ProyectoFinal_Redes

Proyecto II

Curso: Redes (IC-7602)

Sede: Campus Tecnológico Central Cartago

Periodo: Primer semestre 2023

Profesor: Nereo Campos Araya

Estudiantes:

- Fiorella Arias Arias 2020023639
- Jarod Cervantes Gutiérrez 2019243821
- Esteban Ignacio Durán Vargas 2020388144
- Luis Diego Alemán Zúñiga 2018135386
- Leonardo David Fariña Ozamis 20200045272

Ejecución

DNS Interceptor

DNS API

Primero es importante indicar que el DNS API se encuentra automatizado mediante el uso de Docker, para lo que se utiliza el siguiente Dockerfile:

```
FROM alpine:latest
RUN apk update
RUN apk add py-pip
RUN apk add --no-cache python3-dev
RUN pip install --upgrade pip
WORKDIR /app
COPY . /app
RUN pip --no-cache-dir install -r requirements.txt
CMD ["python3", "main.py"]
```

Donde, el Dockerfile utiliza una imagen base de Alpine Linux, instala las dependencias necesarias, copia el código fuente de la aplicación en el contenedor, instala las dependencias adicionales especificadas en el archivo requirements.txt y finalmente ejecuta el archivo main.py que contiene el servidor DNS API.

Sabiendo que el DNS API se encuentra automatizado, los pasos para ponerlo en ejecución serían:

- Ejecutar el siguiente comando para construir la imagen Docker: docker build -t dns-api .
- Ejecutar el siguiente comando para iniciar el contenedor: docker run -d -p 5000:5000 dns-api

Esto iniciará el contenedor y vinculará el puerto 5000 del contenedor al puerto 5000 del host.

Ahora, una vez explicada la forma para ejecutar el DNS API, se procede a explicar el código que se implementó:

Para implementar el DNS API, se decidió utilizar un servidor de Flask que proporciona una API para resolver consultas DNS y almacenar información relacionada con las consultas en una base de datos Firestore de Firebase.

Para configurar la aplicación Flask, se crea una instancia de la aplicación Flash y se inicializa la aplicación de Firebase utilizando un archivo de clave de servicio, tal como se muestra a continuación:

```
app = Flask(__name__)

cred = credentials.Certificate("serviceAccountKey.json")
```

Luego, se realiza la conexión con el servidor DNS externo, esto leyendo su dirección desde el archivo “remote_dns_server.txt”.

Seguidamente, se realizan varias funciones de procesamiento de consultas DNS, cada una de estas funciones corresponde con un tipo distinto de registro de los datos que hay en firebase, por ejemplo, la función “process_multi”, procesa una consulta DNS de tipo “multi”.

Finalmente, se definen las rutas que utilizará el API, siendo la principal la ruta de “/api/dns_resolver”, la cual es utilizada para resolver consultas DNS, y recibe parámetros url (URL consultada) y src (origen de la consulta) mediante una solicitud POST o GET. La función dns_resolver determina el tipo de consulta y utiliza las funciones de procesamiento correspondientes para obtener la dirección del servidor DNS adecuado. Además, se tiene la ruta de “/api/dns_request”, la cual es una ruta para enviar consultas DNS al servidor DNS externo, y recibe una consulta DNS codificada en base64 a través de una solicitud POST, la decodifica, envía la consulta al servidor DNS externo y devuelve la respuesta codificada en base64.

Es importante indicar que el código finaliza ejecutando la aplicación Flask en el host '0.0.0.0', lo que significa que estará disponible para recibir solicitudes en todas las interfaces de red.

métodos usados por el UI

Consisten en 5 métodos, que cumplen la funcion de un CRUD sobre los registros que se almacenan en Firebase

/getAll

es un método **get** que trae todos los registros existentes response:

```
{
    "documents": [
        {
            "data": {
                ...
            },
            "id": ""
        },
       ...
    ],
    "success": true | false
}
```

/get/{{id}}

es un método **get** que trae el registro al que le corresponda la llave response:

```
{
    "document": {
        ...
    },
    "success": true | false
}
```

/new

es un método **put** que crea el registro especificado request:

```
{
    "url": "",
    "document" : {
        ...
    }
}
```

response:

```
{
    "message": "",
```

```
    "success": true | false
}
```

/update

es un método **post** que actualiza el registro especificado request:

```
{
    "url" : "",
    "document": {
        ...
    }
}
```

response:

```
{
    "message": "",
    "success": true | false
}
```

/del

es un método **delete** que borrar el registro especificado request:

```
{
    "url" : ""
}
```

response:

```
{
    "message": "",
    "success": true | false
}
```

DNS 2.0 UI

Pruebas realizadas

DNS Interceptor

DNS API

Para verificar el funcionamiento del DNS API se utilizó Postman, cada prueba consistió en el llamado del método y corroborar que su repuesta fuera adecuada.

/getAll

se realiza un llamado get a **http://127.0.0.1:5000/getAll** esperando todos los registros de firebase

response:


```
{
    "documents": [
        {
            "data": {
                "index": 7,
                "servers": [
                    "128.8.8.8",
                    "129.9.9.9"
                ],
                "type": "multi"
            },
            "id": "www.itcr.ac.cr"
        },
        {
            "data": {
                "cr": "1.1.1.1",
                "jamaica": "420.420.420.420",
                "type": "geo",
                "uk": "0.0.0.0"
            },
            "id": "www.test.com"
        },
        {
            "data": {
                "servers": [
                    {
                        "server": "189.9.9.9",
                        "w": 10
                    },
                    {
                        "server": "188.9.9.9",
                        "w": 90
                    }
                ],
                "type": "weight"
            },
            "id": "www.ucr.ac.cr"
        },
        {
            "data": {
                "cr": "158.8.8.8",
```

```
                "type": "geo",
                "uk": "159.9.9.9"
            },
            "id": "www.una.ac.cr"
        },
        {
            "data": {
                "server": "149.9.9.9",
                "type": "single"
            },
            "id": "www.uned.ac.cr"
        }
    ],
    "success": true
}
```

/get/{{url}}

se realiza un llamado get a **http://127.0.0.1:5000/get/www.test.com** esperando el registro existente

response:

```
{
    "document": {
        "cr": "1.1.1.1",
        "type": "geo",
        "uk": "0.0.0.0"
    },
    "success": true
}
```

/new

se realiza un llamado put a **http://127.0.0.1:5000/new** esperando crear un registro

request:

```
{
    "url": "www.test.com",
    "document" : {
        "cr": "1.1.1.1",
        "type": "geo",
        "uk": "0.0.0.0"
    }
}
```

response:

```
{
    "message": "Document created successfully",
    "success": true
}
```

/update

se realiza un llamado post a **http://127.0.0.1:5000/update** esperando actualizar un registro

request:

```
{
    "url" : "www.test.com",
    "document": {
        "cr": "1.1.1.1",
        "type": "geo",
        "canada": "1.2.3.4"
    }
}
```

response:

```
{
    "message": "Document updated successfully",
    "success": true
}
```

/del

se realiza un llamado delete a **http://127.0.0.1:5000/del** esperando borrar un registro

request:

```
{
    "url" : "www.test.com"
}
```

response:

```
{
    "message": "Document deleted successfully",
    "success": true
}
```

DNS 2.0 UI

El DNS 2.0 UI permite las acciones de consulta, creación, edición, o borrado de los registros del dns. Fué creado sobre el framework Angular. Consta de 3 componentes, los cuales son la pantalla principal, el formulario de creación y el formulario de edición, siendo estos dos muy similares visualmente pero su lógica de funcionamiento es diferente. Además de eso esta aplicación tiene un servicio el cual es utilizado en los 3 componentes el cual se encarga de la comunicación con el backend, en este caso el DNS API, esta comunicación se realiza por medio de HTTP utilizando las operaciones POST, GET, PUT y DELETE.

