THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

CREATE CHANGE

Cloud Computing (INFS3208)

Lecture 4: Docker II:
Docker Compose and Docker Swarm

Lecturer: Dr Sen Wang

School of Electrical Engineering and Computer Science

Faculty of Engineering, Architecture and Information Technology

The University of Queensland

Re-cap

- Container
- What is Docker
- Basic concepts in Docker
  - Images
  - Containers
  - Registry
  - Layer architecture
- Docker Commands
- Containerisation and Dockerfile
  - Dokckerfile instructions

Lightweight VT, providing Isolation and Consistency

One popular Container implementation

READ-only template, small, no OS kernel, can be created by docker cmds and dockerfiles

Running instances of docker images, can be changed and committed. (1:many)

A place for sharing, public vs private, pull vs push

Max # of layers, readable only, good df writing practices

Cmds for images and containers (proficiency)

A plain file that contains INSTRUCTIONS, transparent (secure) and handy for sharing

Outline

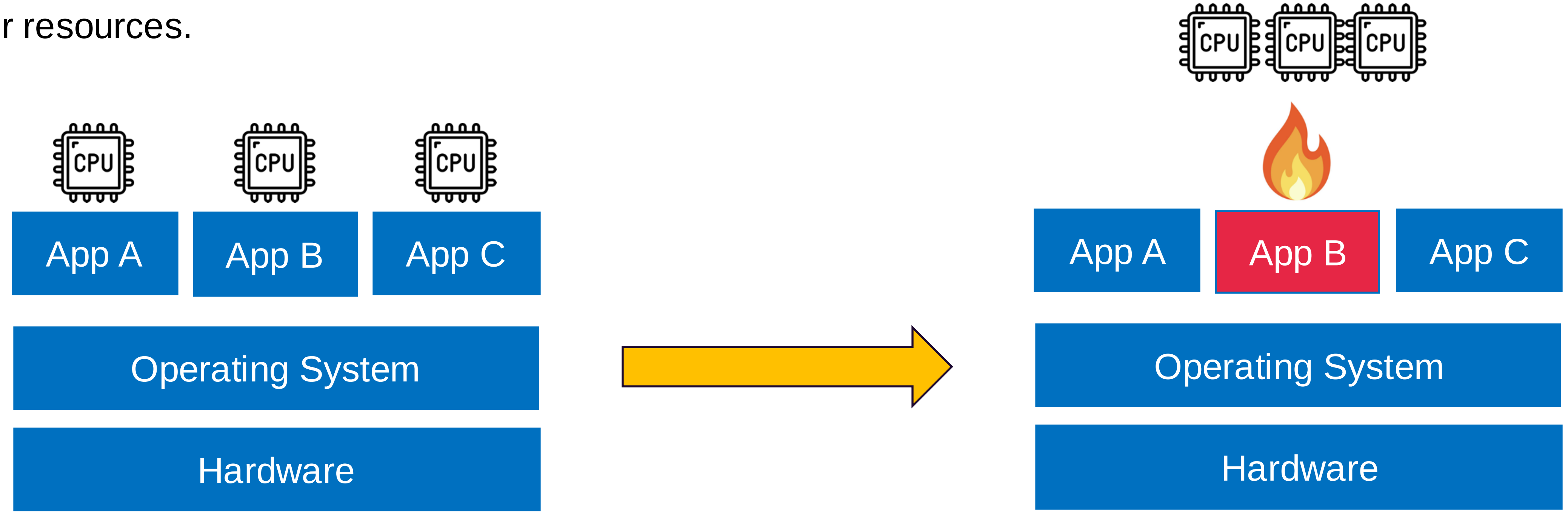
- ➡ Microservices
- Docker Compose
- Docker Swam
  - Docker Machine
  - Create a Swarm
  - Deploy Services to a Swarm
  - Deploy a Stack to a Swarm

An Example of Traditional Deployment

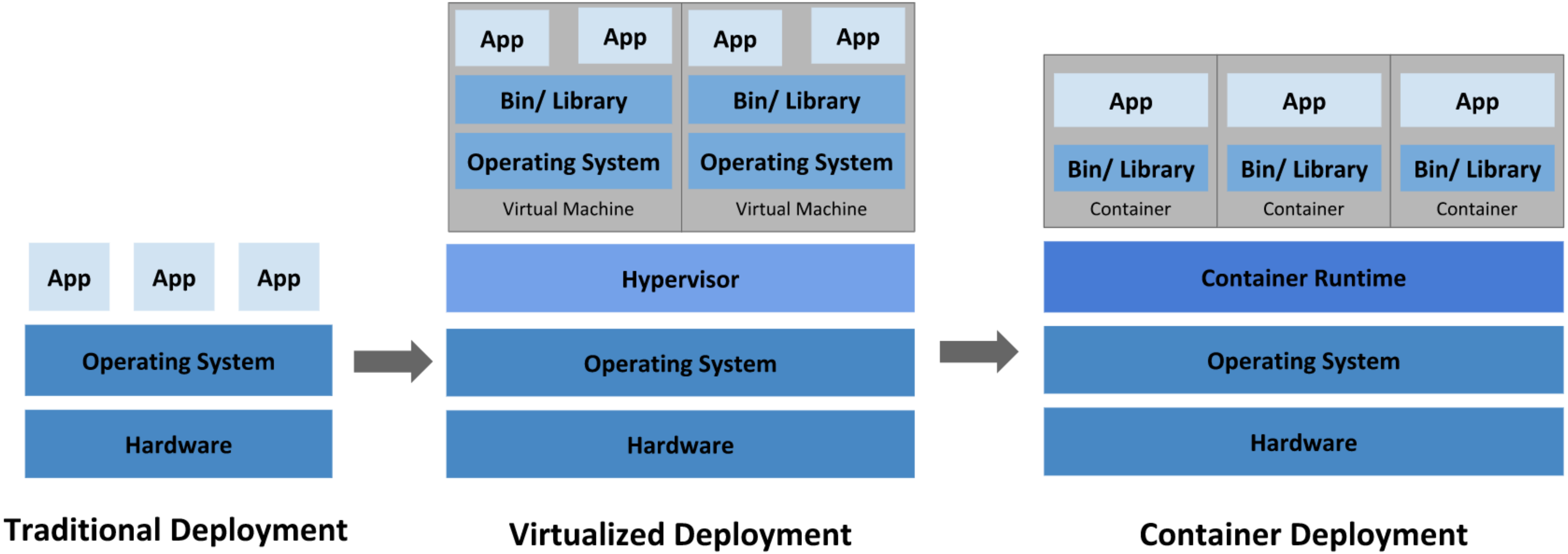
Imagine you have a physical server that hosts three different applications:

- **Application A**: A web server handling client requests.
- **Application B**: A data processing application that uses multi-threading.
- **Application C**: A database service handling storage and retrieval.

All three applications are running on the same operating system, sharing the same CPU, memory, and other resources.



Deployment Evolution

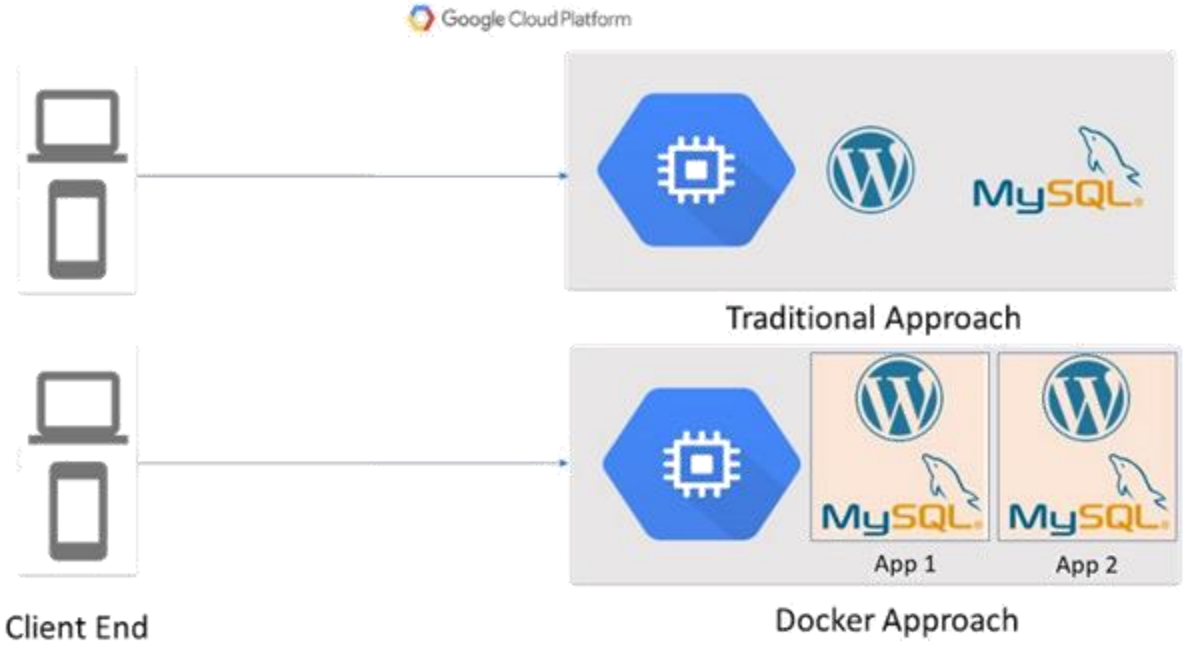


https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

One-for-all container?

Future weeks…

- Ubuntu base image
- MySQL RDBMS, WordPress
- …
- MongoDB/Redis
- Nginx or Apache HTTP Server
- Programming Language support: Java, Python, Go, etc.

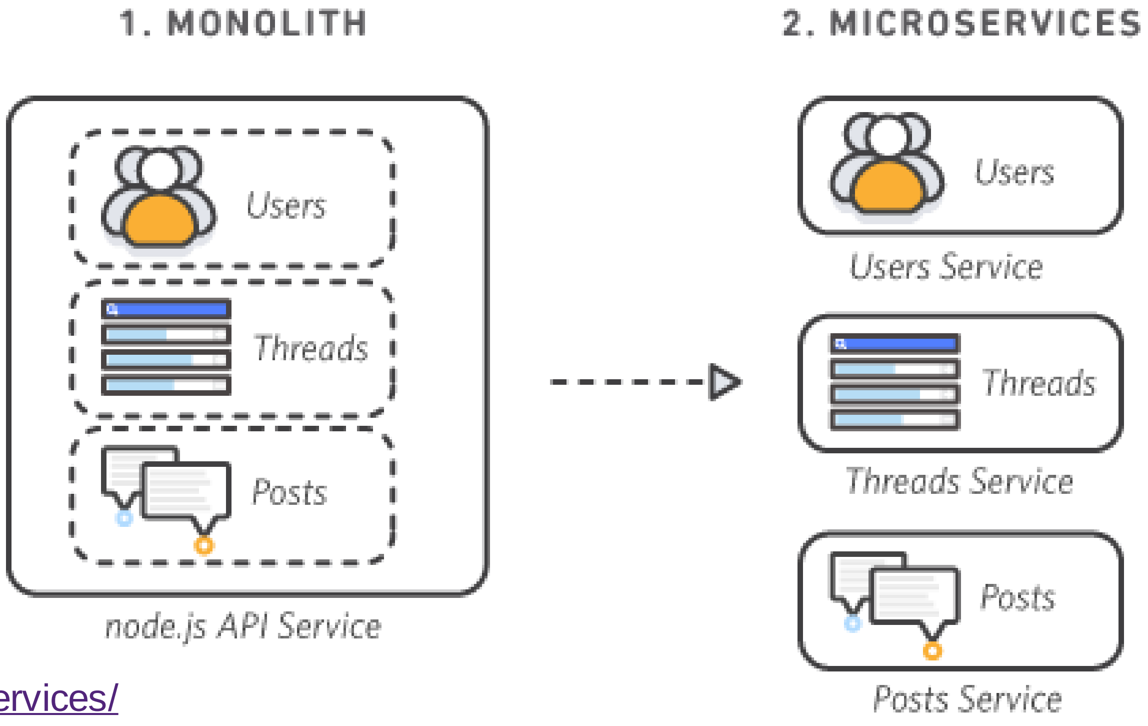


Should we have a one-for-all container for the project?

Image taken from web

Monolithic Architecture vs Microservice Architecture

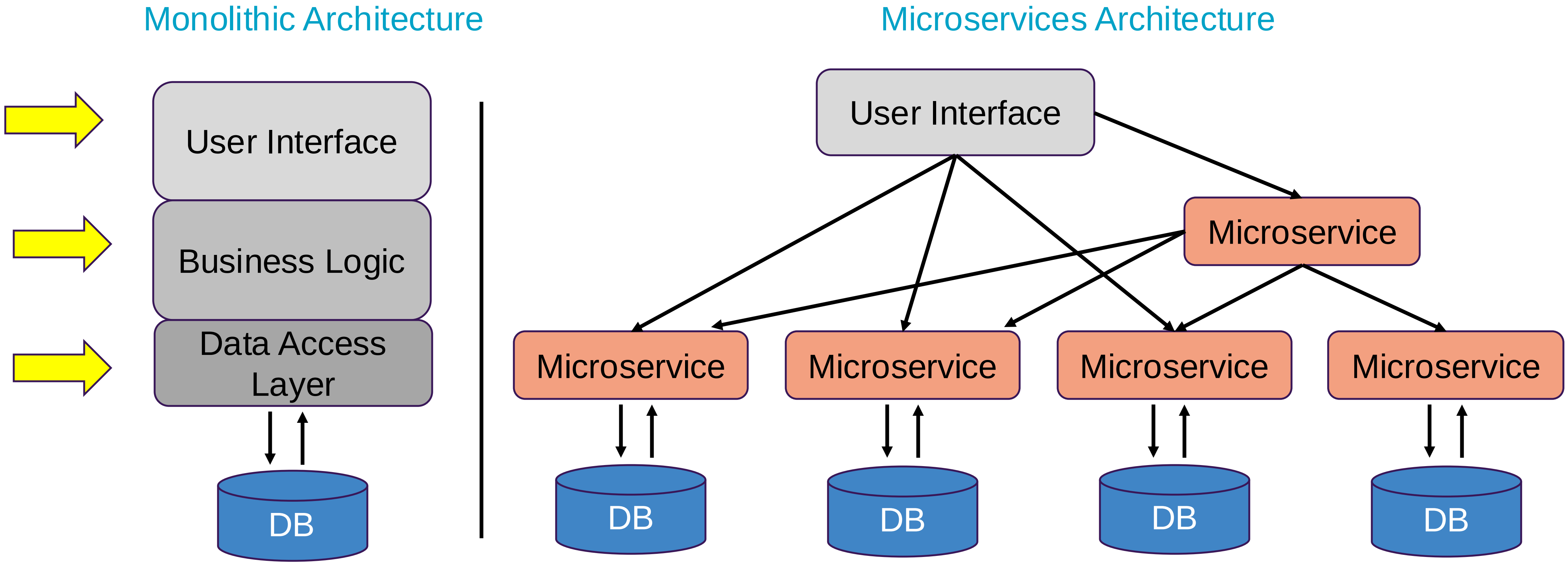
- Separate business logic functions
- Instead of one big program, several smaller applications
- Communicate via well-defined APIs – usually HTTP



https://aws.amazon.com/microservices/

Monolithic Architecture vs Microservice Architecture

Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs.



https://aws.amazon.com/microservices/?nc1=h_ls

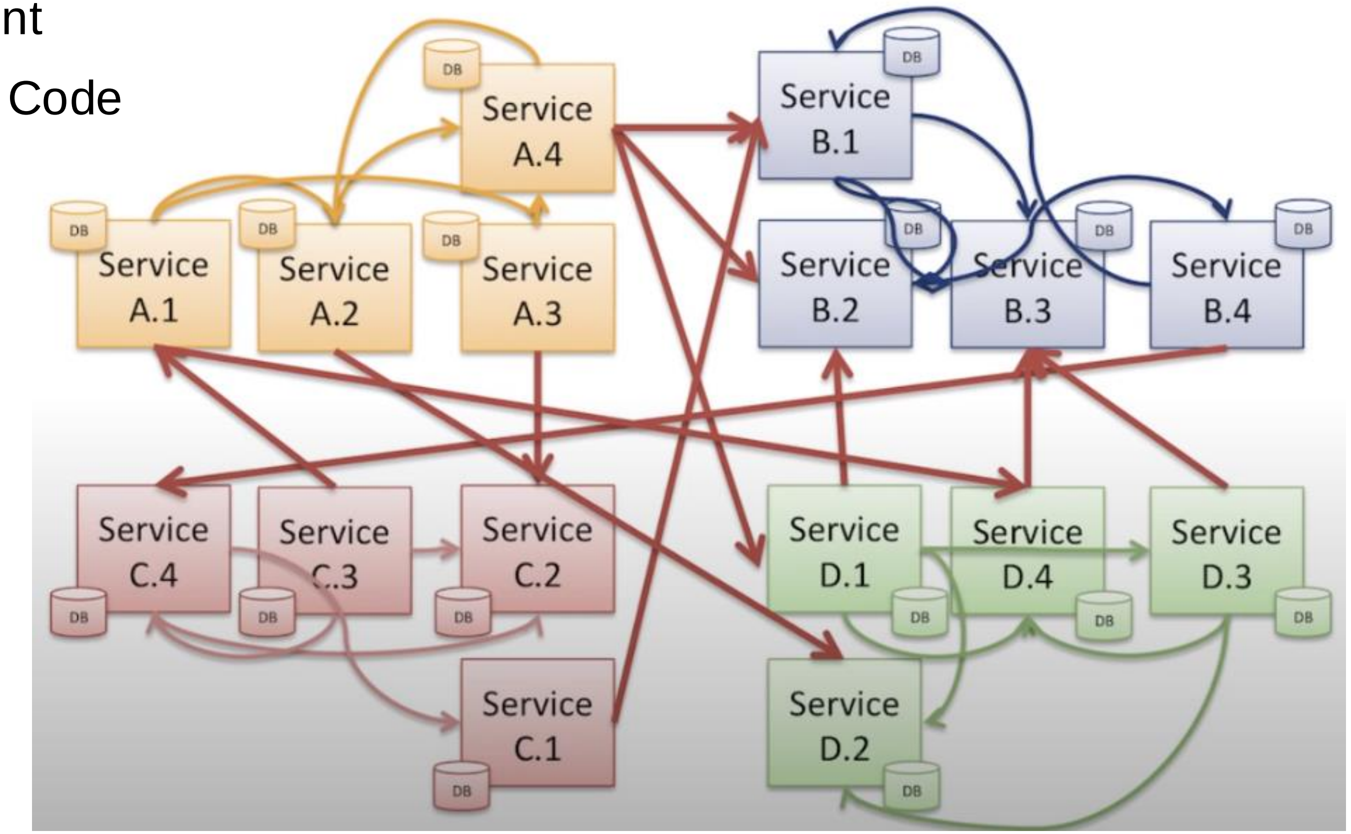
Microservices - Pros and Cons

Pros

- Technological Freedom - Language independent
- Easy Deployment - Fast iterations & Reusable Code
- Agility (Small teams)
- Resilience (Fault Isolation)
- Flexible Scaling (Scalable)

