# Curso Python, Django, Docker, Kubernetes

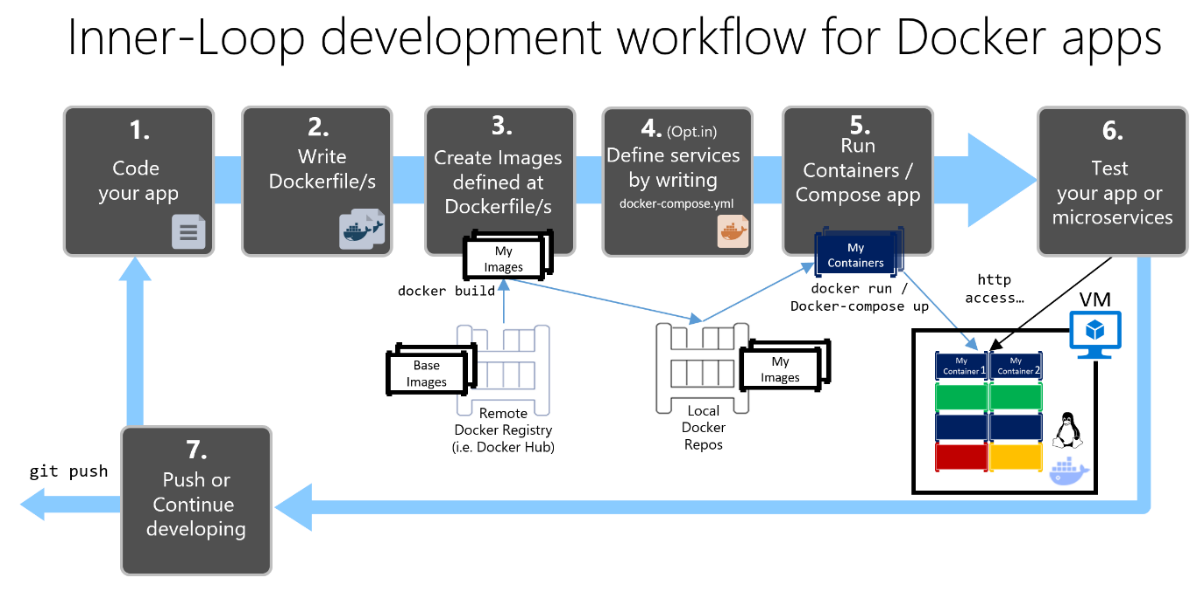
1. Copyleft Sergio A. Alonso

TODO: reconvertir ejemplo GCE a AWS, quitar kompose.

## Objetivos

* Acostumbrarse al ciclo de integración continua que propone Docker y Kubernetes
* Obtener soltura en Linux, MySQL y Python
* Portar fácilmente aplicaciones hacia la nube.

Flujo de trabajo



Ciclo de trabajo con Docker segun Microsoft

## Clase 1 - Introducción

### Entorno de trabajo

Se propone con el alumno trabajar lo más posible con su propio equipo.

A partir de la segunda clase, se trabajará con Docker. Los usuarios de Linux y de Mac no tendrán mas complicación que correr unos pocos comandos. Pero la instalación de Docker en Windows puede ser complicada debido a versiones, caracteristicas del procesador y otras situaciones. Se proveen instrucciones, y se espera que la segunda clase el alumno concurra con esta herramienta instalada. Si no llega a tiempo, debe solicitar una clase extra de apoyo.

En todos los casos, para todos los sistemas operativos, tambien hará falta que el alumno habilite las opciones de Virtualización en la BIOS de su computadora.

