Agenda

1. Docker Compose

2. Docker Swarm

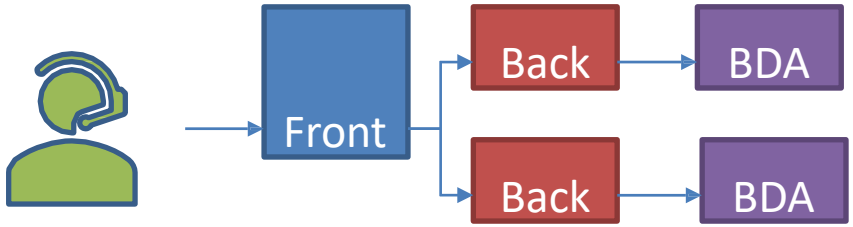
3. Kubernetes

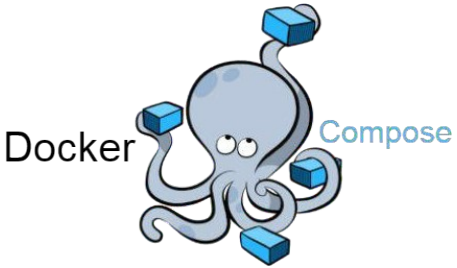
1

Docker Compose

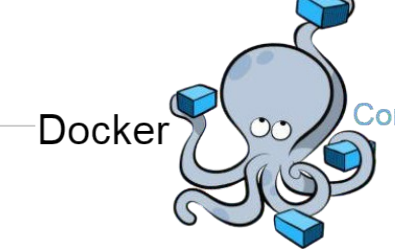
Docker Compose

- Compose is a tool for **defining** and running **complex applications** with docker
- With Docker Compose, you can define a multi-container application in a **single file** and then
  use **single command** to manage all them
- Usually the file is called “docker-compose.yml”





