@digitallab_pe
/digitallab.academy
@digitallab.academy

Docker & Kubernetes

Digital Lab Academy

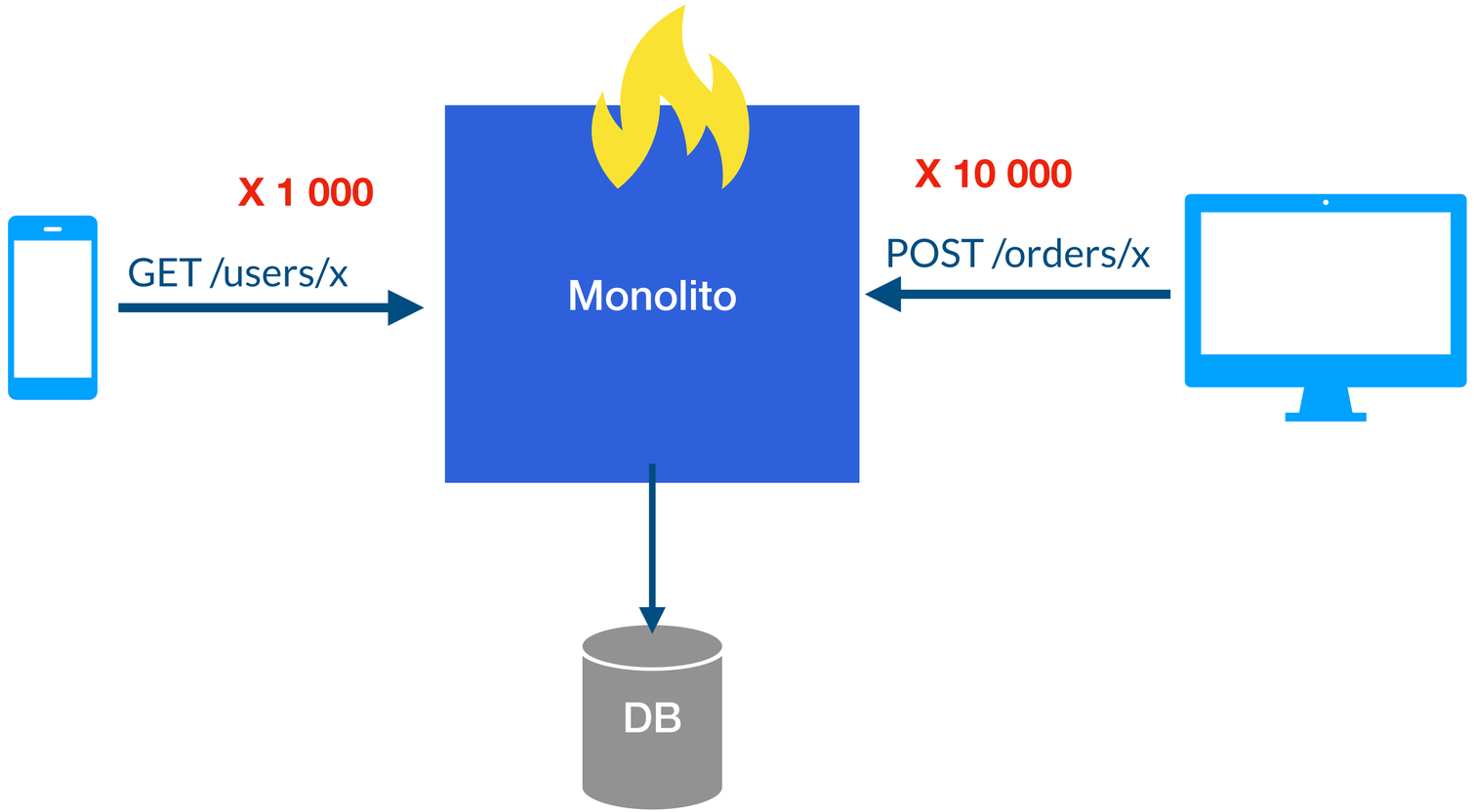
@edumar111
/edumar111
/edumar111

Eduardo Marchena

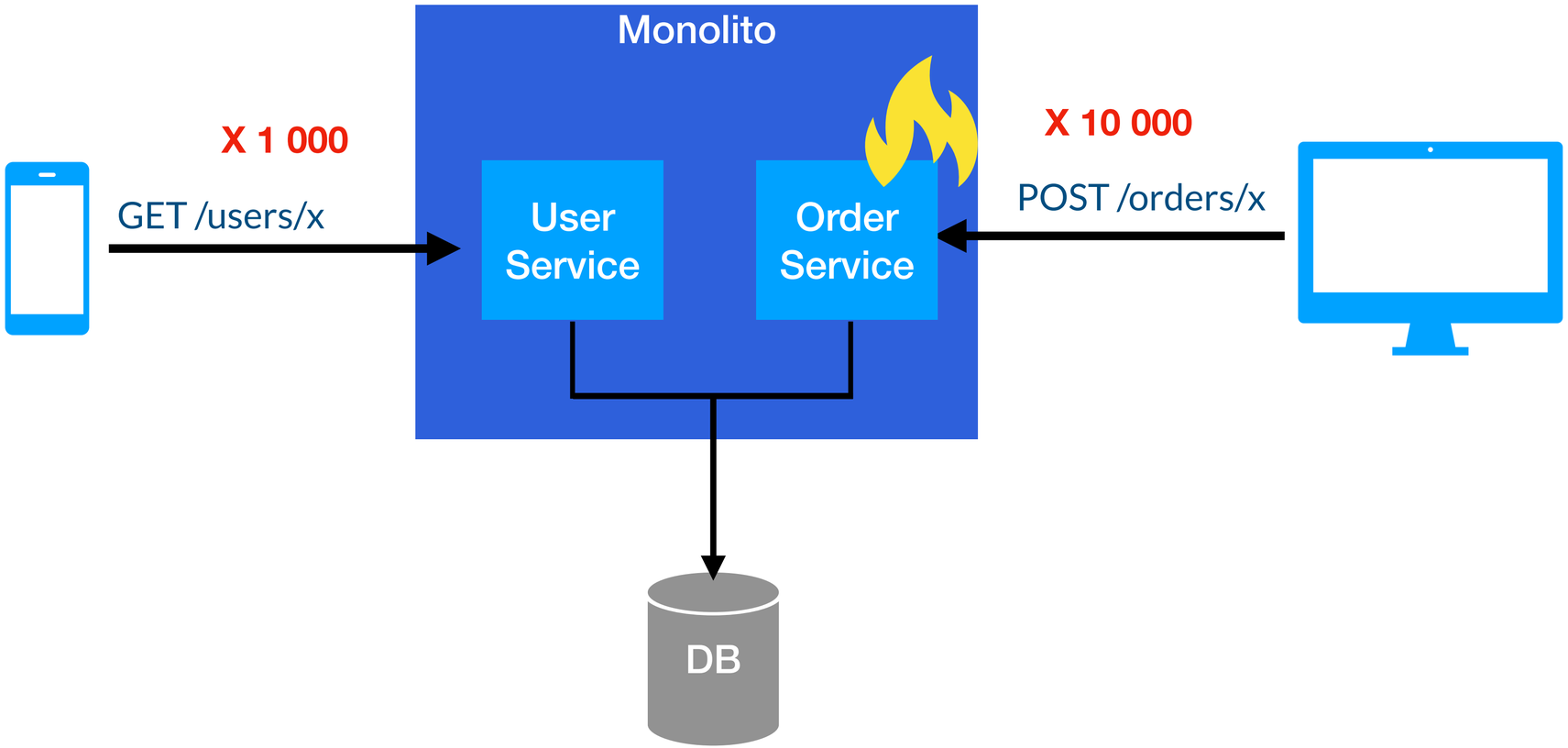
Software Architect

Monolitos

Los sistemas monolíticos o aplicaciones monolíticas son aquellos sistemas que agrupan sus