### Demostración - breve introducción

Docker puede trabajar en forma imperativa, y complementariamente, en forma declarativa

Fuente: https://docker-curriculum.com/#webapps-with-docker

#### Ejemplo de comandos imperativos

Bajamos una imagen desde Dockerhub, la instanciamos en memoria, y la ponemos a correr en el puerto 8888

docker run -p 8888:5000 --name contenedor\_michos prakhar1989/catnip

Para verlo corriendo, lanzamos un navegador en http://localhost:8888

Nota: si en lugar de Docker para Linux o de Docker para Windows se trabaja con Docker Toolbox (ver mas adelante), se debe obtener la ip interna donde está corriendo el contenedor, mediante el programa lazydocker, o con el comando

docker inspect contenedor\_michos | grep IPAddress  
  
Ejemplo: "IPAddress": "172.17.0.3"

Ahora si, lanzar un navegador en http://172.17.0.3:5000

#### Ejemplo de personalización en forma declarativa

Supongamos que queremos modificar algo de este imagen que obtuvimos en Dockerhub. Primero debemos obtener su Dockerfile:

cd /tmp  
git clone --depth=1 https://github.com/prakhar1989/docker-curriculum  
cd docker-curriculum/flask-app

* Como desarrolladores, modificamos el app.py de acuerdo a nuevas necesidades, si las hubiera, requirements.txt, templates/index.html, etc
* Como devops, modificamos el Dockefile a antojo.

Creamos una nueva, y nuestra, imagen local

docker build . -f Dockerfile -t mi-imagen-catnip

Notar la diferencia entre la imagen que bajamos de Dockerhub, y la que hicimos localmente

$ docker images | grep catnip  
  
mi-imagen-catnip latest e984937991cb 29 minutes ago 926MB  
prakhar1989/catnip latest 34ca2beec464 2 months ago 699MB

Borramos el contenedor en memoria “contenedor\_michos”

docker stop contenedor\_michos  
docker rm contenedor\_michos

Instanciamos nuestra propia imagen

docker run -p 8888:5000 --name contenedor\_michos mi-imagen-catnip

Por curiosidad, intervenimos el contenedor corriendo, desde otra terminal:

Primero listamos los contenedores corriendo:

$ docker ps | grep catnip  
d3918dfe2e32 mi-imagen-catnip "python ./app.py" 58 seconds ago Up 56 seconds 0.0.0.0:8888->5000/tcp contenedor\_michos

Ahora entramos con

$ docker exec -it contenedor\_michos sh  
  
# ls -l  
total 20  
-rw-r--r-- 1 root root 299 Feb 18 15:34 Dockerfile  
-rw-r--r-- 1 root root 199 Feb 18 15:34 Dockerrun.aws.json  
-rw-r--r-- 1 root root 1996 Feb 18 15:34 app.py  
-rw-r--r-- 1 root root 11 Feb 18 15:34 requirements.txt  
drwxr-xr-x 2 root root 4096 Feb 18 15:36 templates  
  
# pwd  
/usr/src/app

#### Docker en Windows

Los usuarios de Windows Home no podrán instalar Docker for Windows Desktop edition, pues este requiere Hyper-V virtualization, como Windows Professional y Enterprise editions. Usuarios de otros Windows no oficiales de tipo MiniOS, u “All in One”, probablemente tampoco terminarán correctamente la instalación, o Docker se negará a arrancar.

Los usuarios de Windows Home users deben instalar Docker Toolbox, que usa VirtualBox en su lugar: https://docs.docker.com/toolbox/toolbox\_install\_windows/ - release downloads available here: https://github.com/docker/toolbox/releases - Toolbox instalará todo lo necesario, incluyendo VirtualBox.

Una vez que Docker for Windows o Docker Toolbox terminen de instalarse, el alumno debe probar lanzar un pequeño contenedor desde la línea de comandos:

docker run hello-world

Esto debería arrojar, tras terminar, un mensaje **hello-world**

Una importante diferencia entre Docker Desktop for Windows vs. Docker Toolbox es que ya no podrá usar localhost. En su lugar, deberá acceder a su maquina mediante la dirección 192.168.99.100

#### Docker en el aula: sesión de Linux

En caso que la instalación en Windows se complique mucho, se ofrece incluida en el curso, una sesión en Linux en server propio, con el cual trabajar. Esta sesión conviene personalizarla para hacer mas amena la instrucción.

