Despliegues de sistemas de ML orientado a microservicios

FAMAF Facultad de Matemática, Astronomía, Física y Computación
UNC
Córdoba Technology Cluster
CCAD Centro de Computación de Alto Desempeño de la Universidad Nacional de Córdoba

Información útil

- Disertantes:
  - Mariano Garagiola: mgaragiola@unc.edu.ar
  - Matias Silva: mgs0113@famaf.unc.edu.ar
- Canal de slack: **#despliegues-ml**
- Links de interés.. A lo largo de la presentación.
- Días
  - Viernes de 18 a 22
  - Sábado de 10 a 14
- Método de evaluación
  - Entrega y aprobación de 2 trabajos práctico.

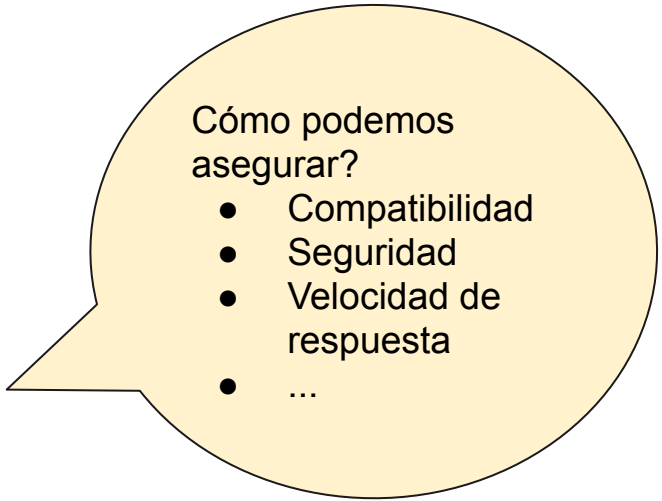
Objetivo y motivación

Motivación

- Tenemos nuestro modelo de ML listo para producción
  - Python3 + numpy, scikit-learn, tensorflow, etc.
- (Posiblemente) Deseamos correr el modelo en GPUs por motivos de eficiencia
- **Cliente 1** desea integrarlo a su aplicación en Android para dispositivos móviles
- **Cliente 2** desea integrarlo en una aplicación Web
- **Cliente 3** quiere distribuirlo como un instalador para máquinas con Windows con acceso a internet

Motivación

- Tenemos nuestro modelo de ML listo para producción
  - Python3 + numpy, scikit-learn, tensorflow, etc.
- (Posiblemente) Deseamos correr el modelo en GPUs por motivos de eficiencia
- **Cliente 1** desea integrarlo a su aplicación en Android para dispositivos móviles
- **Cliente 2** desea integrarlo en una aplicación Web
- **Cliente 3** quiere distribuirlo como un instalador para máquinas con Windows



Motivación

Motivación

En particular es este curso veremos un enfoque en una **Web API.** Es decir una API que pueda ser consultada a través de un protocolo HTTP. Para ello es necesario el **despliegue** y **publicación** en la red sobre un **servidor on-premise o cloud**.

ON PREMISE

- En servidores propios

+ Versatilidad
+ Control
- Mantenimiento
- Lentitud

CLOUD

- En algún proveedor de servicios en la nube.

+ Estabilidad
+ Rapidez
- Costos

amazon web services

Google Cloud Platform

Azure

Objetivo

Crear un entorno más amigable para poder ser utilizado productivamente por un usuario final. Es objetivo de este curso aprender y aplicar algunas técnicas que nos permitan **desplegar nuestro propio modelo de ML en un entorno web** y dejarlo así disponible tanto en plataformas cloud como on-premise para ser utilizado por un cliente externo.

Roadmap

Temas

- Introduccion web, HTTP, REST y alternativas
- Microservicios vs Monolitos
- Herramientas para portabilidad de código.
  - Máquinas virtuales
  - Docker
  - Kubernetes

- Actividad docker, swarm, kubernetes

Temas

- Balanceadores de carga y DNS resolver
- Comunicación entre servicios.
  - Redis
  - ~~RabbitMQ~~
  - Kafka

- Actividad: kafka cluster

Temas

- Frameworks web para desarrollo de APIs
  - Stacks
  - Datos y ORM
  - Autenticacion y ~~autorizacion~~
- Ingeniería de software aplicada a APIs de modelos de ML
  - Diseño
  - Retroalimentación para mejorar precisión

