DESPLIEGUE DE APLICACIÓN ESCALABLE

Santiago Alzate[1], Sebastian Urrego[2], Juan Pablo Gómez Triana[3]

Resumen

Se busca desarrollar habilidades, así como poner en práctica, los conceptos de virtualización y contenedores para el despliegue de aplicaciones seguras que utilizan el protocolo https, a la vez que se usa una arquitectura basada en Cloud Computing. Se utilizan tecnologías como Docker y AWS para poner en práctica dichos conceptos. También se utiliza un despliegue de un CMS como lo es WordPress y de una aplicación que utiliza el stack MERN. Para el aseguramiento de las aplicaciones se utiliza la Certificate Authority (CA) Let's Encrypt. Finalmente se busca diseñar e implementar la aplicación hacia un ambiente escalable y robusto utilizando conceptos como VPC y herramientas de AWS como Load Balancers, Target Groups y Auto Scaling Groups.

Palabras Clave: Virtualización, Contenedores, HTTPS, Cloud Computing, Docker, AWS, CMS, WordPress, MERN, Certificate Authority, Let's Encrypt, Escalabilidad, VPC, Load Balancers, Target Groups, Auto Scaling Groups.

Introducción

En este documento se numeran los pasos requeridos para desplegar la aplicación escalable propuesta en el proyecto 2.

[1] Estudiante de Ingeniería de Sistemas de la Universidad EAFIT, Medellín, Antioquia. E-mail: salzatec1@eafit.edu.co
[2] Estudiante de Ingeniería de Sistemas de la Universidad EAFIT, Medellín, Antioquia. E-mail: surregog@eafit.edu.co
[3] Estudiante de Ingeniería de Sistemas de la Universidad EAFIT, Medellín, Antioquia. E-mail: jpgomezt@eafit.edu.co

VPC

En esta sección mostraremos los pasos que fueron requeridos para la elaboración y despliegue de la VPC.

Diseño

Para lograr un despliegue con alta disponibilidad, se van a implementar 2 zonas de disponibilidad (Availability Zone) al igual que servicios de Load Balancing y Auto Scaling.
Para este diseño, se consideraron 7 subredes: 2 públicas, y 5 privadas.

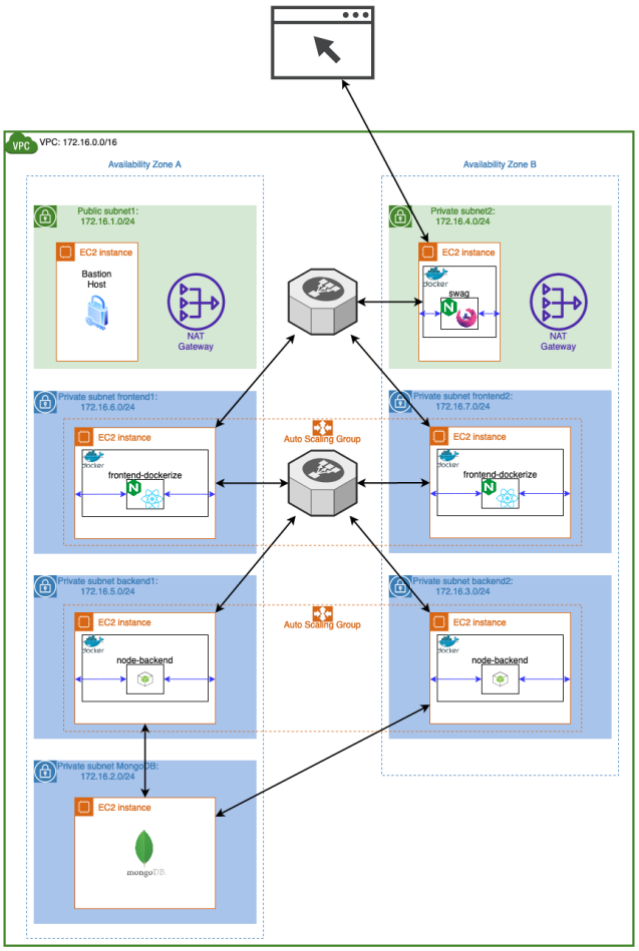

Figura 1. Diseño de la VPC.

En las subredes públicas se debe localizar todos los recursos como la instancia que soporta el Bastion Host, la instancia que soporto nuestro Proxy Server y el NAT Gateway. En la subred privada, se deben localizar, tanto la instancia que soporta el Frontend, la instancia que soporta el Backend, y finalmente la instancia que soporta la base de datos MongoDB.
Primero, se debe de crear la respectiva VPC con sus respectivas subredes.

