EDEM

Máster en Data Analytics

Containers Management

Alberto Hernández Cantos

Agenda

1. Docker Compose

2. Docker Swarm

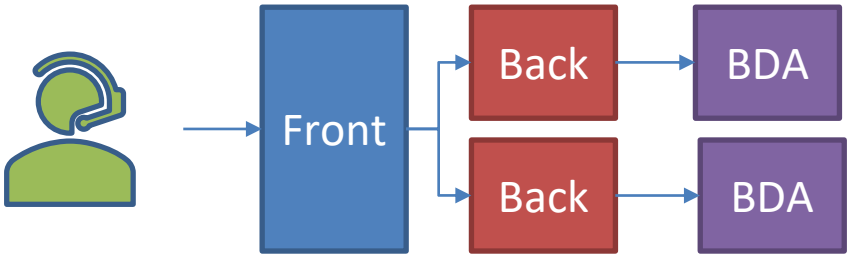
3. Kubernetes

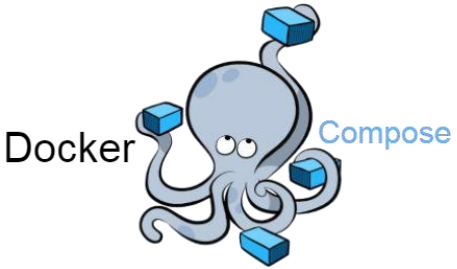
4. Tech. Summary

1 Docker Compose

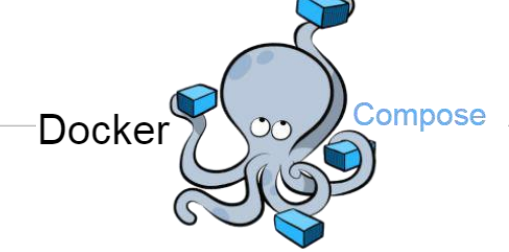
Docker Compose

- 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”

