

DESPLIEGUE DE INFRAESTRUCTURA PARA UN SERVICIO WEB CON FLASK Y MONGODB

Realizado por:

José Calderón Valdivia

Ana Casado Sánchez

Francisco Javier Galán Sales

Paula Gómez Matos

Rocío Goñi Medina

María Lourdes Linares Barrera

Pablo Reina Jiménez

Ana Ruiz López

Sevilla, diciembre de 2022

Índice

1. Resumen	2
2. Introducción	3
3. Estado de la tecnología	4
3.1. Flask	4
3.2. MongoDB	4
3.2.1. Bases de datos relacionales VS Bases de datos no relacionales:	4
3.3. Apache	6
3.4. Nginx	6
3.5. LXC	6
4. Descripción de la Arquitectura	8
5. Características de alta disponibilidad	15
6. Escalabilidad	18
7. Mantenimiento	22
8. Limitaciones, trabajo futuro y conclusiones	30
8.1. Limitaciones	30
8.2. Trabajo futuro	32
8.3. Conclusiones finales	32
9. Referencias	32
ANEXO. Acerca de MongoDB	36

1. Resumen

En este trabajo de desarrollo vamos a explicar el proceso de creación de distintas máquinas virtuales, contenedores y plantillas para poder desplegar y visualizar una página web creada gracias al uso combinado de un servidor Apache y Flask. Para la base de datos usaremos *MongoDB*. Tanto la base de datos como el servidor Apache se encontrarán replicados y serán gestionados mediante balanceadores de carga.

2. Introducción

Como anteriormente se ha mencionado el objetivo principal del trabajo es conseguir una página web sobre *Apache*. En nuestro caso la idea central es gestionar el servicio de un servidor web que cargue noticias. Por ello lo primero sería construirlo y almacenar la información que iba a ser mostrada en la página web en una base de datos, haciendo uso de *mongodb*. Toda estructura será desplegada en Proxmox usando contenedores *LXC* (Linux Containers).

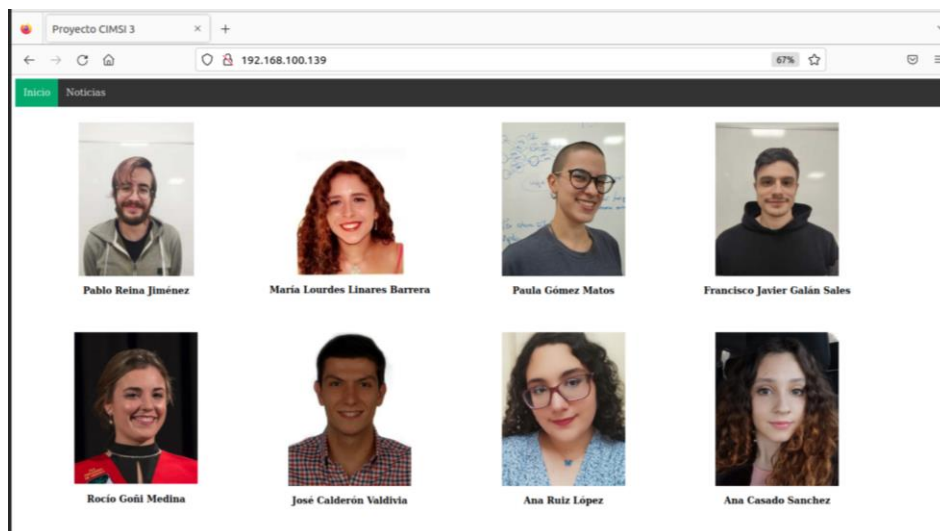


Figura 1. Información de los integrantes del grupo en la página de inicio del servidor web.

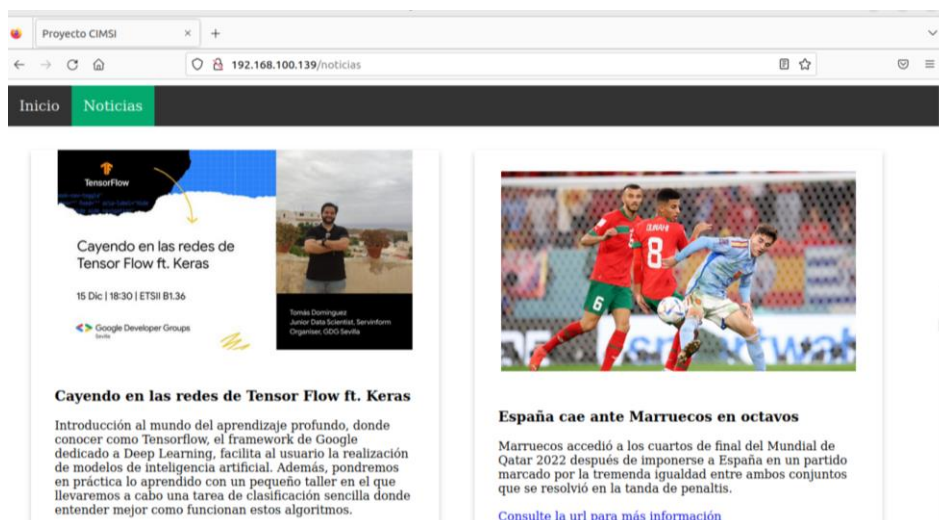


Figura 2. Apartado de noticias del servidor web.

Además, *Proxmox* es accesible desde cualquier parte si se dispusiera de una IP pública y de los determinados puertos pero, por las políticas de la Universidad de Sevilla, esto no ha sido posible de obtener. Al cambiar a la IP pública, podríamos conectarnos a las máquinas virtuales desde cualquier dispositivo, por lo que se tendría el servicio disponible en Internet.

3. Estado de la tecnología

En este apartado enumeramos las tecnologías más destacadas en la actualidad para el despliegue de páginas web y el almacenamiento de datos y sus principales características.

3.1. Flask

Flask es un framework que permite crear aplicaciones de forma sencilla usando Python. En este trabajo se habían barajado diversas opciones entre las que estaban el uso de JavaScript, Node.js y Flask. Finalmente decidimos usar Flask, ya que era el que permitía una conexión más sencilla con mongodb.

3.2. MongoDB

Dado que otro de los grandes objetivos del proyecto era la investigación, hemos decidido empezar a familiarizarnos con las bases de datos no relacionales, ya que a nivel de carrera únicamente hemos utilizado bases de datos relacionales.

En particular, hemos escogido explorar el funcionamiento básico de MongoDB, debido a que es una herramienta muy extendida. A continuación exponemos los conocimientos adquiridos tras nuestra labor de documentación.

3.2.1. Bases de datos relacionales VS Bases de datos no relacionales:

Una vez expuestos nuestros objetivos es el momento de desarrollar los conceptos mencionados. En primer lugar es necesario hacer hincapié en las diferencias existentes entre bases de relacionales y no relacionales.

Conceptualmente las bases de datos relacionales o SQL se definen por ser un tipo de base de datos que almacena y proporciona acceso a puntos de datos relacionados entre sí. Destacan por almacenar los datos en un conjunto de tablas (o relaciones) cuyas columnas representan campos o atributos característicos del conjunto de datos que estamos tratando. Por su parte cada fila de las tablas es un registro que tiene normalmente un valor por cada atributo, así como una ID única. Esto último implica que se evita la duplicación de registros al mismo tiempo que se garantiza la integridad de los datos.

De este modo, las SQL presentan un paradigma que nos confiere sencillez gracias a su lenguaje estructurado, uniformidad de los datos en todas las aplicaciones y copias de la propia base (instancias), al mismo tiempo que nos permiten combinar de forma eficiente las distintas tablas para extraer información referida a su relación.

En la actualidad los tipos de gestores de bases de datos relacionales más usados son: Oracle, MySQL, Microsoft SQL Server, PostgreSQL y DB2.

Empleados							
ID_e	1º Apellido	2º Apellido	Nombre	Nº SS	Calle	CP	Municipio
1							
2							
3							
4							

Figura 3. Ejemplo de base de datos SQL.

Por su parte las bases de datos no relacionales o NoSQL plantean modelos de datos específicos y flexibles, almacenando la información en documentos en lugar de tablas. Este sistema es relativamente nuevo y, aunque su popularidad está creciendo con la migración de muchas empresas a este modelo, aún siguen predominando las SQL. Aquí cabe aclarar que el uso de la terminología SQL/NoSQL hace referencia al lenguaje SQL, que es el utilizado en las consultas para las bases de datos relacionales. Las no relacionales no utilizan dicho lenguaje para realizar sus consultas, sino se emplean lenguajes propios como JSON, CQL o GQL.

```
{
  "responseHeader":{
    "status":0,
    "QTime":0,
    "params":{
      "indent":"on",
      "start":0,
      "q":"Carlos",
      "version":"2.2",
      "rows":10
    }
  },
  "response":[
    {
      "agencia":"00001 - Guatemala",
      "agente":"0005786 - Carlos Martinez",
      "vehiculo":1995,
      "colaborador":"Daniel Pérez",
      "marca":"TOYOTA",
      "descModelo":"TERCEL"
    },
    {
      "agencia":"000485 - TECPAN",
      "agente":"0005455 - Carlos Sagastume",
      "vehiculo":2005,
      "colaborador":"Luis Hernandez",
      "marca":"KIA",
      "descModelo":"SORENTO"
    }
  ]
}
```

Figura 4. Ejemplo de base de datos NoSQL.

Ahora bien, si las SQL protegen nuestros datos y presentan una arquitectura estandarizada y simple, ¿Por qué cambiarla? ¿Qué motiva la aparición de esta nueva alternativa? La respuesta a estas incógnitas se reduce a una palabra: escalabilidad. Cada vez nuestras aplicaciones presentan requisitos más únicos y necesitan mayores volúmenes de datos para operar.

En este contexto las NoSQL nos aportan una gran flexibilidad que nos permite almacenar nuestros datos incluso cuando no tenemos una noción clara sobre estos que pueda llegar a dar forma a una estructura. Además, tienen un alto rendimiento que viene

de la mano de una baja exigencia, haciendo posible su ejecución aún poseyendo recursos mínimos. De hecho, las NoSQL proponen una gran escalabilidad horizontal, es decir, podemos manejar muchos datos sin ralentizar el sistema.

3.3. Apache

Por otro lado, otra de las herramientas que se han usado en este trabajo de despliegue es *Apache*, un servidor HTTP de código abierto. Actualmente es muy usada para el almacenamiento de webs y su posterior visualización a través de diferentes navegadores. Además, su código es mejorado continuamente por la gran comunidad de desarrolladores con la que cuenta, lo cual hace que tenga un alto nivel de seguridad. En este mismo campo nos encontramos también con herramientas como Wordpress, Joomla y webflow que nos facilitan la creación de páginas webs.