Creación VPC

Se siguen los pasos planteados en la guía propuesta. Es importante resaltar que el *"wizard"* de AWS solo nos permite configurar 6 subredes (2 públicas y 4 privadas). Es por esto por lo que después de crear nuestra VPC, debemos de añadir la quinta subred privada que necesitamos.

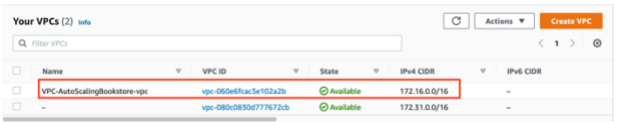

Figura 2. Creación de la VPC.

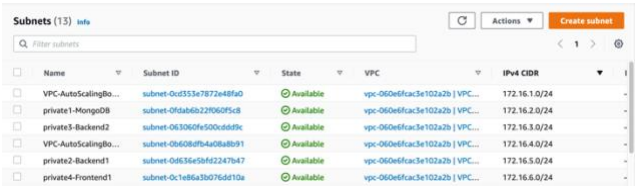

Figura 3. Subredes de la VPC.

Se puede observar en la figura 3 como se crearon las 6 subredes, y la subred faltante es una de las encargadas del frontend (frontend2). Para crearla simplemente debemos ir al menú de "Subnets", y presionar en el botón "Create Subnet". Ahora configuramos la subred con lo valores, teniendo en cuenta a cuál Availability Zone pertenecerá, y la respectiva dirección IP con su mascará de subred:

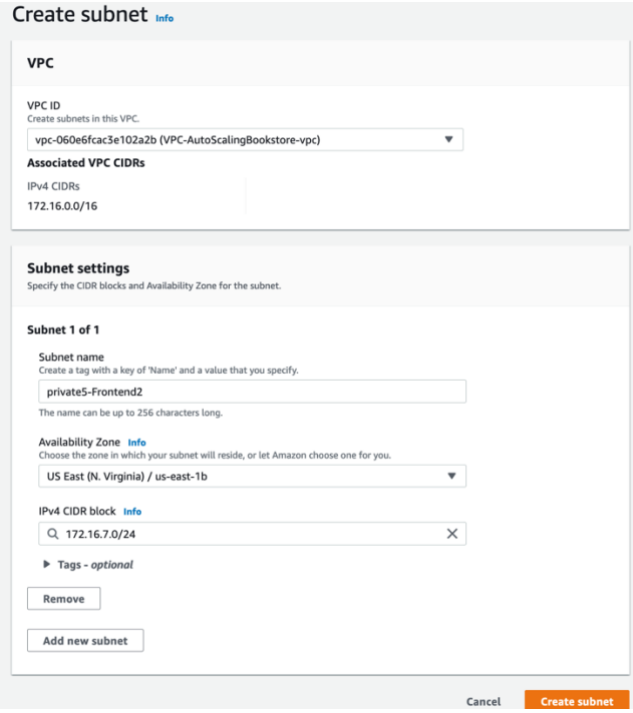

Figura 4. Creación de la subred.

Con esto ya tendríamos nuestra subred, solo faltaría asociarle una Routing Table, la cual le indique a nuestra red, cuál es su respectiva NAT Gateway, y cuál es la red local. Para esto, vamos al menú "Route Tables" y creamos una. Solo es necesario agregarle los respectivos "Destinations" y "Target":

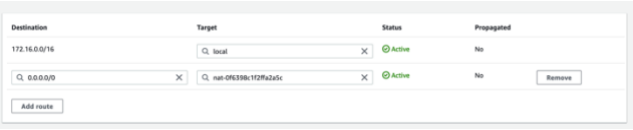

Figura 5. Creación de la subred.

Finalmente, solo sería cuestión de asignarle dicha "Routing Table" a la subred frontend2, y tendríamos nuestra VPC con sus respectivas Availability Zones y Subredes listas:

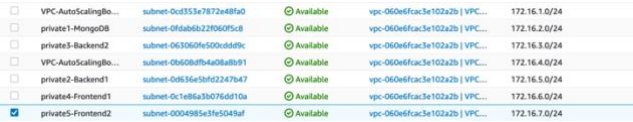

Figura 6. Visualización de la VPC.

Bastion Host

Es una instancia EC2 que se encarga de permitir el acceso seguro a las demás instancias EC2 que se encuentran localizada en las diferentes subredes privadas de la VPC, todo esto a través de una sesión ssh utilizando una llave (un .pem) como método de autenticación. Para esto debemos crear la instancia EC2.