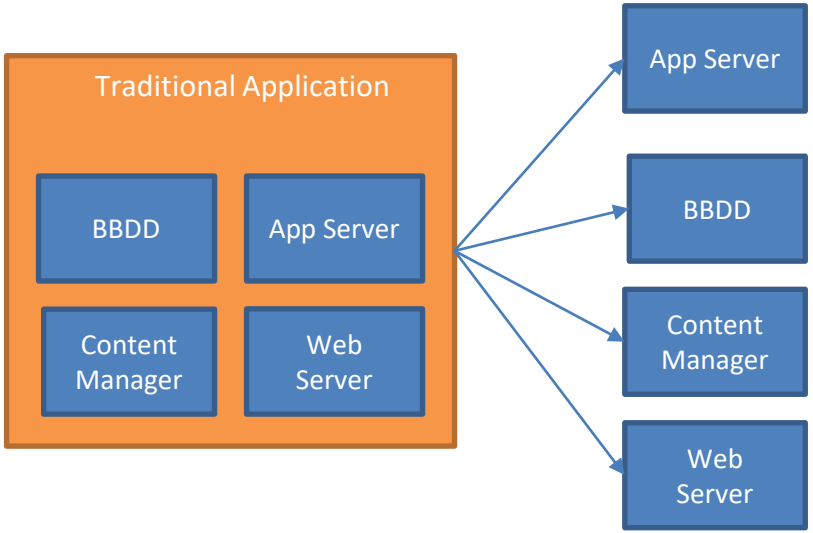




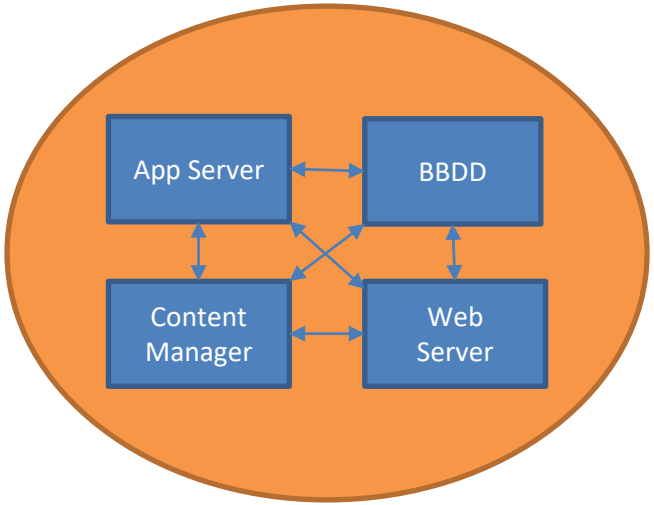
Docker Compose



Microservices



Docker Compose



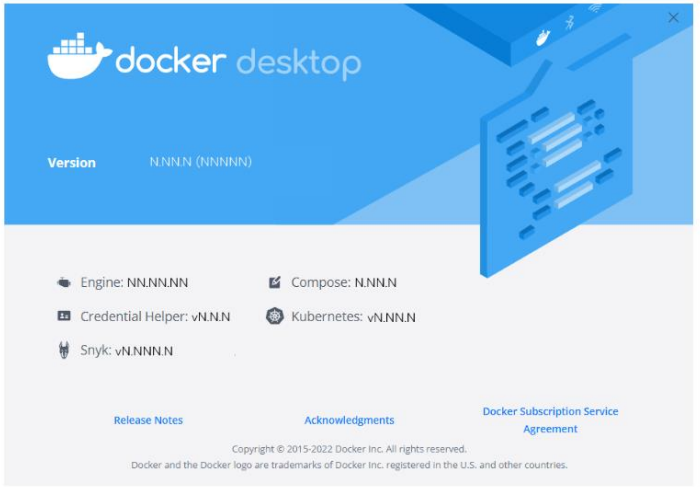
docker-compose.yml

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 … (**Docker Desktop**)



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'
services:
  srv-nginx:
    image: nginx
    ports:
      - "80:80"
```

  - 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”
  - Execute “docker-compose down”

Hands-on Monitoring

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

```
    Container          Repository    Tag       Image Id       Size
-----------------------------------------------------------------------
compose_1_nginx_1     nginx         latest    670dcc86b69d   141.5 MB
compose_1_python_1    python        latest    3a49f9c9c80e   919.7 MB
compose_1_tomcat_1    tomcat        latest    e303233ea761   482.9 MB
root@ESPC012268:/mnt/c/Users/aohz/git/edem/compose_1#
```

```
nginx_1   | 2022/07/28 14:19:08 [notice] 1#1: signal 29 (SIGIO) received
nginx_1   | 2022/07/28 14:19:08 [notice] 1#1: signal 17 (SIGCHLD) received from 25
nginx_1   | 2022/07/28 14:19:08 [notice] 1#1: worker process 25 exited with code 0
nginx_1   | 2022/07/28 14:19:08 [notice] 1#1: worker process 27 exited with code 0
nginx_1   | 2022/07/28 14:19:08 [notice] 1#1: exit
tomcat_1  | 28-Jul-2022 14:13:38.474 INFO [main] org.apache.coyote.AbstractProtocol.init In
 ["http-nio-8080"]
tomcat_1  | 28-Jul-2022 14:13:39.215 INFO [main] org.apache.catalina.startup.Catalina.load
9995] milliseconds
tomcat_1  | 28-Jul-2022 14:13:40.366 INFO [main] org.apache.catalina.core.StandardService.s
ice [Catalina]
tomcat_1  | 28-Jul-2022 14:13:40.379 INFO [main] org.apache.catalina.core.StandardEngine.st
et engine: [Apache Tomcat/10.0.23]
tomcat_1  | 28-Jul-2022 14:13:40.686 INFO [main] org.apache.coyote.AbstractProtocol.start S
```

```
compose_1_srv-nginx_1
  UID        PID    PPID   C  STIME   TTY      TIME                    CMD
-------------------------------------------------------------------------------------
root       3950   3895   1  16:35   ?     00:00:00   nginx: master process nginx -g daemon off;
systemd+   4177   3950   0  16:35   ?     00:00:00   nginx: worker process
systemd+   4180   3950   0  16:35   ?     00:00:00   nginx: worker process
systemd+   4181   3950   0  16:35   ?     00:00:00   nginx: worker process
systemd+   4182   3950   0  16:35   ?     00:00:00   nginx: worker process
systemd+   4183   3950   0  16:35   ?     00:00:00   nginx: worker process
systemd+   4184   3950   0  16:35   ?     00:00:00   nginx: worker process
systemd+   4185   3950   0  16:35   ?     00:00:00   nginx: worker process
systemd+   4186   3950   0  16:35   ?     00:00:00   nginx: worker process