Los DevOps trabajan la mayor parte del tiempo en la terminal. Por lo tanto conviene setear ésta acordemente. E incluso si el alumno ha logrado setear correctamente su entorno en Windows, se explicarán los pasos necesarios: los devops pasan una gran parte del día conectados remotos, y conviene de hacerse de un entorno amigable para poder trabajar rápidamente y sin errores. Por el camino, hay una gran cantidad de meta conocimiento o de “residuo cognitivo” necesario para desempeñarse adentro de los contenedores con Linux.

### Manejador de sesiones en consola

Se trata de gestores de sesiones, que conviene instalar con apt-get, yum, pacman, etc, y aprender a usarlos en sus páginas oficiales: Screen, Tmux o Byobu.

Para los ejemplos en clase, se trata de usar byobu.

#### Shell

Se propone por su similitud con Mac y por sus funciones avanzados de instrospección de comandos, al shell Zsh por sobre el clásico Bash que viene por defecto con Linux. Instrucciones generales para setear una maquina con Ubuntu:

sudo apt-get install zsh git rake curl

Cambiar “para siempre” de Bash a Zsh

$ chsh  
Contraseña:  
Cambiando la consola de acceso para usuario  
Introduzca el nuevo valor, o presione INTRO para el predeterminado  
Consola de acceso [/bin/bash]: /usr/bin/zsh

Todavía hace falta “enchular” a Zsh. Le bajamos el instalador del proyecto

sh -c "$(wget https://raw.github.com/robbyrussell/oh-my-zsh/master/tools/install.sh -O -)"

Una vez instalado conviene ademas agregar algún tema como Powerlevel 10k

git clone https://github.com/romkatv/powerlevel10k.git $ZSH\_CUSTOM/themes/powerlevel10k

Ahora sí, alteramos las siguientes variables de ~/.zshrc

plugins=(git nvm node kubectl django python pip docker docker-compose)  
  
ZSH\_THEME="powerlevel10k/powerlevel10k"

Para probar los cambios, salimos de la sesión con exit y volvemos a entrar.

#### Vim como IDE

Mientras que en Windows se pueden instalar toda clase de magnificos editores, una gran parte del tiempo se pasa conectado a sesiones “shell” remotas, con Linux. Si bien todas las distribuciones de Linux incluyen Vi, se recomienda instalar el último “Vim” posible. La m de “vim” mnemotecnicamente la vamos a considerar para “mejorado”.

sudo apt-get install vim

Se puede “enchular” también a Vim, agregando el paquete [Janus](https://github.com/carlhuda/janus):

curl -L https://bit.ly/janus-bootstrap | bash

Conviene ademas definir una Leader Key. En ~/.vimrc agregar

let mapleader = ","

La Leader key sirve para lanzar ciertas funciones, como el explorador de archivos “Nerd Tree”, con , (coma) seguida de n. Puede ver una lista de teclas interesantes en [Taza Vim](http://www.eim.esc.edu.ar/bunker.org.ar/incubadora.varios/TazaVimSoloJpeg/TazaVim11.jpg)

## Clase 1 - Codificar una primer versión estable del proyecto

Crearemos un pequeño programa en Python que consulta una base de datos MySQL llamada “inventario”. El propósito es mostrar como a pesar de funcionar bien, es sumamente incómoda de portar a otros sistemas operativos. Docker por supuesto, vendrá luego a salvarnos.

