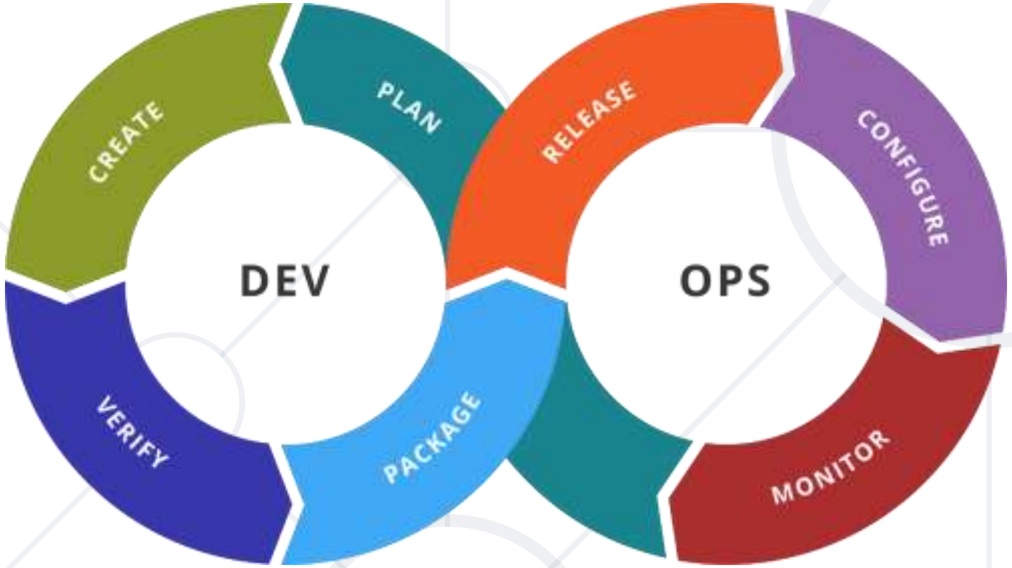
Advanced Docker

Advanced Techniques. Distributed Applications. Clusters



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

You Have Questions?

Software University

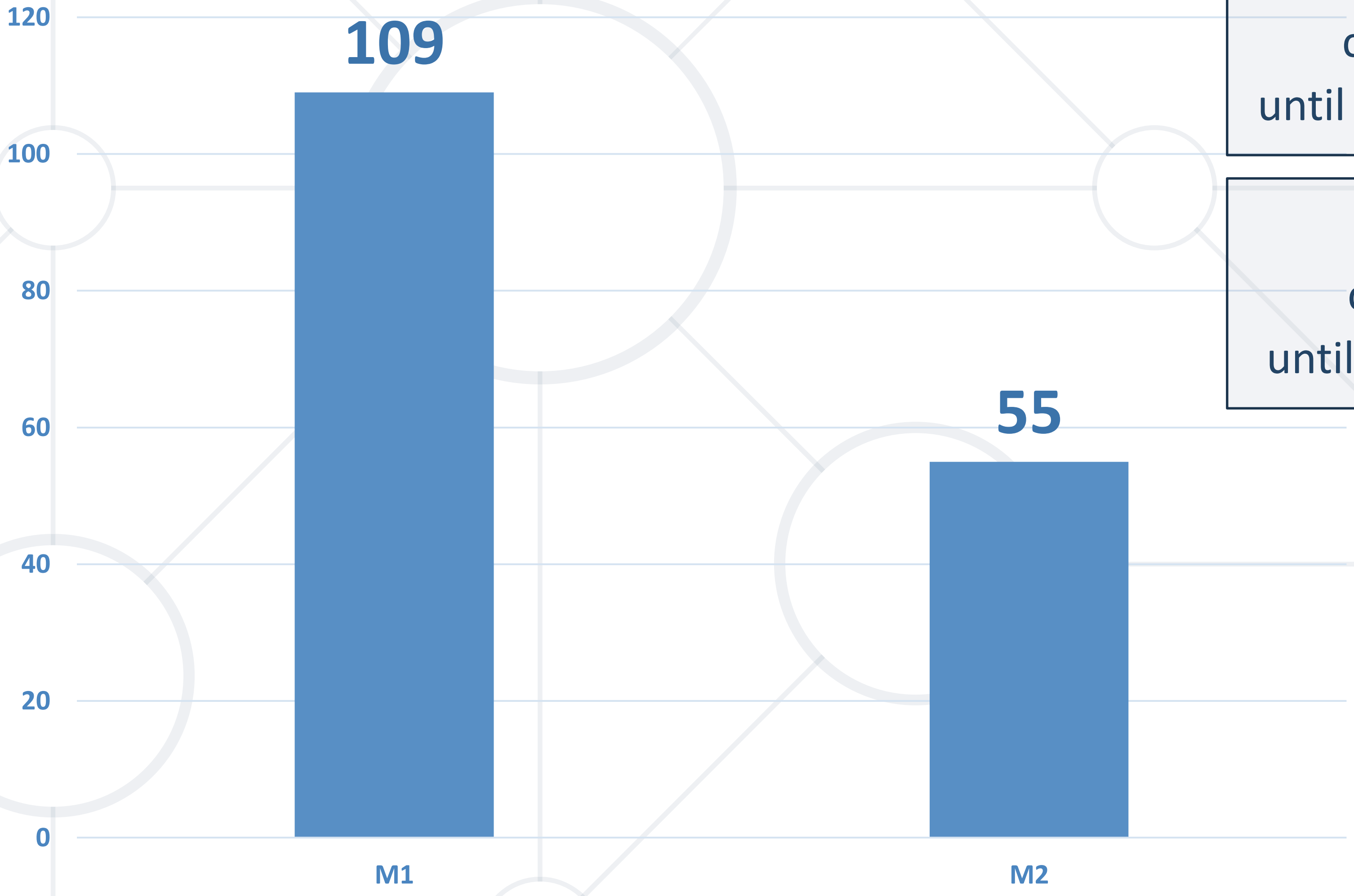
sli.do

#DevOps-23

facebook.com

/groups/DevOpsContainerizationCICDMonitoringJanuary2023

Homework Progress



Solutions for M2
can be submitted
until 23:59 on 15.02.2023

Solutions for M3
can be submitted
until 23:59 on 22.02.2023

Previous Module (M2)

Quick Overview

What We Covered

1. Containerization
2. Introduction to Docker
3. Docker in Action
4. Create Our Own Images

This Module (M3)

