ST0255 TELEMÁTICA
PROYECTO N1
Programación en Red

Fecha de publicación: marzo 9 de 2024
Última actualización: marzo 9 de 2024.
Fecha de Entrega: abril 9 de 2024.

1. Introducción

El objetivo de este proyecto es desarrollar las competencias necesarias en el diseño y desarrollo de aplicaciones concurrentes en red. Para lograr esto, se empleará la API Sockets con el fin de escribir un protocolo de comunicaciones que permita a una aplicación cliente comunicarse con una aplicación servidor.

En el contexto de las aplicaciones de red en la actualidad, MQTT (Message Queuing Telemetry Transport), es un protocolo esencial en el Internet de las Cosas (IoT) que facilita la comunicación eficiente entre dispositivos conectados.

MQTT opera mediante un modelo de publicación/suscripción, permitiendo a los clientes publicar mensajes en temas específicos o suscribirse a temas de interés. Se considera un protocolo con un diseño liviano, por tal razón, se considera ideal para dispositivos con recursos limitados. En ese sentido, MQTT destaca por su confiabilidad y escalabilidad, desempeñando un papel crucial en las diferentes aplicaciones IoT en la actualidad.

2. Antecedentes

2.1. Sockets

Tradicionalmente, a los desarrolladores de aplicaciones, las APIs de desarrollo de los diferentes lenguajes de programación, les realizan abstracciones que les ocultan los detalles de implementación de muchos aspectos, y el proceso de transmisión de datos no es la excepción.

Un socket es una abstracción a través de la cual las aplicaciones pueden enviar y recibir datos. Al respecto, el socket se puede entender como el mecanismo que utilizan las aplicaciones para “conectarse” a la red, específicamente con la arquitectura de red o “stack” de protocolos y de esta forma comunicarse con otras aplicaciones que también se encuentran “conectadas” a la red. De esta forma, la aplicación que se ejecuta en una máquina escribe los datos que dese transmitir en el socket y estos datos los puede leer otra aplicación que se ejecuta en otra máquina.

2.2. Tipos de Sockets

Existen diferentes tipos de sockets. Para este proyecto vamos a utilizar la API Sockets Berkeley, específicamente los sockets tipo “Stream (SOCK_STREAM)” o “Datagram (SOCK_DGRAM)”. En el marco de este proyecto, acorde al protocolo que está implementando usted debe decidir qué tipo de socket va a emplear y justificar a la luz de los requerimientos de su aplicación.

2.3. MQTT

Este proyecto se enfoca en la creación de un sistema de comunicación MQTT, compuesto por un cliente MQTT y un broker MQTT. El cliente MQTT será la interfaz que permitirá a los dispositivos y aplicaciones enviar y recibir mensajes en el ecosistema MQTT. Sus funciones incluirán la capacidad de publicar información en temas específicos y suscribirse a temas de interés, estableciendo así una conexión directa con el broker MQTT.

Por otro lado, el broker MQTT actuará como un servidor centralizado, facilitando la intercomunicación entre los clientes conectados. Este intermediario gestionará de manera eficiente las sesiones de los clientes, enrutará los mensajes hacia los destinatarios adecuados según las suscripciones y mantendrá la coherencia en la entrega de información. La implementación exitosa de este cliente-broker MQTT proporcionará una infraestructura robusta y escalable para la comunicación en entornos de Internet de las Cosas (IoT) y otras aplicaciones conectadas, asegurando la flexibilidad y la seguridad en el intercambio de datos.

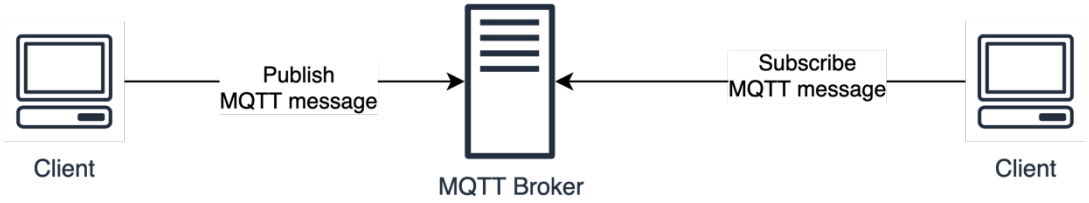

Figura 1. Conexion entre dos clientes y un broker MQTT.

De esta forma, se espera que el cliente inicie la conexión hacia el broker empleando TCP/IP.

3. Requerimientos.

Se requiere que usted implemente tanto una aplicación cliente como servidor (broker). De esta forma el cliente y el bróker se comunicarán a través de los mensajes definidos para el protocolo MQTT.

3.1. Cliente MQTT

Se requiere que usted implemente una aplicación cliente que permita realizar peticiones a un bróker MQTT. En primer lugar, la aplicación debe ser capaz de establecer y gestionar conexiones con el servidor MQTT, utilizando la interfaz TCP/IP en el puerto predeterminado 1883 (via la API sockets). Se espera que la aplicación permita la identificación del cliente mediante el envío de un identificador único, con la opción de proporcionar un nombre de usuario y contraseña durante el proceso de conexión.

Una vez la conexión se ha establecido, los clientes son capaz de intercambiar mensajes empleado tópicos. Para esto, la aplicación cliente debe facilitar la publicación de mensajes en temas (tópicos) específicos. Asimismo, se espera que admita la suscripción a temas de interés, proporcionando una forma fácil, simple, clara y accesible a través de la interfaz (consola) para gestionar las suscripciones. La gestión de sesiones persistentes es crucial para garantizar la

recuperación de mensajes perdidos después de una reconexión, y la aplicación debe enviar mensajes CONNECT con el identificador de cliente previamente utilizado durante tales eventos.