3.4. Nginx

Para crear el balanceador de carga hemos utilizado *Nginx*, un software de código abierto. De esta forma, conseguimos distribuir las solicitudes al grupo de servidores (*Apache* en nuestro caso) para aumentar la disponibilidad, confiabilidad y escalabilidad. Existen tres métodos principales para balancear la carga:

- Round-robin: suele ser el que se usa por defecto, distribuye las solicitudes de forma “redonda”, es decir, recorre la lista de nodos de forma secuencial y vuelve al primero de los servidores reales tras llegar al último.
- Selección del menos ocupado/menos conexiones: se usa un algoritmo *slow-start* que evita la sobrecarga de los servidores, y elige el servidor con menos conexiones activas.
- Ip-hashing: se hace uso de una función *hash* que determina qué servidor se debe seleccionar en cada solicitud en función de la IP del cliente. Esto permite también la persistencia de la sesión, ya que asocia esta IP a un servidor particular.

3.5. LXC

Una de las tecnologías de virtualización para sistemas operativos Linux empleadas ha sido Linux Containers (LXC). LXC es una plataforma de contenedores y, al igual que *Apache*, es de código abierto. Una de las ventajas que presenta esta tecnología es que su consumo de recursos es bastante inferior con respecto al de una máquina virtual y, además, tienen una interfaz de usuario estándar, lo cual permite que la administración de varios contenedores de forma simultánea sea relativamente sencilla. Para completar,

esta herramienta está en pleno auge, sobre todo gracias a la plataforma Docker que normalizó el uso de contenedores Linux.

Al ser de código abierto cuenta con una gran comunidad de desarrolladores que proporcionan actualizaciones de la herramienta haciéndola más segura.

4. Descripción de la Arquitectura

La arquitectura elegida para el servicio sería la siguiente:

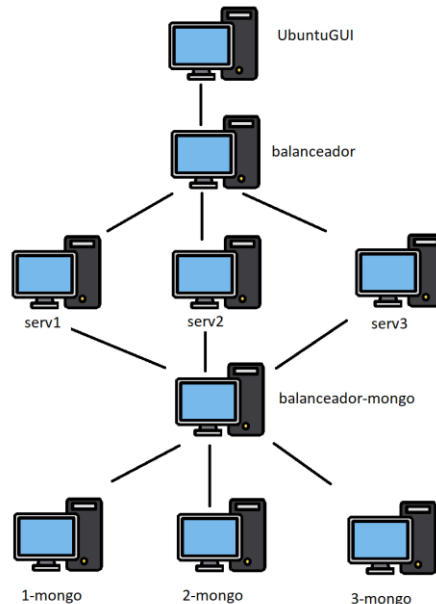


Figura 5. Esquema de la arquitectura.

A groso modo las funciones de cada máquina serían:

- *UbuntuGUI*: el ordenador del usuario que quiere acceder al servicio web
- *balanceador*: un balanceador de carga para el servicio web
- *servX*: contenedores con el servidor web
- *balanceador-mongo*: un balanceador de carga para la base de datos
- *x-mongo*: contenedores con la base de datos

En este trabajo de despliegue, una de las herramientas principales que se han usado ha sido *Terraform* con *Proxmox*. Esta herramienta permite el despliegue de la infraestructura, es decir, la creación de máquinas virtuales y contenedores. Para ello disponemos de un fichero, *main.tf*, donde se indica la versión a utilizar así como la información referente al proveedor de servicio, que en nuestro caso será *Proxmox*. Además de rellenar esos campos, también es necesario indicar datos como la url a la que se manda el servidor *Proxmox* y algunos tokens de identificación. Una vez completada esta información se pasa a declarar las máquinas o contenedores. El contenido de nuestro *main.tf* es el siguiente:

```

1 terraform {
2   required_version = ">= 1.1.0"
3   required_providers {
4     proxmox = {
5       source = "telmate/proxmox"
6       version = ">= 2.9.5"
7     }
8   }
9 }
10
11 provider "proxmox" {
12   pm_tls_insecure = true
13   pm_api_url = 
14   pm_api_token_id=
15   pm_api_token_secret=
16   pm_otp = ""
17   pm_debug=true
18 }

```

Figura 6. Versión y provider.

```

61
62 resource "proxmox_lxc" "mongodb" {
63   target_node = 
64   hostname    = "mongodb"
65   osetemplate = "local:vztmpl/debian-10-turnkey-mongodb_16.1-1_amd64.tar.gz"
66   // para el usuario root
67   password    = 
68   unprivileged = 
69   pool        = "G3"
70   start       = true
71
72   template = true
73
74
75   rootfs {
76     storage = "local-raid1"
77     size    = "8G"
78   }
79
80
81   network {
82     name = 
83     bridge = 
84     ip    = 
85     gw    = 
86   }
87 }

```

Figura 7. Contenedor mongo

```

116 resource "proxmox_lxc" "ubuntu18-1" {
117   target_node = 
118   hostname    = "ubuntu18-1"
119   otemplate   = "local:vztmpl/ubuntu-18.04-standard_18.04.1-1_amd64.tar.gz"
120   // para el usuario root
121   password    = 
122   unprivileged = 
123   pool        = "G3"
124   start       = true
125
126   template    = true
127
128
129   rootfs {
130     storage = "local-raid1"
131     size    = "8G"
132   }
133
134
135   network {
136     name   = 
137     bridge = 
138     ip     = 
139     gw     = 
140   }
141 }
142

```

Figura 8. Contenedor ubuntu con el servidor web

```

144 resource "proxmox_lxc" "balanceador" {
145   target_node = 
146   hostname    = "balanceador"
147   otemplate   = "local:vztmpl/ubuntu-18.04-standard_18.04.1-1_amd64.tar.gz"
148   // para el usuario root
149   password    = 
150   unprivileged = 
151   pool        = "G3"
152   start       = true
153
154   // template = true
155
156
157   rootfs {
158     storage = "local-raid1"
159     size    = "8G"
160   }
161
162
163   network {
164     name   = 
165     bridge = 
166     ip     = 
167     gw     = 
168   }
169 }

```

Figura 9. Balanceador de carga

Uno de los comandos más utilizados, dentro de este trabajo en Terraform, es *terraform apply*, para que se hagan efectivos los cambios. Al ejecutar este comando hace la comprobación correspondiente a *terraform plan* y, además, indica qué es lo que va a

ocurrir con los recursos en la infraestructura, es decir, si se van a añadir, eliminar, modificar, etc.

Gracias a *Terraform* hemos podido crear las máquinas virtuales y contenedores necesarios para realizar este proyecto. Las características de dicha arquitectura las detallaremos posteriormente.

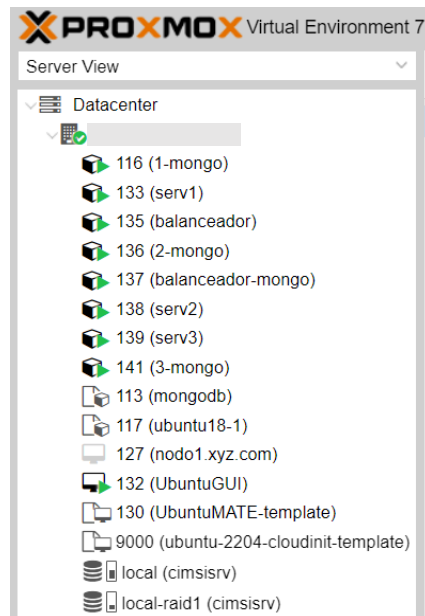


Figura 10. Máquinas virtuales y contenedores en Proxmox.

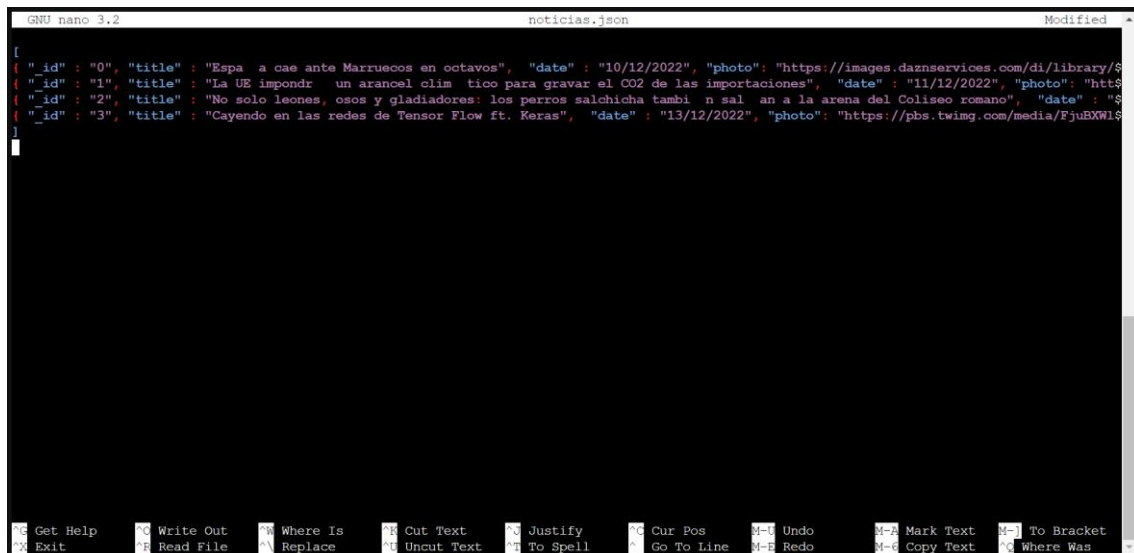
A continuación, vamos a desglosar el comportamiento de cada máquina virtual, comenzando por *UbuntuGUI*. Dicha máquina se compone únicamente de una interfaz gráfica, la cual nos permitirá acceder a la web anteriormente mencionada desde su navegador.

Además, tenemos los distintos “serv” (serv1, serv2 y serv3), donde están los servidores *Apache* con la página web creada. Es importante que use el sistema operativo Ubuntu 18.1, ya que el resto de las versiones causan problemas de compatibilidad con librerías de Python. Éstas realizan una *query* a la base de datos, la cual está albergada en las máquinas “x-mongo”. Mediante el uso de wsgi y un VirtualHost de Apache hemos conseguido redireccionar el tráfico del servidor Flask a través de Apache, es decir, al realizar una petición a Apache, realmente se la estamos realizando a Flask.

Los distintos balanceadores funcionan de tal forma que, al hacer una petición desde UbuntuGUI, éste te devuelve el servidor *Apache* con menor tráfico. De igual forma, el balanceador de *mongoDB* devuelve la réplica de la base de datos albergada en el servidor con menos conexiones o tráfico correspondiente.

Respecto a *mongoDB*, vamos a trabajar con la base de datos database en la que hemos creado una colección llamada noticias.