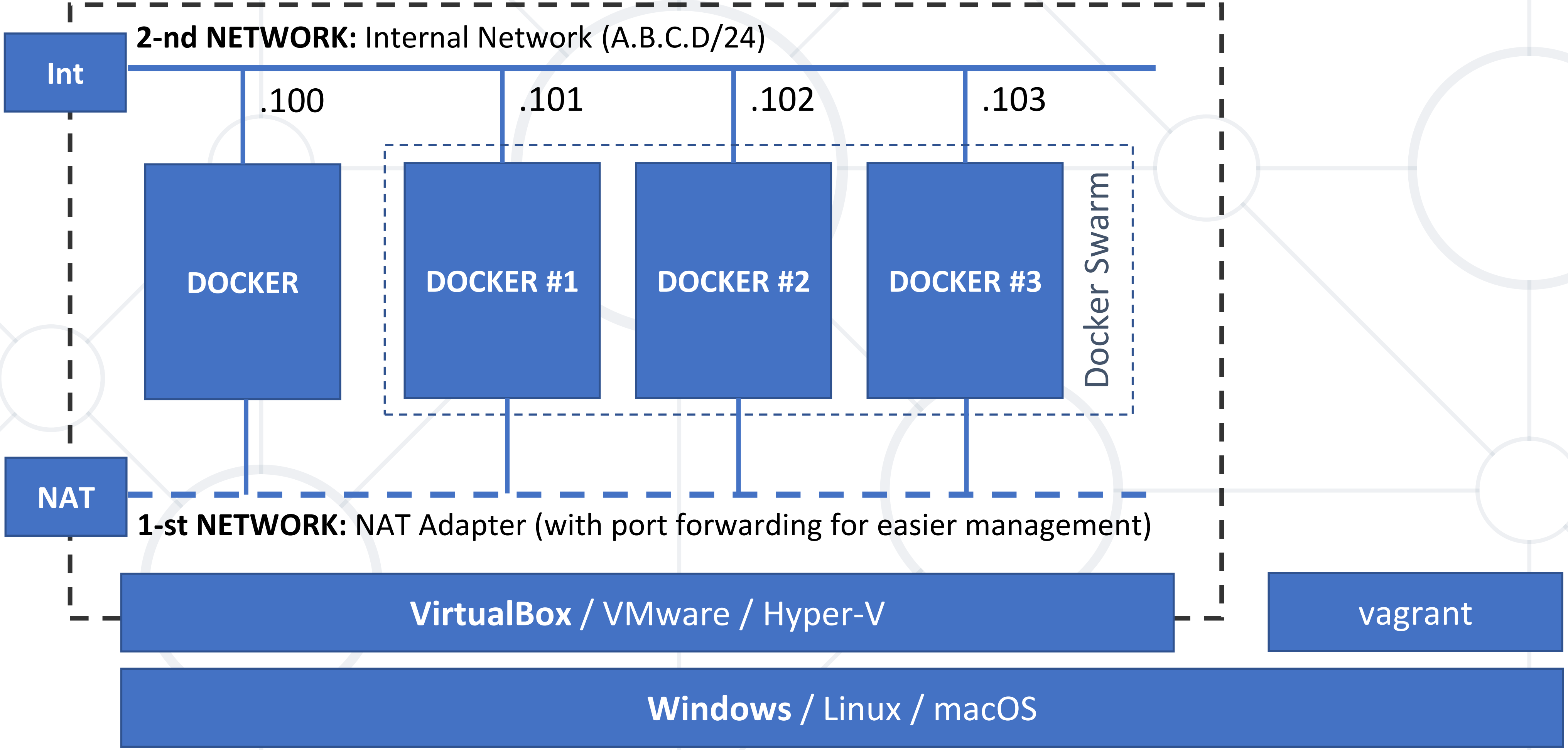
Topics and Lab Infrastructure

Table of Contents

Software University

1. Advanced techniques
   - Networking
   - Volumes
2. Distributed Applications
   - Linking Methods
   - Docker Compose
3. Docker Clusters
   - Components and Principles
   - Docker Swarm

Lab Infrastructure


Int
2-nd NETWORK: Internal Network (A.B.C.D/24)
.100
.101
.102
.103
DOCKER
DOCKER #1
DOCKER #2
DOCKER #3
Docker Swarm
NAT
1-st NETWORK: NAT Adapter (with port forwarding for easier management)
VirtualBox / VMware / Hyper-V
vagrant
Windows / Linux / macOS

DevOps

Communication

Networks: Overview and Usage

Docker Network

- Uses **pluggable drivers**. There is a **set of preinstalled** drivers
- **bridge** is the default driver. It allows containers connected to the same bridge to communicate while isolating them from the rest
- **host** uses the host's networking directly
- **overlay** connects multiple Docker daemons together and enables swarm services to communicate with each other
- **ipvlan** gives total control over both IPv4 and IPv6 addressing
- **macvlan** allows for assigning specific MAC addresses to containers
- **none** disables all networking for a container

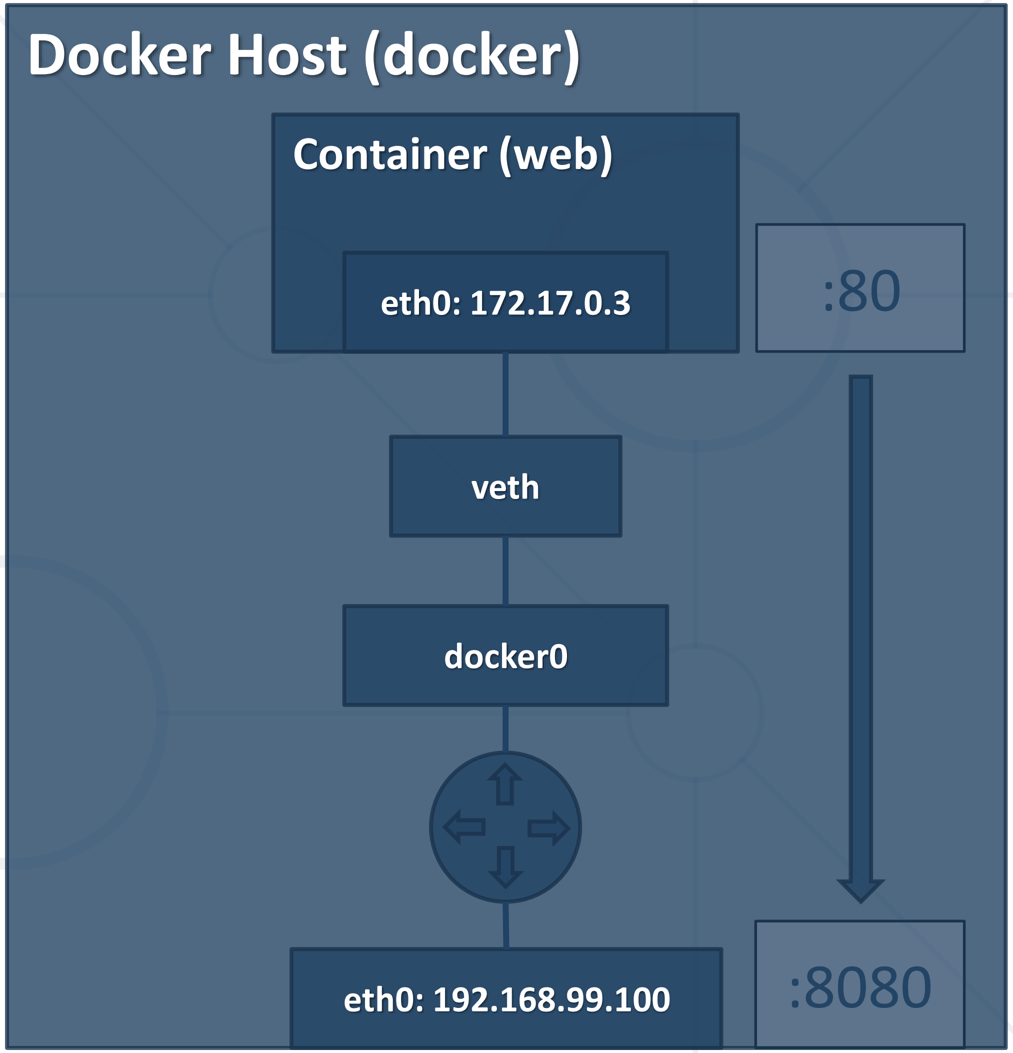
Default Network

- Start a container with no explicit network settings

```
docker container run -d --name web \
  img-web
```

- We can expose a container port

```
docker container run -d --name web \
  -p 8080:80 img-web
```