Conclusiones

1. Durante el desarrollo del proyecto, se adquirieron habilidades en la implementación de servicios de capa de aplicación sobre protocolos de transporte UDP y TCP(por ejemplo, con el uso del puerto 53 para el DNS interceptor), lo que proporcionó una comprensión más profunda de estos protocolos y su funcionamiento.
2. La interacción con la documentación formal de redes fue fundamental para comprender las especificaciones y estándares relacionados con el procesamiento de las peticiones DNS, como el RFC-2929.
3. La configuración de diferentes servicios de capa de aplicación, como el DNS Interceptor y el DNS API, permitió comprender los requisitos y desafíos asociados con la configuración y la interacción entre estos componentes.
4. El uso de Firebase como base de datos para almacenar registros DNS y la integración con el DNS Interceptor y DNS 2.0 UI proporcionó una experiencia valiosa en el manejo de datos y en la implementación de diferentes tipos de registros DNS.
5. La implementación de un DNS Interceptor permitió explorar técnicas de procesamiento de paquetes DNS y realizar acciones específicas según el tipo de paquete recibido, como codificar solicitudes a BASE64 y enviarlas al DNS API.
6. La implementación de un DNS Interceptor permitió adquirir mayor conocimiento sobre el manejo y la estructura de paquetes.
7. Se ampliaron los conocimientos sobre la resolución de nombres y la comunicación con otros sistemas DNS al implementar un DNS API e integrarlo con un cliente UDP/DNS para enviar solicitudes a un servidor DNS remoto.
8. La implementación de DNS 2.0 UI utilizando React proporcionó experiencia en el desarrollo de interfaces de usuario interactivas y en la gestión de registros DNS y la base de datos IP to Country.
9. El uso de contenedores Docker para ejecutar los diferentes componentes del proyecto facilitó la implementación y el despliegue, permitiendo una mayor flexibilidad y portabilidad.
10. La creación, modificación y eliminación de registros DNS de diferentes tipos (single, multi, weight, geo) a través de DNS 2.0 UI permitió comprender en detalle la estructura y el formato de los registros DNS, así como las implicaciones de cada tipo en la resolución de nombres.

Recomendaciones

1. Se recomienda realizar un análisis amplio de los requerimientos del proyecto para así asegurarse de comprender completamente lo que se espera de cada componente, y poder organizar el proyecto de forma equitativa y eficiente con los miembros del grupo.

2. Se debe estar familiarizado con el RFC-2929 y el RFC-1035, ya que son las especificaciones relevantes para el procesamiento de peticiones DNS y la implementación de un servicio de DNS.
3. Es importante investigar sobre la comunicación entre servicios no orientados a conexión y servicios orientados a conexión, y cómo se puede transportar un servicio sobre otro, esto para tener un mejor contexto sobre lo que se realiza en el proyecto.
4. Se recomienda tener conocimiento sobre firebase, para comprender cómo utilizarlo como base de datos para almacenar registros DNS y la base de datos IP to Country, y para saber como conectarlo con Python.
5. Se recomienda comprender los diferentes tipos de registros DNS que se van a gestionar, para saber cómo administrarlos desde el DNS API y DNS 2.0 UI.
6. Es importante realizar pruebas unitarias y de integración para conocer sobre el comportamiento del sistema y ver las posibles mejoras.
7. Para futuras versiones del proyecto, se podría considerar la implementación de mecanismos de caché para mejorar el rendimiento del servicio DNS, evitando consultas repetitivas a la base de datos y al servidor DNS remoto.
8. Se recomienda para futuras versiones del proyecto, mejorar los mecanismos de manejo de errores en cada componente,ya que esto si se realiza en el proyecto actual, pero aún se podrían dar notificaciones más explícitas sobre lo que está ocurriendo.
9. Es importante mantener una comunicación constante y clara con el equipo de trabajo, además de dividir las tareas de manera equitativa y coordinando los avances para así asegurar un proceso eficiente en la implementación del proyecto.
10. Es recomendable llevar control de versiones, para mantener un registro de los cambios realizados y facilitar la colaboración entre los miembros del equipo.