Docker Compose



Microservices

Traditional Application

BBDD

App Server

Content Manager

Web Server

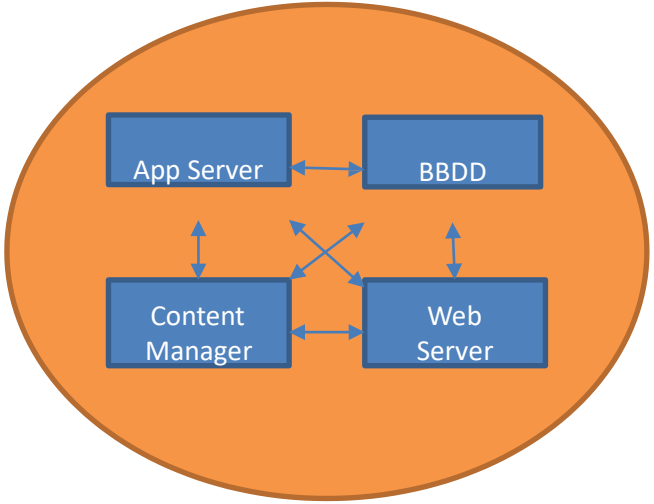
App Server

BBDD

Content Manager

Web Server

Docker Compose



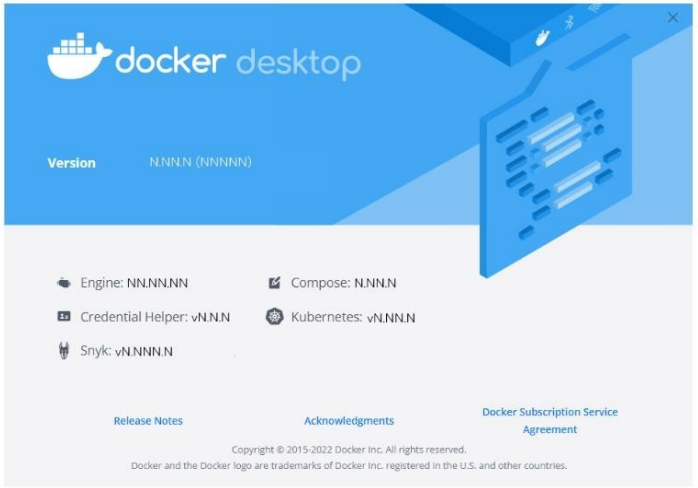
docker-compose.yml

Docker Compose

```yaml
version: '3'
services:
    app:
        build:
          context: ./app
          dockerfile: Dockerfile
        volumes:
          - /datastore/app:/app
        ports:
            - "5000:5000"
            - "9001:9001"
            - "80:80"
        depends_on:
            - influxdb
    influxdb:
        image: influxdb
        volumes:
          - /datastore/influx:/var/lib/influxdb
        ports:
            - "8086:8086"
    grafana:
        build:
          context: ./grafana
          dockerfile: Dockerfile
        volumes:
          - /datastore/grafana:/var/lib/grafana
        ports:
            - "3000:3000"
```

Docker Compose Installation

- **Docker Compose in your laptop**
  - There are two versions of Docker-compose v1 and v2
    - **V1:** you need to run Docker-compose ….
    - **V2**: you need to run Docker compose … (**Docke**



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

Hands-on – Basic Comands

- $ docker-compose **–help**

- $ docker-compose **pull**
- $ docker-compose **up**
- $ docker-compose **run**
- $ docker-compose **start –scale {Service}=3**

- $ docker-compose **ps**
- $ docker-compose **stop**
- $ docker-compose **down**

```
        Name                      Command               State                Ports
-------------------------------------------------------------------------------------------
compose_1_python_1       python3                       Exit 0
compose_1_srv-nginx_1    /docker-entrypoint.sh ngin    Up       0.0.0.0:80->80/tcp,:::80->80/
                         ...                                    tcp
compose_1_srv-tomcat_1   catalina.sh run               Up       0.0.0.0:8080->8080/tcp,:::808
                                                                0->8080/tcp
```

Hands-on Basics

Exercise 1

- Create the following docker-compose file

```
version: '3.8'

services:
  jupyter:
    image: jupyter/scipy-notebook:latest
    container_name: jupyter_container
    ports:
      - "8888:8888"
```

- Execute “Docker-compose pull”
- Execute “Docker-compose up”
- Execute “Docker ps”
- Execute “Docker-compose ps”
- Execute “Docker-compose stop”
- Add Python Service (python image) and retry

- Execute “docker-compose start”
- Execute “docker-compose start –scale python=3” (BEWARE OF THE CUSTOM NAME)
- Execute “docker-compose down”

Hands-on Monitoring

- $ docker-compose images
- $ docker-compose logs
- $ docker-compose top
- $ docker compose events

Hands-on Monitoring

- **Exercise 2**
  - Create a docker compose file
    - Use this file:
    - https://docs.docker.com/compose/wordpress/
  - $ docker compose up –d
  - Try to understand the file
  - Get logs and running processes of running containers

Hands-on Building

- Docker Compose can also be used to build your docker images

```yaml
version: '3.8'

services:
  jupyter:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: jupyter_custom_container
    ports:
      - "8888:8888"
    volumes:
      - ./notebooks:/home/jovyan/work
    environment:
      JUPYTER_TOKEN: "my_secret_token"
    restart: unless-stopped
```

```
# Utilizamos como base la imagen oficial de Jupyter con Scipy
FROM jupyter/scipy-notebook:latest

# Opcionalmente, instalamos paquetes adicionales usando pip o conda
RUN pip install pandas seaborn

# Exponemos el puerto 8888 para Jupyter
EXPOSE 8888

# El comando de inicio es el mismo que ya viene configurado en la imagen base
CMD ["start-notebook.sh"]
```

- “**docker compose build**” builds image if the tag **build** is set

Hands-on Building

- **Exercise 3**
  - Use one Dockerfile of a previous exercise to create your own Docker image with compose
  - Execute "docker-compose up”
  - Execute “docker docker-compose stop”
  - Execute “docker docker-compose ps –a”
  - Execute “docker docker-compose down”
  - Execute “docker docker-compose ps –a”

