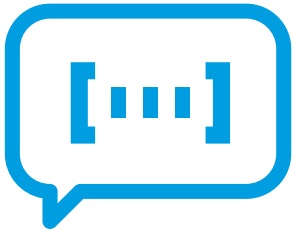
netmind
a bts company

Spring Boot

# Contenirización y despliegue en cloud

netmind
a bts company

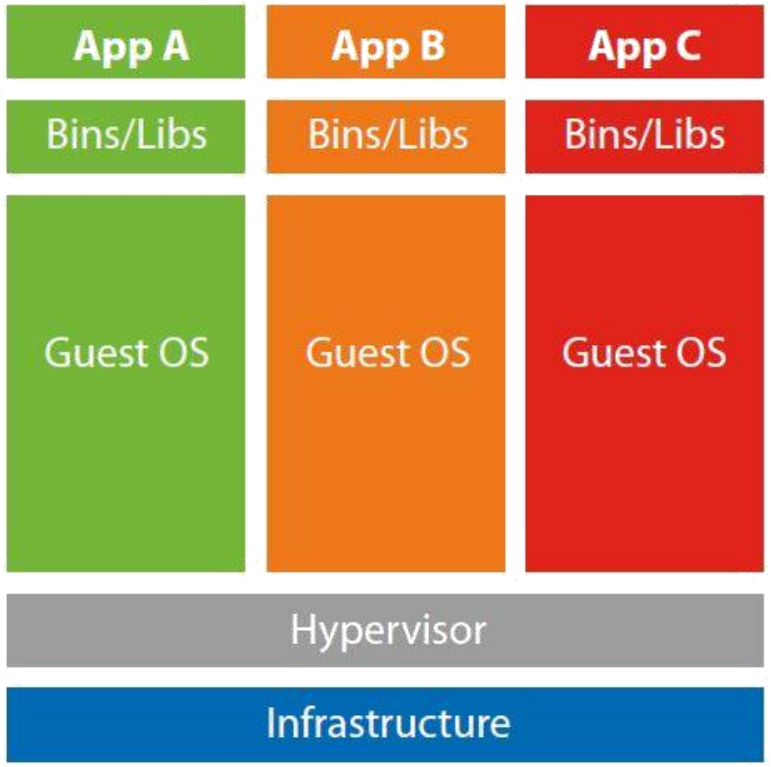


# Fundamentos de Docker

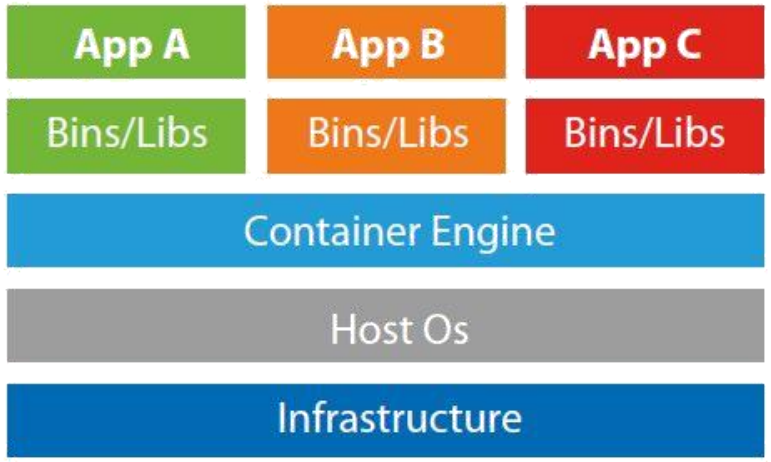
01

# Containers vs VMS

## Virtual Machines

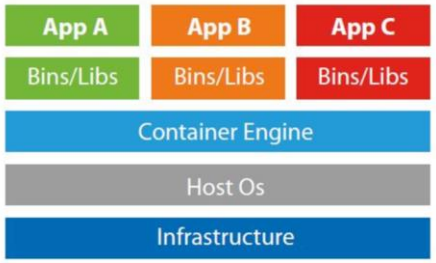



## Containers



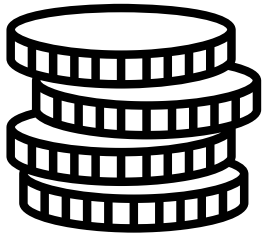
# Containers advantages

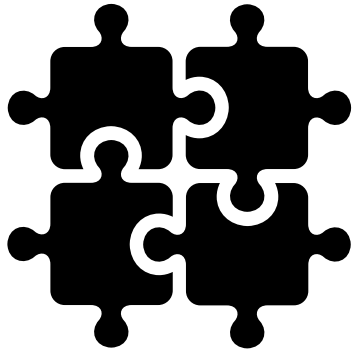
- Density and performance
- Native Cloud Applications(Scale-Out)
- Cost (licensing)
- DevOps
- CI/CD – faster deploy





NETFLIX





# Docker


- Docker is an open platform for **developing, shipping, and running applications**.
- Docker enables you to **separate your applications from your infrastructure** so you can deliver software quickly**.**
- You can **manage your infrastructure in the same ways you manage your applications**.
- By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly **reduce the delay between writing code and running it** in production**.**

