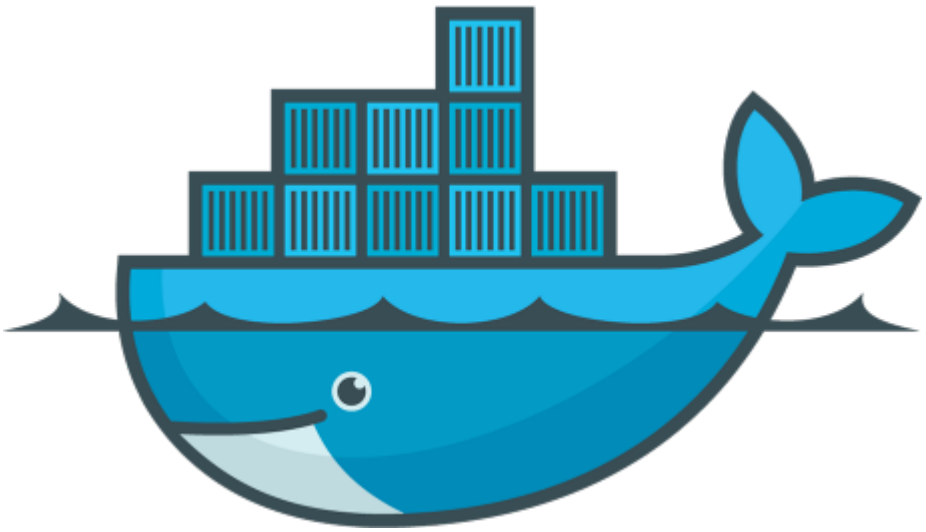
LABORATORIO DE REDES Y SISTEMAS OPERATIVOS

TRABAJO PRÁCTICO FINAL

INSTALACIÓN DE DOCKER



PROFESOR:

_ José Luis Di Biase

INTEGRANTES:

_ Alejandro Carrascosa

_ Juan Manuel Vallejos

¿Qué es Docker?

Docker es una plataforma abierta para el desarrollo, el transporte, y la ejecución de aplicaciones. Docker está diseñado para crear sus aplicaciones más rápido. Con Docker se puede separar las aplicaciones de su infraestructura y su tratamiento como una aplicación administrada. Docker ayuda a transportar e implementar código y testear más rápido, además acorta el ciclo entre el código escrito y el código que se ejecuta.

¿Para qué puedo usar Docker?

Docker es perfecto para que lo ayude con el ciclo de vida de desarrollo. Además permite a los programadores desarrollar en contenedores locales que contienen sus aplicaciones y servicios. Estos contenedores se pueden integrar en un flujo de trabajo de integración y despliegue continuo.

Por ejemplo, los desarrolladores pueden escribir código de forma local y compartir su pila de desarrollo con sus colegas. Una vez terminado, se lleva el código en el que está desarrollando a un entorno de prueba y donde se ejecutan los tests exigidos. Desde el entorno de prueba, se puede descargar las imágenes de Docker en producción y desplegar su código.

Implementacion mas facil

La plataforma basada en el contenedor de Docker permite cargas de trabajo altamente portátiles. Contenedores acoplables pueden ejecutar en el host local de un desarrollador, en máquinas físicas o virtuales en un datacenter, o en la nube.

La portabilidad de Docker y la naturaleza ligera facilitan la forma de gestionar dinámicamente las cargas de trabajo. Puede utilizar Docker para escalar rápidamente hacia arriba o derribar aplicaciones y servicios. La

velocidad del Docker significa que la escala puede ser casi en tiempo real.

En Resumen, con Docker:

1. Podés construir imágenes que tienen tus aplicaciones.
2. Podés crear contenedores Docker de esas imágenes Docker para ejecutar las aplicaciones.
3. Podés compartir esas imágenes Docker vía Docker Hub o en su propio registro.

¿Por qué Docker?

Porque es un entorno diseñado para trabajar mejor. Los contenedores de Docker , y el flujo de trabajo que viene con ellos , ayuda a sus desarrolladores, administradores de sistemas , la gente de control de calidad , y los ingenieros a trabajar juntos para obtener el código en producción y hacerlo útil . El contenedor es un formato estándar que permite a los desarrolladores que se preocupen por sus aplicaciones, mientras que los administradores de sistemas y operadores pueden trabajar en el manejo del recipiente en su despliegue. Esta separación de funciones agiliza y simplifica la gestión y despliegue de código.