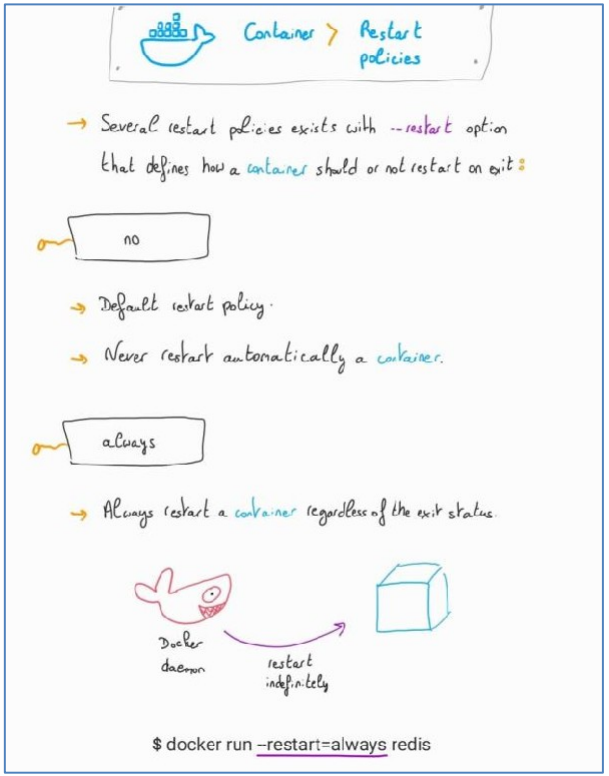
Environment Variables

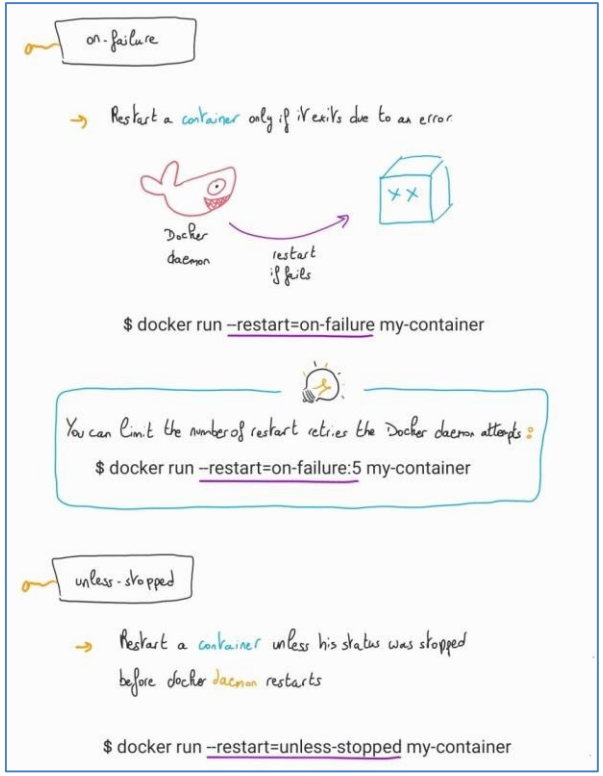
- Docker Compose can also manage env variables.

```yaml
services:
  jupyter:
    image: jupyter/scipy-notebook:latest
    container_name: jupyter_container
    ports:
      - "8888:8888"
    volumes:
      - ./notebooks:/home/jovyan/work
    environment:
      JUPYTER_TOKEN: "my_secret_token"
    restart: unless-stopped
```

- Properties files can be used to externalize configuration

Restart Policies





Hands-on Volumes

- As expected, Docker Compose can manage the same Volume types as Docker
  - Anonymous Volumes
  - Named Volumes
  - Bind Mounts

```
    volumes:
      - ./notebooks:/home/jovyan/work
    environment:
      JUPYTER_TOKEN: "my_secret_token"
    restart: unless-stopped

  python_service:
    image: python:3.9-slim

    ports:
      - "5000:5000"
    volumes:
      - ./python_app:/app
    restart: unless-stopped
```