### 1.1. MySQL

[Instalar MySql en Ubuntu 18:](https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-18-04) Resumen de comandos:

sudo apt-get install mysql-server  
sudo mysql\_secure\_installation

En su primera sesión con MySQL, fabrique una base de datos, y asigne permisos para su uso:

$ mysql -u root -p  
mysql> CREATE DATABASE inventario;  
mysql> GRANT ALL PRIVILEGES ON inventario.\* TO 'inventario\_user'@'localhost' IDENTIFIED BY 'MiViejaMula' WITH GRANT OPTION;  
mysql> FLUSH PRIVILEGES;  
mysql> exit;

Inyecte ahora el archivo inventario.sql en su nueva base

mysql -u root -p inventario < inventario.sql

Pruebe que puede entrar a su nueva base, con el usuario creado a tal fin:

mysql -u inventario\_user -p inventario  
mysql> SELECT \* FROM amigos;

### Clase 1.2 - Instalar Python (y Django) con Conda

Si bien Python ya viene instalado en todos los sistemas operativos Linux y Mac, éste se reserva para el uso del propio sistema operativo, y particularmente para ser usado por el root. Tampoco conviene llenar el sistema operativo con librerías cruzadas entre Python 2 y Python 3. En su lugar conviene instalar un manejador de entornos o “VirtualEnv”. El problema entonces es que tanto Windows, como Mac como Linux tienen formas distintas de acometer este paso. La mejor manera unificada de acometerlo es empleando una versión simple de Anaconda, llamada miniconda.

Sírvase acceder al excelente material documentativo creado en este sitio. No hace falta instalar todavía Django: basta con la parte de Python: [Instalar Python y Django](https://hcosta.github.io/instalardjango.com/). Resúmen de comandos (válido para Arch, sin necesidad de yay):

Como usuario, para que los binarios de conda se instalen en el home, carpeta ~/miniconda

wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86\_64.sh  
chmod u+x Miniconda3-latest-Linux-x86\_64.sh  
./Miniconda3-latest-Linux-x86\_64.sh

Tras estar seguros que estas líneas figuran al final ~/.zshrc y en ~/.bashrc, salimos de la terminal con exit, o reiniciamos el equipo, y volvemos a entrar. Tras lo cual, el comando “conda” debería respondernos, para por ejemplo, setear un primer entorno virtual:

python -V (para ver la versión de Python del sistema operativo que "no" queremos usar)  
  
which python (para ver la ruta que no queremos usar, ejemplo, /usr/bin/python2.7)  
  
conda create -n py36 python=3.6  
  
conda info --envs (para ver este y otros environments que hubieramos creado)  
  
conda activate py36  
  
python -V (Python ahora debería sestar compilado por Anaconda Inc.)  
  
which python (la ruta debería apuntar ahora a nuestro propio home)

### Clase 1.3 - Instalar las librerías necesarias para que Python trabaje con MySQL

Retomamos proyecto. Se incluye a modo de ejemplo un archivo requirements.txt para pip

conda activate py36  
pip install -r requirements.txt

### Clase 1.4 - Probar el programa

El modo debug -m pudb es optativo, solo para tener un mejor control del programa.

python -m pudb inventario.py

## Clase 2 - Portar el programa a Docker

Para trabajar con Docker hay dos caminos iniciales: el imperativo y el declarativo.

El imperativo se encuentra en (demasiada) abundancia en cientos de tutoriales en Internet, y sirve para situaciones puntuales, para corregir o debuggear. Un novato abusará de esta modalidad por ejemplo creando un contenedor a la manera del entorno “py36” creado con Miniconda, de la siguiente manera

docker run -d --name py36 -ti python:3.6 /bin/bash  
docker exec -ti py36 /bin/bash

Tras en prompt conseguido, trabajará alli adentro cuanto le plazca, perderá su trabajo la primer vez al morir la instancia, adosará volumenes la siguiente vez, agregará puertos para comunicarlo con otros contenedores, usará lineas de comandos cada vez mas largas, y finalmente persistirá su trabajo con docker commit, docker export, etc.

La otra modalidad de trabajo es complementaria a la anterior, y se denomina declarativa. Se construye un archivo Dockerfile basado en las “recetas madre” presentes en Dockerhub. Luego se hace un docker build, run, y se interviene en forma imperativa cuando sea necesario.

