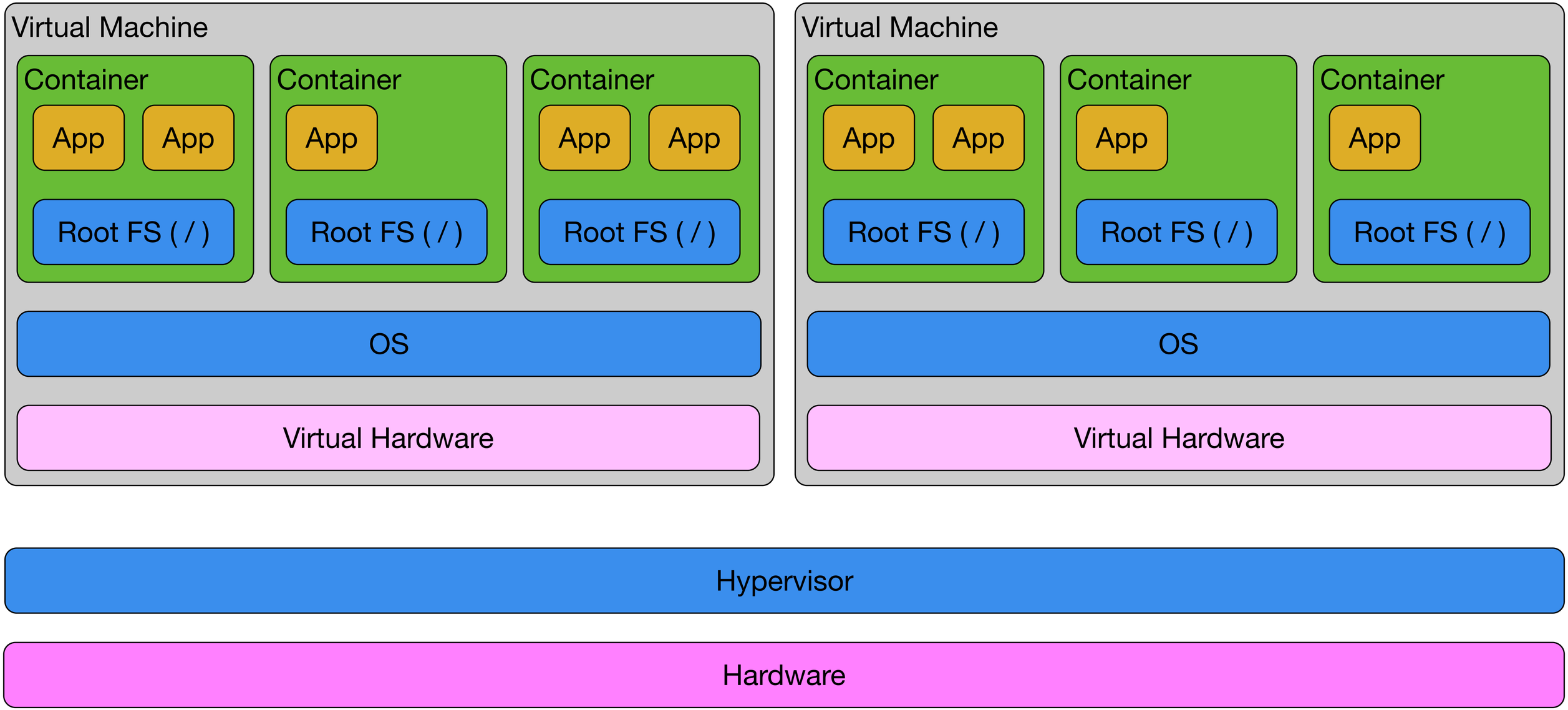
Lightweight, standalone, and executable software package

- Application code, Configuration files, Runtime libraries, Dependencies
- Pre-configured runtime environment
- Executable with a single command

Write once, run anywhere!



Containers inside Virtual Machines



Docker vs Containers

What's the Difference?

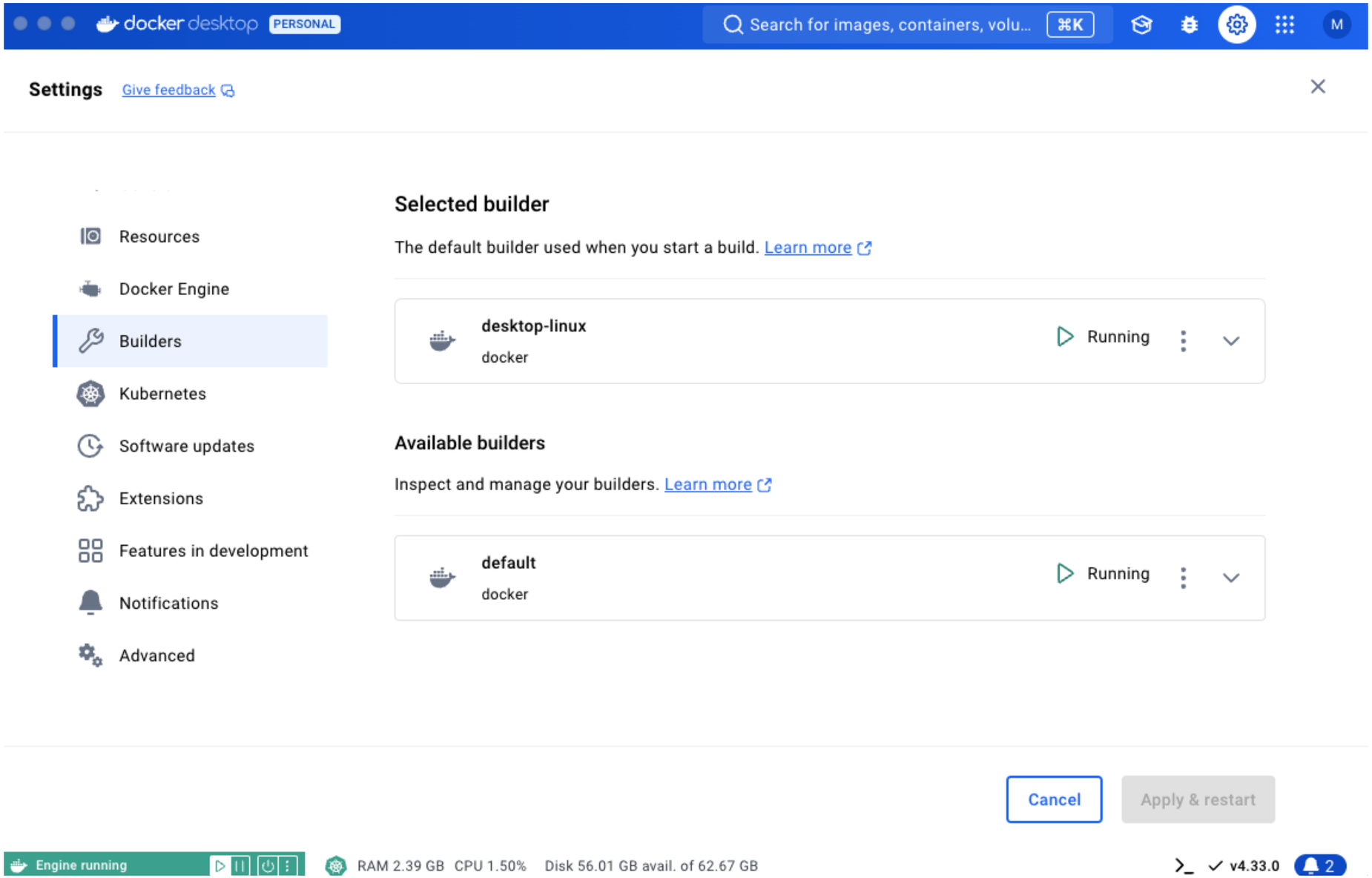
- **Containers:** lightweight packaging of an application and its dependencies
  - Can run consistently across different environments
  - Share the host system's kernel, making them more efficient
- **Docker** is a platform and set of tools
  - Simplify the creation, deployment, and management of containers
  - Easy-to-use interface, standardized container format
  - Comprehensive ecosystem for building, distributing, and running containers

Docker vs LXC

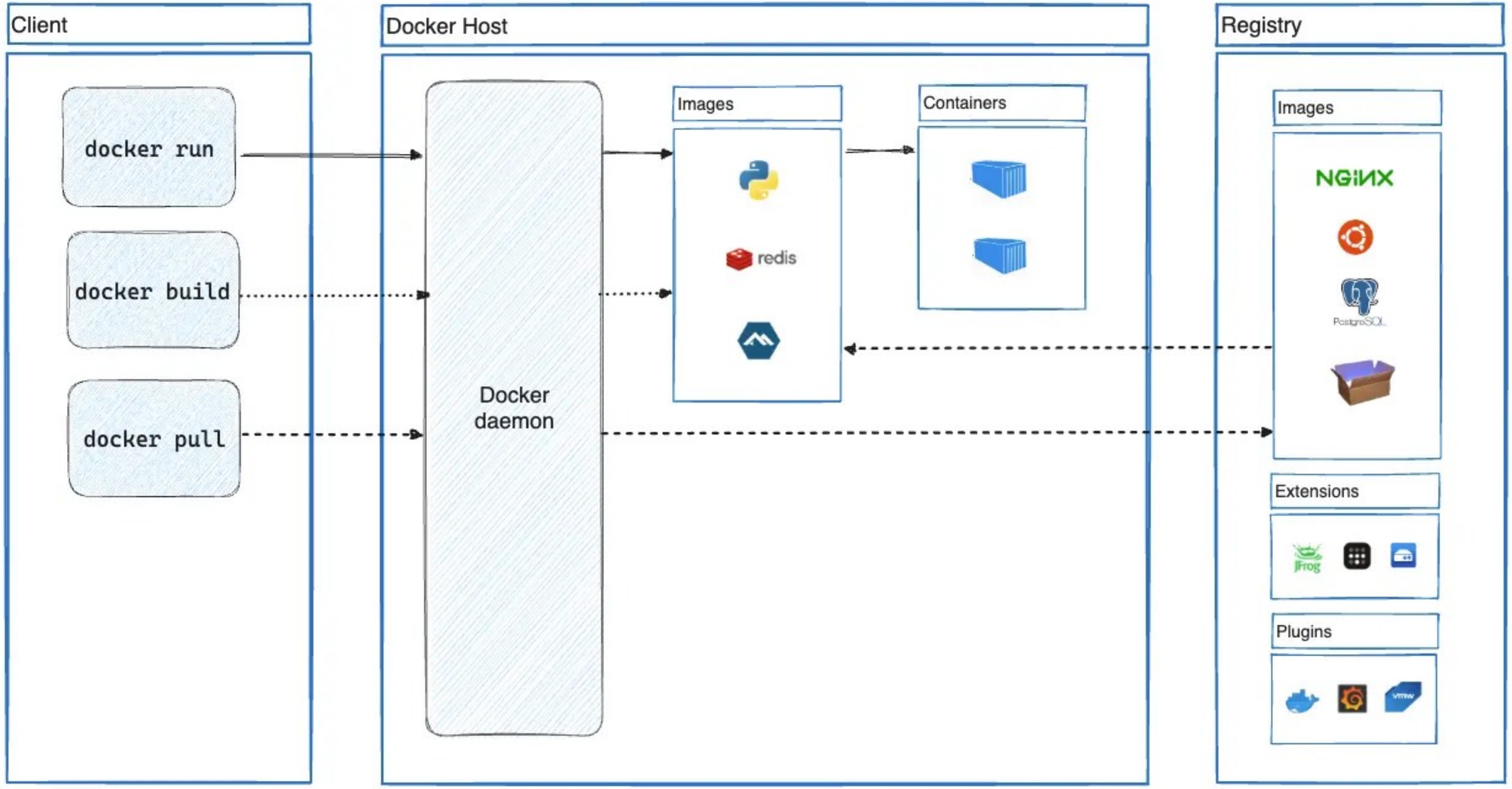
What's the Difference?

- Docker is based on the **runC** container runtime
  - Portable and compliant with the Open Container Initiative (OCI) standards
- LXC is tightly coupled to Linux environments (not portable)
- Docker is more than LXC
  - Command line tools and Desktop Client

Docker Desktop



Docker Architecture



Joke time

Why did the Docker container break up with the virtual machine?