- Actividad: ML API

Temas

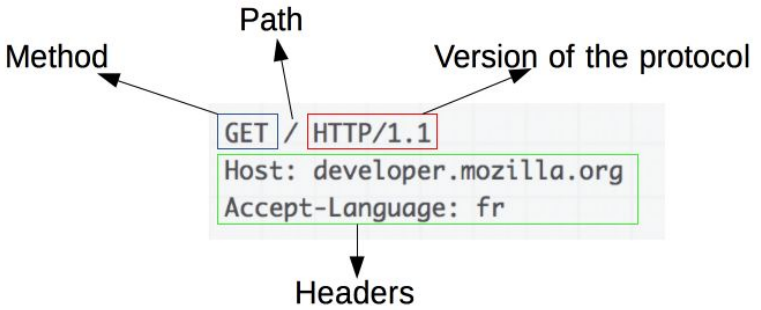
- Servicios cloud
  - Amazon Web Services
  - ~~GCloud~~
  - ~~Azure~~
- Testing
  - Tipos de testing
  - Testing de carga para APIs
- Monitoreo y análisis de performance
- Actividad: Monitoreo de performance

Introduccion web HTTP, REST y alternativas

Web APIs protocolos

Sabemos que para comunicarnos a través de un protocolo HTTP debemos hacer uso de sus tres componentes principales.

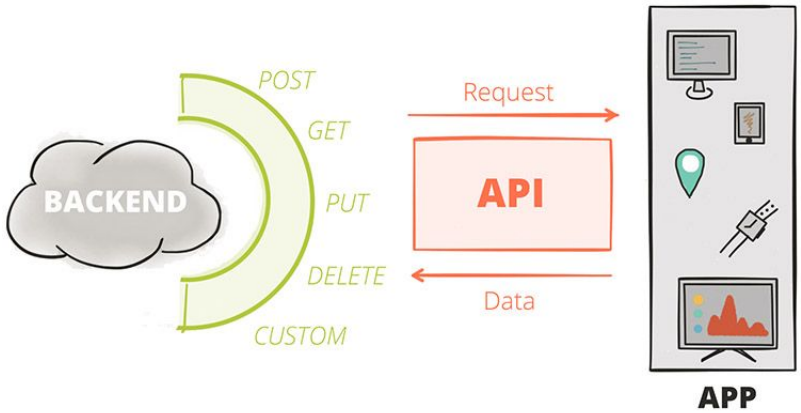
- Verbo (GET, POST, PUT, etc)
- URL (eg: http://my.api.com/instances)
- Payload con encabezados



Web APIs protocolos

Existen diversos protocolos para el desarrollo de web APIs aunque el más utilizado hoy en dia es REST el cual consta de 6 principios básicos.

- Cliente-Servidor
- Stateless
- Cacheable
- Interfaz uniforme
- Sistema por capas
- Código bajo demanda(optional)



Web APIs protocolos (alternativas)

Web APIs protocolos (alternativas)

comunicación XML similar a REST

Único endpoint mayor versatilidad. Dumb API / Smart Frontend

Enfoque basado en microservicios.

Microservicios

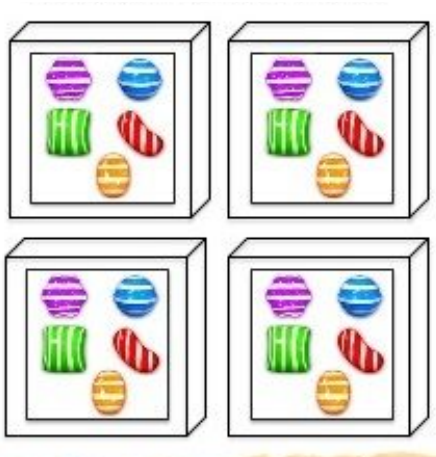
Sea el caso de un dispensador de grageas donde la demanda de las grageas verdes es muy superior al resto. Es decir el público consume mucho más las grageas verdes que las rojas.

Por otro lado nuestro proveedor nos vende las grageas para reposición pero en pack. Es decir no podremos comprar solo grageas verdes sino que debemos comprar un pack con un surtido de los 5 tipos de grageas.

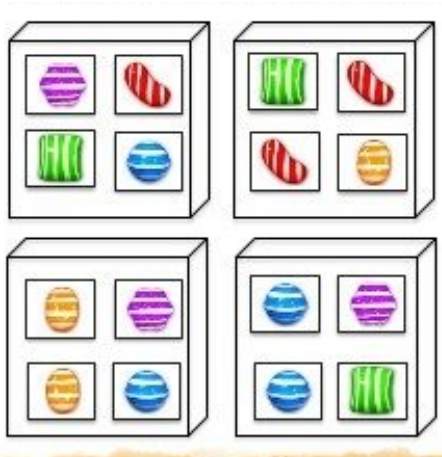
Aquí vemos un alto acoplamiento. Es decir vamos a tener un problema cuando quisiéramos reponer nuestro faltante de grageas verdes pues esto supone una compra innecesaria de grageas de los demás colores.