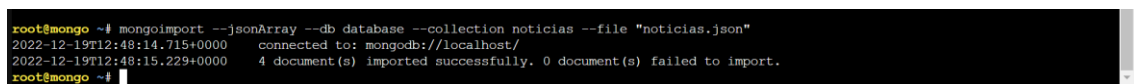
Para la creación de la base de datos partimos de un archivo .json que contendrá todos los objetos que queremos guardar. Para la importación será necesario tener este archivo en nuestro sistema, tal y como se muestra en la siguiente imagen.



```
GNU nano 3.2 noticias.json Modified
[
  { "_id": "0", "title": "Espa a cae ante Marruecos en octavos", "date": "10/12/2022", "photo": "https://images.daznservices.com/di/library/9",
  { "_id": "1", "title": "La UE impondr un arancel clim tico para gravar el CO2 de las importaciones", "date": "11/12/2022", "photo": "https://",
  { "_id": "2", "title": "No solo leones, osos y gladiadores: los perros salchicha tambi n sal an a la arena del Coliseo romano", "date": "12/12/2022", "photo": "https://",
  { "_id": "3", "title": "Cayendo en las redes de Tensor Flow ft. Keras", "date": "13/12/2022", "photo": "https://pbs.twimg.com/media/FjuBXW1S"
]
```

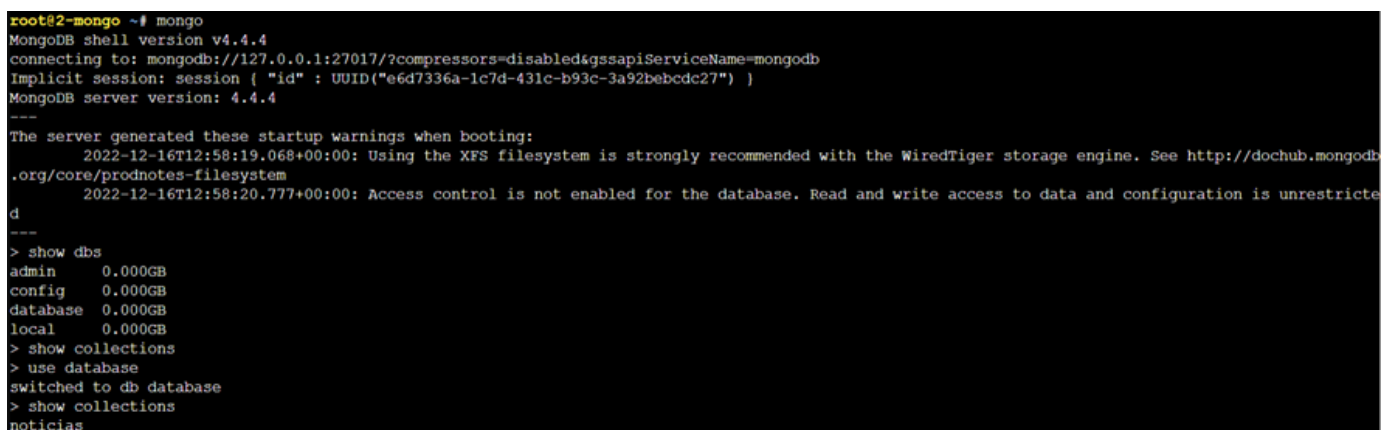
Figura 11. Base de datos Noticias.

Tras esto, incluir estos datos en nuestro sistema con mongoDB se reduce al uso de un comando: `mongoimport --jsonArray --db database --collection noticias --file "noticias.json"`.



```
root@mongo ~# mongoimport --jsonArray --db database --collection noticias --file "noticias.json"
2022-12-19T12:48:14.715+0000 connected to: mongodb://localhost/
2022-12-19T12:48:15.229+0000 4 document(s) imported successfully. 0 document(s) failed to import.
root@mongo ~#
```

Figura 12. Importación de la base de datos.



```
root@2-mongo ~# mongo
MongoDB shell version v4.4.4
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("e6d7336a-1c7d-431c-b93c-3a92bebc27") }
MongoDB server version: 4.4.4
---
The server generated these startup warnings when booting:
  2022-12-16T12:58:19.068+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
  2022-12-16T12:58:20.777+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
---
> show dbs
admin      0.000GB
config     0.000GB
database   0.000GB
local      0.000GB
> show collections
> use database
switched to db database
> show collections
noticias
```

Figura 13. Elementos esenciales en nuestra infraestructura de MongoDB.

Tal y como su propio nombre indica, los documentos pertenecientes a la colección representan noticias que vendrán caracterizadas por, además del id que ya vimos que es obligatorio, los siguientes atributos característicos de cualquier noticia: title (título), date (fecha de publicación), photo (enlace a una imagen representativa), description (breve resumen del contenido de la noticia) y url (enlace para acceder a la noticia). De este modo estamos imitando a los principales sitios de noticias al listarlas incluyendo una imagen y su cabecera (título y entrada de la noticia).

```
> db.noticias.find()
{ "_id" : "3", "title" : "Cayendo en las redes de Tensor Flow ft. Keras", "date" : "13/12/2022", "photo" : "https://pbs.twimg.com/media/FjuBXW1XgAWJR8z7format-jpg?name=small", "description" : "Introducción al mundo del aprendizaje profundo, donde conocer como Tensorflow, el framework de Google dedicado a Deep Learning, facilita al usuario la realización de modelos de inteligencia artificial. Además, pondremos en práctica lo aprendido con un pequeño taller en el que llevaremos a cabo una tarea de clasificación sencilla donde entender mejor como funcionan estos algoritmos.", "url" : "https://gdg.community.dev/events/details/google-gdg-sevilla-presents-cayendo-en-las-redes-de-tensorflow-ft-keras/" }
{ "_id" : "0", "title" : "España cae ante Marruecos en octavos", "date" : "10/12/2022", "photo" : "https://images.daznservices.com/di/library/DAZN_News/C/23/marruecos-espana-mundial-2022_d0eh0apv4fpx1gw7hmu0hujao.jpg?t=-400690756", "description" : "Marruecos accedió a los cuartos de final del Mundial de Qatar 2022 después de imponerse a España en un partido marcado por la tremenda igualdad entre ambos conjuntos que se resolvió en la tanda de penaltis.", "url" : "https://www.dazn.com/es-ES/news/f%C3%BAtbol/marruecos-vs-espana-octavos-final-mundial-qatar-2022-goles-resumen-highlights/eljbfk2j4xlc1wt7o4dcm5dz6" }
{ "_id" : "1", "title" : "La UE impondrá un arancel climático para gravar el CO2 de las importaciones", "date" : "11/12/2022", "photo" : "https://imagenes.elpais.com/resizer/UKTI80bDLVSnjv2yz3xeFwsV6XU=/1200x0/cloudfront-eu-central-1.images.arcpublishing.com/prisa/OHAHYNU53VAYNH2DEKREPLIW4.jpg", "description" : "La nueva tasa afectará a productos como el hierro, el acero, el cemento, el aluminio, los fertilizantes y la electricidad, y se empezará a aplicar gradualmente a partir de octubre 2023.", "url" : "https://elpais.com/clima-y-medio-ambiente/2022-12-13/la-ue-impondra-un-arancel-climatico-para-gravar-el-co-de-las-importaciones.html" }
{ "_id" : "2", "title" : "No solo leones, osos y gladiadores: los perros salchicha también salían a la arena del Coliseo romano", "date" : "13/12/2022", "photo" : "https://imagenes.elpais.com/resizer/GSHLPb8uYZa7xfqWYHImhvXUfDw=/1200x0/cloudfront-eu-central-1.images.arcpublishing.com/prisa/LMF5RB4SMNE3BJDEPAC6ZKE4V4.aspx", "description" : "Una reciente excavación en las alcantarillas del monumento halla restos de varios animales, grandes variedades de fruta, monedas e información clave sobre el sistema hidráulico de la época.", "url" : "https://elpais.com/cultura/2022-12-13/no-solo-leones-osos-y-gladiadores-los-perros-salchicha-tambien-salian-a-la-arena-del-coliseo-romano.html" }
```

Figura 14. Estructura de los documentos de la colección noticias.

```
> db.noticias.find({"_id":"2"})
{ "_id" : "2", "title" : "No solo leones, osos y gladiadores: los perros salchicha también salían a la arena del Coliseo romano", "date" : "13/12/2022", "photo" : "https://imagenes.elpais.com/resizer/GSHLPb8uYZa7xfqWYHImhvXUfDw=/1200x0/cloudfront-eu-central-1.images.arcpublishing.com/prisa/LMF5RB4SMNE3BJDEPAC6ZKE4V4.aspx", "description" : "Una reciente excavación en las alcantarillas del monumento halla restos de varios animales, grandes variedades de fruta, monedas e información clave sobre el sistema hidráulico de la época.", "url" : "https://elpais.com/cultura/2022-12-13/no-solo-leones-osos-y-gladiadores-los-perros-salchicha-tambien-salian-a-la-arena-del-coliseo-romano.html" }
```

Figura 15. Consultar una noticia particular.

Si quisiéramos centrarnos en una noticia en particular, no tendríamos más que hacer una consulta imponiendo la id como condición o cualquier otro campo que la distinga del resto. Por ejemplo:

De cara a nuestro servicio estas noticias serán a las que se acceda para mostrarlas a modo de página web usando Apache.

Para conectar *Flask* con *MongoDB* son necesarios dos archivos: *database.py* (Figura 13) y *app.py* (Figura 14). En el primer fichero mencionado se utilizará la librería de Python *PyMongo* para acceder a la base de datos que está en una dirección IP. En nuestro caso particular debemos acceder a la dirección del balanceador de carga de *MongoDB* cuya dirección es 192.168.100.138 y cuyo puerto por defecto es el 27017. De esta manera ya podemos acceder a la base de datos *database* donde se encuentra la colección de noticias. El otro archivo importante es, como hemos dicho antes, *app.py*, que es el que carga la página. Aquí tenemos varias funciones, entre otras, la función *noticias()* con la que accedemos a la colección de noticias llamando a la base de datos y con *render template* pasamos los datos de la tabla, obtenidos mediante *db.find()*, al archivo *html*.

```

import pymongo

class DataBase:
    collection: pymongo.collection.Collection

    def __init__(self):
        myclient = pymongo.MongoClient("mongodb://192.168.100.138",27017)
        db = myclient["database"]
        self.collection = db["noticias"]

    def get_collection(self):
        return self.collection

```

Figura 16. Fichero database.py

```

from flask import Flask, render_template, request, redirect, url_for
from database import DataBase
from utils import *
import pymongo

app = Flask(__name__)

def get_bd():
    return DataBase()

@app.route('/')
def hello():
    return render_template('index.html')

@app.route('/noticias')
def noticias():
    db = DataBase().get_collection();
    return render_template('noticias.html', tabla=db.find())

if __name__ == "__main__":
    app.run(debug = True)

```

Figura 17. Fichero app.py

5. Características de alta disponibilidad

Para la mejora de la disponibilidad se ha optado por el uso de balanceadores de carga y réplicas, tanto para el servidor apache como para la base de datos mongodb, realizándose 3 copias de cada uno de ellos. Se ha usado la herramienta nginx para la creación de los balanceadores de carga, además de herramientas de monitorización, como veremos más adelante. Como política de balanceo se ha optado por el método de menos conexiones, redirigiendo al servidor que se encuentre menos solicitado en cada momento. Además, con el objetivo de realizar una demo más visual, se ha modificado la réplica de cada uno de los servidores apache y mongodb, para poder ver cómo cada vez que se realiza una petición nos redirige a una distinta. Evidentemente, en el caso de un despliegue real, esto no sería así y todas las réplicas serían iguales. Como ejemplo podemos ver el código del balanceador apache en el que le indicamos las ips a balancear y el método a usar.

```
server {  
    listen 127.0.0.1:80;  
    server_name 127.0.0.1;  
    location /nginx_status {  
        stub_status;  
        allow 127.0.0.1;  
        deny all;  
    }  
}
```

Figura 18. Código balanceador de carga.