funcionalidades y servicios en una base única de código. Esto implica que todo lo relacionado al sistema se encuentra dentro de un mismo proyecto



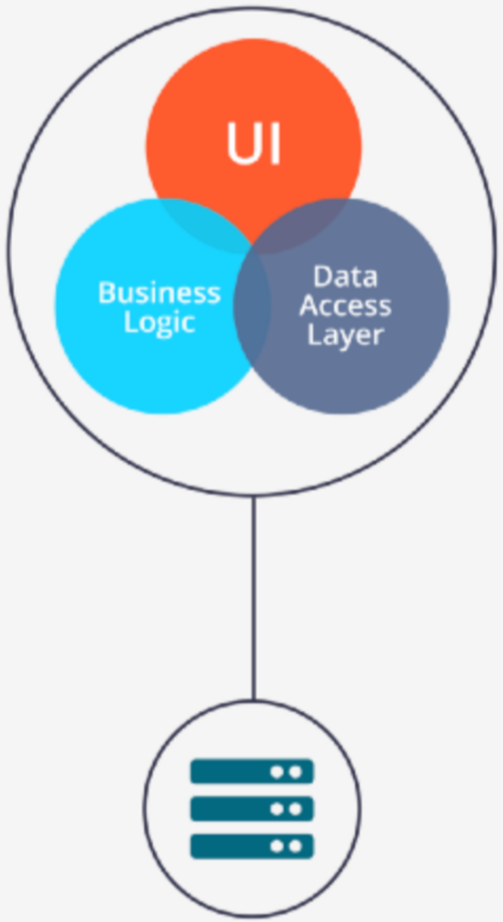
Monolitos


Monolito
X 1 000
GET /users/x
User Service
Order Service
X 10 000
POST /orders/x
DB

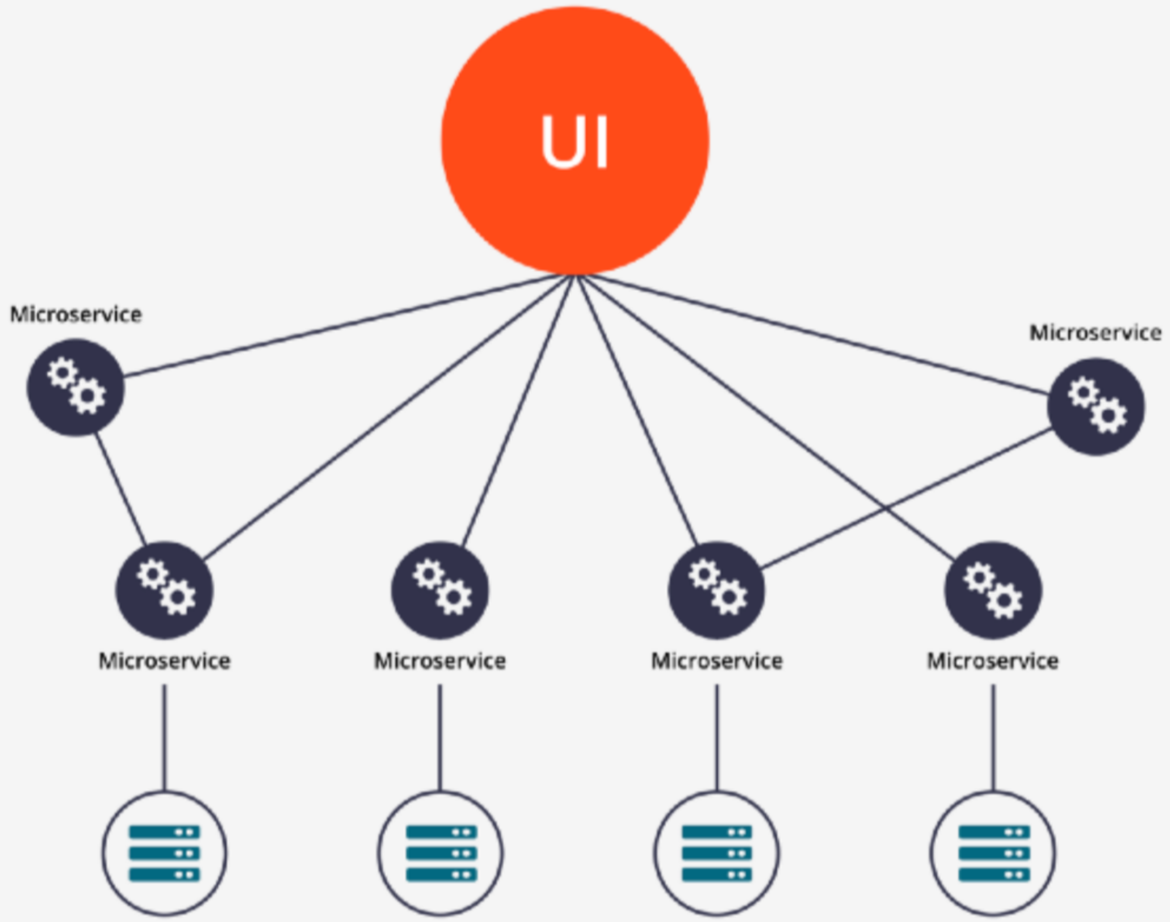
¿Que son los Microservicios?