Cons

- Infrastructure Overhead
  - Servers and databases
- Complicated networking



https://aws.amazon.com/microservices/?nc1=h_ls

Three-Layer, Three-Tier, & Microservice Architectures

- **Microservices Architecture** breaks down an application into **small (atomic)**, **independent**, and **loosely coupled services**. Each service is responsible for **a specific piece of functionality** and can be **developed, deployed, and scaled independently**.
- **Three-Layer Architecture** (mainly taught in INFS3202/7202) is a **logical separation of** an application into three layers: Data Access Layer (Model), Presentation Layer (View), and Business Logic Layer (Controller).
  - Separation of Concerns, Maintainability, <u>Not Focused on Scalability</u>.
- **Three-Tier Architecture** is a **physical extension** of the three-layer arch, where each layer is **physically separated** and can be **hosted on different servers**: Presentation Tier, Application Tier, and Data Tier.
  - Physical Separation and Scalability

Differences:

- **Granularity**: Microservices break down the application into **finer-grained services**.
- **Scalability**: Microservices and Three-Tier architectures are designed with **scalability**.
- **Development Complexity**: Microservices may introduce additional complexity in deployment, monitoring, and inter-service communication.
- **Technology Stack**: Microservices allow for the use of different technologies for each service, while the other two architectures often use a consistent technology stack across layers or tiers.

https://aws.amazon.com/microservices/?nc1=h_ls

Question Revisited

Question: Should we have a One-for-all container for each project?

- Ubuntu base image
- MySQL RDBMS
- MongoDB/Redis
- Nginx or Apache HTTP Server
- Programming Language support: Java, Python, Go, etc.

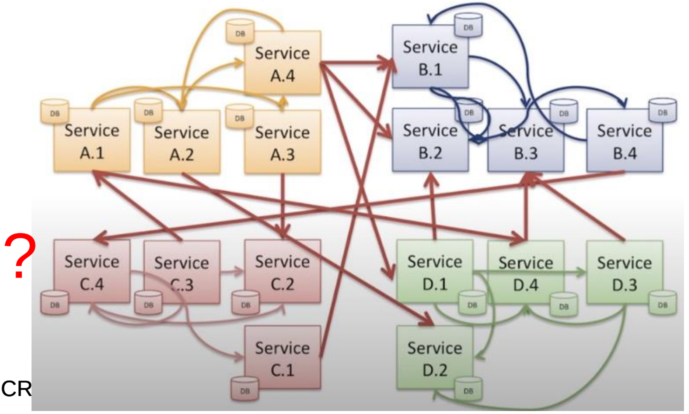
Issues: Monolithic-like container is chunky and not atomically scalable

Solution: containerise microservices and run multiple containers

Example: Personal Blog in Practical session

- WordPress container + MariaDB container

What if 100+ containers or even much more containers?



Outline

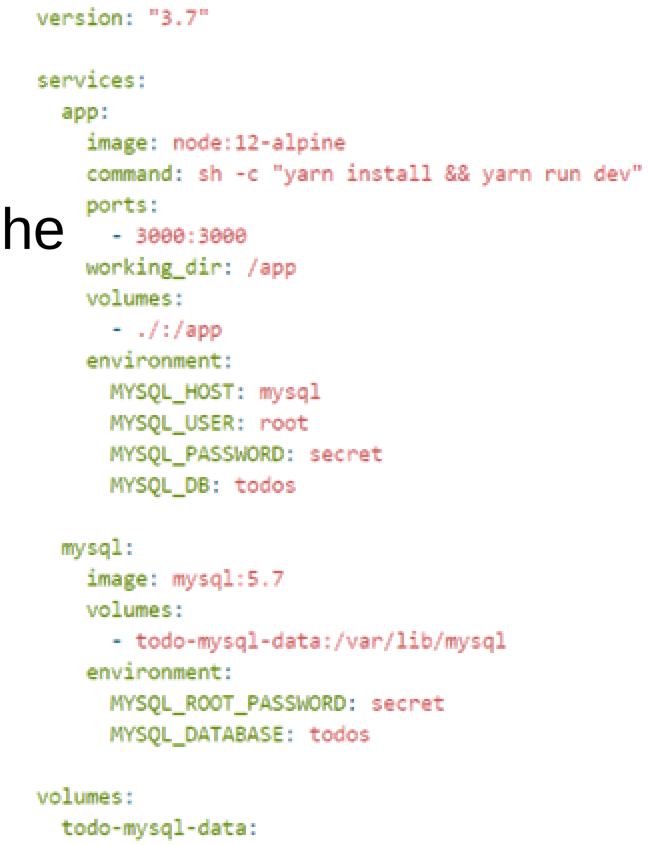
- Microservices
- ➡ Docker Compose
- Docker Swam
  - Docker Machine
  - Create a Swarm
  - Deploy Services to a Swarm
  - Deploy a Stack to a Swarm

Docker Compose

- Compose is a tool for defining and managing multi-container Docker applications.
- Use a Compose file (docker-compose.yml) to configure application's services.

docker-compose –f docker-compose.json up

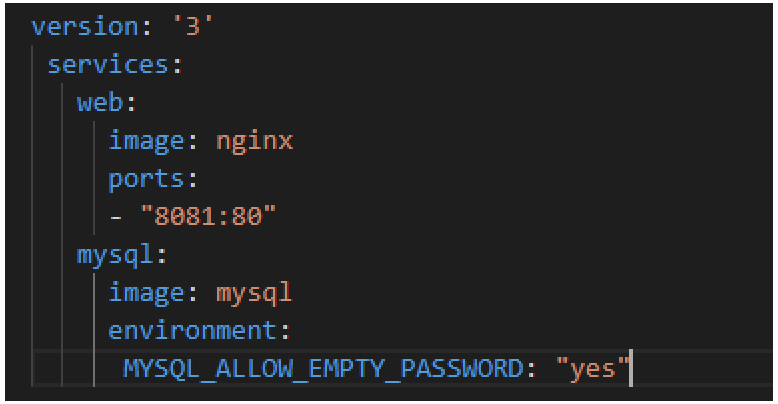
- Use a single command to create and start all the services from the configuration (docker-compose up).
- **Three-step process**:
  1. Define your app's environment with a Dockerfile or existing images
  2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
  3. Lastly, run docker-compose up, and Compose will start running your entire app.



```
version: "3.7"

services:
  app:
    image: node:12-alpine
    command: sh -c "yarn install && yarn run dev"
    ports:
      - 3000:3000
    working_dir: /app
    volumes:
      - ./:/app
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: secret
      MYSQL_DB: todos

  mysql:
    image: mysql:5.7
    volumes:
      - todo-mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: todos

volumes:
  todo-mysql-data:
```

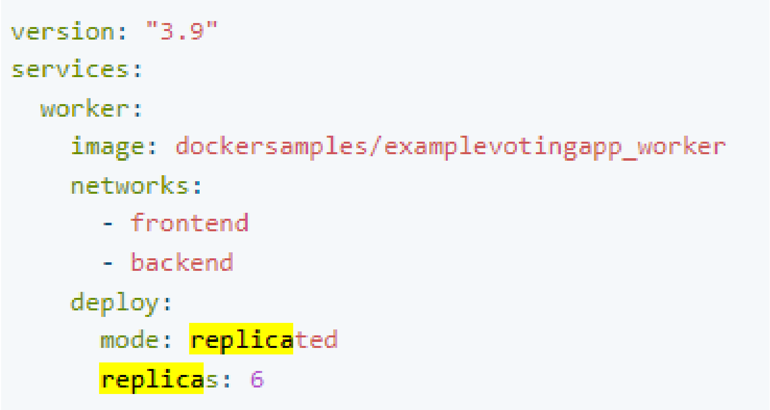
An example of docker-compose.yml

Docker Compose



```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "8081:80"
  mysql:
    image: mysql
    environment:
      MYSQL_ALLOW_EMPTY_PASSWORD: "yes"
```

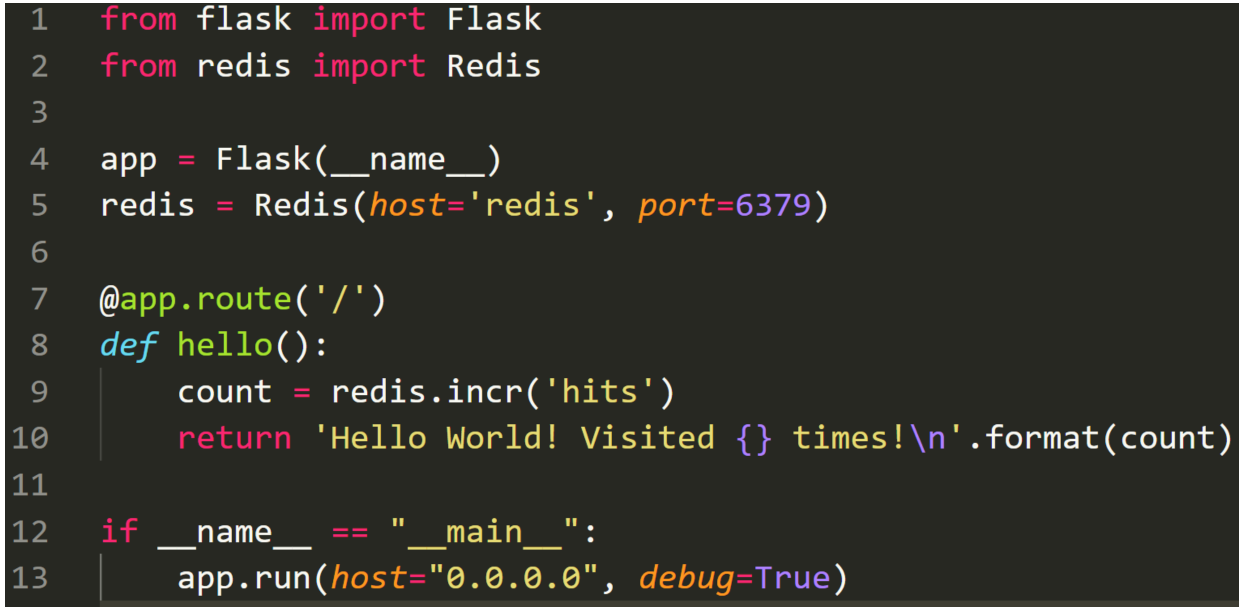
- Two basic concepts in docker-compose:
  - Service: running containers
    - One instance per image;
    - Multiple instances per image as replicas
  - Project: a complete business unit (consists of multiple linked containers)
- Compose is targeting at the project management and has commands for managing the whole lifecycle of your application:
  - Start, stop and rebuild services
  - View the status of running services
  - Stream the log output of running services



```
version: "3.9"
services:
  worker:
    image: dockersamples/examplevotingapp_worker
    networks:
      - frontend
      - backend
    deploy:
      mode: replicated
      replicas: 6
```

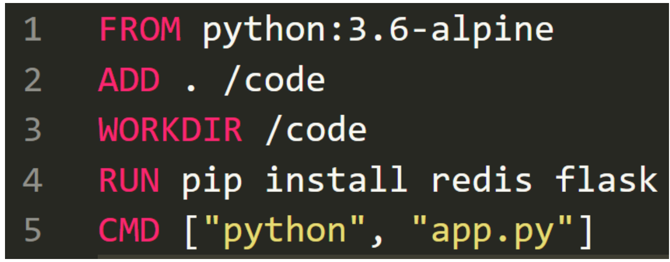
An Example of Docker Compose

- Web application in "app.py"



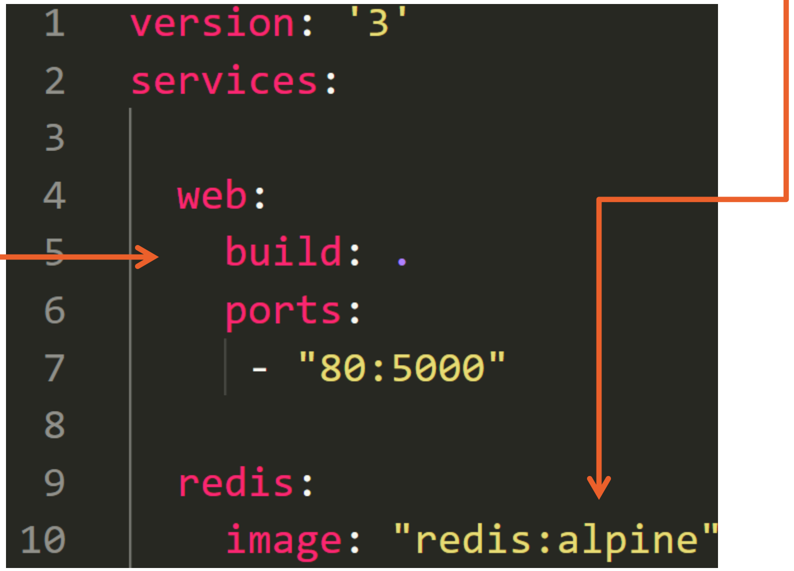
```python
from flask import Flask
from redis import Redis

app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    count = redis.incr('hits')
    return 'Hello World! Visited {} times!\n'.format(count)

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

- Dockerfile



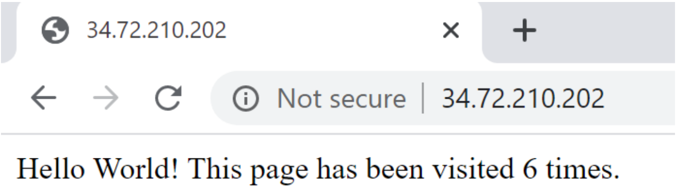
```dockerfile
FROM python:3.6-alpine
ADD . /code
WORKDIR /code
RUN pip install redis flask
CMD ["python", "app.py"]
```

- docker-compose.yml