Microservicios

Arquitectura monolítica

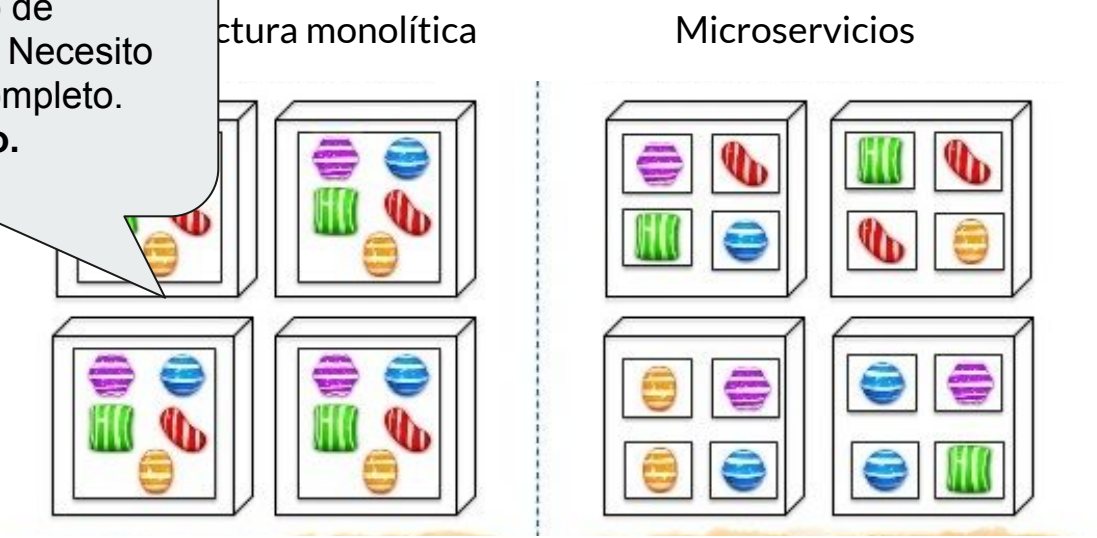


Microservicios



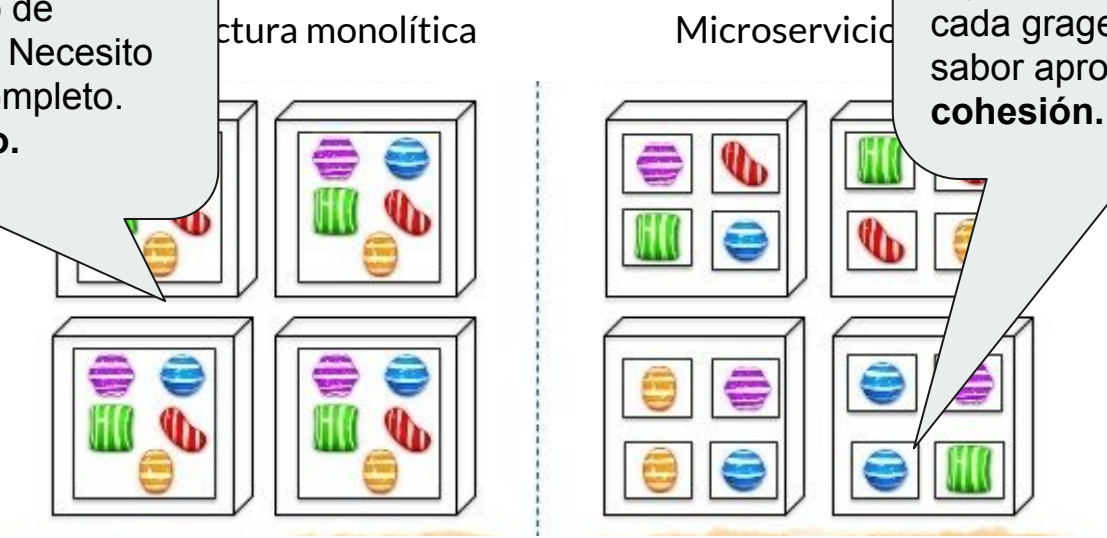
Microservicios

Si en un kiosko nuevo quiero vender un solo tipo de grageas no puedo. Necesito comprar el pack completo. **Alto acoplamiento.**



Microservicios

Si en un kiosko nuevo quiero vender un solo tipo de grageas no puedo. Necesito comprar el pack completo. **Alto acoplamiento.**