# Docker Ecosystem

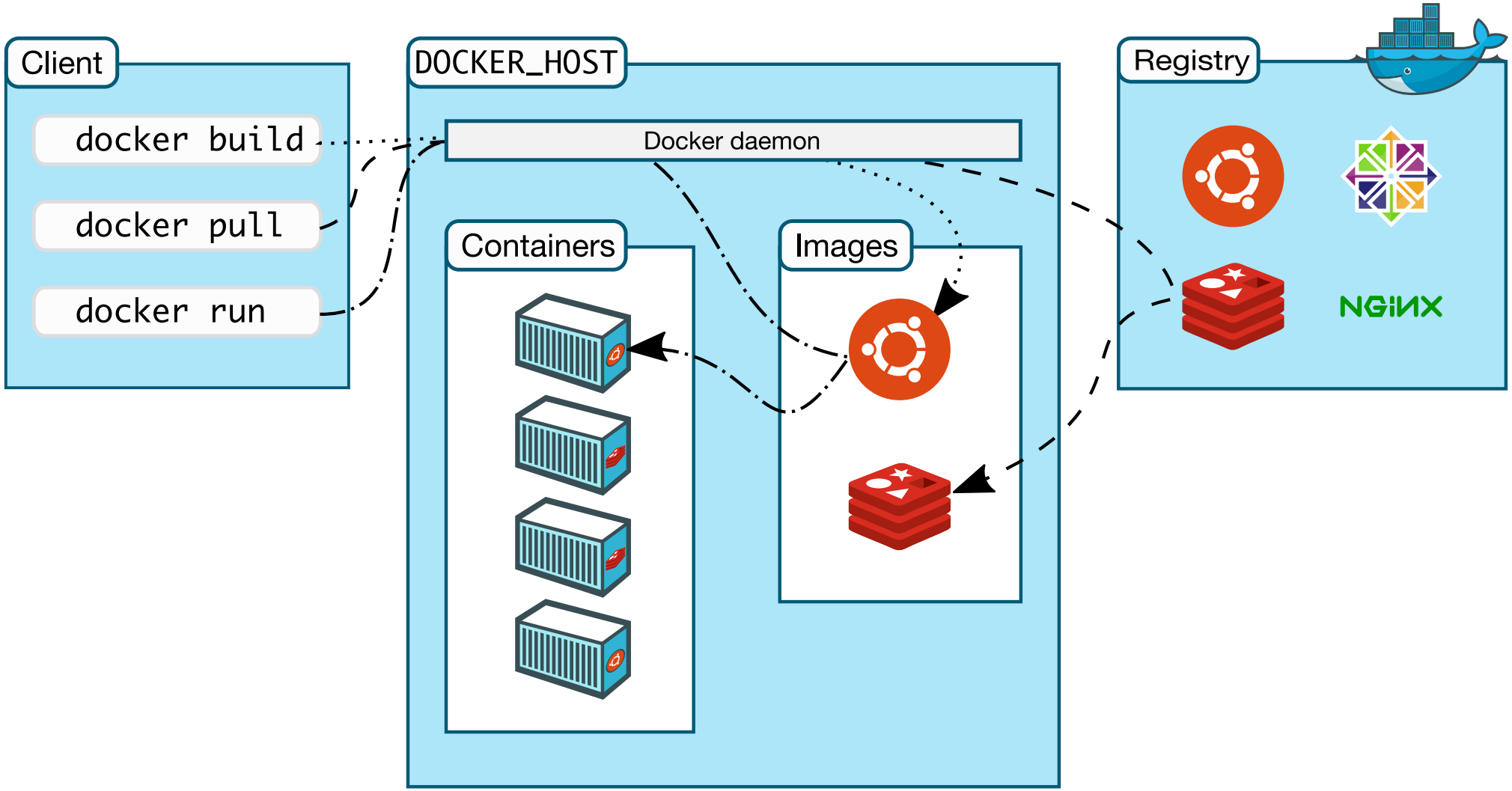



Image Source: docker.com

# Docker Ecosystem

- **Docker engine**
  - Container creation engine
  - Container Execution Engine
  - **Docker Daemon**
    - Build images → containers
    - Manage containers
    - RESTful APIs
  - **Docker-CLI**
    - Command line for Docker management
- **DockerHub**
  - https://hub.docker.com/
  - Offers Docker services
  - Public Image Library
  - Storage of our images
  - Automated builds

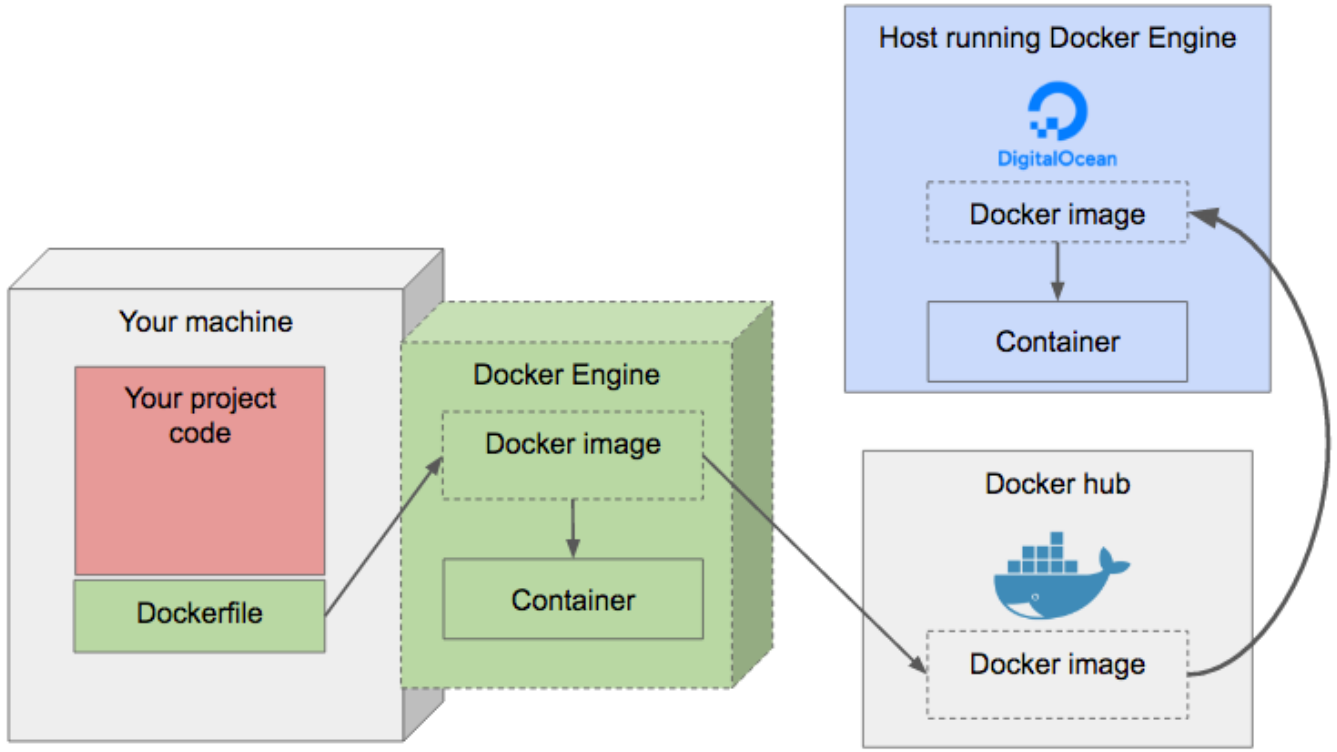



Image Source: lukewilson.net

# Installation in WSL Ubuntu

🔗 **Uninstall old versions**

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

🔗 **Set up the repository**

```
sudo apt-get update

sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

# Installation in WSL Ubuntu

🔗 **Install Docker Engine**

```
sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

🔗 **Start the daemon manually**

```
sudo dockerd
```

🔗 **Test installation**

```
sudo docker run hello-world
```

🔗 **Stop the daemon manually**

```
CTRL + C
```

# Executing dockerd and docker without sudo

### 🔗 Dockerd

```
asudo apt-get install -y uidmap

id -u
whoami
grep ^$(whoami): /etc/subuid
grep ^$(whoami): /etc/subgi

sudo apt-get install -y dbus-user-session

#exit and enter again
```

### 🔗 Docker

```
Sudo usermod -aG docker ${USER}
su - ${USER}
groups
sudo usermod -aG docker <username>
```

# Docker commands

🖇 **Check that docker is correctly running and that you have permission to use the engine**

```
docker info
```

🖇 **Pull an image from the official registry, eg: debian:latest (you can browse https://store.docker.com if you want to find other images).**

```
docker pull debian:latest
```

🖇 **check images present in the docker engine:**

```
docker images
```

🖇 **Run a container from an image**

```
docker run debian:latest
```

🖇 **Show running and non running containers**

```
docker ps
docker ps -a
```

# Docker commands

### 🔗 run a command

```
docker run debian ls /bin
docker run debian cat /etc/motd
```

### 🔗 interact with the shell

```
docker run -i Debian
# -t for allocate a tty
docker run -t -i debian
# -d keeps running in the background
docker run -d -t -i debian
```