```yaml
version: '3'
services:

  web:
    build: .
    ports:
    - "80:5000"

  redis:
    image: "redis:alpine"
```

- Run docker-compose up

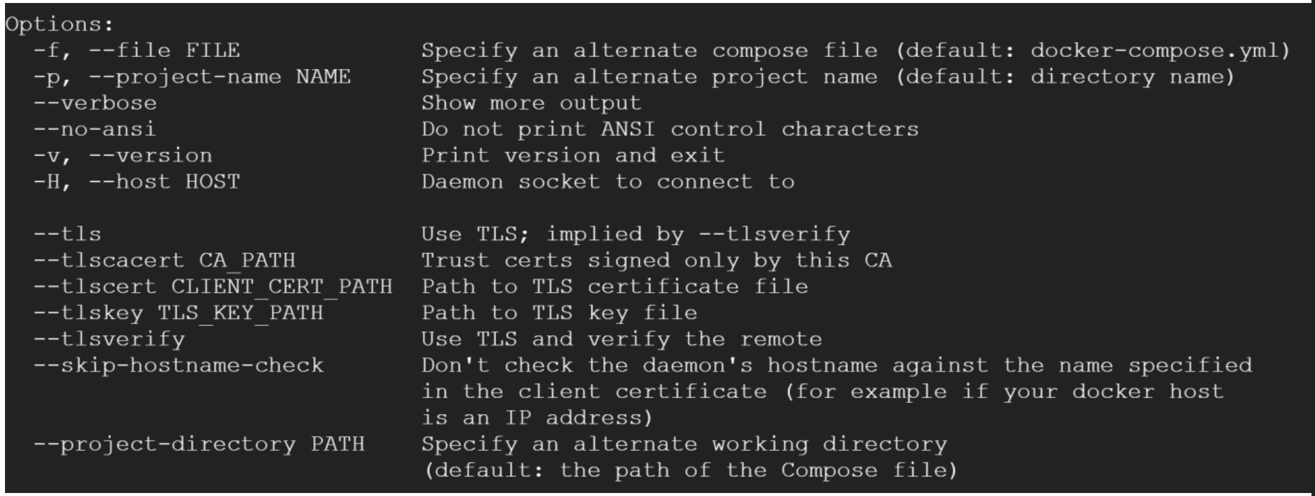


Hello World! This page has been visited 6 times.

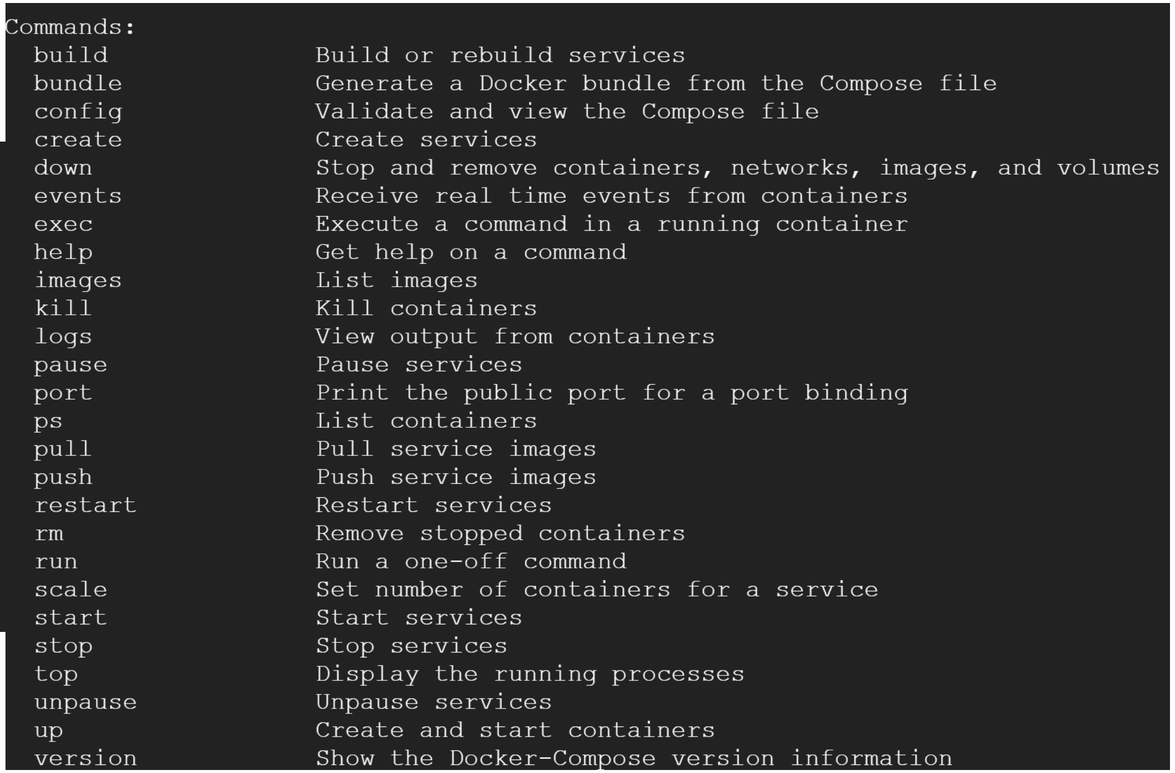
Compose commands

The objectives of most of compose commands are either project (by default) or services/containers within the project.

docker-compose [-f=<arg>...] [options] [COMMAND] [ARGS...]



```
Options:
  -f, --file FILE             Specify an alternate compose file (default: docker-compose.yml)
  -p, --project-name NAME     Specify an alternate project name (default: directory name)
  --verbose                   Show more output
  --no-ansi                   Do not print ANSI control characters
  -v, --version               Print version and exit
  -H, --host HOST             Daemon socket to connect to

  --tls                       Use TLS; implied by --tlsverify
  --tlscacert CA_PATH         Trust certs signed only by this CA
  --tlscert CLIENT_CERT_PATH  Path to TLS certificate file
  --tlskey TLS_KEY_PATH       Path to TLS key file
  --tlsverify                 Use TLS and verify the remote
  --skip-hostname-check       Don't check the daemon's hostname against the name specified
                              in the client certificate (for example if your docker host
                              is an IP address)
  --project-directory PATH    Specify an alternate working directory
                              (default: the path of the Compose file)
```



```
Commands:
  build              Build or rebuild services
  bundle             Generate a Docker bundle from the Compose file
  config             Validate and view the Compose file
  create             Create services
  down               Stop and remove containers, networks, images, and volumes
  events             Receive real time events from containers
  exec               Execute a command in a running container
  help               Get help on a command
  images             List images
  kill               Kill containers
  logs               View output from containers
  pause              Pause services
  port               Print the public port for a port binding
  ps                 List containers
  pull               Pull service images
  push               Push service images
  restart            Restart services
  rm                 Remove stopped containers
  run                Run a one-off command
  scale              Set number of containers for a service
  start              Start services
  stop               Stop services
  top                Display the running processes
  unpause            Unpause services
  up                 Create and start containers
  version            Show the Docker-Compose version information
```

Compose commands - up

docker-compose up [options] [--scale SERVICE=NUM...] [SERVICE...]

- Builds, (re)creates, starts, and attaches to containers for a service.
- Important options:
  - "-d": keep all the containers in yaml file running in the background.
  - "--build": force to rebuild the image
  - "--no-recreate": ignore existing running containers and start all the stopped containers.
  - "--force-recreate": force Compose to stop and recreate all containers.
  - "--no-deps -d <SERVICE_NAME>": stop, recreate, and restart a specific container.

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

Compose commands – change state

For all services…

docker-compose build

docker-compose down [options]

- Stops containers and removes containers, networks, volumes, and images created by up.

For a certain service…

docker-compose run [SERVICE…]

- creates containers from images built for the services mentioned in the compose file

docker-compose start [SERVICE…]

- Starts runs any stopped containers for a service.

docker-compose stop, restart, kill, rm (un)paused

https://docs.docker.com/compose/reference/overview/

Compose commands – check information

docker-compose logs [options] [SERVICE…]

- Displays log output from services.

docker-compose images

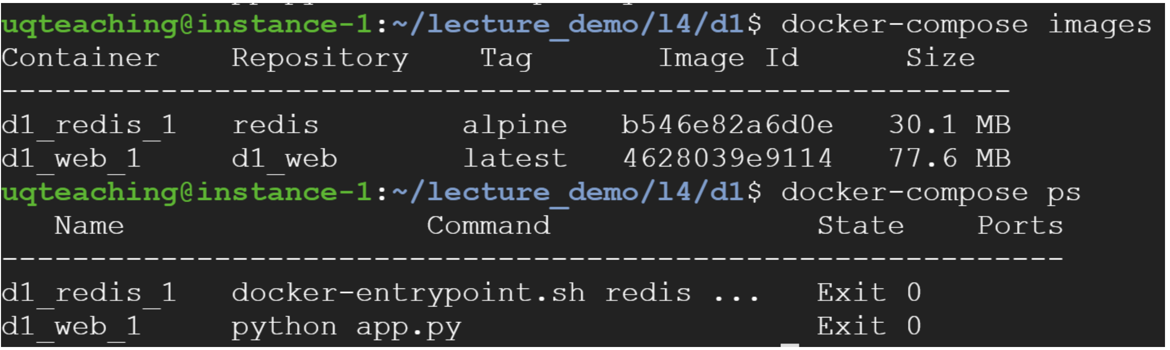
- Lists images included in the docker-compose.yml file.

docker-compose ps [options] [SERVICE…]

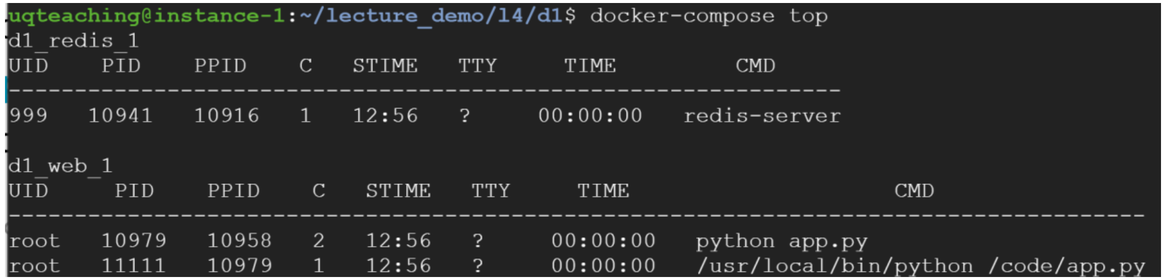
- Lists containers.

docker-compose top

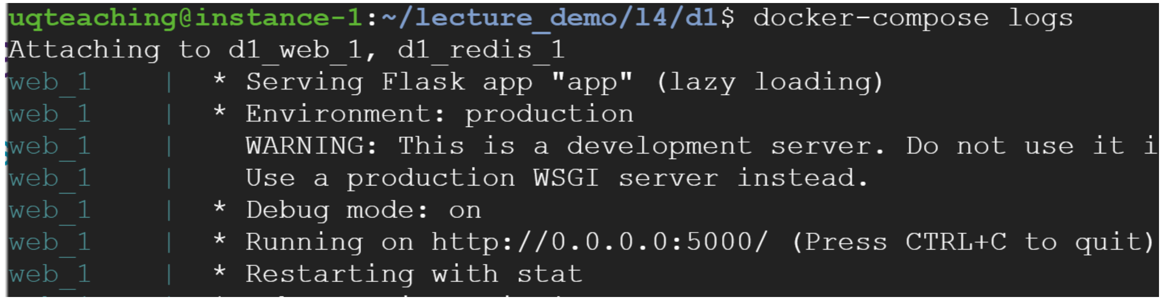
- Displays the running processes.



```
uqteaching@instance-1:~/lecture_demo/l4/d1$ docker-compose images
Container     Repository    Tag       Image Id       Size
-----------------------------------------------------------
d1_redis_1    redis         alpine    b546e82a6d0e   30.1 MB
d1_web_1      d1_web        latest    4628039e9114   77.6 MB
uqteaching@instance-1:~/lecture_demo/l4/d1$ docker-compose ps
   Name                  Command               State    Ports
--------------------------------------------------------------
d1_redis_1    docker-entrypoint.sh redis ...   Exit 0
d1_web_1      python app.py                    Exit 0
```



```
uqteaching@instance-1:~/lecture_demo/l4/d1$ docker-compose top
d1_redis_1
UID    PID     PPID    C   STIME   TTY     TIME        CMD
-----------------------------------------------------------------
999    10941   10916   1   12:56   ?     00:00:00   redis-server

d1_web_1
UID    PID     PPID    C   STIME   TTY     TIME                    CMD
-------------------------------------------------------------------------------------
root   10979   10958   2   12:56   ?     00:00:00   python app.py
root   11111   10979   1   12:56   ?     00:00:00   /usr/local/bin/python /code/app.py
```

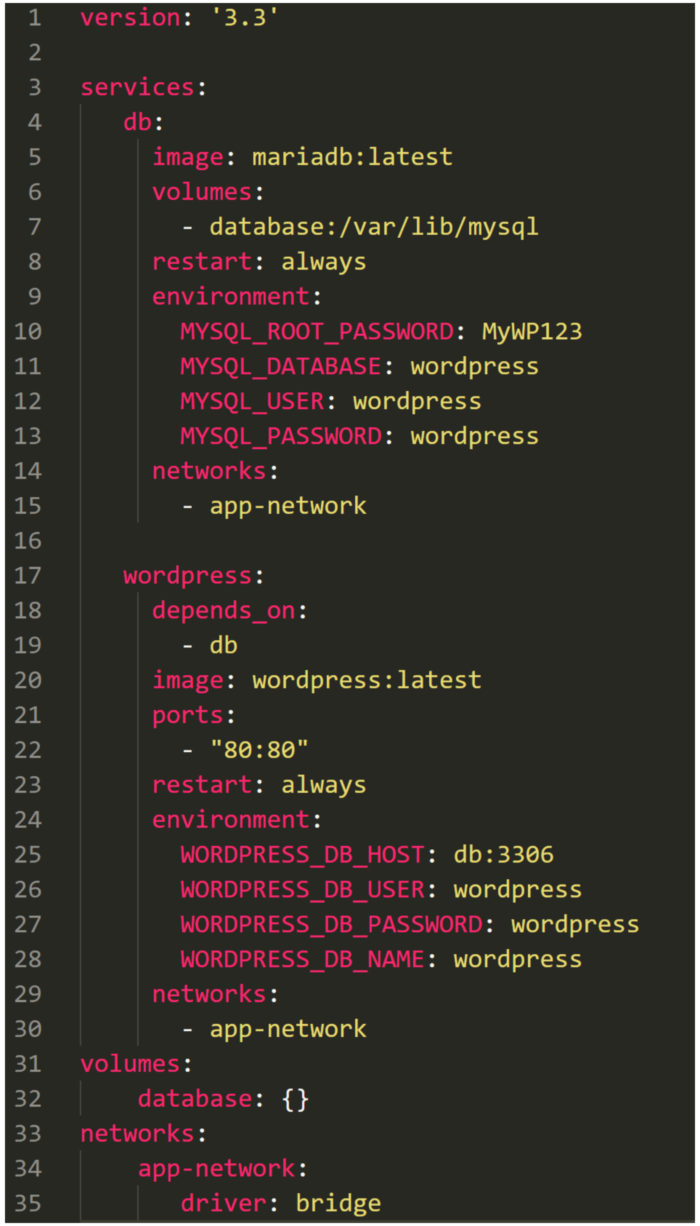


```
uqteaching@instance-1:~/lecture_demo/l4/d1$ docker-compose logs
Attaching to d1_web_1, d1_redis_1
web_1     |  * Serving Flask app "app" (lazy loading)
web_1     |  * Environment: production
web_1     |    WARNING: This is a development server. Do not use it i
web_1     |    Use a production WSGI server instead.
web_1     |  * Debug mode: on
web_1     |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
web_1     |  * Restarting with stat
```

Compose files

Multi-service applications are defined in a configuration file

- docker-compose.yml (by default, but can use "-f" to read .yml file with a customised file name).
- YAML is a superset of JSON.
- consists of multiple layers that are split using tab stops or spaces
- Four main components in each Compose-File:
  - Compose file's version
  - Services (containers)
  - Volumes (storage)
  - Networks (linking)



```yaml
version: '3.3'

services:
    db:
      image: mariadb:latest
      volumes:
        - database:/var/lib/mysql
      restart: always
      environment:
        MYSQL_ROOT_PASSWORD: MyWP123
        MYSQL_DATABASE: wordpress
        MYSQL_USER: wordpress
        MYSQL_PASSWORD: wordpress
      networks:
        - app-network

    wordpress:
      depends_on:
        - db
      image: wordpress:latest
      ports:
        - "80:80"
      restart: always
      environment:
        WORDPRESS_DB_HOST: db:3306
        WORDPRESS_DB_USER: wordpress
        WORDPRESS_DB_PASSWORD: wordpress
        WORDPRESS_DB_NAME: wordpress
      networks:
        - app-network
volumes:
    database: {}
networks:
    app-network:
        driver: bridge
```

Compose files

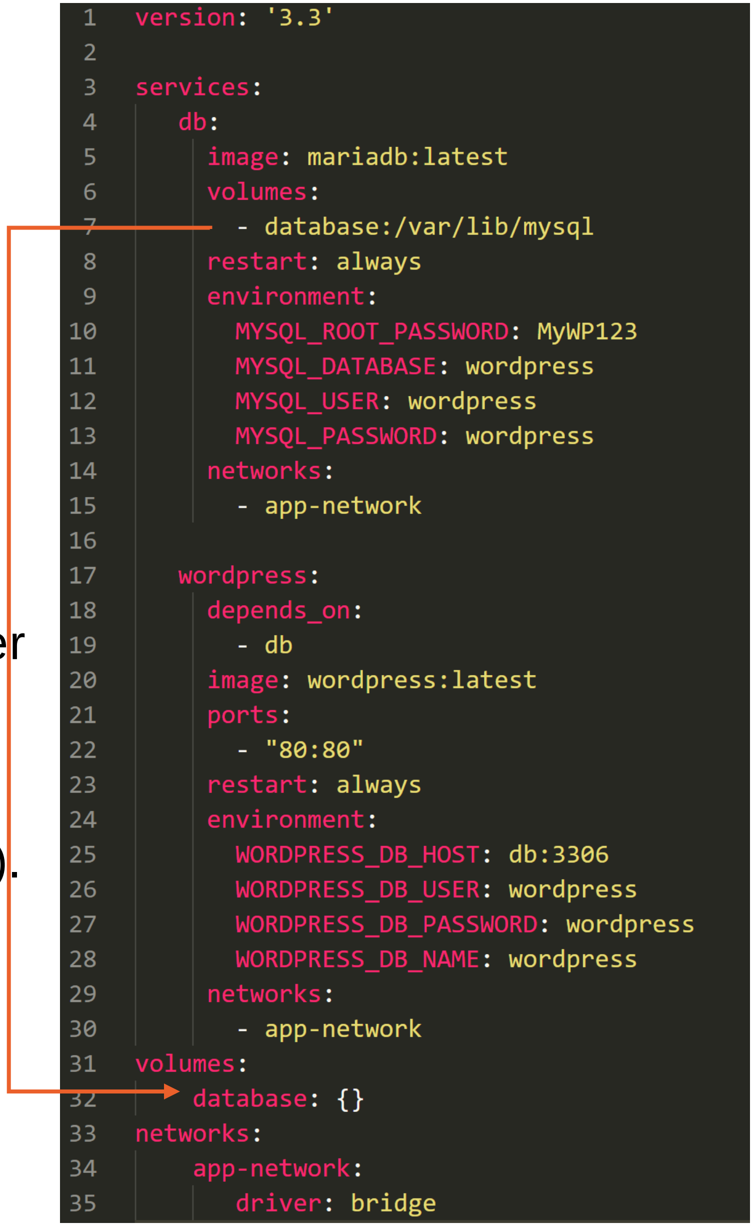
Second-level definitions in “services”:

- “db” (relational database - MySQL)
- “wordpress” (Free and open-source CMS - WP).
- The service name can be arbitrary.

Instructions in “db“:

- “image”:pulls the latest image of MySQL and uses it to create the container for this service

  “build” can be used with “Dockerfile” to build the customised image
- “volumes”: mounts folder database (source) to /var/lib/mysql (in container).
- “restart”: specifies restart policy for containers.
- “environment”: Add environment variables.