compose_1_srv-tomcat_1
UID    PID    PPID   C    STIME   TTY      TIME                    CMD
-------------------------------------------------------------------------------------
root   3957   3912   84   16:35   ?     00:00:05   /opt/java/openjdk/bin/java -Djava.util.logging.config.file=/usr/lo
                                                   cal/tomcat/conf/logging.properties
```

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

```
version: '3.0'
services:
  srv-web:
    build: .
    image: mynginx
    container_name: web_frontal
    ports:
      - "80:80"
```

- “**docker compose build**” builds the image if the tag **build** is set
- “**docker compose up**” builds the image if it is not available in your docker registry

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.

```
services:
  db:
    image: mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=xx
      - MYSQL_DATABASE=xx
      - MYSQL_USER=xx
      - MYSQL_PASSWORD=xx
```

```
services:
  db:
    image: mysql
    restart: always
    env_file:
      - mysql.properties
```

- Properties files can be used to externalize configuration

Restart Policies

Container > Restart policies

→ Several restart policies exists with --restart option that defines how a container should or not restart on exit:

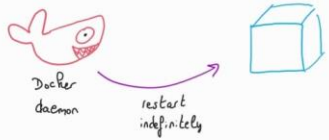
no

→ Default restart policy.

→ Never restart automatically a container.

always

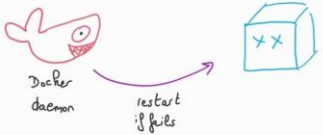
→ Always restart a container regardless of the exit status.



$ docker run --restart=always redis

on-failure

→ Restart a container only if it exits due to an error



$ docker run --restart=on-failure my-container

You can limit the number of restart retries the Docker daemon attempts:

$ docker run --restart=on-failure:5 my-container

unless-stopped

→ Restart a container unless his status was stopped before docker daemon restarts

$ docker run --restart=unless-stopped my-container

Hands-on Volumes

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

```yaml
services:
  db:
    image: xxx
    volumes:
      - /var/lib/mysql          # Anonymous Volume
      - db:/var/lib/mysql       # Named Volume
      - /tmp/db:/var/lib/mysql # Bind Mount

volumes:                  # Named Volumes Only
  db:
```

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

Hands-on Volumes

- **Exercise 4**
  - Use the dockerfile from https://docs.docker.com/samples/wordpress/
  - Add a named volume to wordpress service
  - A- What happens when you run docker-compose up?
  - B- What happens when you run docker-compose down?
  - C- What files are inside each volume?
  - D- How can you delete all volumes?

Docker Compose Profiles

- Profiles allow adjusting the Compose application by selectively enabling services.
- This is achieved by assigning each service to zero or more profiles
- If a profile is not defined for a service, it belongs to all profiles