- **Docker compose down** doesn't remove volumes, use **docker volume prune** instead

Docker Compose Resources

- Docker compose allows to define the CPU and memory a container can use
- The way to define them, depends a lot on the compose version
- **Limits**: maximum amount of memory and CPU a container can use
- **reservations**: minimum amount of Memory and CPU is reserved for a container

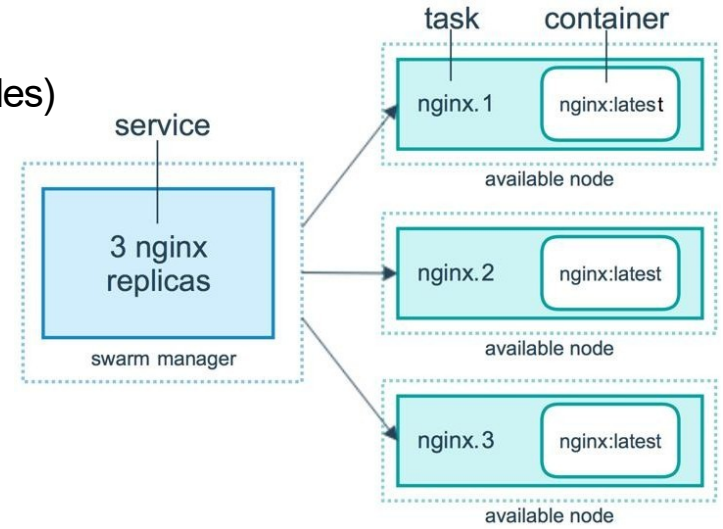
```
version: '3.2'  services:
  db:
    image: mysql  deploy:
      resources:  limits:
        cpus: '0.001'
          memory: 50M
        reservations:
          cpus: '0.0001'
          memory: 20M
```

2

Docker Swarm

Docker Swarm

- Large and small software companies deploying thousands of container instances daily
  - How can we manage this complexity?
- Container orchestrator
- Cluster Manager (with master and workers nodes)
- Works with the concept of Services and Tasks



Docker Swarm

Describe the application in a yml
```
docker-compose.yml
```

Init host as a swarm host
```
docker swarm init
```

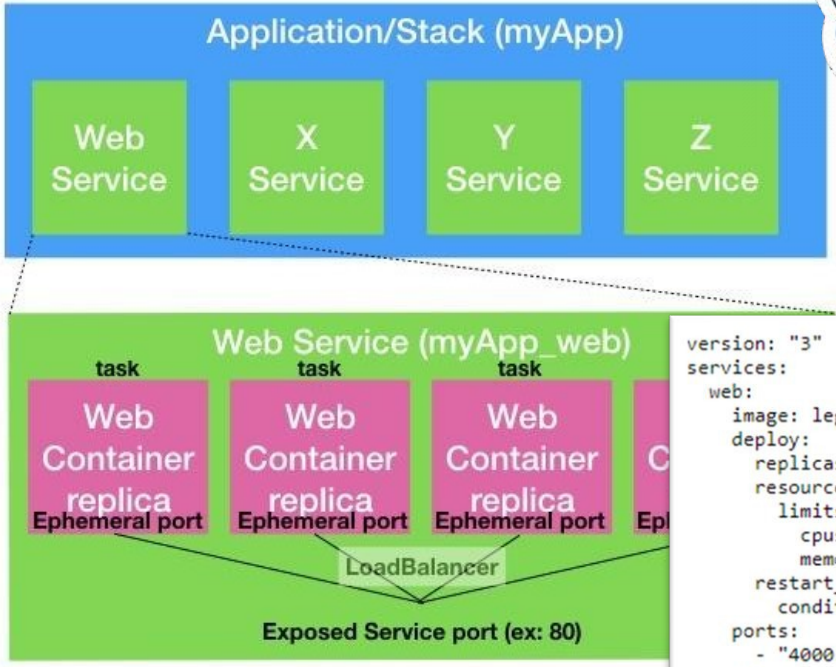
Deploy application
```
docker stack deploy -c docker-compose.yml myApp
```