### 🔗 start a stopped container

```
docker ps –a
# -i to have stdin
docker start -i 85bcdca6c38f
docker start -i 85bcd
docker start -i 85
docker start -i 85bcdca6c38f07e3f8140cbf8b4ad37fd80d731b87c6945012479439a450a443
docker start -i pensive_hodgkin
```

# Docker commands

### 🔗 commit

```
# You can modify files inside a container. If you restart the same container you can note that these changes
are still present. However they will not be present in the other container (even if they are running the same
image) because docker uses a copy-on-write filesystem. Use the command docker diff to show the difference of
a container from its image.
# Remember that all changes inside a container are thrown away when the container is removed. If we want save
a container filesytem for later use, we have to commit the conainer (i.e take a snapshot).

docker commit CONTAINER

# This operation creates a new image. This image in turn can be used to start a new container.

docker run -it debian:jessie
         git
         apt-get update && apt-get install -y git
         git
         exit
docker commit <cont_id> <repo>/debian:<tag>
docker images
```

### 🔗 remove

```
docker rm
# removes all dead container.
docker prune
```

# Docker commands

### 🔗 docker run options

*--rm* to remove the container automatically when it terminates
*-d/--detach* to run a container in the background
*-u/--user* to run the container as a different user
*-w/--workdir* to start the container in a different directory
*-e/--env* to set an environment variable
*-h/--hostname* to set a different hostname (the host name inside the container)
*--name* to set a different name (the name of the container in the docker engine)

also you may type **docker run --help** to display all configuration keys

### 🔗 other docker commands

**docker cp** to transfer files from/into the container
**docker exec** to have launch a separate command (very useful for providing a debugging shell -> docker exec -t -i CONTAINER bash)
**docker top** to display the processes running inside the container
**docker stats** to display usage statistics
**docker logs** to display the container output
**docker attach** to reattach to the console of a detached container

### 🔗 ports

```
docker run -it -d -p 8888:8080 tomcat:8.0
docker logs <container_id>
```

# Docker commands

🔗 **push** publishes an image in dockerhub. You will need an account.

```
docker login <REGISTRY_HOST>:<REGISTRY_PORT>
docker tag <IMAGE_ID> <REGISTRY_HOST>:<REGISTRY_PORT>/<APPNAME>:<APPVERSION>
docker push <REGISTRY_HOST>:<REGISTRY_PORT>/<APPNAME>:<APPVERSION>

# push to dockerhub
docker login
docker tag my-image myhubusername/debian:v1
docker push myhubusername/debian:v1


# push to custom repo
docker login repo.company.com:3456
docker tag 19fcc4aa71ba repo.company.com:3456/myapp:0.1
docker push repo.company.com:3456/myapp:0.1
```

🔗 **delete images**

```
docker image rm <image_id>
docker rmi $(docker images | grep ‘<text>')
```

# Building containers

🔗 **Build** creates a new image from a previous image or a Dockerfile

```
# Generate an image from another existing (a copy with another page)
docker tag jboss/wildfly myimage:v1

#from a Dockerfile
docker build -t <repo>/<image_name>:<tag> <Dockerfile_path>
docker build -t <repo>/<image_name>:<tag> -f <NotDockerFileName_path>
#cleaning the cache
docker build -t <repo>/<image_name>:<tag> < Dockerfile_path> --no-cache=true

docker build -t dockerapp:v0.1 .
docker run -d -p 5000:5000 <new_image_id>
```

🔗 **Dockerfile** is a file that defines a new image to be created. It uses keywords like

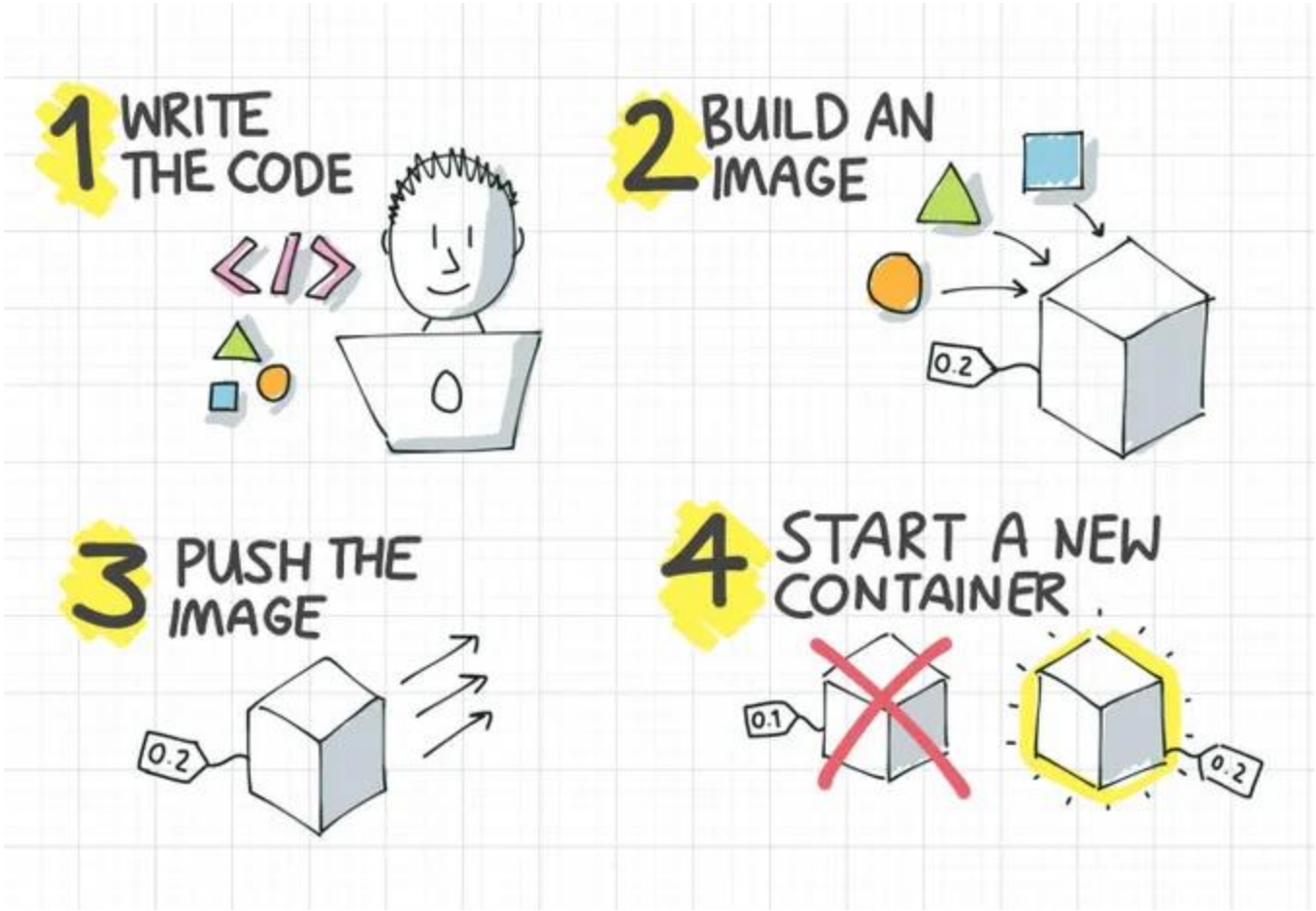
*FROM* defines the base image
WORKDIR sets the working directory or context inside the image
*RUN* a build step
*CMD* the command the container executes by default
*COPY* copies a file/folder to the container from a location to a destination in the Docker container.
*ADD* copy files/directories into a Docker image
*EXPOSE*  expose a port inside of the image to the outside world.