```
version: '3.3'

services:
   db:
     image: mariadb:latest
     volumes:
       - database:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: MyWP123
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress
     networks:
       - app-network

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "80:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
       WORDPRESS_DB_NAME: wordpress
     networks:
       - app-network
volumes:
    database: {}
networks:
    app-network:
       driver: bridge
```

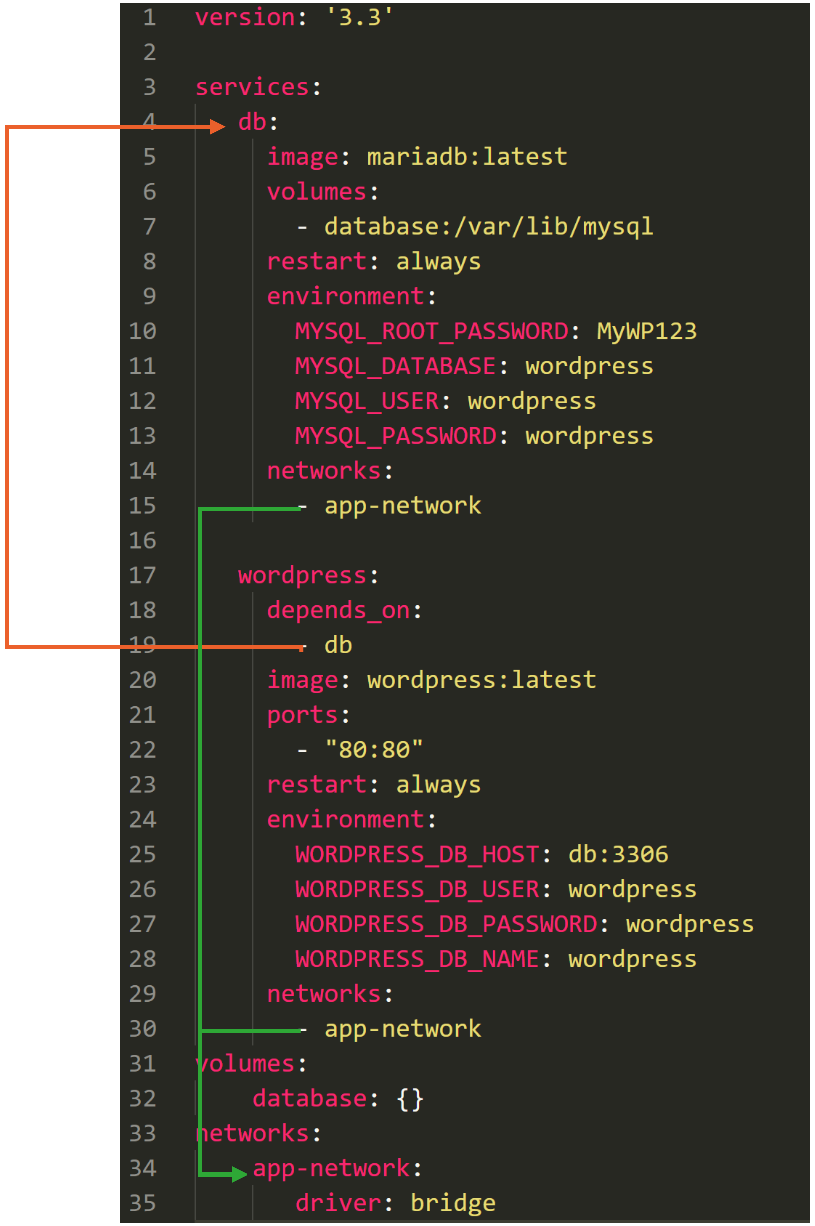
https://hub.docker.com/_/mysql
https://hub.docker.com/_/wordpress

Compose files

Instructions in “wordpress”:

- “depends_on”: expresses dependency between services. Service dependencies cause the following behaviours:
  - docker-compose up starts services in dependency order.
    - E.g. “db” is started before “wordpress”.
  - docker-compose up [SERVICE] automatically includes SERVICE’s dependencies.
    - E.g. docker-compose up wordpress also creates and starts “db”.
  - docker-compose stop stops services in dependency order.
    - E.g. “wordpress” is stopped before “db”.
- “ports”: exposes or maps ports
  - (HOST_PORT:CONTAINER_PORT)
    - E.g. “ports: 8000:80”
  - Port range: “-” and protocal: “/”
    - E.g. “127.0.0.1:8000-8009:5000-5009” and “6000:6000/tcp”
- “networks”: defines the communication rules between containers

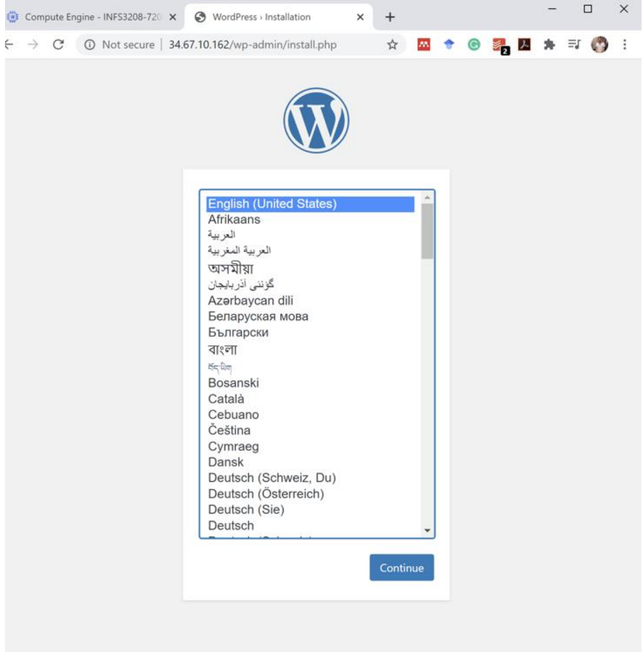
https://docs.docker.com/compose/
https://hub.docker.com/



```
version: '3.3'

services:
  db:
    image: mariadb:latest
    volumes:
      - database:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: MyWP123
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
    networks:
      - app-network

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "80:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
    networks:
      - app-network
volumes:
  database: {}
networks:
  app-network:
    driver: bridge
```

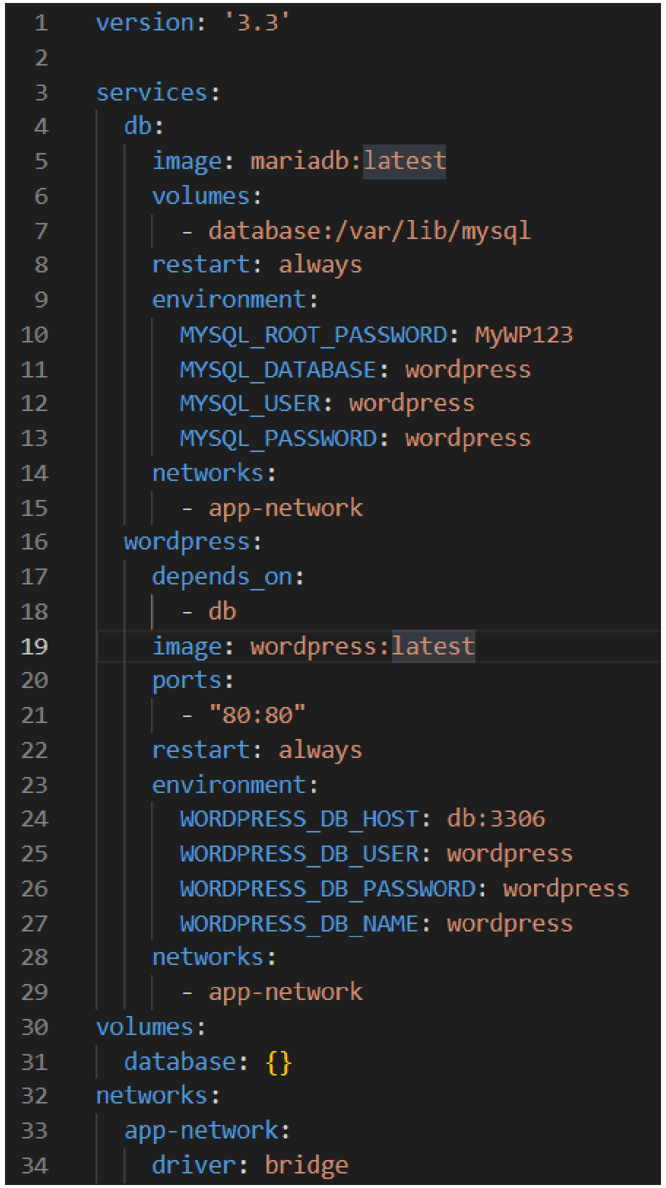
Docker-compose Demo

MariaDB/MySQL + WordPress



Compute Engine - INFS3208-72 × | WordPress › Installation ×

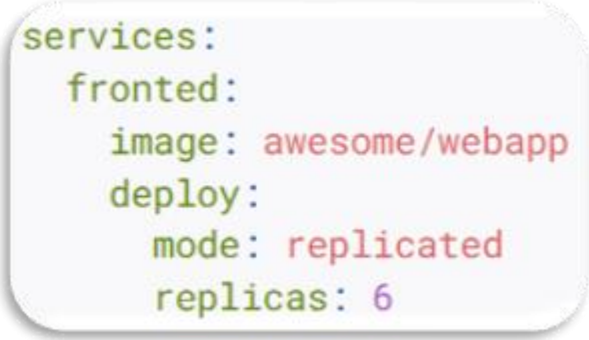
Not secure | 34.67.10.162/wp-admin/install.php



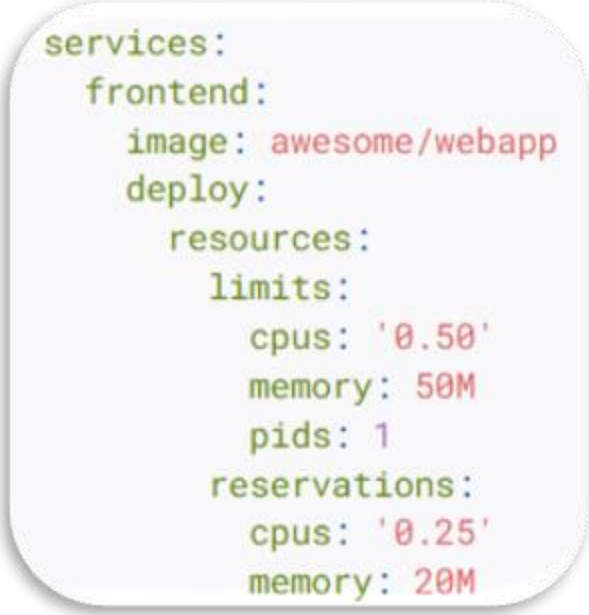
```yaml
version: '3.3'

services:
  db:
    image: mariadb:latest
    volumes:
      - database:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: MyWP123
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
    networks:
      - app-network
  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "80:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
    networks:
      - app-network
volumes:
  database: {}
networks:
  app-network:
    driver: bridge
```

Benefits

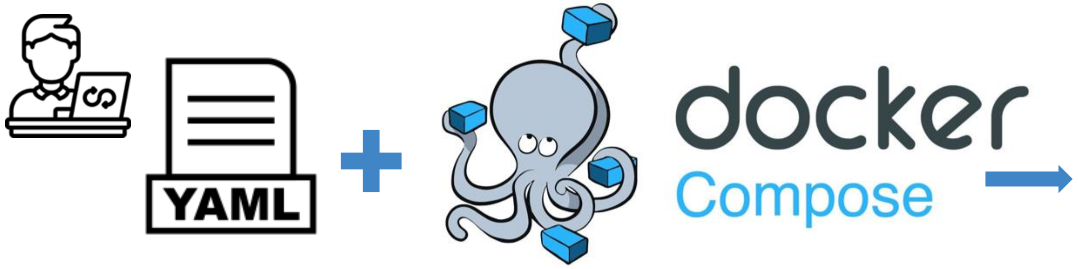
- **Simplified Configuration**: With Docker Compose, you can describe your system's **entire configuration**, including services, networks, and volumes, in a single YAML file, making it easier to **understand** and **maintain** your system.
- **Ease of Deployment**: A single command (**docker-compose up**) with a configuration file, which ensures that all containers are started in the **correct order**, with the **proper settings (env variables, networks, volumes, etc)**.
- **Scalability and Load Balancing**: With a few simple commands, Docker Compose enables you to scale specific services up or down to handle changes in load.
- **Resource Allocation**: You can specify **CPU, memory limits, and other resources** on a per-service basis, giving you fine-grained control over resource allocation.
- ...

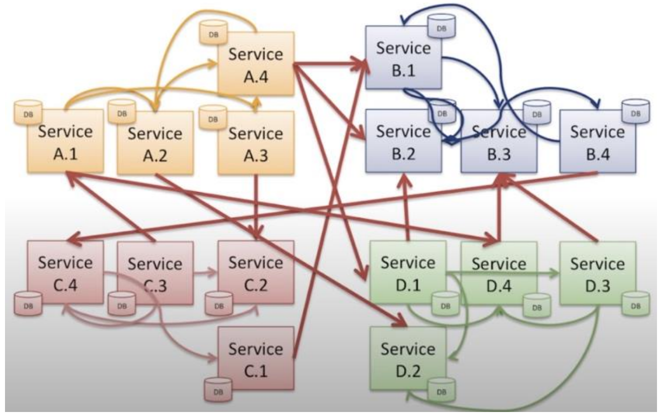


```
services:
  fronted:
    image: awesome/webapp
    deploy:
      mode: replicated
      replicas: 6
```