Los **Microservicios**, son tanto una arquitectura como un modo para el desarrollo de software que consiste en construir una aplicación como un conjunto de pequeños **servicios independientes entre sí**, los cuales se **ejecutan en su propio proceso** y se comunican con mecanismos ligeros (normalmente REST). Pueden estar programado en **distintos lenguajes** y usar diferentes tecnologías de almacenamiento de datos

Monolitico VS Microservicios



Monolithic Architecture



Microservice Architecture

Microservicios: Ventajas

- Escalabilidad
- Funcionalidad modular, Desacoplamiento.
- Desarrollar y desplegar servicios de forma independiente.
- Uso de contenedores permitiendo el despliegue y el desarrollo de la aplicación rápidamente.

Contenedores

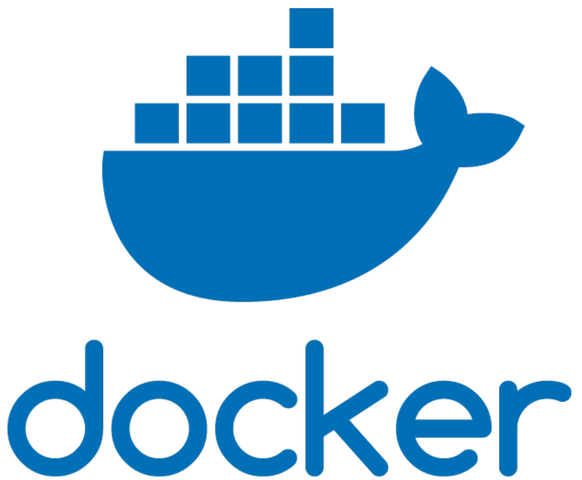
Son un paquete de elementos que permiten ejecutar una aplicación determinada en cualquier sistema operativo, que incluyen todo lo necesario para ejecutarse, implementando y ajustando la escala de aplicaciones en cualquier entorno, velozmente.

