



DECODED

ACADEMY

FULL STACK DEVELOPER BOOTCAMP

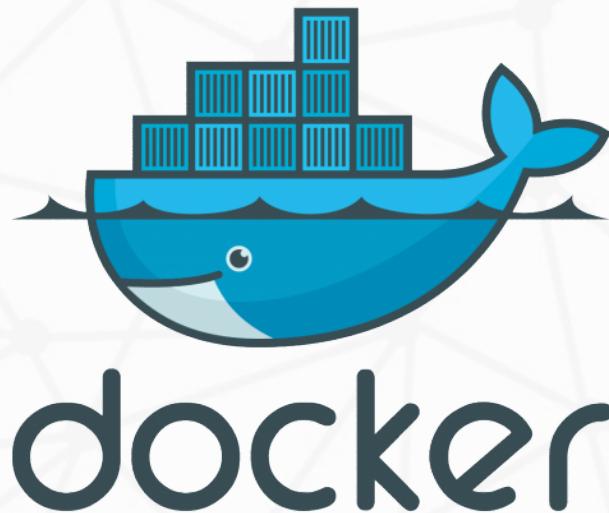
Desarrollamos
{ talento }

The background of the slide features a light gray polygonal grid pattern. Overlaid on this are several large, semi-transparent orange triangles. One triangle is positioned in the upper left quadrant, pointing downwards. Another is at the bottom left, pointing upwards. A smaller orange diamond shape is located near the top center. A single horizontal orange line segment is located in the middle right area.

DOCKER

Introducción

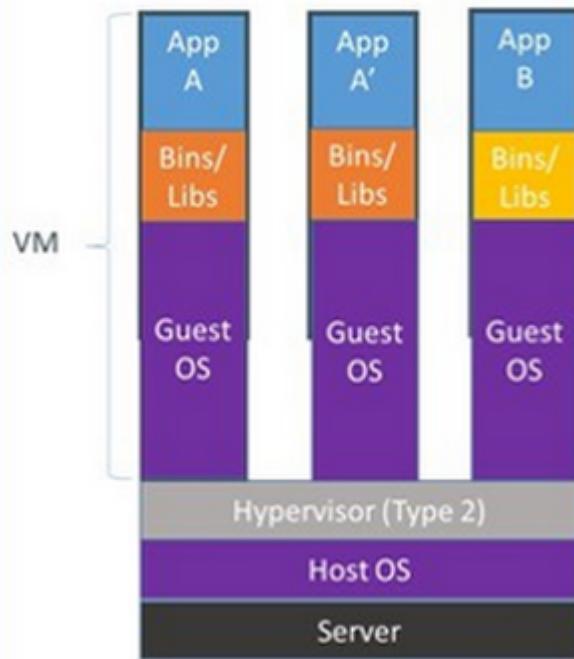
Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de Virtualización a nivel de sistema operativo en Linux



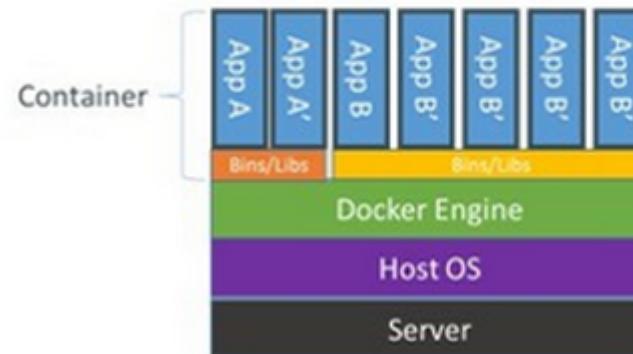
Docker le proporciona una manera estándar de ejecutar su código. Docker es un sistema operativo para contenedores. De manera similar a cómo una máquina virtual virtualiza (elimina la necesidad de administrar directamente) el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor. Docker se instala en cada servidor y proporciona comandos sencillos que puede utilizar para crear, iniciar o detener contenedores.

Docker vs Virtual Machines

Containers vs. VMs



Containers are isolated, but share OS and, where appropriate, bins/libraries



Imágenes y Contenedores

Los términos que hay que manejar con Docker son principalmente 2, las imágenes y contenedores.