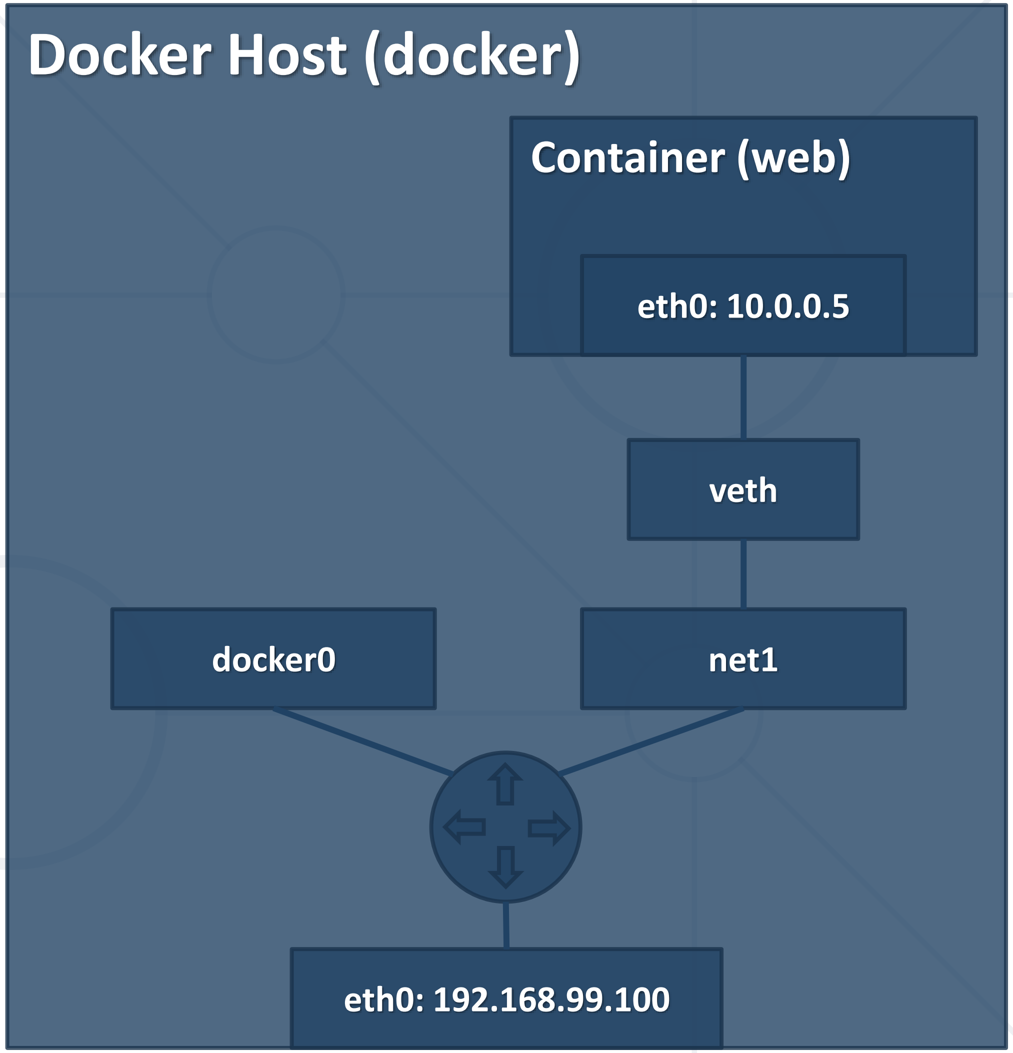
Custom Network

- Create a bridge network

```
docker network create -d bridge net1
```

- Start container connected to specific network

```
docker container run -d --name web \
  --net net1 img-web
```



Two Networks

- Create a bridge network

```
docker network create -d bridge mynet
```

Start container connected to the default network

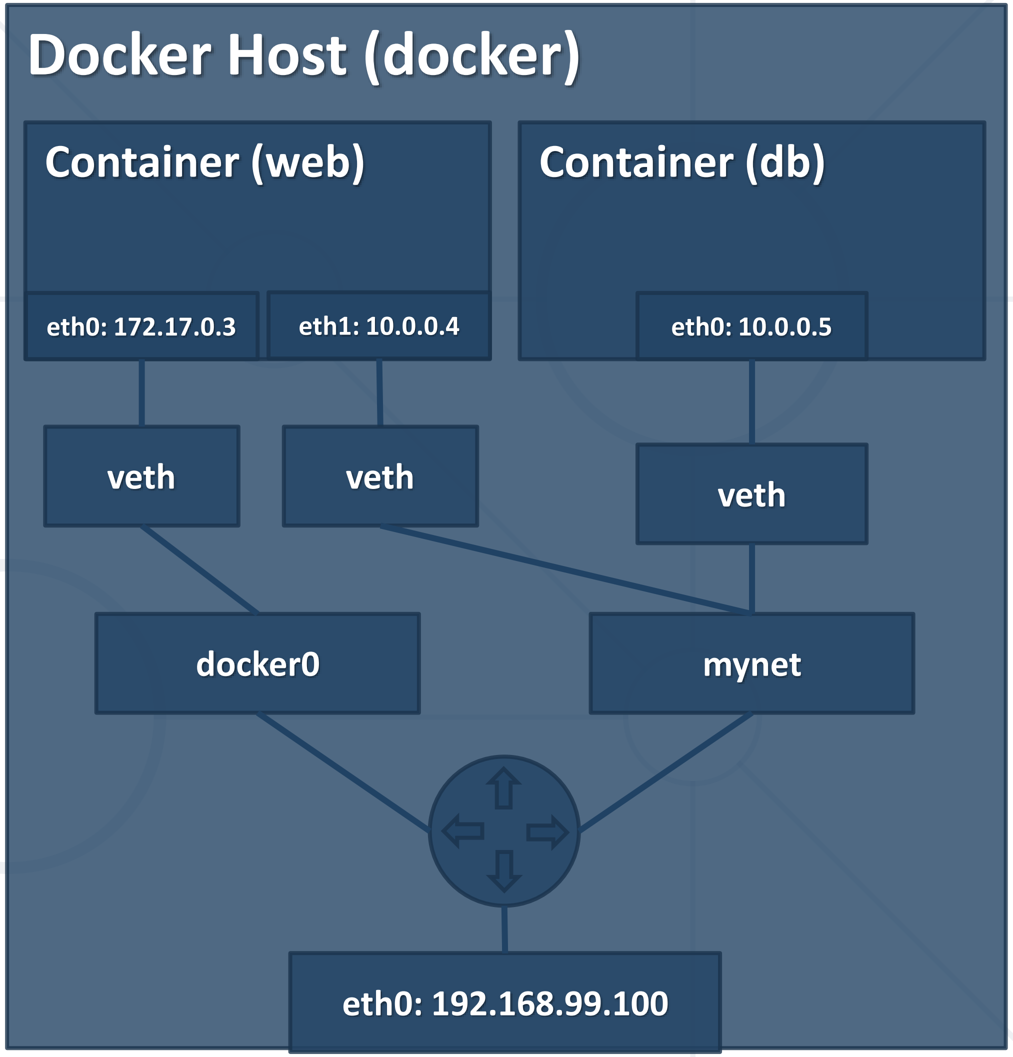
```
docker container run -d --name web \
  img-web
```

- Start container connected to specific network

```
docker container run -d --name db \
  --net mynet img-db
```

- Connect container to another network

```
docker network connect net1 web
```



network ls

- Purpose
  - List networks
- Syntax

```
docker network ls [options]
```

- Example

```
# list IDs of all networks
docker network ls -q
# list all networks that satisfy the filter
docker network ls -f driver=bridge
```

network inspect

- Purpose
  - Display detailed information on one or more networks
- Syntax

```
docker network inspect [options] network [network]
```

- Example

```
# show network details
docker network inspect dob-network
```

network connect

- Purpose
  - Connect a container to a network
- Syntax

```
docker network connect [options] network container
```

- Example

```
# connect container to a network
docker network connect \
   dob-bridge \
   cont-001
```

network disconnect

- Purpose
  - Disconnect a container from a network
- Syntax

```
docker network disconnect [options] network container
```

- Example

```
# disconnect container from a network
docker network disconnect -f \
   dob-bridge \
   cont-001
```

network create

- Purpose
  - Create a network
- Syntax

```
docker network create [options] network
```

- Example

```
# create new bridge network
docker network create -d bridge \
   --subnet 10.0.0.1/24 \
   dob-bridge
```

network rm

- Purpose
  - Remove one or more networks
- Syntax

```
docker network rm network [network]
```

- Example

```
# remove networks net-1 and net-2
docker network rm net-1 net-2
```

network prune

- Purpose
  - Remove all unused networks
- Syntax

```
docker network prune [options]
```

- Example

```
# remove all unused networks without asking
docker network prune --force
# remove all network satisfying a filter
docker network prune --filter driver=bridge
```

Persistent Data

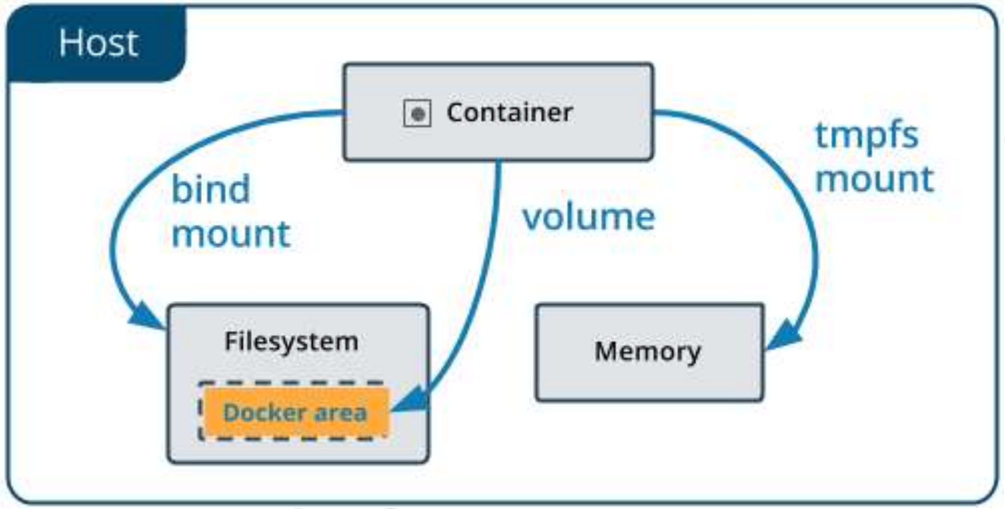
Volumes: Overview and Usage

Volume Overview

- Allow for external data in containers
- Two types
  - Data volumes
  - Data volume containers
- Created upfront, during run or build phase (VOLUME command)
- Data volumes can be shared
- Data volumes persist
- Data volumes are not deleted automatically