# Building containers

🔗 **Dockerfile**

```
#Dockerfile
FROM busybox
RUN touch testfile
RUN /bin/bash -c echo "Next build step..."
COPY testfile /

#Build the image
docker build -t local_busybox -f Dockerfile ./

# https://docs.docker.com/engine/reference/builder/
```

# Developing with Containers




**Read:** https://www.tutorialworks.com/container-development-workflow/

# Docker Compose

- Docker Compose is a tool for running **multi-container** applications on Docker defined using the Compose file format.
- A **Compose file** is used to define how the one or more containers that make up your application are configured.
- Once you have a Compose file, you can create and start your application with a single command: **docker compose up**.

# Docker Compose

Using Docker Compose is basically a three-step process:

1. Define your app's environment with a **Dockerfile** so it can be reproduced anywhere.
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 and run your entire app.

# Docker Componse

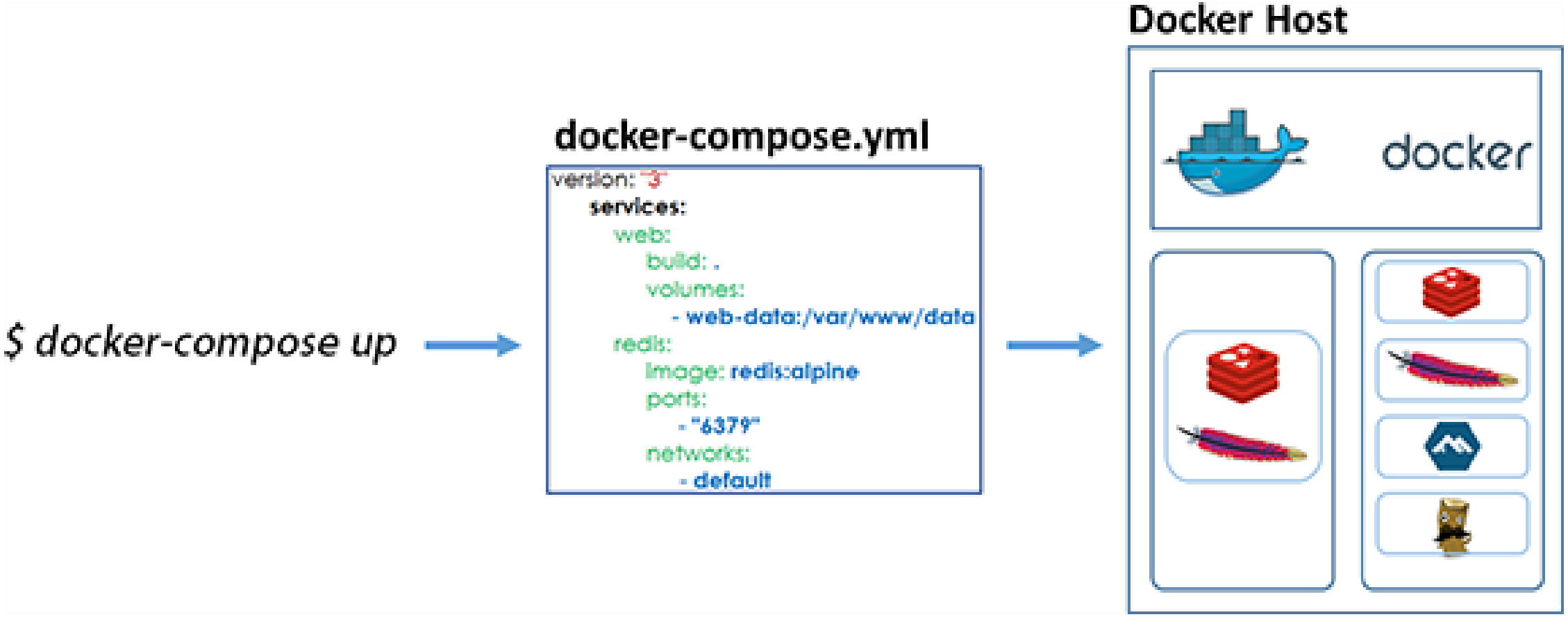
🔗 **docker compose yml**

```
# example 1
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
  redis:
    image: redis
```

🔗 **docker compose yml**

```
# example 2

version: "3.9"  # optional since v1.27.0
services:
  web:
    build: .
    ports:
      - "8000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

# Docker Compose Architecture



# Docker Compose commands

🔗 **show commands**
docker-compose --help
docker-compose <command> --help

🔗 **Run a compose with default docker-compose.yaml**
docker-compose up
docker-compose up -d

🔗 **With specific file**
docker-compose –f <file.yaml> up

🔗 **Show associated containers**
docker-compose –f <file.yaml> ps

🔗 **Delete associated containers**
docker-compose –f <file.yaml> rm

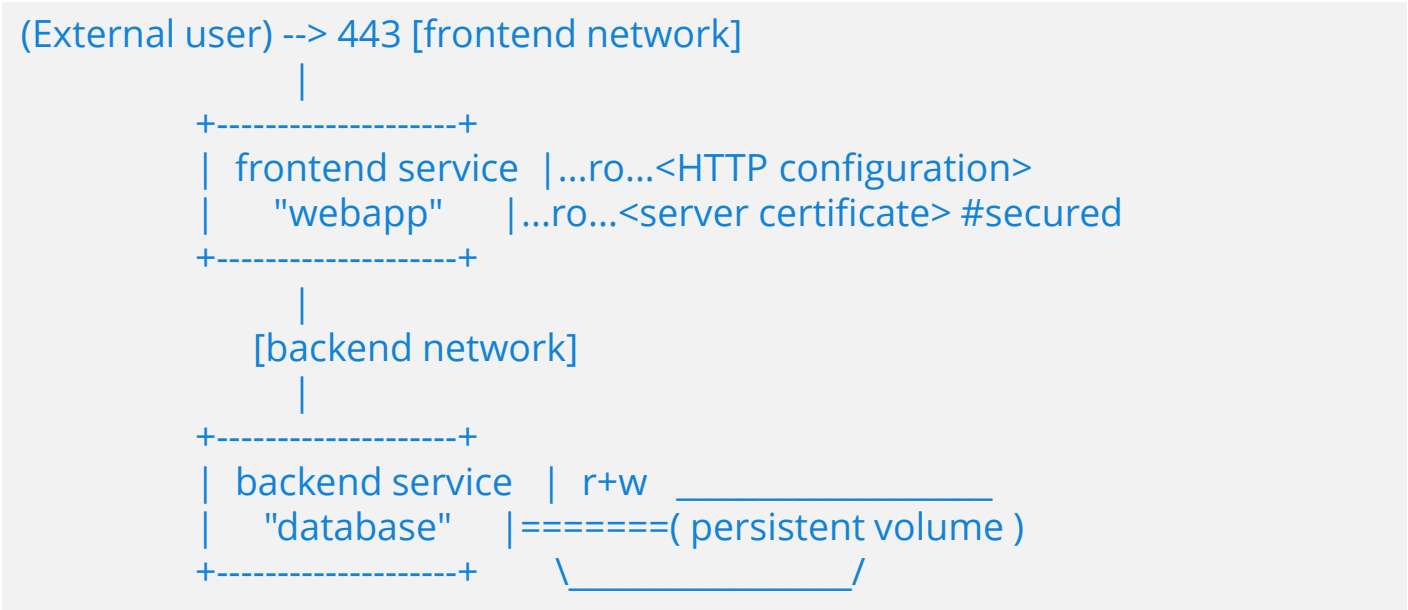
🔗 **Scale a service**
docker-compose –f <file.yaml> up --scale <service1>=<num> --scale <service2>=<num>
*# Must allocate ports in yaml, example:*
*# ports:*
*#  -"[min-max]:inner"*

🔗 **logs**
docker-compose logs
docker-compose logs –f
docker-compose logs <service-name>

# Docker Compose services, volumes and networks

- Computing components of an application are defined as **Services**.
  - Some services require configuration data that is dependent on the runtime or platform. For this, the specification defines a dedicated concept: Configs.
  - A Secret is a specific flavor of configuration data for sensitive data that SHOULD NOT be exposed without security considerations.
- Services communicate with each other through **Networks**.
- Services store and share persistent data into **Volumes**.
- A **Project** is an individual deployment of an application specification on a platform.

# Docker Compose services, volumes and networks



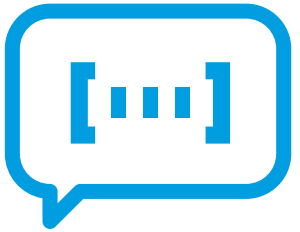

Extract from: https://docs.docker.com/compose/compose-file/

- 2 services, backed by Docker images: webapp and database
- 1 secret (HTTPS certificate), injected into the frontend
- 1 configuration (HTTP), injected into the frontend
- 1 persistent volume, attached to the backend
- 2 networks

# Docker Compose services, volumes and networks

**docker compose yml**

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

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

volumes:
  db-data:
    driver: flocker
    driver_opts:
      size: "10GiB"

configs:
  httpd-config:
    external: true

secrets:
  server-certificate:
    external: true

networks:
  # The presence of these objects is sufficient to
define them
  front-tier: {}
  back-tier: {}
```



