UNIVERSIDAD
NEBRIJA

DESARROLLO DE UNA APLICACIÓN WEB Y ESTUDIO DE HERRAMIENTAS PARA EL DESPLIEGUE DE SU INFRAESTRUCTURA

UNIVERSIDAD NEBRIJA

GRADO EN INGENIERÍA INFORMÁTICA

MEMORIA PRÁCTICAS EN EMPRESA

Óscar Salvador Sotoca

Enero/2023

Tutor académico: Carlos Castellanos Manzaneque

Índice

Índice de figuras

1 Experiencia en empresa

tbc

1.1. Introduccion, mis responsabilidades

1.2. Onboarding

1.3. Organigrama

2 Proyecto

El estado del arte para el aprovisionado de infraestructura es contratarla, de manera flexible, a un proveedor cloud. En particular, aún más recientemente se ha apostado por la descripción de la infraestructura a contratar, en lenguajes declarativos y de programación general.
Este nuevo paradigma, Infraestructura como Código, ofrece replicabilidad y reutilización, pero en particular permite afrontar el mantenimiento de esta con técnicas de desarrollo convencionales como el control de versiones.

En este proyecto exploraré dos de las soluciones de Infraestructura como Código más competentes y populares: Terraform y Ansible (en particular Ansible Playbooks). Las usaré para contratar en un proveedor cloud los recursos necesarios para un sistema full stack desarrollado específicamente para este proyecto.

Después de explicar cómo funciona cada una y ponerlas en uso, compararé sus características y contrastaré sus ventajas.

2.1. Antecedentes

Este no es un proyecto original, la aplicación que quiero implementar debería ser una maqueta representativa de los elementos y comportamientos del sistema moderno medio del mercado. El valor añadido será la explicación del de esta, y la comparación entre herramientas comerciales.

2.2. Estudio del problema

He descompuesto el problema en tres fases de trabajo:

1. Programación de componentes lógicos (front-end, middleware) en local, usando Docker para prototipar e iterar rápido
2. Creación manual de los recursos y migración a la nube
3. Descripción de la infraestructura como código con ambas herramientas y comparación de ellas

2.3. Objetivos

1. Implementar un sistema full-stack representativo de la arquitectura que se podría encontrar en una aplicación comercial
    a) Servidor de estáticos, con el Front-end
    b) Servidor de contenido, con imágenes que se usen en el Front-end

   c) Middleware de una sola capa, API con GraphQL
      - Base de datos para tokens de sesión
   d) Back-end: Persistencia usando una base de datos MongoDB

2. Desarrollar el código necesario para contratar la infraestructura necesaria en Terraform y Ansible, y compararlos

3 Aplicación

Basada en los contenidos de la asignatura de desarrollo front-end, similar a las prácticas cuatro (full stack, CRUD en REST) y cinco (solo front-end, GraphQL), y un ejercicio de clase (full stack, GraphQL con tokens de verificación). todas en React.

La página permite ver "posts", compuestos por: el nombre del usuario que lo ha publicado, un comentario, y una imagen. Los comentarios pueden verse sin iniciar sesión (login). Un usuario sin identificarse puede iniciar sesión o registrarse. La segunda crea un usuario y después dispara el login automáticamente. Un usuario identificado puede: Hacer posts, lo que implica subir una imagen y un comentario; borrar los comentarios de los que sea autor; y hacer cerrar sesión (logout).

Las diferencias principales frente a las anteriores son: el uso de una base de datos Redis, para guardar los tokens solo en memoria; el almacenamiento de imágenes, usando un S3; y un reverse proxy para solucionar el problema que presenta la seguridad contra CORS.

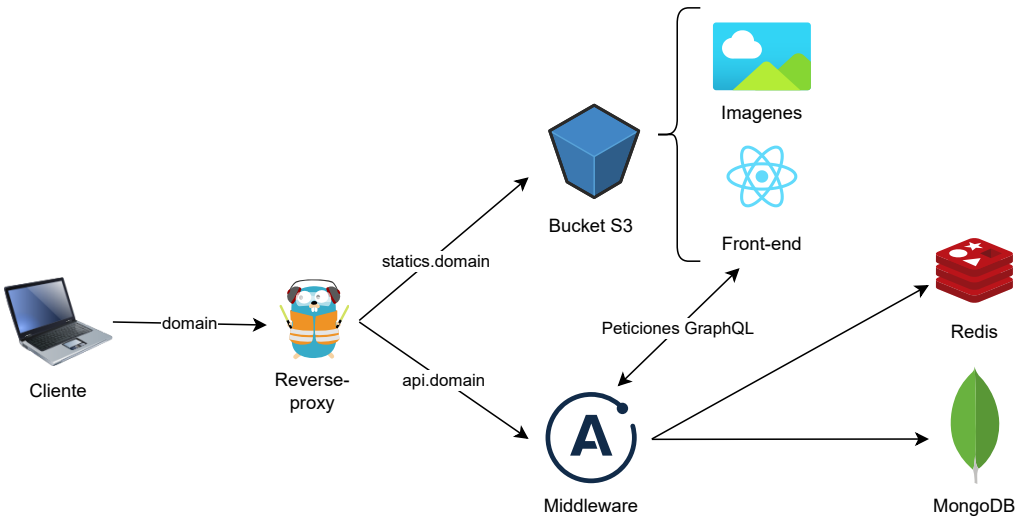

Figura 1: Diseño a alto nivel (O. Salvador 2022)

Según el proveedor cloud, algunos de los elementos se pueden contratar como Software as a Service, pagando por el uso en lugar de la maquina sobre la que correr el componente.

Hasta la entrega parcial estoy en la primera fase del problema, desarrollando en local con Docker. Todas las siguientes llamadas están implementadas, pero la interacción para subir una imagen al s3 está a medias. Para este hito, solo he conseguido subir la imagen a un s3 público. Tendré que solucionar el manejo de permisos para que, aunque la lectura sea publica, la escritura dependa de credenciales.

3.1. Inicio de sesión

El cliente front-end estará preparado en el S3. El cliente entra al dominio, el proxy dirige la petición inicial hacia los estáticos, y le sirve el JavaScript del front-end.

Desde el cliente se piden los “posts”. Estos comentarios están guardados en mongo. Cada post contiene una dirección a su imagen correspondiente. Esta esta guardada en el S3, y solo referenciada en el post. Una vez el cliente tiene la lista de posts, para cada uno, pide su archivo correspondiente. Esta operación no necesita login, es pública. El bucket tendrá permisos de lectura públicos, pero de escritura restringidos.