- Docker
- RKT - Rocket
- Mesos Container

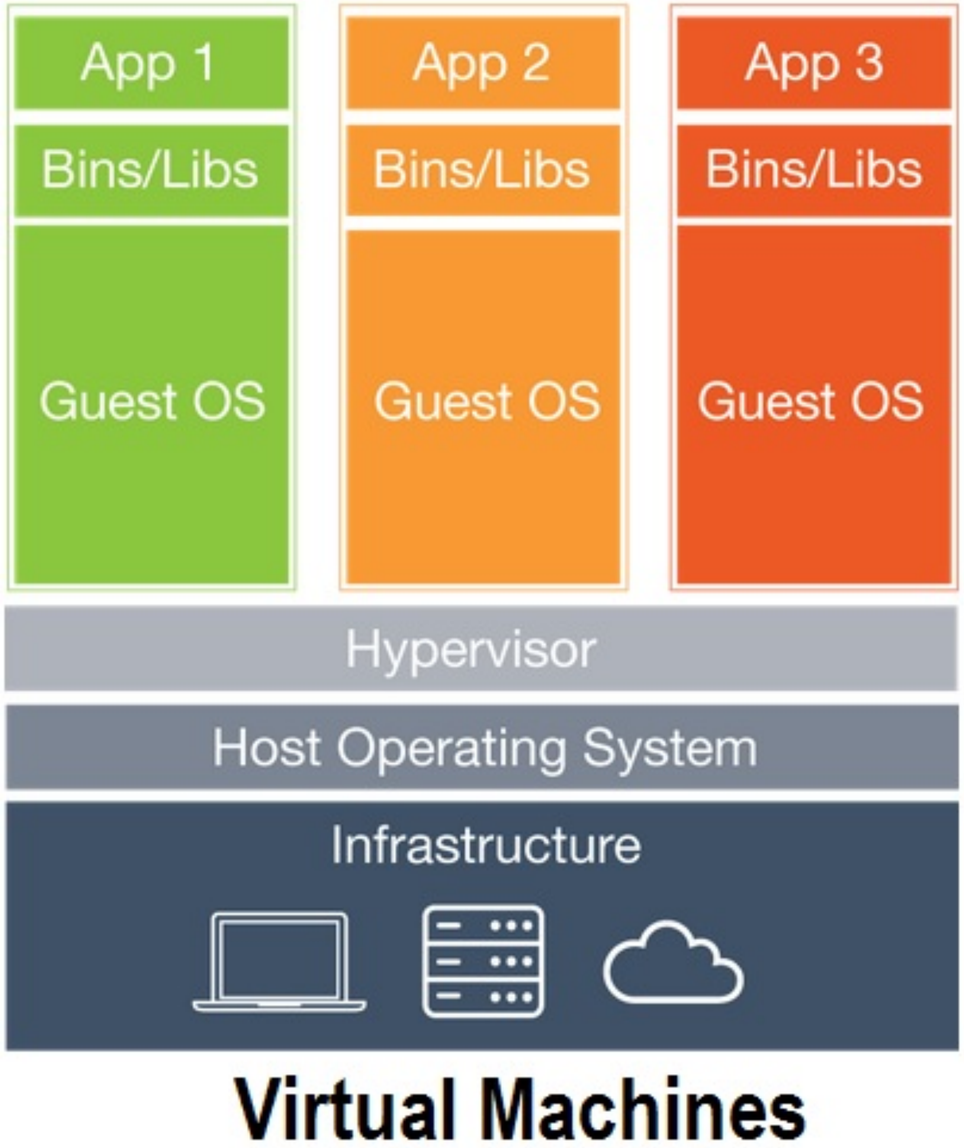
Docker

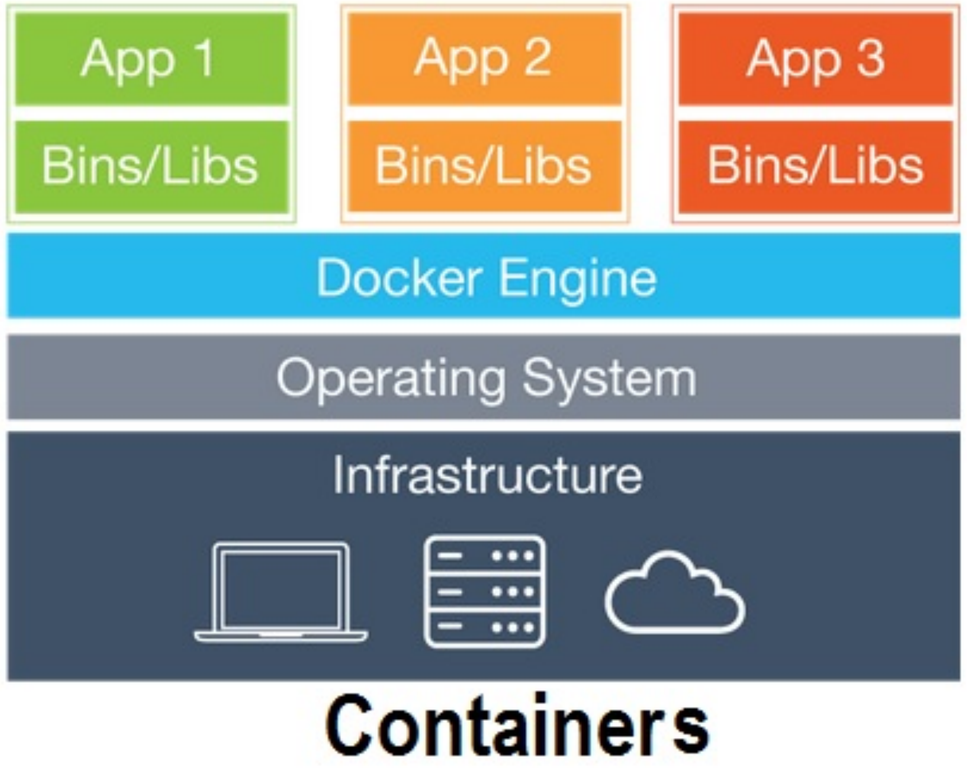
- Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos, que permite a los desarrolladores y administradores de sistemas a que desarrollen, envíen y ejecuten aplicaciones distribuidas, ya sea en computadoras portátiles, maquinas virtuales de centros de datos o en la nube

"Build, Ship and Run, any App, anywhere"



Docker VS Virtual Machine





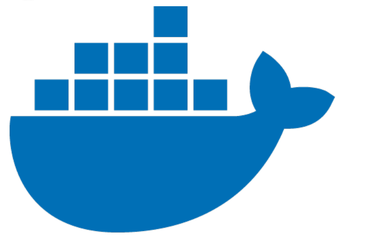
Docker Características

- Aislamiento: Aislamiento de los recursos a nivel de Kernel:
  - C groups
  - Namespaces

- Poco consumo de recursos.

- Livianos: Facilitan el almacenaje, distribución y despliegue.

- Rápida Ejecución: Puesta en marcha de los servicios en segundos.