Con Docker es más fácil construir nuevos contenedores, permiten una rápida iteración de sus aplicaciones, y aumentar la visibilidad de los cambios. Esto ayuda a todos en la organización a comprender cómo funciona una aplicación y cómo se construye.

Los contenedores Docker son ligeros y rápidos. Estos tienen tiempos de lanzamiento en sub-segundos, lo que reduce el tiempo del ciclo de desarrollo, prueba y despliegue.

¿Como funciona Docker?

Docker está escrito en Go y hace uso de varias características de Linux para ofrecer la funcionalidad que hemos visto

Instalación

De acuerdo al sistema operativo que se tenga, hay distintos comandos y formas para su correcta instalación. En este tutorial nos vamos a parar sobre la instalación para el sistema operativo:

"Ubuntu Trusty 14.04 (LTS) (64-bit)"

Primero comenzaremos por instalar el paquete que nos provee Docker:

```
$ sudo apt-get update
$ sudo apt-get install docker.io
```

Luego para que podamos autocompletar los comandos de Docker en Bash, tenemos dos opciones, la primera es reiniciar Bash, y la segunda es ejecutar en terminal la siguiente línea:

```
$ source /etc/bash_completion.d/docker.io
```

Para continuar, se debe verificar que el sistema APT (Advanced Packaging Tool) trabaja con https, para verificarlo se debe constatar de que la siguiente ruta exista '/usr/lib/apt/methods/https'. En caso de no existir entonces se lo debe configurar de la siguiente manera, tipeando por terminal:

```
[ -e /usr/lib/apt/methods/https ] || {
  apt-get update
  apt-get install apt-transport-https
}
```

Una vez verificado esto, se debe almacenar localmente la clave del repositorio de Docker:

```
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
36A1D7869245C8950F966E92D8576A8BA88D21E9
```

Ahora nos queda agregar el repositorio Docker a la lista de fuentes de apt, actualizar e instalar el paquete lxc-docker:

Quizás aparezca una advertencia de que el paquete no es de confianza. Como sabemos que es de confianza, tipear sí/yes para continuar la instalación.

```
$ sudo sh -c "echo deb https://get.docker.com/ubuntu docker main\
> /etc/apt/sources.list.d/docker.list"
$ sudo apt-get update
$ sudo apt-get install lxc-docker
```

Ya tenés Docker instalado! Para verificar que todo esté funcionando correctamente ejecutar la siguiente línea:

```
$ sudo docker run -i -t ubuntu /bin/bash
```

Esta ejecución lo que hace es correr bash sobre la imagen ubuntu hosteada en Docker. Entonces primero descarga la imagen y luego inicia bash en un contenedor.

Primeros pasos con Docker Hub

Docker Hub es un recurso centralizado para trabajar con Docker y sus componentes. Nos ayuda a colaborar con colegas y obtener el máximo provecho. Para ello, nos brinda los siguientes servicios:

- *Alojamiento de imágenes.*
- *Autenticación de usuarios.*
- *Integración con GitHub y BitBucket.*

Para utilizar Docker Hub, primero tendrá que registrarse y crear una cuenta. No te preocupes, la creación de una cuenta es sencillo y gratuito.

Se puede crear una cuenta en Docker Hub via web mediante https://hub.docker.com/account/signup/ *, o tambien mediante la linea de comando insertando la siguiente linea por terminal:*

```
$ sudo docker login
```

Ahora debemos entrar a nuestra cuenta de email, ya que recibiremos un mail de bienvenida en donde tendremos que confirmar para activar la cuenta.

Ejecutando Aplicaciones con Docker

Con Docker toda aplicacion se ejecuta dentro de un contenedor. Toda ejecución inicia con un único comando: 'docker run'.

PROBAR UN HELLO WORLD

Empecemos ejecutando la siguiente línea:

```
$ sudo docker run ubuntu:14.04 /bin/echo 'Hello world'
Hello world
```

Bien, pero ¿Cómo sucedió esto?

Como dijimos antes, toda aplicación de Docker empieza con un 'docker run'. Siguiendo la inspección de nuestra línea de comandos, notaremos que se toma como primer parámetro 'ubuntu:14.04', lo cuál es la imagen en la que se basa para correr el contenedor. El resto hace una impresión en pantalla como lo haríamos en una terminal de ubuntu en su versión 14.04 (aunque en este caso cualquier versión lo correría, debido a que es una simple impresión en pantalla).

CONTENEDOR INTERACTIVO

Nos referimos como contenedor interactivo, a un contenedor que se queda esperando por recibir instrucciones.