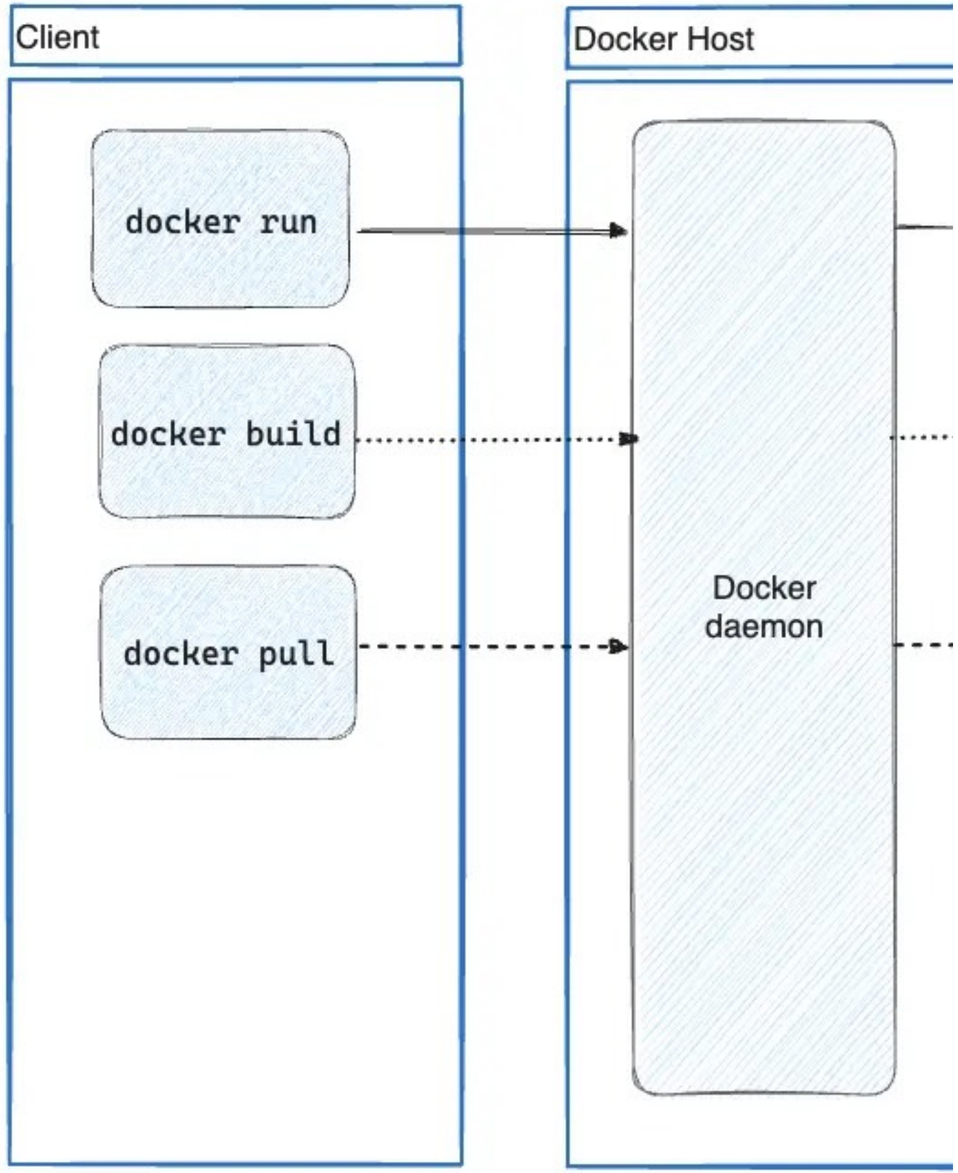
It needed some space and felt they were too heavy

Docker Components

Docker Engine

The Docker Daemon (Server)

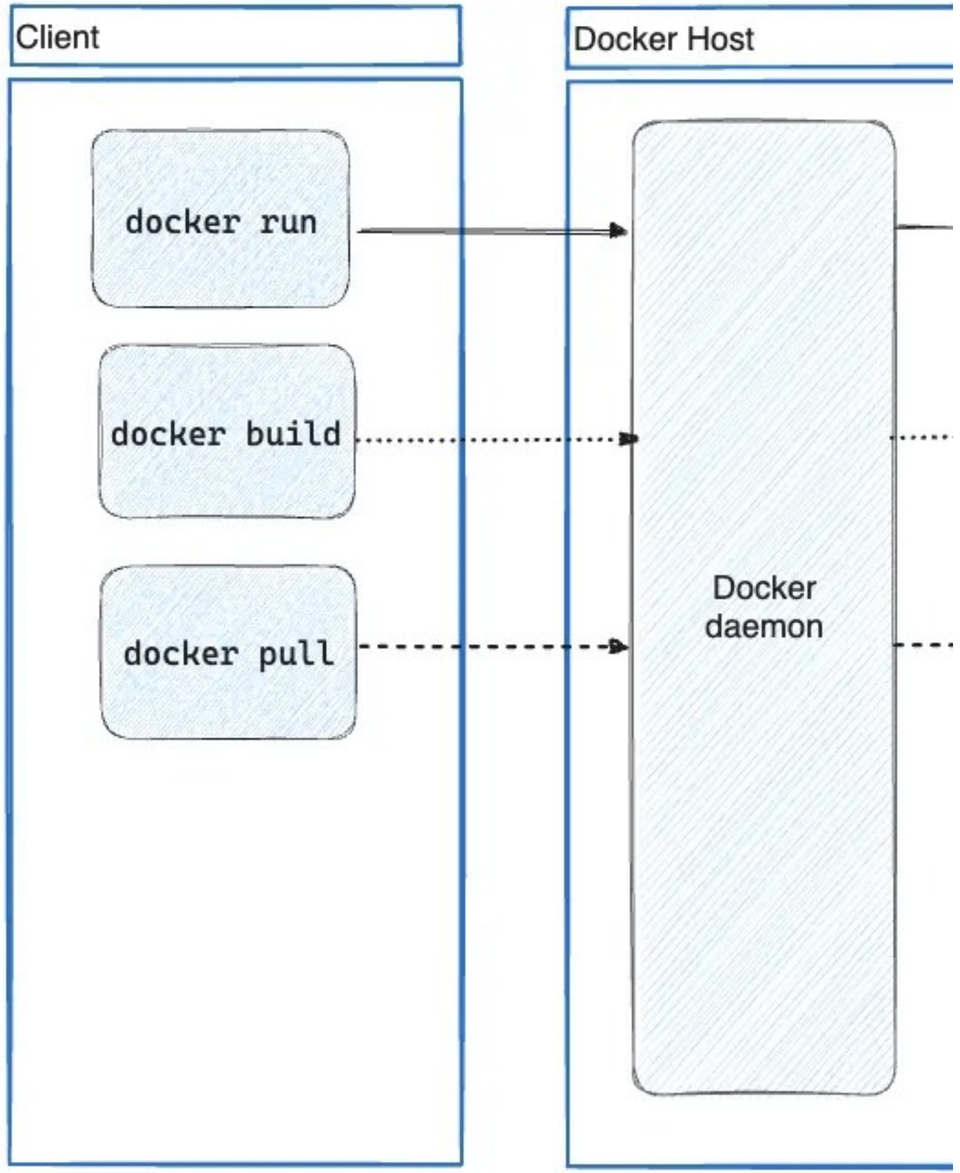
- Runs and manages containers on a host system
  - Including the container runtime
- Client-Server Architecture
  - Clients sends commands to the daemon which handles container operations
- Supports building, running, and managing containers



Docker Command

The Docker Command Line Client

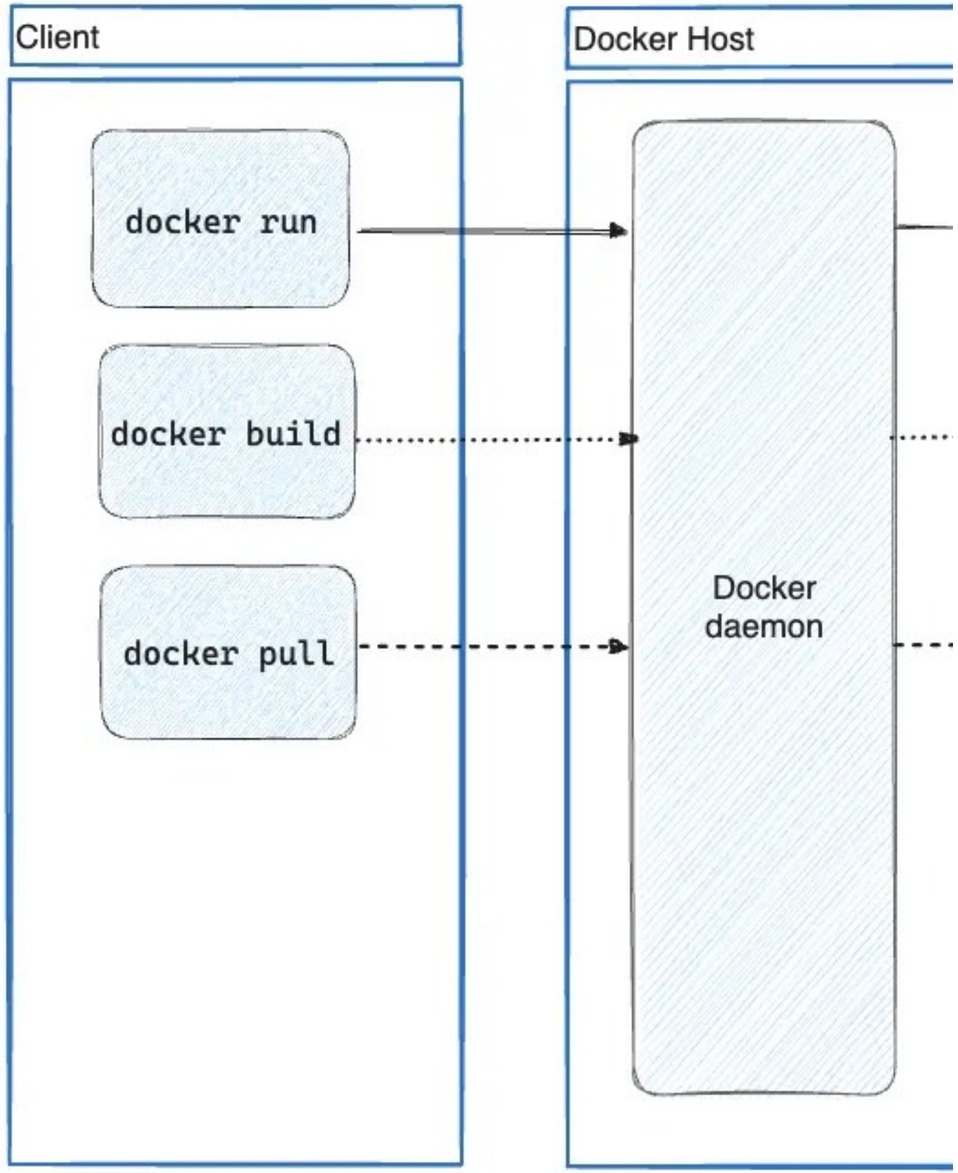
- Docker command uses the Docker APIs
  - Interact with the Daemon
- Docker is written in Go

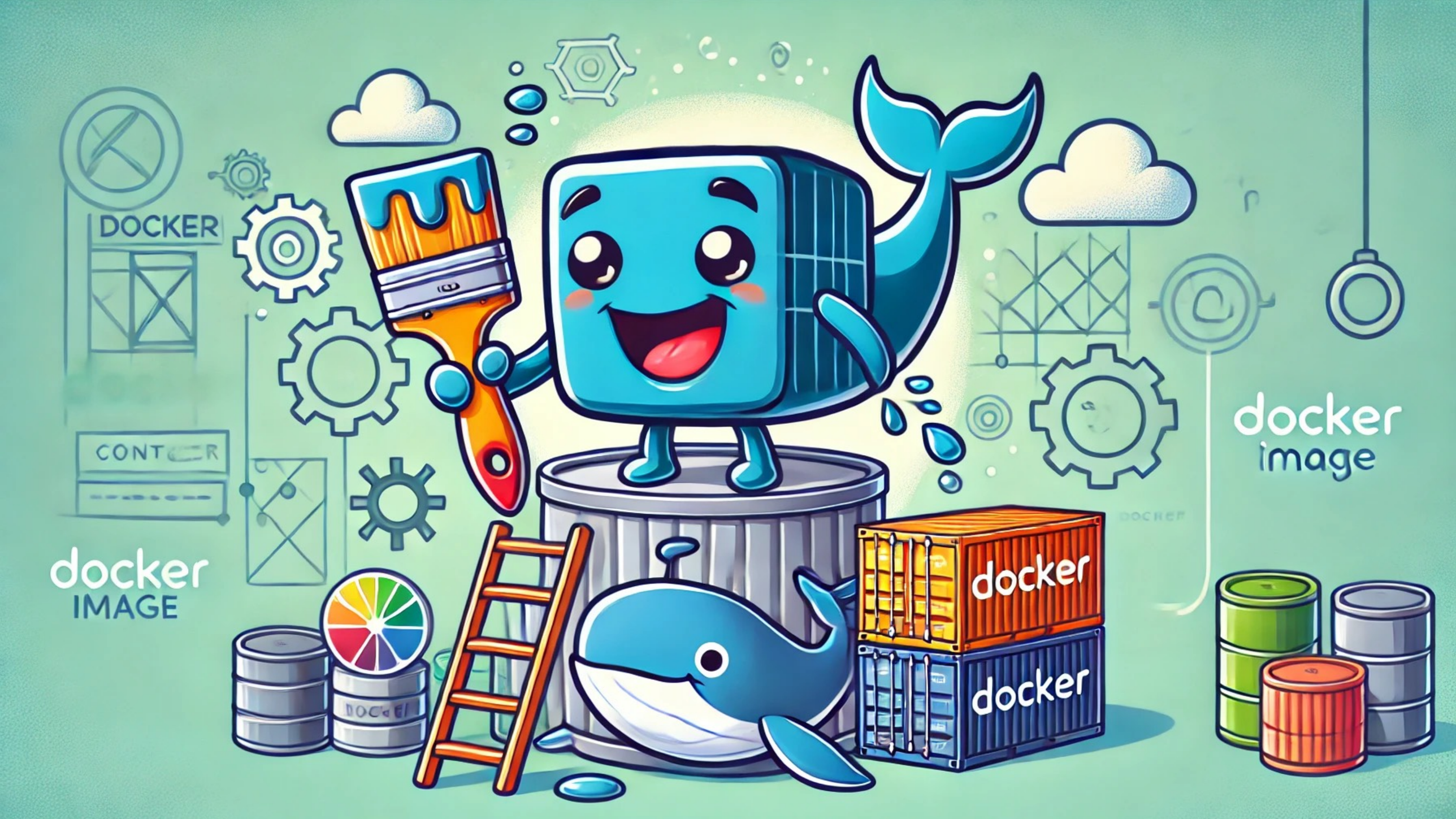


Docker API

REST-based API

- Client libraries in most languages
  - Go SDK:
    - go get github.com/docker/docker/client
  - Python
    - pip install docker
  - Unofficial: C#, Java, Ruby, Node.js, …