Volume Overview #2

- **Bind Mounts** are dependent on the OS and file system structure
- **Volumes** are managed by Docker
- **tmpfs mount** is for non-persistent state data
- **--volume** (**-v**) is simpler, and **--mount** is more explicit and verbose



https://docs.docker.com/storage/volumes/

volume ls

- Purpose
  - List volumes
- Syntax

```
docker volume ls [options]
```

- Example

```
# list IDs of all volumes
docker volume ls -q
# list all volumes satisfying a filter
docker volume ls --filter driver=local
```

volume inspect

- Purpose
  - Display detailed information on one or more volumes
- Syntax

```
docker volume inspect [options] volume [volume]
```

- Example

```
# show details about volume test-vol
docker volume inspect test-vol
```

volume create

- Purpose
  - Create a volume
- Syntax

```
docker volume create [options] [volume]
```

- Example

```
# create local volume test-vol
docker volume create test-vol
# create local volume lv-1 with label
docker volume create lv-1 --label mode=dev
```

volume rm

- Purpose
  - Remove one or more volumes
- Syntax

```
docker volume rm [options] volume [volume]
```

- Example

```
# remove volume test-vol
docker volume rm test-vol
```

volume prune

- Purpose
  - Remove all unused volumes
- Syntax

```
docker volume prune [options]
```

- Example

```
# remove all unused volumes without asking
docker volume prune -f
# remove all volumes satisfying a filter
docker volume prune --filter driver=local
```