Por otro lado, sobre el Fault Tolerance del sistema, si uno de los servidores se cayese, el balanceador envía probe request periódicamente, para comprobar que, efectivamente los servidores están disponibles y, al comprobar que hay uno que no lo está, dejaría de hacerle peticiones. Por tanto, la caída de éste no sería percibida por el usuario al tener dos réplicas todavía intactas. En el momento en el que el servidor volviera a responder, sería incluido de nuevo en el balanceo de carga.

Para este apartado se han realizado pruebas de capacidad con la herramienta estudiada en el laboratorio de la asignatura *Apache Benchmark*, la cual permite realizar pruebas de carga. Vamos a ilustrar un ejemplo hecho sobre el balanceador de carga de *Apache*, siendo análogo para el balanceador de *MongoDB*.


```
#!/bin/bash
for i in `seq 1 1 15`;
do
    echo $i
    sleep 10
    ab -n $((2**$i*1000)) -c $((2*$i)) http://192.168.100.139/ >>exp$i.1
    sleep 10
    ab -n $((2**$i*1000)) -c $((2*$i)) http://192.168.100.139/ >>exp$i.2
    sleep 10
    ab -n $((2**$i*1000)) -c $((2*$i)) http://192.168.100.139/ >>exp$i.3
done
```

Figura 19. Script usado para la prueba de capacidad del balanceador de carga de Apache.

Como ejemplo nos podemos fijar en los tests para 3 y 8 usuarios. Como podemos observar, el servidor sigue respondiendo, pero al aumentar el número de usuarios cada vez tarda más en responder. Es cuando alcanzamos aproximadamente doce usuarios cuando los servidores dejan de responder, por lo que sería el momento de escalar el sistema.

```
GNU nano 6.2 exp3.2
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.100.139 (be patient)

Server Software:      nginx/1.14.0
Server Hostname:      192.168.100.139
Server Port:          80

Document Path:        /
Document Length:      1380 bytes

Concurrency Level:     6
Time taken for tests:  2.109 seconds
Complete requests:     8000
Failed requests:        0
Total transferred:     12568000 bytes
HTML transferred:      11040000 bytes
Requests per second:   3792.55 [#/sec] (mean)
Time per request:      1.582 [ms] (mean)
Time per request:      0.264 [ms] (mean, across all concurrent requests)
Transfer rate:         5818.45 [Kbytes/sec] received

Connection Times (ms)
  min  mean[+/-sd] median  max
Connect:    0    0  0.0      0    1
Processing:  1    1  0.3      1    5
Waiting:    1    1  0.3      1    5
Total:      1    2  0.3      2    5
```

Figura 20. Test para tres usuarios simultáneos.

```

GNU nano 6.2 exp8.2
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.100.139 (be patient)

Server Software:      nginx/1.14.0
Server Hostname:      192.168.100.139
Server Port:          80

Document Path:        /
Document Length:       1380 bytes

Concurrency Level:    16
Time taken for tests:  125.466 seconds
Complete requests:     256000
Failed requests:        0
Total transferred:     402176000 bytes
HTML transferred:      353280000 bytes
Requests per second:   2040.39 [#/sec] (mean)
Time per request:      7.842 [ms] (mean)
Time per request:      0.490 [ms] (mean, across all concurrent requests)
Transfer rate:         3130.32 [Kbytes/sec] received

Connection Times (ms)
      min    mean[+/-sd] median   max
Connect:    0      0   0.1      0      3
Processing:  1      8   4.4      7     131
Waiting:    1      8   4.4      7     131
Total:      1      8   4.4      7     131

```

Figura 21. Test para ocho usuarios simultáneos.

6. Escalabilidad

La idea original para manejar la escalabilidad era utilizar Docker, para hacer uso del Docker Swarm. De hecho, llegamos a realizar una imagen de Docker con todo el contenido necesario para la página web. Sin embargo, esta opción tuvo que ser desechada, debido a los problemas de Proxmox y Docker por la persistencia de los contenedores, que podría hacer demasiada compleja la gestión de la red. Tras barajar múltiples opciones, la elegida para la replicación fue crear una plantilla con el servidor web configurado y clonarla. Así, si se quisieran más servidores de *Apache* de los que ya tenemos, simplemente habría que hacer copias de la plantilla que tenemos y modificar el balanceador de carga para que los soporte.

No obstante, aunque esto sea posible, lo ideal es conseguir hacerlo de forma automática, para ello creamos un script para automatizar el despliegue de contenedores creando el archivo *main.tf* con tantas réplicas como se le indicara; sin embargo, por temas de permisos, no era posible acceder a las plantillas del servidor configurado ni de la base de datos completa. Por temas de tiempo y dificultades se optó a realizar la clonación manual.

```
automatiza.sh
1  #!/bin/bash
2
3  echo "terraform {
4      required_version = ">= 1.1.0"
5      required_providers {
6          proxmox = {
7              source = "telmate/proxmox"
8              version = ">= 2.9.5"
9          }
10     }
11 }
12
13 provider "proxmox" {
14     pm_tls_insecure = true
15     pm_api_url = 
16     pm_api_token_id= 
17     pm_api_token_secret=""
18     pm_otp = ""
19     pm_debug=true
20 }
21
22 resource "proxmox_lxc" "balanceador" {
23     target_node = 
24     hostname = "balanceador"
25     osteamplate = "local:vztmpl/ubuntu-18.04-standard_18.04.1-1_amd64.tar.gz"
26     // para el usuario root
27     password = 
28     unprivileged = 
29     pool = "G3"
30     start = true
31
32     rootfs {
33         storage = "local-raid1"
34         size = "8G"
35     }
36
37     network {
38         name = 
39         bridge = 
40         ip = 
41         gw = 
42     }
43 }
44
45
46 resource "proxmox_lxc" "balanceador-mongo" {
47     target_node = 
48     hostname = "balanceador-mongo"
49     osteamplate = "local:vztmpl/ubuntu-18.04-standard_18.04.1-1_amd64.tar.gz"
50     // para el usuario root
51     password = 
52     unprivileged = 
53     pool = "G3"
54     start = true
55
56     rootfs {
57         storage = "local-raid1"
58         size = "8G"
59     }
60
61     network {
62         name = 
63         bridge = 
64         ip = 
65         gw = 
66     }
67 }
68
69
70 " > main.tf
71
```

Figura 22. Script automatización (Parte I)

```

72 if [[ "$#" -eq 2 ]]; then
73     for i in `seq 1 1 $1`;
74     do
75         echo "resource proxmox_lxc" "mongodb"$i" {
76             target_node = 
77             hostname = "mongodb"$i"
78             ostemplate = "plantilla de mongo con los datos guardados"
79             // para el usuario root
80             password = 
81             unprivileged = 
82             pool = "G3"
83             start = true
84
85             rootfs {
86                 storage = "local-raid1"
87                 size = "8G"
88             }
89
90             network {
91                 name = 
92                 bridge = 
93                 ip = 
94                 gw = 
95             }
96         }
97     }
98     " >> main.tf
99     done
100     for i in `seq 1 1 $2`;
101     do
102         echo "resource proxmox_lxc" "ubuntu18-1-"$i" {
103             target_node = 
104             hostname = "ubuntu18-1-"$i"
105             ostemplate = "plantilla de ubuntu18.04 con el servidor web configurado"
106             // para el usuario root
107             password = 
108             unprivileged = 
109             pool = "G3"
110             start = true
111
112             rootfs {
113                 storage = "local-raid1"
114                 size = "8G"
115             }
116
117             network {
118                 name = 
119                 bridge = 
120                 ip = 
121                 gw = 
122             }
123         }
124     }" >> main.tf
125     done
126 elif [[ "$#" -eq 1 ]]; then
127     echo "Error: Falta el numero de replicas de apache"
128 fi

```

Figura 23. Script automatización (Parte II)

Hemos diseñado un balanceador de carga para las máquinas con MongoDB de la misma manera que el usado para los servidores Apache, aunque realmente no se necesite ya que el propio servicio de MongoDB permite la lectura y escritura de peticiones simultáneas gracias a su sistema “multi-granularity locking”.

De manera nativa MongoDB trae consigo un sistema de gestión de bases de datos activo-pasivo, donde la base de datos principal, también llamada primaria, será la encargada de gestionar todas las peticiones realizadas y donde una o varias máquinas en

estado pasivo, también llamadas secundarias, están esperando a que ocurra un fallo en la principal para que, de manera aleatoria (con un heartbeat), una de ellas pase a ser la nueva máquina principal.

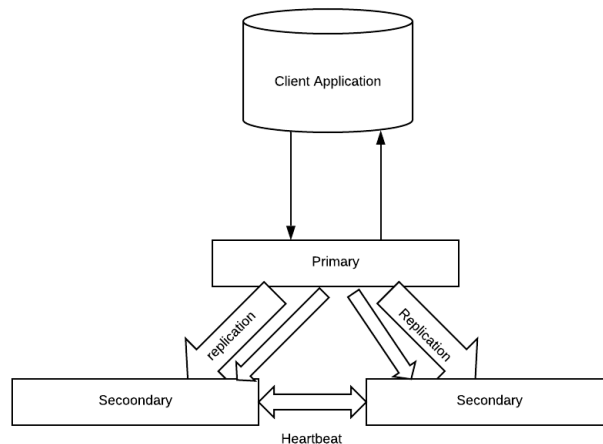


Figura 24. Esquema de conjunto de replicación de mongo

En un primer momento intentamos montar una arquitectura igual a la mostrada en la imagen anterior, pero se necesitaban de comandos que no se encontraban disponibles en los contenedores de mongo, además de una serie de permisos de lectura y escritura que no se conseguían cambiar, por lo que decidimos imitar lo usado con los servidores web ya que, aunque no se consiga una mejora sustancial en la velocidad de transferencia de datos, es posible que de esta forma se puedan evitar que problemas del entorno físico puedan afectar de manera simultánea a la estructura de bases de datos. Distanciar geográficamente cada máquina podría evitar este tipo de fallos, ya sea un apagón local o el derrame de cualquier sustancia líquida en todos los sistemas a la vez.

7. Mantenimiento

Respecto al mantenimiento del sistema tenemos la monitorización y aplicaciones como *Apache*, *NGINX* y *Flask*, así como las distintas copias de seguridad que se irán haciendo en caso de necesitarlas.

Proxmox ofrece la posibilidad de hacer distintas copias de seguridad gracias a los *snapshots*, que nos permiten volver al momento en el que los creamos.