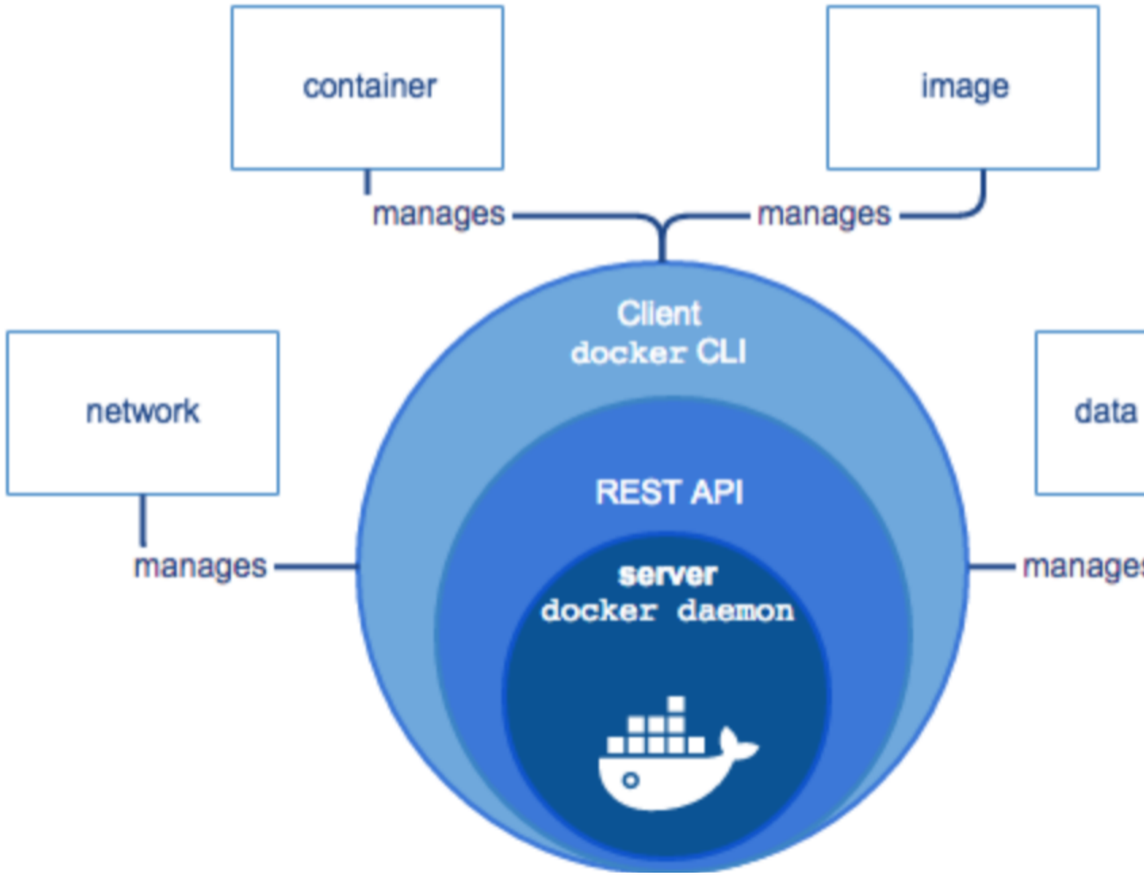
docker

Docker - Componentes

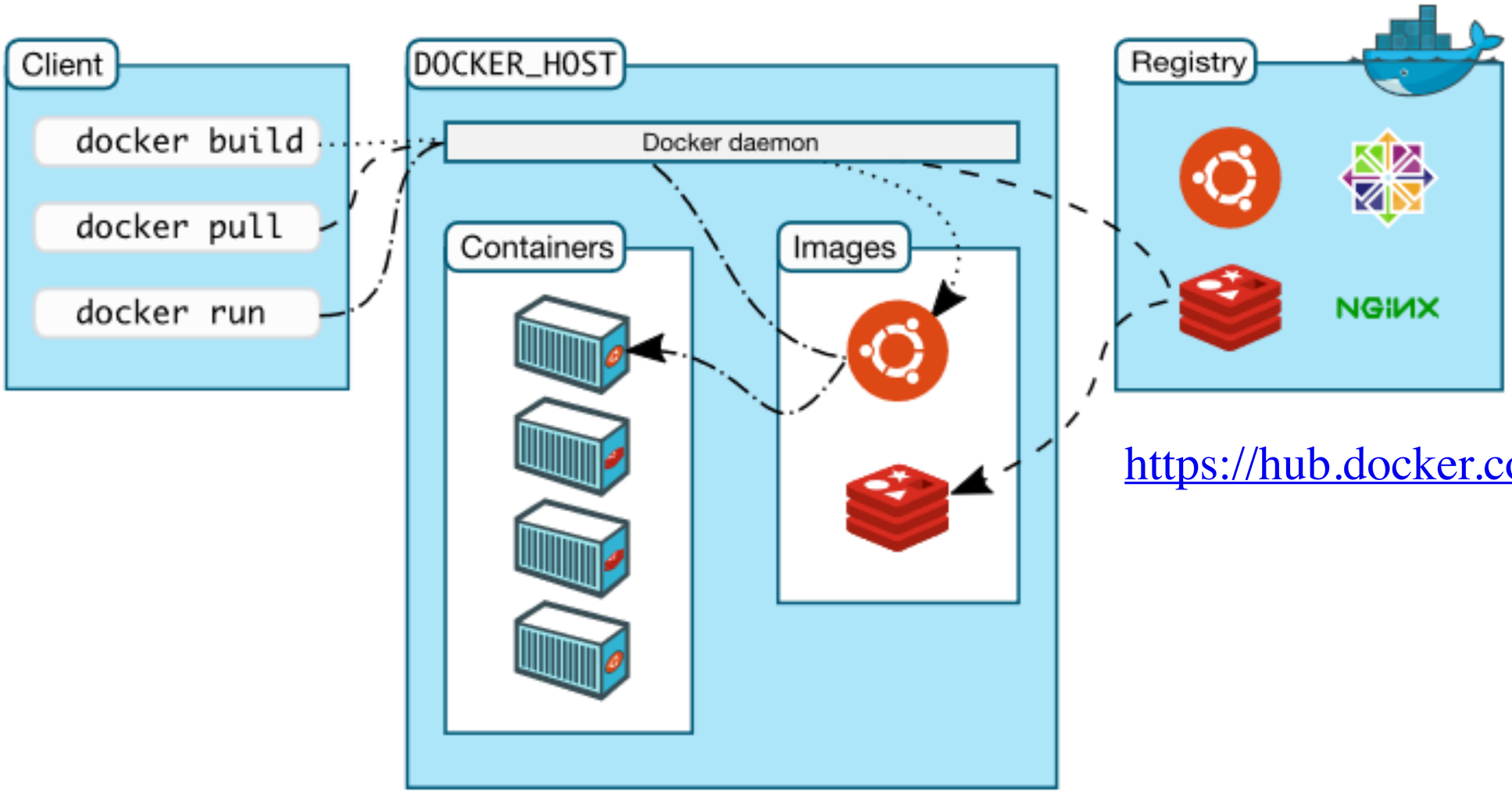
- **Server** que es un tipo de programa de larga duración llamado proceso daemon.