Configuración EC2

Para iniciar se instancio una EC2 con S.O Amazon Linux. Es importante que, en las configuraciones de red, seleccionemos la VPC que creamos anteriormente, y como subred, seleccionar una publica (seleccionamos la perteneciente a la Availability Zone A). Esta instancia debe poseer el puerto 22 (puerto por defecto para sesiones ssh) abierto. Es importante tener presente que llave se utilizara para la conexión ssh (en nuestro caso, se crea una llave para este proyecto llamada AutoScalingBookstore.pem), y es importante que se utilice esta para todas las instancias EC2 que se creen dentro de la VPC.
Una vez lanzada la instancia, debemos de enviarle la llave. En este caso lo hacemos utilizando SCP:

$ scp -i AutoScalingBookstore.pem -r AutoScalingBookstore.pem ec2-user@52.90.116.57:/home/ec2-user/

Esto con el objetivo de que esta instancia actúe como punto de entrada para las demás instancias EC2. Ya solo basta agregar esa llave a nuestro agente de autenticación SSH:

$ ssh-add -k AutoScalingBookstore.pem

Ya con esto, nos podemos conectar con cualquier instancia EC2 dentro de nuestra VPC que utilice la misma llave desde nuestro Bastion Host:

$ ssh -A <IP de la instancia>

Persistencia de Datos

En esta capa es donde la información es gestionada y almacenada. Los datos se almacenan en MongoDB, un motor de bases de datos orientado a documentos. Para esto, utilizaremos una maquina EC2 Amazon Linux.

Configuración EC2

Para iniciar se instancio una EC2 con S.O Amazon Linux. Es importante que, en las configuraciones de red, seleccionemos la VPC que creamos anteriormente, y como subred, seleccionar una privada (seleccionamos la perteneciente a la Availability Zone A). Esta instancia debe poseer el puerto 27017 (puerto por defecto para MongoDB) abierto y debemos utilizar la misma llave que utilizamos en el Bastion Host. Ya con esto nos podemos conectar a nuestra instancia EC2 desde el Bastion Host:

$ ssh -A 172.16.2.230

Una vez hecho esto, ya instalamos el motor de Mongo. Para esto seguimos el tutorial dispuesto en la página oficial de MongoDB:

Primero debemos crear el archivo de configuración de los repositorios de yum, para que este pueda instalar Mongo. Este archivo será *"/etc/yum.repos.d/mongodb-org-5.0.repo"*. Una vez lo creamos, debemos de insertar la siguiente información:

[mongodb-org-5.0]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/amazon/2/mongodb-org/5.0/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-5.0.asc

Una vez configurado el repositorio, ya podemos instalar mongo.

$ sudo yum install -y mongodb-org

Una vez instalado, podemos correr la aplicación de MongoDB.

$ sudo systemctl start mongod

Ya una vez inicializada la aplicación, lo único que tenemos que hacer es crear la base de datos, con la respectiva colección y documentos.

$ sudo mongosh

Para crear la base de datos *"bookstore"*, simplemente la tenemos que instanciar.

test> use bookstore

Para que la base de datos quede creada, debemos insertar datos en ella, para ello, insertaremos en la colección *"books"* los datos de los 2 libros de Leonardo da Vinci.

*bookstore> db.books.insert(<**json libro 1**>)*

*bookstore> db.books.insert(<**json libro 2**>)*

Finalmente, creamos el usuario que utilizaremos para conectarnos a la Base de Datos. Este usuario se crea en la base de datos por defecto *"admin"*, ya que tendrá permisos de lectura y escritura en todo el Mongo:

bookstore> db.createUser({user: "jpgomezt", pwd: "password", roles:[{role: "readWriteAnyDatabase" , db:"admin"}]})

Con esto, ya tenemos la base de datos configurada. Ahora tenemos que configurar a Mongo para que permita conexiones de cualquier dirección, y que tenga el módulo de autenticación activo para asegurar las conexiones que se intentan hacer. Este archivo se encuentra en *"/etc/mongod.conf"*. Una vez entramos en esta, debemos de ir al campo *"net"* y abrir el campo *"bindIp"*. Este por defecto trae la dirección loopback. Para abrirlo a cualquiera, usamos la wildcart *"0.0.0.0"*. Para activar la verificación de usuario, descomentamos el módulo *"security"*, y le agregamos la siguiente configuración *"authorization: 'enabled'"*. El archivo debe quedar así:

Finalmente, reiniciamos la aplicación de MongoDB para que los cambios hagan efecto.

$ sudo systemctl restart mongod

Con esto, terminamos la capa de Persistencia de Datos.