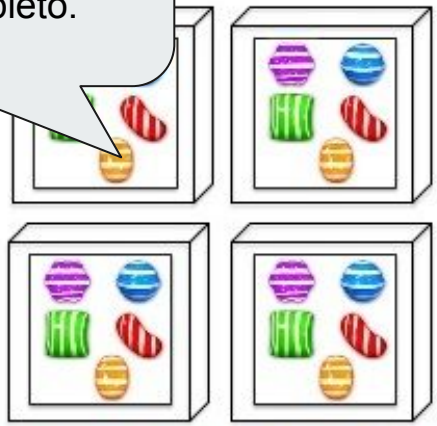


Manejo individual permite una **mayor versatilidad** en la distribución. Los diversos ingredientes se unen en cada gragea para lograr el sabor apropiado. **Alta cohesión.**

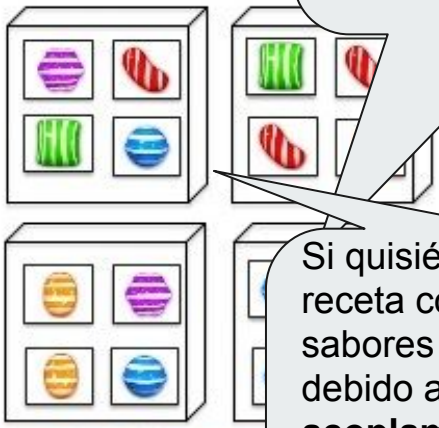
Microservicios

Si en un kiosco nuevo quiero vender un solo tipo de grageas no puedo. Necesito comprar el pack completo. **Alto acoplamiento.**

ctura monolítica



Microservici



Manejo individual permite una **mayor versatilidad** en la distribución. Los diversos ingredientes se unen en cada gragea para lograr el sabor apropiado. **Alta cohesión.**

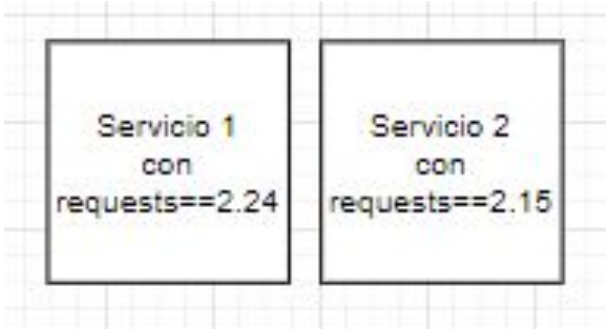
Si quisiéramos hacer una receta con grageas de varios sabores ya no sería tan fácil debido a su **bajo acoplamiento.** Deberíamos conseguir las grageas individuales en cada kiosco por separado (**comunicación**).

Portabilidad de código

Portabilidad de código

Dada una arquitectura de microservicios al desplegar más de un servicio en un mismo host podemos encontrarnos con problemas de dependencias requeridas en distintas versiones como en este caso la dependencia requests.

Para ello veremos algunas herramientas de portabilidad de código que no solo resuelven este problema sino que también ayudarán en la agilización del despliegue.



Portabilidad de código

Dada una arquitectura de microservicio
desplegar mas de un servicio en un mi
podemos encontrarnos con proble as d
dependencias requeridas en distintas versiones
como en este caso la dependencia requests.

Para ello veremos algunas herramientas de portabilidad de codigo que no solo resuelven este problema sino que tambien ayudaran en la agilizacion del despliegue.

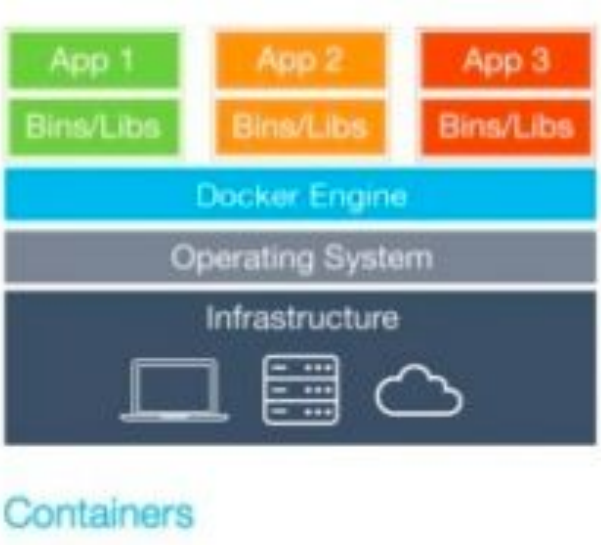
Si bien los entornos virtuales de python son una buena práctica para desarrollo no son suficiente pues solo servirán para resolver las dependencias relacionadas con python.