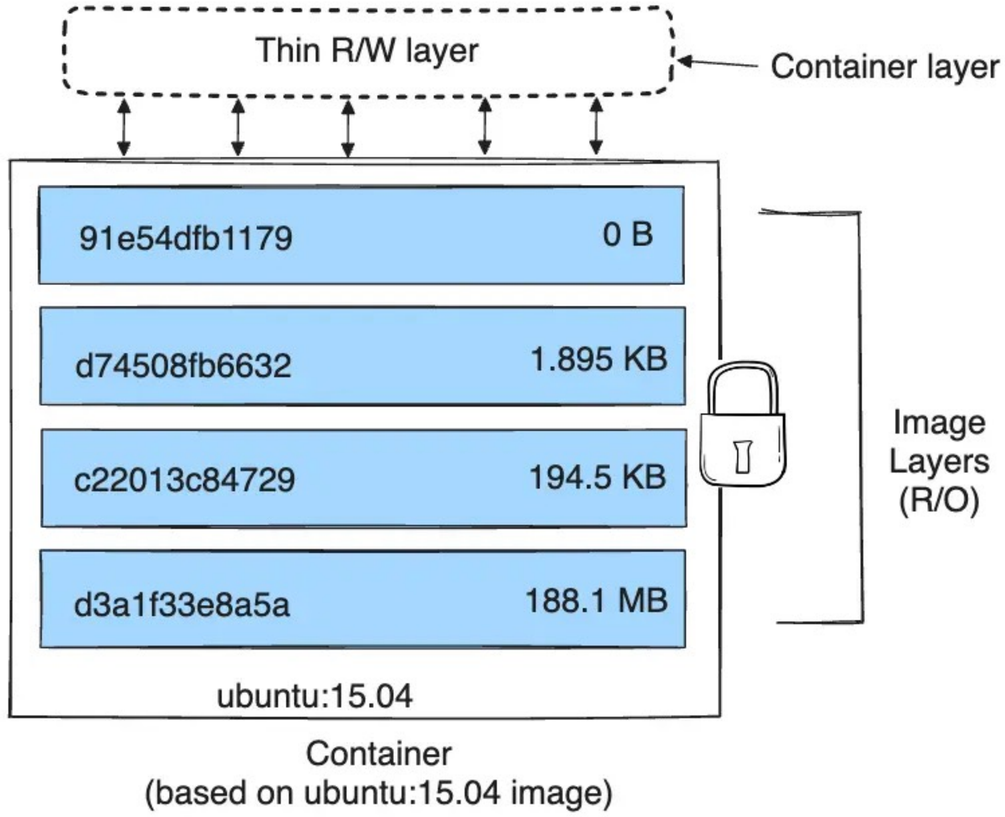



DOCKER
docker
IMAGE
docker
image
docker
docker

Docker Image

Immutable snapshot of an application and its dependencies

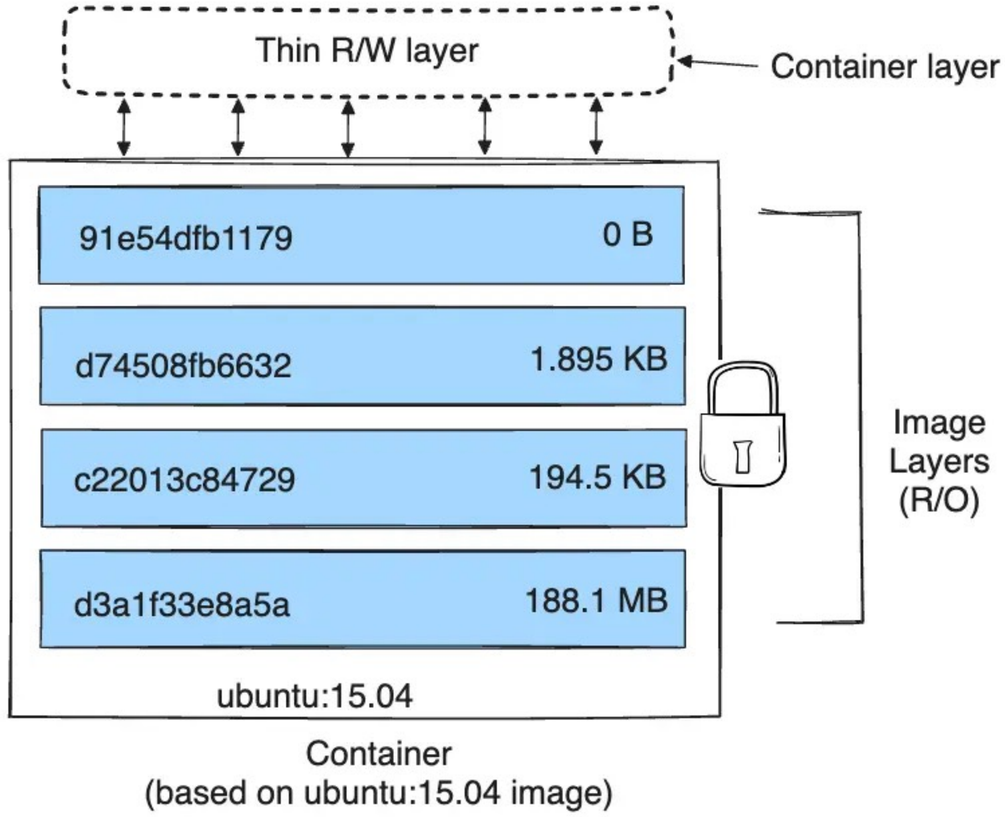
- Used to create containers
- Composed of layers
  - Each layer represents a change or addition to a base layer
  - Enhancing efficiency through caching



Docker Image

Immutable snapshot of an application and its dependencies

- Reusable: Images can be shared and reused across environments and projects
  - Images are built from scratch or pulled from repositories
- Versioned: Tagged with versions
  - Ease management and rollback



How to Identify a Docker Image?

Immutable snapshot of an application and its dependencies

- Format: **[repository]:[tag]**
- Repository Name
  - Examples: nginx, golang, python, myuser/myapp
- Tag
  - Default: latest
  - Examples: nginx:1.19, golang:1.23.0-alpine3.20

How to Identify a Docker Image?

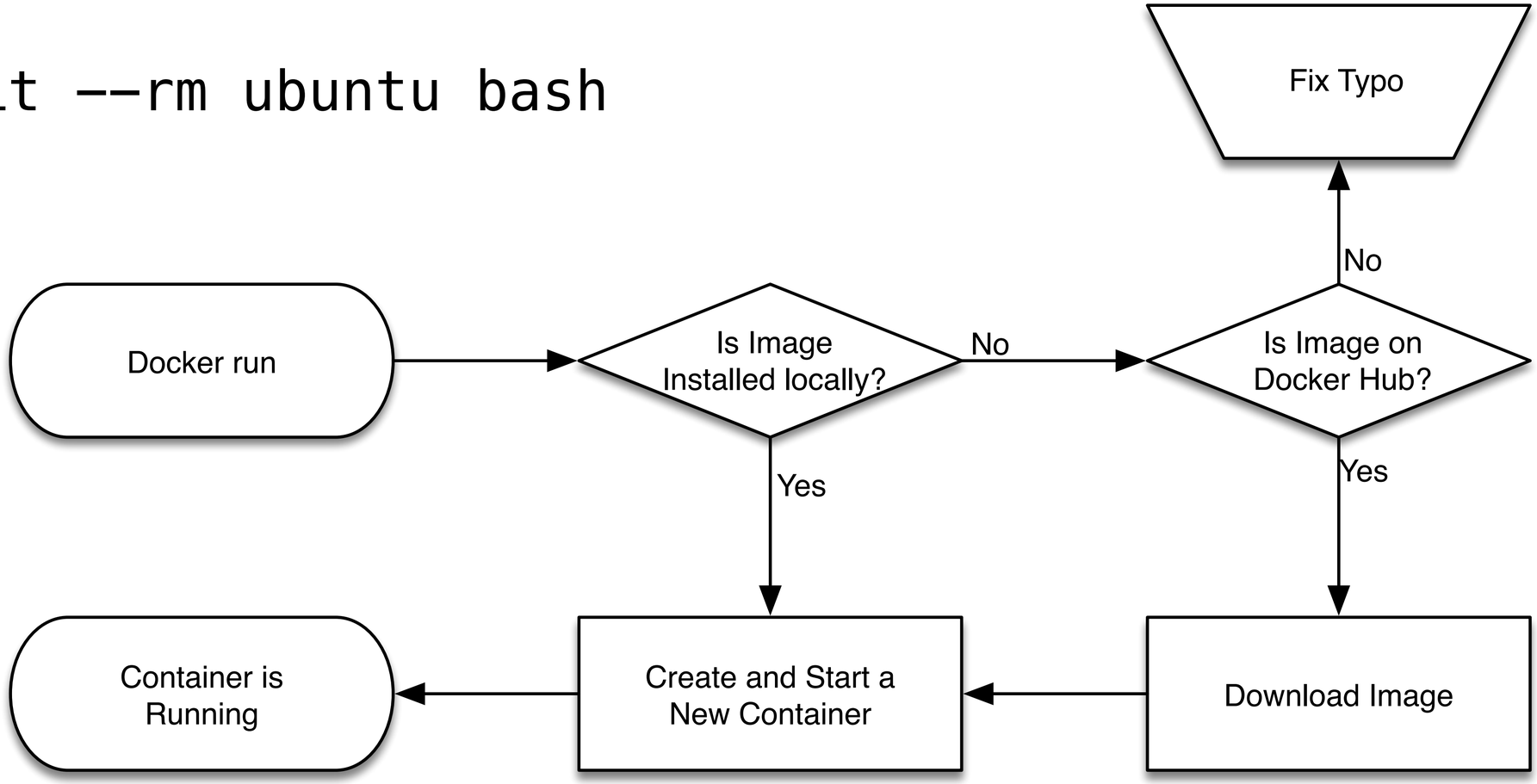
Immutable snapshot of an application and its dependencies

```
docker pull golang

docker pull golang:1.20-alpine

docker pull python

docker images
```

How to Identify a Docker Image?