- **API Rest** que especifica las interfaces que los programas pueden usar para hablar con el demonio e indicarle qué hacer.

- **Client CLI** - utiliza la API REST de Docker para controlar o interactuar con el demonio Docker a través de scripts o comandos directos.



Docker - Arquitectura



https://hub.docker.com/

Docker - Imagen

Una **imagen** de Docker contiene todo lo necesario para **ejecutar una aplicación** como un **contenedor**.

Una imagen de Docker se crea a partir de una serie de capas. Cada capa representa una instrucción en el Dockerfile de la imagen. Cada capa, excepto la última, es de solo lectura

Incluye:

- Código
- Runtime
- Libreria
- Variables de Entorno
- Archivos de configuración

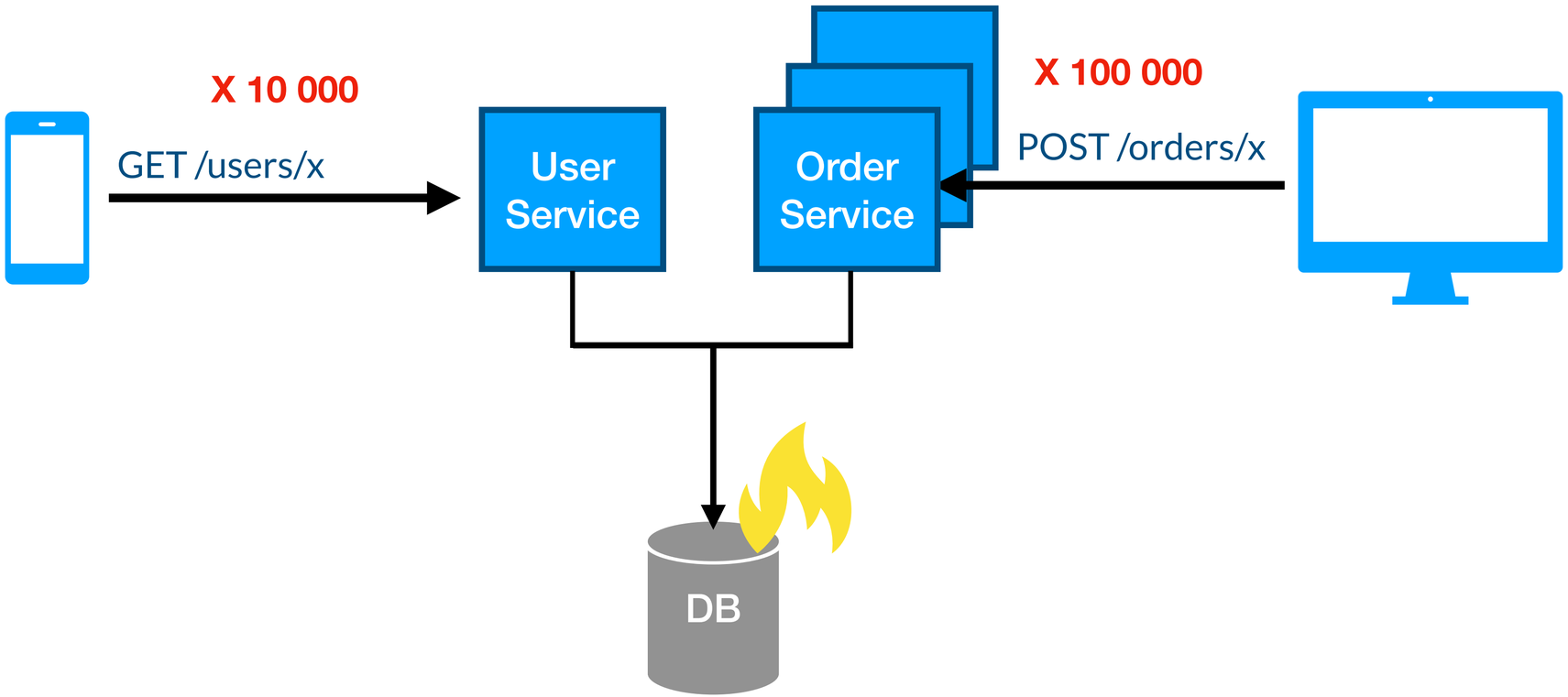
```
FROM openjdk:11-jre-slim
COPY /build/libs/product-service-1.0.0.jar /usr/apps/

WORKDIR /usr/apps
ENTRYPOINT ["java","-jar","product-service-1.0.0.jar"]
EXPOSE 8091
```

Docker - Container

Un **contenedor** Docker es una **instancia** en tiempo de ejecución de una **imagen**, que se ejecuta como un proceso discreto en la máquina **Host**. Tiene su propio proceso que corre de forma aislada al resto de procesos del sistema. Maneja su propio sistema usuarios, ficheros y red.

```
> docker run -it -p 80:80 nginx
```

Microservicios Monolitos



Docker -Install

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

1. Update the apt package index

```
$ sudo apt-get update
```

2. Install packages to allow apt to use a repository over HTTPS:

```
$ sudo apt-get install  apt-transport-https ca-certificates  curl \
      gnupg-agent software-properties-common
```

3. Add Docker’s official GPG key:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Docker -Install

4. Use the following command to set up the **stable** repository

```
 $ sudo add-apt-repository    "deb [arch=amd64] https://download.docker.com/linux/ubuntu    $
(lsb_release -cs) stable"
```

5. Update the apt package index.

```
    $ sudo apt-get update
```

6. Install the *latest version* of Docker Engine

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

7. Add User

```
    $ sudo usermod -aG docker $USER
```

Docker Comandos

- Descargar un imagen

```
$ docker pull nginx
```

- Listas las images

```
$ docker image ls / docker images
```

- Correr un contenedor

```
$ docker run -it -p 80:80 nginx
```

- Listas los contendores en ejecución

```
$ docker ps
```

Dockerfile

- Si vamos a crear nuestras propias imágenes, necesitamos un archivo llamado **DockerFile** que es la "receta" que utiliza Docker para crear imágenes.