Como ejemplo, probemos para abrir un contenedor interactivo que corra sobre bash:

```
$ sudo docker run -t -i ubuntu:14.04 /bin/bash
```

Antes de seguir, veremos qué significan los comandos ingresados. Si hacemos una suerte de comparación con la línea ingresada anteriormente, nos daremos cuenta que al comando 'docker run' se le sumaron dos parámetro más. Estos parámetros son los flags '-t' y '-i'. El primero nos permite correr una terminal en el contenedor creado. El segundo flag, nos indica la norma (Standard Input) por la cual se van a recibir los input en dicha terminal.

Nuestro contenedor creado ya esta ejecutandose y se vé de la siguiente manera:

```
root@af8bae53bdd3:/#
```

Cómo prueba podríamos ejecutar comandos bash, como por ejemplo, 'pwd' para ver donde estamos parados, y 'ls'. De esta forma

comprobamos que estamos en una imagen completamente distinta a la de nuestro ubuntu, a una imagen ubuntu que se descargo de Docker.

Ahora si queremos salir de dicha terminal, lo único que debemos ejecutar es:

```
root@af8bae53bdd3:/# exit
```

CONTENEDOR NO INTERACTIVO (DAEMON)

Se podría ver como claro ejemplo de contenedor no interactivo, a un proceso que corra en background (segundo plano), como ya sabemos, en la jerga informática a estos procesos o servicios se lo denomina 'DAEMon' (nombre procedente de 'Disk And Execution Monitor)'

Para probar uno de estos contenedores probemos como ejemplo tipear:

```
$ sudo docker run -d ubuntu:14.04 /bin/sh -c "while true; do echo hello world; sleep 1; done"
```

Para entender esto, vayamos por partes. Mirando la línea, nos daremos cuenta de que se insertó un nuevo flag: '-d'. Este flag es el que nos esta indicando que dicho contenedor corra en background. Luego ejecuta un pequeño programa que nos imprimirá un 'hello world' indefinidamente con un lapso de impresion de un segundo.

¿No estas viendo la impresión en pantalla? A no desesperarse, recordemos que dicho proceso esta corriendo en segundo plano, pero entonces, ¿Cómo hacemos para verificar que nuestro programa esta haciendo lo que creemos que hace?

Ejecutemos el siguiente comando:

```
$ sudo docker ps
```

Como vemos, esto nos dió toda la información de los procesos en ejecución. De la información dada, por ahora prestemos atención a que el proceso contiene un ID (identificador único) y un nombre que se autogeneró y se asoció a dicho daemon.

Teniendo en cuenta el ID o el nombre del proceso en ejecución hagamos lo siguiente:

```
$ sudo docker logs id_o_nombre_del_proceso
```

De esta forma veremos que efectivamente el proceso está corriendo como nosotros queríamos.

Ahora que ya no nos interesa más este daemon, necesitamos pararlo y eliminarlo (se recomienda eliminar todo proceso que no se utilice más).

Para parar el proceso:

```
$ sudo docker stop id_o_nombre_del_proceso
```

Para eliminarlo (solo si se lo paró anteriormente):

```
$ sudo docker rm id_o_nombre_del_proceso
```

Aplicaciones Web con Docker

Antes de continuar, aclaremos un poco algo, como hemos visto, Docker les provee a sus usuarios imágenes, que pueden haber sido creados por otros usuarios que lo quisieron compartir, o creados por los desarrolladores de Docker con el objetivo de probar tutoriales.

Todas las imágenes que provee Docker para su práctica en tutoriales empiezan con un 'training' seguido de una '/' y el nombre.

Para buscar imágenes se utiliza el comando 'search', entonces hagamos una prueba y miremos todas la imagenes con el prefijo 'training'

```
$ sudo docker search training
NAME                                    STARS
training/webapp                         7
training/sinatra                        5
training/docker-fundamentals-image      3
training/postgres                       2
training/jenkins                        1
training/namer                          1
bkimminich/juice-shop                   1
training/notes                          1
training/showoff                        1
ijiayi/docker-training-hellokitty       0
gcivitella/trainingapache               0
*
*
*
```

Ahora que sabemos esto, prosigamos. Para nuestra práctica en aplicaciones web, vamos a necesitar la primer imagen del listado.