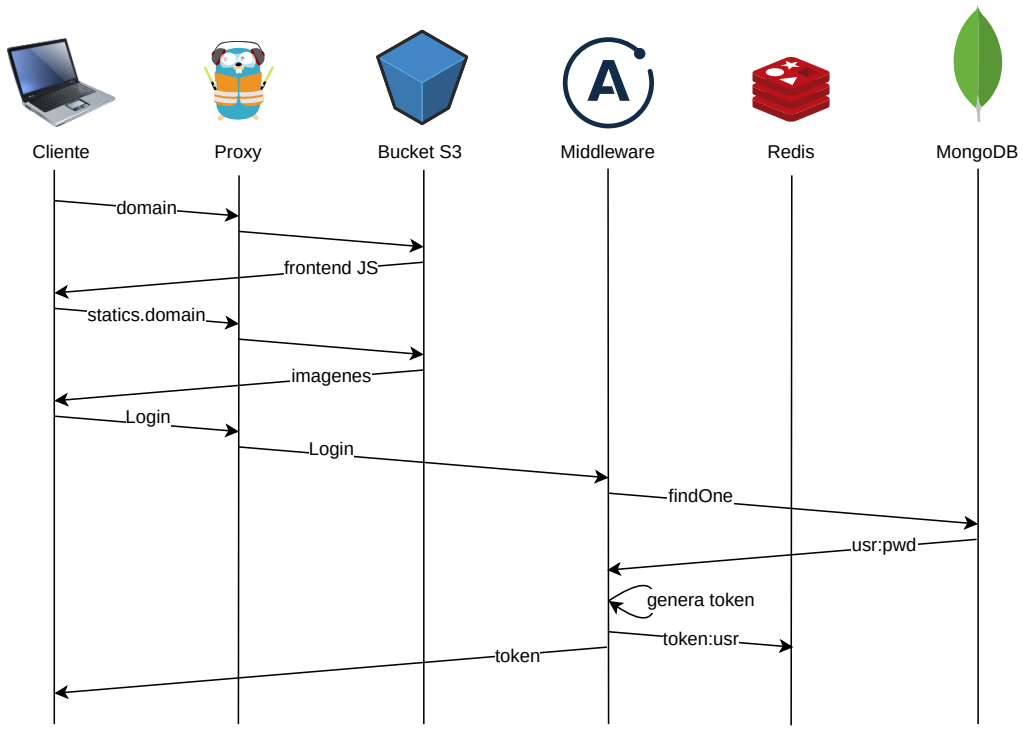

Figura 2: Paso de mensajes durante login (O. Salvador 2022)

Con el cliente renderizado, el usuario puede hacer clic en el único botón, login. Desde aquí, puede iniciar sesión si ya se tiene usuario, o crear uno nuevo. Ninguna de estas dos peticiones necesita autorización.

Crear usuario pide un nombre y contraseña. la última se tiene que escribir dos veces y debe tener caracteres variados, por seguridad. Si ya hay un usuario con ese nombre, el back-end rechazara la petición. En el caso contrario, la operación se completa y de vuelta en el cliente se dispara un login con las credenciales del usuario recién creado.

El usuario puede hacer login en el cliente, enviando su usuario y contraseña, por ahora en claro (mi objetivo es pasarlo por TLS, lo hare cuando la esté bien y tenga certificados). La petición es reconocida en el proxy y reenviada al middleware. Por la escala de la práctica y su longevidad, he hecho un middleware unificado, en lugar de separarlo en front (más expuesto, y sin estado) y back (con sesión).

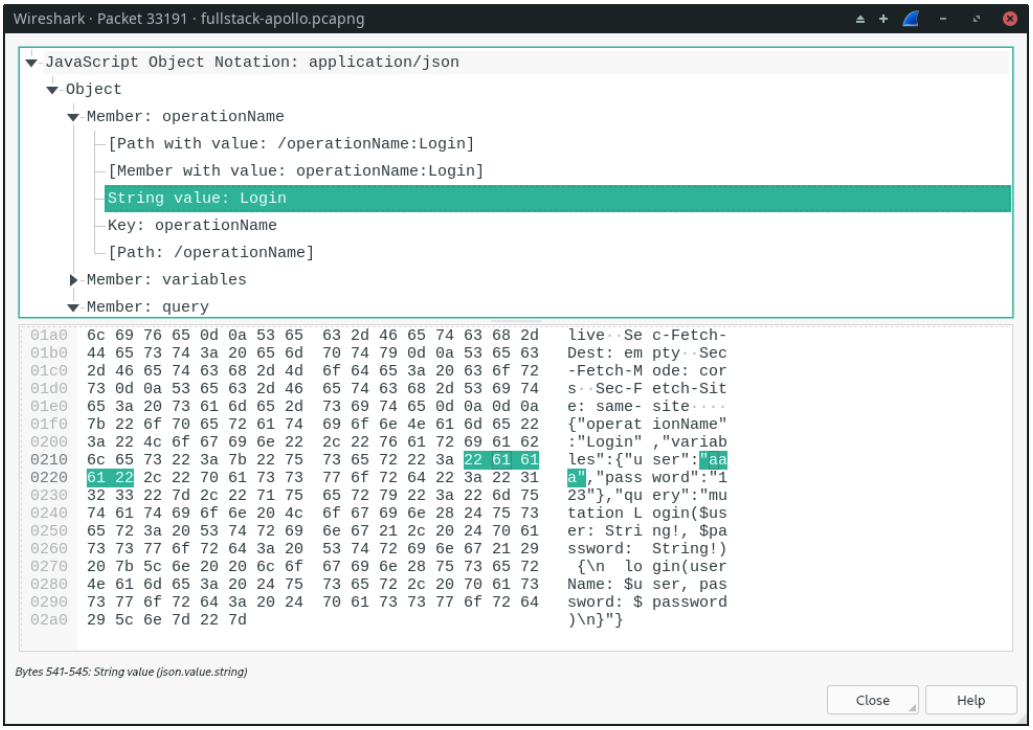

Figura 3: Inspección de los mensajes durante login usando Wireshark (O. Salvador 2022)

Desde el middleware, se hace una petición a Mongo y comprueban las credenciales en la colección “users”. Si son correctos, se genera un token, y se añade este y el nombre de usuario como entrada en Redis. finalmente, se devuelve al cliente el token del usuario, donde es guardado como una cookie el token tiene una caducidad de una hora.

Inmediatamente después de hacer login, se vuelve a preguntar al middleware los posts, enviando el token. Ahí, este es resuelto al nombre de usuario, y comparado con el de los posts, y se levanta un flag en todos los posts que sean del usuario haciendo la petición.

El resto de las peticiones al middleware requieren del cliente el uso de la cabecera .Authorization”, con un token valido, que se comprueba con cada una. La lista es recibida por el cliente y se vuelve a pintar. Los posts del usuario reciben un botón para borrarlos. Si estos son pulsados, se envía una petición con el _id en la colección de mongo, con el que se identifican en el back y borrados de la colección.

3.2. CRUD de contenido de usuarios

Partiendo de un usuario autenticado, cuando el usuario selecciona el botón “Send” en el Modal post, dispara la petición GraphQL “addPost”, que recibe el comentario escrito. El middleware, soluciona el token del usuario a su nombre. Además, con los credenciales del bucket, que nunca salen del middleware, se genera una URL firmada en la que el cliente puede subir una imagen. Esta es añadida junto a los otros detalles como una nueva entrada a la colección de Mongo. La URL es devuelta al cliente, y la imagen se sube desde este.

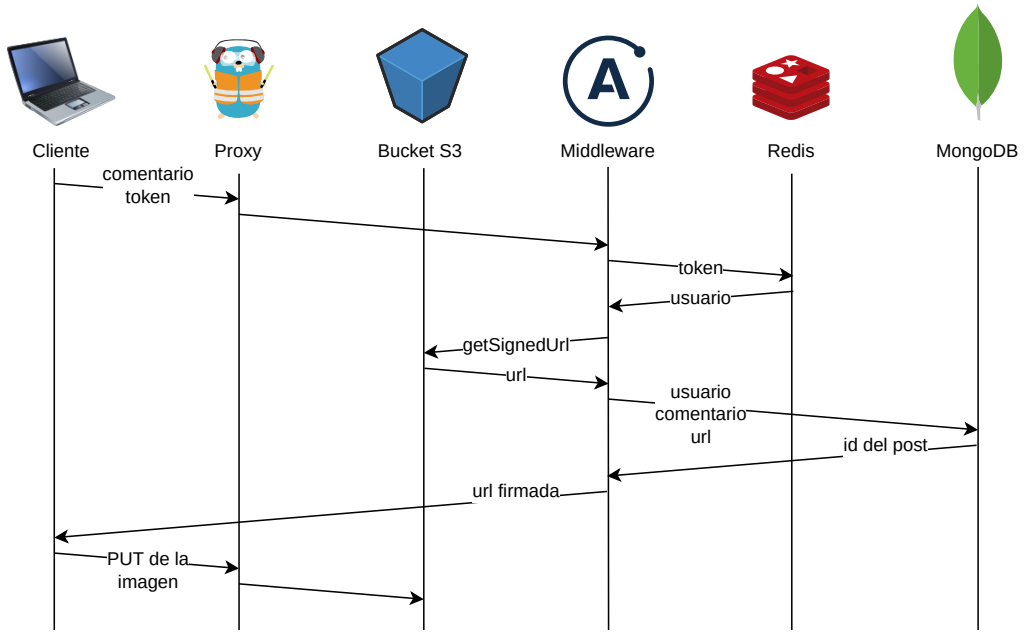

Figura 4: Paso de mensajes para añadir post (O. Salvador 2022)

Este método permite mantener los credenciales seguros, no dejándolos pasar por el cliente en ningún momento. Al mismo tiempo, permite ahorrar ancho de banda, subiendo la imagen directamente al bucket, en lugar de hacerla pasar primero al middleware.

4 Infraestructura

tbc

5 Comparación

tbc

6 Conclusión