Secuencia de ejemplo para portar un programa en Python

* Entrar a [Proyecto Python en DockerHub](https://hub.docker.com/_/python/) y escojer algun tag que coincida con la versión de Python empleada en nuestro programa.
* Buscar el Dockerfile y adaptarlo a nuestra necesidades. Dejo un Dockerfile de ejemplo en esta carpeta.
* Crear (build) la imagen local y correrla. El -f es a propósito para poder tener varios Dockerfile en la misma carpeta.

docker build -t python-inventario-imagen:0.1 . -f Dockerfile.python-simple  
  
docker run --rm --name python-inventario-instancia imagen-python-inventario:0.1

## Clase 3 - Portar los servicios dependientes con docker-compose

En este momento el contenedor con el programa en Python “sale” a la maquina real, o a la red real, a conectarse a una base de datos. Por razones pedagógicas y para ilustrar la portabilidad del conjunto, portaremos también a MySQL, con docker-compose.

Antes, hay que señalar el hecho que la alta carga de las bases de datos requiere a veces de que éstas no estén contenidas y ni siquiera virtualizadas. Tambien es la razón por la cual muchas veces se porta una capa intermedia de datos en el contenedor, del tipo que corren en RAM, como Redis, que no necesitan de bajar a disco todo el tiempo, en tanto que a MySQL, PostgreSQL o RDBMS similares se las deja en su propia VM, o tercerizadas en servicios específicos ofrecidos por AWS.

Tambien en este punto conviene habituarse a lanzar y a administrar el proyecto con el comando docker-compose en lugar del comando docker. Esto es porque ambos servicios se encuentran enlazados.

Recuerde ayudarse en estos pasos donde hay muchos contenedores corriendo, mediante algún GUI como lazydocker o portainer.

### Clase 4 - Compartir fuentes, dockerfiles y recetas vía GIT

Esta necesidad de “compartir el código” no solo tiene su origen en un ambiente con varios programadores. Es necesario para cuando el flujo de trabajo descripto al principio se hace más intenso, cuando se agregan ambientes de prueba, y especialmente cuando se automatiza mediante alguna herramienta de integración continua como Jenkins, CircleCI, Travis, etc.

Estas herramientas trabajan mejor con GIT, al configurarlas para detectar nuestras entregas (git push). Ellas realizan automaticamente los builds y testeos necesarios, envian las imagenes al registry, y ordenan a Kubernetes el despliegue final.

Opciones de servicio de GIT durante el cursado:

* Emplear Git existentes en clouds: Github, Gitlab
* Construir nuestro “propio servidor Git” en infraestructura propia: servidor git simple o servidor Gitlab-ce (Community Edition) instalado localmente. Este paso requiere de una clase extra, y se recomienda para empresas.

Pasos de ejemplo con Github

* Crear cuenta en Github
* Generar llaves, tanto de la maquina de desarrollo, como de la consola de GCE

ssh-keygen -t rsa -f ~/.ssh/id\_rsa -N '' (no poner contraseña: que se genere sola)  
cat ~/.ssh/id\_rsa.pub

Estas llaves, ambas subirlas a https://github.com/settings/keys

* Solo necesario en Github: hacerse un fork de este proyecto https//www.github.com/pancutan/Taller\_Python\_Conda\_MySQL\_Kubernetes
* En una carpeta vacía, clonar proyecto. Si se trata de Gitlab, el profesor solo debe invitar a los alumnos y éstos van por la url original. Es decir:

Con Github:

git clone git@github.com:USUARIO\_ALUMNO/Taller\_Python\_Conda\_MySQL\_Kubernetes.git

Con Gitlab (ejemplo):

git clone git@gitlab.com:pancutan/Taller\_Python\_Conda\_MySQL\_Kubernetes.git

Poner allí adentro las modificaciones de código, comitear, pushear, realizar pull request si en los apuntes del profesor algo debiera corregirse.