Ahora vamos a ejecutar:

```
$ sudo docker run -d -P training/webapp python app.py
```

Estos comandos nos indican varias cosas. En primer lugar se lo corre en segundo plano, como vimos, con el flag '-d'. Luego aparece otro flag que no hemos visto hasta el momento: '-P'. Este argumento le asigna al contenedor y su aplicación un puerto en nuestra máquina. Siguiendo, tenemos la imagen, el lenguaje que precisa la aplicación, y la aplicación en sí (app.py).

Listemos los procesos nuevamente:

```
$ sudo docker ps
```

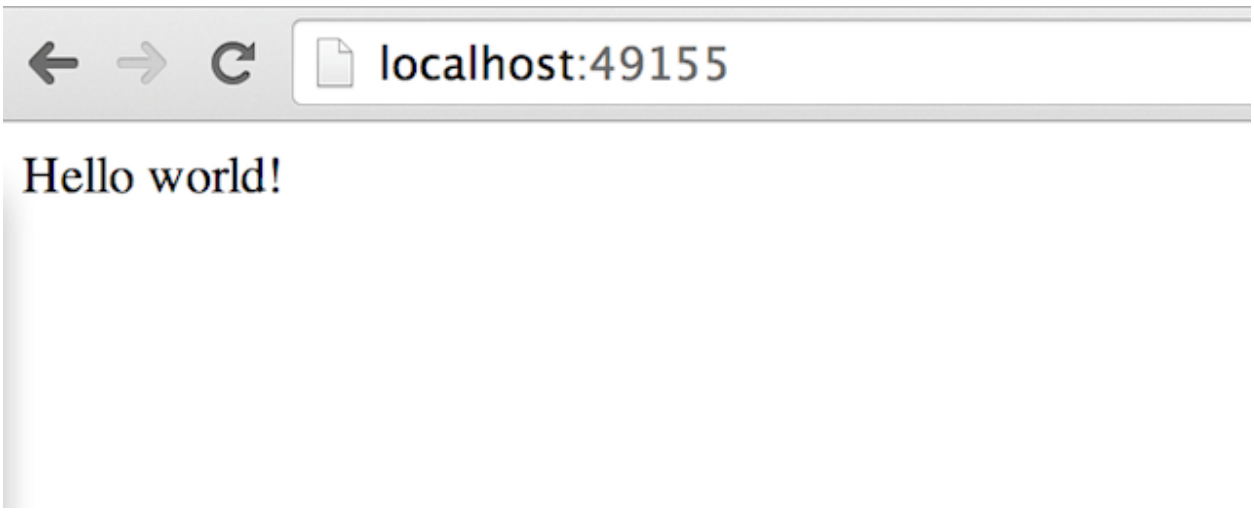
Ahora para ver lo que la aplicación está haciendo miremos el puerto que se le asignó automáticamente con el flag '-P'.
En la sexta columna, tendría que aparecer algo del siguiente estilo:

```
PORTS
0.0.0.0:49155->5000/tcp
```

Esto se debe a que los puertos que se le asignan a esta aplicación tienen por default el 5000 via tcp, entonces se le asigna un puerto rondando dicho puerto, en este caso, el 49155.

Probemos ahora la aplicación ya teniendo este dato.

En nuestro navegador ingresamos como URL 'localhost:', seguido del puerto indicado:



Felicitaciones, la aplicación se esta ejecutando!

Veamos la interacción a nivel http que hay en dicha aplicación:

```
$ sudo docker logs -f id_o_nombre_del_proceso
```

A lo visto, se agregó el flag '-f'. Que lo que hace es ver la interacción desde fuera.

Entonces obtendremos como resultado algo parecido a esto:

```
* Running on http://0.0.0.0:5000/
10.0.2.2 - - [23/May/2014 20:16:31] "GET / HTTP/1.1" 200 -
10.0.2.2 - - [23/May/2014 20:16:31] "GET /favicon.ico HTTP/1.1" 404 -
```

Ahora si queremos inspeccionar lo que está pasando por JSON:

```
$ sudo docker inspect id_o_nombre_del_proceso
```

Y así vemos la salido en JSON de la aplicación:

```
[{
    "ID": "bc533791f3f500b280a9626688bc79e342e3ea0d528efe3a86a51ecb28ea20",
    "Created": "2014-05-26T05:52:40.808952951Z",
    "Path": "python",
    "Args": [
        "app.py"
    ],
    "Config": {
        "Hostname": "bc533791f3f5",
        "Domainname": "",
        "User": "",
. . .
```

Para finalizar, tendremos que parar el proceso y eliminarlo como hemos visto:

Parar:

```
$ sudo docker stop id_o_nombre_del_proceso
```

Eliminar:

```
$ sudo docker rm id_o_nombre_del_proceso
```

Trabajando con las imágenes en Docker

A lo largo de este tutorial, pudimos ver de lo que son capaces las imagenes. Pero hasta ahora lo estamos mirando muy desde afuera. Es por eso que en esta sección veremos:

- *El manejo y trabajo de imagenes locales sobre el host que nos provee Docker*
- *La creación de imágenes simples*
- *Subir nuestras imagenes a Docker Hub*

Podemos empezar viendo la imágenes que estan almacenadas localmente.