# Dockerización de microservicios

02

# Dockerizar los servicios Spring Boot

1. Añadir el plugin **dockerfile-maven-plugin** de Spotify en el pom de cada proyecto.

```
<plugins>...
          <plugin>
                <groupId>com.spotify</groupId>
                <artifactId>dockerfile-maven-plugin</artifactId>
                <version>1.4.13</version>
                <executions>
                    <execution>
                        <id>default</id>
                        <goals>
                            <goal>build</goal>
                            <goal>push</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <repository>dockerspringgcp-${project.artifactId}</repository>
                    <tag>${project.version}</tag>
                    <buildArgs>
                        <JAR_FILE>${project.build.finalName}.jar</JAR_FILE>
                    </buildArgs>
                </configuration>
             </plugin>
...</plugins>
```

# Dockerizar los servicios Spring Boot

2. Crear **dockerfiles** para cada proyecto.
    - Reemplazar **compiled_project** con el nombre del ID de artefacto del proyecto.

🔗 **Dockerfile**

```
FROM adoptopenjdk/openjdk11:alpine-jre

# Refer to Maven build -> finalName
ARG JAR_FILE=target/<compiled_project>.jar

# cd /opt/app
WORKDIR /opt/app

# cp target/spring-boot-web.jar /opt/app/app.jar
COPY ${JAR_FILE} app.jar

# java -jar /opt/app/app.jar
ENTRYPOINT ["java","-jar","app.jar"]
```

🔗 **Crear imagen**

```
docker build -t <account>/<project>:<ver>.
```

🔗 **Crear imagen**

```
docker run -d -p 9090:9090 <account>/<project>:<ver>
```

# Dockerizar los servicios Spring Boot

3. Crear un archivo **docker-compose** para administrar los múltiples microservicios.

```
version: '3.8'
services:
  eureka-server:
    image: dockerspringgcp-eurekaserver:0.0.1-SNAPSHOT
    build: eurekaserver/
    ports:
      - 8761:8761
  eureka-client:
    image: dockerspringgcp-eaurekaclient:0.0.1-SNAPSHOT
    build: eaurekaclient/
    depends_on:
      - eureka-server
      - journal-server
    environment:
      SPRING_APPLICATION_JSON:
'{"eureka":{"client":{"serviceUrl":{"defaultZone":"http://eureka-
server:8761/eureka"}}}}'
    ports:
      - 8081:8081
  journal-server:
    build: journal_server/
    image: dockerspringgcp-journal_server:0.0.1-SNAPSHOT
    depends_on:
      - eureka-server
    environment:
      SPRING_APPLICATION_JSON:
'{"eureka":{"client":{"serviceUrl":{"defaultZone":"http://eureka-
server:8761/eureka"}}}}'
    ports:
      - 8080:8080
```

# Dockerizar los servicios Spring Boot

4. Actualizar los archivos **application.properties** de los servicios.
   - **Eurekaserver: applications.properties**

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
logging.level.com.netflix.eureka=OFF
logging.level.com.netflix.discovery=OFF
spring.cloud.service-registry.auto-registration.fail-fast=true
```

   - **Client (eurekaclient) applications.properties**

```
spring.application.name=journalclient
spring.cloud.service-registry.auto-registration.fail-fast=true
spring.cloud.discovery.enabled=true
eureka.client.service-url.defaultZone=${EUREKA_SERVER:http://eureka-server:8761/eureka}
```

   - **Journal Server applications.properties**

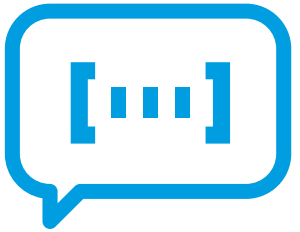
# Dockerizar los servicios Spring Boot

4. Actualizar los archivos **application.properties** de los servicios.
    - **Service Server applications.properties**