```
services:
  frontend:
    image: awesome/webapp
    deploy:
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
          pids: 1
        reservations:
          cpus: '0.25'
          memory: 20M
```





https://docs.docker.com/compose/compose-file/deploy/

Outline

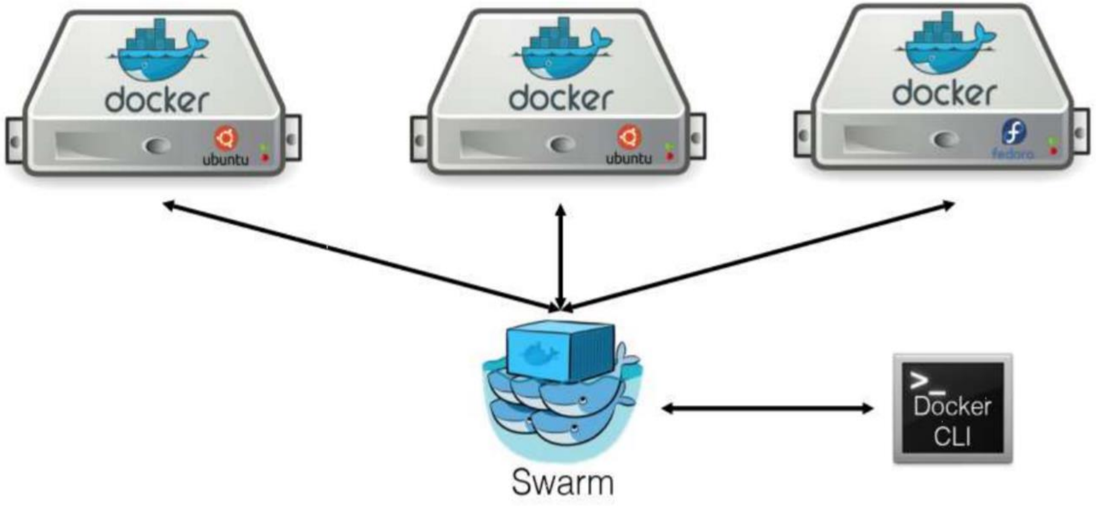
- Microservices
- Docker Compose
- ➡ Docker Swam
    - Docker Machine
    - Create a Swarm
    - Deploy Services to a Swarm
    - Deploy a Stack to a Swarm

Orchestration

- The portability and reproducibility of a containerised process help us
  - **migrate** containerised applications to clouds efficiently;
  - **scale** the containerised applications on the clouds effectively.
- How do we make docker work across multiple nodes?
  - Share containers among each other
  - replace failed containers automatically,
  - manage the rollout of updates and reconfigurations of those containers during their lifecycle.
  - etc.

Orchestration

- **Definition**: automated configuration, management, and coordination of computer systems, applications, and services.
- Tools to manage, scale, and maintain containerised applications are called orchestrators.
  - Examples: Apache Mesos, Kubernetes, Docker Swarm, Amazon ECS (Elastic Container Service).



Docker Swarm

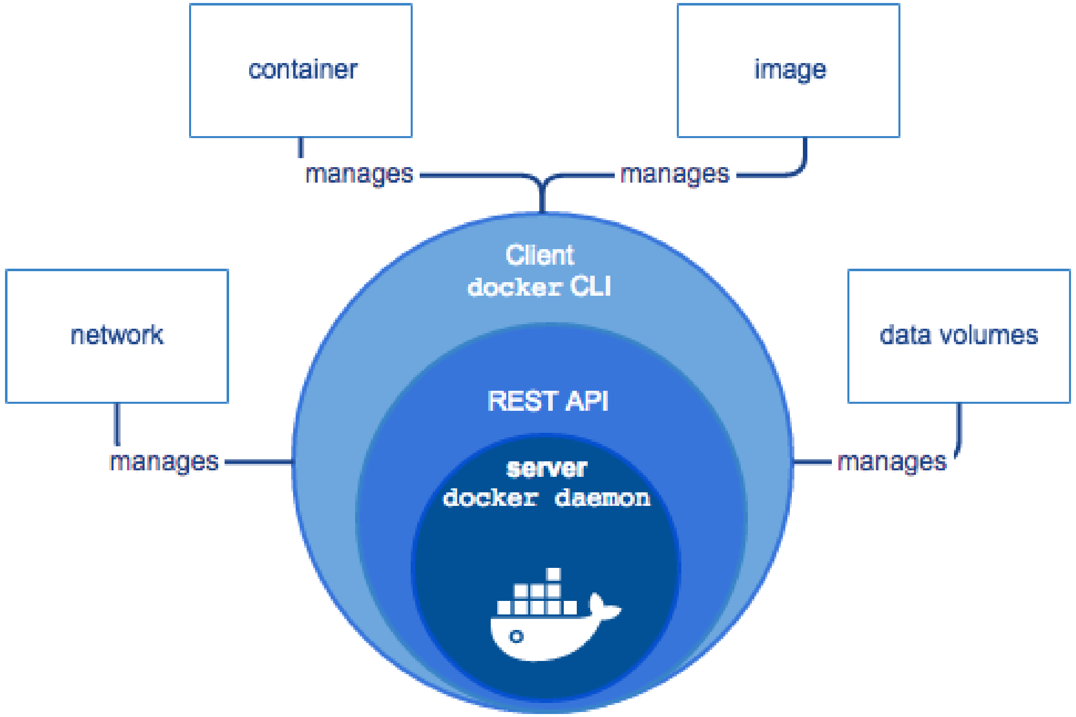
- Docker Swarm manages a cluster of Docker **nodes** and schedule containers

- Each node of a docker swarm is a Docker daemon and all Docker daemons interact using the Docker API

- Docker daemon is responsible for
  - Pulling images, starting containers
  - Managing volumes, networks

- The REST API provides access to the daemon

- The Docker CLI is simply making API requests

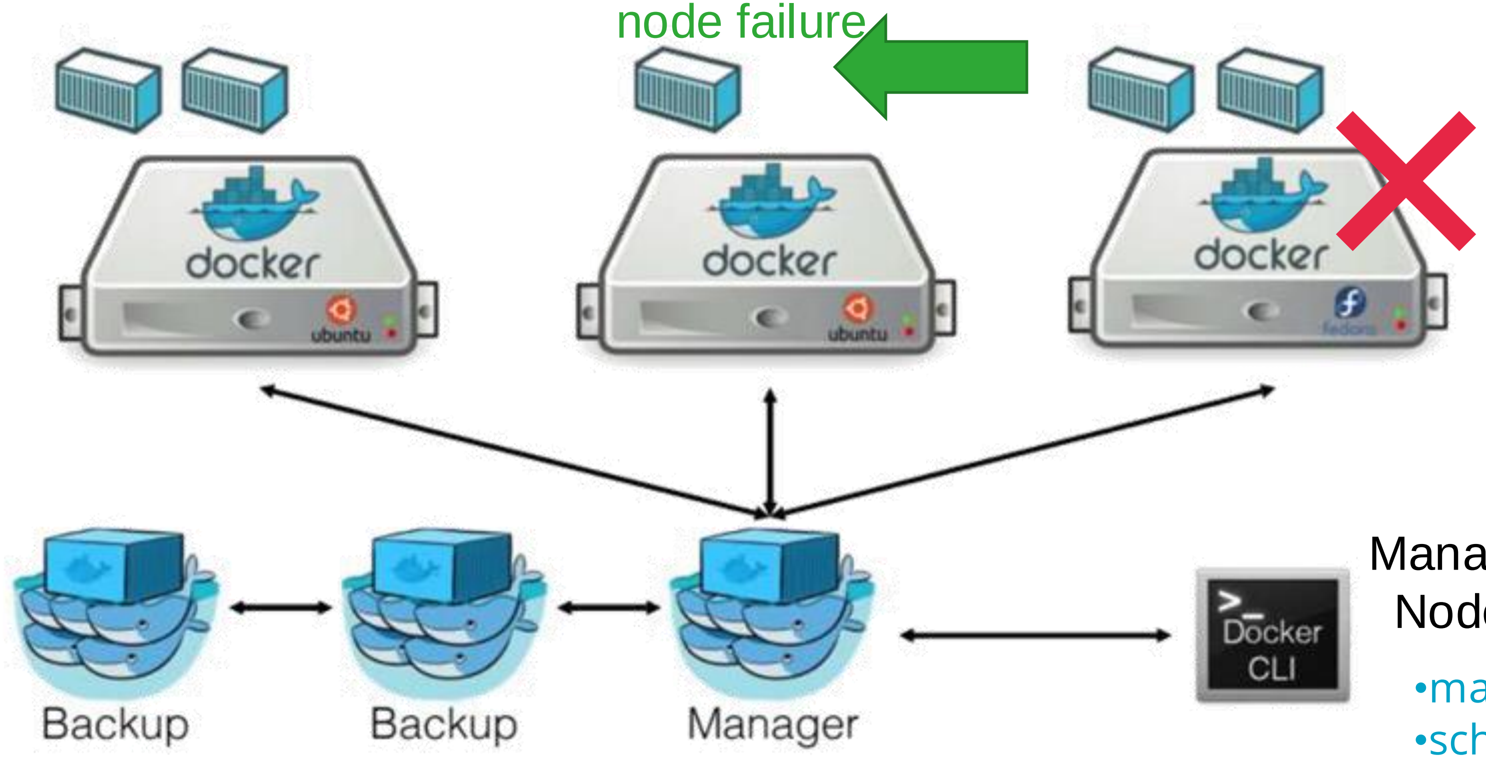


Architecture of Docker Engine

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

Docker Swarm

- Docker Swarm manages a cluster of Docker **nodes** and schedule containers
- Each node of a docker swarm is a Docker daemon and all Docker daemons interact using the Docker API



Reschedule containers on node failure

Failure

Containers

Docker daemons

Worker Nodes

Manager Nodes

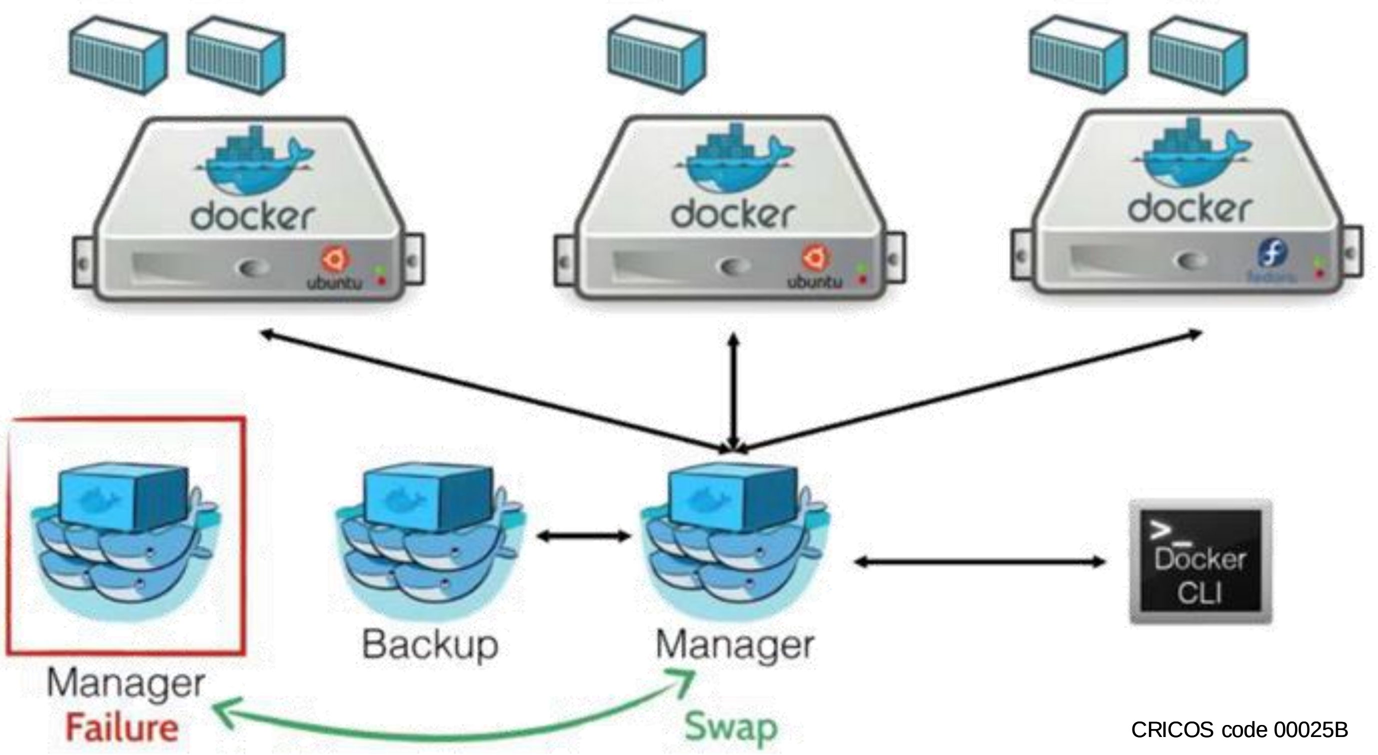
- •maintaining cluster state
- •scheduling services

Docker Swarm

- Docker Swarm manages a cluster of Docker **nodes** and schedule containers
- Each node of a docker swarm is a Docker daemon and all Docker daemons interact using the Docker API

Containers

Docker daemons



Key Features of Docker Swarm

- Cluster management integrated with Docker Engine
- Decentralized Design
- Scaling
- Load Balancing
- Secure by Default
- ✓ Rolling Updates

Getting Started with Swarm Mode

- Initialize a cluster of Docker Engines in swarm mode
- Adding nodes to swarm
- Deploying application services to the swarm
- Managing the swarm once you have everything running

https://docs.docker.com/engine/swarm/swarm-tutorial/

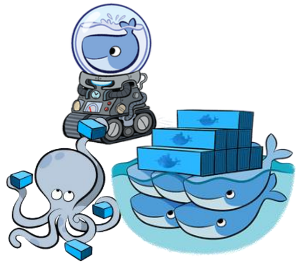
Outline

- Microservices
- Docker Compose
- Docker Swam
  - ➡ Docker Machine
  - Create a Swarm
  - Deploy Services to a Swarm
  - Deploy a Stack to a Swarm

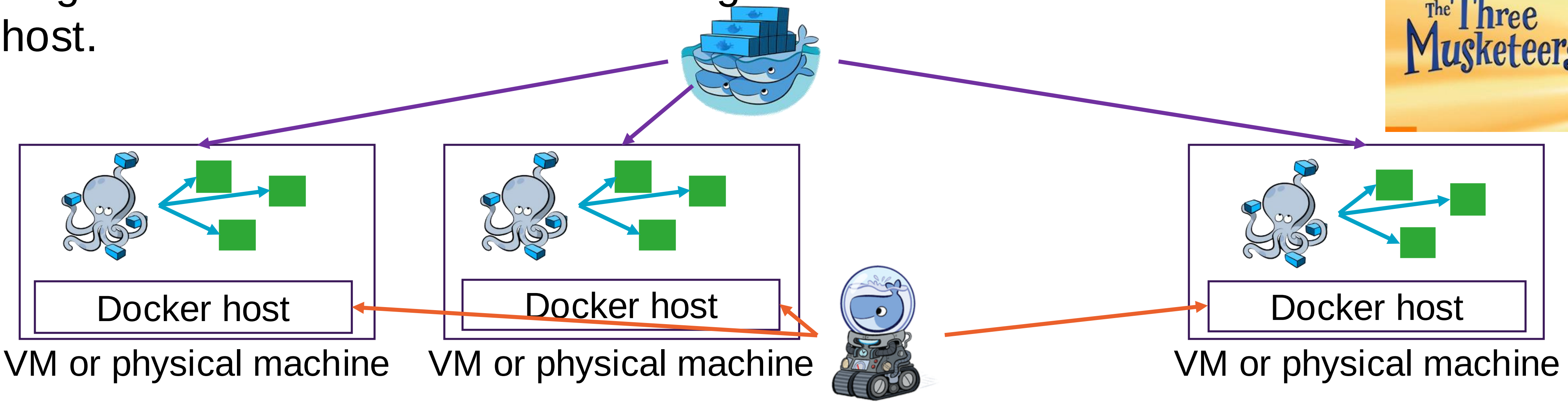
Docker Machine – Docker Hosts Management

Docker Machine allows you to create and manage dockers in a variety of environments,

- virtual machines either on local systems or on cloud providers,
- physical computers.

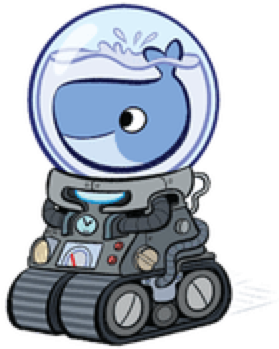


Docker Machine creates a Docker host, and you use the Docker Engine client as needed to build images and create containers on the host.



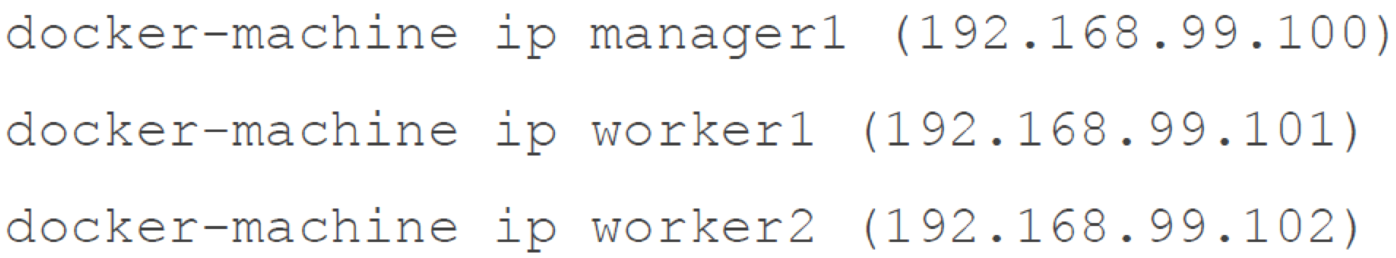
"MICKEY, DONALD AND GOOFY: THE THREE MUSKETEERS" ADDED BACK ONTO DISNEY+

Docker Machine – Docker Hosts Management



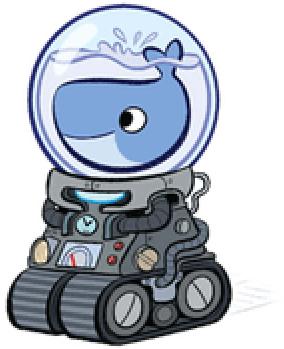
Example: Create three different machines using docker-machine create

1. Manager1 (docker-machine create –driver virtualbox manager1)
2. Worker1 (docker-machine create –driver virtualbox worker1)
3. Worker2 (docker-machine create –driver virtualbox worker2)



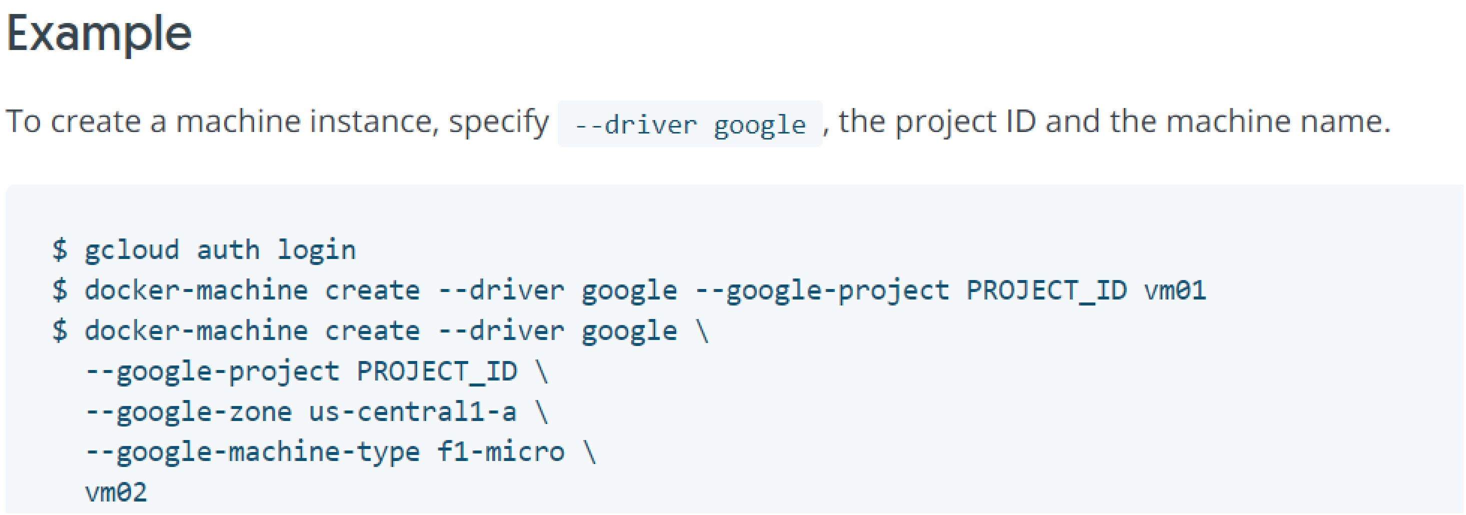
```
docker-machine ip manager1 (192.168.99.100)
docker-machine ip worker1 (192.168.99.101)
docker-machine ip worker2 (192.168.99.102)
```

Docker Machine – Docker Hosts Management



To create a virtual machine, you supply Docker Machine with the name of the driver you want to use.

- For a local Mac or Windows system, the driver is typically Oracle VirtualBox.
- For provisioning physical machines, a generic driver is provided.
- For cloud providers, Docker Machine supports drivers such as AWS, Microsoft Azure, Google Compute Engine, etc.



Example

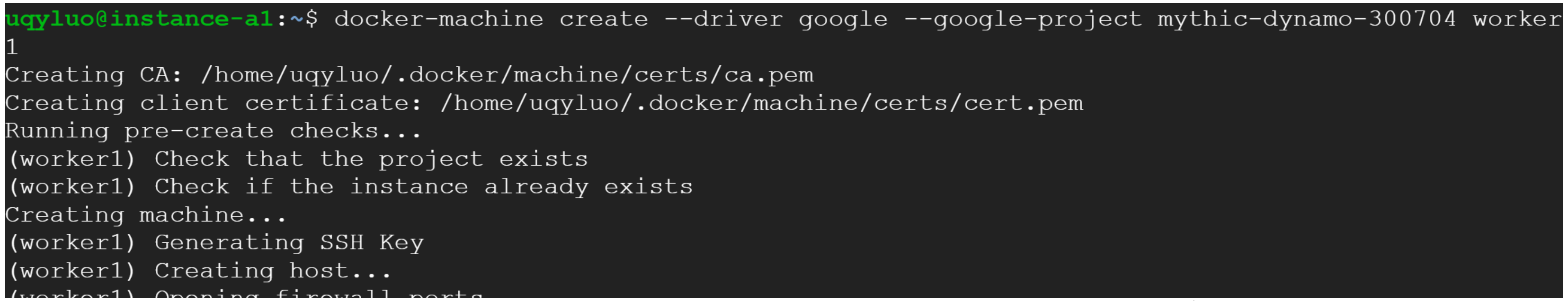
To create a machine instance, specify `--driver google`, the project ID and the machine name.