Con el comando 'images':

```
$ sudo docker images
```

Como veremos, las imagenes que hemos probado anteriormente para ejecutarse, se descargaron, y la tenemos almacenadas.

Si queremos descargar una imagen directamente, con 'pull':

```
$ sudo docker pull nombre_imagen
```

Para ver como se suben los cambios, descargaremos una imagen Docker de prueba sobre sinatra: 'training/sinatra'

Y ejecutaremos lo siguiente:

```
$ sudo docker run -t -i training/sinatra /bin/bash
root@0b2616b0e5a8:/#
```

-Notemos que luego del arroba nos muestra el id del contenedor creado-

Ahora hagamos un cambio cualquiera, a modo de prueba, solo para ver como se suben los cambios:

Por ejemplo, agregando la gema json y salgamos:

```
root@0b2616b0e5a8:/# gem install json
root@0b2616b0e5a8:/# exit
```

Es hora de subir los cambios con el comando 'commit':

```
$ sudo docker commit -m="Se agrego la gema json" -a="Fidel" \
0b2616b0e5a8 nuestro_usuario/sinatra:v2
```

Este comando recibe en principio dos argumentos, el mensaje ('-m=') y el autor del commit ('-a='). Luego de la '/' recibe otros dos parámetros más, el ID del contenedor que hizo los cambios, y la imagen, que por convención, primero se debe escribir el nombre de nuestro usuario seguido del nombre del proyecto.

Si prestamos atención veremos que a nuestro proyecto se le agregó un ':v2'. Esto es a modo de que puedan ver que se le pueden agregar etiquetas de esta forma, poniendo dos puntos y seguido del nombre de la etiqueta.

CREAR IMAGEN CON DOCKERFILE

Hasta el momento, sólo hemos usado imagenes ya creadas.
Es por eso que vamos a crear nuestra imagen propia. Para esto necesitamos crear un directorio en nuestra PC, y una vez adentro crear un documento 'Dockerfile'.

Para crear directorio e insertar un documento llamado 'Dockerfile':

```
$ mkdir sinatra
$ cd sinatra
$ touch Dockerfile
```

Ahora veamos un ejemplo en nuestra imagen creada sobre sinatra. Para este ejemplo, el Dockerfile debe tener los siguientes datos:

```
# Esto es un comentario
FROM ubuntu:14.04
MAINTAINER Juan Perez <jperez@ejemplo.com>
RUN apt-get update && apt-get install -y ruby ruby-dev
RUN gem install sinatra
```

Cada instrucción inicia en mayúscula y sigue con su declaración. Con la instrucción FROM le indicamos la imagen en la que se quiere correr. Con MAINTAINER, le indicamos quién es el que mantiene la imagen creada seguido opcionalmente de su mail. Con RUN le vamos instalando y configurando lo que necesitemos para nuestro proyecto.

Ahora que tenemos nuestro Dockerfile creado, nos queda construir la imagen.

Construimos la imagen con el comando 'build':

```
$ sudo docker build -t="nuestro_usuario/sinatra:v2" .
```

Como dijimos el comando build construye la imagen. Y el flag '-t' le indica el nombre de la imagen con su etiqueta. Prestemos atención que esta sentencia termina con un punto. Este punto le indica a docker que el Dockerfile esta en el directorio actual.

-El Dockerfile no puede tener más de 127 líneas-

Teniendo nuestra cuenta en DockerHub subamos nuestra imagen.

Subimos nuestra imagen con ‘push’:

```
$ sudo docker push nuestro_usuario/sinatra:v2
```

Listo, ahora si entramos a nuestra cuenta, notaremos que se nos agrego la imagen a nuestro repositorio.

En caso de querer eliminar algún repositorio del host, se puede eliminar desde terminal.

Se utiliza el comando ‘rmi’:

```
$ sudo docker rmi training/sinatra
```