Practice: Networks & Volumes

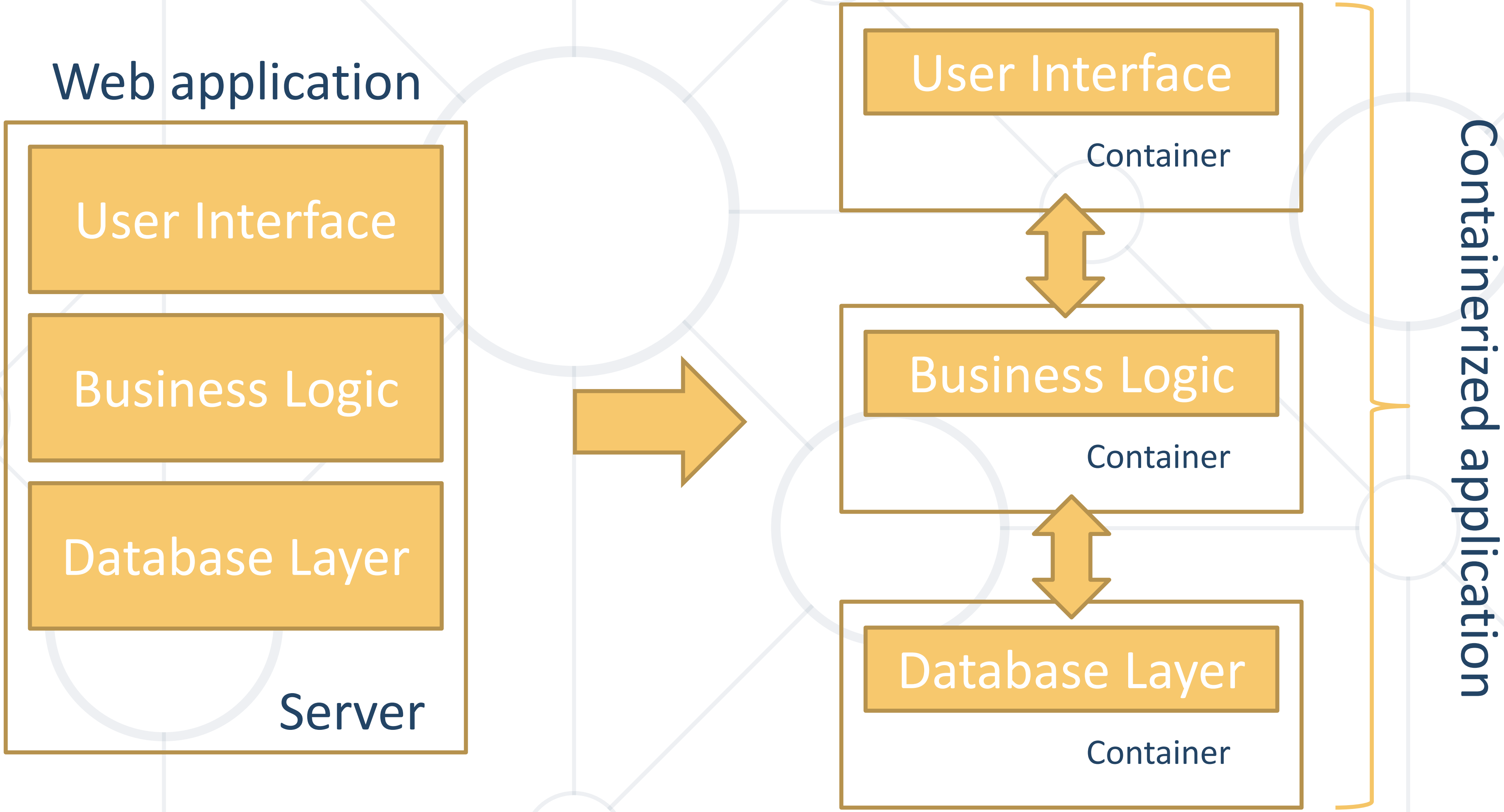
Live Demonstration in Class

DevOps

Distributed Applications

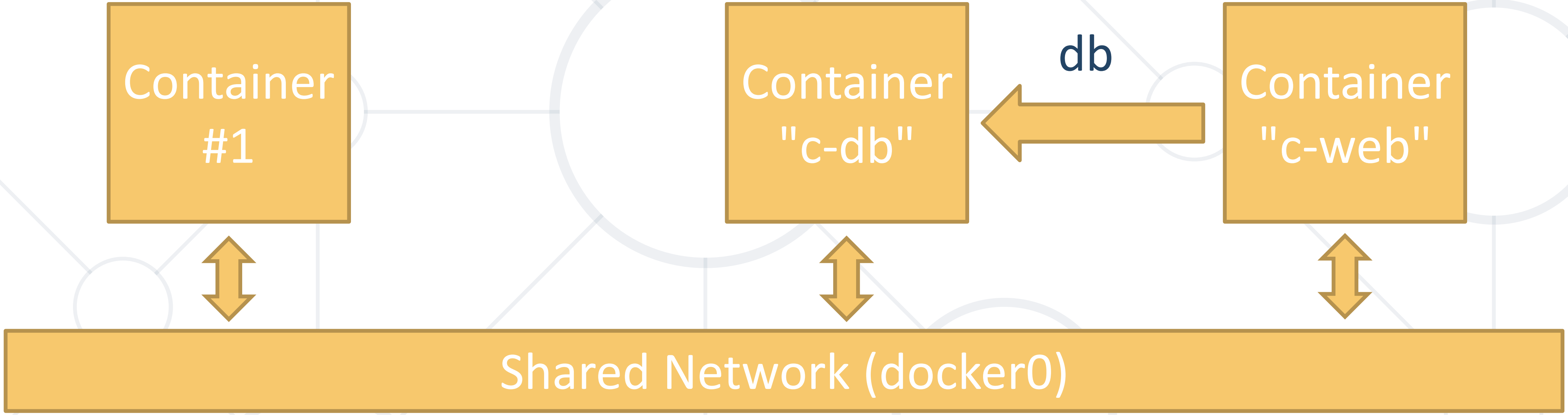
Overview and Implementation

Distributed Applications



Link Containers (Legacy) *

- By name alias



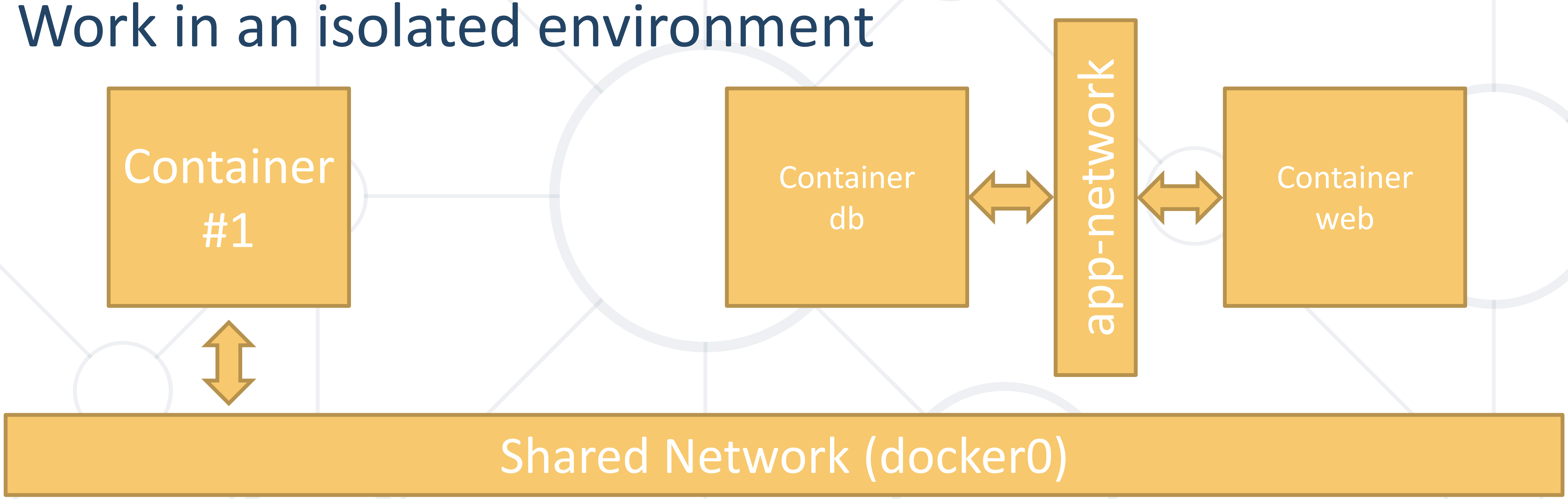
docker container run -d ... -p 8080:80 --link c-db:db ...

Linkage in the form **name:alias**

* Should be avoided as it is legacy and may be removed in future versions

Isolated Network

- Work in an isolated environment



docker container run -d ... -p 8080:80 --net app-network ...

Attached to the isolated network

DevOps

Docker Compose

Docker Compose

- Define and run **multi-container** Docker applications
- Multiple **isolated environments** on a single host
- **Preserve volume data** when containers are created
- Only recreate containers that **have changes**
- Supports **variables**
- Use cases
  - Development environments
  - Automated testing
  - Single host deployments

Configuration

Version (up to 3.9)
(optional since v1.27.0)

Services Definition

Networks Definition

```
version: "2.1"
services:
    com-php:
        build: ./web/
        ports:
            - 8080:80
        volumes:
            - "${PROJECT_ROOT}:/var/www/html:ro"
        networks:
            - com-network
networks:
    com-network:
```

docker-compose.yaml

```
PROJECT_ROOT=/home/docker/app

DB_ROOT_PASSWORD=12345
```

.env

docker compose build

- Purpose
  - Build or rebuild services
- Syntax

```
… build [options] [--build-arg key=val...] [SERVICE...]
```

- Example

```
# rebuild all services
docker compose build
# rebuild particular service with no-cache
docker compose build --no-chache my-php
```

docker compose up

- Purpose
  - Build, (re)create, start, and attach to containers for a service
- Syntax

```
… up [options] [--scale SERVICE=NUM...] [SERVICE...]
```

- Example

```
# start all containers and aggregate the output
docker compose up
# start all containers in a daemon mode
docker compose up -d
```

docker compose down

- Purpose
  - Stop containers and remove everything created by up
- Syntax

```
docker compose down [options]
```

- Example

```
# remove everything including all images
docker compose down --rmi all
# remove declared named volumes and anonymous volumes
docker compose down --volumes
```

docker compose ps

- Purpose
  - List containers
- Syntax

```
docker compose ps [options] [SERVICE...]
```

- Example

```
# list running containers
docker compose ps
# display ID for a particular container
docker compose ps -q com-php
```

docker compose logs

- Purpose
  - View output from containers
- Syntax

```
docker compose logs [options] [SERVICE...]
```

- Example

```
# view logs for all containers
docker compose logs
# follow the log for com-php service
docker compose logs -f com-php
```

docker compose start

- Purpose
  - Start existing containers
- Syntax

```
docker compose start [SERVICE...]
```

- Example

```
# start all containers
docker compose start
# start particular container / service
docker compose start com-php
```

docker compose stop

- Purpose
  - Stop running containers without removing them
- Syntax

```
docker compose stop [options] [SERVICE...]
```

- Example

```
# stop all containers
docker compose stop
# stop particular container / service with timeout
docker compose stop -t 20 com-php
```

docker compose rm

- Purpose
  - Remove stopped service containers
- Syntax

```
docker compose rm [options] [SERVICE...]
```

- Example

```
# remove all stopped containers
docker compose rm
# stop all containers and remove them without asking
docker compose rm -s -f
```

Practice: Docker Compose

Live Demonstration in Class

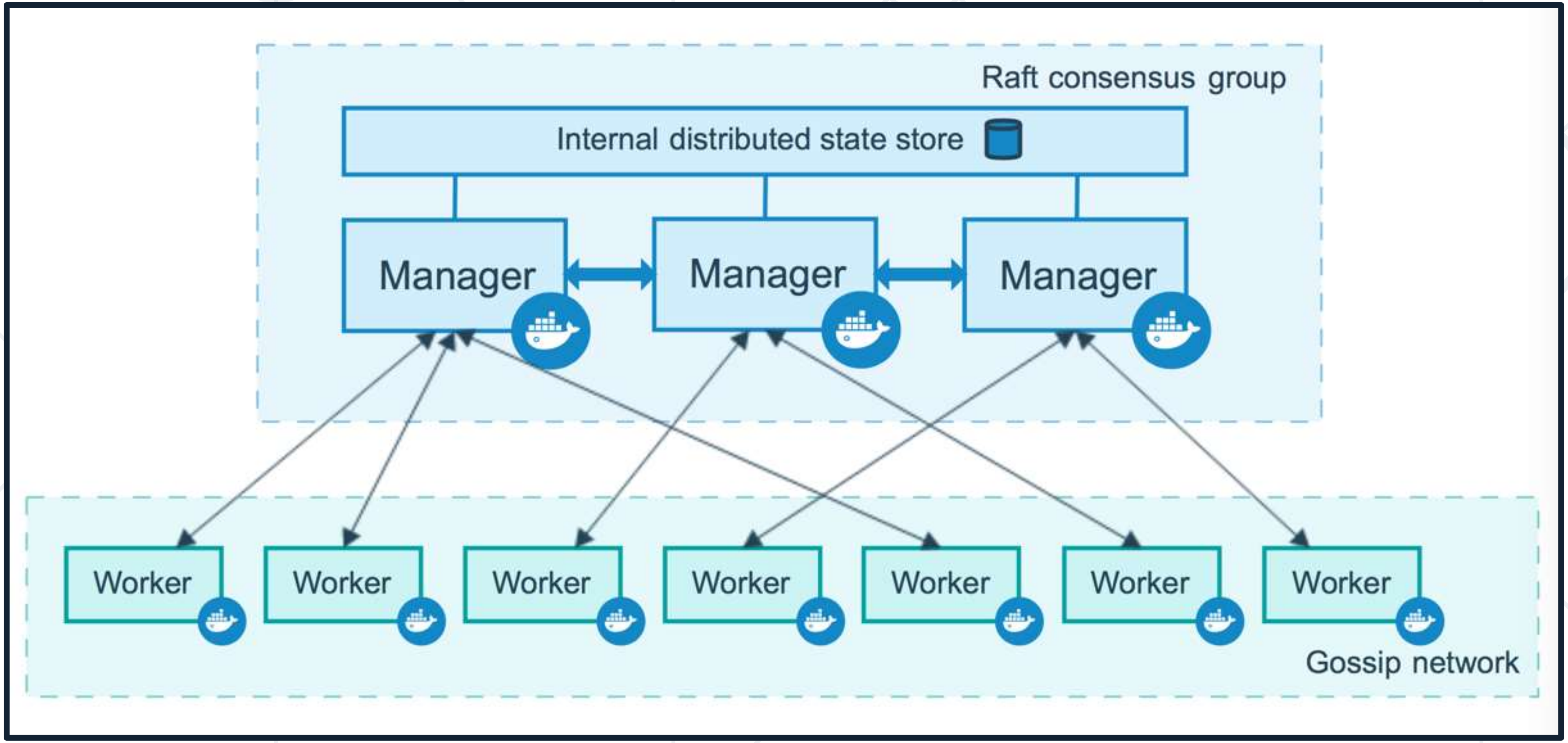
Docker Swarm

What is it? How it works?