```
spring.application.name=<service_name_server>
server.port=8081
eureka.client.service-url.defaultZone=${EUREKA_SERVER:http://eureka-server:8761/eureka}
spring.cloud.service-registry.auto-registration.fail-fast=true
spring.cloud.discovery.enabled=true
```

5. Lanzar Docker compose.

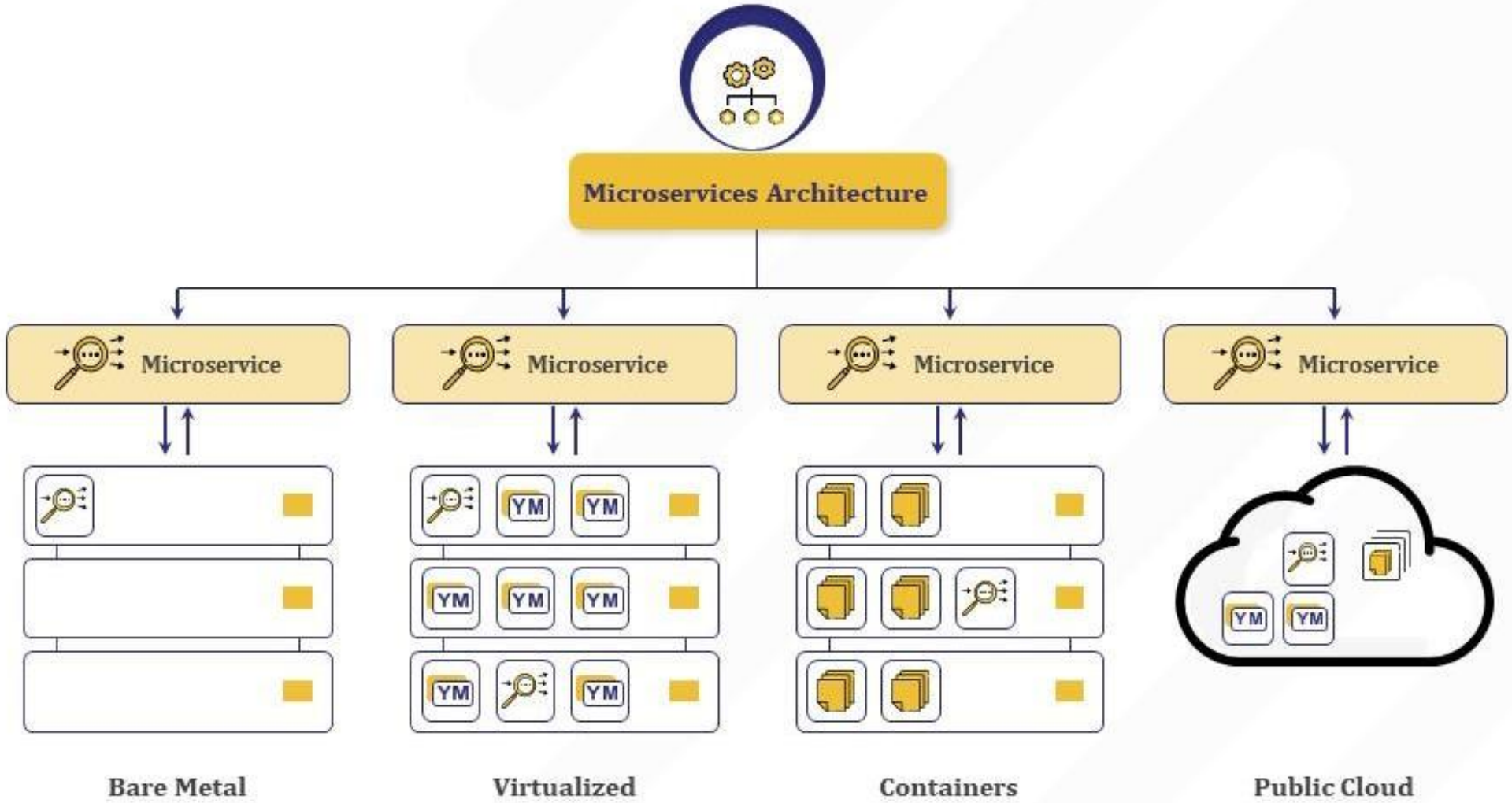
```
docker-compose up
```



# Microservicios y la Cloud

03

# Microservicios y la cloud



# Opciones AWS

- **EC2:** desplegado en un ASG (Auto Scaling Group). Para pequeños proyectos es rápido y fácil de configurar.
- **Lambda for containers:** Es el vínculo entre la portabilidad del contenedor y las potentes funciones lambda.
- **ECS/Fargate:** ECS es un servicio administrado de AWS. Está bien integrado con otros servicios como IAM o Fargate, lo que lo convierte en la elección perfecta para implementar aplicaciones acoplables en AWS, hasta ahora. Almancenaremos nuestra imagen en ECR
- **EKS/Fargate:** brinda la flexibilidad para iniciar, ejecutar y escalar aplicaciones de Kubernetes en AWS. Al igual que con ECS, se puede usar con Fargate para crear una solución sin servidor.

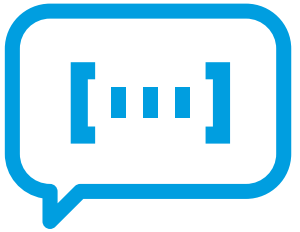
- **Saber más:** https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/microservices-on-aws.pdf

# Comparación de opciones

| Criteria | EC2 | ECS/Fargate | Classic Lambda | Lambda for container |
|---|---|---|---|---|
| Maintenance | High | Low | None | None |
| Configuration | Medium | Low | Minimal | Minimal |
| Pricing | Per hour | Per VCPU and per Gb Per hour | Per request and duration | Per request and duration |
| Portability | Very High | High | Limited | High |
| Technical Limitations | none | Medium (Windows containers are not supported) | Medium<br>Runtimes Available<br>Timeout<br>No vertical scaling | Medium<br>Lambda supports only Linux-based container images.<br>Timeout<br>No vertical scaling |