Auto Scaling Group

Como pudimos ver en nuestro diagrama de despliegue, tendremos Auto Scaling Group para la capa del BackEnd y FrontEnd. Para poder crear estos grupos, necesitamos primero crear una AMI de las instancias del BackEnd y FrontEnd (de esta forma, se guardarán el contenido del boot disk y las nuevas instancias desplegadas a partir de esta, se van a instanciar con un contenido idéntico), y ya con esto podremos crear nuestros Launch templates, el Target Group, y finalmente los Auto Scaling Group.

Lógica de Negocio (BackEnd)

En esta capa se ejecuta y lleva a cabo la lógica de negocio. El backend se encuentra desarrollado empleando tecnología de scripting del lado del servidor, para este caso Node.JS. Igualmente utilizamos un framework de desarrollo de aplicaciones web denominado Express, así como Mongoose para la conexión con la base de datos. Para esto, utilizaremos una maquina EC2 Amazon Linux.

Configuración EC2

Para iniciar se instancio una EC2 con S.O Amazon Linux. Es importante que, en las configuraciones de red, seleccionemos la VPC que creamos anteriormente, y como subred, seleccionar una privada (seleccionamos la perteneciente a la Availability Zone A). Esta instancia debe poseer el puerto 5000 y 22 (puerto por donde el servidor escuchara las peticiones) abierto y debemos utilizar la misma llave que utilizamos en el Bastion Host. Ya con esto nos podemos conectar a nuestra instancia EC2 desde el Bastion Host:

$ ssh -A 172.16.3.20

Una vez hecho esto, ya instalamos Docker, para realizar el despliegue de este servidor. Para esto seguimos el tutorial dispuesto en la página oficial de Amazon para la instalación de Docker. Solo hace falta instalar el paquete Docker Engine más reciente (el repo de estas instancias ya lo tienen listo).

$ sudo amazon-linux-extras install Docker

Una vez instalado, podemos correr la aplicación de Docker:

$ sudo systemctl start docker

$ sudo systemctl enable docker

Ya debemos cambiar el URL de conexión con la base de datos, que se encuentra en el archivo *".env"* del proyecto backend entregado en el laboratorio:

URL_DB_CONNECTION = mongodb://jpgomezt:password@ 172.16.2.230/bookstore

Finalmente, debemos crear nuestro archivo Dockerfile.

FROM node:14.19.1-alpine3.14

WORKDIR /usr/src/backend

COPY . .

RUN npm install

EXPOSE 5000

CMD ["node", "server.js"]

Para este, utilizamos la imagen de Node.JS que contiene la versión 14.19 (la misma versión utilizada en el laboratorio para evitar problemas de versiones). Configuramos el directorio donde trabajamos, que será *"usr/src/backend"*. Luego copiamos todo el contenido de nuestro fichero backend dentro del espacio de trabajo en el contenedor. Instalamos los módulos de Node.JS que utiliza el proyecto corriendo *"npm install"*, exponemos el puerto 5000 (el puerto que se tiene configurado para escuchar las peticiones) y

ponemos a correr el servidor utilizando el archivo *"server.js"*.
Ya con esto, tenemos todo lo necesario para correr nuestro servidor dockerizado. Pasamos el fichero a nuestra instancia EC2 utilizando Git o SCP (secure copy).
Una vez poseemos el fichero en nuestra instancia EC2, solo debemos crear la imagen utilizando nuestro Dockerfile.

$ sudo docker build -t node-backend .

Ya con esto, tenemos lista nuestra instancia para poder crear nuestra AMI.

Creación AMI

Para crear la AMI, en el menú de las instancias seleccionamos nuestra EC2 que contiene el Backend, y seleccionamos la opción de "Create Image". Le damos el nombre de "BackEnd-AMI", y ya con esto podemos guardarla para tener la AMI del Backend:

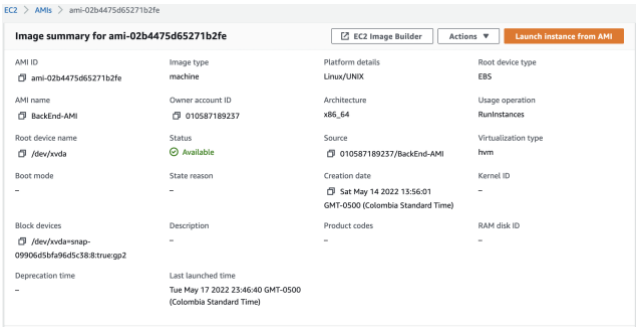

Figura 7. Visualización de la AMI del Backend.

Creación Launch Templates

Para crear el Launch templates, seleccionamos la opción de "Create Launch Template". Le damos de nombre "BackEnd-Template". Debemos de selecciona la opción de "Provide guidance to help me set up a template that I can use with EC2 Auto Scaling". Luego en "Application and OS Images", debemos selección la opción "My AMIs" y seleccionar la AMI que creamos en el Backend (BackEnd-AMI). La "Instance type" es t2.micro. La key pair podemos seleccionar la usada en el Bastion Host si queremos conectarnos en estas instancias. No llenamos el campo de "subnet" y le añadimos el mismo security group que tiene la instancia EC2 del Backend. Finalmente, el "Advanced details", debemos llenar el campo de "User data", en el cual podemos poner scripts. En este caso, pondremos un script que permite que cuando se instancien las maquinas, esta cree el contenedor con la imagen "node-backend":

#!/bin/bash
sudo docker run -d --name backend-dockerize -p 5000:5000 node-backend

Ya con esto tenemos todo lo necesario para el BackEnd.

Creación Target Group

El nombre de nuestro Target Group es "TG-BackEnd". En la opción "target type" seleccionamos "Instances", en la opción "Protocol" seleccionamos HTTP:5000 (el puerto por el que escucha el servido Backend las peticiones), en la VPC seleccionamos la VPC creada, y finalmente en "Protocol versión" seleccionamos "HTTP1".

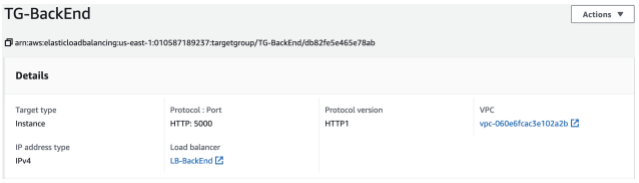

Figura 8. Visualización del Target Group.

Creación Load Balancer

El Load Balancer que crearemos es un "Application Load Balancer". El nombre de nuestro Load Balancer es "LB-BackEnd". La opción "scheme" seleccionamos "Internal". En la VPC seleccionamos la VPC creada con mapping hacia ambas Availability Zones. En el security group, creamos uno que acepte peticiones por el puerto 5000 (puerto por donde se realizaran las peticiones al Backend). Finalmente, en "Listeners and routing", seleccionamos HTTP en el puerto 5000, y que redireccione a nuestro Target Group creado "TG-BackEnd".

Figura 9. Visualización del Load Balancer.

Presentación (FrontEnd)

Esta capa se encarga de todos los aspectos relacionados con la interfaz de usuario. A esta capa se accede a través del browser. Igualmente, se encarga de la comunicación con el backend. Esta se encuentra desarrollado utilizando React. Para esto, utilizaremos una maquina EC2 Amazon Linux.

Configuración EC2

Para iniciar se instancio una EC2 con S.O Amazon Linux. Es importante que, en las configuraciones de red, seleccionemos la VPC que creamos anteriormente, y como subred, seleccionar una privada (seleccionamos la perteneciente a la Availability Zone A). Esta instancia debe poseer el puerto 80 y 22 (puerto por donde el servidor escuchara las peticiones) abierto y debemos utilizar la misma llave que utilizamos en el Bastion Host. Ya con esto nos podemos conectar a nuestra instancia EC2 desde el Bastion Host:

$ ssh -A 172.16.7.100

React con Dockerfile

Una vez instalados estos, debemos de acceder a nuestro fichero frontend dispuesto por el laboratorio, en donde crearemos otros archivos adicionales de configuración. Primero, crearemos nuestro archivo Dockerfile, el cual se encarga de correr la aplicación hecha en React y disponerla en un servidor Nginx.

```
#React build
FROM node:14.19.1-alpine3.14 as react-build

WORKDIR /usr/src/frontend

COPY . ./

RUN npm install

RUN npm run build

#Nginx setup
FROM nginx:1.12.0

RUN rm -f /etc/nginx/nginx.conf

COPY ./nginx.conf /etc/nginx/nginx.conf

COPY --from=react-build /usr/src/frontend/build /usr/share/nginx/html/bookstore
```

Vemos que el archivo cuenta con dos bloques, *"React build"*, donde creamos el build de la aplicación React, y *"Nginx setup"*, donde configuramos nuestro servidor Nginx. Para el bloque del React, utilizamos la imagen de Node.JS que contiene la versión 14.19 y le asociamos a este bloque el tag de *"react-build"*. Configuramos el directorio donde trabajamos, que será *"usr/src/frontend"*. Luego copiamos todo el contenido de nuestro fichero frontend dentro del espacio de trabajo en el contenedor. Instalamos los módulos de Node.JS que utiliza el proyecto corriendo *"npm install"*, y creamos el built corriendo *"npm run build"*. Ahora configuramos el Nginx, utilizando la imagen que contiene la versión 1.12 (la misma utilizada en el laboratorio). Primero eliminamos el archivo de configuración por defecto del Nginx y luego, lo reemplazamos por la configuración planteada en el laboratorio (se muestra más adelante). Finalmente, ubicamos el proyecto React dentro del servidor Nginx en el fichero *"bookstore"*.

En el archivo de configuración de Nginx, es necesario que el web server reciba las peticiones dirigidas al api: /api/books y las redirecciones al backend. Para esto agregamos la directiva upstream con la dirección DNS del Load Balancer del backend, antes de la de server en el archivo nginx.conf.

```
…
upstream backend {
```

```
        server        internal-LB-BackEnd-
2139315818.us-east-1.elb.amazonaws.com:5000;
    }
...
```

Ahora se configura el servidor indicándole la dirección root en donde se encuentra el build en React.

```
...
server {
        listen      80 default_server;
        listen      [::]:80 default_server;
        server_name  _;
        root        /usr/share/nginx/html/bookstore;
...
```

Finalmente, en esta sección *"server"*, debemos de indicarle al servidor la ubicación de los recursos *"/api/books"*.

```
location /api/books {
        proxy_pass http://backend;
}
```

Ya con esto, tenemos todo lo necesario para correr nuestro servidor dockerizado. Pasamos el fichero a nuestra instancia EC2 utilizando Git o SCP (secure copy).
Una vez poseemos el fichero en nuestra instancia EC2, solo debemos crear la imagen utilizando nuestro Dockerfile.

```
$ sudo docker build -t frontend .
```

Ya con esto, tenemos lista nuestra instancia para poder crear nuestra AMI.

Creación AMI

Para crear la AMI, en el menú de las instancias seleccionamos nuestra EC2 que contiene el Frontend, y seleccionamos la opción de "Create Image". Le damos el nombre de "FrontEnd-AMI", y ya con esto podemos guardarla para tener la AMI del Frontend:

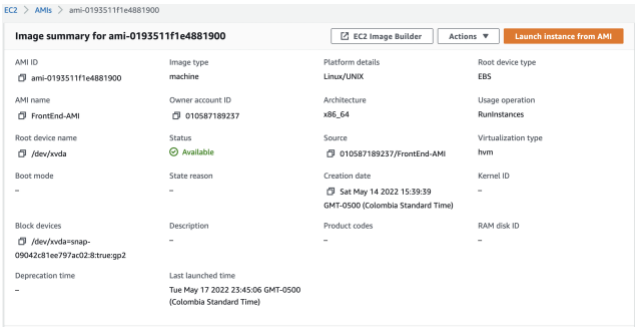

Figura 10. Visualización de la AMI del Frontend.

Creación Launch Templates

Para crear el Launch templates, seleccionamos la opción de "Create Launch Template". Le damos de nombre "FrontEnd-Template". Debemos de selecciona la opción de "Provide guidance to help me set up a template that I can use with EC2 Auto Scaling". Luego en "Application and OS Images", debemos selección la opción "My AMIs" y seleccionar la AMI que creamos en el Frontend (FrontEnd-AMI). La "Instance type" es t2.micro. La key pair podemos seleccionar la usada en el Bastion Host si queremos conectarnos en estas instancias. No llenamos el campo de "subnet" y le añadimos el mismo security group que tiene la instancia EC2 del Frontend. Finalmente, el "Advanced details", debemos llenar el campo de "User data", en el cual podemos poner scripts. En este caso, pondremos un script que permite que cuando se instancien las maquinas, esta cree el contenedor con la imagen "frontend":

```
#!/bin/bash
sudo docker run -dit --name frontend-dockerize -p
80:80 frontend
```

Ya con esto tenemos todo lo necesario para el FrontEnd.

Creación Target Group

El nombre de nuestro Target Group es "TG-FrontEnd". En la opción "target type" seleccionamos "Instances", en la opción "Protocol" seleccionamos HTTP:80 (el puerto por el que escucha el servido Frontend las peticiones), en la VPC seleccionamos la VPC creada, y finalmente en "Protocol version" seleccionamos "HTTP1".

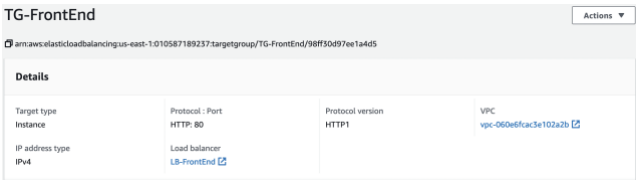

Figura 11. Visualización del Target Group.