What is it?

- Docker **engines joined** in a cluster
- Commands are executed by the **swarm manager**
- There could be more than one manager, but only one is **Leader**
- Nodes that are not managers are called **workers**
- **Both** managers and workers are **running containers**
- There are different **strategies** to run containers
- Nodes can be **physical** or **virtual**

The Big Picture*



* https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/

Three Simple Actions

- Initialize cluster
  - **`docker swarm init`**
- Join to a cluster
  - **`docker swarm join`**
- Leave a cluster
  - **`docker swarm leave`**

Deployment Options

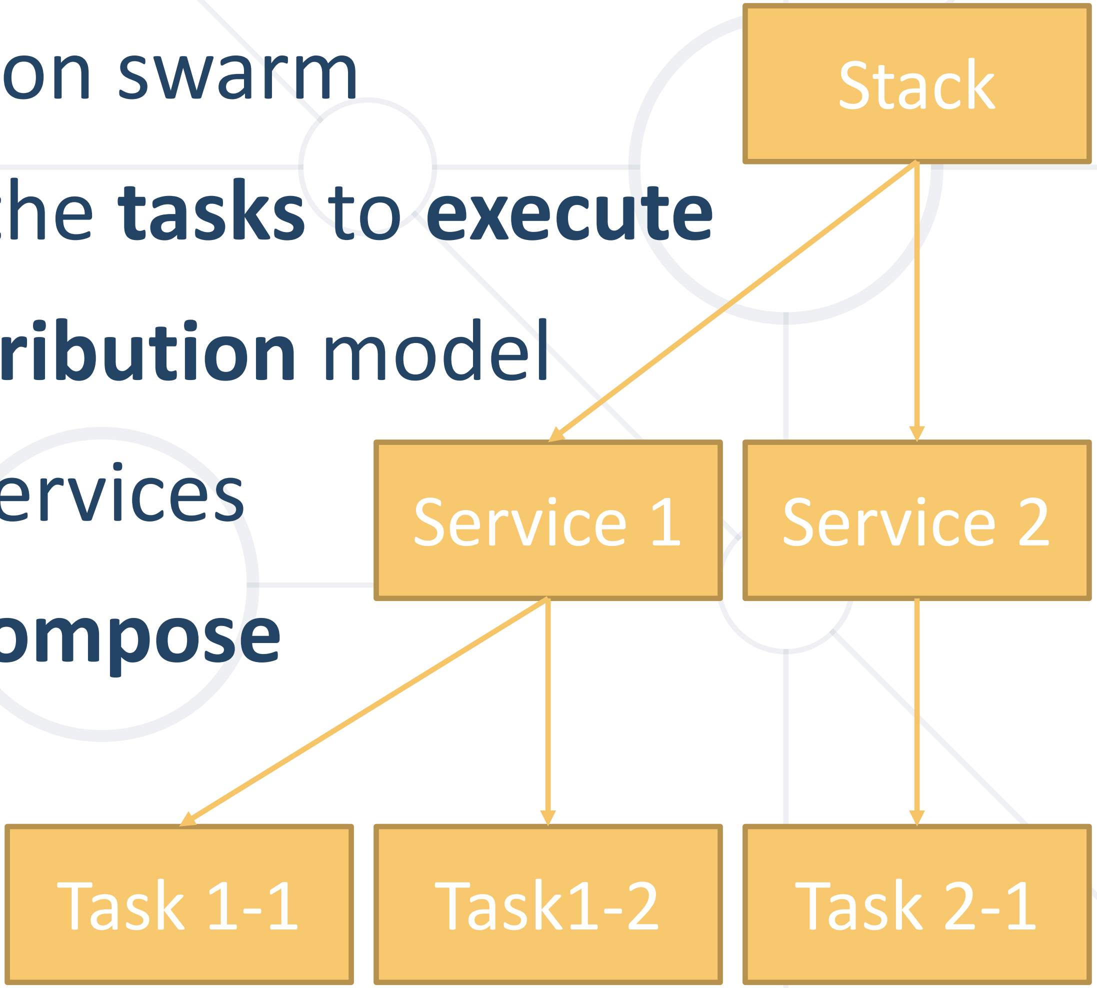
- Options
  - Cloud (Azure, AWS, …)
  - On-premise - VM, Bare-metal
- Deployment Strategy (on-premise)
  - (Semi) Manual } Today's practice
  - Automated } Additional practice – homework ☺

Stacks and Compose

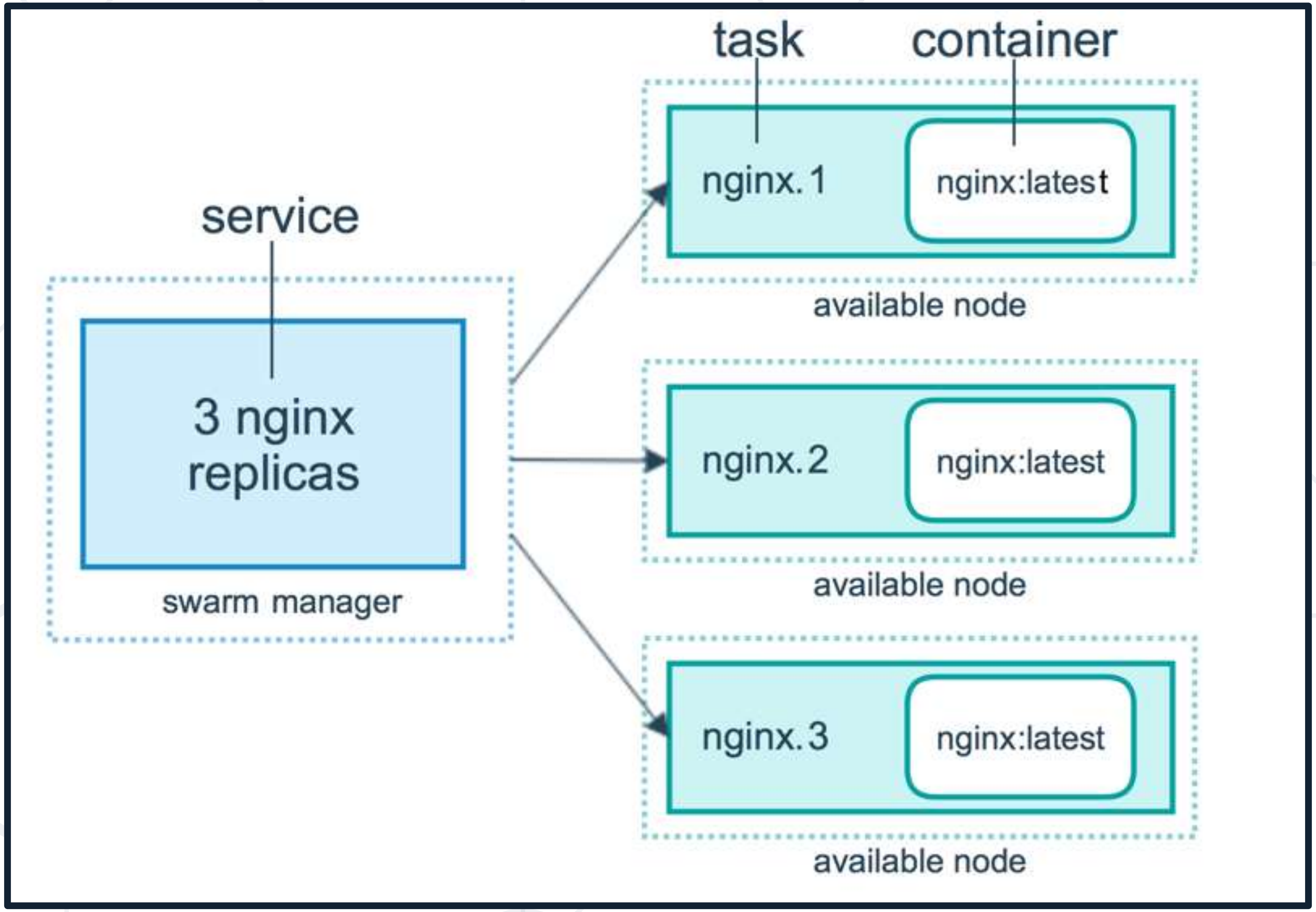
Deployment Automation

Tasks, Services, and Stacks

- Tasks are **units** of **work** distributed to nodes
- **Service** is an **application** deployed on swarm
- In fact, service is the **definition** of the **tasks** to **execute**
- **Replicated** and **global** services **distribution** model
- **Stacks** are **groups** of **interrelated** services
- Stacks are **deployed** with **docker-compose**