netmind
a bts company

04

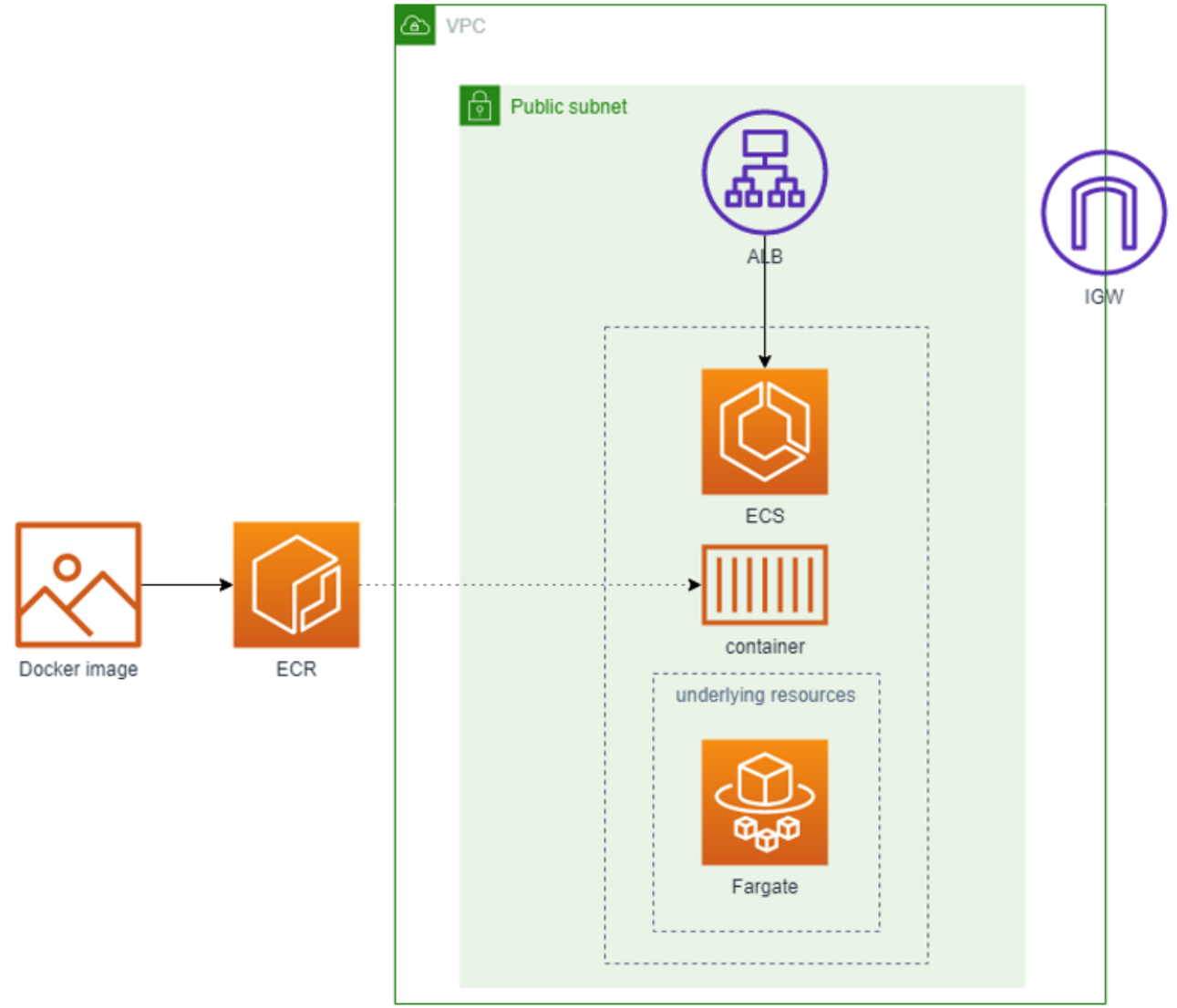


# Despligue en ECS con Fargate

# ECS


- Amazon **Elastic Container Service** (Amazon ECS) es un servicio de administración de contenedores rápido y altamente escalable que facilita la ejecución, la detención y la administración de contenedores en un clúster.
- Los contenedores se definen en una **definición de tarea** que utiliza para ejecutar tareas individuales o tareas dentro de un servicio.
- En este contexto, un **servicio** es una configuración que le permite ejecutar y mantener una cantidad específica de tareas simultáneamente en un clúster.
- Se pueden ejecutar tareas y servicios en una **infraestructura serverless** administrada por AWS Fargate.

# Arquitectura



# Conceptos clave

- **Cluster:** Una agrupación lógica de sus servicios o tareas
- **Task Definition:** Especificación sobre cómo se deben ejecutar sus contenedores en AWS ECS
- **Task:** Una instancia de la definición de tarea
- **Service:** un administrador de tareas

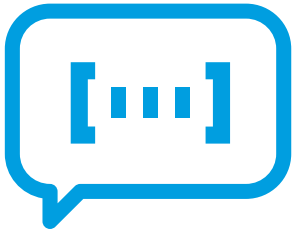
- **ECR:** Servicio de registro de imágenes de AWS
- **IAM:** Servicio de gestión de identidad de AWS
- **Rol:** un conjunto de permisos para acciones y recursos en AWS. Es tomado por una aplicación o otro servicio.

# Proceso

1. **Crear un clúster de AWS ECS**
2. **Crear una definición de tarea de AWS ECS**
3. **Crear un servicio ECS de AWS**
4. **Acceder al servicio**

**Saber más:** https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/deploy-java-microservices-on-amazon-ecs-using-amazon-ecr-and-load-balancing.html

netmind
a bts company



# Despligue en Kubernetes/EKS

05

# Kubernetes

*From Greek **helmsman** or **pilot***