Immutable snapshot of an application and its dependencies

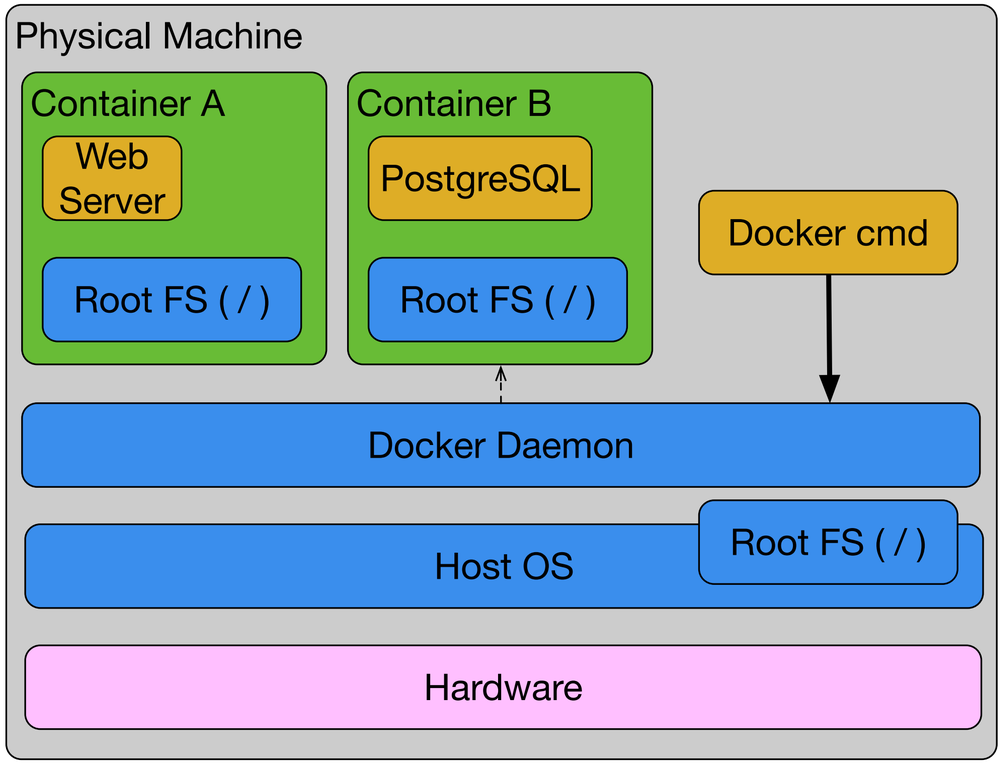
```
docker run -it --rm ubuntu bash
```



Docker Container

Running instance of an image

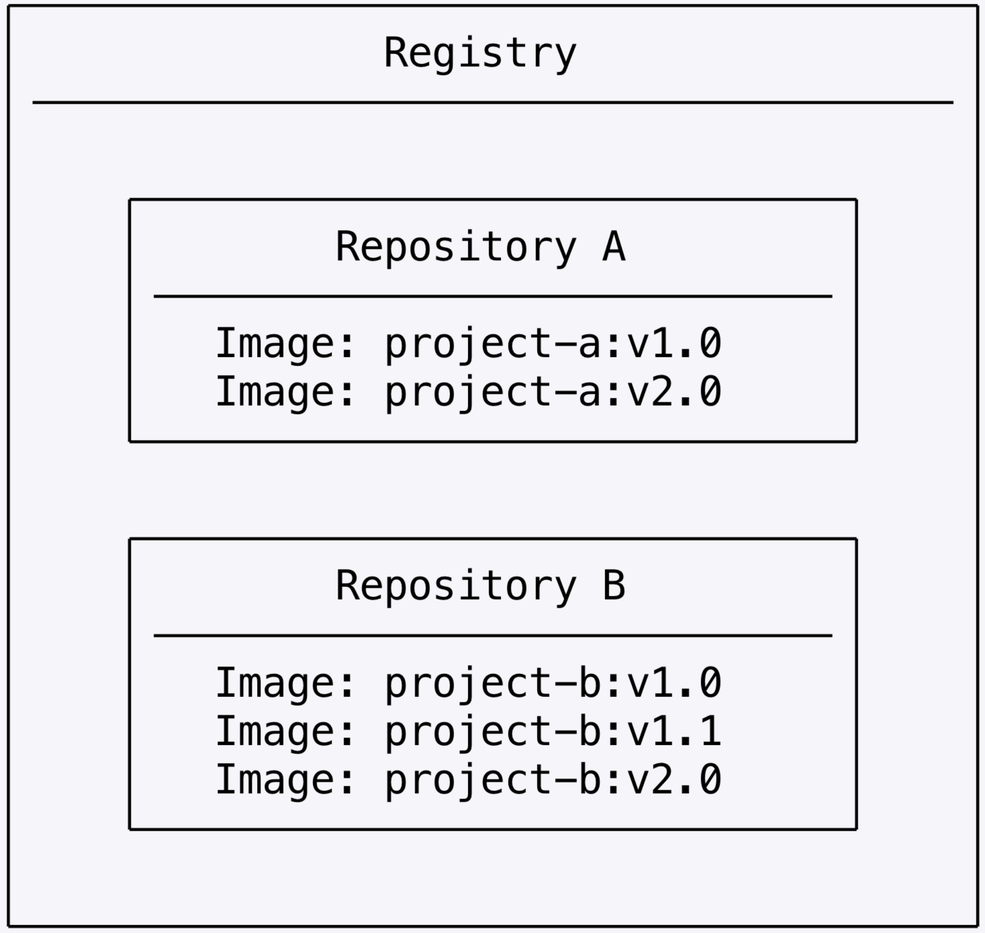
- Isolated environment for an application
- Lightweight and portable
  - Share host OS kernel
  - Minimize resource overhead
- Containers are live and ephemeral
  - Users can interact with them
  - Administrators can adjust their settings using Docker commands



Docker Registry

Registry, Repositories, and Images

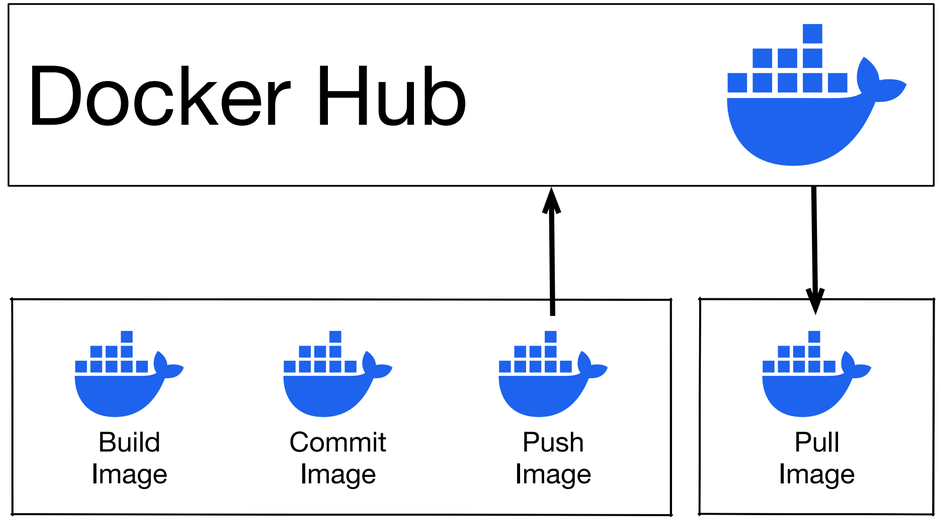
- Registry is a centralized location that stores and manages container images
- Repository is a collection of related container images
  - Each repository contains one or more container images
  - Tagged for identification



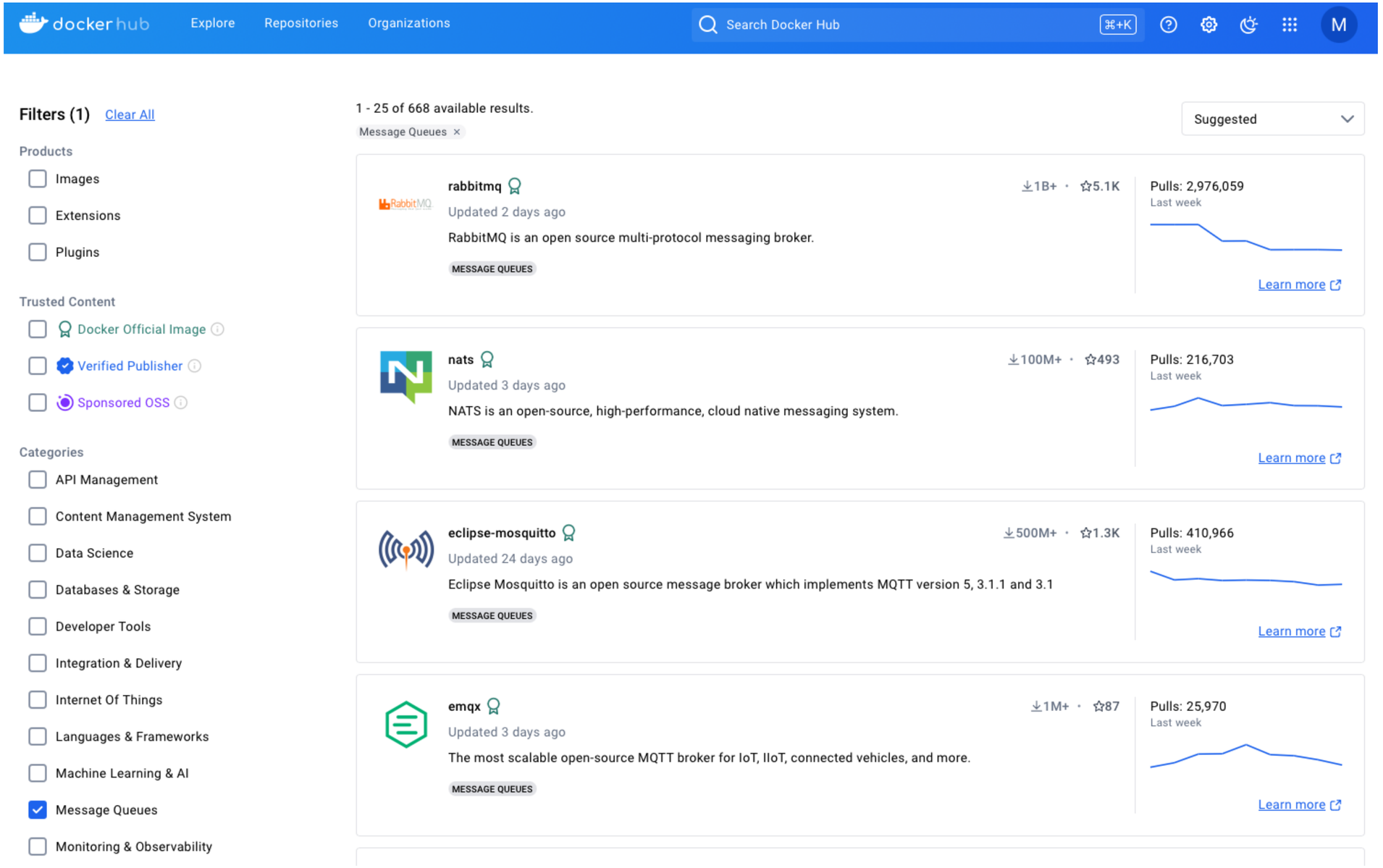
Docker Hub

Registry Service Provided by Docker

- Repositories for sharing container images with Docker community
- Official images
  - Clear doc, best practices, scanned for vulnerabilities…
- Publish images (external vendors)
- Private/Company images



Docker Hub


docker hub
Explore
Repositories
Organizations
Search Docker Hub
⌘+K
M
Filters (1)
Clear All
Products
Images
Extensions
Plugins
Trusted Content
Docker Official Image
Verified Publisher
Sponsored OSS
Categories
API Management
Content Management System
Data Science
Databases & Storage
Developer Tools
Integration & Delivery
Internet Of Things
Languages & Frameworks
Machine Learning & AI
Message Queues
Monitoring & Observability
1 - 25 of 668 available results.
Message Queues ×
Suggested
rabbitmq
Updated 2 days ago
RabbitMQ is an open source multi-protocol messaging broker.
MESSAGE QUEUES
1B+ · ☆5.1K
Pulls: 2,976,059
Last week
Learn more
nats
Updated 3 days ago
NATS is an open-source, high-performance, cloud native messaging system.
MESSAGE QUEUES
100M+ · ☆493
Pulls: 216,703
Last week
Learn more
eclipse-mosquitto
Updated 24 days ago
Eclipse Mosquitto is an open source message broker which implements MQTT version 5, 3.1.1 and 3.1
MESSAGE QUEUES
500M+ · ☆1.3K
Pulls: 410,966
Last week
Learn more
emqx
Updated 3 days ago
The most scalable open-source MQTT broker for IoT, IIoT, connected vehicles, and more.
MESSAGE QUEUES
1M+ · ☆87
Pulls: 25,970
Last week
Learn more

The Dockerfile

The Dockerfile

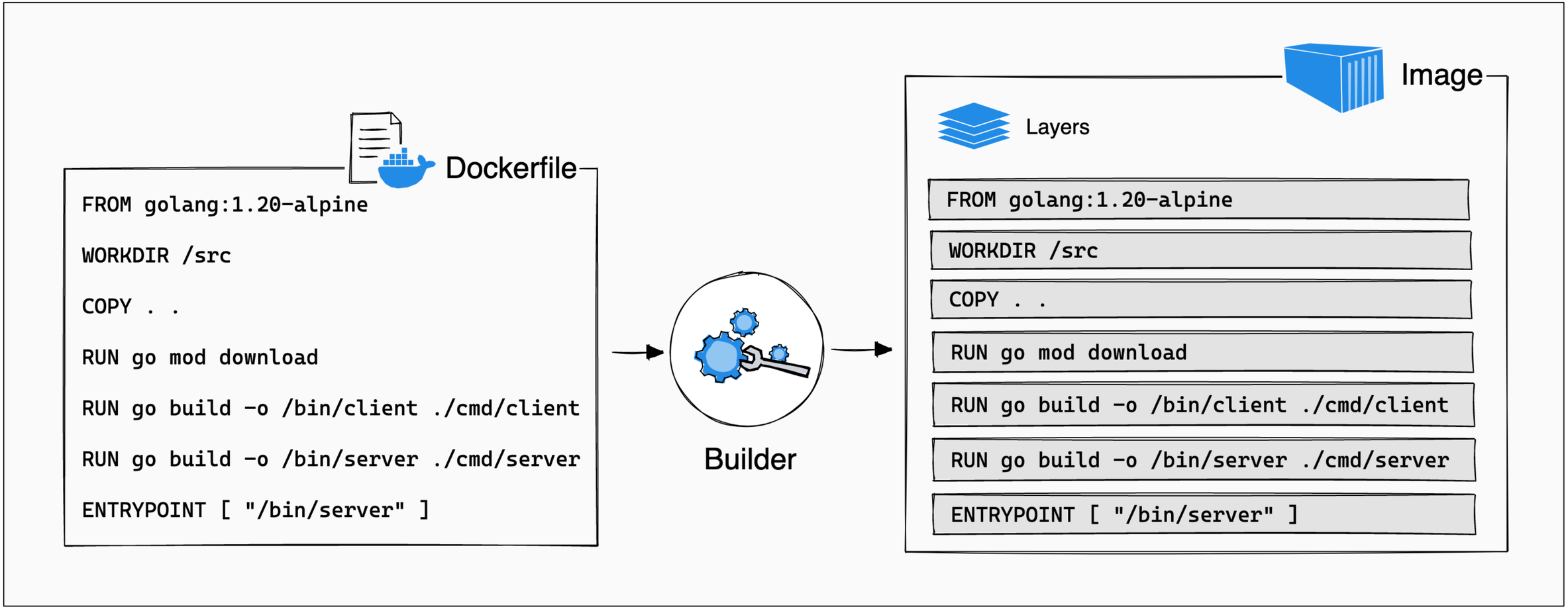
Text File for Image Construction

- Docker builds images based on Dockerfile
- Standard set of commands

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.