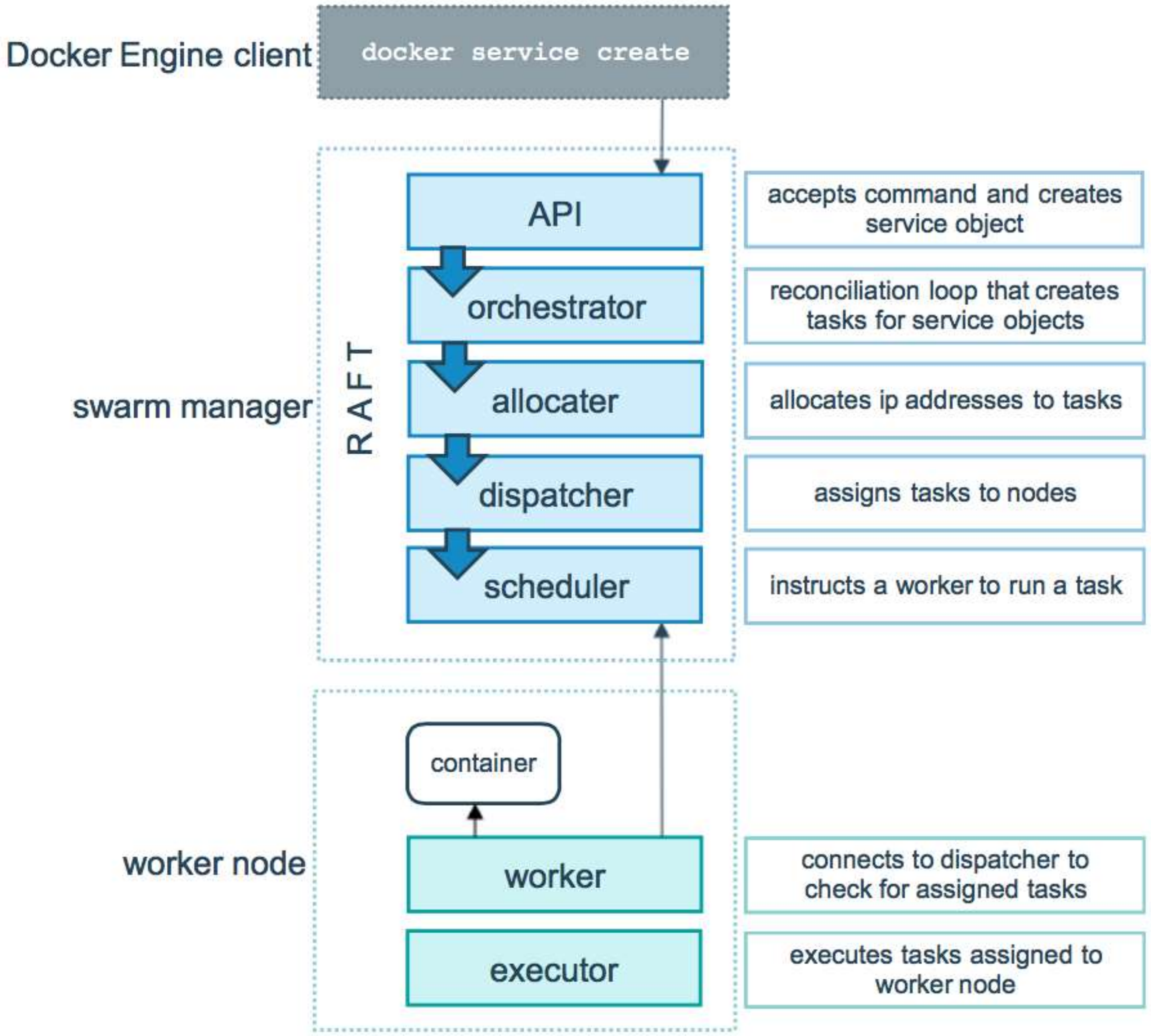


Containers, Tasks, and Services*



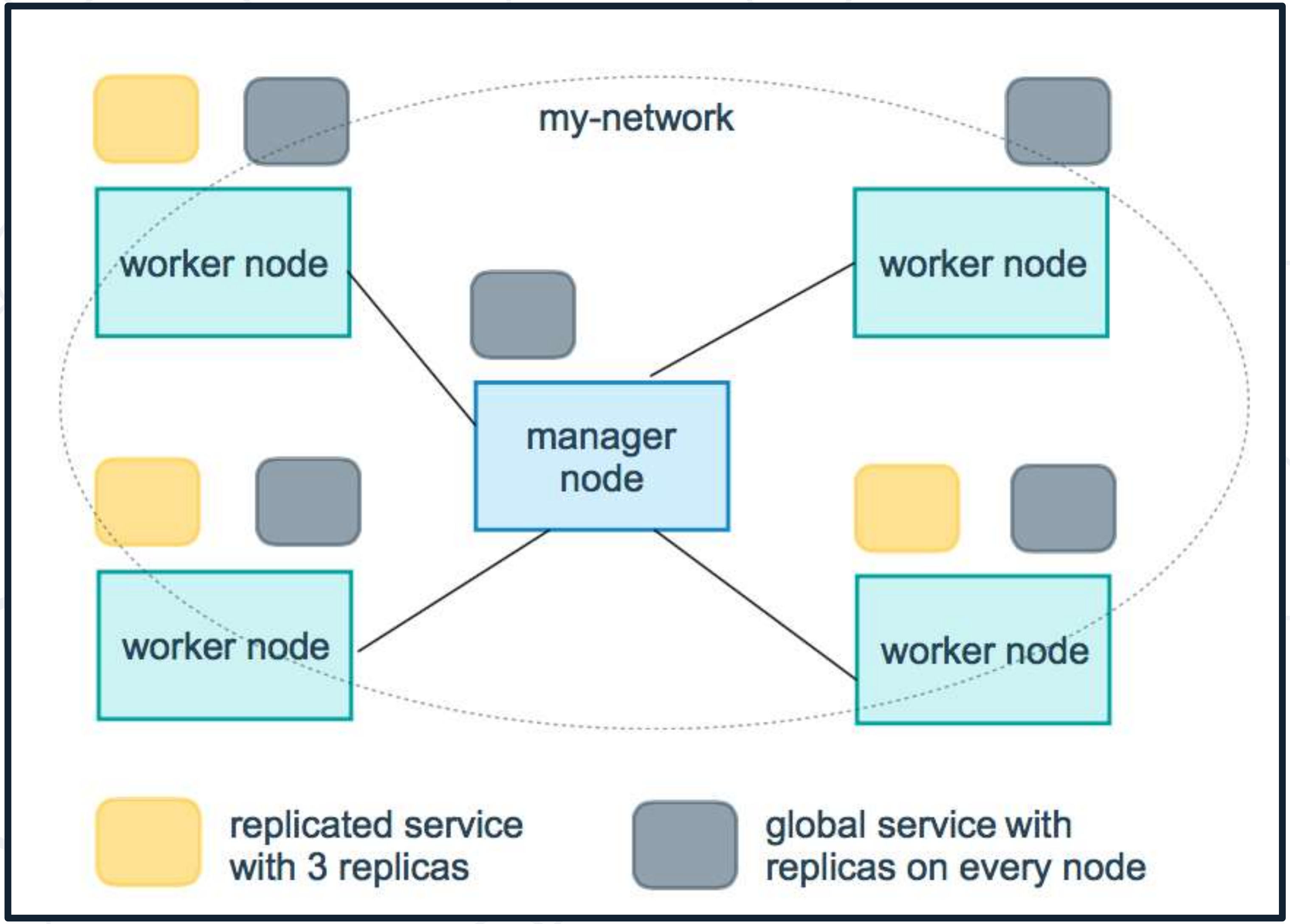
* https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/