Máquinas Virtuales

- La aparición de máquinas virtuales nos permite ahora independizar las apps no solo de lenguaje sino también a nivel sistema ya que replicamos por cada aplicación desplegada un sistema operativo aislado. De esta forma no hay solapamiento de dependencias aunque contamos con un overhead de recursos.

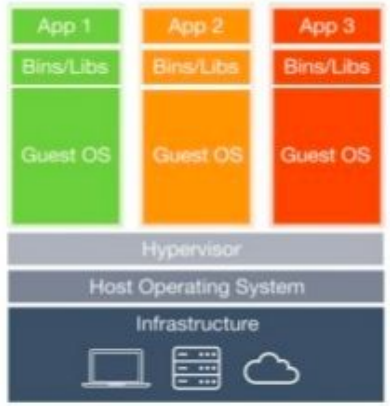
Contenedores

- En búsqueda de mayor portabilidad mediante este mecanismo el sistema operativo no se replica. Es decir **se reutiliza el kernel del sistema** en cada contenedor instanciado evitando así sobrecarga por encapsulamiento.



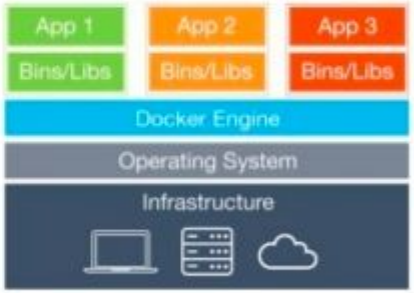
VM's vs containers

- Curva de aprendizaje más simple para ser utilizado. Herramientas más user-friendly
- Pueden ser el punto de partida para usar docker. No es excluyente su uso.



Virtual Machines

- Consumen menos recursos.
- Arranque mucho más rápido.
- Mayor portabilidad para ser utilizados en casi cualquier entorno sin dependencias.



Containers

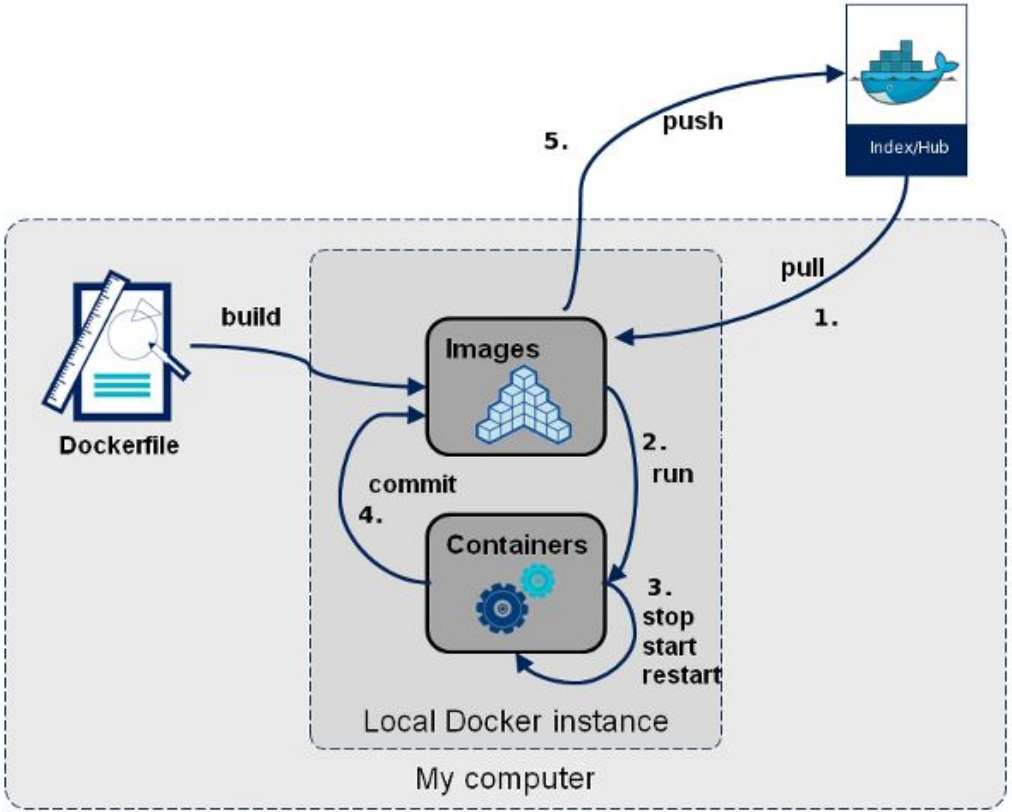
Portabilidad de código (definiciones)

Container: es la unidad básica de procesamiento de docker. La instancia de una imagen.