The Dockerfile

Text File for Image Construction



Docker Layers

Caching, Efficient Storage, Incremental Changes

- Layers represent changes in a Docker image
  - Each command in a Dockerfile (like RUN, COPY, ADD) creates a new layer
- Layers are shared across images (for efficiency)
  - Layers are stored only once
  - Faster image builds
- Layer caching speeds up image builds
  - Unmodified layers are reused

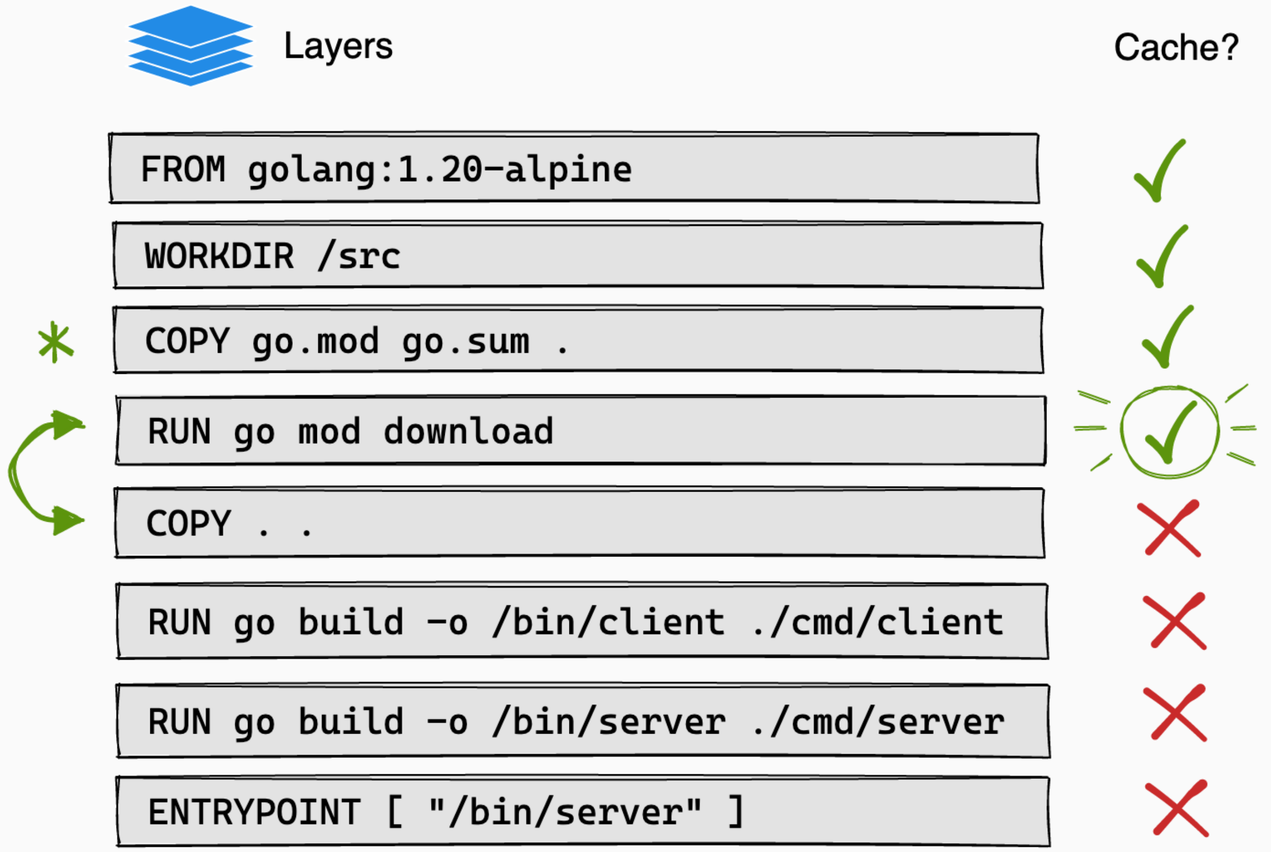
Docker Layer Caching

Cached Layers

Layers	Cache?
`FROM golang:1.20-alpine`	✓
`WORKDIR /src`	✓
`COPY . .`	✗
`RUN go mod download`	✗
`RUN go build -o /bin/client ./cmd/client`	✗
`RUN go build -o /bin/server ./cmd/server`	✗
`ENTRYPOINT [ "/bin/server" ]`	✗

Docker Layer Caching

Instruction order matters



Joke time

Why do Docker containers never get lonely?

Because they always have a lot of "friends" (processes) running inside.

Docker Networking

Docker Networking

The Network Subcommand

```
docker network ls

docker network inspect bridge

docker network inspect host
```

Default networks

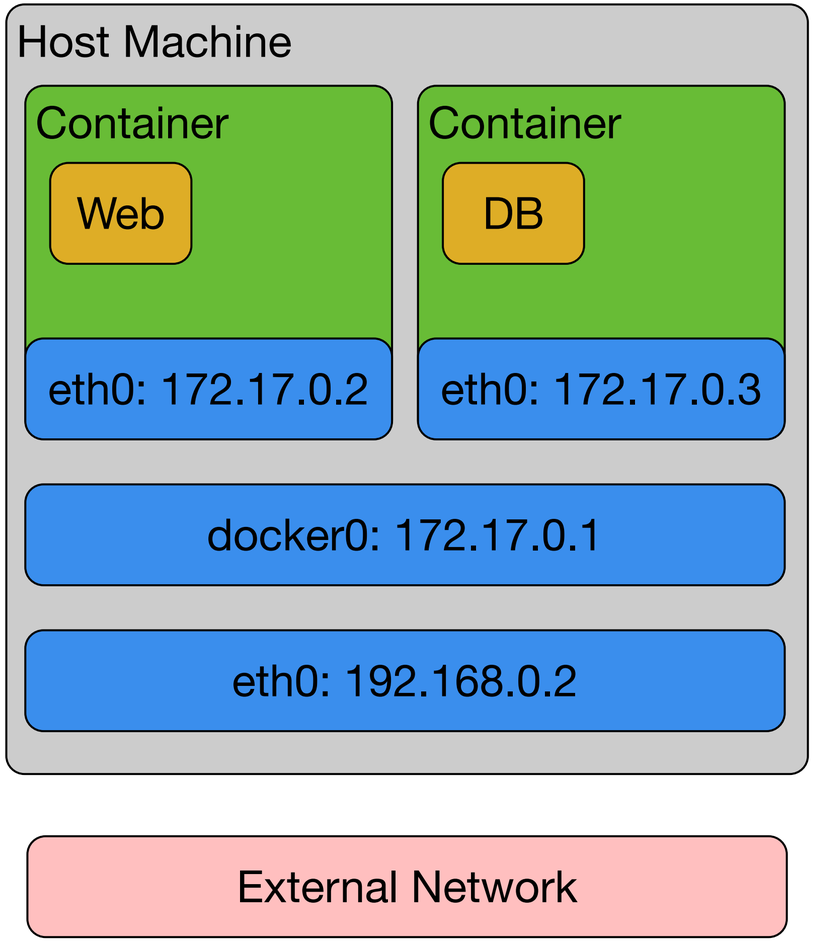
```
docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
9dcd3906d274   bridge    bridge    local
22096b6a3066   host      host      local
f10efee3f300   none      null      local
```

Docker Networking

Bridge Networking (Default)

- Network for containers on a single host (few hosts)
- Provides **isolated network** among containers
  - Automatic IP addressing
- External access via **port mapping**
- Slight performance overhead compared to host networking

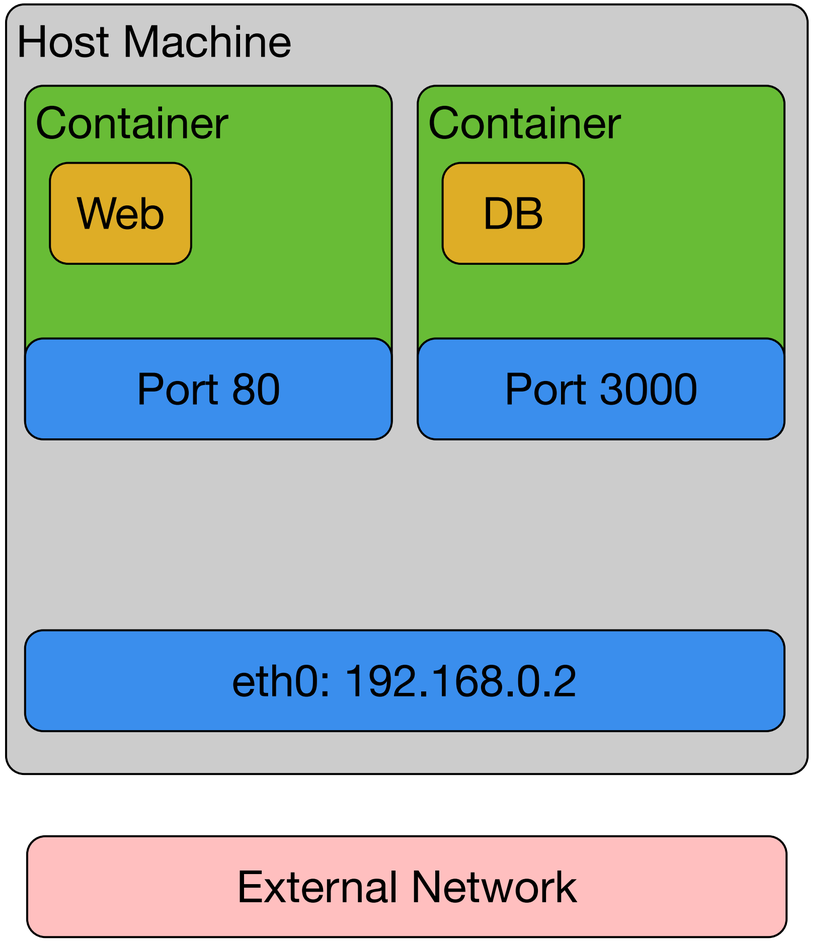
```
docker run ubuntu
```



Docker Networking

Host Networking

- Containers **share** the host's **network stack**
- Reduces network isolation
  - Share host's network namespace (IP, port)
  - Potentially **less secure**
- **Better performance** than bridge networking

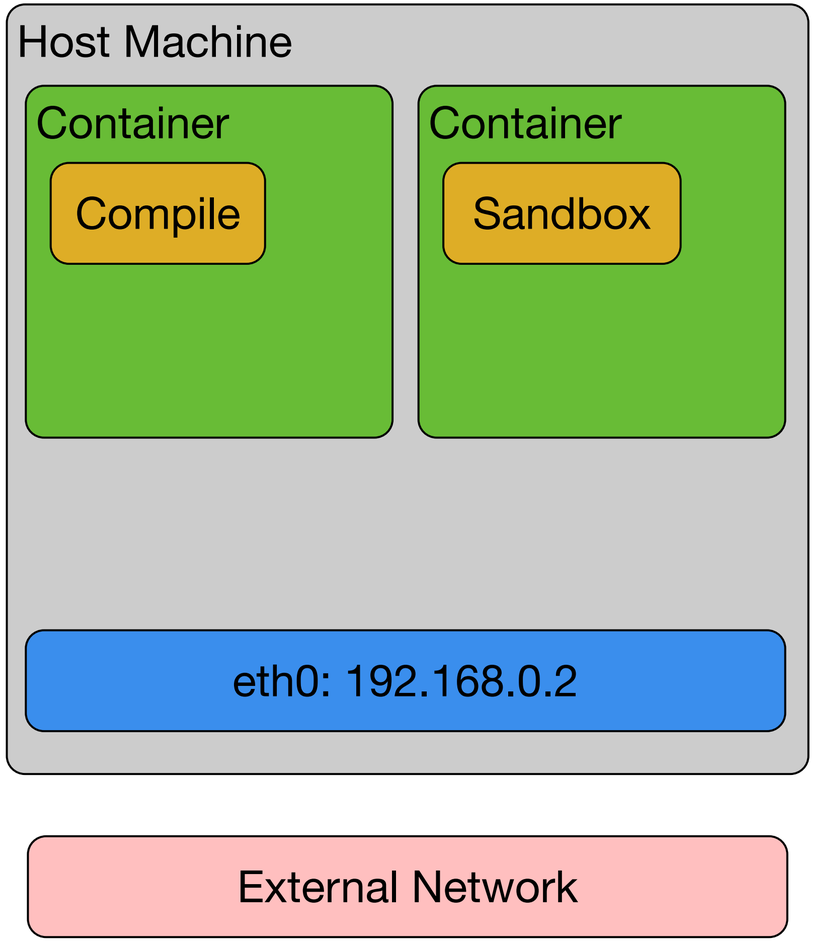



```
docker run --network=host ubuntu
```

Docker Networking

None Network

- Containers with **complete network isolation**
- Useful for tasks that don't require network access
  - Consistent and reproducible builds
  - Difficult to configure dependencies and environment
- **Output to file system** only