```
$ gcloud auth login
$ docker-machine create --driver google --google-project PROJECT_ID vm01
$ docker-machine create --driver google \
  --google-project PROJECT_ID \
  --google-zone us-central1-a \
  --google-machine-type f1-micro \
  vm02
```

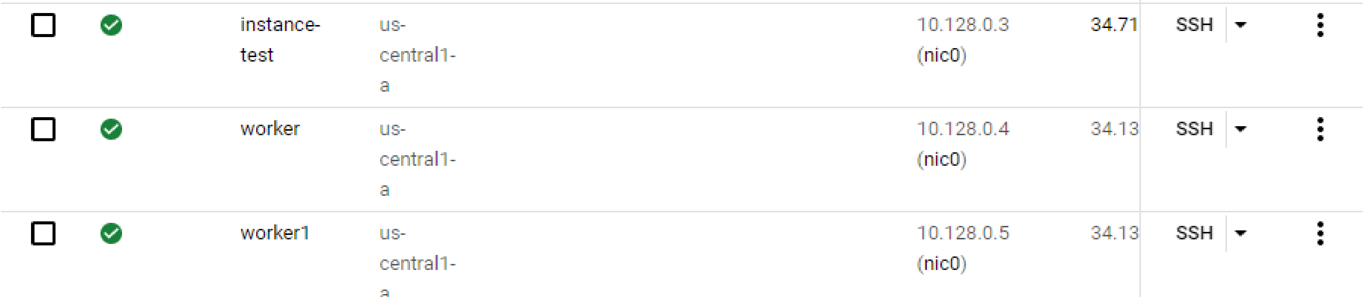
https://docs.docker.com/machine/drivers/gce/

Docker Machine - GCP

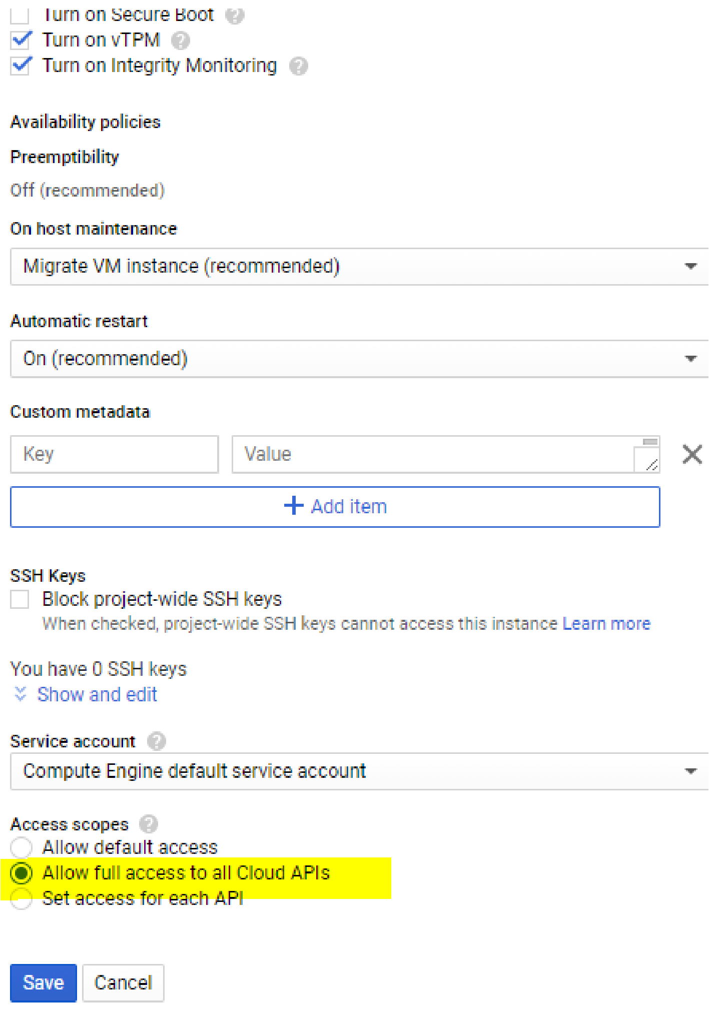
1. Stop the vm instance and edit the vm setting
   - in API access scopes select "Allow full access to all Cloud APIs" and click in save
2. Restart the vm instance, install docker-machine
3. Then run:



```
uqyluo@instance-a1:~$ docker-machine create --driver google --google-project mythic-dynamo-300704 worker
1
Creating CA: /home/uqyluo/.docker/machine/certs/ca.pem
Creating client certificate: /home/uqyluo/.docker/machine/certs/cert.pem
Running pre-create checks...
(worker1) Check that the project exists
(worker1) Check if the instance already exists
Creating machine...
(worker1) Generating SSH Key
(worker1) Creating host...
(worker1) Opening firewall ports
```



□	✓	instance-test	us-central1-a	10.128.0.3 (nic0)	34.71	SSH ▾ ⋮
□	✓	worker	us-central1-a	10.128.0.4 (nic0)	34.13	SSH ▾ ⋮
□	✓	worker1	us-central1-a	10.128.0.5 (nic0)	34.13	SSH ▾ ⋮



□ Turn on Secure Boot
☑ Turn on vTPM
☑ Turn on Integrity Monitoring

Availability policies

Preemptibility

Off (recommended)

On host maintenance

Migrate VM instance (recommended) ▾

Automatic restart

On (recommended) ▾

Custom metadata

Key | Value ✕

+ Add item

SSH Keys

□ Block project-wide SSH keys

When checked, project-wide SSH keys cannot access this instance Learn more

You have 0 SSH keys

Show and edit

Service account

Compute Engine default service account ▾

Access scopes

○ Allow default access
● Allow full access to all Cloud APIs
○ Set access for each API

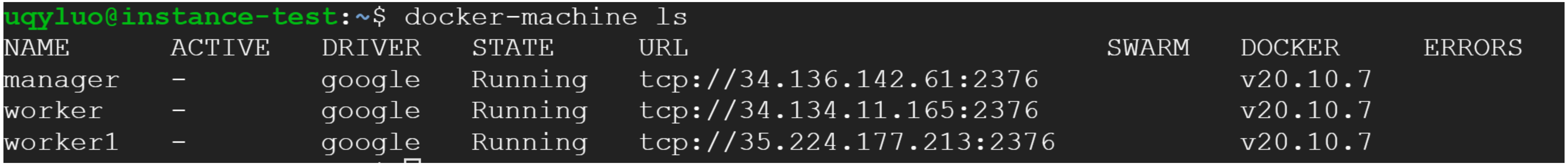
Save | Cancel

https://docs.docker.com/machine/drivers/gce/

Docker Machine – Additional Network Check

The following ports must be available. On some systems, these ports are open by default.

- TCP port 2377 for cluster management communications
- TCP and UDP port 7946 for communication among nodes
- UDP port 4789 for overlay network traffic



```
uqyluo@instance-test:~$ docker-machine ls
NAME       ACTIVE    DRIVER    STATE      URL                           SWARM    DOCKER       ERRORS
manager    -         google    Running    tcp://34.136.142.61:2376               v20.10.7
worker     -         google    Running    tcp://34.134.11.165:2376               v20.10.7
worker1    -         google    Running    tcp://35.224.177.213:2376              v20.10.7
```

https://docs.docker.com/engine/swarm/swarm-tutorial/

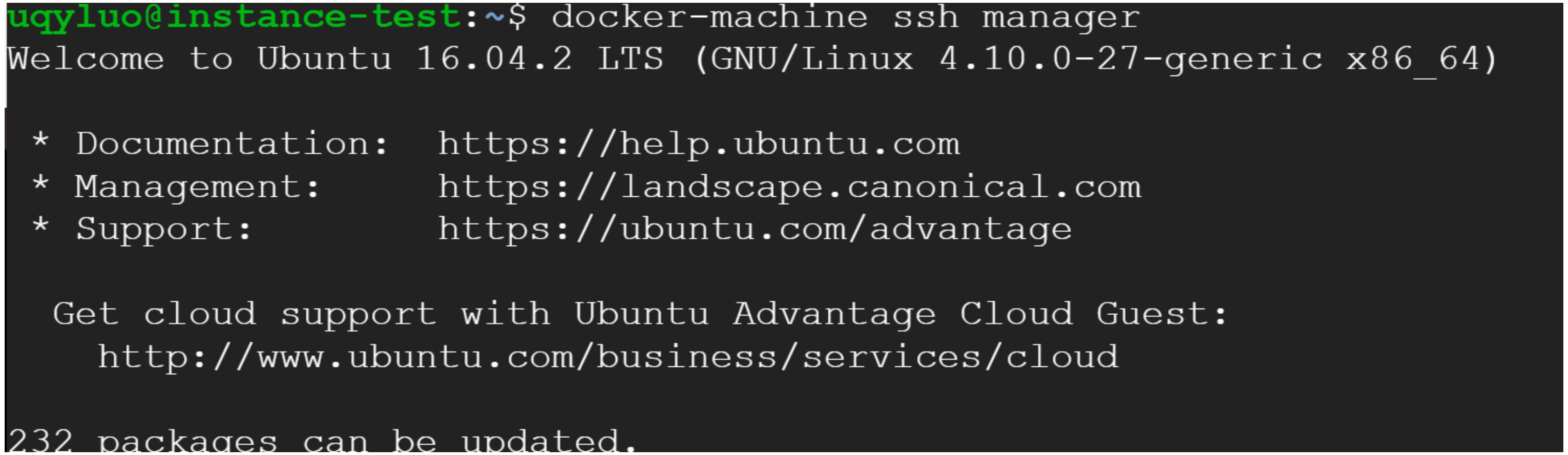
Outline

- Microservices
- Docker Compose
- Docker Swam
  - Docker Machine
  - ➡ Create a Swarm
  - Deploy Services to a Swarm
  - Deploy a Stack to a Swarm

Create a Swarm

Make sure the Docker Engine daemon is started on the host machines.

1. Open a terminal and ssh into the machine where you want to run your manager node. If you use Docker Machine, you can connect to it via SSH using the following command:

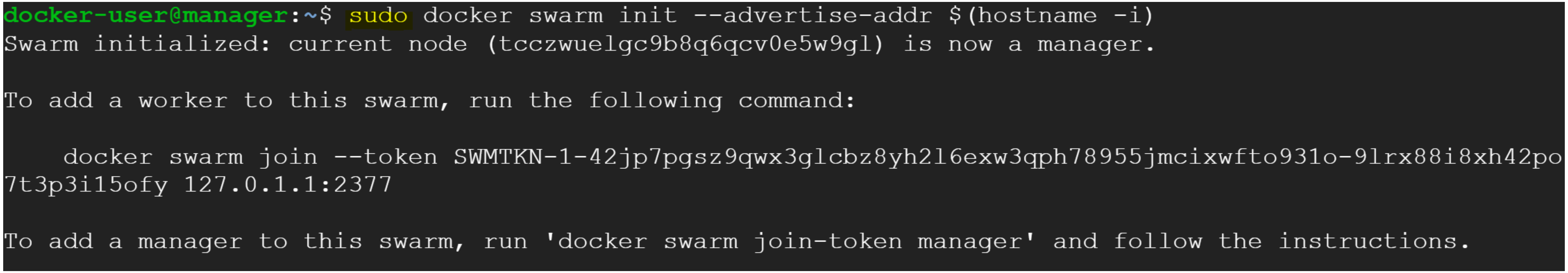


```
uqyluo@instance-test:~$ docker-machine ssh manager
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.10.0-27-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

232 packages can be updated.
```

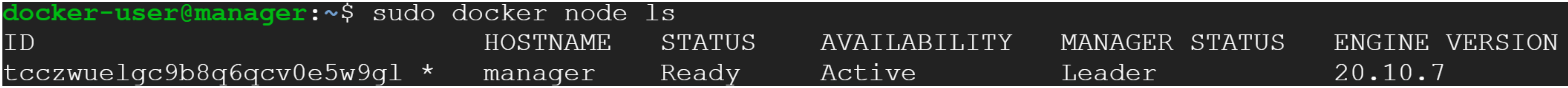
https://docs.docker.com/engine/swarm/swarm-tutorial/

Create a Swarm

2. Run the following command to create a new swarm on the manager node:



```
docker-user@manager:~$ sudo docker swarm init --advertise-addr $(hostname -i)
Swarm initialized: current node (tcczwuelgc9b8q6qcv0e5w9gl) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-42jp7pgsz9qwx3glcbz8yh2l6exw3qph78955jmcixwfto931o-9lrx88i8xh42po
7t3p3i15ofy 127.0.1.1:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

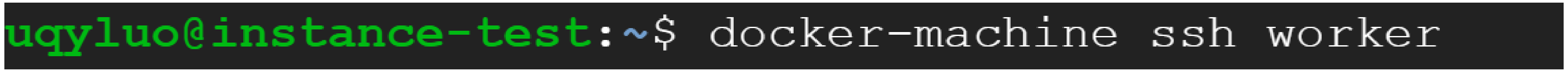


```
docker-user@manager:~$ sudo docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
tcczwuelgc9b8q6qcv0e5w9gl *   manager    Ready     Active         Leader           20.10.7
```

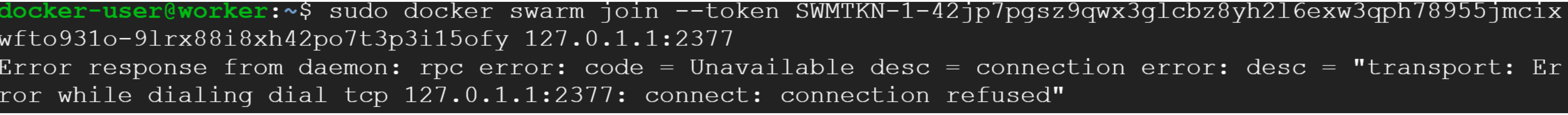
https://docs.docker.com/engine/swarm/swarm-tutorial/

Create a Swarm

3. Jump to a worker node and join to the swarm:

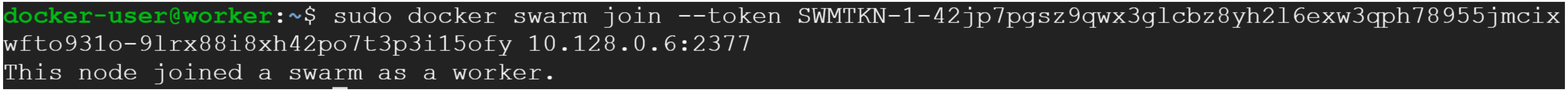


```
uqyluo@instance-test:~$ docker-machine ssh worker
```

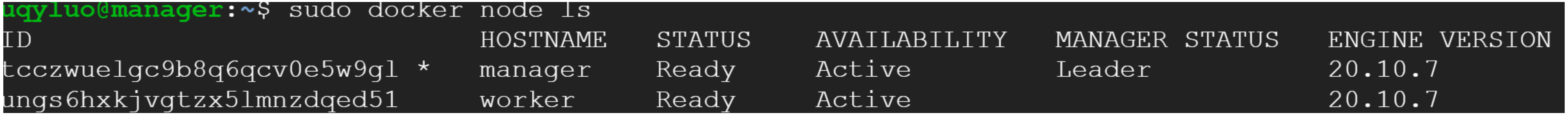


```
docker-user@worker:~$ sudo docker swarm join --token SWMTKN-1-42jp7pgsz9qwx3glcbz8yh2l6exw3qph78955jmcix
wfto931o-9lrx88i8xh42po7t3p3i15ofy 127.0.1.1:2377
Error response from daemon: rpc error: code = Unavailable desc = connection error: desc = "transport: Er
ror while dialing dial tcp 127.0.1.1:2377: connect: connection refused"
```