Tasks and Scheduling*


Docker Engine client
docker service create
swarm manager
RAFT
API
orchestrator
allocater
dispatcher
scheduler
accepts command and creates service object
reconciliation loop that creates tasks for service objects
allocates ip addresses to tasks
assigns tasks to nodes
instructs a worker to run a task
worker node
container
worker
executor
connects to dispatcher to check for assigned tasks
executes tasks assigned to worker node

* https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/

Replicated and Global Services*


my-network
worker node
worker node
manager node
worker node
worker node
replicated service with 3 replicas
global service with replicas on every node

* https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/

DevOps

Sharing Data

Configuration and sensitive data

Share Data

- Parts of a service can be scheduled on different nodes
- They may be driven by external information
- We can store the data on every node and mount it from there
- While this is working, it is not the best solution
- Especially for **configuration data** and **sensitive information**
- For these we can use one of the two special object types
  - **Configs**
  - **Secrets**

Docker Configs

Not encrypted

- Configs are available only in Swarm mode
- Can be generic strings or binary data (up to 500 KB in size)
- Mounted directly in the container's filesystem
- Can be added or removed at any time
- Multiple services can share a config
- Managed via separate set of commands

```
docker config ACTION [options]
```

- Where ACTION is either **create**, **inspect**, **ls** or **rm**

https://docs.docker.com/engine/swarm/configs/

Docker Secrets

Encrypted

- Secrets are available only in Swarm mode
- Can be usernames, passwords, SSH keys, certificates, generic strings or binary data (up to 500 KB in size)
- Mounted via RAM disk to the containers
- Access to secrets and be added or removed at any time
- Services can share a secret
- Managed via separate set of commands

```
docker secret ACTION [options]
```

- Where ACTION is either **create**, **inspect**, **ls** or **rm**

https://docs.docker.com/engine/swarm/secrets/

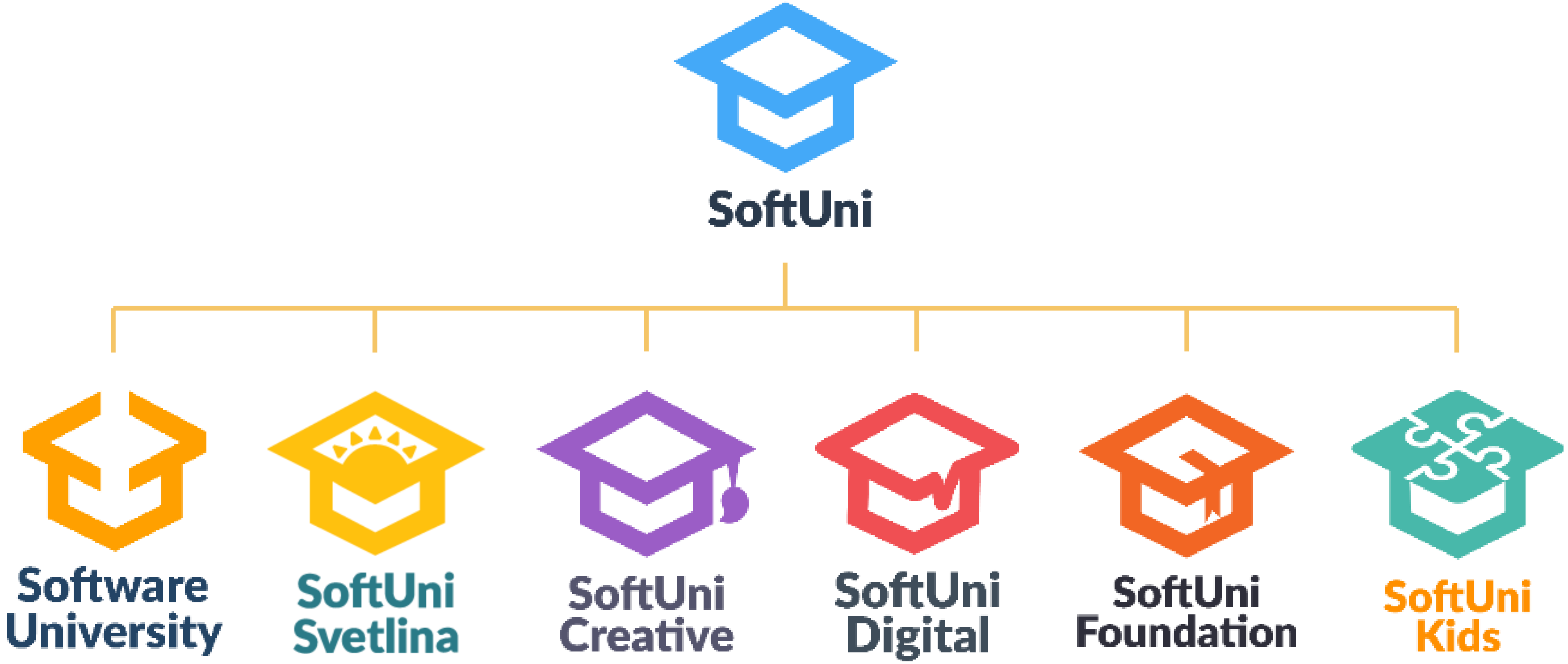
Practice: Swarm, Services and Stacks

Live Demonstration in Class

Summary

- Networking - inspect, tune, add, and remove
- Volumes - types, inspect and manage
- Distributed applications and Docker Compose
- Docker Swarm
  - How it works
  - Deployment options
  - Stacks and Compose

Questions?



SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

SCHWARZ

LIDL

Kaufland

SCHWARZ Produktion

pre zero

Coca-Cola HBC Bulgaria

POKERSTARS | CAREERS

Postbank

Решения за твоето утре

AMBITIONED

DXC TECHNOLOGY

SOFTWARE GROUP

Bosch.IO

INDEAVR

Serving the high achievers

DRAFT KINGS

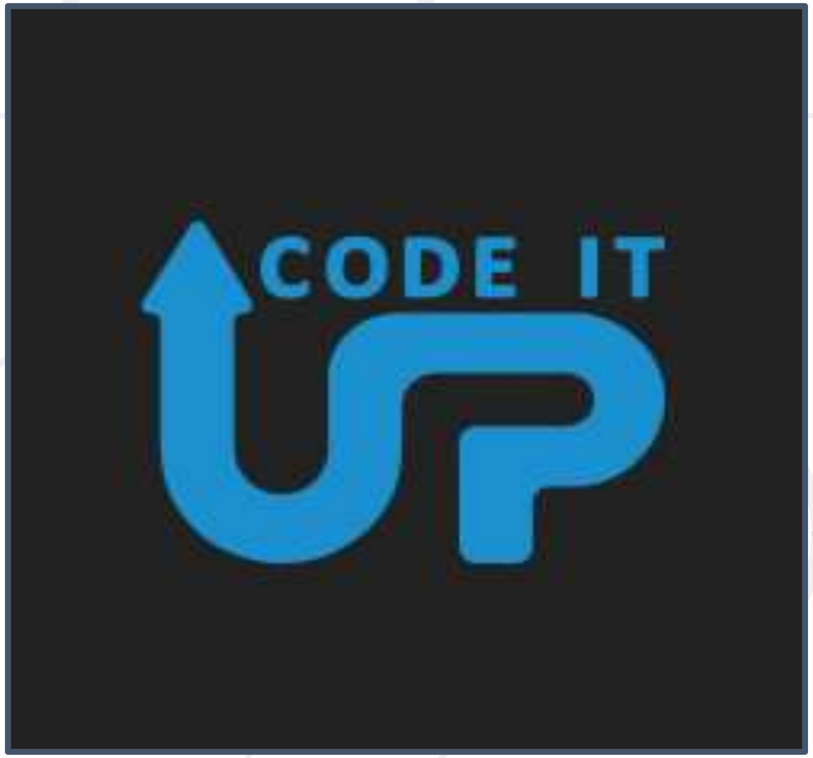
PHAR VISION

SmartIT

createX

SUPER HOSTING .BG

Educational Partners



License

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

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg