INFORME SEGUNDO PARCIAL
SISTEMAS DISTRIBUIDOS

Jaime Andres Aristizabal Lozada
11104055

1. Realice una consulta en internet sobre las recomendaciones al aprovisionar empleando contenedores. Debe incluir referencias

   Dentro de la página de Docker, se encontró las recomendaciones que ellos sugerían al momento de realizar aprovisionamientos con contenedores. Estas recomendaciones son:

   - *Los contenedores deben ser efímeros*: Con esto se refieren a que los contenedores deben ser capaces de ser iniciados y detenidos en cualquier momento, garantizando que se puedan volver a un estado inicial. Deben ser maquinas fáciles de iniciar y detener.
   - *Usar un archivo .dockerignore*: Lo recomendable es que donde se encuentre el DockerFile sea un directorio vacío, donde este solo contenga lo necesario para compilar el contenedor. Si se llega el caso que no sea posible ubicarlo en un directorio vacío, se puede hacer un archivo con extensión .dockerignore, con el objetivo que este ignore los directorios puestos en este archivo. Esto se hace para que al momento de la compilación, docker lo haga lo mas eficiente posible.
   - *Evitar instalar paquetes innecesarios*: Para reducir la complejidad en las dependencias, el tamaño de los archivos y los tiempos de compilacion, se deben de evitar instalar paquetes que no son necesarios en los servicios que vaya a ofrecer el contenedor.
   - *Correr un solo proceso por contenedor*: En casi todos los casos, solo se debe ejecutar un solo proceso por cada contenedor. El desacoplamiento de las aplicaciones y la escalabilidad tanto horizontal como vertical se facilita con utilizar un solo proceso por contenedor. Si es necesario que un proceso de un contenedor se enlace con otro proceso, se puede utilizar la herramienta container linking.
   - *Minimizar el numero de capas*: Hay que encontrar un balance entre la legibilidad y el mantenimiento de un DockerFile, por lo tanto hay que ser estrategico al utilizar una cantidad de capas sobre el sistema.
   - *Organizar argumentos en varias lineas*: Organice de forma alfanumerica las dependencias que se vayan a instalar, esto hara que sea mas facil la revision al momento del control.

2. La realizacion del proyecto se planteo sobre dos contenedores, y un docker compose. El primer contenedor fue el encargado de almacenar y correr el micro framework de CherryPy, y el segundo contenedor almaceno el motor de base de datos en PostgreSQL.

Contenedor CherryPy:

Este contenedor como se hablo anteriormente, va a contener el micro framework de CherryPy. La estructura de carpetas que se encuentra en este contenedor es la siguiente:

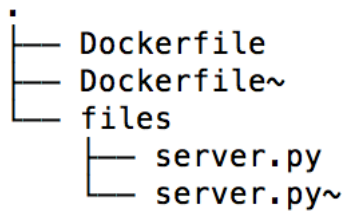


Figura 1, Estructura de carpetas contendor Cherry

Lo que contiene esta carpeta es principalmente el Dockerfile que va a permitir aprovisionar la maquina, mas adelante se mostrara el contenido. El otro archivo que se encuentra es el script del servidor de consulta hacia la base de datos, este script tiene el siguiente contenido:

```
import cherrypy
import psycopg2
import pprint

class DatabaseConsult(object):
    @cherrypy.expose
    def index(self):
        host = "host="+'192.168.130.124'+" "
        port = "port="+str(5432)+" "
        db = "dbname="+'swn'+" "
        user = "user="+'pi'+" "
        passwd = "password="+'security++'+" "

        conn_string = host+port+db+user+passwd
        conn = psycopg2.connect(conn_string)
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM devices")
        records = cursor.fetchall()
        #pprint.pprint(records)
        #print((records['device_name']))
        return "This are database values"+" id_divice: "+str(records[0][0]) + " name_device: "+str(records[0][1])

    cherrypy.config.update({'server.socket_host': '0.0.0.0', 'server.socket_port': 5000})
if __name__ == '__main__':
    cherrypy.quickstart(DatabaseConsult())
```

Figura 2, Contenido del script de Server.py

Lo que realiza este script se divide en dos etapas, levantar el servicio de consulta y realizar la consulta de la base de datos. El servicio se levanta sobre la ip local de la maquina y la consulta tambien se realiza sobre la direccion de la maquina, pero esta direccion tiene que colocarse como aparece fisicamente la ip. Asi mismo el Dockerfile tiene el siguiente contenido:

```
FROM ubuntu:14.04
RUN apt-get update && apt-get install -y python python-pip python-cherrypy python-psycopg2
RUN apt-get install python3-pip -y
RUN useradd -ms /bin/bash cherry
RUN apt-get remove python-cherrypy -y
RUN pip install cherrypy
USER cherry
WORKDIR /home/cherry
ADD ./files/server.py /home/cherry
EXPOSE 5000
EXPOSE 5432
USER root
CMD ["python", "server.py"]
```

Figura 3, Contenido del Dockerfile del contenedor Cherry

Este archivo instala las librerias necesarias para cherrypy y las librerias de consulta de la base de datos, como es psycopg. Ademas agrega un usuario cherry, el cual almacenara el script server.py en la direccion /home/. Ademas se exponen dos puertos, el puerto de consulta al servicio de consulta de la base de datos y el puerto del motor de la base de datos. Por ultimo, cuando se ejecute el contendor quedara ejecutando el script server.py el cual permitira ver cuales son las consultas HTTP que se estan realizando.

Contenedor PostgreSQL:

Como se menciono en la introduccion, este contenedor almacena y ejecuta el motor de la base de datos PostgreSQL. La estructura de carpetas de este contenedor se presenta en la figura 4.

```
.
├── Dockerfile
├── Dockerfile~
└── files
    ├── configure_postgresql.sh
    ├── create_schema.sql
    ├── pg_hba.conf
    ├── pg_hba.conf~
    ├── postgresql.conf
    └── postgresql.conf~
```

Figura 4, Estructura de carpetas de PostgreSQL

Esta estructura contempla la misma estructura del primer parcial. Contiene los archivos de configuracion de postgres, que busca exponer en el puerto 5432 el servicio, permitir conecciones por fuera de la maquina y crea el esquema de la base de datos. El contenido del create_schema se presenta en la figura 5, la cual muestra como se estructura la tabla, el usuario y la contraseña de esta tabla.

```
--CREATE EXTENSION postgis;
create user pi with password 'security++';
alter role pi with createdb;
create database swn owner pi;
\connect swn
CREATE TABLE devices(
       id     integer NOT NULL PRIMARY KEY,
       device_name  varchar(10) NOT NULL UNIQUE
);
grant all privileges on table devices to pi;
CREATE TABLE consumption(
       id   integer NOT NULL PRIMARY KEY,
       date date  NOT NULL,
       device_id integer references devices(id)
);
INSERT INTO devices
(id, device_name)
VALUES
(15 , 'express');
```

Figura 5, contenido del script de la tabla de la base de datos

Este script deja expuesto una tabla que almacena nombre de dispositivos y un id que hara referencia a que dispositivo se esta consultando. Los scripts de configuracion no se presentan en el documento porque no se ve la necesidad, ya que es igual al trabajo realizado en la practica de Vagrant. Sin embargo, se presenta la estructura del Dockerfile la cual levanto el contenedor de Postgres. La figura 6 presenta la estructura del Dockerfile.

```
FROM ubuntu:14.04
RUN apt-get update
RUN apt-get install -y python python-pip postgresql postgresql-contrib wget
RUN useradd -ms /bin/bash admin
COPY ./files/create_schema.sql /tmp/create_schema.sql
USER postgres
RUN service postgresql start && \
        sleep 40 &&\
        cat /tmp/create_schema.sql | psql
COPY ./files/pg_hba.conf /etc/postgresql/9.3/main/pg_hba.conf
COPY ./files/postgresql.conf /etc/postgresql/9.3/main/postgresql.conf
EXPOSE 80
EXPOSE 5432
USER postgres
VOLUME ["/etc/postgresql", "/var/log/postgresql", "/var/lib/postgresql"]
CMD ["/usr/lib/postgresql/9.3/bin/postgres", "-D", "/var/lib/postgresql/9.3/main",
        "-c", "config_file=/etc/postgresql/9.3/main/postgresql.conf"]
```

Figura 6, Contenido del Dockerfile de PostgreSQL

Esta maquina se levanta sobre un ubuntu 14.04, donde se instalan librerias necesarias para tener el motor de base de datos PostgreSQL. Algo que se tiene que tener en cuenta, es cuando se pase el schema. Tiene que necesariamente hacerse por medio de un cat, corriendo el servicio de postgresql, y hay que evitar que se detenga, por lo tanto se deja un sleep de 40 segundos. Esto es para poder

copiar el schema de la tabla de la base de datos cuando se encuentra ejecutando el postgres como servicio. Estos cambios tienen que realizarse por medio del usuario postgres. Asi mismo tambien se copian los archivos de configuracion y se exponen los puertos de consulta de la base de datos. Por ultimo, se ejecuta sobre bash postgresql y se cambia el archivo de configuracion por el que se ha pasado desde el aprovisionamiento.

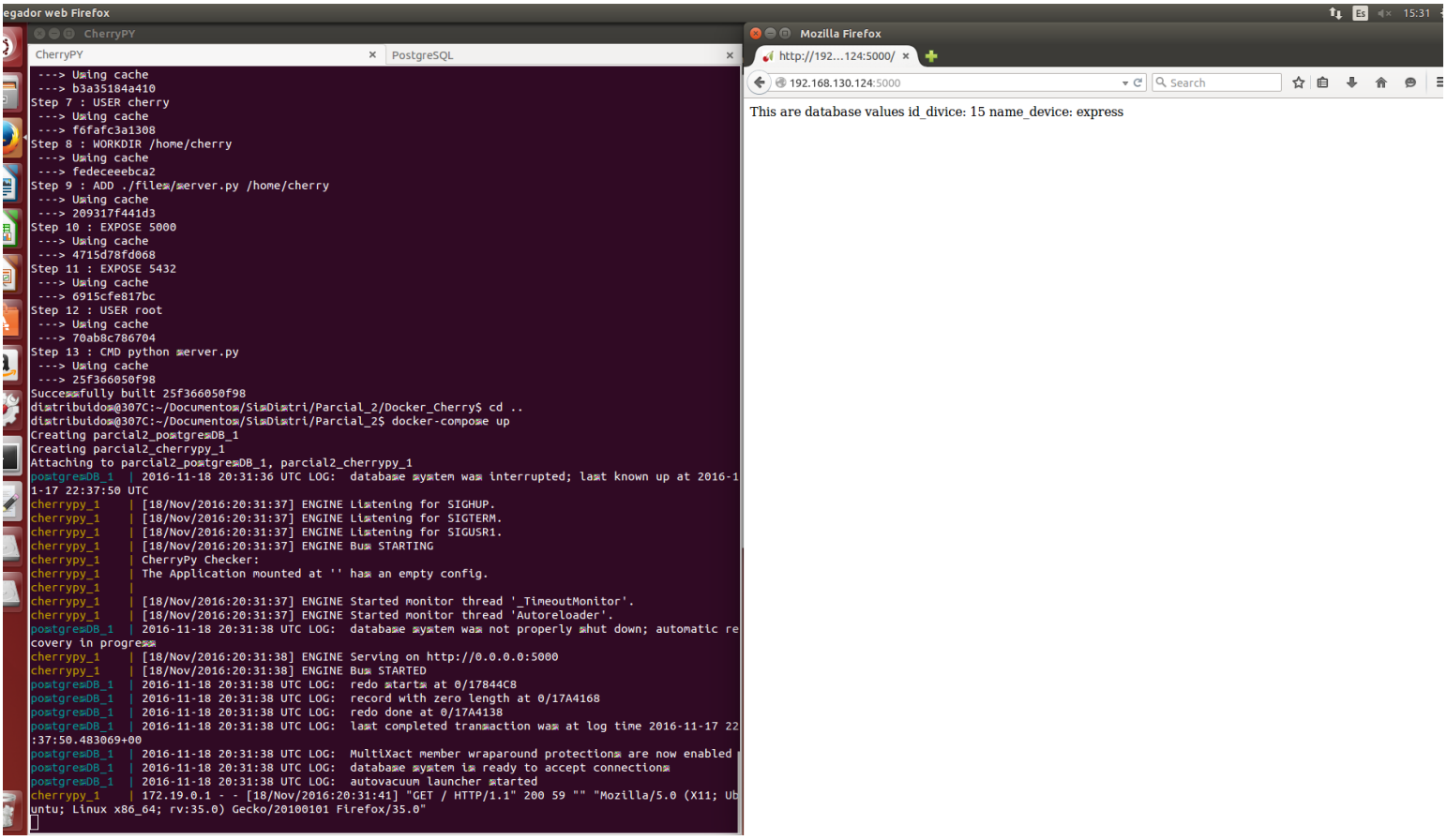
Docker-Compose:

Antes de pasar por este paso, es necesario compilar los contenedores anteriores con los nombres cherry y postgres, para que docker los tenga como imágenes. Cuando ya se tengan compilados estos dos contenedores se pasa a levantar el docker compose. La figura 7 muestra la configuracion del docker-compose.

```
version: '2'
services:
  cherrypy:
    image: cherry:latest
    ports:
     - "5000:5000"
    links:
     - postgresDB
    depends_on:
     - postgresDB
  postgresDB:
    image: postgresql:latest
    ports:
     - "5432:5432"
```

Figura 7, Configuracion del docker-compose.yml

La funcion del docker compose es levantar las dos maquinas y fijar las dependencias entre ellas. Deja ademas expuestos los puertos por donde se exponen los servicios. Hay algo importante que se tiene que tener en cuenta, es que cuando se levantan las dos maquinas con este mecanismos sin especificarles las direcicones ip, van a levantarse sobre la direccion 0.0.0.0 que es la misma direccion fisica de la maquina anfitriona. Estos es importante porque al momento de la consulta es necesaria hacerla con la direccion de la maquina anfritiona. El resultado de la consulta de la base de datos y su respectivo funcionamiento de los dos contenedores se presenta a continuacion en la figura 8.


egador web Firefox
CherryPY
CherryPY
PostgreSQL
---> Using cache
---> b3a35184a410
Step 7 : USER cherry
---> Using cache
---> f6fafc3a1308
Step 8 : WORKDIR /home/cherry
---> Using cache
---> fedeceeebca2
Step 9 : ADD ./files/server.py /home/cherry
---> Using cache
---> 209317f441d3
Step 10 : EXPOSE 5000
---> Using cache
---> 4715d78fd068
Step 11 : EXPOSE 5432
---> Using cache
---> 6915cfe817bc
Step 12 : USER root
---> Using cache
---> 70ab8c786704
Step 13 : CMD python server.py
---> Using cache
---> 25f366050f98
Successfully built 25f366050f98
distribuidos@307C:~/Documentos/SisDistri/Parcial_2/Docker_Cherry$ cd ..
distribuidos@307C:~/Documentos/SisDistri/Parcial_2$ docker-compose up
Creating parcial2_postgresDB_1
Creating parcial2_cherrypy_1
Attaching to parcial2_postgresDB_1, parcial2_cherrypy_1
postgresDB_1 | 2016-11-18 20:31:36 UTC LOG: database system was interrupted; last known up at 2016-11-17 22:37:50 UTC
cherrypy_1 | [18/Nov/2016:20:31:37] ENGINE Listening for SIGHUP.
cherrypy_1 | [18/Nov/2016:20:31:37] ENGINE Listening for SIGTERM.
cherrypy_1 | [18/Nov/2016:20:31:37] ENGINE Listening for SIGUSR1.
cherrypy_1 | [18/Nov/2016:20:31:37] ENGINE Bus STARTING
cherrypy_1 | CherryPy Checker:
cherrypy_1 | The Application mounted at '' has an empty config.
cherrypy_1 |
cherrypy_1 | [18/Nov/2016:20:31:37] ENGINE Started monitor thread '_TimeoutMonitor'.
cherrypy_1 | [18/Nov/2016:20:31:37] ENGINE Started monitor thread 'Autoreloader'.
postgresDB_1 | 2016-11-18 20:31:38 UTC LOG: database system was not properly shut down; automatic recovery in progress
cherrypy_1 | [18/Nov/2016:20:31:38] ENGINE Serving on http://0.0.0.0:5000
cherrypy_1 | [18/Nov/2016:20:31:38] ENGINE Bus STARTED
postgresDB_1 | 2016-11-18 20:31:38 UTC LOG: redo starts at 0/17844C8
postgresDB_1 | 2016-11-18 20:31:38 UTC LOG: record with zero length at 0/17A4168
postgresDB_1 | 2016-11-18 20:31:38 UTC LOG: redo done at 0/17A4138
postgresDB_1 | 2016-11-18 20:31:38 UTC LOG: last completed transaction was at log time 2016-11-17 22:37:50.483069+00
postgresDB_1 | 2016-11-18 20:31:38 UTC LOG: MultiXact member wraparound protections are now enabled
postgresDB_1 | 2016-11-18 20:31:38 UTC LOG: database system is ready to accept connections
postgresDB_1 | 2016-11-18 20:31:38 UTC LOG: autovacuum launcher started
cherrypy_1 | 172.19.0.1 - - [18/Nov/2016:20:31:41] "GET / HTTP/1.1" 200 59 "" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:35.0) Gecko/20100101 Firefox/35.0"
Mozilla Firefox
http://192...124:5000/
192.168.130.124:5000
Search
This are database values id_divice: 15 name_device: express