Universidad San Carlos de Guatemala

Facultad de Ingenieria

Escuela de Ciencias y Sistemas

Redes de Computadoras 2

Ing. Pedro Pablo Hernandez Ramirez

Aux. Wilson Guerra

Manual Técnico - Práctica 4

GRUPO 13

Carnet	Nombre
201020126	Sandy Fabiola Mérida Hernández
201503600	Edgar Daniel Cil Peñate
201504231	Jose Carlos Bautista Mazariegos

:bookmark_tabs: Índice

:x: Definición del problema

La Escuela de Ciencias y Sistemas de la Universidad de San Carlos de Guatemala desea reemplazar el sistema actual de envío de reportes de actividades para los alumnos que realizan sus prácticas finales. Dicho sistema se alojará en un único servidor en la nube. Se ha identificado que la mayoría de practicantes espera a las fechas límites para realizar y enviar sus reportes, por lo que en dichas fechas se experimenta una caída en el sistema actual. Es por esto, que es de vital importancia que el nuevo sistema a desarrollar sea fácilmente escalable. Se plantea el uso de docker para contenerizar las aplicaciones o servicios que conformen el nuevo sistema. Se le solicita a usted como estudiante de la escuela, realizar un demo funcional.

:white_check_mark: Solución

Arquitectura

Para la solucion implementada se utilizo la siguiente arquitectura con docker-compose, en la cual se definen 3 redes virtuales diferentes con las siguientes direcciones de red

Red	Direccion de Red
Frontend Network	192.168.53.0/24
Backend Network	172.35.73.0/24
DB Network	10.10.13.0/24

Estas redes fueron definidas de la siguiente manera

```
networks:
 db_network:
   driver: bridge
   ipam:
     driver: default
     config:
       - subnet: 10.10.13.0/24

 service_network:
   driver: bridge
   ipam:
     driver: default
     config:
       - subnet: 172.35.73.0/24

 frontend_network:
   driver: bridge
   ipam:
     driver: default
     config:
       - subnet: 192.168.53.0/24
```

Todos los contenedores fueron creados dentro de una instancia de Amazon EC2.

Docker-compose

Como se menciono anteriormente, para la creacion e inicio de los contenedores se utilizo `Docker Compose` en el cual con la etiqueta `services:` se definen cada uno de los contenedores a utilizar.

En cada contenedor se realizo la siguiente configuracion

> Nombre contenedor

Se define el nombre que tendra el contenedor

```
container_name: <nombre_contenedor>
```

> Puerto expuesto

Se define el puerto que estara expuesto en el contenedor para acceder a los servicios utilizados

```
ports:
 - '[puerto_fisico]:<puerto_contenedor>'
```

> Variables de entorno

Se definen las variables de entorno que se utilizaran en el contenedor, para acceder a la informacion mas general de manera sencilla.

```
environment:
      <nombre_variable>: <valor_variable>
```

> Red

Para asignar un contenedor a una red creada (explicadas anteriormente) se utiliza el siguiente comando y automaticamente docker asigna una `IP` al contenedor.

```
networks:
      - <nombre_red>
```

Si se desea asignar una ip predefinida al contenedor se puede realizar con la siguiente configuracion

```
networks:
    <nombre_red>:
        ipv4_address: <direccion_ip>
```

Front End

Para el frontend se utilizo la herramienta de react, ya que es mucho más sencillo y rápido de implementar. Y con esta herramienta también se puede crear una app web mucho más liviana.

Para las peticiones se utilizo la libreria axios de Javascript

```
let response = await axiosInstance
            .get(process.env.BACK||"http://172.35.73.40:3001/")
            .then((res) => {
                data = res.data;
            return res.data;
            })
            .catch((e) => {
                console.log(e);
            });
            setData(response);
```

Para las acciones a implementar se programaron 3 vistas las cuales son:

Nuevo reporte

En esta vista se creó un formulario en el cual se debe ingresar los siguientes datos:

> - Carnet del practicante
>   - Nombre del practicante
>   - Curso asignado
>   - Cuerpo del reporte

Listado de reportes

En esta vista se realiza una peticion `GET` al servidor y obtiene como respuesta una lista con todos los reportes almacenados en la BD.

Reporte individual

Al seleccionar un item de la vista anterior y presionar su opcion para visualizar el reporte individual, se muestra un cuadro de dialogo en el que se exponen todos los datos necesarios del reporte.

Balanceador de Carga

Para el balanceador de carga se utilizo Nginx como servidor proxy inverso en el cual se realizo la siguiente configuracion para redirigir a los distintos servidores de NodeJs.

```
events {}
http {
    upstream servidores {
        server servidor1:3002 fail_timeout=10s max_fails=5;
        server servidor2:3003 fail_timeout=10s max_fails=5;
        server servidor3:3004 fail_timeout=10s max_fails=5;

    }

    server {
        listen 3001;

        location / {
            add_header 'Access-Control-Allow-Origin' '*' always;
            proxy_pass http://servidores;
        }
    }
}
```

Backend

Para el backend se utilizo nodejs junto con la libreria express para crear un `API Rest` para lo cual se tienen las siguientes linea de codigo mas importantes

> Conexion a base de datos

```
let mongoDB = "mongodb://mongo-container:27017/redes2";
mongoose.connect(mongoDB);
mongoose.Promise = global.Promise;
let db = mongoose.connection;
```

Con el modelo creado para los documentos almacenados en MongoDB, basta con las siguientes lineas para guardar, listar y obtener un reporte

> Guardar reporte

```
await newReport.save();
return res.status(200).json({nombre: newReport.nombre, id:newReport.id, date:newReport.createdAt})
```

> Listar reportes

```
const reporte = await Reporte.find();
res.status(200).json(reporte);
```

> Obtener reporte

```
const reporte = await Reporte.findById({_id:req.params.id});
reporte.servidoractual = process.env.CARNET_1;
res.status(200).json(reporte);
```

Base de datos

Para la base de datos se creo un contenedor basado en la imagen de `MongoDB`, la cual se encuentra en la pagina oficial de Docker Hub

Para este contenedor se creo un volumen para que los datos sean persistentes, es decir, que aunque se elimine o reinicie el contenedor los datos seguiran integros. Para ello se implemento la siguiente linea de codigo

```
volumes:
      - /home/dataMongo:/data/db
```