```
services:
  db:
    image: mysql
    profiles:
     - development

  wordpress:
    image: wordpress
    profiles:
     - development
     - production
```

- To enable profiles:
  - docker-compose –profile development up
  - “COMPOSE_PROFILES=development” (environment variable)
    “docker-compose up”

Hands-on Profiles

- **Exercise 5**
  - Use the dockerfile from https://docs.docker.com/samples/wordpress/
  - Add different profiles to each service (development, production)
  - Run docker compose up
  - Run docker compose –profile development up

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

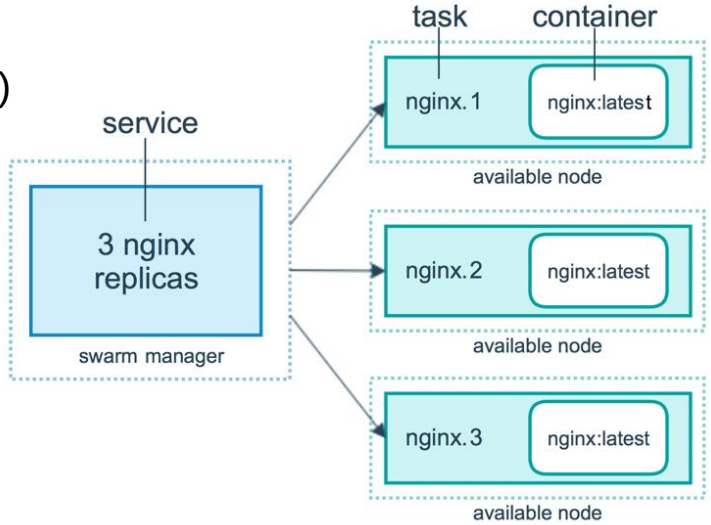
```
version: '3.2'
services:
  db:
    image: mysql
    deploy:
      resources:
        limits:
          cpus: '0.001'
          memory: 50M
        reservations:
          cpus: '0.0001'
          memory: 20M
```

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



EDEM

Docker Swarm

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

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

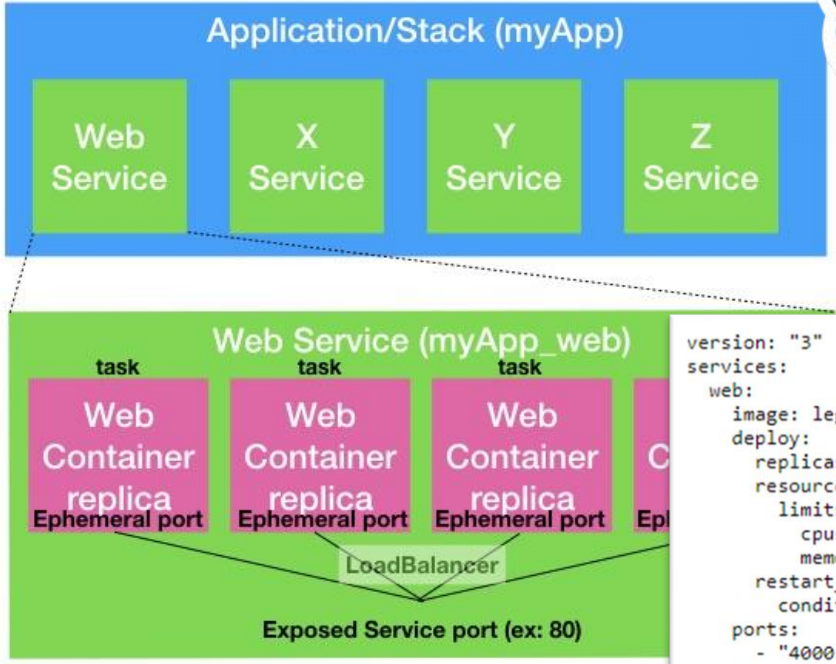
Deploy application
```
docker stack deploy -c docker-compose.yml myApp
```

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
```