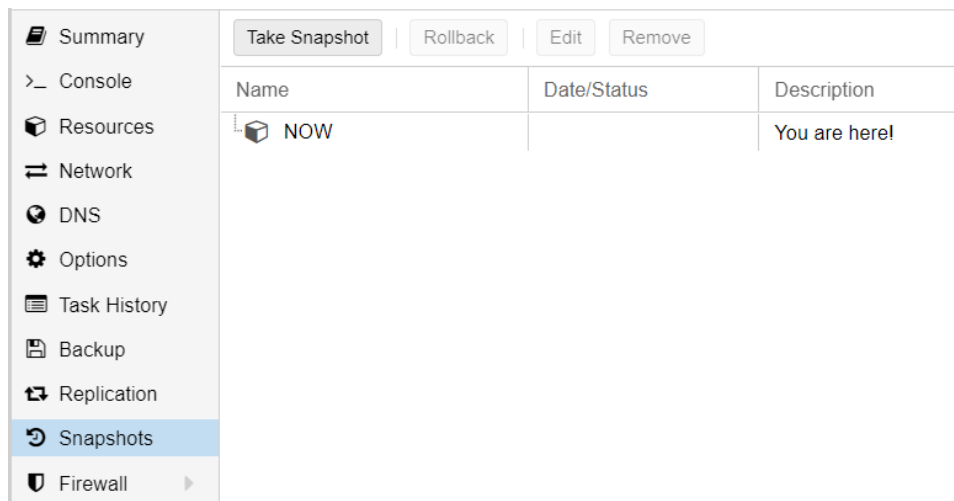


Figura 25. Snapshot de Proxmox.

En el caso de los servidores *Apache* y *MongoDB* bastaría con hacer copia de seguridad de las plantillas. En cuanto al orden del apagado, primero empezamos con el balanceador de apache, servidores de apache, balanceador de MongoDB y su base de datos, respectivamente.

Asimismo, también hemos hecho uso de los archivos de logs que nos proporcionan los distintos servicios usados. Éstos nos sirven principalmente para informarnos del estado de cada uno de los componentes de la arquitectura, de las peticiones recibidas y, en caso de error, indicarnos su causa. En un despliegue real, estos archivos serán muy útiles para el mantenimiento y durante la fase de desarrollo nos han permitido corregir errores del servidor apache y el balanceador nginx.

```
127.0.0.1 - - [19/Dec/2022:06:02:18 +0000] "GET /nginx_status HTTP/1.1" 200 118 "-" "nginx-amplify-agent/1.8.1-1"
127.0.0.1 - - [19/Dec/2022:06:02:38 +0000] "GET /nginx_status HTTP/1.1" 200 118 "-" "nginx-amplify-agent/1.8.1-1"
127.0.0.1 - - [19/Dec/2022:06:02:58 +0000] "GET /nginx_status HTTP/1.1" 200 118 "-" "nginx-amplify-agent/1.8.1-1"
127.0.0.1 - - [19/Dec/2022:06:03:18 +0000] "GET /nginx_status HTTP/1.1" 200 118 "-" "nginx-amplify-agent/1.8.1-1"
```

Figura 26. Log de acceso de NGINX.

Para la monitorización del sistema hemos utilizado la herramienta de *Amplify*, ya que es una herramienta desarrollada y soportada por *NGINX*, la cual es tecnología usada

para el balanceador de carga, donde está toda la posterior instalación configurada. Por tanto, *Amplify* era la mejor opción gracias a su integración con esta última. Además, para la monitorización de la base de datos, hemos utilizado *MongoDB*, lo cual explicaremos con más detalle posteriormente.

Para poder trabajar con *Amplify* era necesario acceder a la página oficial y crear una cuenta que permitiera acceder a los datos de visualización. La clara ventaja es que gracias al inicio de sesión con dicha cuenta en cualquier dispositivo (ordenador, dispositivos *Android*, ...) es posible acceder a los datos de monitorización y ver el estado del sistema. Una vez creada la cuenta se genera un API Key y permite descargar un *script* de instalación, que puede ser configurado mediante el API Key.

El *script* de instalación es genérico para cualquier sistema *Linux*, con comandos “estándar”. Esto dio lugar a varios problemas a la hora de configurarlo para este sistema particular, como el hecho de que determinados paquetes no se pudieran instalar. Los errores se debían a que los certificados de esos paquetes habían expirado, por lo que la solución era hacer cambios directamente desde el *script* de instalación para forzarlo a realizar instalaciones “no seguras” (las cuales, al ser dependencias de *NGINX*, realmente sí que contaban con ciertas garantías de seguridad). Respecto a los errores de certificados, se tuvo que instalar en la máquina *NGINX* paquetes de *CA Certificates* mediante el comando asociado a estos. Los errores de repositorio se solucionaron gracias al paquete *GNUpg2*.

Una vez instalado todo basta con modificar el archivo *Nginx*, que hace de balanceador, añadiéndole las siguientes líneas de código:

```
server{  
    listen 127.0.0.1:80;  
    server_name 127.0.0.1;  
    location /nginx_status {  
        stub_status;  
        allow 127.0.0.1;  
        deny all;  
    }  
}
```

Después de todo esto ya estaría todo listo para poder monitorizar nuestro servidor. Para ello accedemos a la página web con la cuenta que hemos creado y podremos encontrar el estado de la conexión.

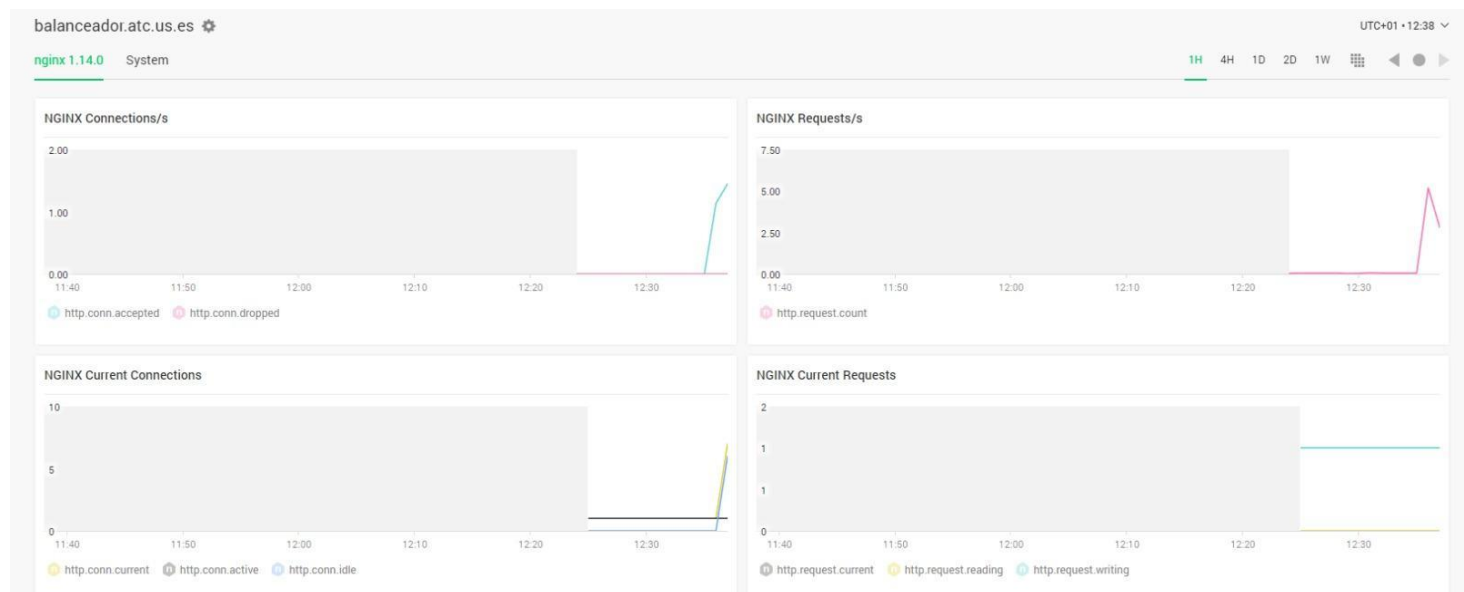


Figura 27. Ejemplo de monitorización del balanceador.

Utilizando *MongoDB* podemos también monitorizar nuestra base de datos, database en este caso, mediante el comando `db.enableFreeMonitoring()`.

```
> db.enableFreeMonitoring()
{
  "state" : "enabled",
  "message" : "To see your monitoring data, navigate to the unique URL below. Anyone you share the URL with will also be able to view this page. You can disable monitoring at any time by running db.disableFreeMonitoring().",
  "url" : "https://cloud.mongodb.com/freeMonitoring/cluster/V35F5JIP3MOZFFM3QNH25ND6VUUR165BH",
  "userReminder" : "",
  "ok" : 1
}
```

Figura 28. Comando para obtener enlace a los datos de monitorización.

A continuación, si accedemos a la url mostrada mediante nuestro navegador, accederemos a la herramienta *FreeMonitoring*, que nos presentará gráficas con diversas estadísticas de utilidad para tener información que nos permita medir el rendimiento de nuestro servidor, que, en el ejemplo mostrado, es el 1-mongo.

En primer lugar nos mostrará el estado actual de algunos parámetros fundamentales de medición. En particular destacamos que se nos indica que estábamos utilizando un 1.13% de la CPU de nuestro sistema en el momento de la captura que mostraremos próximamente. Asimismo, la herramienta nos ofrece ciertos enlaces de interés para documentarnos debidamente acerca de las diversas funcionalidades de MongoDB.

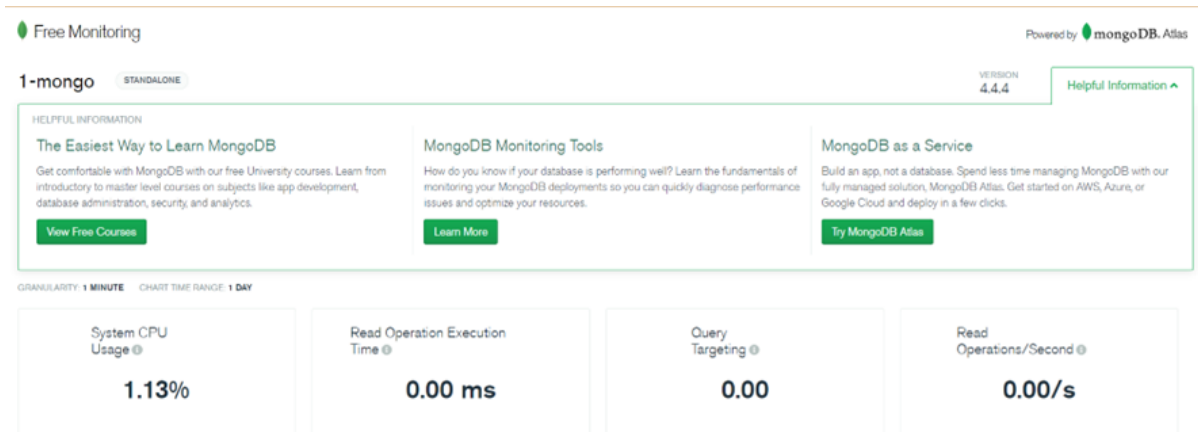


Figura 29. Cabecera de FreeMonitoring.