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
```

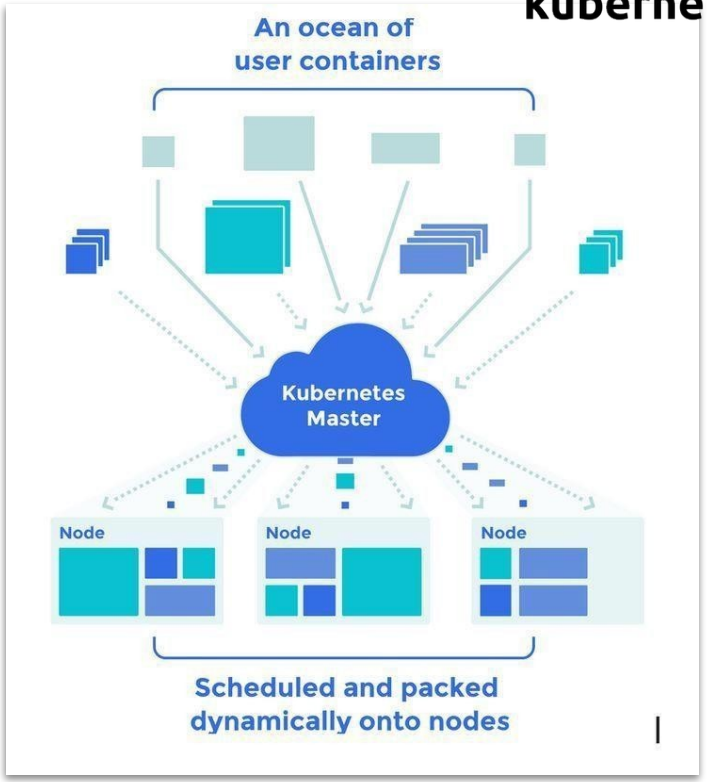



