UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

TREBALL DE FI DE GRAU

TÍTULO DEL TFG: Contenerizacion IoT en raspis

TITULACIÓN: Grau en Enginyeria Telemàtica

AUTOR: Alfredo Varela Romero

DIRECTOR: Juan López Rubio

SUPERVISOR: Gerard Sole Castellvi

FECHA: 21 de septiembre de 2016

visiongeneral

Título: Contenerizacion IoT en raspis

Autor: Alfredo Varela Romero

Director: Juan López Rubio

Supervisor: Gerard Sole Castellvi

Fecha: 21 de septiembre de 2016

Resumen

En los últimos años, el avance sobre los temas de IoT han llevado a un crecimiento muy elevado en la investigación tanto en dispositivos como de tecnologías para el cumplimiento de todas las barreras que nos ponían el software y el hardware que disponíamos hasta conseguir protocolos muy simples y sensores extremadamente pequeños y baratos. La demanda de este tipo de productos hace que muchas empresas cambie su orientación de mercado y se centren en el mundo de los sensores o incluso invierten en hacer una instalación de este tipo para mejorar su productividad.

Las investigaciones no solo se centran en crear nuevos protocolos o nuevas tecnologías sino también en reutilizar las que ya se conocen aplicando pequeños cambios o incluso sin mejoras ya que son completamente compatibles.

Por las razones expuestas, surge este proyecto para realizar un nuevo producto para una empresa la cual podrá disponer de toda sus infraestructura de manera remota y poder desplegarla en un lugar el cual el acceso o la conexión a internet sea escaso o nulo, hayan ocurrido desastres naturales... Y paralelamente la investigación de una nueva tecnología para desplegar aplicaciones utilizada sobretodo en el ámbito de servidores y serán llevadas al mundo de IoT para solucionar problemas como el alto consumo de ancho de banda al hacer una actualización del sistema o desplegar nuevos sistemas y el tiempo que conlleva realizar estas acciones.

En el proyecto se utilizara una Raspberry Pi para simular un nodo central o sumidero de datos en el cual mediante Docker se desplegará toda la infraestructura del core de la empresa AlterAid, aaaida. Se utilizarán sensores que mediante bluetooth low energy se conectaran a la Raspberry y se establecerá la red la cual permitirá el envío de los datos y su correcto almacenamiento para ser gestionados por la aplicacion.

Title : This is the Document's title

Author: Alfredo Varela Romero

Advisor: Juan López Rubio

Supervisor: Gerard Sole Castellvi

Date: September 21, 2016

Overview

This document contains guidelines for writing your TFC/PFC. However, you should also take into consideration the standards established in the document Normativa del treball de fi de carrera (TFC) i del projecte de fi de carrera (PFC), paying special attention to the section Requeriments del treball, as this document has been approved by the EETAC Standing Committee

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

INTRODUCCIÓN

En los últimos años, el avance sobre los temas de IoT han llevado a crecimiento muy elevado en la investigación tanto como dispositivos como de tecnologías para el cumplimiento de todas las barreras que nos ponían el software y el hardware que disponíamos hasta conseguir protocolos muy simples y sensores extremadamente pequeños y baratos. La demanda de este tipo de productos hace que muchas empresas cambie su orientación de mercado y se centren en el mundo de los sensores o incluso invierten en hacer una instalación de este tipo para mejorar su productividad.

Por eso se decidió realizar este proyecto, el cual se llevará a cabo el despliegue de la plataforma aaaida en una Raspberry. La cual nos puede permitir el uso de dicha plataforma y sus aplicaciones en lugares a los cuales la conexión a internet es nula o escasa, hayan sufrido un desastre natural... ya que es una plataforma centrada en Ihealth que nos permite la monitorización del estado de un paciente. Gracias a las conexiones de la Raspberry podremos realizar una red con sensores que mediante Bluetooth se podrán comunicar y almacenar los datos.

Por otra parte el despliegue de la aplicación se llevará a cabo usando Docker, un software que permite contenerizar todo aquello que nuestra aplicaciones necesiten. El cual no sera muy util ya que nos permite instalar o actualizar contenedores por separado y no todo el sistema cada vez. Docker es utilizado básicamente en el mundo de servidores y no de IoT lo cual las ventajas que nos proporciona a la hora del despliegue son contrarrestadas con su complejidad en sistemas con capacidades limitadas.

Nuestra intención es poder desplegar toda la infraestructura de la empresa utilizando un sistema nuevo para IoT, comprobar su viabilidad y beneficios, ofreciendo un producto el cual podría ser de gran utilidad en caso de desastres como seria un sistemas de monitorización de pacientes.

La utilización de la Raspberry hace que esta infraestructura no sea extrapolable al cien por cien de los despliegues de IoT ya que sus especificaciones son bastantes superiores a las de sensores utilizados normalmente, pero como aproximación y prueba piloto para la tecnología nombrada anteriormente es suficiente. Docker podría abrirse un hueco en el mundo de IoT, aunque tenga este orientado a arquitecturas de servidor con recursos ilimitados, pero en dispositivos con recursos más limitados podría ser utilizado.

CAPÍTULO 1. VISIÓN GENERAL DEL PROYECTO

El propósito de este proyecto es desplegar toda la infraestructura de la empresa AlterAid en un dispositivo móvil en nuestro caso una Raspberry Pi. Con la finalidad de poder conseguir desplegar las aplicaciones en lugares remotos o sin conexión a internet o que han sufrido un desastre natural.

Como segundo objetivo del proyecto queremos implementar un pequeño despliegue sensores haciendo que nuestra Raspberry sea el nodo central o sumidero de datos. Para poder llevar a cabo todo esto es necesario contar con la utilización de los contenedores usando la tecnología Docker. Que nos facilitaran el trabajo y es motivo de investigación su utilización fuera del mundo de servidores.

1.1. Docker

La idea detrás de Docker es crear contenedores ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues.

Figura 1.1: Logotipo de Docker

1.2. Aaaida

Es una plataforma desarrollada por la empresa AlterAid la cual nos permite crearnos un usuario y con este usuario poder tener varios vínculos. Un Vínculo es un ente que representa aquello por el que el Usuario se ocupa y preocupa. Ejemplos de Vínculos de Usuario pueden ser familiares, pacientes, amigos, etc. Esta es una plataforma web la cual recoge todos los datos de las aplicaciones de la empresa y los almacena. Por eso la importancia de poder desplegar toda la infraestructura en un dispositivo que sea fácil de transportar y poder llegar a cualquier lugar.

Figura 1.2: Logotipo de Aaaida

1.3. Motivación

Conseguir el despliegue de toda la infraestructura de una empresa en un dispositivo de reducidas prestaciones y tamaño como seria una Raspberry Pi, puede abrir campos de investigación y mercado ya que se podrían desarrollar otro tipo de aplicaciones para casos de emergencia o lugares con pocos medios. Utilizando una tecnología de servidores como es Docker conlleva un gran avance en el mundo de las IoT. Un mundo en que las limitaciones tanto de hardware como de software son muy estrictas, nuestro proyecto no es un ejemplo válido ya que el dispositivo utilizado no sería un nodo común utilizado pero sí podría ser un pequeña prueba de concepto para versiones futuras en las cuales se podría utilizar estas tecnologías para facilitar el despliegue de las aplicaciones como de sus actualizaciones ya que al utilizar docker solo se deberia actualizar el contenedor independiente y no todo el sistema.

1.4. Objetivos

Los objetivos fijados para el desarrollo de este proyecto son los siguientes:

- Analizar y escoger las tecnologías a utilizar para el desarrollo del proyecto
- La instalaciones de Docker en una Raspberry Pi
- El despliegue de la infraestructura de la empresa AlterAid
- La creación de pequeños nodos
- El despliegue de la red
- Contemplar la viabilidad de creación de redes smart con estas tecnologías.

1.5. Organización del proyecto

En primer lugar para poder cumplir todos los puntos de los objetivos necesitaremos estudiar un poco más a fondo todas las tecnologías que se pueden utilizar y utilizaremos durante todo el proyecto que serán explicadas en sus capítulos.

Como se ha explicado previamente se utilizara una Raspberry para desplegar la aplicación y esto se hará mediante Docker para ello se tiene que estudiar las diferentes posibilidades, como si es posible ejecutar docker en un sistema ARM, si se podrá virtualizar la Raspberry para poder hacer las pruebas de una manera mas comoda etc. todo se podrá ver en sus capítulos, capítulo 2 y capítulo 3.

Una vez solucionado todos los problemas de desplegar la aplicación y ejecutarla, para poder arrancar aaaida, en el capítulo 4 podremos ver una pequeña explicación del funcionamiento de la plataforma y sus funcionalidades.

Para terminar la parte técnica, vendría el paso de crear la red de sensores para poder comunicarnos con la Raspberry una vez se haya llevado a cabo esta conexión el envío de los datos y su correcto almacenamiento para poder ser visualizados en la plataforma aaaida todo esto se comentará en el capítulo 5.

Por último, con la plataforma en la Raspberry y la red de sensores funcionando obtendremos el último capítulo donde se podrán ver las conclusiones y resultados sobre la viabilidad de este proyecto.

CAPÍTULO 2. VIRTUALIZACIÓN

2.1. ¿Qué es la virtualización?

La virtualización es la creación de una versión virtual basada en software de algo, en lugar de una física. Se puede aplicar a sistemas operativos, almacenamiento, servidores, aplicaciones, redes... y es una manera de reducir gastos, aumentar eficiencia y agilidad en las empresas.

2.1.1. Tipos de virtualización

Estos son los 4 tipos de virtualización más habituales.

2.1.1.1. Virtualización de servidores

La virtualización en servidores ayuda a evitar ineficiencias ya que permite ejecutar varios sistemas operativos en una máquina física como máquinas virtuales, con acceso a los recursos de todos. También permite generar un cluster de servidores en un único recurso para así mejorar mucho más la eficiencia y la reducción de costes. También permite el aumento de rendimiento de las aplicaciones, la disponibilidad al aumentar la velocidad en la carga de trabajo.

2.1.1.2. Virtualización de escritorios

La implementación de escritorios virtualizados permite ofrecer a las sucursales o empleados externos... de forma rápida y sencilla un entorno de trabajo y una reducción de la inversión a la hora de gestionar cambios en estos.

2.1.1.3. Virtualización de red

Se trata de reproducir una red completa física mediante software, para poder ejecutar los mismo servicios que una red convencional y dispositivos. Cuentan con las misma características y garantías que las redes físicas con las ventajas que nos ofrece la virtualización más la liberación del hardware.

2.1.1.4. Almacenamiento definido por software

La virtualización del almacenamiento permite prescindir de los discos de los servidores, los combina en depósitos de almacenamiento de alto rendimiento y los distribuye como software. Este nuevo modelo es permite aumentar la eficiencia en el guardado de datos.

2.1.2. Ventajas de la virtualización

Como se ha podido apreciar en los tipos de virtualización, esta conlleva una mejora considerable tanto en el rendimiento, agilidad, flexibilidad, escalabilidad... Como en una reducción de costes considerables tanto en tiempo como monetarios y una simplificación en la gestión de la infraestructura.

- Reduce los costes de capital y los gastos operativos.
- Minimiza o elimina los tiempos de inactividad.
- Aumenta la productividad, la eficiencia, la agilidad y la capacidad de respuesta.
- Implementa aplicaciones y recursos con más rapidez.
- Garantiza la continuidad del negocio y la recuperación ante desastres.
- Simplifica la gestión del centro de datos.

2.2. ¿Qué es Docker?

Docker es un proyecto Open Source basado en contenedores de Linux, es básicamente un motor de contenedores que usa características del Kernel de Linux.

La idea detrás de Docker es crear contenedores ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues.

De una manera más sencilla Docker los que nos proporciona es la opción de poder meter en pequeños contenedores todo aquello que nuestra aplicación necesite y poder desplegarla en cualquier maquina que tenga instalado Docker sin preocuparnos de nada más.

Se podría decir que son pequeñas “máquinas virtuales” pero muchos más ligeras ya que utilizan el sistema operativo de donde se ejecuta y el contenido relevante para ejecutar la aplicación está dentro de los contenedores.

Docker es:

- Open-Source para la gestión de ”virtualización de contenedores”
- Aísla múltiples sistemas de archivos en el interior del mismo host
  - Las instancias se llaman Contenedores
  - Te dan la ilusión de estar dentro de una máquina virtual
- Piensa en entornos de ejecución o ”sandboxes”
- No hay necesidad de un hypervisor (rápido de ejecutar)
- Requiere x64 y Linux kernel 3.8+

Docker no es:

- Un lenguaje de programación
- Un sistema operativo
- Una máquina virtual
- Una imagen en el concepto tradicional de la máquina virtual basada en hipervisor

2.3. Máquinas Virtuales vs Docker

Las máquinas virtuales, incluyen toda la aplicación, los binarios y librerías necesarias, y todo un sistema operativo cosa que hace que que ocupen mucho espacio, el tiempo de ejecución sea lento, la necesidad de un Hipervisor para su utilización.

Por lo contrario Docker container incluyen la aplicación y todas sus dependencias pero comparten el núcleo con otros contenedores, funcionando como procesos aislados en el sistema de ficheros del sistema operativo. Docker container no están vinculados a ninguna infraestructura específica: se ejecutan en cualquier ordenador, en cualquier infraestructura, y en cualquier cloud.

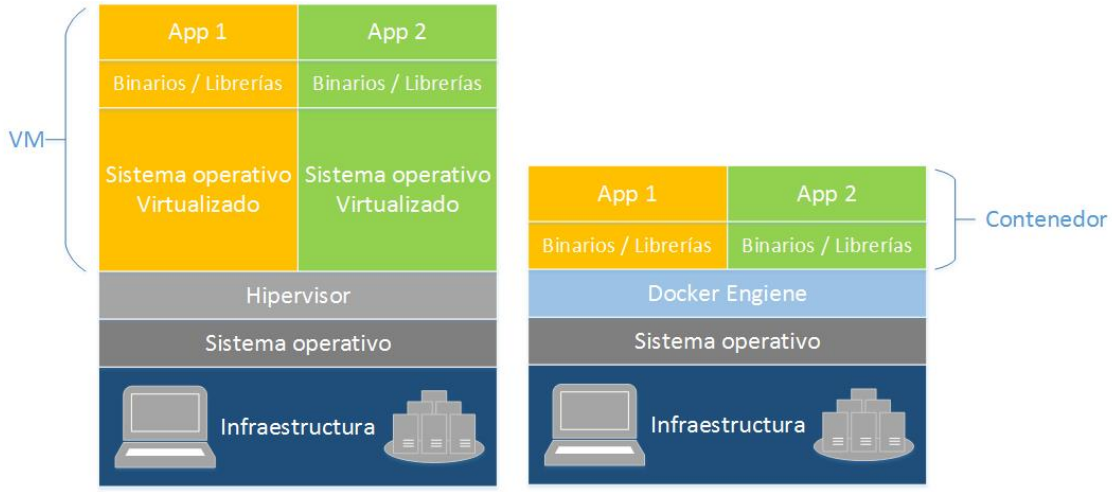

Figura 2.1: Comparativa de máquina virtual y Docker

En la imagen 2.1 se puede ver las diferencias comentadas anteriormente, mientras la primera columna sería la virtualización de 2 aplicaciones se puede dectectar que está la capa intermedia del Hipervisor el cual nos permite ejecutar los sistemas operativos virtualizados, todos los binarios y librerías que requieren cada aplicación y finalmente la aplicación. En la segunda columna tenemos la contenerización de 2 aplicaciones, podemos observar que no hace falta un hipervisor ya que utilizan el sistema operativo de la infraestructura donde se ejecuta, pero si son necesarios los binarios y las librerías. A simple vista se puede apreciar que si eliminamos el sistema operativo la infraestructura completa se hace más liviana y rápida de ejecutar.

2.4. ¿Por qué Docker?

Por lo motivos listados en el apartados anteriores se decidió en utilizar Docker para realizar el despliegue de las aplicaciones, ya que se necesita de una tecnología la cual pueda ser almacenada en dispositivos con una memoria reducida en este caso docker la cumple.

Su rapidez a la hora de levantar el servicio, en el mundo de IoT el tiempo es un bien preciado y los sistemas se pueden apagar y encender constantemente para evitar gastos de energía innecesarios.

La facilidad para poder desplegar los servicios, con Docker desplegar servicios es muy sencillo solo requiere tenerlo instalado y ejecutar el contenedor para que el servicio esté activo.

La sencillez a la hora de mantener el sistema, si hay que hacer actualizaciones o controles de una pequeña parte del servicio solo habria que cambiar o actualizar ese contenedor no todo el sistema. Esto es un gran ahorro en recursos y tiempo.

Por todo esto se piensa que Docker puede ser una buena tecnología para desplegar sistemas de Iot. También se tiene en cuenta que la utilización de una Raspberry 3 no es un aparato con unas capacidades limitadas como podría ser un sensor utilizado normalmente pero si que puede validar todas las funciones listadas anteriormente y servir como preámbulo para la utilización en el resto de despliegues.

2.5. ¿Cómo funciona Docker?

En este punto se explicara brevemente aspectos de la arquitectura, funcionamiento y puesta en marcha de Docker.

2.5.1. Arquitectura

Docker usa una arquitectura cliente-servidor. El cliente de Docker se comunica con el Daemon de Docker para crear, ejecutar y distribuir los contenedores. El cliente como el Daemon puede estar en el mismo sistema o poder conectarte remotamente. Como Docker usa el kernel de linux para su ejecución si el sistema operativo del sistema no es este se deberá usar una pequeña capa extra en la arquitectura la cual sería una VM (boot Docker) para poder correr docker en la máquina.

2.5.1.1. Cliente de Docker

Es la principal interfaz de usuario para docker, acepta los comandos del usuario y se comunica con el daemon de docker.

2.5.1.2. Imágenes de Docker (Docker Images)

Las imágenes de Docker son plantillas de solo lectura, que nos permitirá crear contenedores basados en su configuración.

2.5.1.3. Registros de Docker (Docker Registries)

Los registros de Docker guardan las imágenes, estos son repos públicos o privados donde podemos subir o descargar imágenes. Sería similar a GitHub para imágenes de docker (Docker Hub).

2.5.1.4. Contenedores de Docker (Docker Containers)

El contenedor de docker contiene todo lo necesario para ejecutar una aplicación. Cada contenedor se crea de una imagen de docker y es una plataforma aislada.

En la figura 2.2 podemos apreciar gráficamente cómo sería la arquitectura básica.

El cliente podría hacer los comandos básicos de docker:

`Docker Build`: Hace un build de un DockerFile y generar una imagen de docker, en la figura está representado siguiendo las flechas rojas en la cual podemos ver que el cliente se comunica con el Daemon y este genera la imagen.

`Docker Pull`: El cual nos permite descargar una imagen de los repositorios de Docker, en la imagen se puede observar siguiendo las flechas verdes. Que igual que el comando anterior se comunica el cliente con el Daemon para que este proceda a hacer la descarga de la imagen del repositorio.

`Docker Run`: Ejecuta una imagen para generar un contendor de esta. Si seguimos las flechas azules, veremos como nuevamente el cliente al ejecutar esa comanda se comunica con el Daemon este busca la imagen a la cual se le quiere hacer ejecutar y se levanta un contenedor con la configuración de la imagen.

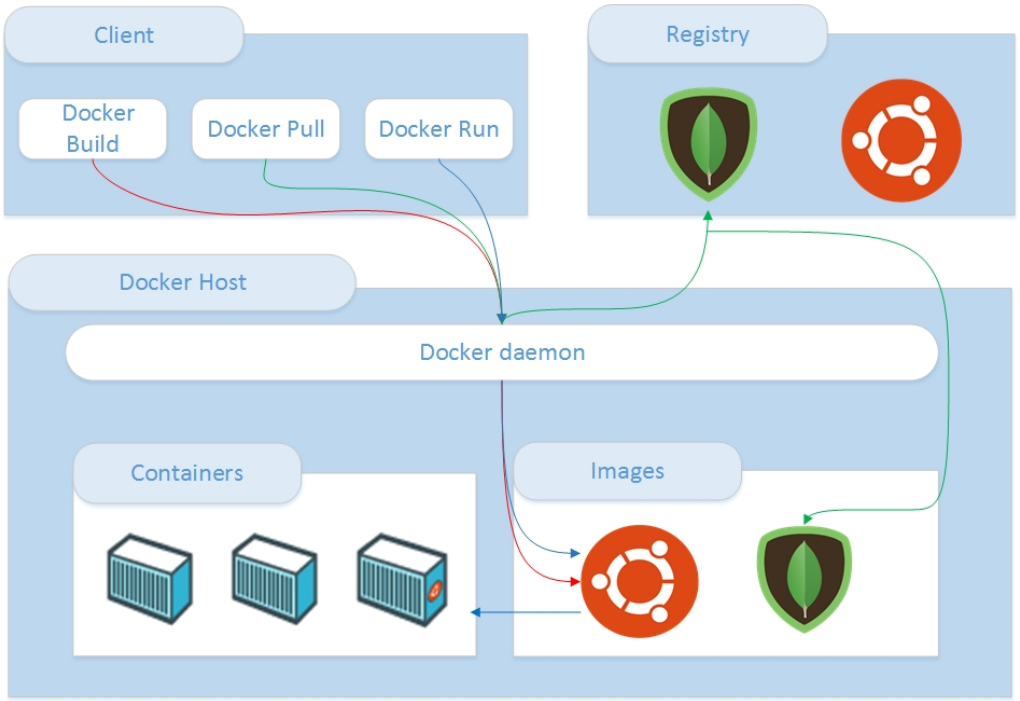

Figura 2.2: Infraestructura de Docker

2.5.2. Creación de Imágenes

Como se comentó en el punto anterior las imágenes de docker son las plantillas para poder levantar los contenedores, por eso la importancia de saber crear imágenes y personalizarlas ya que solo permiten lectura y los cambios que hagamos en los contenedores no se verán reflejados en estas.

La manera más sencilla de crear una imagen es descargarla del Docker Hub con el comando explicado con anterioridad:

```
docker pull [OPTIONS] NAME[:TAG|@DIGEST]
```

Este comando nos permite descargar una imagen en una versión concreta o tag dependiendo de nuestras necesidades, por defecto si no se pone nada descargara la ultima.

(a) Docker pull

(b) Docker images

Figura 2.3: Comandos de docker (I)

En las figuras 2.3 podemos apreciar como se descarga la última versión de la imagen de mongo y nos genera la imagen.

Una vez tenemos la imagen se pasara a levantar el contenedor para poder ejecutar el servicio, con otro de los comandos explicados.

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

```
alfredo@alfredo:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
alfredo@alfredo:~$
```

(a) Docker ps

```
alfredo@alfredo:~$ docker run mongo
2016-09-19T14:54:09.623+0000 I CONTROL  [initandlisten] MongoDB starting : pid=1 port=27017 dbpath=/data/db 64-bit host=d62f7dce38c2
2016-09-19T14:54:09.623+0000 I CONTROL  [initandlisten] db version v3.2.9
2016-09-19T14:54:09.623+0000 I CONTROL  [initandlisten] git version: 22ec9e93b40c85fc7cae7d56e7d6a02fd811088c
2016-09-19T14:54:09.623+0000 I CONTROL  [initandlisten] OpenSSL version: OpenSSL 1.0.1t  3 May 2016
2016-09-19T14:54:09.623+0000 I CONTROL  [initandlisten] allocator: tcmalloc
2016-09-19T14:54:09.623+0000 I CONTROL  [initandlisten] modules: none
2016-09-19T14:54:09.623+0000 I CONTROL  [initandlisten] build environment:
2016-09-19T14:54:09.623+0000 I CONTROL  [initandlisten]     distmod: debian81
2016-09-19T14:54:09.623+0000 I CONTROL  [initandlisten]     distarch: x86_64
2016-09-19T14:54:09.623+0000 I CONTROL  [initandlisten]     target_arch: x86_64
2016-09-19T14:54:09.623+0000 I CONTROL  [initandlisten] options: {}
2016-09-19T14:54:09.631+0000 I STORAGE  [initandlisten] wiredtiger_open config: create,cache_size=3G,session_max=20000,eviction=(threads_max=4)
,config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),check
point=(wait=60,log_size=2GB),statistics_log=(wait=0),
2016-09-19T14:54:10.421+0000 I CONTROL  [initandlisten]
2016-09-19T14:54:10.421+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2016-09-19T14:54:10.421+0000 I CONTROL  [initandlisten] **        We suggest setting it to 'never'
2016-09-19T14:54:10.421+0000 I CONTROL  [initandlisten]
2016-09-19T14:54:10.421+0000 I CONTROL  [initandlisten] ** WARNING: /sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2016-09-19T14:54:10.421+0000 I CONTROL  [initandlisten] **        We suggest setting it to 'never'
2016-09-19T14:54:10.421+0000 I CONTROL  [initandlisten]
2016-09-19T14:54:10.422+0000 I FTDC     [initandlisten] Initializing full-time diagnostic data capture with directory '/data/db/diagnostic.data
'
2016-09-19T14:54:10.422+0000 I NETWORK  [HostnameCanonicalizationWorker] Starting hostname canonicalization worker
2016-09-19T14:54:10.692+0000 I NETWORK  [initandlisten] waiting for connections on port 27017
```

(b) Docker run

```
alfredo@alfredo:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
eee37a9c1602        mongo               "/entrypoint.sh mongo"   4 seconds ago       Up 3 seconds        27017/tcp           kickass_pike
alfredo@alfredo:~$
```

(c) Docker ps

Figura 2.4: Comandos de docker (II)

En las figuras 2.4 se puede apreciar cómo utilizando el comando `docker ps` que nos permite ver que contenedores están levantados,no hay ninguno (a) y al inicializar con el comando `docker run mongo` nos levanta el servicio (b) y esta vez si aparece el contenedor (c).

La segunda manera de crear y personalizar imágenes es mediante un `DockerFile`, que es un documento de texto donde se encuentra todos los comandos que se deben ejecutar para generar nuestra imagen.

El comando `docker build` comunica al daemon de docker que debe de leer el `DockerFile` del directorio actual y seguir las instrucciones línea por línea para la creación de nuestra imagen. Este proceso se va pintando los resultados por pantalla y generando imágenes intermedias para obtener así una cache que nos permitirá en caso de errores, una vez corregido el DockerFile, continuar desde el punto conflictivo y no compilarlo todo de nuevo.

```
FROM docker/whalesay:latest
RUN apt-get -y update && apt-get install -y fortunes
CMD echo "Proyecto CoIoTe" | cowsay
```

Aquí tenemos un ejemplo sencillo de DockerFile que nos servirá para explicar de una manera rápida como crearlos.

`FROM` indica la imágen base que va a utilizar para seguir futuras instrucciones. Buscará si la imagen se encuentra localmente, en caso de que no, la descargará de internet. En nuestro ejemplo utiliza la última versión de la imagen docker/whalesay.

La instrucción `CMD` solo puede aparecer una vez en un DockerFile, si colocamos más de uno, solo el último tendrá efecto. El objetivo de esta instrucción es proveer valores por defecto a un contenedor. Estos valores pueden incluir un ejecutable u omitir un ejecutable

que en dado caso se debe especificar un punto de entrada o entrypoint en las instrucciones. En nuestro caso pintaremos un texto que se le pasara a la aplicación cowsay.

Existen muchas más instrucciones, alguna de estas serán explicadas más adelante cuando sea necesario su uso.

Una vez ejecutado el comando `docker build -t docker-whale .` donde el `-t` nos permite darle nombre a la imagen y el `.` encontrar el DockerFile para ser compilado. Como se ve en la Figura 2.5 descarga del repositorio la imagen ya que no la tenemos en local creará una imagen intermedia que nos proporciona la caché en caso de fallo, continuará con la siguiente orden creando una nueva imagen nueva borrando las anteriores hasta obtener la imagen definitiva.

```
alfredo@alfredo:~/mydockerbuild$ sudo docker build -t docker-whale .
Sending build context to Docker daemon 3.072 kB
Step 1 : FROM docker/whalesay:latest
latest: Pulling from docker/whalesay
e190868d63f8: Pull complete
909cd34c6fd7: Pull complete
0b9bfabab7c1: Pull complete
a3ed95caeb02: Pull complete
00bf65475aba: Pull complete
c57b6bcc83e3: Pull complete
8978f6879e2f: Pull complete
8eed3712d2cf: Pull complete
Digest: sha256:178598e51a26abbc958b8a2e48825c90bc22e641de3d31e18aaf55f3258ba93b
Status: Downloaded newer image for docker/whalesay:latest
 ---> 6b362a9f73eb
Step 2 : CMD echo "Proyecto CoIoTe" | cowsay
 ---> Running in 598204c0e3a3
 ---> ac80256777e9
Removing intermediate container 598204c0e3a3
Successfully built ac80256777e9
```

Figura 2.5: Build del DockerFile

Una vez hecho el build del DockerFile podemos comprobar que nuestra imagen está creada correctamente (Figura 2.6) y pasaremos a levantar el contenedor. Una vez levantado nos saldrá por pantalla el icono de docker “diciendo” la frase que le indicamos en el fichero. (Figura 2.7)

```
Successfully built ac80256777e9
alfredo@alfredo:~/mydockerbuild$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
docker-whale        latest              ac80256777e9        About a minute ago   247 MB
mongo               latest              48b8b08dca4d        2 weeks ago          366.4 MB
docker/whalesay     latest              6b362a9f73eb        16 months ago        247 MB
alfredo@alfredo:~/mydockerbuild$
```

Figura 2.6: Docker images

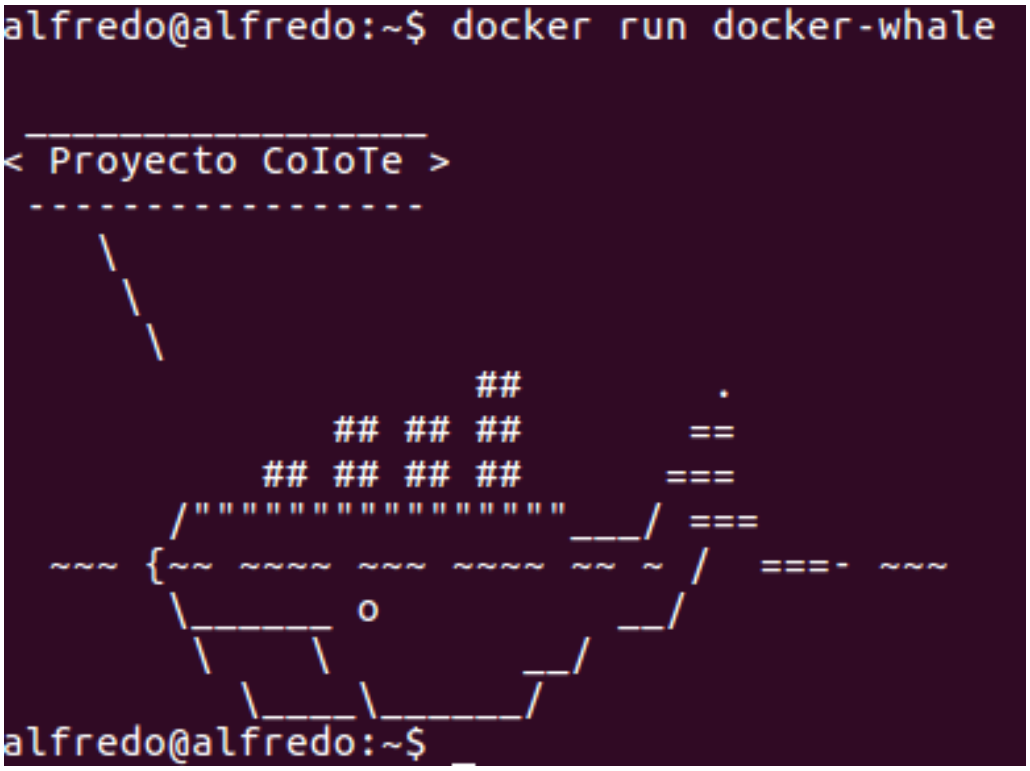

Figura 2.7: Docker run docker-whale]

2.5.3. Docker Compose

Docker Compose es un orquestador que nos permite ejecutar aplicaciones que utilicen varios contenedores a la vez. Se creará un archivo docker-compose.yml donde se configurar todos los servicios necesarios para nuestra aplicación. Una vez ejecutado este archivo nos generará todas las imágenes y con estas los contenedores especificados a la vez que arrancará la aplicación.

Los comandos que se utilizarán serán similares a los utilizados en la creación de imágenes.

```
docker-compose up
```

Permite ejecutar el servicio y se nos levante todos los contenedores.

```
docker-compose stop
```

Detendrá el servicio y detener los contenedores.

Docker-compose también nos permite ejecutar solo partes del.yml para levantar solo algunos de los contenedores igual que detener solo algunos de ellos, pasándole por parámetro el nombre del contenedor.

```
version: '2'
services:
 mongo:
  container_name: mongo
  restart: always
  image: partlab/ubuntu-arm-mongodb
  volumes:
   - mongo-data:/data/db
  command: /usr/bin/mongod --smallfiles --journal
 aaaida:
  container_name: aaaidaArm
  restart: always
  image: alteraid/aaaida-datastore-arm
  links:
   - mongo:mongo
  environment:
   - NODE_ENV=docker
  ports:
   - "40000:40000"
volumes:
 mongo-data:
   driver: local
```

CONCLUSIONS

Escriure aquí les conclusions del projecte.

BIBLIOGRAFÍA

[1] Cognoms-autor, Inicial-nom. “Títol del capítol”. *Títol del llibre*. (Editor. Ciutat. Any publicació): pagina1–paginaN.

[2] Cognoms-autor, Inicial-nom. “Títol de l’article”. *Títol de la revista*. **volum**(numero), pagina1–paginaN. (Any publicació)

APÉNDICES

APÉNDICE A. EXEMPLE DE PROVA D'UN APÈNDIX

Text de prova