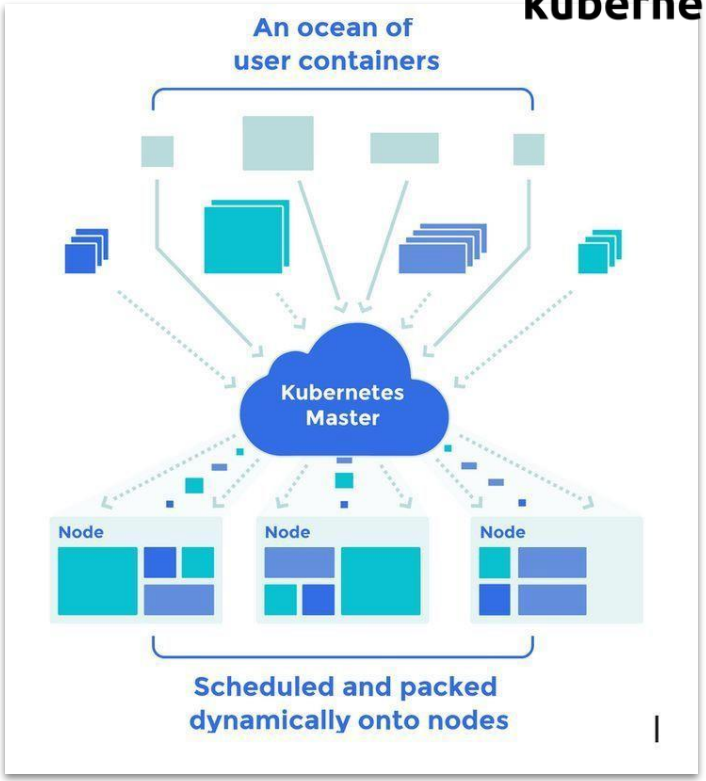



```
version: "3"
services:
  web:
    image: legabz/hellokube:swagger
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 1G
      restart_policy:
        condition: on-failure
    ports:
      - "4000:8080"
    networks:
      - webnet
networks:
  webnet: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

3 Kubernetes

Kubernetes

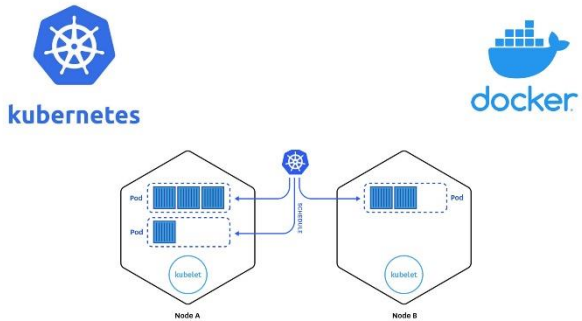
- Large and small software companies deploying thousands of container instances daily
  - How can we manage this complexity?
- Originally developed by Google.
- Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized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



Kubernetes

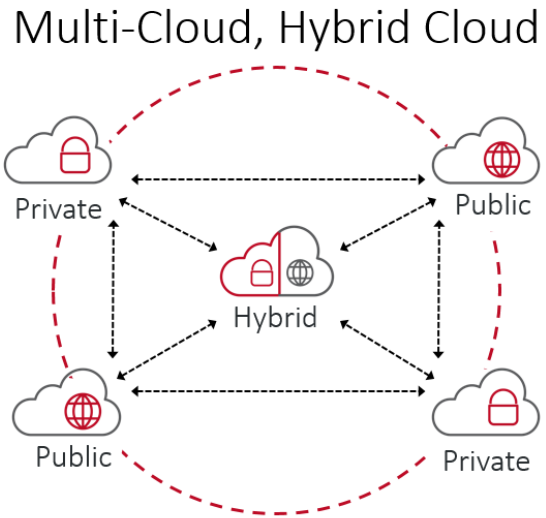
Features:

- 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

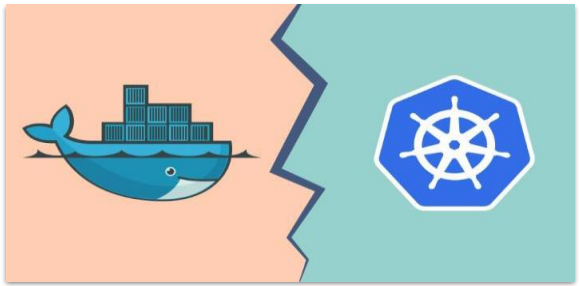
Microsoft Azure

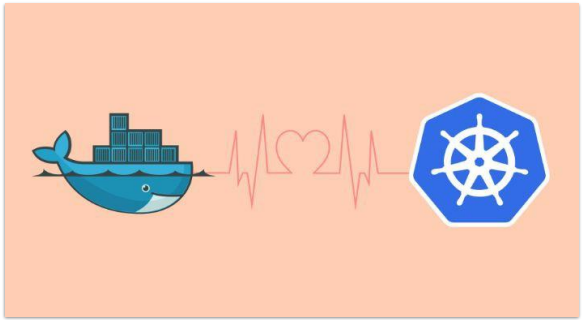
aws

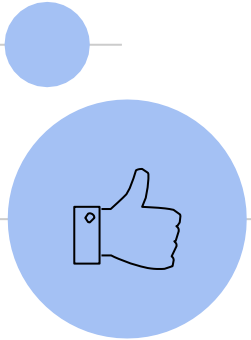
4 Tech Summary

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







Thanks!

Any Questions ?

You can find me at

https://www.linkedin.com/in/alberto-hernandez-cantos-9502a8a3/

alhercan@gmail.com