- Las imágenes en Docker se podrían ver como un componente estático, pues no son más que un sistema operativo base, con un conjunto de aplicaciones empaquetadas,
- Un contenedor es la instancia o ejecución de una imagen, pudiendo ejecutar varios contenedores a partir de una misma imagen.

Haciendo una analogía con la POO una imagen es una clase y un contenedor es la instancia de una clase, es decir un objeto.

Instalación de Docker.

Instalar en ubuntu

Instalamos la clave GPGP

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

Añadimos el Repositorio de Docker.

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

Instalar en ubuntu

Actualizamos el repositorio

```
sudo apt-get update
```

Comprobamos que ya disponemos en el repositorio Docker para Instalar

```
apt-cache policy docker-ce
```

Instalamos Docker

```
sudo apt-get install -y docker-ce
```

Instalar en Windows

En windows 10 directamente descargando desde [aquí](#)

En windows versiones anteriores o en MAC debemos utilizar Docker ToolBox que podemos descargar de [aquí](#)

Inicio con Docker

Ahora ejecutamos una imagen de test que se denomina **hello-world**

```
$ docker run hello-world
```

Inicio con Docker

Podemos comprobar las imágenes descargadas :

```
//Recuperar todo el listado.  
$ docker image ls
```

Por último podemos comprobar los contenedores generados con:

```
$ docker container list --all
```

Creando nuestra primera imagen

Podemos disponer de un DockerFile donde podemos gestionar la imagen de nuestro contenedor
para ello creamos un archivo **.dockerfile** o **Dockerfile**

```
$ nano Dockerfile
```

Creando nuestra primera imagen

El archivo dockerfile contiene las siguientes Tags.

- FROM -> Configura la imagen base para usar.
- WORKDIR -> Establece el directorio para las directivas de CMD que se ejecutarán.
- RUN->Ejecuta un comando y cambia la imagen.
- ENV -> Inicializa variables de entorno
- EXPOSE -> Expone un puerto al exterior
- MAINTAINER-> Establece los datos de autor/propietario del Dockerfile.

- ENTRYPOINT -> Punto de entrada por defecto de la aplicación desde el contenedor
- CMD ->Configure comandos por defecto para ser ejecutado
- USER-> Establece el usuario para ejecutar los contenedores de la imagen.
- VOLUMEN -> Monta un directorio desde el host al contenedor.

Creando nuestra primera imagen

Creando nuestro DockerFile

Añadimos los siguientes parámetros para crear la imagen.

```
FROM node:9.11
WORKDIR /usr/src/app

COPY package*.json ./
RUN npm install
# If you are building your code for production
# RUN npm install --only=production
COPY .

EXPOSE 8080
CMD [ "npm", "start"]
```

.dockerignore

Podemos crear un archivo para ignorar al hacer la creación de imágenes igual a gitignore, en estos caso .dockerignore

```
// Contenido .dockerignore  
node_modules  
npm-debug.log
```

Proveer archivos

Ahora debemos de crear una package.json, crearemos además un server.js con una respuesta simple con express y un script "npm start" para el inicio de nuestro servidor

```
'use strict';

const express = require('express');

// Constants
const PORT = 8080;

// App
const app = express();
app.get('/', (req, res) => {
  res.send('Hello world\n');
});

app.listen(PORT, () => {
  console.log(`Running on => http://${
    `localhost` }:${PORT}`);
});
```

Creando nuestra primera imagen

Ahora sólo debemos ejecutar el siguiente comando para generar la imagen.

```
$ cd dirname/  
$ sudo docker build . -t node_9-11_diapositivas
```

Creando nuestra primera imagen

Ahora ya podemos lanzar nuestro nuevo contenedor a partir de una imagen

```
$ docker run \[nombre\_\_de\_\_la\_\_imagen\]
```

No va a ser accesible desde el exterior, pero le podemos asignar una redirección de puertos además de un alias para reconocer nuestro nuevo contenedor.

```
docker run --name aliasDelContenedor -p 49160:8080 -d node_9-11_diapositivas
```

Probando Express desde comandos.

Para poder probar desde comandos nuestra aplicación de NodeJs + Express utilizaremos "curl"

```
$ curl -i 192.168.99.100:49160
```

Respuesta:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 12
ETag: W/"c-M6tWOb/Y57lesdjQuHeB1P/qTV0"
Date: Tue, 17 Jul 2018 16:05:19 GMT
Connection: keep-alive

Hello world
```

Comandos básicos de Docker

Comandos básicos para imágenes

- **docker build ./primeraImagen -t "prueba"** Construir imagen.
- **docker build [path] –build-arg [args] -t [name]** Construir detras de un proxy.

```
docker build ./primeraImagen --build-arg  
HTTP_PROXY=http://10.20.30.2:1234 -t "prueba_proxy"
```

- **docker pull [nombre-imagen]** Descargar una imagen ya construida:
\$ docker pull mongo
- **docker inspect [nombre-imagen]** Inspecciona una imagen mostrando información detallada en formato json.

- **docker save [nombre-imagen] > [nombre-fichero].tar** Guardar la imagen en un archivo de tipo tar.

```
docker save mongo > ./miMongo.tar
```

- **docker rmi [nombre-imagen]:[TAG]** Eliminar una imagen.
- **docker rmi docker images -q** Eliminar todas las imágenes.
- **docker images** Lista todas las imágenes.
- **docker exec -it [id-imagen] /bin/bash**

Comandos básicos para contenedores

- **docker stop [OPTIONS] CONTAINER [CONTAINER...]** Parar un contenedor

```
$ docker stop my_container
```

- **docker run -v [unidad-host]:[unidad-docker] –name[nombre-contedor] -d [nombre-imagen]** Ejecutar contenedor.

```
docker run --name aliasDelContenedor -p 49160:8080 -d node_9-11_diapositivas
```

- **docker ps** Lista todos los contenedores en ejecución, con el argumento **-a** muestra todos los contenedores.

- **docker rm [nombre_contenedor]** Eliminar un contenedor.
- **docker rm \$(docker ps -a -q)** Eliminar todos los contenedores parados.
- **docker logs -f [nombre_contenedor]** Muestra el log del proceso principal (CMD)

```
docker logs -f mi-contenedor
```

Docker-compose

Un servicio de docker es una solución que nos proporciona docker para utilizar diversas imágenes y comunicarlas entre ellas.

Mediante la utilización de **docker-compose**

Creando nuestro servicio de docker

Para poder definir nuestro contenedor, Docker utiliza un descriptor que se almacena en formato [yaml](#)

En la que especificamos los diferentes contenedores y sus propiedades, básicamente podemos indicar las mismas propiedades que establecemos en el dokerfile

Creando nuestro servicio de Docker

Elementos del docker-file

- Version -> Se refiere al formato del archivo de compose
- Services:(db, web) -> Indicamos los diferentes elementos a ejecutar
- Elementos(db,web)-> Se indican los contenedores a levantar.
- container_name -> Nombre del contenedor.
- image -> Nombre de la imagen que se creará nuestro docker.
- ports -> Para indicar los puertos que se utilizaran
- environments -> Variables de entorno a utilizar.
- volumes -> Carpetas relacionadas a nuestro contenedor.

Creando nuestro servicio de Docker

Ejemplo de docker-compose.yml

```
rabbit1:  
  image: "rabbitmq"  
  environment:  
    RABBITMQ_ERLANG_COOKIE: "SWQOKODSQALRPCLNMEQG"  
    RABBITMQ_DEFAULT_USER: "admin"  
    RABBITMQ_DEFAULT_PASS: "admin"  
    RABBITMQ_DEFAULT_VHOST: "/"  
  ports:  
    - "15672:15672"  
    - "5672:5672"  
  volumes:  
    - "./enabled_plugins:/etc/rabbitmq/enabled_plugins"
```

Comandos de docker compose

Disponemos de diferentes comandos para ejecutar docker-compose

Levantar los servicios de docker

```
$ docker-compose up
```

Ejecutar comandos sólo en un servicio de nuestro docker-compose

```
$ docker-compose run [nombre del servicio]
```

Ejecutar comandos desde otra ubicación

```
docker-compose -f "path/to/docker-compose.yml" up -d --build
```

Comandos de docker compose

Podemos ver los servicios levantados

```
$ docker-compose ps
```

Para parar los servicios

```
$ docker-compose stop
```

Para parar alguna de las propiedades del servicio

```
$ docker-compose down --volumes
```

Comandos de docker compose

- build
- bundle
- config
- top
- logs
- restart
- pull
- push