- Compilamos nuestra imagen

```
$ docker build  -t edumar111/product-service:1.0.0 -f Dockerfile .
```

- Ejecutamos un contenedor de la imagen

```
$ docker run  -itd  --name product-service -p 8091:8091 edumar111/product-service:1.0.0
```

- Ver los de nuestro contenedor

```
$ docker logs <id-container>
```

Docker en producción

En entornos Cluster, Docker no provee :

- Despliegue de contenedores.
- Service discovery.
- Balance de Carga.
- Configuración de Red.
- Autoescalamiento.
- Respuesta a Fallos.

Orquestador

La misión principal de un orquestador es, dado un contenedor a ejecutar, seleccionar un nodo donde ejecutarlo de una manera segura, escalaba y tolerante a fallos.

Funcionalidades:

- Service Discovery
- Balanceo de Carga
- Configuración de red
- Escalabilidad
- Logging y Monitoreo
- Respuesta a Fallos

Orquestadores en el mercado

- **Docker Swarm** Es un orquestador muy potente y sencillo de usar.

- **Mesosphere** orquestador más potente y que funciona a mayor escala de los últimos años, pero la industria parece que se ha puesto de acuerdo en adoptar Kubernetes como el orquestador estándar, y está quedando en desuso.

- **ECS y AWS Fargate** En este caso no necesitamos de tener un cluster pre-creado para lanzar contenedores, si no que nos ofrecen una API donde podemos lanzar contenedores bajo demanda.

- **Kubernetes** se ha convertido en el estándar *de facto* para la orquestación de contenedores.

Kubernetes

Nació como un proyecto interno de Google a mediados de 2014. Es una evolución de Borg, el orquestador que utiliza Google en producción, pero adaptado a usar contenedores Docker. Por tanto, es un orquestador que nace con muchos años de experiencia de Google ejecutando contenedores en producción, es muy potente y escala muy bien.

Kubernetes es un orquestador de contenedores, y que se ha convertido en el estándar de facto para desplegar aplicaciones cloud.

Kubernetes - Características

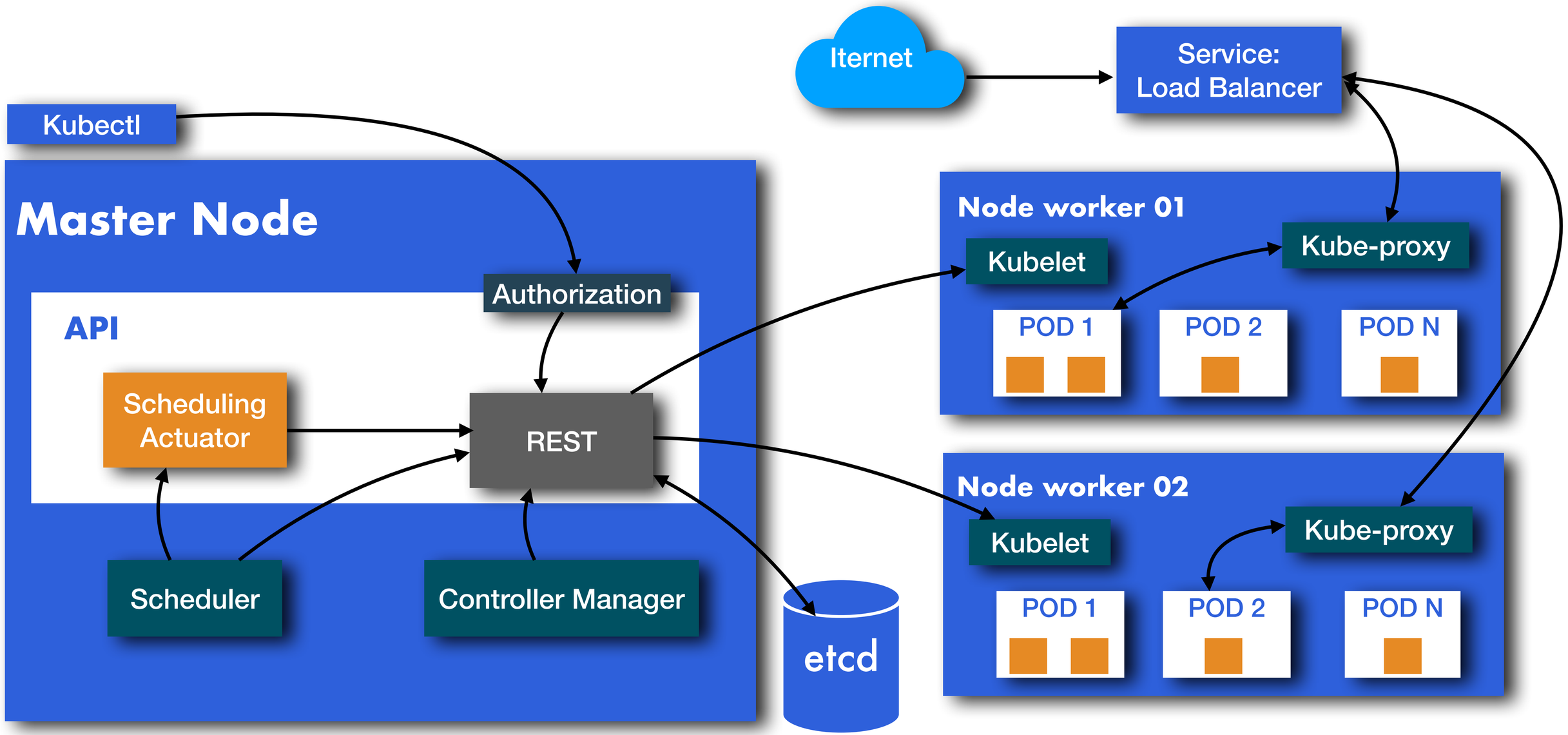
- Lanzar contenedores en un cluster de máquinas.
- Gestiona múltiples contenedores por máquina
- Declarativo (Declaras el estado de la aplicación)
- Escala los contenedores
- Corre en Cloud público y privado (multi-cloud)