```
version: "3"
services:
  web:
    image: legabz/hellokube:swagger
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 1G
      restart_policy:
        condition: on-failure
    ports:
      - "4000:8080"
    networks:
      - webnet
networks:
  webnet:
```

3 Kubernetes

Kubernetes

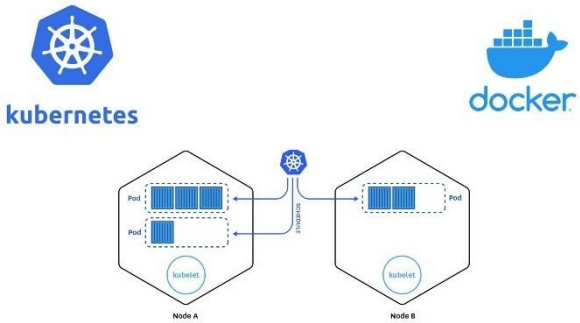
- Large and small software companies deploying thousands of container instances daily
  - How can we manage this complexity?
- Originally developed by Google.
- Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications
- Kubernetes makes it easy to deploy and operate applications in a microsevice architecture



Kubernetes

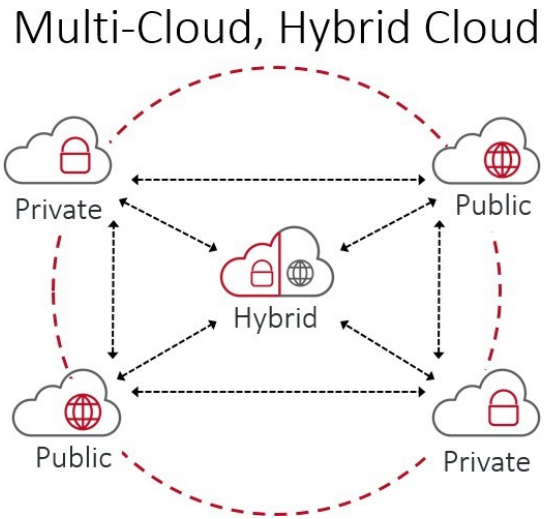
Features:

- Controlling resource consumption by application or team
- Evenly spreading application load across a host infrastructure
- Automatically load balancing requests across the different instances of an application
- Monitoring resource consumption and resource limits
- Moving an application instance from one host to another
- Automatically leveraging additional resources made available when a new host is added
- Work with the concepts of Service and Pod



Kubernetes Strengths

- Hybridization
- Multi-cloud



Google Cloud Platform

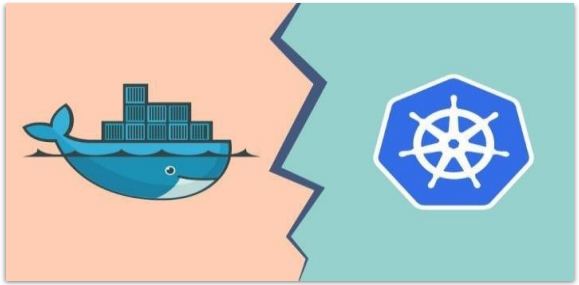
Microsoft Azure

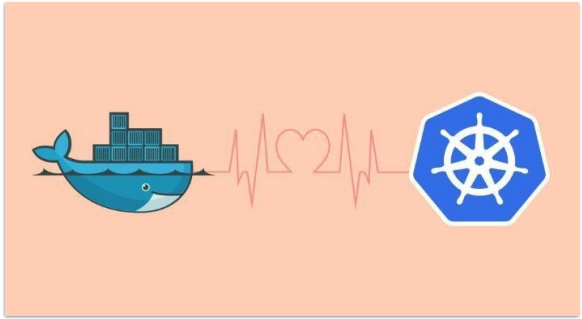
aws

4 Tech Summary

Tech Summary

- **Docker** is (in many cases) the core technology used for containers and can deploy single, containerized applications
- **Docker Compose** is used for configuring and starting multiple Docker containers **on the same host–so**
- **Docker swarm** is a container orchestration tool that allows you to run and connect containers on multiple hosts
- **Kubernetes** is a container orchestration tool that is similar to Docker swarm, but has a wider appeal due to its ease of automation and ability to handle higher demand





4

SQL - Docker

POSTGRES

- https://raw.githubusercontent.com/jupyter/docker-stacks/refs/heads/main/images/base-notebook/Dockerfile

POSTGRES

```
services:
  postgres:
    container_name: postgres_container
    image: postgres:12.1
    environment:
      POSTGRES_USER: ${POSTGRES_USER:-postgres}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:-Welcome01}
      PGDATA: /data/postgres
    volumes:
       - postgres:/data/postgres
    ports:
      - "5432:5432"
    networks:
      - postgres
    restart: unless-stopped
```

POSTGRES

```
pgadmin:
  container_name: pgadmin_container
  image: dpage/pgadmin4:4.16
  environment:
    PGADMIN_DEFAULT_EMAIL: ${PGADMIN_DEFAULT_EMAIL:-pgadmin4@pgadmin.org}
    PGADMIN_DEFAULT_PASSWORD: ${PGADMIN_DEFAULT_PASSWORD:-admin}
  volumes:
     - pgadmin:/root/.pgadmin
  ports:
    - "${PGADMIN_PORT:-5050}:80"
  networks:
    - postgres
  restart: unless-stopped
```

THANKS!

JOSE LUIS GOMEZ ORTEGA

- PhD Computer Science
- Msc Engineering with Management
- Diplomatura Ingenieria Industrial

Currently:
Data Science Lead instructor @ TheBridge

https://www.linkedin.com/in/jlgortega/
jose.lgo.datascience@gmail.com