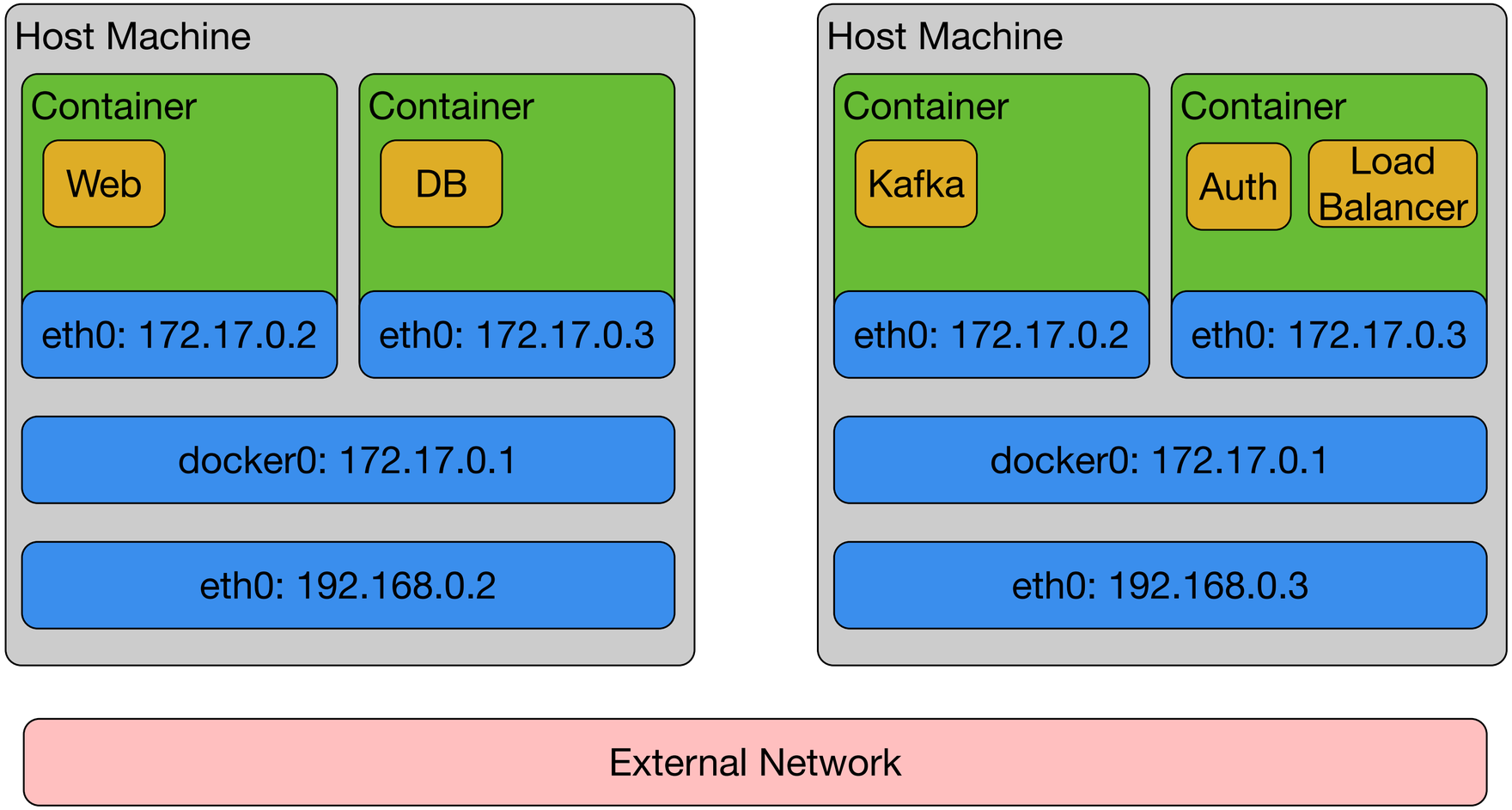

```
docker run --network=none ubuntu
```

Docker Networking

Overlay Networking

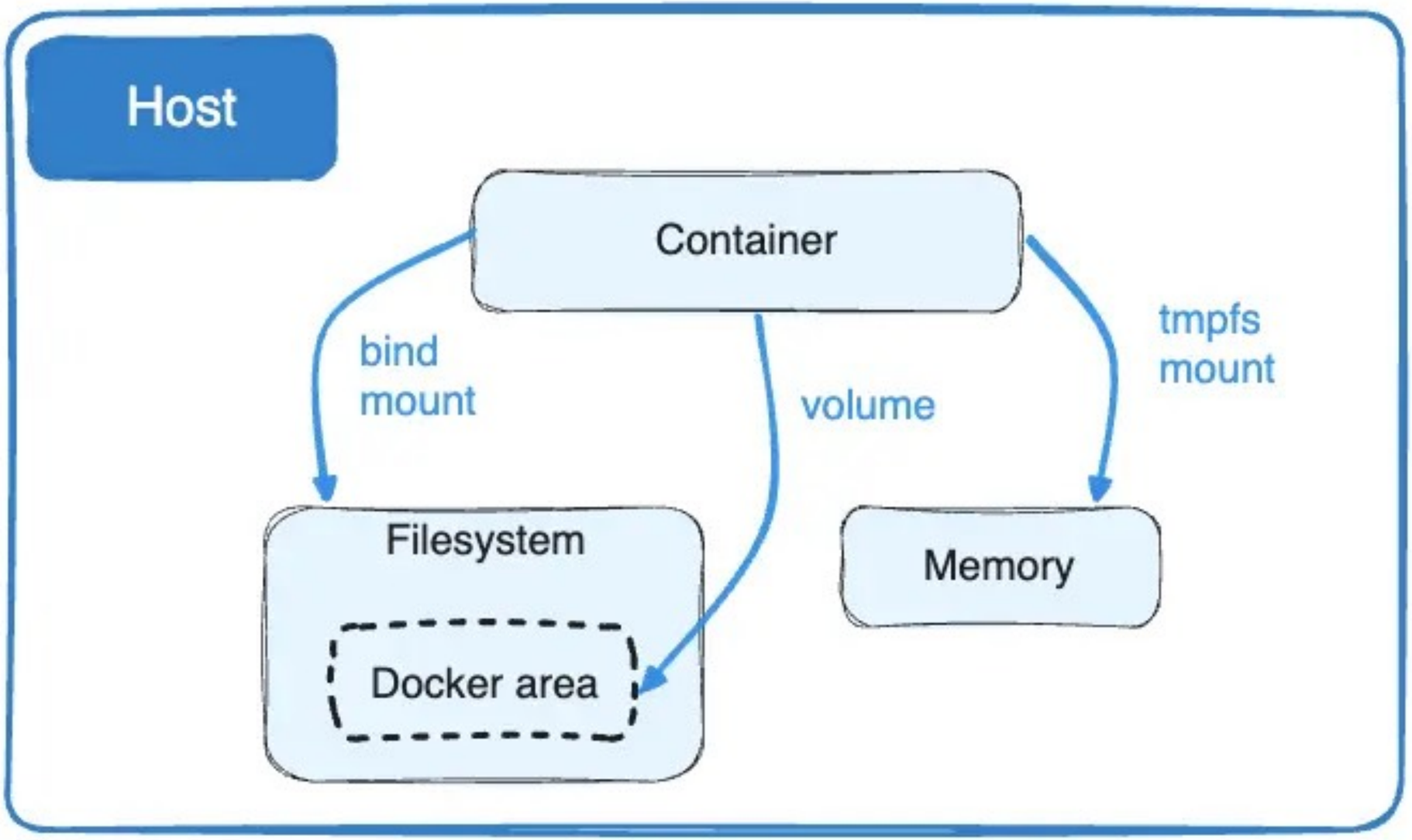
- Multi-host networking
  - Communication between **containers on different hosts**
- Automatic service discovery and configuration
- Hosts managed by Docker Swarm or Kubernetes clusters

Overlay Networking



Docker Storage

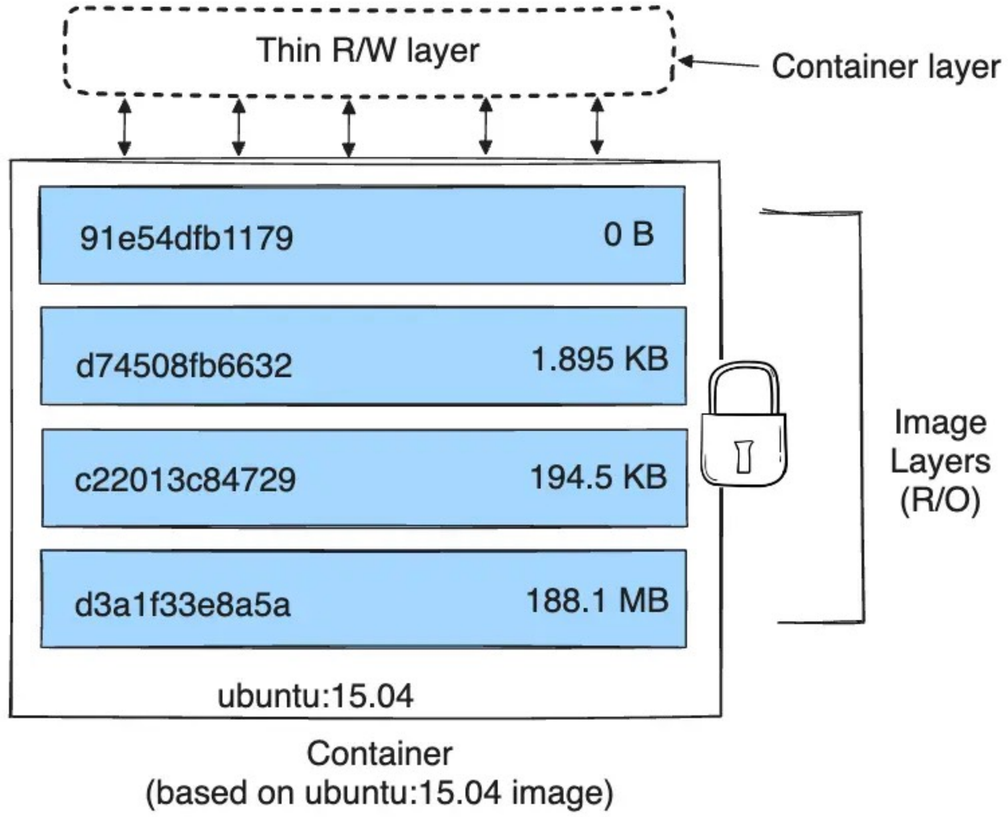
Docker Storage Options



Docker's Default Storage Driver

Layered File System

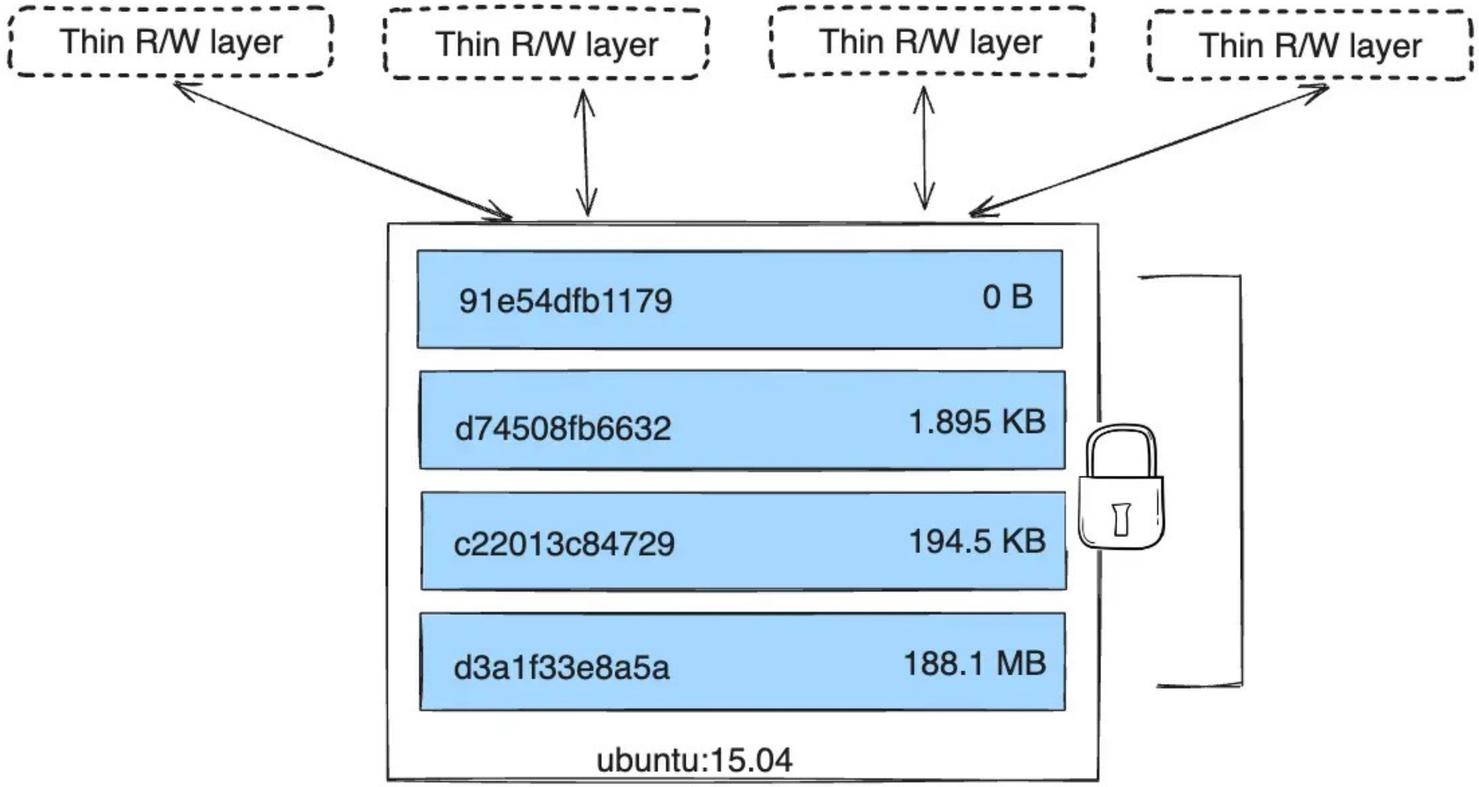
- Ephemeral storage
- **Data retained** across stops and **restarts** of the container
- **Data is lost** when the container is **deleted**
  - Mainly useful for temporary data