Imagen: Plantilla que tiene los binarios y librerías necesarias para correr la aplicación

Registro: Repositorio de imágenes. Puede ser de público acceso o privado. De aquí se obtienen y se suben las imágenes para su posterior uso.

Docker runtime



Dockerfile

```
FROM ubuntu ← alpine
MAINTAINER Kimbro Staken ← LABEL key=value
USER ${user:some_user}← Default values ARG user=some-user
ARG user
USER $user
RUN apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
RUN echo "deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen" | tee -a ..
RUN apt-get update
RUN apt-get -y install apt-utils
RUN apt-get -y install mongodb-10gen
COPY mongodb.conf /etc/mongodb.conf ← ADD
CMD ["/usr/bin/mongod", "--config", "/etc/mongodb.conf"] ← ENTRYPOINT
```

Orquestadores: docker-compose

```
docker run -d \
      -p 8080:5000 \
      -v $(pwd)/flask-app/code:/code \
      -e redis-ip=redis \
      flask-app

docker run -d redis:alpine
```

```
version: '3'
services:
  web:
    image: flask-app
    build: ./flask-app
    ports:
     - "8080:5000"
    volumes:
     - ./flask-app/code:/code

    environment:
     - redis-ip=redis
  redis:
    image: "redis:alpine"
```

Orquestadores: docker-compose

Como poner en marcha un servicio ?

- `docker-compose up -d <NAME OF SERVICE>`

Como escalar un servicio ?

- `docker-compose scale <NAME_OF_SERVICE>=<AMOUNT_OF_REPLICAS>`

Orquestadores: docker-compose

Como poner en marcha un servicio ?

- `docker-compose up -d <NAME OF SERV`

Como escalar un servicio

- `docker-compose sc <NAME_OF_SERVICE>=<AMOUNT_OF_REPLICAS>`

Es probable que en este punto al escalar nuestros servicios nuestro host quede “chico”. Es por eso que existen orquestadores multihost permitiéndonos funcionar en modo **cluster** y aprovechar así el potencial de todo nuestro hardware interconectado.

Portabilidad de código (definiciones cont.)

Container - **Imagen** - **Registro**

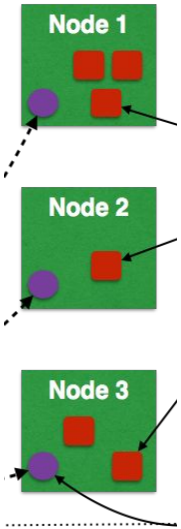
Nodo: Unidad básica de operaciones de un cluster. Suele ser un host de la red.

- **Worker**: Nodo esclavo con funcionalidad reducida
- **Master**: Nodo líder. Todos reportan a el y este tiene acceso a todas las operaciones del cluster.

Cluster: Conjunto de nodos que trabajan interconectados con mismo propósito

Orquestadores multihost: docker swarm

Multihost significa que tendremos las aplicaciones distribuidas a lo largo de nuestros nodos comunicadas como si fueran un mismo hardware. Esto nos permite hacer mayor aprovechamiento de los recursos y no necesitar supercomputo.

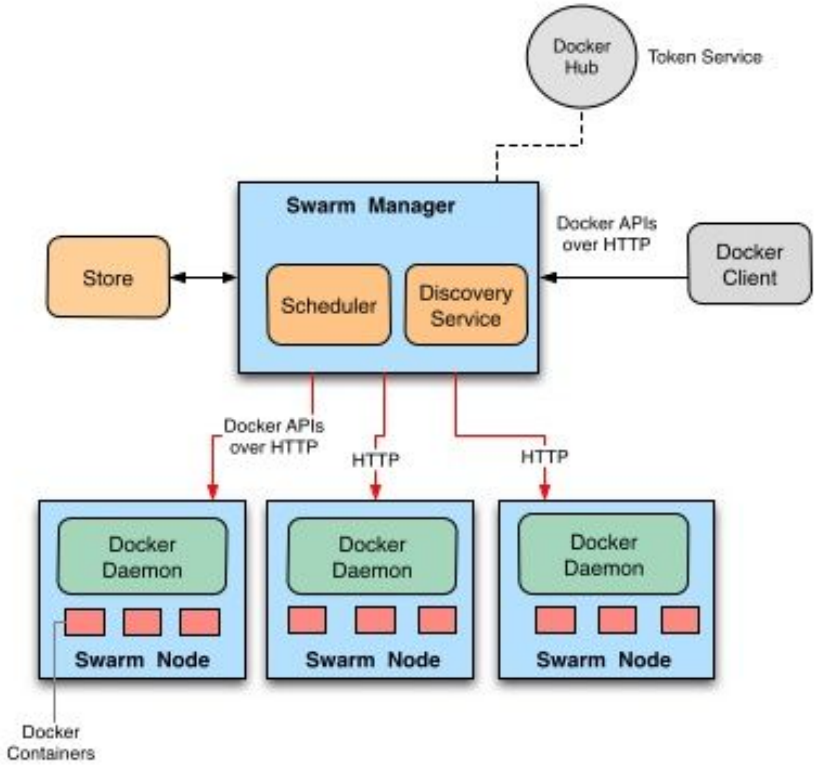


- model1ML
- model1ML
- model2ML

- model2ML

- API
- LoadBalancer
- model2ML

Orquestadores multihost: docker swarm



Portabilidad de código (definiciones cont.)