Kubernetes - Arquitectura

Kubernetes divide los nodos de un clúster en master nodes y en worker nodes. Los master nodes son la capa de control y en principio no ejecutan contenedores de usuario. Los worker nodes son los responsables de ejecutar los contenedores de usuario.

Kubernetes es la tecnología y software que puedes ejecutar por ti mismo desde tu computadora, en tus propios servidores , en la nube o la combinación de estos si lo deseas.

Kubernetes - Arquitectura



Kubernetes - Pods

Un **Pod** se compone de uno o más contenedores, que deben estar estrechamente relacionados, y es una abstracción de una máquina virtual donde corren esos contenedores. Quiere decir esto, por ejemplo, que los contenedores corriendo en el mismo Pod comparten el mismo stack de red (son accesibles entre ellos usando *localhost*). También tienen acceso a los mismos volúmenes e incluso a memoria compartida.

La unidad de despliegue de kubernetes

Kubernetes - Services

Un Service se asocia a un conjunto de Pods. Cada Service tiene un endpoint de acceso fijo que se corresponde con su nombre (aunque también podríamos acceder a él vía un Load Balancer externo o un ingress). Por tanto, cada vez que accedamos por DNS a nombre de un servicio, el servicio redirigirá la petición a uno de los Pods a los que está asociado, ofreciendo un mecanismo eficaz de **service discovery** y de balanceo de carga.

Los Pods se crean y destruyen constantemente en un clúster de Kubernetes, por lo que el acceso entre aplicaciones no se puede basar en la IP de los Pods. De alguna manera, es necesario tener un *endpoint* permanente que de acceso a un conjunto de Pods.

Kubernetes - Services

Los **ServiceTypes** de Kubernetes le permiten especificar qué tipo de servicio desea. El valor predeterminado es **ClusterIP**. Los valores de tipo y sus comportamientos son:

- **ClusterIP,** Expone el Servicio en una IP interna del clúster. Al elegir este valor, solo se puede acceder al Servicio desde el clúster.

- **NodePort** Expone el servicio en la IP de cada nodo en un puerto estático (el NodePort). Se crea automáticamente un servicio ClusterIP, hacia el cual se enruta el servicio NodePort. Podrá ponerse en contacto con el servicio NodePort, desde fuera del clúster, solicitando <NodeIP>: <NodePort>.

- **LoadBalancer** Expone el Servicio externamente usando el balanceador de carga de un proveedor de la nube. Los servicios NodePort y ClusterIP, a los que se enrutan las rutas externas del equilibrador de carga, se crean automáticamente.

Kubernetes - Recursos

Pods.
Replication Controller
Deployment
Services
ConfigMap
Volumes
Jobs
Secret
Label
Namespaces
Ingress

Kubernetes - Recursos

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-service
  labels:
    app-name: product-service
spec:
  replicas: 2
  selector:
    matchLabels:
      app-code: product-service
  template:
    metadata:
      labels:
        app-code: product-service
    spec:
      containers:
        - name: product-service
          image:  edumar111/product-service:1.0.0
          imagePullPolicy: Always
          resources:
            requests:
              memory: "512Mi"
            limits:
              memory: "1024Mi"
          ports:
            - containerPort: 8091
              name: http
              protocol: TCP
          livenessProbe:
            tcpSocket:
              port: 8091
            initialDelaySeconds: 90
            periodSeconds: 30
            timeoutSeconds: 3
            failureThreshold: 3
```

```yaml
kind: Service
apiVersion: v1
metadata:
  name: product-service-svc
  labels:
    svc-name: product-service-svc
spec:
  type: NodePort
  selector:
    app-name: product-service
  ports:
  - protocol: TCP
    port: 8091
    targetPort: 8091
    name: http
  externalIPs:
    - 192.168.33.10
```

Kubernetes - Install

Instalando Minikube en Ubuntu.

https://minikube.sigs.k8s.io/docs/start/

```
  $ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube_1.9.0-0_amd64.deb \
 && sudo dpkg -i minikube_1.9.0-0_amd64.deb
```

Pre-requsitos

```
  $ sudo apt install conntrack
```

- Inicializar minikube

```
  $ sudo minikube start  —vm-driver=none
```

- Status de minikube

```
  $ sudo minikube status
```

Kubernetes - Install

Instalando kubectl.

https://kubernetes.io/docs/tasks/tools/install-kubectl/#install-kubectl-on-linux

```
$ curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.18.0/bin/linux/amd64/kubectl
```

```
$ chmod +x ./kubectl
$ sudo mv ./kubectl /usr/local/bin/kubectl
$ kubectl version --client
```

Docker & Kubernetes

Digital lab Academy

@digitallab_pe
/digitallab.academy
@digitallab.academy

www.digitallab.academy