Change to the internal IP of the manager node



```
docker-user@worker:~$ sudo docker swarm join --token SWMTKN-1-42jp7pgsz9qwx3glcbz8yh2l6exw3qph78955jmcix
wfto931o-9lrx88i8xh42po7t3p3i15ofy 10.128.0.6:2377
This node joined a swarm as a worker.
```



```
uqyluo@manager:~$ sudo docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
tcczwuelgc9b8q6qcv0e5w9gl *   manager    Ready     Active         Leader           20.10.7
ungs6hxkjvgtzx5lmnzdqed51     worker     Ready     Active                          20.10.7
```

https://docs.docker.com/engine/swarm/swarm-tutorial/

Manage Nodes in a Swarm

docker node ls

- Displays view a list of nodes in the swarm

docker node promote/depromote [HOSTNAME]

- Change the role of node e.g., manager to worker

docker node rm [HOSTNAME]

- Delete a node

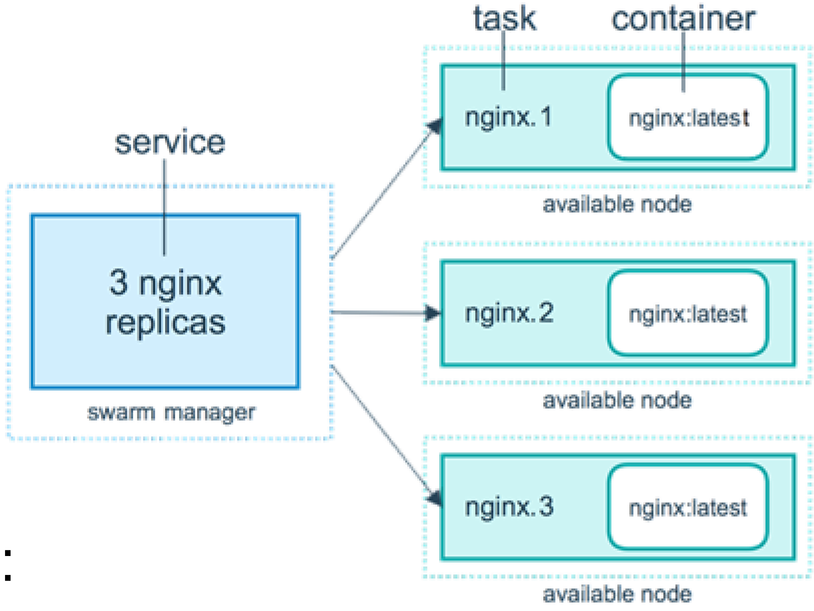
https://docs.docker.com/engine/swarm/manage-nodes/

Outline

- Microservices
- Docker Compose
- Docker Swam
  - Docker Machine
  - Create a Swarm
  - ➡ Deploy Services to a Swarm
  - Deploy a Stack to a Swarm

Deploy Services to a Swarm

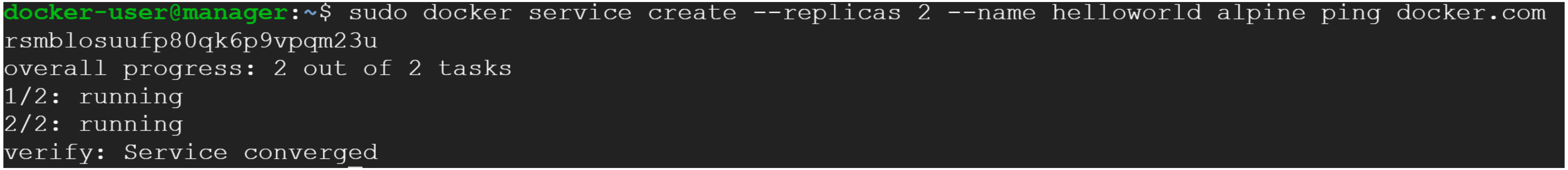
- Service is specified by its desired state:
  - Image and #instances
  - Commands and the options:
    - Ports, overlay network, CPU/Mem limits, etc.
- Task corresponds to a specific container, assigned to a specific node
  - One directional mechanism (progress monotonically):
    - assigned, prepared, running, etc



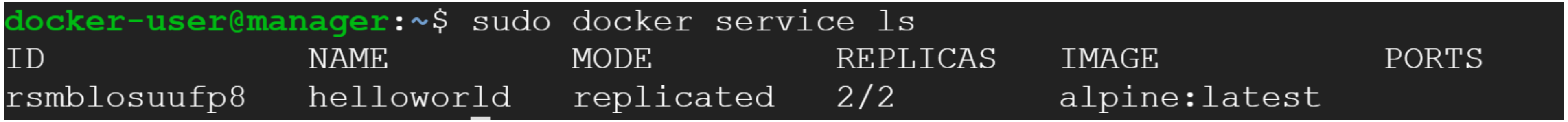
https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/

Deploy Services to a Swarm

- Open a terminal and ssh into the machine where you run your manager node
- specify a command that the service’s containers should run, by adding it after the image name. The below example starts a service called *helloworld* which uses an alpine image and runs the command ping docker.com:



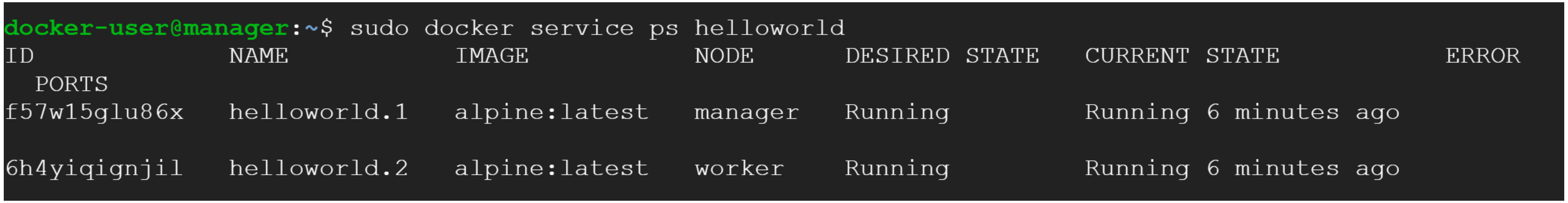
```
docker-user@manager:~$ sudo docker service create --replicas 2 --name helloworld alpine ping docker.com
rsmblosuufp80qk6p9vpqm23u
overall progress: 2 out of 2 tasks
1/2: running
2/2: running
verify: Service converged
```



```
docker-user@manager:~$ sudo docker service ls
ID             NAME         MODE         REPLICAS   IMAGE            PORTS
rsmblosuufp8   helloworld   replicated   2/2        alpine:latest
```

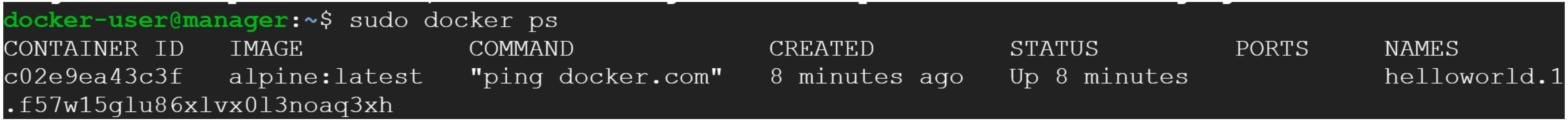
Deploy Services to a Swarm

- Run docker service ps helloworld to see which nodes are running the service:



```
docker-user@manager:~$ sudo docker service ps helloworld
ID              NAME            IMAGE             NODE       DESIRED STATE    CURRENT STATE            ERROR
  PORTS
f57w15glu86x    helloworld.1    alpine:latest     manager    Running          Running 6 minutes ago

6h4yiqignjil    helloworld.2    alpine:latest     worker     Running          Running 6 minutes ago
```

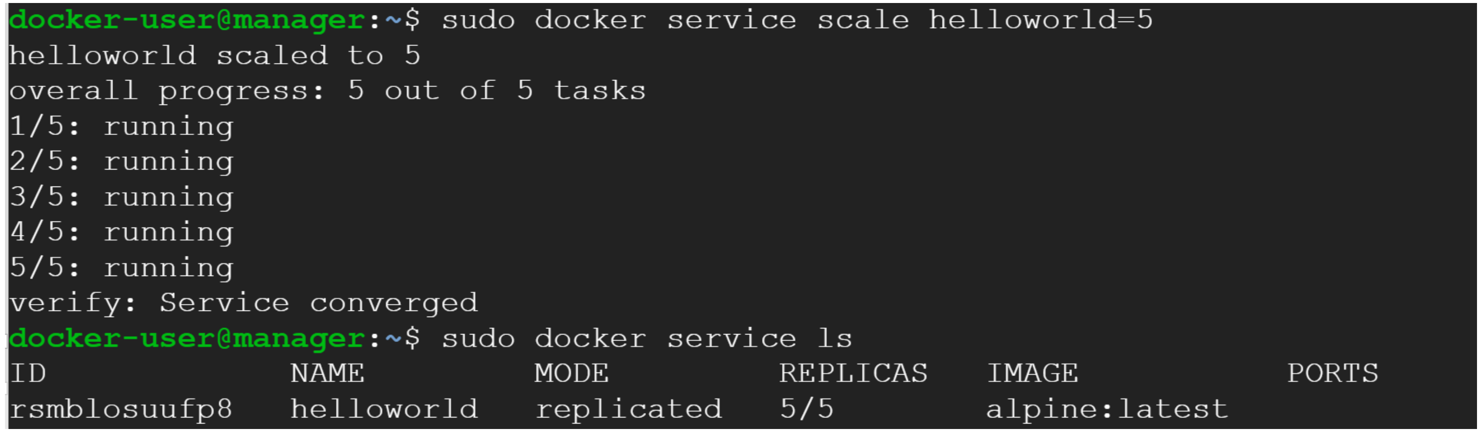
- Run docker ps on the node where the task is running to see details about the containers



```
docker-user@manager:~$ sudo docker ps
CONTAINER ID   IMAGE           COMMAND               CREATED          STATUS          PORTS      NAMES
c02e9ea43c3f   alpine:latest   "ping docker.com"     8 minutes ago    Up 8 minutes               helloworld.1
.f57w15glu86xlvx0l3noaq3xh
```

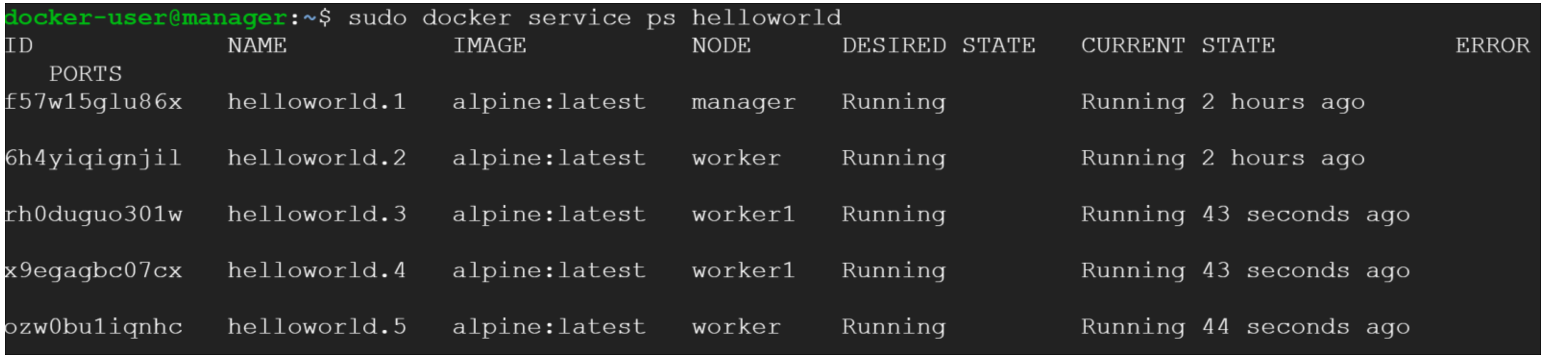
Scale Services in a Swarm

- Run the following command to change the desired scale of the service running in the swarm



```
docker-user@manager:~$ sudo docker service scale helloworld=5
helloworld scaled to 5
overall progress: 5 out of 5 tasks
1/5: running
2/5: running
3/5: running
4/5: running
5/5: running
verify: Service converged
docker-user@manager:~$ sudo docker service ls
ID             NAME         MODE         REPLICAS   IMAGE             PORTS
rsmblosuufp8   helloworld   replicated   5/5        alpine:latest
```

docker service scale SERVICE=REPLICAS [SERVICE=REPLICAS...]

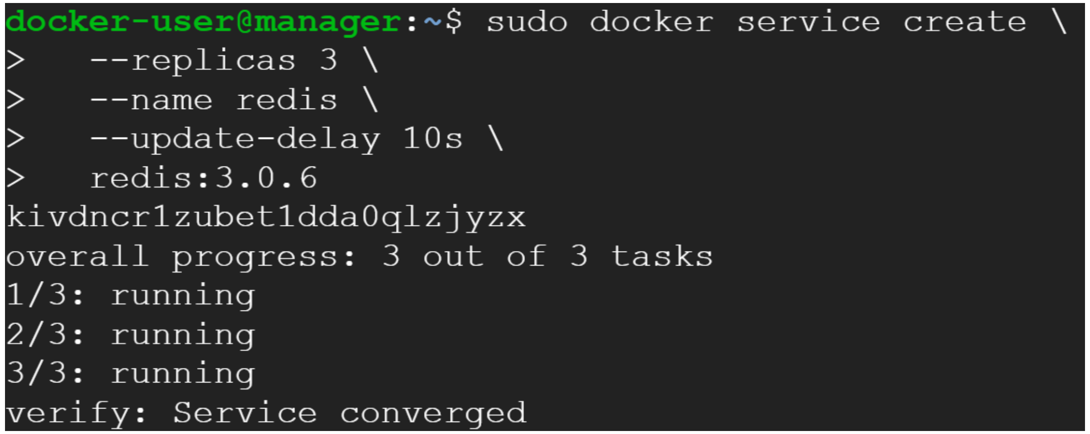


```
docker-user@manager:~$ sudo docker service ps helloworld
ID             NAME           IMAGE           NODE      DESIRED STATE   CURRENT STATE            ERROR
   PORTS
f57w15glu86x   helloworld.1   alpine:latest   manager   Running         Running 2 hours ago

6h4yiqignjil   helloworld.2   alpine:latest   worker    Running         Running 2 hours ago

rh0duguo301w   helloworld.3   alpine:latest   worker1   Running         Running 43 seconds ago

x9egagbc07cx   helloworld.4   alpine:latest   worker1   Running         Running 43 seconds ago

ozw0bu1iqnhc   helloworld.5   alpine:latest   worker    Running         Running 44 seconds ago
```

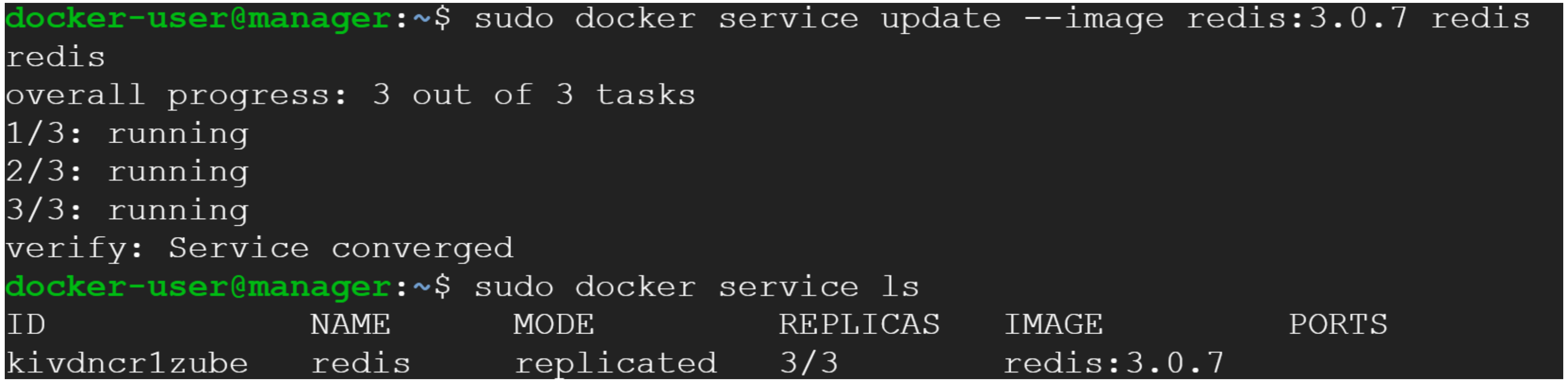
How to scale down the service helloworld?