- Kubernetes es una plataforma **portable, extensible y de código abierto para administrar cargas de trabajo y servicios en contenedores**, que facilita tanto la configuración declarativa como la automatización.
- Un sistema de **orquestación de contenedores** de grado de producción.
  - https://kubernetes.io/
- Kubernetes proporciona:
  - Detección de servicios y equilibrio de carga
  - Orquestación de almacenamiento
  - Despliegues y reversiones automatizados
  - Embalaje automático en contenedores
  - Autosanación
  - Gestión de secretos y configuraciones
- Kubernetes se conoce popularmente como **K8**.

# Componentes de Kubernetes

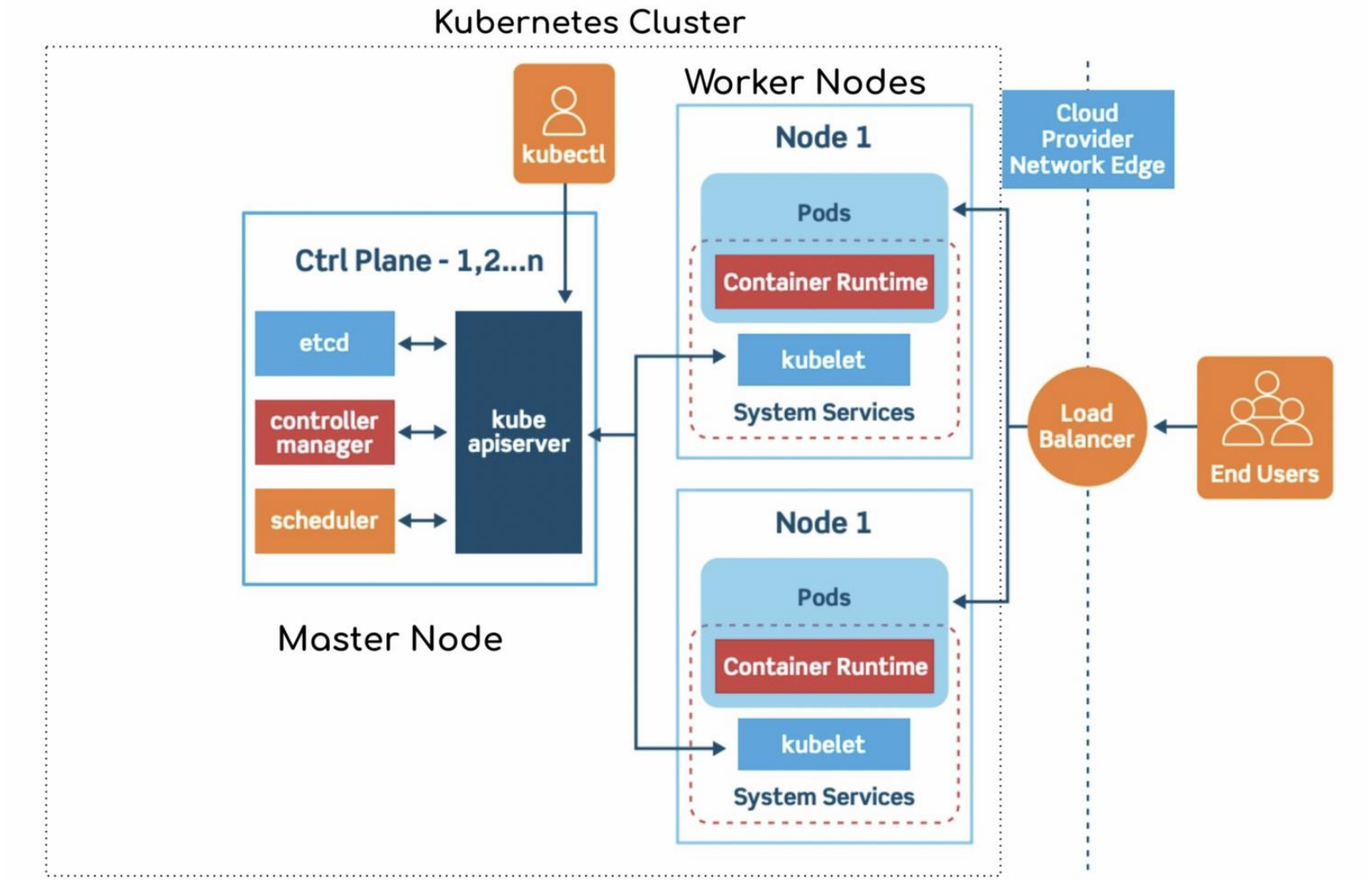


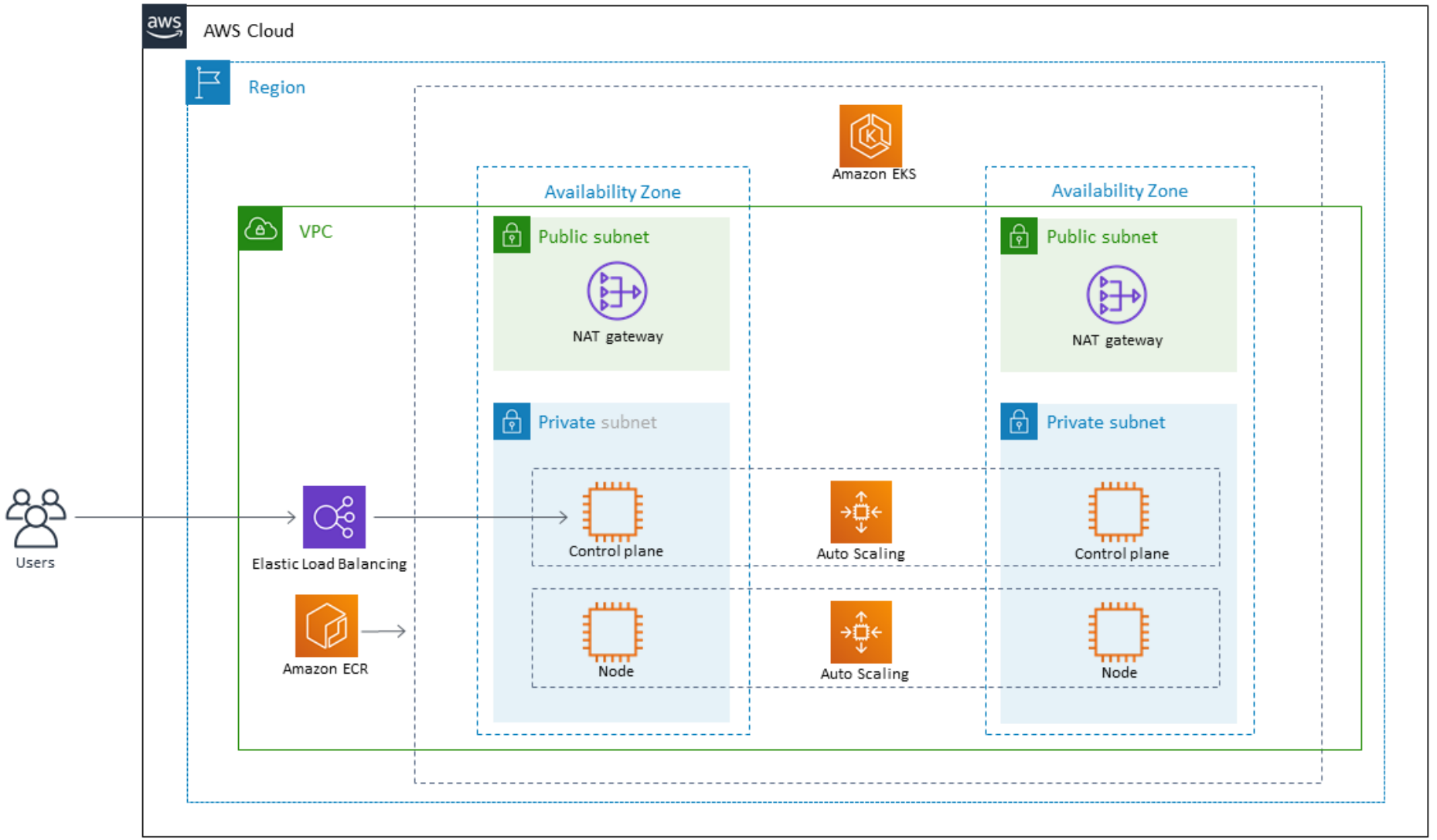

Image Source: medium.com/the-programmer/

Source : https://kubernetes.io/docs/concepts/overview/components/

# EKS


- Amazon **Elastic Kubernetes Service** (Amazon EKS) es un servicio administrado que puede utilizar para ejecutar Kubernetes en AWS sin necesidad de instalar, operar y mantener planos de control o nodos de Kubernetes.

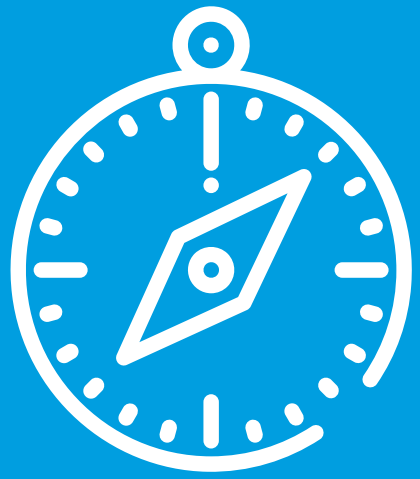
# Arquitectura



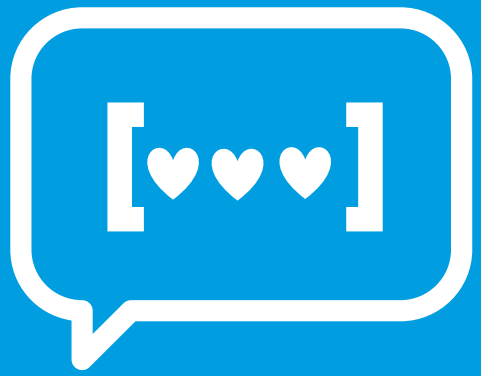
# Proceso

1. **Crear un clúster de AWS EKS (usando eksctl)**
2. **Crear un repositorio ECR y subir las imágenes**
3. **Despligar los microservicios (usando kubectl )**
4. **Acceder al servicio**

**Saber más:** https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/deploy-a-sample-java-microservice-on-amazon-eks-by-using-amazon-ecr-and-eksctl.html



# Next steps



**We would like to know your opinion!**

Please, let us know what you think about the content.
From Netmind we want to say thank you, we appreciate time and effort you have taking in answering all of that is important in order to improve our training plans so that you will always be satisfied with having chosen us
quality@netmind.es

netmind
a bts company

# Thanks!

Follow us: