Dirección de Desarrollo de Talento
Gerencia de Desarrollo de Capital Humano
Unidad Operativa de Servicio Social y Prácticas Profesionales

REPORTE 0 (2 de 2) Reporte global de MyTardis (¿Qué es?, componentes, resolución de bugs de implementación y posibles servicios donde implementar la aplicación).

Fecha 13 de septiembre de 2022

Nombre: Villegas Sánchez Alexis

Reporte No. 6 (seis) Correspondiente al periodo: Agosto/Septiembre

Escuela: Escuela Superior de Cómputo Carrera: Sistemas computacionales

Modalidad: Servicio Social

Contenido

1. ¿Qué es MyTardis?

La aplicación MyTardis trata de resolver el problema de los usuarios de almacenar una gran cantidad de datos y compartirlos con colaboradores. Este software encuentra enfocada principalmente en la integración de instrumentos científicos, su instalación, la infraestructura informática y el almacenamiento de una investigación; abordado los retos de almacenamiento, su acceso, colaboración y publicación.

Su implementación es mediante el lenguaje de programación Python con ayuda del framework para desarrollo web Django. A su vez se apoya de un índex de elasticsearch , una cola de tareas basada en RabbitMQ para realizar búsquedas más rápidas, un servidor ngix y una base de datos en PostgreSQL. Además, para crear y manejar este servicio se emplearon kubernetes y tecnologías de la nube con el propósito de poder brindar una mayor seguridad y escalabilidad.

En cuanto al código fuente, se encuentra almacenado en el repositorio de GitHub y cuenta con varias ramas, donde las de mayor importancia son las que llevan por nombre ***DEVELOP*** y ***MASTER***. Sin embargo, existen limitantes en cuanto la versión implementada en la rama **develop** ya que se recomienda que esta nunca debe de ser implementada debido a que esta rama es la que está abierta para realizar modificaciones dependiendo de los propósitos específicos de cada desarrollador además de que cuenta con varios *bugs* que se comen la información que se le almacena.

Por otra parte la rama de ***master*** es la versión estable que incluye el arreglo de *bugs* de versiones anteriores y es la que se recomienda para el entorno de producción.

Por supuesto que, en caso de ser necesaria una versión en específico, se cuentan con las ramas que llevan, literalmente, el nombre de la versión por ejemplo *4.6.1*.

Las características de la aplicación son las siguientes:

- La aplicación requiere que los usuarios estén cuenten con una cuenta para poder acceder.
- La aplicación cuenta con sistema de archivos que va de la jerarquía de *experimentos*, *data set* y archivos de datos o *data file*. Donde los *data set* deben tener al menos un *experimento* padre
- Los usuarios pueden seleccionar (de su ordenador creo yo) elegir el archivo de datos a subir a un *experimento* o *data set*. Así como descargar un *archivo de datos*, un *data set* o experimento (para ello es necesariamente entrar a lo que se quiera descargar y se mostrará los *data set* o *archivo de datos* contenidos; estos se descargan en formato .tar (similar a ZIP) sin embargo para los usuarios de Windows existe problemas para manejar este tipo de archivos por lo que es necesario instalar software especializado como 7-zip.
- Los usuarios pueden publicar o compartir sus experimentos con otros usuarios o con un grupo para que estos tengan acceso a los experimentos. Otra manera es que se haga mediante links temporales (por un mes) para compartir los *experimentos* de manera “privada”
- Al momento en que un usuario quiera compartir sus bancos de datos con otro deberá ingresar ya sea el nombre o correo del usuario al que se quiere compartir y se desplegará una serie de posibles usuarios a los cuales compartir. Además de poder otorgar ciertos privilegios para poder manipular el *experimento* (solamente lectura, lectura y edición, etc.).

- Una vez que se compartan con otros usuarios, el dueño del *experimento* es capaz de editar el contacto al que se le comparte el *experimento* así como negarles el acceso (borrarlos para que dejen de ver el *experimento*)
- Los links temporales que se pueden generar se pueden crear en cualquier momento (siempre y cuando sea el dueño), son generados de tal manera que no parezca un link y todo aquel que cuente con él puede entrar. Su duración es de 1 mes. Puede ser compartido directamente vía correo. Proveen el acceso completo al recipiente a excepción de que el *experimento* sea de acceso público.
- El usuario es capaz de poner de "manera pública" sus *experimentos* ya que por defecto estos estarán privados. Estos deben de ser publicados con la licencia adecuada la cual será seleccionada por el mismo usuario además de que se le debe de dar cierta información sobre la licencia que eligió.
- Una manera segura de acceder únicamente para lectura es empleando SFTP pero para ello previamente se debió instalar un cliente SFTP. Así pues mediante SSH y las llaves públicas y privadas generadas por RSA se tenga acceso. Así pues, será necesario cargar al sistema la llave publica o el software será capaz de generar un par nuevo pero no es capaz de recuperarlo. Además de poder borrar llaves anteriormente generadas.
- Para poder instalar MyTardis es necesario Python 3 y el sistema operativo Ubuntu 18.04.

2. Componentes de software de MyTardis

Python: Este es lenguaje de programación en el cual la aplicación fue escrita para debido a su fácil sintaxis, flexibilidad y gran soporte de su comunidad a la solucion de problemas. La versión que utilizan la versión estable y de desarrollo es la versión 3.6 de Python

Django: Framework web más utilizado para Python. La cual ayuda a que la API tenga una arquitectura RESTful que permita manipular la aplicación mediante la red mostrando solo lo necesario (ver imagen 1) y ocultando las operaciones internas de la página, por medio del protocolo HTTP y el empleo de verbos HTTP como GET, POST, PUT y DELETE, ver imagen 2. La versión utilizada para la versión estable el Django 3.2.5 y la de desarrollo es la 3.2.7.

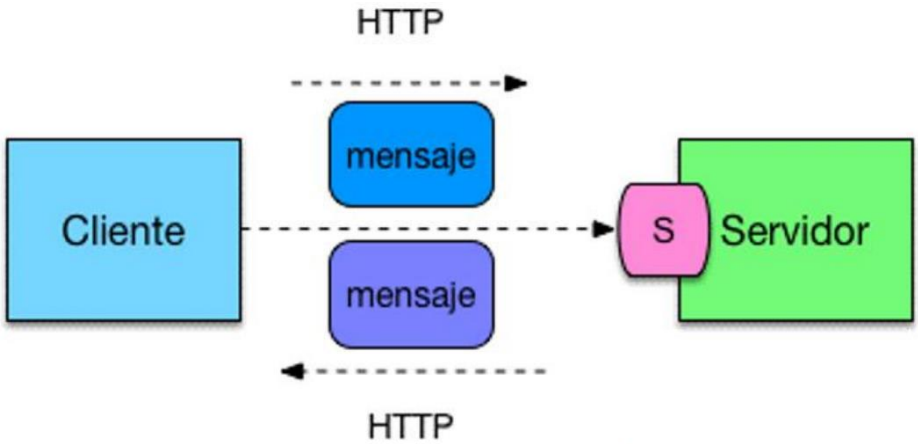


Imagen 1: Funcionamiento general de una API REST

Imagen 2: Verbos HTTP más utilizados en las API tipo RESTful

Elasticsearch: Es un motor de búsqueda que, en general, nos permitirá durante nuestra navegación en la aplicación web, realizar búsquedas rápidas de nuestros datos almacenados en una base de datos o el monitoreo de los logs generados.
RabbitMQ: es un software conocido como message broker. Debido a que Mytardis es una aplicación web que utiliza el protocolo HTTP que se basa en una solicitud y una respuesta, RabbitMQ nos ayudará como intermedio a que, en caso de que haya un error, en el envío de la respuesta que el servicio pueda reanudarse, haciendo a la aplicación más tolerable a errores, ver imagen 3.

Imagen 3: Funcionamiento general de un massage broker

Ngix: Es un servidor web que le brindará a la aplicación Mytardis rapidez, un mejor rendimiento y que permite la transmisión de componentes web modernos además, de ser escalable, manejar el tráfico entrante para poder distribuirlo y manejar de manera eficiente las tareas que ralenticen el servidor.
PostgreSQL: Es un sistema de base de datos de tipo SQL en donde se almacenarán todos los datos de aplicación como son los bancos de datos de Mytardis (experimentos, dataset, los archivos de datos, usuarios y licencias).

3. Instalación de MyTardis con sus componentes y una BD en PostgreSQL

A pesar de que en el sitio oficial de Mytardis existe la documentación para realizar una “instalación rápida” la instalación no se hace al 100% ya que las dependencias como el servidor nginx, la cola de mensajes de RabbitMQ y Elasticsearch no se encuentran habilitadas además de que la base de datos que construida en la guía corresponde a una BD en SQL lite 3.

Para poder realizar una correcta instalación es necesario realizar los siguientes pasos:

1. Abrir una terminal y posicionarnos en la carpeta en la cual se requiera instalar la aplicación de Mytardis. Posteriormente ejecutar el siguiente *comando* ***git clone -b master***

https://github.com/mytardis/mytardis.git (Para la rama de develop únicamente hay que cambiar la palabra **master**). Es altamente recomendable que no se ejecute el comando anterior como super usuario ya que en los próximos pasos nos generará un problema al querer realizar las pruebas con ayuda de npm.

2. Movernos a la carpeta /mytardis/ y ejecutar el comando: **sudo bash install-ubuntu-py3-requirements.sh,** que instalará todos los requerimientos que la computadora necesita previo a la ejecución de la aplicación
3. Realizar la instalación de todas las dependencias como pip, postgresql, nginx, y rabbitmq ejecutando el siguiente comando: **sudo apt install python3-pip python3-dev libpq-dev postgresql postgresql-contrib nginx curl rabbitmq-server -y,** la bandera -y es para aceptar todo aquello que necesite autorización de instalación
4. Configurar la base de datos en PostgreSQL siguiendo los comandos del cuadro o en la imagen 1.

```
sudo -u postgres psql
CREATE DATABASE tardis_db;

CREATE USER tardis WITH PASSWORD 'yourpassword';

ALTER ROLE tardis SET client_encoding TO 'utf8';
ALTER ROLE tardis SET default_transaction_isolation TO 'read
committed';
ALTER ROLE tardis SET timezone TO 'UTC';

GRANT ALL PRIVILEGES ON DATABASE tardis_db TO tardis;
ALTER USER tardis CREATEDB;
\q
```

Configurar PostgreSQL

```
## login to postgres with the postgres user
sudo -u postgres psql
```

```
## enter the following psql commands to create the db
CREATE DATABASE tardis_db;

## add a password
CREATE USER tardis WITH PASSWORD 'yourpassword';

## set tardis time zone settings
ALTER ROLE tardis SET client_encoding TO 'utf8';
ALTER ROLE tardis SET default_transaction_isolation TO 'read committed';
ALTER ROLE tardis SET timezone TO 'UTC';

## give privileges to tardis to work with the db
GRANT ALL PRIVILEGES ON DATABASE tardis_db TO tardis;
ALTER USER tardis CREATEDB;

## quit out of psql
\q
```

Imagen 1: Comandos para ejecutar para configurar la base de datos en PostgreSQL

5. Creación de un entorno virtual con virtualenvwrapper. Para esto ejecutamos los comandos del siguiente recuadro o de la imagen 2.

```
sudo -H pip3 install --upgrade pip
sudo -H pip3 install virtualenvwrapper
sudo -H pip3 install virtualenv
```

```
## virtualenvwrapper pip3 installation
sudo -H pip3 install --upgrade pip
sudo -H pip3 install virtualenvwrapper
sudo -H pip3 install virtualenv
```

Imagen 2: Comando a ejecutar para la instalación de virtualenvwrapper

Posterior a esto, virtualenvwrapper no tendrá la capacidad de crear entornos virtuales, ya que primero es necesario realizar una modificación de su script, por lo que se deberá de ejecutar el siguiente comando: **sudo vim ~/.bashrc** y añadimos las siguientes líneas:

Después presionamos la tecla **esc** y escribimos **:wq**.

```
export WORKON_HOME=$HOME/.virtualenvs
export PROJECT_HOME=$HOME/Devel
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
```

Con esto decimos que se use virtualenvwrapper y que versión de Python estaremos usando cada que creemos un nuevo entorno.

Así, podemos crear el entorno virtual ejecutando el comando **mkvirtualenv <nombre entorno>**

6. Nos movemos a la carpeta /mytardis/ donde se encuentra todos los archivos de la aplicación y ejecutamos los comandos de la imagen 3, realizando la instalación de las librerías necesarias para que la aplicación Mytardis funcione de manera correcta.

```
pip install -U pip
pip install -U -r requirements.txt
pip install -U -r requirements-postgres.txt
```

```
## move up a directory
cd ..
## install requirements into mytardis virtual environment
pip install -U pip
pip install -U -r requirements.txt
pip install -U -r requirements-postgres.txt
```

Imagen 3: Instalación de los requerimientos necesarios para la aplicación

7. Es necesario construir un archivo de configuración nuevo para que la aplicación, almacene en algún lugar la BD además de especificar la tecnología a utilizar, las credenciales para poder realizar las peticiones y el host donde se encuentra localizada. Además de configuraciones para nginx como servidor proxy.

```
from .default_settings import *
from pathlib import Path
import os

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Add site specific changes here
ALLOWED_HOSTS = ['your_domain.com', 'www.your_domain.com', 'localhost']
#ALLOWED_HOSTS = ['server_ip_address', 'localhost']

# additional applications
INSTALLED_APPS += ('tardis.apps.mydata',)

# Turn on django debug mode.
DEBUG = True

# Swapped to postgressql
# The database needs to be named something other than "tardis" to avoid
# a conflict with a directory of the same name.
DATABASES['default']['ENGINE'] = 'django.db.backends.postgresql_psycopg2'
DATABASES['default']['NAME'] = 'tardis_db'
DATABASES['default']['USER'] = 'tardis'
DATABASES['default']['PASSWORD'] = 'yourpassword'
DATABASES['default']['HOST'] = 'localhost'

# It is recommended to pass protocol information to Gunicorn. Many web frameworks use this
# information to generate URLs. Without this information, the application may mistakenly
# generate ‘http’ URLs in ‘https’ responses, leading to mixed content warnings or broken ap
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')

# Next, move down tot he bottom of the file and add a setting indicating where the static f
# be placed. This is necessary so that Nginx can handle requests of these items. The follow
# Django to place them in a directory called static in the base project directory:
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static/')
```

Para esto es necesario estar dentro de la carpeta /mytardis/ y ejecutar lo siguiente: **sudo vim tardis/settings.py** y escribir las líneas contenidas en la imagen 4.

Imagen 4: Creación del archivo de configuración que nos permitirá utilizar una BD en PostgreSQL

En la parte de allowed host es necesario agregar la cadena ‘127.0.0.1’ a pesar de que es lo mismo a escribir localhost, al correr la aplicación se desplegará un error que nos dirá que la dirección 127.0.0.1 se debe agregar a los allowed host

8. El siguiente paso es la instalación de los módulos npm para realizar las pruebas de la aplicación, ejecutamos los siguientes comandos: **npm install --production**, **npm install && npm test**, **npm run-script build**.

9. Es necesario la construcción de una llave secreta por motivos de seguridad de la aplicación, por lo que es necesario ejecutar el siguiente comando:

```
sudo python -c "import os; from random import choice; key_line =
'%sSECRET_KEY=\"%s\"  # generated from build.sh\n' % ('from
.default_                                                    else '',
''.join([ch                                                  =+)') for i
in range                                                     ose()"
```

```
python manage.py migrate
python manage.py createcachetable default_cache
python manage.py createcachetable celery_lock_cache
```

10. Una vez creada la llave, se procede a realizar la creación de las tablas para la BD ejecutando los comandos de la imagen 5.

```
python manage.py migrate
python manage.py createcachetable default_cache
python manage.py createcachetable celery_lock_cache
```

Imagen 5: Comandos para la creación de las tablas para la BD

11. Creamos un super usuario para la aplicación ejecutando el siguiente comando: **python manage.py createsuperuser** e ingresando los datos que se nos pida.
12. Corremos el servidor con el siguiente comando**: python manage.py runserver** y se desplegará una vista como la de la imagen 6. Además de poder entrar a la plataforma usando el super usuario que creamos, ver imagen 7.

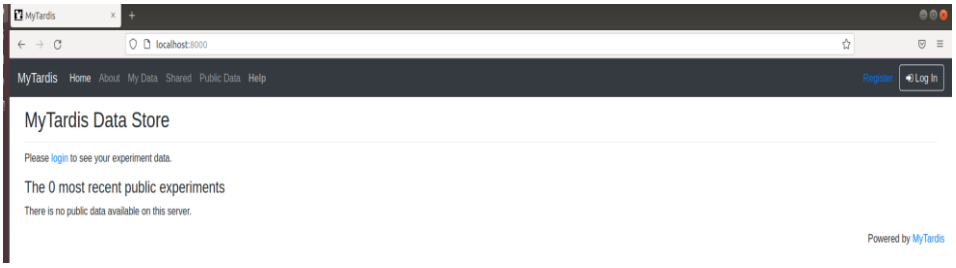


Imagen 6: Vista principal de Mytardis sin usuario loggeado

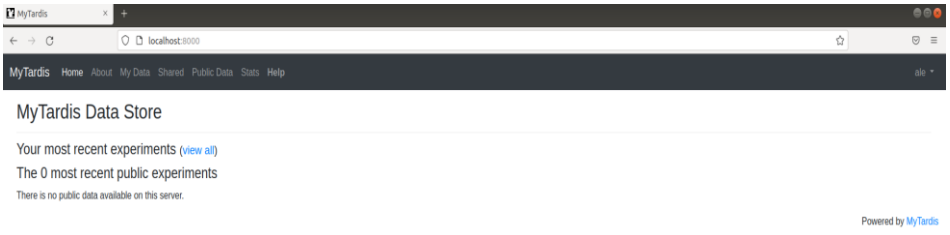


Imagen 7: Vista principal después de entrar con la cuenta de super usuario anteriormente creada.

13. Por último, hace falta realizar la configuración de elasticsearch. Primero necesitaremos la instalación de este, en la imagen 8 se muestran los comandos para su instalación.

curl -fsSL https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add –

echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee -a /etc/apt/sources.list.d/elastic-7.x.list

sudo apt update

sudo apt install elasticsearch

Install Elasticsearch

```
# Installing Elasticsearch
curl -fsSL https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -

# Add Elastic source list to sources.list.d
echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee -a /etc

sudo apt update

sudo apt install elasticsearch
```

Imagen 8: Instrucciones a ejecutar en la terminal para la instalación de eleasticsearch

Entramos al YAML para sobre escribir su configuración. En la imagen 9 se muestra exactamente qué parte del YAML ejecutando el comando **sudo vim /etc/elasticsearch/elasticsearch.yml** se debe de sobre escribir.

```
. . .
# ---------------------------------- Network -----------------------------------
#
# Set the bind address to a specific IP (IPv4 or IPv6):
#
network.host: localhost
. . .
```

Imagen 9: Modificación del YML de elasticsearch.

Posterior a esto es necesario recargar elasticsearch tras modificar su configuración mediante los comandos **sudo systemctl daemon-reload** y
sudo systemctl restart elasticsearch y permitimos que inicie al momento en el que haya un inicio de un servidor con el siguiente comando **sudo systemctl enable elasticsearch**.
Posteriormente creamos un firewall para elasticsearch con la siguiente instrucción: **sudo ufw allow from 198.51.100.0 to any port 9200**
Realizamos una prueba introduciendo lo siguiente: **curl -X GET 'http://localhost:9200'.** La salida será algo como lo mostrado en la imagen 10.

```
ale@ale-MT:~$ curl -X GET 'http://localhost:9200'
{
  "name" : "ale-MT",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "KVHbVijqTY6Z7rZM00_bug",
  "version" : {
    "number" : "7.17.3",
    "build_flavor" : "default",
    "build_type" : "deb",
    "build_hash" : "5ad023604c8d7416c9eb6c0eadb62b14e766caff",
    "build_date" : "2022-04-19T08:11:19.070913226Z",
    "build_snapshot" : false,
    "lucene_version" : "8.11.1",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Imagen 10: Prueba de funcionamiento de elasticsearch

Una vez configurado hace falta integrarlo a Mytardis por lo que hay que realizar un cambio en el archivo de configuración de la aplicación (tardis/settings.py). para ello es necesario localizarse dentro de la capeta de archivos de la aplicación y ejecutar el comando **sudo vim tardis/settings.py** e ingresar la configuración mostrada en la imagen 11.

```
INSTALLED_APPS += ('django_elasticsearch_dsl','tardis.apps.search',)
SINGLE_SEARCH_ENABLED = True
ELASTICSEARCH_DSL = {
    'default': {
        'hosts': 'http://localhost:9200'
    },
}
ELASTICSEARCH_DSL_INDEX_SETTINGS = {
    'number_of_shards': 1,
    'number_of_replicas': 0
}
```

```
INSTALLED_APPS += ('tardis.apps.mydata','django_elasticsearch_dsl','tardis.apps.search'

...

# Elasticsearch settings; defaults to False
SINGLE_SEARCH_ENABLED = True

# ELASTICSEARCH_DSL default settings - Once Elasticsearch is set up, and
# Single Search is enabled (i.e. the SINGLE_SEARCH_ENABLED option in settings
# is set to True) Elasticsearch DSL will automatically register the addition
# of and changes to models and reflect these in the search index.
ELASTICSEARCH_DSL = {
    'default': {
        'hosts': 'http://localhost:9200'
    },
}
ELASTICSEARCH_DSL_INDEX_SETTINGS = {
    'number_of_shards': 1,
    'number_of_replicas': 0
}
```

Imagen 11: Información a ingresar en el archivo de configuración

Por último activamos el entorno virtual en que estaremos trabajando (ejecutar **workon <nombre venv>** y tenemos que realizar un rebuil de la aplicación con la instrucción **python manage.py search_index –rebuild.**

Instrucciones e imágenes sacadas de: https://blog.spatialmsk.com/mytardis-mydata/

Pruebas de funcionamiento de la aplicación en una VM Ubuntu 18.04

En las imágenes 4 a 6 se podrá observar cómo es que la aplicación estable de Mytardis ya se encuentra Corriendo en la máquina virtual.

```
Quit the server with CONTROL-C.
(mytardis) ale@ale-MT:~/Documents/MTStable/mytardis$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 07, 2022 - 07:47:32
Django version 3.2.5, using settings 'tardis.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
[07/May/2022 07:47:50] "GET / HTTP/1.1" 200 9902
[07/May/2022 07:47:53] "GET /static/bundles/main-a55b9ba408fc16d2a6d7.styles.css HTTP/1.1" 304 0
[07/May/2022 07:47:53] "GET /static/bundles/main-a55b9ba408fc16d2a6d7.js HTTP/1.1" 304 0
[07/May/2022 07:47:53] "GET /static/bundles/lib-a55b9ba408fc16d2a6d7.js HTTP/1.1" 304 0
[07/May/2022 07:47:53] "GET /static/bundles/tardis_portal_index-a55b9ba408fc16d2a6d7.js HTTP/1.1" 304 0
[07/May/2022 07:47:53] "GET /static/bundles/index_page_badges-a55b9ba408fc16d2a6d7.js HTTP/1.1" 304 0
[07/May/2022 07:48:02] "GET / HTTP/1.1" 200 9902
[07/May/2022 07:48:14] "GET /about/ HTTP/1.1" 200 12318
[07/May/2022 07:48:15] "GET /static/bundles/fontawesome-webfont.woff2 HTTP/1.1" 304 0
[07/May/2022 07:48:22] "GET / HTTP/1.1" 200 9902
[07/May/2022 07:48:27] "GET /mydata/ HTTP/1.1" 200 10099
[07/May/2022 07:48:27] "GET /static/bundles/tardis_portal_my_data-a55b9ba408fc16d2a6d7.js HTTP/1.1" 200 2054
[07/May/2022 07:48:34] "GET /logout/ HTTP/1.1" 302 0
[07/May/2022 07:48:35] "GET / HTTP/1.1" 200 7956
```

Imagen 4: Levantamiento del servidor web de Mytardis

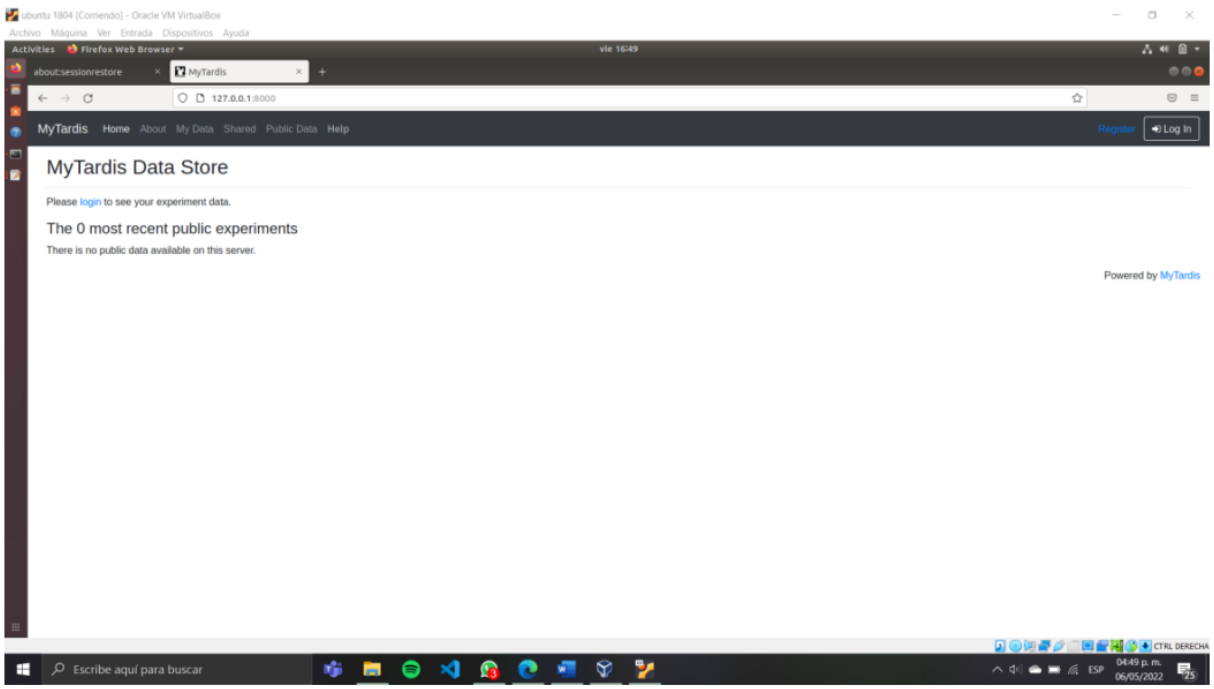


Imagen 5: acceso a la plataforma desde local host

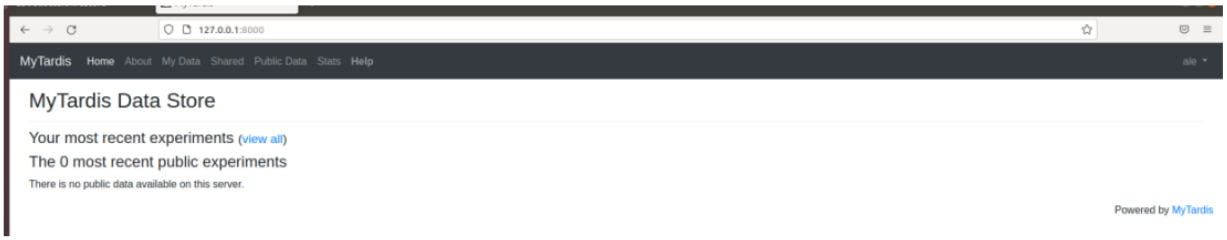


Imagen 6: Log in como super usuario

Se realizó un video demostrativo de la aplicación corriendo en una maquina virtual con el sistema operativo Ubuntu 18.04. Y que puede ser consultado en el siguiente enlace: https://drive.google.com/file/d/1ZNoQY_w8qsyveujFGxxxhVFx4dkMax_u/view?usp=sharing

3.1. Error de instalación al momento de descargar las dependencias necesarias para el software Mytardis (No matching distribution found for pystache=0.5.4)

Al momento de descargar la versión más estable de software Mytardis (la rama master de su repositorio de git) en una máquina virtual Ubuntu en su versión 18.04 LTS que cuenta con Python en su versión 3.6 y seguir las indicaciones mostradas en la imagen 1 procedentes del pdf que nos muestra cómo realizar su implementación de manera local, cuando se realiza la instalación de los módulos necesario para el software y que se encuentran en documento de texto por nombre ***requirements.txt*** mediante el comando ***pip install -U -r requirements.txt*** en consola será mostrado un mensaje de error, como el mostrado en la imagen 2, la cual no dice que no se pudo ejecutar un comando llamado ***use_2to3*** y, como consecuencia, no se ha podido encontrar una versión de pystache que satisfaga lo especificado en el archivo de ***requirements.txt***

Now upgrade pip and setup tools:

```
pip install -U pip setuptools
```

Check pip version:

```
$ pip -V
pip 21.0.1 from /home/ubuntu/mytardis/mytardis/lib/python3.6/site-packages/pip␣
↪(python 3.6)
```

MyTardis dependencies are then installed with pip:

11.2. Configuration and Administration **43**

MyTardis Documentation, Release 4.6.0

```
pip install -U -r requirements.txt
```

Imagen 1: Comandos a ejecutar para poder implementar el software Mytardis

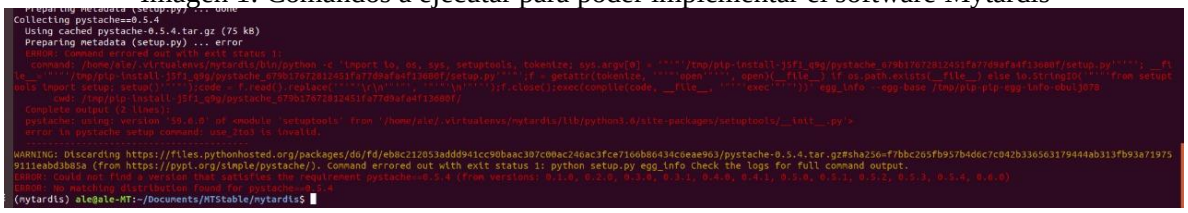


Imagen 2: Mensaje de error desplegado en consola.

Al realizar una investigación en internet se llegó a una solución y es que este error es ocasionado por la versión que el comando ***pip install -U pip setuptools*** ya que este comando instala tanto la versión más actual de pip, que sea compatible con Python 3.6, es decir la versión **21.3.1**, así como la del módulo de setuptools . Siendo la este último la clave para solucionar este problema ya que a partir de su versión 58.0.0 de setuptools el comando **use_2to3** fue removido (ver imagen 3) por lo que será necesario instalar y por tanto ejecutar una versión de setuptools menor a la versión 58. Para esto se ejecuta el comando ***pip install setuptools==57.5.0*** (versión anterior más actual).

Una vez que se realizó la instalación ejecutamos el comando ***pip list*** o ***pip freeze*** y buscamos que se haya instalado la versión 57.5.0 de setuptools (ver imagen 4)
Une vez realizado esto, podremos ejecutar de nuevo el comando ***pip install -U -r requirements.txt*** y ya no habrá mensaje de error (ver imagen 5)

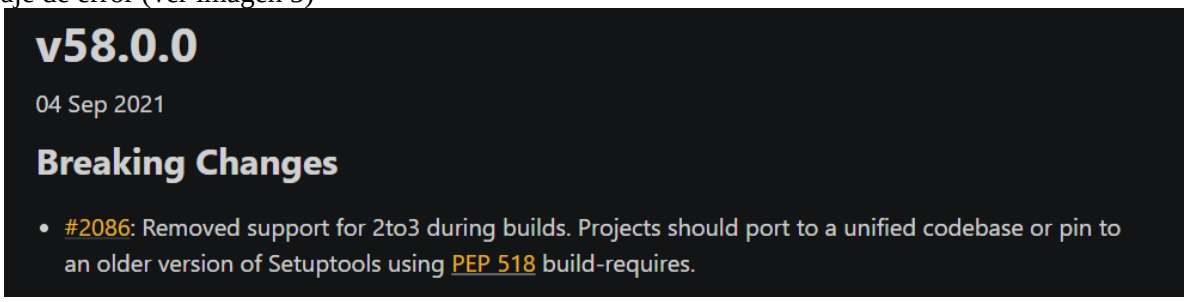


Imagen 3: A partir de la v58.0.0 se removió el soporte a 2to3. Extraído de https://bit.ly/3xElKJm

Imagen 4: Comprobación de una correcta instalación de la versión 57.5.0 de setuptools

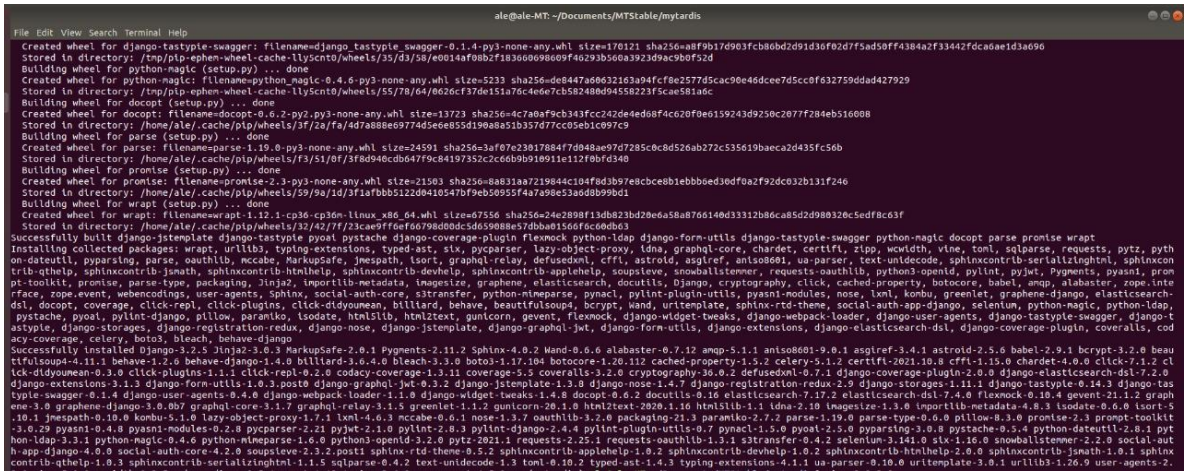


Imagen 5: Correcta instalación de los módulos requeridos para la ejecución del software Mytardis.

4. Posible software para la implementación de la aplicación de MyTardis

4.1. Servicio de Amazon EC2

El servicio de EC2 nos ofrece la creación de instancias computacionales (máquinas virtuales) dependiendo de la carga de trabajo que se le vaya a dar. Existen instancias de uso general, informática optimizada, memoria optimizada, almacenamiento optimizado y computación acelerada que proporcionan el equilibrio óptimo de informática, memoria, almacenamiento y redes para sus cargas de trabajo. de uso general, informática optimizada, memoria optimizada, almacenamiento optimizado y computación acelerada que proporcionan el equilibrio óptimo de informática, memoria, almacenamiento y redes para sus cargas de trabajo.

Las imágenes de máquina de Amazon (AMI) que se pueden precargar a una instancia de EC2 son los siguientes:
Microsoft Windows server
Amazon Linux 2
Ubuntu
RHEL

CentOS
SUSE
Debian
Para tener el acceso hacia la instancia se tiene que configurar una IP a la instancia y se podrá acceder mediante SSH.
Este servicio puede ser vinculado con otros servicios para realizar el monitoreo del uso de CPU, memoria y tráfico como Amazon CloudWatch y asistentes como Compute Optimizer que recomienden ciertas los recursos más adecuados para las cargas de trabajo para reducir costos y aumentar el rendimiento.
En una cuenta gratuita de AWS este servicio viene incluido con una prueba de 1 año disponiendo de 750 horas por mes. Permitiendo la creación de instancias *micro* t2 o t3, esta última solo está disponible para regiones en donde la instancia t2 no está disponible.

Nombre de la instancia	CPU virtual	Memoria (GiB)	Almacenamiento	Rendimiento de red
T2.micro	1	1	EBS	Bajo a moderado*
T3.micro	2	1	EBS	Hasta 5Gbps

Tabla 1: Instancias de EC2 disponibles con una cuenta gratuita.

*: Amazon no da detalles acerca de la velocidad de estos puestos sin embargo se puede intuir que la velocidad podría ir de los 100 Mb al 1Gb de velocidad.

Como se puede ver en la tabla 1 este servicio viene acompañado de otro llamado EBS (Elastic Block Store) por el cual podremos presentar almacenamiento vía SAN a nuestras instancias de EC2 o correr bases de datos. En una cuenta gratuita EBS tiene una prueba de 30GB de almacenamiento.
Existen varios paquetes para contratar EC2 una vez acabada la prueba. El más recomendado por Amazon es el paquete ***bajo demanda*** ya que se recomienda por estas razones:
Ideal para usuarios que prefieren aprovechar el bajo costo y la flexibilidad de Amazon EC2 sin necesidad de realizar pagos por adelantado o asumir un compromiso a largo plazo
Aplicaciones con cargas de trabajo a corto plazo, con picos de demanda o imprevisibles que no admitan interrupciones
Aplicaciones que se estén desarrollando o probando en Amazon EC2 por primera vez
En cuanto al precio las instancias para **uso general** con 2 o 4 CPU 's virtuales van desde los 0.005 dólares hasta los 0.34 dólares. Mientras que las instancias **optimizadas para almacenamiento** su costo va desde los 0.17 dólares hasta 0.5 dólares de 2 o 4 vCPU. Si es que estas residen en el norte de virginia

4.1.1. Requerimientos para el almacenamiento de la aplicación

- Una cuenta de AWS la cual tenga acceso al servicio de EC2
- Creación de una instancia de EC2 con Ubuntu Server 18.04 con acceso hacia internet y que permita la conexión remota vía SSH

Para más información se pueden consultar los siguientes links:

- Características de EC2: https://aws.amazon.com/es/ec2/features/
- Tipos de instancias de EC2: https://aws.amazon.com/es/ec2/instance-types/
- Precio de las instancias bajo demanda: https://aws.amazon.com/es/ec2/instance-types/

4.2. Docker

Docker es una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.

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 le permite entregar código con mayor rapidez, estandarizar las operaciones de las aplicaciones, transferir el código con facilidad y ahorrar dinero al mejorar el uso de recursos. Con Docker, obtiene un solo objeto que se puede ejecutar de manera fiable en cualquier lugar. La sintaxis sencilla y simple de Docker le aporta un control absoluto. La amplia adopción significa que existe un gran ecosistema de herramientas y aplicaciones listas para su uso que puede utilizar con Docker, ver imagen 6.

Imagen 6: Razones para utilizar docker.

Puede utilizar los contenedores de Docker como bloque de construcción principal a la hora de crear aplicaciones y plataformas modernas. Docker facilita la creación y la ejecución de arquitecturas de microservicios distribuidos, la implementación de código con canalizaciones de integración y entrega continúas estandarizadas, la creación de sistemas de procesamiento de datos altamente escalables y la creación de plataformas completamente administradas para sus desarrolladores. La colaboración reciente entre AWS y Docker facilita la implementación de artefactos de Docker Compose en Amazon ECS y AWS Fargate.

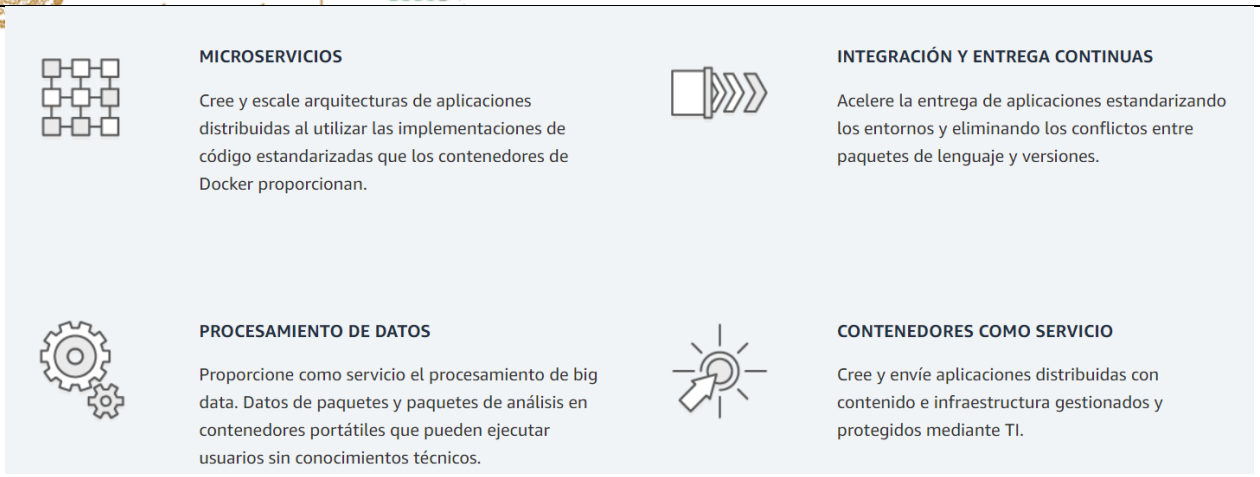


Imagen 7: Cuando usar docker.

Docker, por sí solo, puede gestionar contenedores individuales. Si comienza a utilizar cada vez más contenedores y aplicaciones que se alojan en ellos con cientos de elementos, se dificultará su gestión y organización. En algún momento, deberá cambiar el enfoque y agrupar los contenedores para prestar los servicios, como las redes, la seguridad y la telemetría, entre otros, en todos los contenedores. Allí entra en juego Kubernetes.

Docker no ofrece las mismas funciones tipo UNIX que se obtienen con los contenedores tradicionales de Linux, como la capacidad para usar los procesos como cron o syslog dentro del contenedor, junto con la aplicación. Esta tecnología también implica otras limitaciones, por ejemplo, en cuanto a la eliminación de los procesos derivados de los secundarios después de borrar estos últimos, lo cual se realiza de manera natural en los contenedores tradicionales de Linux. A pesar de que no resulte evidente a simple vista, se pueden reducir estas complicaciones al modificar el archivo de configuración y establecer esas habilidades desde el comienzo.

Además, hay otros subsistemas y dispositivos de Linux que no cuentan con espacios de nombres, como SELinux, los grupos de control y los dispositivos /dev/sd*. Esto implica que, si un atacante obtiene control de estos subsistemas, el host se ve comprometido. El hecho de compartir el kernel del host con los contenedores para mantener la agilidad podría poner en riesgo la seguridad. En cambio, las máquinas virtuales son diferentes, ya que están bien separadas del sistema host.

Para más información consulte los siguientes links:

- ¿Docker es completamente seguro?: https://opensource.com/business/14/7/docker-security-selinux?extIdCarryOver=true&sc_cid=701f2000001OH7EAAW
- ¿Qué es docker'?: https://www.redhat.com/es/topics/containers/what-is-docker , https://aws.amazon.com/es/docker/ , https://www.ibm.com/ar-es/cloud/learn/docker

5. Implantación de MyTardis en Docker con una Base de datos en PostgreSQL

5.1. creación de una red virtual en docker

Docker nos permite implementar pequeñas virtualizaciones de SO operativo para correr aplicaciones y servicios de software. Sin embargo, muchas de las veces una aplicación no solamente son datos en bruto computarizados por algún usuario para realizar algunas operaciones y obtener una salida en algunas ocasiones como las aplicaciones Web, se necesitan guardar información sobre los usuarios, datos cargados al portal, logs para saber que se hace a la aplicación, quien lo hace, cuando lo hace y desde donde lo hace, entre otra información importante que, en algún momento del tiempo, necesitará ser consultada por un usuario o administrados. Por ello, además de una aplicación web implementada en algún lenguaje de programación, adicionalmente es necesario implementar una base de datos que tendrá toda contendrá toda la información sobre los usuarios y administradores de la aplicación y de alguna manera vincular app y BD.
Docker permite de muchas maneras realizar este vínculo entre aplicaciones y servicios de manera local. La principal y la que se utilizará para poder vincular la aplicación de MyTardis con una BD en Postgres es creando “networks”. En este documento se plasma los pasos a seguir para la creación de la network en Docker para que la aplicación y la BD puedan comunicarse.

1. Como prerrequisitos es necesario contar con la descarga del software Docker y la funcionalidad de WSL2 con una distribución Ubuntu 18.04 activado en el equipo de cómputo donde se vaya a desplegar la aplicación. Posteriormente vincular estas 2 aplicaciones. En caso de no contar con alguno de estos, se puede revisar los siguientes links.
   - Para la instalación de WSL2: https://docs.microsoft.com/en-us/windows/wsl/install
   - Para la instalación de Docker: https://docs.docker.com/desktop/install/windows-install/
   - Para vincular las aplicaciones: https://docs.docker.com/desktop/windows/wsl/
2. En la terminal de WSL ejecutamos el siguiente comando.

docker network create mytardisnet

```
alexis@DESKTOP-HA1546F:~$ docker network create mytardisnet
ada908f24fc2ec9d981ab37d96f2db6f466a3696f62aaab8312497aab3554e8a
```

3. Comprobamos que la red se haya creado y que sea de tipo puente o bridge.

docker network ls

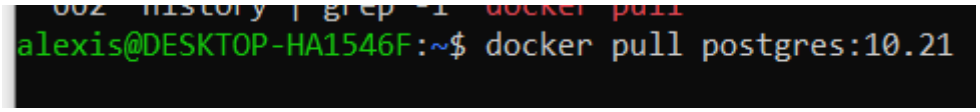
```
alexis@DESKTOP-HA1546F:~$ docker network ls
NETWORK ID     NAME          DRIVER    SCOPE
d07b079f6d16   bridge        bridge    local
0594d965da31   host          host      local
ada908f24fc2   mytardisnet   bridge    local
5026e69bca74   none          null      local
b92a95e84550   tardisnet     bridge    local
```

5.2. Implementación del docker de la Base de Datos en PostgreSQL

Para poder implementar MyTardis utilizando contenedores es ideal separar la aplicación de la BD con el propósito de brindar contenedores ligeros, eficientes, de rápida respuesta.
En este documento se plasman el procedimiento a seguir para poder realizar la implementación del Docker de PostgreSQL.

1. Como prerrequisitos es necesario contar con la descarga del software Docker y la funcionalidad de WSL2 con una distribución Ubuntu 18.04 activado en el equipo de cómputo donde se vaya a desplegar la aplicación. Posteriormente vincular estas 2 aplicaciones. En caso de no contar con alguno de estos, se puede revisar los siguientes links.
    - Para la instalación de WSL2: https://docs.microsoft.com/en-us/windows/wsl/install
    - Para la instalación de Docker: https://docs.docker.com/desktop/install/windows-install/
    - Para vincular las aplicaciones: https://docs.docker.com/desktop/windows/wsl/
2. Una vez instalado Docker y WSL. En la ventana de WSL ejecutamos el siguiente comando:

```
docker pull postgres:10.21
```



   Con este comando indicamos que se quiere realizar la descarga de la imagen para implementar PostgreSQL en su versión 10.21, que es la ideal para MyTardis.

3. Para poner en marcha el Docker de postgres es necesario ejecutar el siguiente mando:

```
docker run --name mytardisdb --network tardisnet -p 5432:5432 -e POSTGRES_USER=tardis -e POSTGRES_PASSWORD=SuperPassw0rd -e POSTGRES_DB=tardis_db -d postgres:10.21
```

   Cada bandera realiza lo siguiente:
   - --name: Le asigna un nombre al Docker. En caso de no especificarla se pondrá un nombre aleatorio generado por Docker. Este parámetro es fundamental ya que ayudará a poder vincularlo con el Docker que contendrá la aplicación
   - --network: Añade el Docker a una red virtual con el propósito de poder brindar comunicación entre dockers. En caso de no contar con una red, se puede consultar en el reporte 1: **Generación de una red en Docker**.
   - -p: se especifica por cual puerto del equipo de cómputo tendrá salida el contenedor y hacía que puerto de la virtualización se conectará.
   - -e: Los Docker tienen la posibilidad de que se les configuren ciertas variables de entorno para personalizar su funcionamiento. En este caso las variables de entorno especificadas fueron:
       - POSTGRES_USER: SE Especifica el usuario que se creará adicional del usuario postgres
       - POSTGRES_PASSWORD: Se asigna una contraseña para el usuario especificado en la variable POSTGRES_USER, si no se especificó un usuario entonces será la contraseña del usuario postgres
       - POSTGRES_DB: crea una base de datos con el nombre especificado
   - -d: Deja en background el proceso de correr el Docker.

4. Por último comprobamos que el servicio de base de datos esté expuesto en nuestra computadora ejecutando el comando en WSL2 y en la interfaz de Docker:

psql -h localhost -p 5432 -U tardis -d tardis_db

```
alexis@DESKTOP-HA1546F:~$ psql -h localhost -p 5432 -U tardis -d tardis_db
Password for user tardis:
psql (10.21 (Ubuntu 10.21-0ubuntu0.18.04.1))
Type "help" for help.

tardis_db=# \list
                                 List of databases
   Name    | Owner  | Encoding |  Collate   |   Ctype    | Access privileges
-----------+--------+----------+------------+------------+-------------------
 postgres  | tardis | UTF8     | en_US.utf8 | en_US.utf8 |
 tardis_db | tardis | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | tardis | UTF8     | en_US.utf8 | en_US.utf8 | =c/tardis        +
           |        |          |            |            | tardis=CTc/tardis
 template1 | tardis | UTF8     | en_US.utf8 | en_US.utf8 | =c/tardis        +
           |        |          |            |            | tardis=CTc/tardis
(4 rows)

tardis_db=#
```

5. Comprobamos que el Docker esté corriendo en la red de Docker que especificamos:

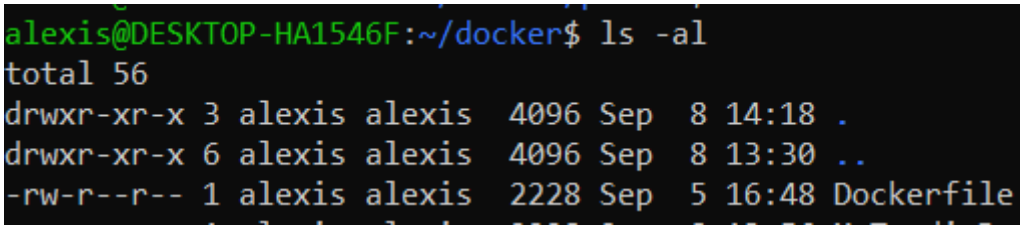
docker network inspect tardisnet


```
alexis@DESKTOP-HA1546F:~$ docker network inspect tardisnet
[
    {
        "Name": "tardisnet",
        "Id": "b92a95e84550892ba202654dd6cebcc6c2fc3c35b43c1b218c469a6722219309",
        "Created": "2022-09-05T16:44:34.4054299Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "777271679b23bc08c09ca56b445b5a1e84b9d0e99c36818b1bb173213a94a863": {
                "Name": "mytardisdb",
                "EndpointID": "b50c4cd2efaec7a67bfd40ac54d828b6c79116c0cc77c9af6fb34e0a5d913977",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            },
            "a1865dc49d8707c3e8c6091a04f8b8f164b8be7a450c229899ec37286454d322": {
                "Name": "mytardisapp",
                "EndpointID": "c4c3ae60d18ed4d76d2ad6b128b5a0f2e3577dcb4575677e921185c3522a55dc",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

5.3. Impelentacion del docker de la aplicación MyTardis

El repositorio de la aplicación My Tardis nos brinda en su rama “master” un archivo de docker para poder implementar esta aplicación (ver siguiente vinculo: https://github.com/mytardis/mytardis/blob/master/Dockerfile) en una pequeña virtualización en el software Docker. Sin embargo, esta implementación es “básica” y utiliza una base de datos ligera e interna de SQL server. Para poder implementar la BD en postgres y se vincule a nuestro contenedor generado anteriormente (ver reporte 2: Pasos para la instalación de PostgreSQL en Docker local) es necesario realizar algunas modificaciones a este archivo. En este documento se plasman todas aquellas correcciones y adiciones al dockerfile.

1. En nuestra terminal de WSL deberemos tener descargado el docker file y nos posicionaremos donde se encuentre almacenado.



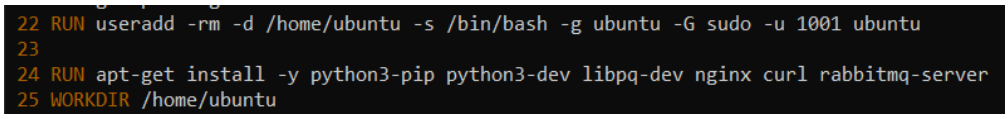
2. A continuación con el comando

vi Dockerfile

Entraremos al editor Vim , después la tecla **esc** y escribimos **:set number**, posteriormente la tecla i podremos empezar a editar nuestro archivo.

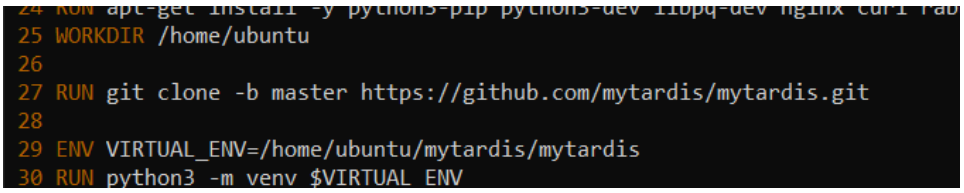
3. Las modificaciones por realizar se listan a continuación:
   - Después de la línea 22 agregar la siguiente línea:

RUN apt-get install -y python3-pip python3-dev libpq-dev nginx curl rabbitmq-server



   - Remplazar la línea 27 por:

RUN git clone -b master https://github.com/mytardis/mytardis.git



- Remplazar la linea 33 por:

```
RUN pip install -U pip setuptools==57.5.0
```

```
31 ENV PATH="$VIRTUAL_ENV/bin:$PATH"
32
33 RUN pip install -U pip setuptools==57.5.0
34
35 WORKDIR /home/ubuntu/mytardis
36
37 RUN pip install -U -r requirements.txt
```

- A partir de la línea línea 37 se tienen que agregar las siguientes líneas

```
RUN pip install -U -r requirements-postgres.txt
RUN echo "from .default_settings import *\n\
from pathlib import Path\n\
import os\n\
BASE_DIR = Path(__file__).resolve().parent.parent\n\
ALLOWED_HOSTS = ['your_domain.com', 'www.your_domain.com', 'localhost', '127.0.0.1']\n\
DEBUG = True\n\
DATABASES['default']['ENGINE'] = 'django.db.backends.postgresql_psycopg2'\n\
DATABASES['default']['NAME'] = 'tardis_db'\n\
DATABASES['default']['USER'] = 'tardis'\n\
DATABASES['default']['PASSWORD'] = 'SuperPassw0rd'\n\
DATABASES['default']['HOST'] = 'mytardisdb'\n\
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')\n\
STATIC_URL = '/static/'\n\
STATIC_ROOT = os.path.join(BASE_DIR, 'static/')\n" >> tardis/settings.py
RUN npm install --production
RUN npm run-script build
RUN python -c "import os; from random import choice; key_line = '%sSECRET_KEY=\"%s\"' %
('from .defa      ult_settings import * \n\n'  if not os.path.isfile('tardis/settings.py') else '',
''.join([choice('a     bcdefghijklmnopqrstuvwxyz0123456789@#%^&*(-_=+)') for i in range(50)]));
f=open('tardis/settings.py'    , 'a+'); f.write(key_line); f.close()"
EXPOSE 8000
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

```
35 WORKDIR /home/ubuntu/mytardis
36
37 RUN pip install -U -r requirements.txt
38 RUN pip install -U -r requirements-postgres.txt
39
40
41 RUN echo "from .default_settings import *\n\
42 from pathlib import Path\n\
43 import os\n\
44 BASE_DIR = Path(__file__).resolve().parent.parent\n\
45 ALLOWED_HOSTS = ['your_domain.com', 'www.your_domain.com', 'localhost', '127.0.0.1']\n\
46 DEBUG = True\n\
47 DATABASES['default']['ENGINE'] = 'django.db.backends.postgresql_psycopg2'\n\
48 DATABASES['default']['NAME'] = 'tardis_db'\n\
49 DATABASES['default']['USER'] = 'tardis'\n\
50 DATABASES['default']['PASSWORD'] = 'SuperPassw0rd'\n\
51 DATABASES['default']['HOST'] = 'mytardisdb'\n\
52 SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')\n\
53 STATIC_URL = '/static/'\n\
54 STATIC_ROOT = os.path.join(BASE_DIR, 'static/')\n" >> tardis/settings.py
55
56 RUN npm install --production
57 RUN npm run-script build
58
59
60 RUN python -c "import os; from random import choice; key_line = '%sSECRET_KEY=\"%s\"' % ('from .de
   ult_settings import * \n\n'  if not os.path.isfile('tardis/settings.py') else '', ''.join([choice(
   bcdefghijklmnopqrstuvwxyz0123456789@#%^&*(-_=+)') for i in range(50)])); f=open('tardis/settings.p
   , 'a+'); f.write(key_line); f.close()"
61
62 EXPOSE 8000
63
64 CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

4. Salimos del editor presionando la tecla **esc** y escribimos **:wq**.
5. Ejecutamos el siguiente comando para construir la imagen de la aplicación

sudo docker build -t mytardis ~/docker

Nota: el nombre de la imagen se puede personalizar al igual que la dirección donde se encuentre el archivo Dockerfile.

```
alexis@DESKTOP-HA1546F:~/docker$ sudo docker build -t mytardis ~/docker
[sudo] password for alexis:
[+] Building 4.2s (2/3)
 => [internal] load build definition from Dockerfile                                    0.0s
 => => transferring dockerfile: 2.27kB                                                  0.0s
 => [internal] load .dockerignore                                                       0.0s
 => => transferring context: 2B                                                         0.0s
 => [internal] load metadata for docker.io/library/ubuntu:18.04                         4.1s
```

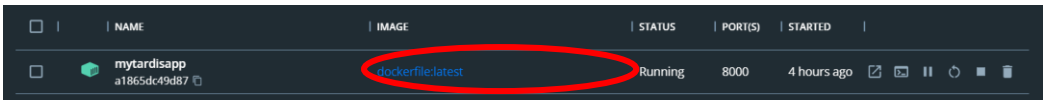
6. Una vez terminada la construcción de la imagen, se debe de realizar la implementación del docker ejecutando el siguiente comando:

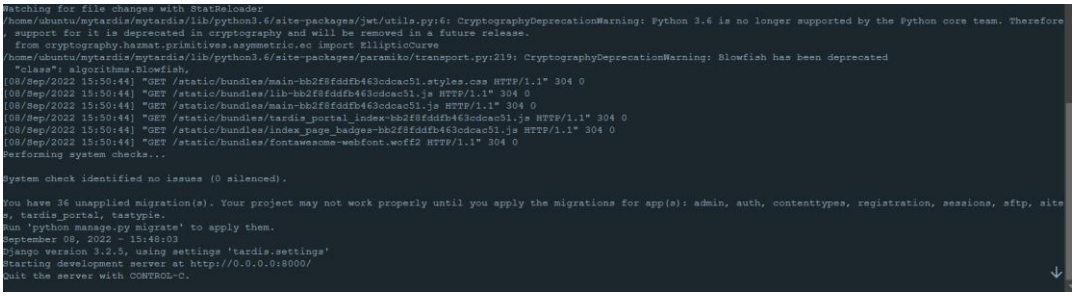
docker run --name mytardisapp --network tardisnet -p 8000:8000 -d mytardis

Las banderas del comando realizan lo siguiente:

- --name: Le asigna un nombre al Docker. En caso de no especificarla se pondrá un nombre aleatorio generado por Docker. Este parámetro es fundamental ya que ayudará a poder vincularlo con el Docker que contendrá la aplicación
- --network: Añade el Docker a una red virtual con el propósito de poder brindar comunicación entre dockers. En caso de no contar con una red, se puede consultar en el reporte 1: **Generación de una red en Docker**.
- -p: se especifica por cual puerto del equipo de cómputo tendrá salida el contenedor y hacía que puerto de la virtualización se conectará.

7. Comprobar en la interfaz de Docker que la aplicación no presente algún error



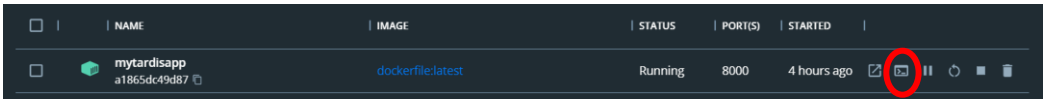


8. Comprobamos que la aplicación y la BD estén en la misma red de docker con el siguiente comando:

docker network inspect tardisnet

9. Abrir la terminal de la aplicación y ejecutar los siguientes comandos:

python manage.py migrate
python manage.py createcachetable default_cache
python manage.py createcachetable celery_lock_cache



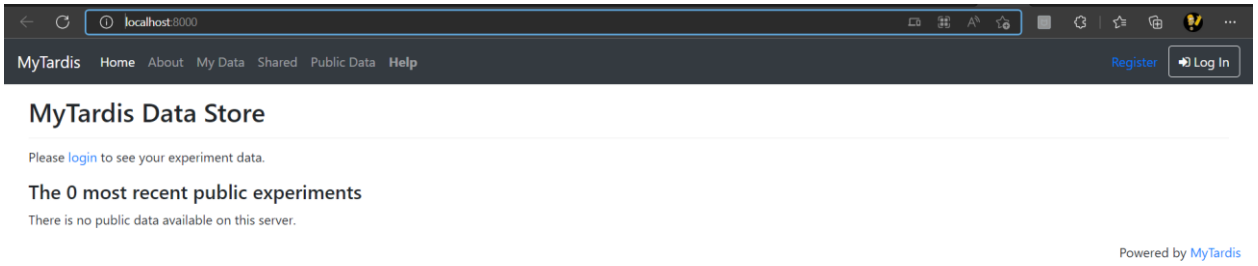
```
# python manage.py migrate
/home/ubuntu/mytardis/mytardis/lib/python3.6/site-packages/jwt/utils.py:6:
yptography and will be removed in a future release.
  from cryptography.hazmat.primitives.asymmetric.ec import EllipticCurve
/home/ubuntu/mytardis/mytardis/lib/python3.6/site-packages/paramiko/transpo
  "class": algorithms.Blowfish,
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, registration, sessions,
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying registration.0001_initial... OK
  Applying registration.0002_registrationprofile_activated... OK
  Applying registration.0003_migrate_activatedstatus... OK
  Applying registration.0004_supervisedregistrationprofile... OK
  Applying registration.0005_activation_key_sha256... OK
  Applying sessions.0001_initial... OK
  Applying tardis_portal.0001_squashed_0011_auto_20160505_1643...
 OK
  Applying sftp.0001_initial... OK
  Applying sites.0001_initial... OK
  Applying sites.0002_alter_domain_unique... OK
  Applying tardis_portal.0012_userauthentication_approved... OK
  Applying tardis_portal.0013_auto_20181002_1136... OK
  Applying tardis_portal.0014_auto_20181002_1154... OK
  Applying tardis_portal.0015_dataset_created_time... OK
  Applying tardis_portal.0016_add_timestamps... OK
  Applying tardis_portal.0017_add_cc_licenses... OK
  Applying tardis_portal.0018_make_default_storage_box_status_online... OK
  Applying tastypie.0001_initial... OK
  Applying tastypie.0002_api_access_url_length... OK
# python manage.py createcachetable default_cache
# python manage.py createcachetable celery_lock_cache
```

10. Creamos el super usuario con el siguiente comando

python manage.py createsuperuser

```
# python manage.py createsuperuser
/home/ubuntu/mytardis/mytardis/lib/python3.6/sit
yptography and will be removed in a future relea
  from cryptography.hazmat.primitives.asymmetric
/home/ubuntu/mytardis/mytardis/lib/python3.6/sit
  "class": algorithms.Blowfish,
Username (leave blank to use 'root'): superuser
Email address: alexisvisa17@gmail.com
Password:
Password (again):
Superuser created successfully.
```

11. Abrir un navegador e ingresar a la dirección **localhost:8000**



6. Anexo

6.1. Estructura del dockerfile para la implementación de MyTardis con sus complementos y una BD en PostgreSQL

A pesar de que el repositorio principal de MyTardis ofrece un Dockerfile para poder implementar la aplicación en un contenedor de Docker esta es muy básica, la base de datos es en una versión ligera de SQL server y los componentes como RabbitMQ, Nginx, Elasticsearch, etc. no son instalados. Además de que algunas configuraciones son añadidas para que la aplicación pueda trabajar con el contenedor de la BD en PostgreSQL.

En este documento se plasma como es que este archivo debe de ser modificado y estructurado.

```
FROM ubuntu:18.04

ENV APT_KEY_DONT_WARN_ON_DANGEROUS_USAGE DontWarn
ENV DEBIAN_FRONTEND noninteractive
ENV PYTHONUNBUFFERED 1

ENV LANG C.UTF-8

RUN apt-get update -y

RUN apt-get install -y \
    sudo git libldap2-dev libmagickwand-dev libsasl2-dev \
    libssl-dev libxml2-dev libxslt1-dev libmagic-dev curl gnupg \
    python3-dev python3-pip python3-venv zlib1g-dev libfreetype6-dev libjpeg-dev

RUN curl -sL https://deb.nodesource.com/setup_12.x | bash -
RUN apt-get install -y nodejs

RUN pip3 install virtualenvwrapper

RUN groupadd -g 1001 ubuntu
RUN useradd -rm -d /home/ubuntu -s /bin/bash -g ubuntu -G sudo -u 1001 ubuntu
RUN apt-get install -y python3-pip python3-dev libpq-dev nginx curl rabbitmq-server
WORKDIR /home/ubuntu

RUN git clone -b master https://github.com/mytardis/mytardis.git
ENV VIRTUAL_ENV=/home/ubuntu/mytardis/Mytardis
```

```
RUN python3 -m venv $VIRTUAL_ENV
ENV PATH="$VIRTUAL_ENV/bin:$PATH"

RUN pip install -U pip setuptools==57.5.0

WORKDIR /home/ubuntu/mytardis

RUN pip install -U -r requirements.txt
RUN pip install -U -r requirements-postgres.txt


RUN echo "from .default_settings import *\n\
    from pathlib import Path\n\
    import os\n\
    BASE_DIR = Path(__file__).resolve().parent.parent\n\
    ALLOWED_HOSTS = ['your_domain.com', 'www.your_domain.com', 'localhost',
'127.0.0.1']\n\
    DEBUG = True\n\
    DATABASES['default']['ENGINE'] = 'django.db.backends.postgresql_psycopg2'\n\
    DATABASES['default']['NAME'] = 'tardis_db'\n\
    DATABASES['default']['USER'] = 'tardis'\n\
    DATABASES['default']['PASSWORD'] = 'SuperPassw0rd'\n\
    DATABASES['default']['HOST'] = 'mytardisdb'\n\
    SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')\n\
    STATIC_URL = '/static/'\n\
    STATIC_ROOT = os.path.join(BASE_DIR, 'static/')\n" >> tardis/settings.py

RUN npm install --production
RUN npm run-script build


RUN python -c "import os; from random import choice; key_line = '%sSECRET_KEY=\"%s\"' %
('from .default_settings import * \n\n'  if not os.path.isfile('tardis/settings.py') else '',
''.join([choice('abcdefghijklmnopqrstuvwxyz0123456789@#%^&*(-_=+)') for i in range(50)]));
f=open('tardis/settings.py', 'a+'); f.write(key_line); f.close()"

RUN curl -fsSL https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add –
RUN echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee -a
/etc/apt/sources.list.d/elastic-7.x.list
RUN sudo apt intall elasticsearch
RUN cat /etc/elasticsearch/elasticsearch.yml | grep -ni "network.host"; sleep 100

EXPOSE 8000

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

```
FROM ubuntu:18.04

ENV APT_KEY_DONT_WARN_ON_DANGEROUS_USAGE DontWarn
ENV DEBIAN_FRONTEND noninteractive
ENV PYTHONUNBUFFERED 1

ENV LANG C.UTF-8

RUN apt-get update -y

RUN apt-get install -y \
    sudo git libldap2-dev libmagickwand-dev libsasl2-dev \
    libssl-dev libxml2-dev libxslt1-dev libmagic-dev curl gnupg \
    python3-dev python3-pip python3-venv zlib1g-dev libfreetype6-dev libjpeg-dev

RUN curl -sL https://deb.nodesource.com/setup_12.x | bash -
RUN apt-get install -y nodejs

RUN pip3 install virtualenvwrapper

RUN groupadd -g 1001 ubuntu
RUN useradd -rm -d /home/ubuntu -s /bin/bash -g ubuntu -G sudo -u 1001 ubuntu
RUN apt-get install -y python3-pip python3-dev libpq-dev nginx curl rabbitmq-server
WORKDIR /home/ubuntu

RUN git clone -b master https://github.com/mytardis/mytardis.git

ENV VIRTUAL_ENV=/home/ubuntu/mytardis/mytardis
RUN python3 -m venv $VIRTUAL_ENV
ENV PATH="$VIRTUAL_ENV/bin:$PATH"

RUN pip install -U pip setuptools==57.5.0

WORKDIR /home/ubuntu/mytardis

RUN pip install -U -r requirements.txt
RUN pip install -U -r requirements-postgres.txt


RUN echo "from .default_settings import *\n\
    from pathlib import Path\n\
    import os\n\
    BASE_DIR = Path(__file__).resolve().parent.parent\n\
    ALLOWED_HOSTS = ['your_domain.com', 'www.your_domain.com', 'localhost', '127.0.0.1']\n
    DEBUG = True\n\
    DATABASES['default']['ENGINE'] = 'django.db.backends.postgresql_psycopg2'\n\
    DATABASES['default']['NAME'] = 'tardis_db'\n\
    DATABASES['default']['USER'] = 'tardis'\n\
    DATABASES['default']['PASSWORD'] = 'SuperPassw0rd'\n\
    DATABASES['default']['HOST'] = 'mytardisdb'\n\
    SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')\n\
    STATIC_URL = '/static/'\n\
    STATIC_ROOT = os.path.join(BASE_DIR, 'static/')\n" >> tardis/settings.py

RUN npm install --production
RUN npm run-script build


RUN python -c "import os; from random import choice; key_line =
'%sSECRET_KEY=\"%s\"' % ('from .default_settings import * \n\n'
if not os.path.isfile('tardis/settings.py') else '',
''.join([choice('abcdefghijklmnopqrstuvwxyz0123456789@#%^&*(-_=+)')
for i in range(50)])); f=open('tardis/settings.py', 'a+'); f.write(key_line); f.close()"

EXPOSE 8000

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

Avala

Dra. Norma Icoquih Zapata Peñasco Asesora(or) Interno	Elaboró Villegas Sánchez Alexis Estudiante