Creación Load Balancer

El Load Balancer que crearemos es un "Application Load Balancer". El nombre de nuestro Load Balancer es "LB-FrontEnd". La opción "scheme" seleccionamos "Internet Facing". En la VPC seleccionamos la VPC creada con mapping hacia ambas Availability Zones. En el security group, creamos uno que acepte peticiones por el puerto 80 (puerto por donde se realizaran las peticiones al Frontend). Finalmente, en "Listeners and routing", seleccionamos HTTP en el puerto 80, y que redireccione a nuestro Target Group creado "TG-FrontEnd".

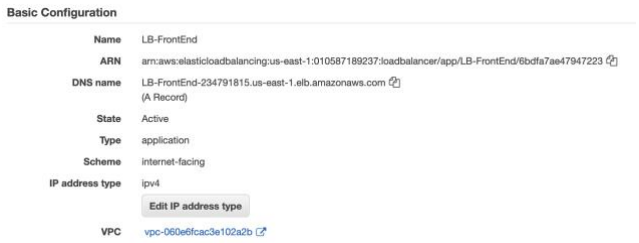

Figura 12. Visualización del Load Balancer.

Creación Auto Scaling Groups

Se crean ambos Auto Scaling Groups, teniendo en cuenta la respectiva "Launch Template", la respectiva VPC con las subredes definidas para el grupo, el respectivo Load Balancer, y pondremos una capacidad deseada, mínima y máxima en 2 (para poder apreciar la creación de más de 1 instancia):

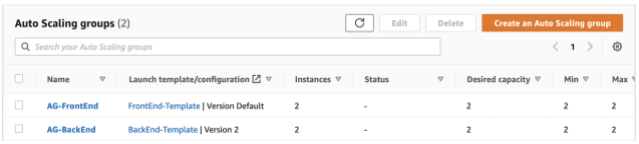

Figura 13. Visualización de los Auto Scaling Groups.

Creación Reverse Proxy

Esta capa se encarga de asegurar la conexión con los clientes que se quieran conectar con la página utilizando Let's Encrypt, y de utilizar un Reverse Proxy para redireccionar al Load Balancer del Frontend. Para esto, utilizaremos una maquina EC2 Amazon Linux.

Configuración EC2

Para iniciar se instancio una EC2 con S.O Amazon Linux. Es importante que, en las configuraciones de red, seleccionemos la VPC que creamos anteriormente, y como subred, seleccionar una publica (seleccionamos la perteneciente a la Availability Zone B). Esta instancia debe poseer el puerto 80, 443 y 22 (puerto por donde el servidor escuchara las peticiones) abierto y debemos utilizar la misma llave que utilizamos en el Bastion Host. Ya con esto nos podemos conectar a nuestra instancia EC2 desde el Bastion Host:

$ ssh -A 172.16.1.119

Dockerizando un servidor SSL con SWAG

Normalmente, tendríamos que generar un servidor SSL descargando Cerbot y configurando a Let's Encrypt como CA, pero ya que estamos utilizando Docker-Compose para configurar y levantar nuestros servicios, lo aprovecharemos para levantar nuestro servidor SSL. Para esto, utilizamos la imagen disponible en Docker Hub de SWAG (Secure Web Application Gateway). Este servicio configura un servidor web Nginx y un reverse proxy con un cliente Certbot integrado que automatiza los procesos de generación y renovación de certificados del servidor SSL utilizando a Let's Encrypt.

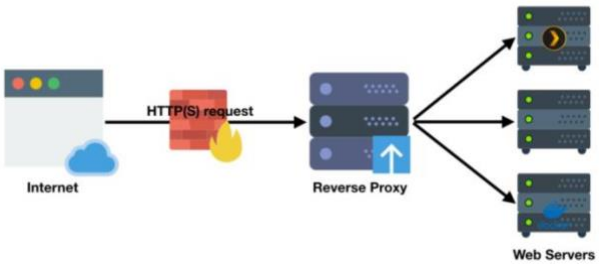

Figura 14. Funcionamiento del Reverse Proxy (tomado de LinuxServer.io).