Containers Reusing Lower Layers

Layered File System

- Different containers can use ubuntu:15.04
- Each container get their own writable layer



tmpfs Mounts

Data stored in the host's memory (RAM)

- Ephemeral storage
- **Data is lost** when the container is **stopped**
  - Useful for caches or session data

Docker Volume

Persistent data storage outside container's filesystem

- **Persistent Storage**
  - Ensure data persists across container restarts
  - Managed by Docker `/var/lib/docker/volumes`
- **Shared Access**
  - Share data between containers by mounting the same volume
  - Share data with host system
  - Decouple data from container lifecycle, easy backups, and migration

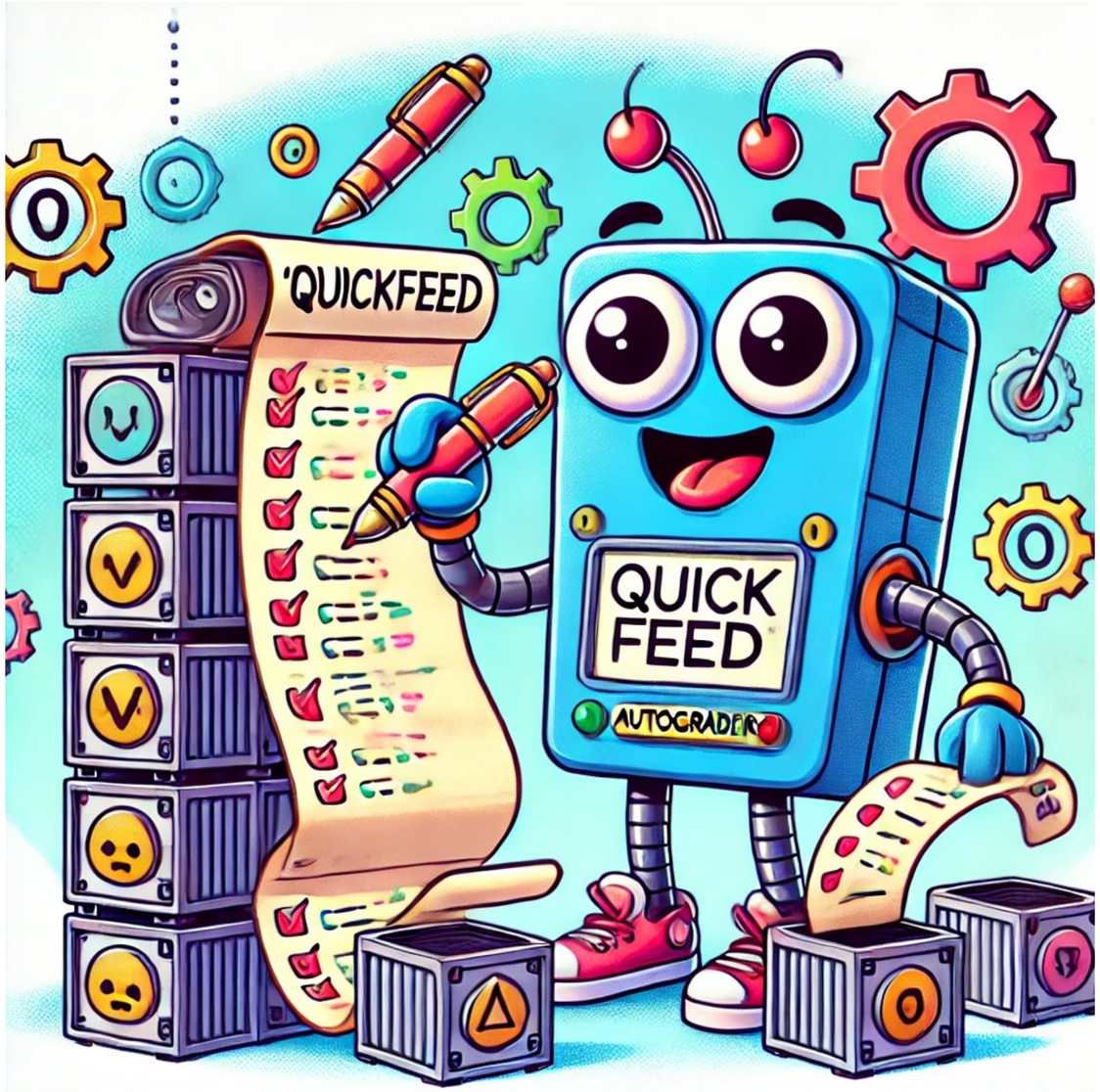
Bind Mounts

- **Map specific file/directory** on **host's filesystem** to a path inside the container
- Use Cases: Access specific content on host file system
  - Source code and configuration files
  - (See QuickFeed example later)
- Tightly coupled to host's filesystem structure
  - Less portable than Docker managed volumes

Summary of Storage Alternatives

Storage Option	Description	Usage	Persistence Across Start/Stop	Persistence After Container Removal
Volumes	Managed by Docker, stored outside the container's filesystem	Sharing data between containers, backups, migration, decoupling from lifecycle	Yes	Yes
Bind Mounts	Directly maps a host directory or file to a path inside a container	Accessing specific host files or directories, development (e.g., syncing code)	Yes	Yes
tmpfs Mounts	Stores data in the host's memory (RAM), not on disk	Temporary data that doesn't need persistence (e.g., caching, session data)	No	No
Layered Filesystem	Manages container's filesystem layers; data in the writable layer	Default container storage, ephemeral data storage within the container	Yes	No

QuickFeed Interlude



QuickFeed's Docker Bind Mount

```
tree -L1 courses

courses
├── dat240-2023
├── dat240-2024
├── dat310-2023
├── dat310-2024
├── dat320-2023
├── dat320-2024
├── dat515-2024
├── dat520-2023
├── dat520-2024
├── dat600-2024
└── qf-103
```

```
courses/dat515-2024/
├── assignments
│   ├── lab1
│   ├── lab2
│   ├── lab3
│   ├── lab4
│   ├── lab5
│   └── lab6
└── tests
    ├── lab1
    ├── lab2
    ├── lab3
    ├── lab4
    ├── lab5
    └── lab6
```

QuickFeed's Docker Bind Mount

```
tree -L1 courses

courses
├── dat240-2023
├── dat240-2024
├── dat310-2023
├── dat310-2024
├── dat320-2023
├── dat320-2024
├── dat515-2024
├── dat520-2023
├── dat520-2024
├── dat600-2024
└── qf-103
```

```
courses/dat515-2024/
├── assignments
│   ├── lab1
│   ├── lab2
│   ├── lab3
│   ├── lab4
│   ├── lab5
│   └── lab6
└── tests
    ├── lab1
    ├── lab2
    ├── lab3
    ├── lab4
    ├── lab5
    └── lab6
```

```
courses/dat320-2023/
├── assignments
│   ├── go.mod
│   ├── go.sum
│   ├── lab1
│   │   ├── generate_token
│   │   ├── git_questions.md
│   │   ├── missing_semester_questions.md
│   │   ├── README.md
│   │   └── shell_questions.md
│   ├── lab2
│   │   ├── c_questions.md
│   │   ├── hello
│   │   └── README.md
│   └── lab3
│       ├── cipher
│       ├── cmd
│       ├── errors
│       ├── go_questions.md
│       ├── multiwriter
│       ├── README.md
│       ├── sequence
│       └── stringer
└── tests
    ├── go.mod
    ├── go.sum
    ├── internal
    │   ├── git
    │   ├── staff
    │   └── token
    ├── lab1
    │   ├── assignment.yml
    │   ├── git_questions_ag_test.go
    │   ├── main_ag_test.go
    │   ├── missing_semester_ag_test.go
    │   ├── shell_questions_ag_test.go
    │   ├── token_ag_test.go
    │   └── tokengen
    ├── lab2
    │   ├── assignment.yml
    │   ├── c_questions_ag_test.go
    │   ├── hello_world_ag_test.go
    │   ├── hello_world_init.go
    │   └── main_ag_test.go
    └── scripts
        ├── Dockerfile
        └── run.sh
```

The DAT320 Dockerfile

```
FROM golang:1.23-alpine

# Install bash and git (and build-base to get gcc)
# (this is required when building FROM: golang:alpine)
RUN apk update && apk add --no-cache git bash build-base docker openrc
RUN wget -O- -nv https://raw.githubusercontent.com/golangci/golangci-lint/master/install.sh | sh -s v1.41.1

WORKDIR /quickfeed
```

Script to Run and Score Student Code

The run.sh script

```
#image/dat320

cd "$ASSIGNMENTS/$CURRENT"
find . \( -name '*_test.go' -and -not -name '*_ag_test.go' \) -exec rm -rf {} \;

cp -r "$TESTS"/* "$ASSIGNMENTS"/

go get -t github.com/quickfeed/quickfeed/kit/score
go mod tidy
SCORE_INIT=1 go test -v ./... 2>&1 | grep TestName
```

Script to Run and Score Student Code

The run.sh script

```
#image/dat320

cd "$ASSIGNMENTS/$CURRENT"
find . \( -name '*_test.go' -and -not -name '*_ag_test.go' \) -exec rm -rf {} \;

cp -r "$TESTS"/* "$ASSIGNMENTS"/

go get -t github.com/quickfeed/quickfeed/kit/score
go mod tidy
SCORE_INIT=1 go test -v ./... 2>&1 | grep TestName


cd "$SUBMITTED/$CURRENT"
find . -name '*_test.go' -exec rm -rf {} \;

cp -r "$TESTS"/* "$SUBMITTED"/

go get -t github.com/quickfeed/quickfeed/kit/score
go mod tidy

go test -v -timeout 30s ./... 2>&1
```

Connect to Docker Daemon

```go
// NewDockerCI returns a runner to run CI tests.
func NewDockerCI(logger *zap.SugaredLogger) (*Docker, error) {
    cli, err := client.NewClientWithOpts(
        client.FromEnv,
        client.WithAPIVersionNegotiation(),
    )
    if err != nil {
        return nil, err
    }
    return &Docker{
        client: cli,
        logger: logger,
    }, nil
}
```

Run a Job

```go
// Run implements the CI interface. This method blocks until the job has been
// completed or an error occurs, e.g., the context times out.
func (d *Docker) Run(ctx context.Context, job *Job) (string, error) {
    if d.client == nil {
        return "", fmt.Errorf("cannot run job: %s; docker client not initialized", job.Name)
    }

    resp, err := d.createImage(ctx, job)
    if err != nil {
        return "", err
    }
    d.logger.Infof("Created container image '%s' for %s", job.Image, job.Name)
    if err = d.client.ContainerStart(ctx, resp.ID, container.StartOptions{}); err != nil {
        return "", err
    }
```

Wait for Container and Get Logs

```
d.logger.Infof("Waiting for container image '%s' for %s", job.Image, job.Name)
msg, err := d.waitForContainer(ctx, job, resp.ID)
if err != nil {
    return msg, err
}

d.logger.Infof("Done waiting for container image '%s' for %s", job.Image, job.Name)
// extract the logs before removing the container below
logReader, err := d.client.ContainerLogs(ctx, resp.ID, container.LogsOptions{
    ShowStdout: true,
})
if err != nil {
    return "", err
}
```

Describes a CI Job

```go
// Job describes how to execute a CI job.
type Job struct {
    // Name describes the running job; mainly used to name docker containers.
    Name string
    // Image names the image to use to run the job.
    Image string
    // Dockerfile contents.
    // If empty, the image is assumed to exist.
    // If non-empty, the image is built from this Dockerfile.
    Dockerfile string
    // BindDir is the directory to bind to the container's /quickfeed directory.
    BindDir string
    // Env is a list of environment variables to set for the job.
    Env []string
    // Commands is a list of shell commands to run as part of the job.
    Commands []string
}

// Runner contains methods for running user provided code in isolation.
type Runner interface {
    // Run should synchronously execute the described job and return the output.
    Run(context.Context, *Job) (string, error)
}
```