Si descendemos con el cursor se nos mostrarán diversos gráficos por horas junto con una leyenda explicativa que hace muy intuitiva y legible la información aportada por cada hora. Vamos a presentar aquellos gráficos más relevantes teniendo primero en cuenta que, antes de tomar las capturas que adjuntamos de los diversos gráficos, se ejecutaron los siguientes comandos aproximadamente a las 17h, que en el horario de la plataforma son las 4PM:

```
> show collections
noticias
> db.noticias.find()
{ "_id" : "0", "title" : "España cae ante Marruecos en octavos", "date" : "1/12/2022", "photo" : "https://images.daznservices.com/di/library/DAEN/News/c/23/marruecos-espana-mundial-2022_d0e0agv4fpxlgw7hmu0hujao.jpg?it=400690756", "description" : "Marruecos accedió a los cuartos de final del Mundial de Qatar 2022 después de imponerse a España en un partido marcado por la tremenda igualdad entre ambos conjuntos que se resolvió en la tanda de penaltis.", "url" : "https://www.dazn.com/es-ES/news/fKc3tBATbol/marruecos-vs-espana-octavos-final-mundial-qatar-2022-goles-resumen-highlights/eljbfk2j4xlc1wt7o4dc5dz6" }
{ "_id" : "1", "title" : "La UE impondrá un arancel climático para gravar el CO2 de las importaciones", "date" : "11/12/2022", "photo" : "https://imagenes.elpais.com/resizer/UKTIB0b0LWSnjv2yz3xeFwSv6XU-/1200x0/cloudfront-eu-central-1.images.arcpublishing.com/prisa/OH8AHYNU53VAYNH2DEKDRPLW4.jpg", "description" : "La nueva tasa afectará a productos como el hierro, el acero, el cemento, el aluminio, los fertilizantes y la electricidad, y se empezará a aplicar gradualmente a partir de octubre 2023.", "url" : "https://elpais.com/clima-y-medio-ambiente/2022-12-13/la-ue-impondra-un-arancel-climatico-para-gravar-el-co-de-las-importaciones.html" }
{ "_id" : "2", "title" : "No solo leones, osos y gladiadores: los perros salchicha también salían a la arena del Coliseo romano", "date" : "13/12/2022", "photo" : "https://imagenes.elpais.com/resizer/GSHLPb8uY2a7xfqWYHmhvXUfDw-/1200x0/cloudfront-eu-central-1.images.arcpublishing.com/prisa/LMF5RB43MNE3BJDEPAC6EKE4V4.aspx", "description" : "Una reciente excavación en las alcantarillas del monumento halla restos de varios animales, grandes variedades de fruta, monedas e información clave sobre el sistema hidráulico de la época.", "url" : "https://elpais.com/cultura/2022-12-13/no-solo-leones-osos-y-gladiadores-los-perros-salchicha-tambien-salian-a-la-arena-del-coliseo-romano.html" }
{ "_id" : "3", "title" : "Cayendo en las redes de Tensor Flow ft. Keras", "date" : "13/12/2022", "photo" : "https://pbs.twimg.com/media/Fju8X0W1XgAwJf8z7format-jpg?name=small", "description" : "Introducción al mundo del aprendizaje profundo, donde conocer como Tensorflow, el framework de Google dedicado a Deep Learning, facilita al usuario la realización de modelos de inteligencia artificial. Además, pondremos en práctica lo aprendido con un pequeño taller en el que llevaremos a cabo una tarea de clasificación sencilla donde entender mejor como funcionan estos algoritmos.", "url" : "https://gdg.community.dev/events/details/google-gdg-sevilla-presents-cayendo-en-las-redes-de-tensorflow-ft-keras/" }
> db.noticias.find({"_id":3})
{ "_id" : "3", "title" : "Cayendo en las redes de Tensor Flow ft. Keras", "date" : "13/12/2022", "photo" : "https://pbs.twimg.com/media/Fju8X0W1XgAwJf8z7format-jpg?name=small", "description" : "Introducción al mundo del aprendizaje profundo, donde conocer como Tensorflow, el framework de Google dedicado a Deep Learning, facilita al usuario la realización de modelos de inteligencia artificial. Además, pondremos en práctica lo aprendido con un pequeño taller en el que llevaremos a cabo una tarea de clasificación sencilla donde entender mejor como funcionan estos algoritmos.", "url" : "https://gdg.community.dev/events/details/google-gdg-sevilla-presents-cayendo-en-las-redes-de-tensorflow-ft-keras/" }
> db.noticias.find({"_id":2})
{ "_id" : "2", "title" : "No solo leones, osos y gladiadores: los perros salchicha también salían a la arena del Coliseo romano", "date" : "13/12/2022", "photo" : "https://imagenes.elpais.com/resizer/GSHLPb8uY2a7xfqWYHmhvXUfDw-/1200x0/cloudfront-eu-central-1.images.arcpublishing.com/prisa/LMF5RB43MNE3BJDEPAC6EKE4V4.aspx", "description" : "Una reciente excavación en las alcantarillas del monumento halla restos de varios animales, grandes variedades de fruta, monedas e información clave sobre el sistema hidráulico de la época.", "url" : "https://elpais.com/cultura/2022-12-13/no-solo-leones-osos-y-gladiadores-los-perros-salchicha-tambien-salian-a-la-arena-del-coliseo-romano.html" }
> db.noticias.find({"_id":1})
{ "_id" : "1", "title" : "La UE impondrá un arancel climático para gravar el CO2 de las importaciones", "date" : "11/12/2022", "photo" : "https://imagenes.elpais.com/resizer/UKTIB0b0LWSnjv2yz3xeFwSv6XU-/1200x0/cloudfront-eu-central-1.images.arcpublishing.com/prisa/OH8AHYNU53VAYNH2DEKDRPLW4.jpg", "description" : "La nueva tasa afectará a pr
```

Figura 30 Comandos (peticiones) previos a la monitorización.

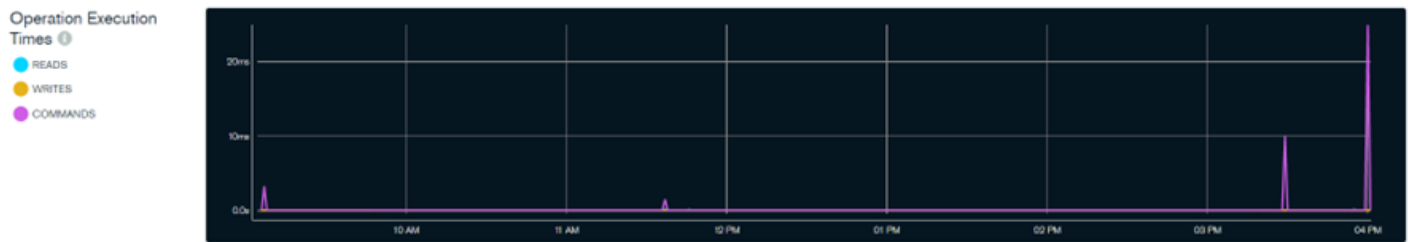


Figura 31. Tiempo de ejecución de las operaciones realizadas.

Dado que las operaciones realizadas no fueron de lectura ni escritura su velocidad es nula. En cambio, podemos observar que, al ejecutar el conjunto de comandos mostrado anteriormente, como eran varias operaciones y muy seguidas, el tiempo de operación fue el más elevado. Si posicionamos el cursor sobre este pico de la gráfica, obtenemos que dicho tiempo es 25ms.

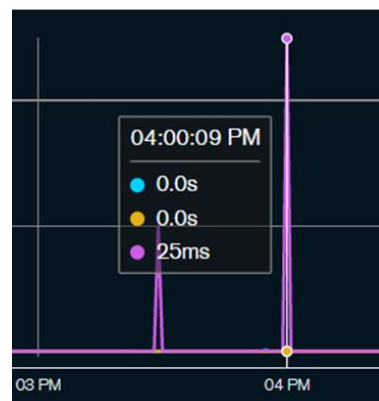


Figura 32. Tiempo de operación a las 4PM.



Figura 33. Utilización del disco.

Aquí podemos comprobar el porcentaje máximo de tiempo durante el cual la partición emite y atiende solicitudes frente al porcentaje medio de tiempo dedicado a esto mismo. Se incluyen las solicitudes de cualquier proceso, no solo los procesos de MongoDB.

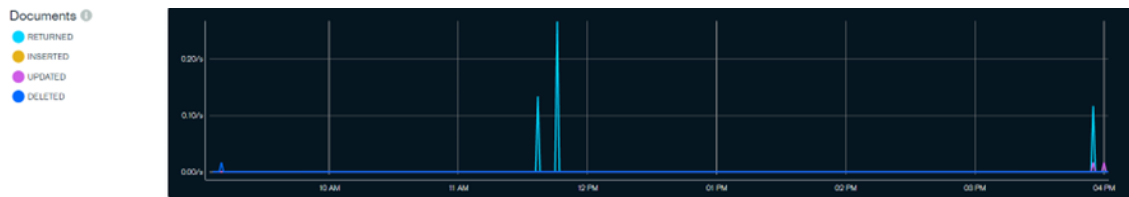


Figura 34. Tratamiento de documentos.

Aquí se refleja que no se hizo ninguna modificación a los documentos presentes en la BD, sino que únicamente se realizaron consultas.



Figura 35. Memoria.

Como cabía esperar, vemos que los bytes utilizados por la memoria virtual son muy superiores a los correspondientes a la memoria física. La línea de MAPPED es nula, lo que es coherente con la explicación que nos ofrece la plataforma.

● Mapped

The WiredTiger storage engine does not use memory mapped files, so this should be 0 or close to 0.

Figura 36. Sobre los archivos mapeados en memoria.



Figura 37. Bytes de E/S.

Aquí se están midiendo el número de bytes físicos enviados desde y hacia nuestro servidor de base de datos.

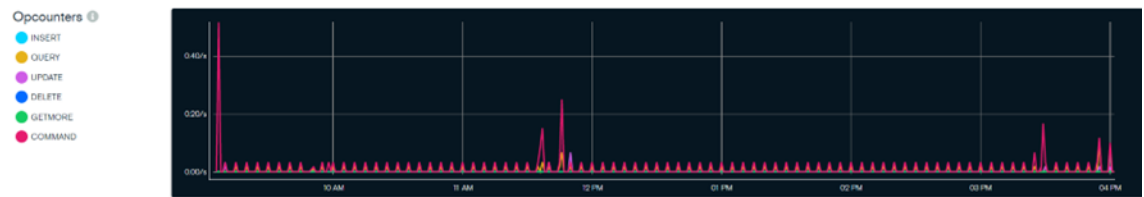


Figura 38. Uso medio de cada operación.

Los picos que observamos se deben a peticiones en algunos momentos puntuales. Por lo general, la tasa de comandos por segundos sigue una cierta periodicidad. Evidentemente hay valores algo más elevados en momentos puntuales en los que se han ejecutado algunas operaciones. Por ejemplo, los que realizamos justo antes de tomar las capturas.

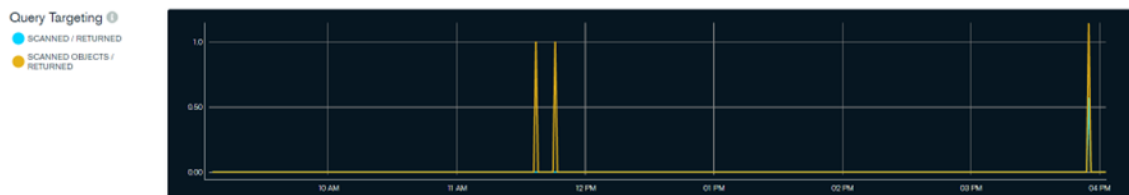


Figura 39. Relación entre elementos escaneados y devueltos.

Esta gráfica nos indica cuántos índices (azul) y documentos (amarillo) debe escanear nuestra query en media, hasta encontrar uno que devolver porque coincida con la condición impuesta en la petición. Dado que nuestra base de datos solo tiene una colección de 6 documentos, es coherente que estos valores sean muy pequeños.

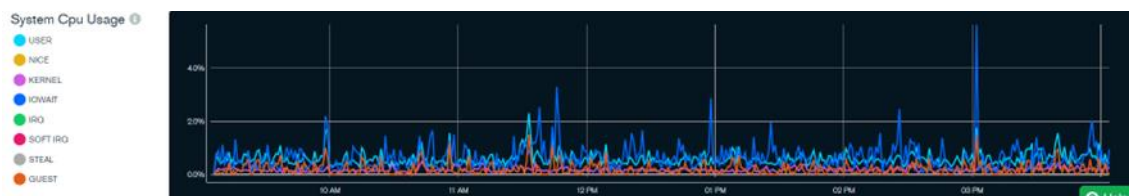


Figura 40. Uso de la CPU del sistema.

Aquí, tal y como su propio nombre indica, podemos consultar el consumo de la CPU por parte de diversos servicios y/o operaciones. De hecho, en cualquier gráfica, para tener una mayor comprensión de los datos mostrados, si hacemos click en el icono presente junto al nombre de cada uno de los títulos de las gráficas, accederemos a información detallada sobre la leyenda de dicha gráfica. Por ejemplo, veamos lo que se

nos mostraría en este caso, que nos ha parecido el más apropiado al ser el que cuenta con más elementos en la leyenda.

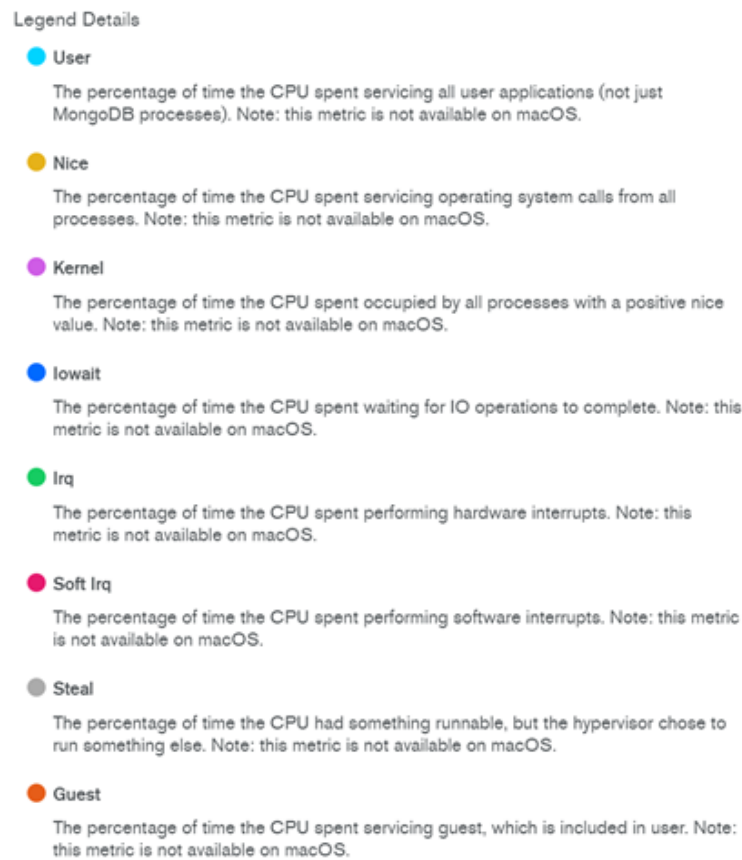


Figura 41. Descripción de la leyenda de uso de la CPU de sistema.

Para finalizar concluimos la monitorización mediante el comando `db.disableFreeMonitoring()`.

```
> db.disableFreeMonitoring()
```

Figura 42. Fin de la monitorización.

Si quisiéramos modificar el servidor o la base de datos tenemos que, en primer lugar, realizar una clonación completa (no con vinculación) de la plantilla, realizar ahí las modificaciones pertinentes y volver a convertir en plantilla. Esta nueva plantilla se replicará como antes y ya tendremos el servicio actualizado.

8. Limitaciones, trabajo futuro y conclusiones

Como en cualquier trabajo de despliegue, no todas las ideas que se tenían en un primer momento han sido exitosas. Sin embargo, a pesar de los problemas que han ido surgiendo a lo largo de todo este proceso, se ha conseguido llegar al resultado buscado. A continuación se comentarán las diferentes dificultades que se han tenido que afrontar y sus respectivas soluciones, así como algunas alternativas que se llegaron a plantear pero que no se llevaron a cabo.

8.1. Limitaciones

En primer lugar, hemos hecho uso de Terraform para lanzar contenedores y máquinas virtuales y es aquí donde comienzan nuestros problemas. Recordemos que Terraform cuenta con un fichero `main.tf` que es donde está guardada la configuración que deseamos tener para crear el Terraform en el Proxmox. Para poder ejecutar dicho archivo es necesario hacerlo desde una consola de comandos en la carpeta que contenga a Terraform, que, por tanto, también contiene el fichero `main.tf`. Para inicializar Terraform es necesario ejecutar el comando *terraform init* y, al hacerlo la primera vez, se nos crearán ciertos archivos de control. De éstos, al que le prestaremos más atención será al `terraform.tf.state`, ya que es el que ha impedido en algún momento el avance del despliegue. Este archivo indica cómo se encuentra la estructura del Proxmox creada con Terraform en dicho momento, sin embargo, no se actualiza en tiempo real, sino que lo hace al ejecutar el comando *terraform plan* o *terraform apply*. Por ejemplo, una de las primeras veces, lo que ocurrió fue que se había estropeado la máquina virtual y se tuvo que eliminar. Para ello se utilizó la interfaz gráfica de Proxmox y, al ejecutar *Terraform plan*, aparecen errores del tipo: falta un id, no se encuentra la máquina... Y estos errores no permiten la realización de ninguna otra acción dentro de Terraform.

Frente al problema mencionado anteriormente, en un principio, al no haber introducido todavía ninguna clase de dato, lo que se hacía para solucionarlo era eliminar todo y volver a iniciarlo cada vez que apareciese el error del *terraform plan*, sin embargo, esa idea en el momento en el que se introdujeran datos, era inviable puesto que éstos se perderían. Otra solución que se planteó fue abrir el fichero de estado, *terraform.tf.state* y borrar, manualmente, la información correspondiente. Sin embargo, sospechamos que era probable que, además de eso, fuera necesario modificar la copia de seguridad, así como algún que otro archivo, de manera que finalmente se descartó esa posibilidad. Es por ello que se decidió utilizar la interfaz gráfica de Proxmox sólo cuando fuese estrictamente necesario.

Por otro lado, durante el trabajo de despliegue, hemos tenido muchos problemas con los permisos con Proxmox. Primeramente, para la creación de máquinas virtuales y contenedores desde Terraform en Proxmox, eran necesarias una serie de plantillas que nos han sido proporcionadas por los profesores de la asignatura, sin embargo, cuando ya se le habían realizado los cambios necesarios para su configuración e intentábamos

convertirlo en una plantilla para lanzar su replicación con un script, no se guardaba en la misma ruta que nos proporcionaban los profesores sino que se creaba una nueva ruta a la que no teníamos acceso. A esa ruta sólo tenía acceso el usuario root e, incluso a pesar de que nos habían proporcionado todos los permisos posibles, era imposible acceder a ella. Así que, finalmente, decidimos hacerlo manualmente, clonando la plantilla desde el Proxmox, teniendo en cuenta que para establecer que algo era una plantilla había que modificar el `main.tf` e indicarlo, puesto que si no produciría errores en la replicación.

Otro de los problemas con los que nos hemos encontrado era que, al crear nuestra máquina virtual, había que asignarle, por requisitos de la asignatura, un pool, en nuestro caso el G3. Para ello, en la configuración de la máquina, hay un apartado en el que se indica el pool de la siguiente manera: `pool = 'G3'`. Al hacerlo y crear la máquina virtual por primera vez, no se producía ningún error, sin embargo, el problema aparecía después. Resulta que si volvíamos a ejecutar *terraform apply* para crear más contenedores o máquinas, terraform reconocía, por algún motivo, lo de poner el pool igual a G3 como un cambio, y, al ejecutar dicho comando saltaba un error de permisos de que sólo root puede cambiar los pools. Esto no suponía un problema realmente, puesto que no impedía seguir utilizando la máquina ni la creación de otras, pero era bastante molesto que saltara error cada vez que se ejecutaba. Después de muchos intentos, descubrimos que si una vez ya creado el pool, se borra el código de `pool=G3`, deja de ocurrir dicho error y todo funciona como esperamos.

Por otro lado, en un principio habíamos planteado este trabajo de despliegue utilizando contenedores Docker, puesto que nos eran familiares por haber sido utilizados en la asignatura. Además, en caso de utilizar contenedores Docker, podríamos hacer uso también de Docker Swarm que es una herramienta de orquestación de contenedores, y nos permite manejar bien las réplicas de una manera sencilla. Es, por ello, que nuestra idea consistía en disponer de varias máquinas virtuales con Docker e instalar directamente la imagen del contenedor del repositorio de Docker, imagen con todos los datos ya introducidos y, una vez hecho esto, replicarlo dentro de cada máquina virtual. Aparentemente el proceso parecía bastante sencillo, lo único que, al hacerlo así, las máquinas virtuales consumen tantos recursos que posiblemente se saturaría el host de Proxmox. Pero ése no era el mayor de los problemas y es que, el verdadero motivo por el que esta idea no era factible, es por el problema de la persistencia de los datos lo que complicaría mucho la gestión de la red y los contenedores. Finalmente, los profesores nos recomendaron el uso de contenedores LXC y fueron ellos quienes nos proporcionaron las plantillas necesarias.

También cabe mencionar los problemas de compatibilidad que han surgido con el uso del servidor Apache, que no permitía el acceso a la base de datos MongoDB. Para solucionar esto, en primer lugar se optó por el uso de Node.js, pero fue rápidamente descartado debido a la gran cantidad de problemas que generaba su uso. Tras esto, se optó por lo que sería la solución definitiva: realizar la conexión y configuración de la página web mediante un servidor Flask, que haciendo uso de VirtualHost, permite comunicarnos con él como si fuera Apache.