Rolling Updates

- Run the following command to change the desired scale of the service running in the swarm



```
docker-user@manager:~$ sudo docker service create \
>   --replicas 3 \
>   --name redis \
>   --update-delay 10s \
>   redis:3.0.6
kivdncr1zubet1dda0qlzjyzx
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service converged
```

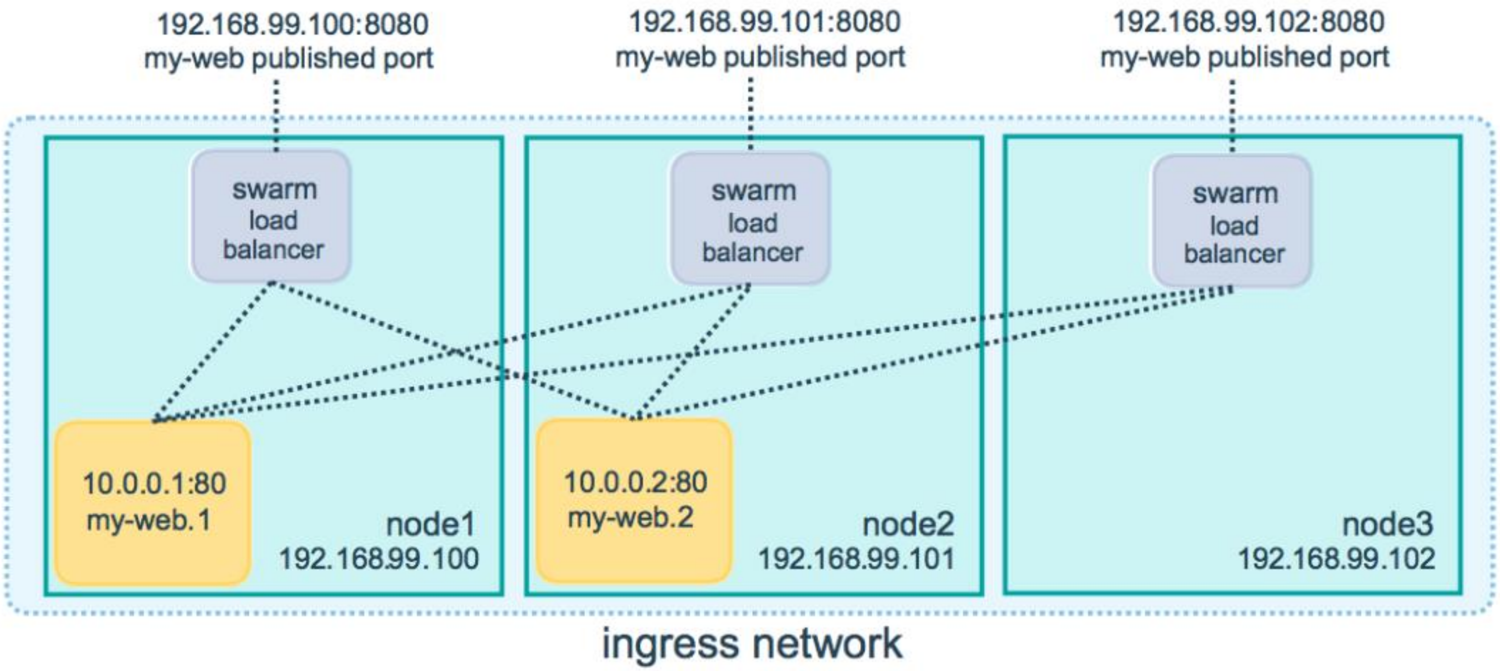


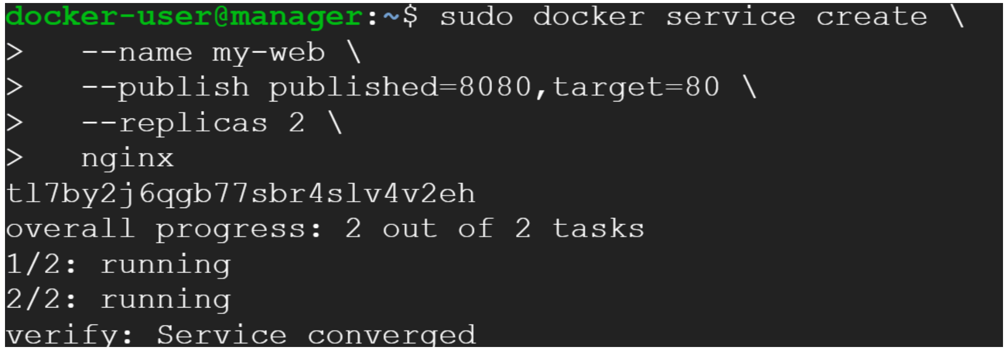
```
docker-user@manager:~$ sudo docker service update --image redis:3.0.7 redis
redis
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service converged
docker-user@manager:~$ sudo docker service ls
ID             NAME      MODE         REPLICAS   IMAGE         PORTS
kivdncr1zube   redis     replicated   3/3        redis:3.0.7
```

Publish Ports

Solution 1: Routing Mesh (-- publish=8080:80 or --publish published=8080, target=80)

- The Swarm makes the service accessible at the target port on every node,
- Ignores whether there is a task for the service running on that node or not.
- Less complex and is the right choice for many types of services.



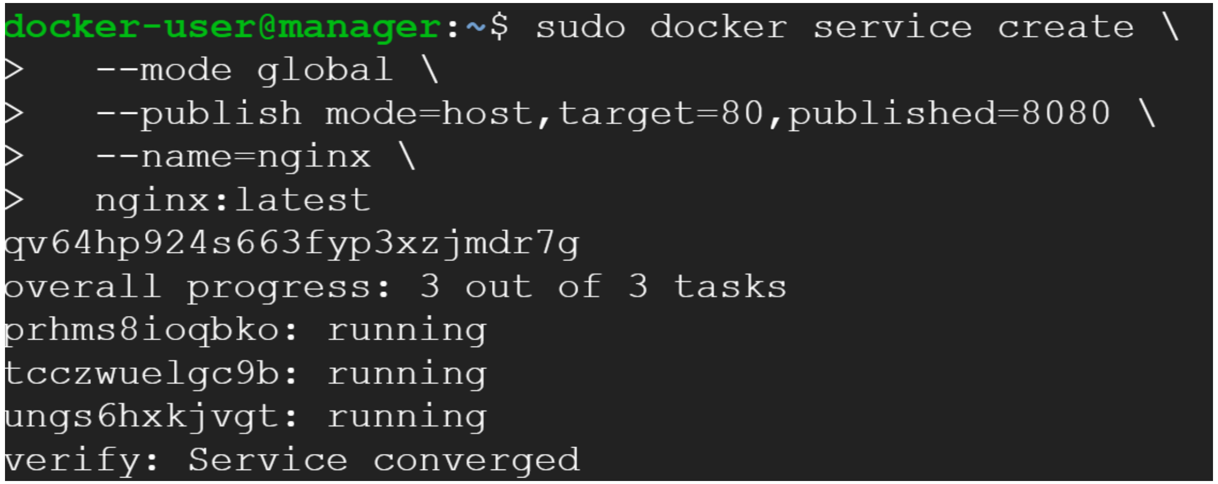


```
docker-user@manager:~$ sudo docker service create \
>   --name my-web \
>   --publish published=8080,target=80 \
>   --replicas 2 \
>   nginx
tl7by2j6qgb77sbr4slv4v2eh
overall progress: 2 out of 2 tasks
1/2: running
2/2: running
verify: Service converged
```

Publish Ports

Solution 2: Port publish on nodes (--publish mode=host, published=8080, target=80)

- Publish a service task’s port directly on the swarm node where that service is running.
- Not using routing mesh, but has maximum flexibility
  - Routing decision is required given the application state
  - More control power – full control of traffic.
- More Responsibility
  - keeping track of where each task is running
  - routing requests to the tasks,
  - load-balancing across the nodes.



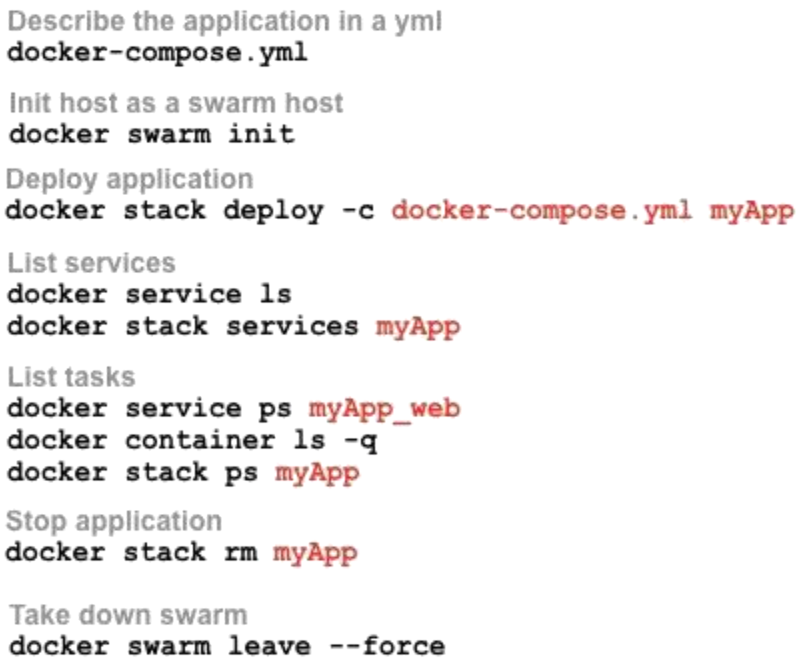
```
docker-user@manager:~$ sudo docker service create \
>    --mode global \
>    --publish mode=host,target=80,published=8080 \
>    --name=nginx \
>    nginx:latest
qv64hp924s663fyp3xzjmdr7g
overall progress: 3 out of 3 tasks
prhms8ioqbko: running
tcczwuelgc9b: running
ungs6hxkjvgt: running
verify: Service converged
```

Outline

- Microservices
- Docker Compose
- Docker Swam
  - Docker Machine
  - Create a Swarm
  - Deploy Services to a Swarm
  - ➡ Deploy a Stack to a Swarm

Deploy a Stack to a Swarm

- When running Docker Engine in swarm mode, you can use docker stack deploy to deploy a complete application stack to the swarm. The deploy command accepts a stack description in the form of a Compose file.



Describe the application in a yml

```
docker-compose.yml
```

Init host as a swarm host

```
docker swarm init
```

Deploy application

```
docker stack deploy -c docker-compose.yml myApp
```

List services

```
docker service ls
docker stack services myApp
```

List tasks

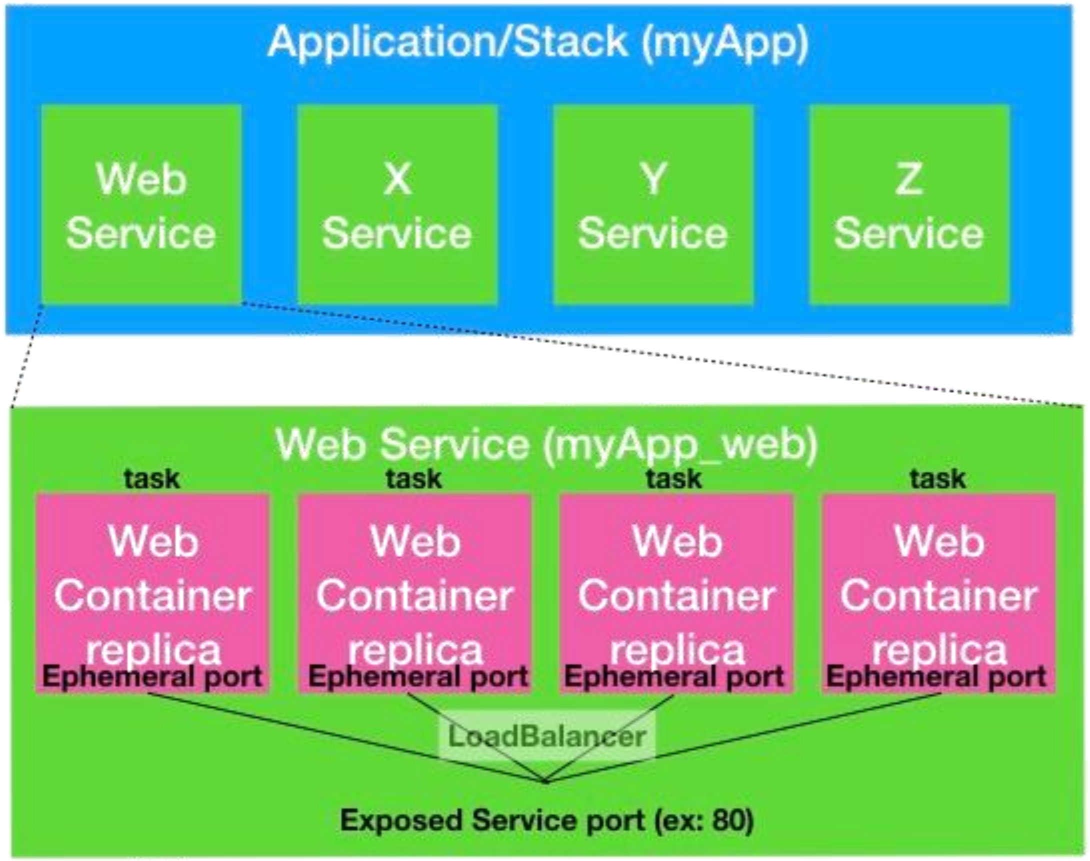
```
docker service ps myApp_web
docker container ls -q
docker stack ps myApp
```

Stop application

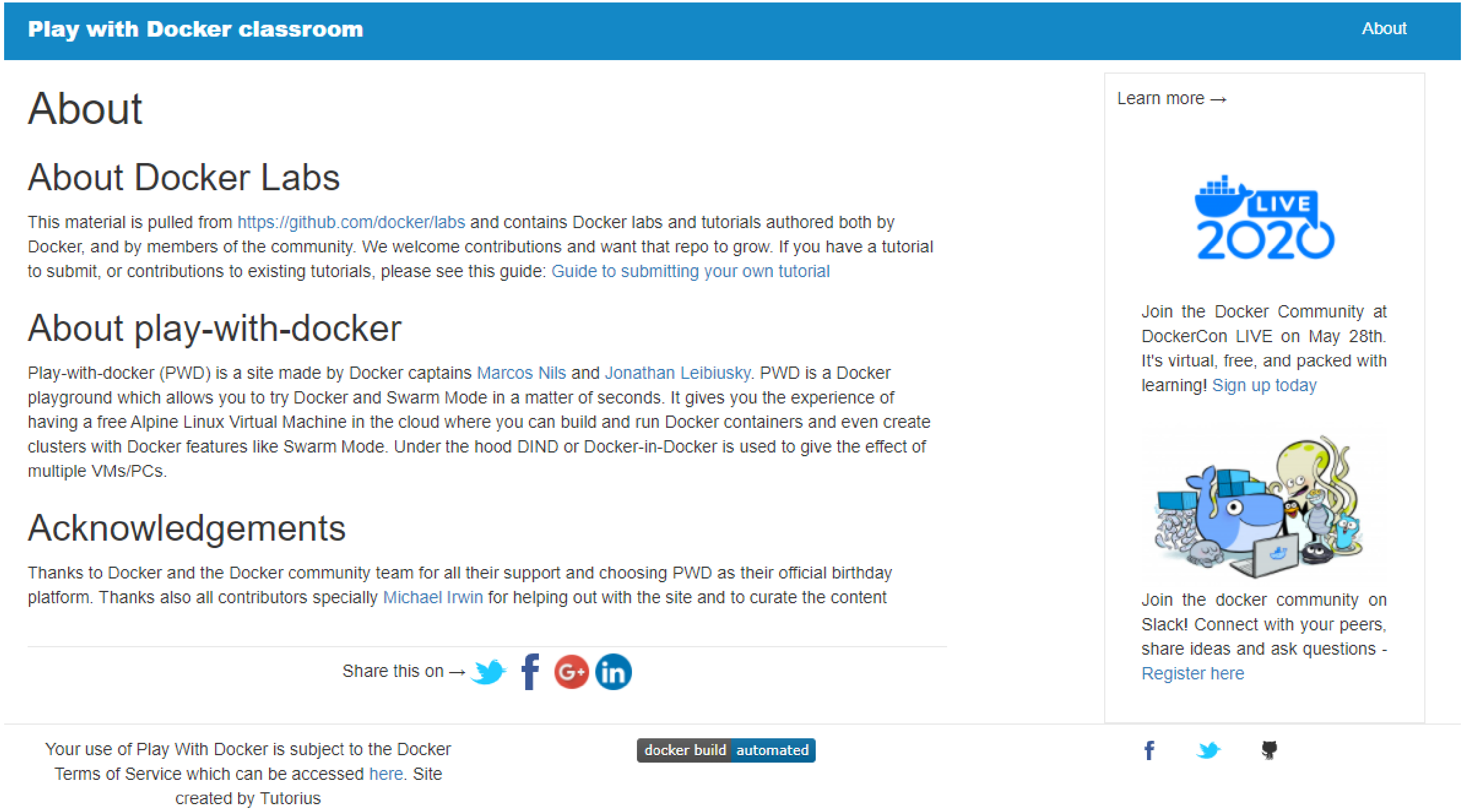
```
docker stack rm myApp
```

Take down swarm

```
docker swarm leave --force
```



Example: Deploy a Stack to a Swarm



About

About Docker Labs

This material is pulled from https://github.com/docker/labs and contains Docker labs and tutorials authored both by Docker, and by members of the community. We welcome contributions and want that repo to grow. If you have a tutorial to submit, or contributions to existing tutorials, please see this guide: Guide to submitting your own tutorial

About play-with-docker

Play-with-docker (PWD) is a site made by Docker captains Marcos Nils and Jonathan Leibiusky. PWD is a Docker playground which allows you to try Docker and Swarm Mode in a matter of seconds. It gives you the experience of having a free Alpine Linux Virtual Machine in the cloud where you can build and run Docker containers and even create clusters with Docker features like Swarm Mode. Under the hood DIND or Docker-in-Docker is used to give the effect of multiple VMs/PCs.

Acknowledgements

Thanks to Docker and the Docker community team for all their support and choosing PWD as their official birthday platform. Thanks also all contributors specially Michael Irwin for helping out with the site and to curate the content

Share this on →

Learn more →

Join the Docker Community at DockerCon LIVE on May 28th. It's virtual, free, and packed with learning! Sign up today

Join the docker community on Slack! Connect with your peers, share ideas and ask questions - Register here

Your use of Play With Docker is subject to the Docker Terms of Service which can be accessed here. Site created by Tutorius

https://training.play-with-docker.com/swarm-stack-intro/

Summary

- Microservices
- Docker Compose
- Docker Swam
  - Docker Machine
  - Create a Swarm
  - Deploy Services to a Swarm
  - Deploy a Stack to a Swarm