ContainerCreate

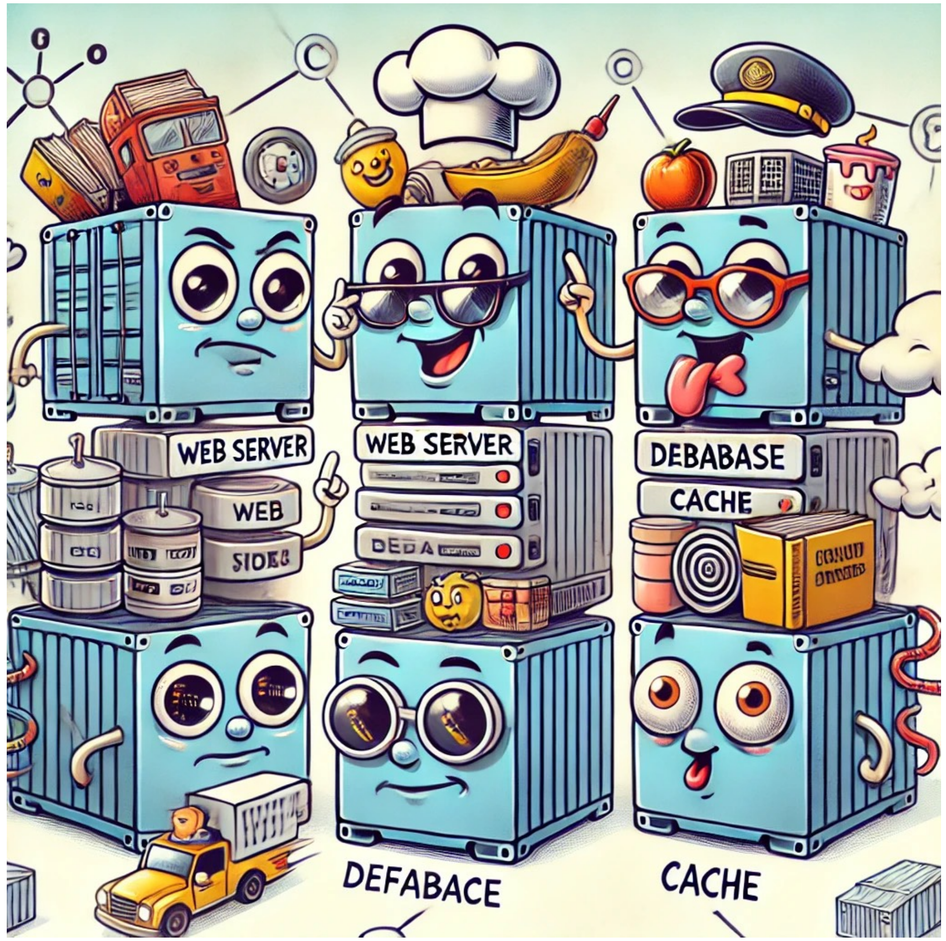
```
var hostConfig *container.HostConfig
if job.BindDir != "" {
    hostConfig = &container.HostConfig{
        Mounts: []mount.Mount{
            {
                Type:   mount.TypeBind,
                Source: job.BindDir,
                Target: QuickFeedPath,
            },
        },
    }
}

create := func() (container.CreateResponse, error) {
    return d.client.ContainerCreate(ctx, &container.Config{
        Image: job.Image,
        User:  fmt.Sprintf("%d:%d", os.Getuid(), os.Getgid()), // Run the image as the current user, e.g., quickfeed
        Env:   job.Env,                                        // Set default environment variables
        Cmd:   []string{"/bin/bash", "-c", strings.Join(job.Commands, "\n")},
    }, hostConfig, nil, nil, job.Name)
}

resp, err := create()
```

Multi-Container Applications

Multi-Container Applications

Microservice Architecture

- **Single Responsibility per Container**
  - Run only one service to ensure modularity and easier management
- **Inter-Container Communication**
  - Containers communicate with each other over a Docker network
  - Internal DNS to resolve service names to container IPs, making services discoverable



Multi-Container Applications

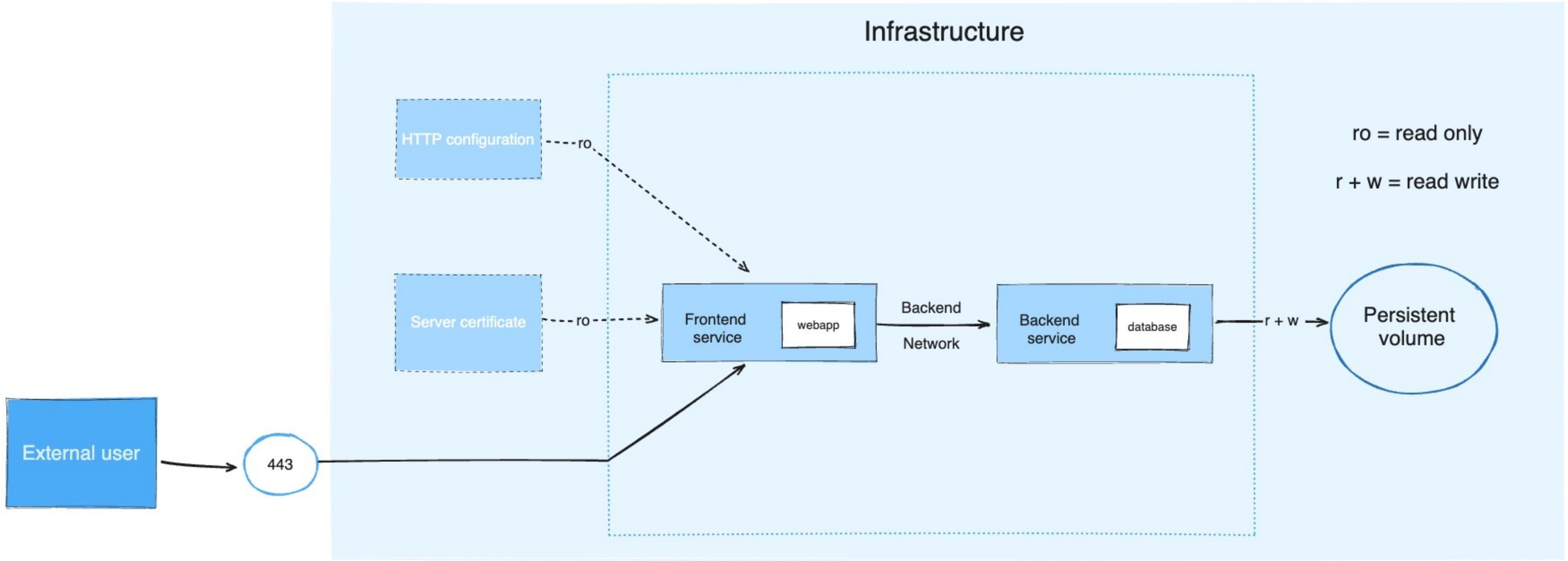
Microservice Architecture

- **Loose Coupling**
  - Isolating services into different containers
  - Updates can be made to one service without affecting other containers
  - More maintainable and resilient applications
- **Scaling and Resilience**
  - Individual services can be scaled independently based on load
  - Load balancer, multiple web servers, a single database

Docker Compose

- Docker Compose is a tool that simplifies managing multi-container applications
- A `compose.yml` file to define and run multiple containers
  - As a single service stack,
  - handling container creation,
  - networking, and
  - volume management

Docker Compose Example



Docker Compose Example

```
services:
  frontend:
    image: example/webapp
    ports:
      - "443:8043"
    networks:
      - front-tier
      - back-tier
    configs:
      - httpd-config
    secrets:
      - server-certificate

  backend:
    image: example/database
    volumes:
      - db-data:/etc/data
    networks:
      - back-tier
```

```
volumes:
  db-data: {}

configs:
  httpd-config:
    external: true

secrets:
  server-certificate:
    external: true

networks:
  front-tier: {}
  back-tier: {}
```

External Secrets

Separation of Secret Management from Container Spec

- Avoid hardcoding into the compose file
- Make the application more secure and flexible
- Secret resources managed externally via Docker Swarm

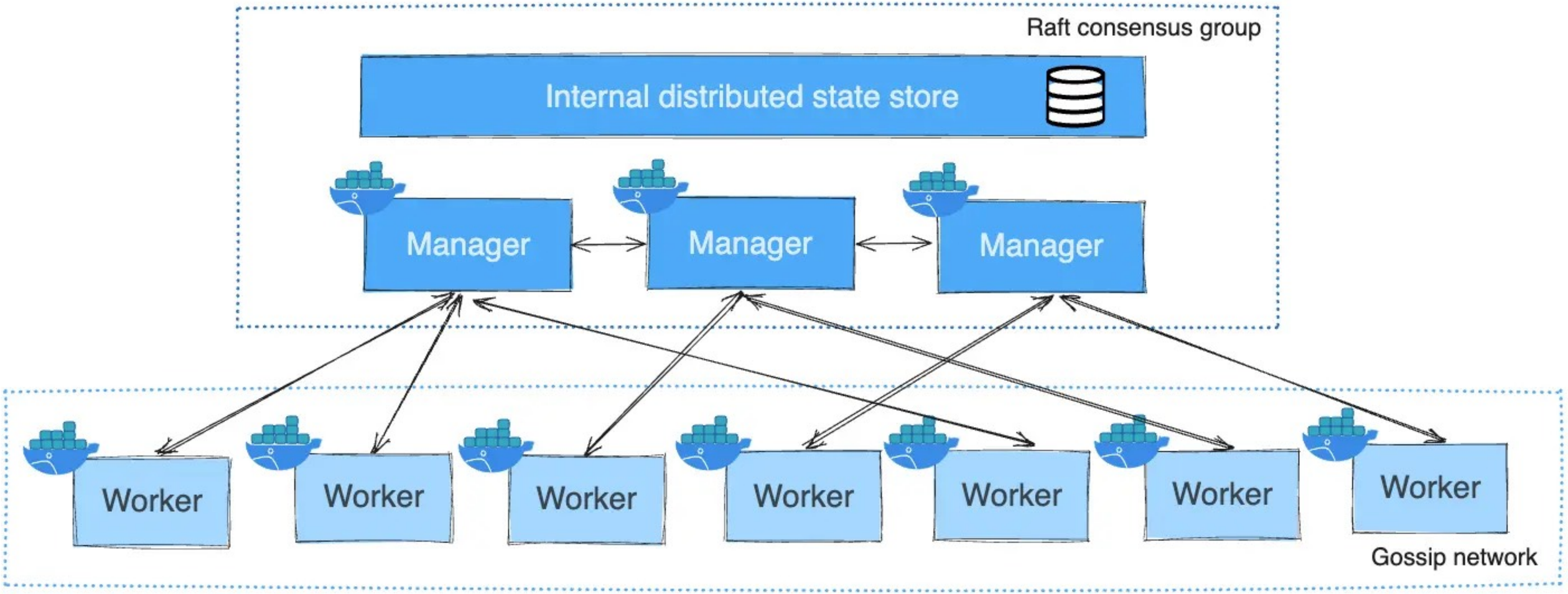
```
docker swarm init
docker secret create server-certificate /path/to/server-certificate.pem
docker secret ls
docker-compose up -d
```

Docker Container Orchestration

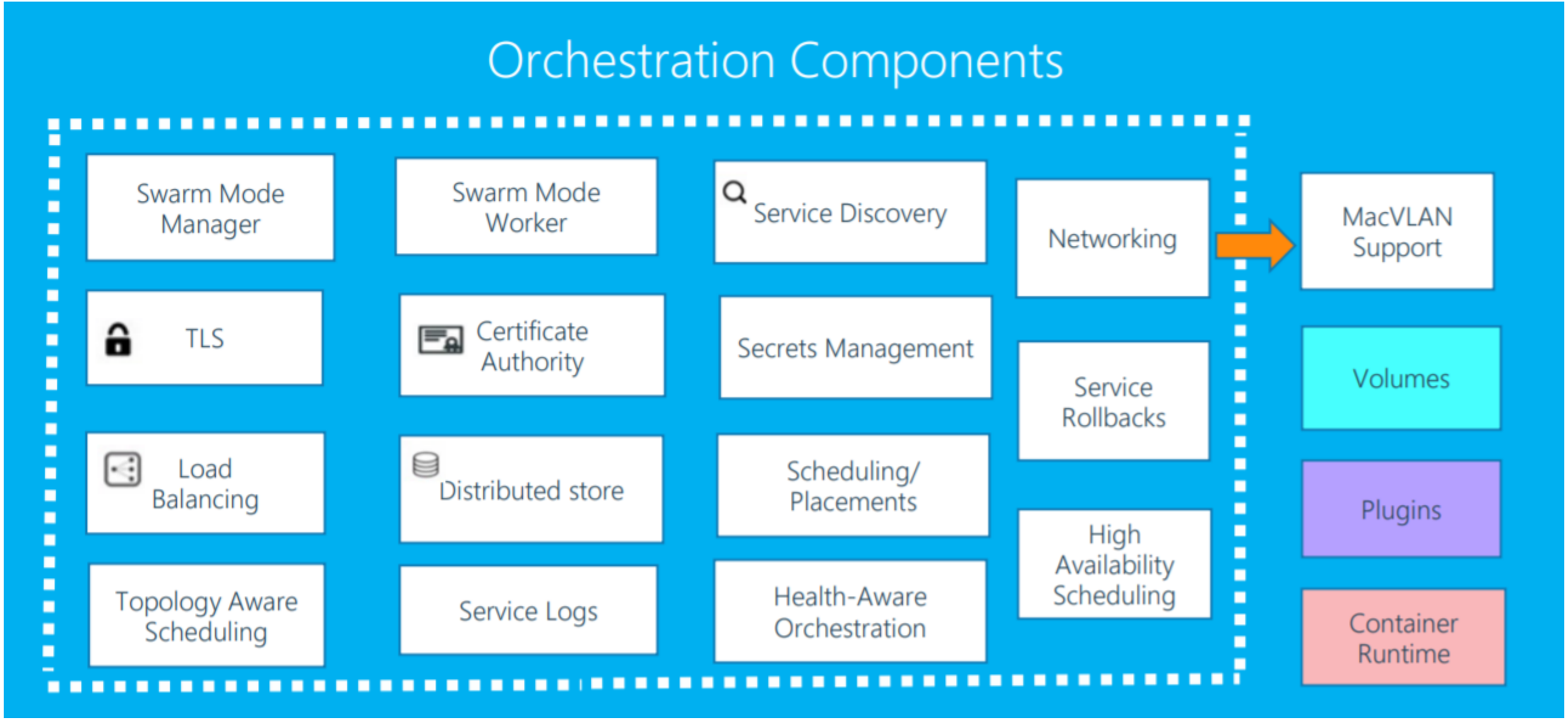
Container Orchestration Services

- **Scaling**
  - Automatically scale containers across multiple nodes in a Docker Swarm
- **Load Balancing**
  - Distribute incoming requests evenly across containers within the service
- **Declarative Management**
  - Define desired state of services
  - Docker ensures the correct number of containers are running

Docker Swarm



Docker Swarm



Questions?