En nuestro caso, no estaríamos tratando con una arquitectura de servidores distribuidos. Estaríamos trabajando con una arquitectura basada en

contenedores, que simularían esos servidores distribuidos. Un contenedor que haría de Reverse Proxy que redireccionara a un Load Balancer.
Lo único que debemos de hacer es agregar el servicio en nuestro archivo docker-compose.yml, el cual si utilizamos la documentación de LinuxServer.io, nos quedaría así:

```
version: '3'
services:
  swag:
    image: linuxserver/swag
    container_name: swag
    volumes:
      - ./config:/config
      - ./default:/config/nginx/site-confs/default
    environment:
      - EMAIL=jpgomezt@eafit.edu.co
      - URL=bookstore-p2.tk
      - SUBDOMAINS=www
      - VALIDATION=http
      - TZ=America/Bogota
      - PUID=500
      - PGID=500
    ports:
      - "443:443"
      - "80:80"
```

Finalmente, en el archivo default, donde se guardarán las configuraciones del reverse server de Nginx, cambiamos el proxy pass para que redireccione al Load Balancer del FrontEnd, utilizando el DNS name de este:

```
...
location @app {
        proxy_pass http://lb-frontend-234791815.us-east-1.elb.amazonaws.com;
...
```

A esta instancia se le asocia una IP elástica para configurar el Freenom con la IP de este Proxy Server:

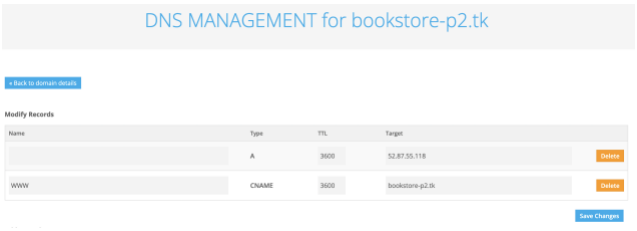

Figura 15. Configuración DNS Freenom.

Ya podemos visitar nuestra página:

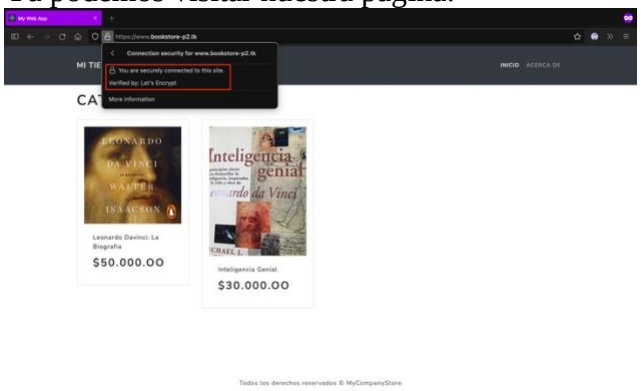

Figura 16. Página MERN Escalable desplegada.

Referencias

[1] AWS. (s. f.). Creating a container image for use on Amazon ECS - Amazon Elastic Container Service. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/create-container-image.html

[2] Docker. (s. f.-a). Install Docker Compose. Docker Documentation. https://docs.docker.com/compose/install/

[3] Docker. (s. f.-b). Install Docker Engine on Ubuntu. Docker Documentation. https://docs.docker.com/engine/install/ubuntu/

[4] Docker. (s. f.-c). Use volumes. Docker Documentation. https://docs.docker.com/storage/volumes/

[5] LinuxServer. (s. f.). SWAG setup - LinuxServer.io. LinuxServer.io. https://docs.linuxserver.io/general/swag#reverse-proxy

[6] London, I. (2016, 23 mayo). How to connect to your remote MongoDB server. Ian London's Blog. https://ianlondon.github.io/blog/mongodb-auth/

[7] MongoDB. (s. f.-a). Create A MongoDB Database. https://www.mongodb.com/basics/create-database

[8] MongoDB. (s. f.-b). Install MongoDB Community Edition on Amazon Linux — MongoDB Manual. MongoDB Documentation. https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-amazon/

[9] OpenJS Foundation. (s. f.). Dockerizing a web app. Node.Js. https://nodejs.org/en/docs/guides/nodejs-docker-webapp/

[10] Sokola, M. (2021, 17 marzo). How to Deploy a React App to Production Using Docker and NGINX with API Proxies. freeCodeCamp.Org. https://www.freecodecamp.org/news/how-to-deploy-react-apps-to-production/

[11] TIBCO. (s. f.). How to Do a Clean Restart of a Docker Instance. TIBCO Docs. https://docs.tibco.com/pub/mash-local/4.1.0/doc/html/docker/GUID-BD850566-5B79-4915-987E-430FC38DAAE4.html

[12] Willimott, C. (2021, 11 diciembre). Quickly setup WordPress & SSL via Let's Encrypt and Certbot using Docker Compose. Medium. https://carlwillimott.medium.com/quickly-setup-wordpress-ssl-via-lets-encrypt-and-certbot-b29e8abf2072

[13] Montoya, J. C. (2022, mayo). Laboratorio N5. https://interactivavirtual.eafit.edu.co/d2l/le/content/74072/viewContent/405963/View