8.2. Trabajo futuro

Respecto al trabajo futuro lo principal sería conseguir una IP pública para poder realizar el despliegue en la nube. Pese a que este proyecto está diseñado justo para ello, por motivos administrativos no hemos podido conseguir este objetivo, debido a la falta de tiempo para solicitarla. No obstante, una vez conseguida dicha IP y teniendo los puertos correspondientes abiertos, el despliegue en la nube es trivial.

Por otro lado, por cuestiones también temporales, no hemos podido desarrollar una forma de escalar el sistema de forma automática, sino que nos hemos centrado simplemente en un modelo manual. Para poder automatizar esta tarea, debemos construir un *cluster* de *Kubernetes* usando *k3s* en *Promox* gracias a *Ansible* y *Terraform*.

También sería muy recomendable almacenar copias de seguridad fuera de Proxmox, para que en caso de que se estropee la infraestructura, por ejemplo, por un cortocircuito, inundación, etc., no se pierdan todas las copias.

Otra propuesta sería desarrollar una interfaz web que permita acceder de manera centralizada a los dashboards de salud de los servidores web y bases de datos para así crear un cuadro de mando único al que acceder y tener una visión global del sistema.

Por último se podría desarrollar un script de parada y arranque del sistema que sea secuencial para evitar errores de dependencias a la hora de parar el sistema para realizar mantenimientos perfectivos o correctivos. Esto es, cerrar primero el puerto de comunicación con el exterior (si estuviera abierto), a continuación cerrar la interfaz web y luego la base de datos.

8.3. Conclusiones finales

Este trabajo nos ha servido para enfrentarnos a un problema real y nos ha obligado a desarrollar un servicio TI que cumpla unos estándares mínimos de calidad. A lo largo del proyecto hemos aprendido a resolver problemas, coordinar distintos equipos con propósitos diferentes, pero relacionados y hemos aprendido muchas tecnologías nuevas. Adicionalmente, nos vemos más capaces de tomar decisiones relacionadas con el desarrollo de un servicio web y de gestionar los objetivos mínimos que deben cumplirse.

Con respecto a la tecnología, Flask resulta ser una herramienta adecuada y de fácil uso. No habría necesidad de envolver la aplicación Flask con el servidor Apache, pero debido a las especificaciones dadas al principio del trabajo, decidimos dejarlo así.

9. Referencias

Esta es la documentación consultada para la parte de Mongo:

- [1] <https://www.iebschool.com/blog/bases-de-datos-nosql-vs-sql-big-data/>
- [2] <https://mongoosejs.com/docs/>
- [3] <https://ayudaleyprotecciondatos.es/bases-de-datos/no-relacional/>
- [4] <https://www.mongodb.com/home>
- [5] <https://geekflare.com/es/mongodb-queries-examples/>
- [6] <https://byspel.com/mongodb-eliminar-documentos-y-colecciones/>
- [7] <https://programmerclick.com/article/44385351/>
- [8] [https://ayudaleyprotecciondatos.es/bases-de-datos/relacional/#Que es una base de datos relacional Definicion](https://ayudaleyprotecciondatos.es/bases-de-datos/relacional/#Que-es-una-base-de-datos-relacional-Definicion)
- [9] <https://stackoverflow.com/questions/51417708/unable-to-install-mongodb-properly-on-ubuntu-18-04-lts>
- [10] <https://github.com/marsamber/CBDNodeMongoose>

Sobre PyMongo:

- [11] https://www.w3schools.com/python/python_mongodb_find.asp
- [12] <https://pymongo.readthedocs.io/en/stable/index.html>

Esto fue necesario para Flask:

- [13] <https://flask-es.readthedocs.io/>
- [14] <https://techtutorguro.com/how-to-install-flask-on-ubuntu-22-04-with-apache/>

Sobre la idea de ejecutar código python dentro de javascript:

- [15] <https://www.analyticslane.com/2022/06/10/comenzando-con-pyscript-ejecutar-python-en-un-navegador/>

Para Docker:

[16] <https://docs.docker.com/>

[17] <https://hub.docker.com/repository/docker/marlinbar/webserver>

Para la idea de utilizar iptables para el balanceo:

[18] <https://gist.github.com/apparentlymart/d8ebc6e96c42ce14f64b>

[19] https://wiki.nftables.org/wiki-nftables/index.php/Load_balancing

Para trabajar en local:

[20] <https://code.visualstudio.com/docs/remote/ssh>

Para el balanceador de cargas:

[21] <https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/>

[22] <https://docs.nginx.com/nginx/admin-guide/load-balancer/tcp-udp-load-balancer/>

Instalación de amplify (monitorización nginx):

[23] <https://amplify.nginx.com/dashboard>

[24] <https://es.linux-console.net/?p=2019#gsc.tab=0>

[25] <https://raw.githubusercontent.com/nginxinc/nginx-amplify-agent/master/packages/install.sh>

[26] <https://github.com/nginxinc/nginx-amplify-agent/issues/75>

Para arreglar problemas de certificado expirado:

[27] https://askubuntu.com/questions/1095266/apt-get-update-failed-because-certificate-verification-failed-because-handshake?_gl=1*_1qjtlab*_ga*MTg4NzY1NzQ1Mi4xNjc1MTUx*_ga_S812YQPLT2*MTY3MTI3NTE1MS4xLjAuMTY3MTI3NTE1Mi4wLjAuMA

[28] <https://serverfault.com/questions/1093511/apt-get-update-failing-because-of-certificate-validation>

Para forzar instalaciones “no seguras”:

[29] <https://askubuntu.com/questions/732985/force-update-from-unsigned-repository>

Para Terraform:

[30] <https://registry.terraform.io/providers/Telmate/proxmox/latest/docs>

[31] <https://registry.terraform.io/providers/Telmate/proxmox/latest/docs/resources/lxc>

[32] https://registry.terraform.io/providers/Telmate/proxmox/latest/docs/resources/vm_gemu

[33] <https://austinsnerdythings.com/2021/09/01/how-to-deploy-vms-in-proxmox-with-terraform/>

Para replicación de MongoDB:

[34] <https://www.mongodb.com/docs/manual/replication/>

ANEXO. Acerca de MongoDB

Como mencionamos previamente, en este trabajo se ha optado por investigar sobre MongoDB. Si accedemos a su web, textualmente se describen a sí mismos como: *“La plataforma de datos para desarrolladores que proporciona los servicios y las herramientas necesarios para crear aplicaciones distribuidas rápidamente, con el rendimiento y la escala que exigen los usuarios.”*. Esto claramente avala los beneficios de las bases de datos NoSQL que hemos comentado.

En primer lugar, es necesario tener una noción básica de la terminología a utilizar al hablar de bases de datos de MongoDB. Para hacernos una idea en relación con lo expuesto en el epígrafe anterior, podemos establecer las siguientes equivalencias: donde antes teníamos tablas divididas en filas, ahora nos encontramos con colecciones divididas en documentos.

Otros puntos básicos por considerar son el hecho de que las bases de datos utilizan formato BSON y que cualquier documento debe tener como primer campo `_id`, el cual deberá ser único dentro de la colección.

```
var mydoc = {
  _id: ObjectId("5099803df3f4948bd2f98391"),
  name: { first: "Alan", last: "Turing" },
  birth: new Date('Jun 23, 1912'),
  death: new Date('Jun 07, 1954'),
  contribs: [ "Turing machine", "Turing test", "Turingery" ],
  views : NumberLong(1250000)
}
```

Figura 43. Ejemplo de un documento.

Una vez establecidas las características fundamentales de MongoDB, vamos a estudiar una serie de comandos básicos que nos serán fundamentales para poder operar en Linux:

Inicializar el servicio: `sudo systemctl start mongod`

Detener el servicio: `sudo systemctl stop mongod`

Comprobar el estado del servicio: `sudo systemctl status mongod`

Reiniciar el servicio: `sudo systemctl restart mongod`

Una vez inicializado y en funcionamiento, serán de vital importancia los siguientes comandos:

Obtención del listado de comandos completo: `db.help()`

Creación de una base de datos o cambio a una existente: por defecto, todos los usuarios usan la base de datos testDB al inicializar el servicio, por lo que es necesario cambiar de base de datos, lo cual es posible con el comando use DB_Name (donde DB_Name es el nombre de la base de datos que queramos utilizar)

Listado de bases de datos: show dbs

A partir de ahora supongamos que hemos decidido utilizar la base de datos DB_Name con el comando use DB_Name. Veamos qué podemos hacer:

Borrar base de datos: db.dropDatabase()

Crear una colección: db.createCollection(nombre_coleccion,opciones)

Ver todas las colecciones: show collections

Borrar una colección: db.nombre_coleccion.drop()

Insertar documentos a una colección: db.nombre_coleccion.insert([documento1, ...,documentoN])

Donde cada documento es de la forma: {campo1: valor1, campo2: valor2,..., campoN: valorN}

Hacer consultas a una colección: db.nombre_coleccion.find(condición_de_filtrado). Si no introducimos ninguna condición de filtrado obtendremos todos los documentos de la colección.

Actualizar documento en una colección:

db.nombre_coleccion.update({condicion},{comando set})

Esto último se comprende mejor con un ejemplo:

```
> db.geekFlareCollection.find()
{ "_id" : ObjectId("5edf3f67d6bfd8125f58cfb"), "product" : "bottles", "Qty" : 100 }
{ "_id" : ObjectId("5edf3f67d6bfd8125f58cfc"), "product" : "bread", "Qty" : 20 }
{ "_id" : ObjectId("5edf3f67d6bfd8125f58cfd"), "product" : "yogurt", "Qty" : 30 }
>
> db.geekFlareCollection.update({"product" : "bottles"},{$set : {"Qty": 10}} )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.geekFlareCollection.find()
{ "_id" : ObjectId("5edf3f67d6bfd8125f58cfb"), "product" : "bottles", "Qty" : 10 }
{ "_id" : ObjectId("5edf3f67d6bfd8125f58cfc"), "product" : "bread", "Qty" : 20 }
{ "_id" : ObjectId("5edf3f67d6bfd8125f58cfd"), "product" : "yogurt", "Qty" : 30 }
```

Figura 44. Modificando datos en una colección.

Vemos que se ha modificado el valor del campo “Qty” para el documento con el valor “bottles” en el campo “product”. Si hubiera más de uno se produciría en todos el cambio. Para evitar esto existe el comando `db.nombre_coleccion.updateOne({clave a actualizar},{comando set})`.

Del mismo modo podremos borrar documentos en nuestras colecciones con los siguientes comandos:

```
db.nombre_coleccion.deleteOne({condición})  
db.nombre_coleccion.deleteMany({condición})
```

En el primer caso se borra el primer documento que cumpla la condición y en el segundo todos los que la cumplan (incluso si no pusieramos ninguna, se borrarían todos los documentos).