Container - **Image** - **Registry** - **Node** - **Cluster**

Pod: Unidad básica dentro de kubernetes es la instanciación de una o mas imagenes.

Kubelet: agente dentro del nodo que se encarga de gestionar los pods y monitorea los recursos

Kube Proxy: agente del nodo que maneja las operaciones de red

Deployment: instancia de un despliegue. Cada despliegue tendrá la posibilidad de ser reiniciado, actualizado, degradadado y monitoreado.

Service: Un servicio es una unidad logica que agrupa un conjunto de pods y políticas que determinan su forma de acceso y aseguran el comportamiento esperado haciendo uso de replicación y reinicio.

Secret: unidad de información que permite compartir informacion entre los pods de manera segura

Orquestadores: Kubernetes

Kubernetes es un orquestador multihost que **requiere un cluster de nodos** para trabajar de al menos un nodo activo. A su vez **usa un container-runtime** para la manipulacion de los contenedores ajeno a su implementacion.

Aunque muchas veces los servicios cloud nos ofrecen esto como un servicio como manager para la creación de nuestro cluster *on-premise* podemos encontrar::

- **minikube**
- docker-desktop feature

kubernetes

Como runtime los mas populares seran:

- **docker**
- containerd
- CRI-o

A su vez para la orquestacion kubernetes nos ofrece un CLI para la manipulacion de los despliegues:

- **kubectl**

Orquestadores: Kubernetes (kubectl)

Para entender como usar kubectl debemos entender que al igual que con docker tenemos un verbo aplicado a un sujeto. Es decir si quisieramos listar nuestros pods aplicariamos el verbo get al sujeto pods.

Verbos:

- get
- create
- logs
- describe

Sujetos:

- service
- deployment
- node
- pod
- all

Ejemplos

- kubectl get deployment
- kubectl describe pod
- kubectl get all

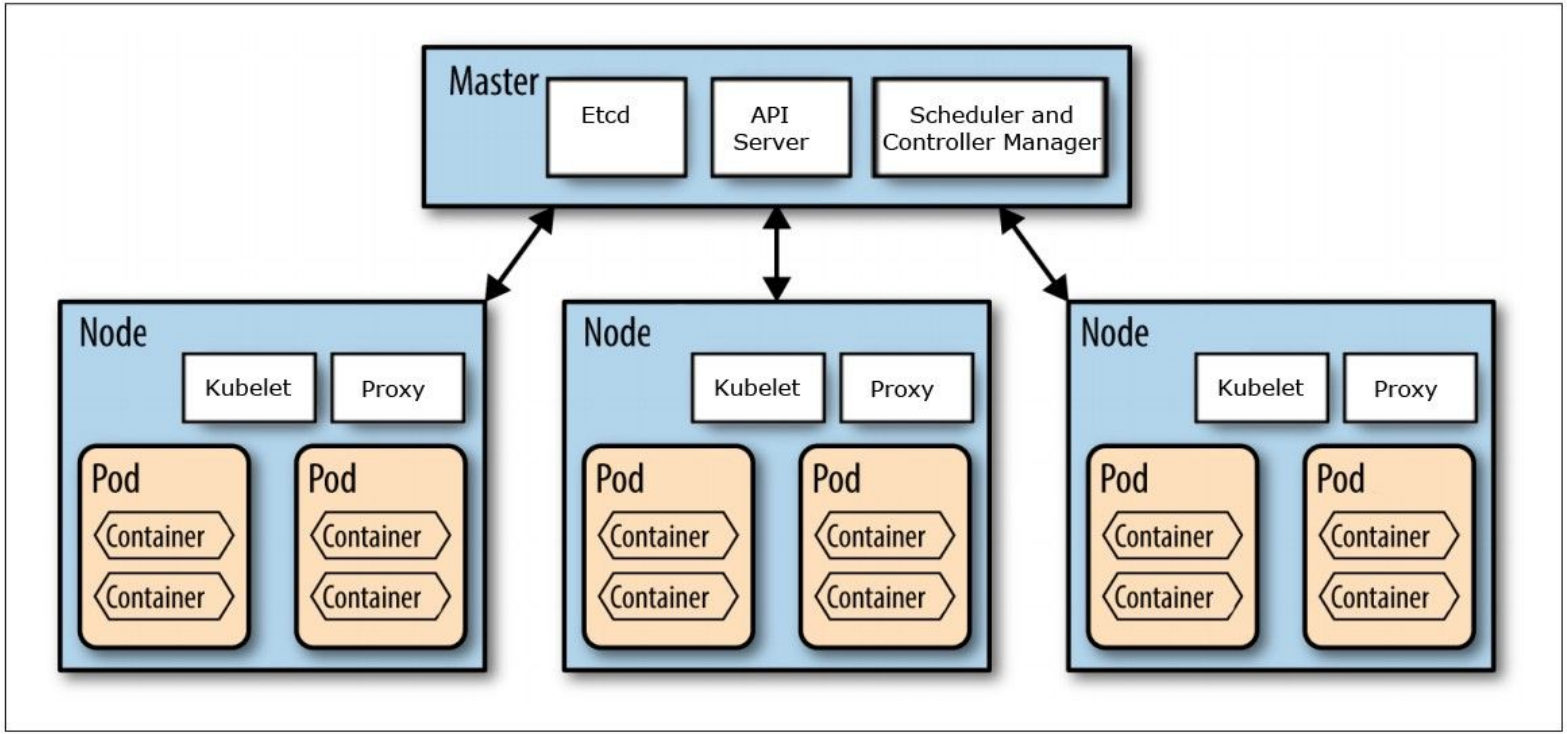
Orquestadores: Kubernetes

Sea entonces el caso de querer desplegar una aplicacion web ¿Que es necesario usar de todo lo que me ofrece kubernetes?

- Si o si debes crear un **deployment.** Esto te permitira trackear la evolucion de tus despliegues. Hacer un rollback o despliegues de prueba tipo A|B o canary.
- Si bien es necesario contar con la creacion de los **pods,** esta será parte de la tarea de gestión del deployment.
- Si es una aplicacion web vamos a querer publicarla probablemente para acceso externo. Es por ello que conectaremos nuestro deployment a un **service.**
- Es recomendable que si haces uso de contraseñas o claves compartidas lo hagas a traves de **secrets** y no en código plano..
- Tambien si necesitas de informacion persistente será vital el uso de **pv** (PersistenVolumes). Recuerda que los pods nacen y mueren tirando toda la informacion de su estado.
- Ademas podras necesitar **replicationcontrollers** o **ingress** (Path resolver)

kubernetes

Orquestadores: Kubernetes



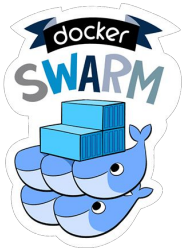
Orquestadores: Kubernetes

Sandbox para practicar

https://www.katacoda.com/courses/kubernetes

Resumen de orquestadores Kubernetes vs Swarm

- Menor curva de aprendizaje
- Admite docker-compose.yml
- Fácil configuración para entorno de desarrollo (cluster con 1 instancia)



Resumen de orquestadores Kubernetes vs Swarm

- Mayor integración y soporte en cloud
- Mayor versatilidad y customización con utilidades interesantes como
  - Autoscaling con load balancer integrado
  - Asignacion de recursos.
  - Rollback tool
- Posibilidad de integración con otros runtimes distintos de docker como *containerd*

kubernetes

Actividad 0:
docker-compose swarm & kubernetes

Actividad 0:

El objetivo de esta actividad es entender qué diferencia las 3 formas de orquestar y que ustedes mismos evalúen complejidad vs versatilidad de cada herramienta.

La actividad propone el despliegue de una aplicación django muy sencilla que permita dejar el registro de visitantes en una página web. Pero la actividad no involucra el desarrollo del código de la funcionalidad sino más bien el desarrollo de la forma en la cual la desplegaremos.

Para ello la propuesta es desplegar y publicar nuestra web de las siguientes 3 formas

- Con orquestador en un host de manera declarativa con docker-compose
- Con orquestacion de manera declarativa bajo docker swarm
- Con orquestacion de manera declarativa bajo kubernetes

Fork from: https://github.com/matisilva/django-guestbook

Hint:

- Instalar docker + kube = docker-desktop
  https://www.docker.com/products/docker-desktop
- Deberán hacer uso de servicios aislados para
  - cache con Redis
  - base de datos persistente con Postgres
- Como servidor de aplicacion podran usar gunicorn dentro de cada pod

- Resolucion: https://github.com/matisilva/kube-django-deploy

<FIN DE CLASE>