## Clase 5 - Pushear a un Registry

La necesidad de este paso previo a Kubernetes consiste en que este orquestador típico de producción no se dedica a hacer builds, como lo podría hacer Swarm vía docker-compose. En sus recetas o “manifiestos” debemos proveerle en su lugar las imagenes. Y para ello debemos subirlas a un Registry. Se puede usar al principio Dockerhub, que es gratis.

No obstante, tarde o temprano, aparecerán mas necesidades de privacidad y/o velocidad. Cuando se arribe a tal situación, hay dos opciones más:

* Crear un Registry en la infraestructura instalada. Es un tanto complejo, y se hace siguiendo cuidadosamente las instrucciones de https://docs.docker.com/registry/ y del archivo que dejo Registry/registry.md
* Debido a la complejidad que tiene este tema, consultar al profesor si se desea contratar una clase especialmente para tratarlo. Se recomienda para empresas.
* Contratar el registry de un cloud como Amazon ECR (Elastic Container Registry), o GCR (Google Container Registry). Tengo previstas ambas situaciones de aprendizaje, una en 599651702009.dkr.ecr.us-west-2.amazonaws.com/inventario-repo y otra en gcr.io/api-project-430007987702/

Para resumir, el siguiente ejercicio trata de

* Revisar entre nuestras imagenes ya generadas
* Taggear aquella que nos interesa (también) como una imagen para GCE, Dockerhub, etc.

$ docker images  
python-inventario-imagen:0.1

En este ejemplo DockerHub, mi usuario arriba es **pancutan**:

$ docker tag python-inventario-imagen:0.1 pancutan/python-inventario-imagen:0.1

Ejemplo de tag para Google Cloud

$ docker tag python-inventario-imagen:0.1 gcr.io/api-project-430007987702/python-inventario-imagen:0.1

Ahora hay que pushear nuestra imagen. Si usamos Dockerhub, basta con loguearnos con docker login, y luego hacer

docker push pancutan/python-inventario-imagen:0.1

Los clouds en cambio requieren de comandos propios (gcloud, awscli, etc) para generar credenciales y loguearnos. Siguiendo el ejemplo de Google Cloud, se generan credenciales mediante la utilidad **gcloud** para poder subir al Registry de GCE.

En el caso de Archlinux, estas utilidades se obtienen con el comando yay, ejemplo, yay google-cloud-sdk

En otros sistemas operativos se usa brew, snap o equivalente, o se siguen fáciles instrucciones de los mismos fabricantes.

Ejemplo cuando se trata de Google Cloud:

gcloud auth configure-docker  
gcloud auth login  
gcloud config set project api-project-430007987702

Ahora si: iniciamos el push

docker push gcr.io/api-project-430007987702/python-inventario-imagen:0.1

Se puede comprobar que se ha cargado correctamente la imagen, en https://console.cloud.google.com/gcr

## Clase 6 - Desplegar en Kubernetes

Consideraciones:

* La empresa que provee Docker, tiene en realidad su propio orquestador, llamado Swarm, que se gestiona sobre el mismo docker-compose.yml
* Kubernetes por su parte, requiere de hacer un transpiling del docker-compose.yml, mediante kompose.

TODO: revisar acotaciones de Pablo Fredikson - https://www.youtube.com/watch?v=q-ZicDSb3Cc&t=1s

Opciones de cursado. Despliegue sobre:

### 6.1. Equipo personal: Ej: Minikube / VirtualBox

### 6.2. VPS / VM en Vcenter / KVM en Proxmox, etc, con Kubeadm

### 6.3. PaaS: Heroku, EBS (con ebcli)

### 6.4. IaaS: EBS, AWS, GCE, Azure, Digital Ocean, etc

#### 6.4.1. Ejemplo caso GCE

* La idea es crear un proyecto Kubernetes en GCE, con tres instancias micro
* Enviar llaves, si no se ha hecho todavía, de la consola de GCE a Github
* Clon o pull del proyecto
* Bajar kompose como se explica en kompose.io, mediante curl, wget, etc - se resetea en cada nueva instancia de cloud shell.
* Dejar disponible kompose en /usr/bin

sudo mv kompose /usr/bin/

* Realizar en transpile del docker-compose.yml a varios archivos .yaml de Kubernetes
* Este paso requiere tener el docker-compose.yml aparte. La razón se encuentra en que las siguientes líneas refieren a ejecutar un build

build:  
 context: .  
 dockerfile: Dockerfile.python-simple

En tanto que Kubernetes requiere de imagenes ya listas para desplegar. La líneas anteriores deben ser comentadas y en su lugar, poner la referencia al registry de imagenes

image: gcr.io/api-project-430007987702/python-inventario-imagen:0.1

Ahora sí estamos en condiciones de hacer el transpiler con kompose create seguido de kompose create -f a cada uno de los archivos generados, o simplemente kompose up

escuelaint@cloudshell:~/Taller\_Python\_Conda\_MySQL\_Kubernetes/conversion-a-kubernetes (api-project-430007987702)$ kompose up  
  
INFO We are going to create Kubernetes Deployments, Services and PersistentVolumeClaims for your Dockerized application. If you need different kind of resources, use the 'kompose convert' and 'kubectl create -f' commands instead.  
  
INFO Deploying application in "default" namespace  
INFO Successfully created Service: db  
INFO Successfully created Deployment: db  
INFO Successfully created PersistentVolumeClaim: db-data of size 100Mi. If your cluster has dynamic storage provisioning, you don't have to do anything. Otherwise you have to create PersistentVolume to make PVC work  
INFO Successfully created Deployment: programa  
INFO Successfully created PersistentVolumeClaim: programa-claim0 of size 100Mi. If your cluster has dynamic storage provisioning, you don't have to do anything. Otherwise you have to create PersistentVolume to make PVC work

Tal como dice, “su aplicación ha sido desplegada”. Se puede monitorear ahora con comandos kubectl

escuelaint@cloudshell:~/Taller\_Python\_Conda\_MySQL\_Kubernetes/conversion-a-kubernetes (api-project-430007987702)$ kubectl get deployment,svc,pods,pvc  
NAME READY UP-TO-DATE AVAILABLE AGE  
deployment.extensions/db 0/1 1 0 42s  
deployment.extensions/hello-app 0/1 1 0 15d  
deployment.extensions/programa 0/1 1 0 42s  
  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
service/db ClusterIP 10.56.3.30 <none> 3307/TCP 42s  
service/kubernetes ClusterIP 10.56.0.1 <none> 443/TCP 15d  
  
NAME READY STATUS RESTARTS AGE  
pod/db-7d4cf4747b-f9xf8 0/1 ContainerCreating 0 42s  
pod/hello-app-576fcf7d5-vmj7s 0/1 ImagePullBackOff 0 15d  
pod/programa-55f878c7-lv5jv 0/1 ContainerCreating 0 42s  
  
NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE  
persistentvolumeclaim/db-data Bound pvc-d02524f6-b6dd-11e9-a798-42010a800016 1Gi RWO standard 43s  
persistentvolumeclaim/programa-claim0 Bound pvc-d042538Tb6dd-11e9-a798-42010a800016 1Gi RWO standard 43s

De acuerdo a la columna STATUS y a comandos tales como kubectl logs, es posible que debamos “afinar” nuestro docker-compose.yml (o los archivos generados con kompose convert) y reiniciar nuestros pods mediante kompose down / kompose up

## Propuestas para un segundo nivel

* Trabajar con otros clouds o profundizar el que hemos usado de ejemplo
* Aprender a automatizar despliegues con Integración Continua (CI)