La aplicación cliente se debe ejecutar de la siguiente forma:

./mqttclient </path/log.log>

- mqttclient: es el nombre del archivo ejecutable.
- /path/log.log: representa la ruta y nombre del archivo que almacena el log.

Su aplicación cliente debe permitir registrar todas las peticiones que se generen. Para tal efecto, el archivo de log, debe permitir el registro de la siguiente información:

<date> <time> <clientIP> <query> <responseIP>

- date: indica la fecha en que se efectuó la transacción.
- time: indica la hora en que se realizó la transacción.
- clientIP: la dirección IP del cliente que realiza la petición.
- query: indica el tipo de mensaje MQTT que se está realizando.
- response: mensaje de respuesta del bróker.

3.2. Servidor - Broker MQTT

En esta sección se requiere que uste implemente un broker MQTT sin el uso de ninguna libreria para los mensajes MQTT.

En primer lugar, se hace necesario que usted aborde la implementación de la lógica fundamental del protocolo MQTT. Para efectos de esta práctica se debe considerar los aspectos relacionados con la gestión de conexiones, suscripciones (a un tópico) y publicaciones.

El aspecto central del broker implica establecer la lógica para la recepción y envío de mensajes PUBLISH, gestionando las sesiones de los clientes y manteniendo un seguimiento de los temas a los que están suscritos. De esta forma usted debe considerar una estructura de datos eficiente para almacenar información relevante sobre los clientes conectados y las suscripciones activas, permitiendo un acceso rápido y preciso durante el procesamiento de mensajes. Asimismo, se deben implementar mecanismos para manejar desconexiones temporales y garantizar la continuidad de la comunicación.

Por otro lado, la escalabilidad del broker MQTT es esencial, por lo que se hace necesario considerar estrategias eficientes para gestionar múltiples conexiones concurrentes (hilos). Además, se espera que se establezca una interfaz clara y accesible para la configuración del broker, permitiendo ajustes como la dirección IP y el puerto del servidor (consola).

Tenga presente que la seguridad no es una preocupación en este escenario específico, por tal razón ustede debe centrarse en garantizar la conformidad con el estándar MQTT y la correcta manipulación de los mensajes PUBLISH, CONNECT y DISCONNECT.

Para efectos de ejecutar su bróker MQTT, lo debe realizar de la siguiente manera:

./broker <ip> <port> </path/log.log>

- ./dnserver: es el ejecutable de la aplicación.
- ip: es la dirección ip en la cual el servidor va a ejecutar las aplicaciones.
- port: es el puerto en el cual se escucharán las peticiones por parte de los clientes. Para este caso es el puerto 1883.
- /path/log.log: representa la ruta y nombre del archivo que almacena el log.

4. Detalles de Implementación y Uso de la Aplicación.

A continuación, se detallan algunos aspectos que se deben considerar para la realización de su proyecto.

- La aplicación cliente puede ser escrita en el lenguaje de preferencia del grupo que soporte la API sockets tal como: Python 3.X, Java, C, C++, C#, Go, etc.
- La aplicación servidor **SOLO** debe ser implementada en lenguaje C.
    - **Nota**: No se recibe ningún proyecto, en su totalidad, con la aplicación servidor desarrollada en un lenguaje diferente a C.
- No se permite utilizar (tanto para cliente como para servidor) ninguna clase existente o personalizada de sockets. Se debe utilizar la API de Berkeley.
- La aplicación servidor se debe desplegar y ejecutar en un servidor en nube. Para esto utilice la cuenta de AWS academy.
- Para documentar el diseño e implementación de su solución, por favor utilice herramientas como UML o cualquier otra con el fin de ilustrar las funcionalidades, así como el funcionamiento de este.

5. Aspectos Para Considerar para el Desarrollo:

- **Equipo de trabajo:** El proyecto debe ser desarrollado en grupos de 3 personas como máximo. **NO** debe ser desarrollado de manera individual.
- **Cronograma**: Defina un plan de desarrollo, hitos, victorias tempranas, hasta la finalización del proyecto.
    - Tenga presente que tiene un mes calendario para desarrollar el proyecto. Se espera que el 60% del tiempo lo invierta en el análisis, diseño, implementación, pruebas y documentación para la aplicación servidor. El otro 40% restante, para la aplicación cliente.
- **Punto de Chequeo 1:**
    - **20 – 25 de Marzo**. Ya para esta fecha, usted debería tener la aplicación Servidor totalmente implementada.
- **Tiempo**: Debe gestionar muy bien su tiempo para poder alcanzar el objetivo final del proyecto. Empiece a trabajar apenas le publiquen su enunciado.
- **Consultas/dudas**: Si tiene alguna duda, por favor, acuda a su profesor y/o coordinador de la asignatura lo mas pronto posible.
- **Cliente o Servidor:** Se sugiere que empiece por desarrollar la aplicación servidor y luego continue con la aplicación cliente

6. Entrega

- **Repositorio:** Trabaje con git. Cree un repositorio privado para su proyecto. Recuerde que usted no puede compartir el código de su trabajo con nadie.
- **Documentación:** La documentación se debe incluir en el repo en un archivo README.md. En este archivo se requiere que usted incluya los detalles de implementación donde como mínimo se esperan las siguientes secciones:
  - Introducción.
  - Desarrollo.
  - Aspectos Logrados y No logrados.
  - Conclusiones.
  - Referencias.
- **Video**: Entregue y sustente al profesor mediante un video creado por el grupo, donde explique el proceso de diseño, desarrollo y ejecución (no más de 20 mins). Posteriormente se le citará a una sustentación presencial. Se debe ser claro en el video el funcionamiento de la solución, tanto para el cliente como para el servidor.
- **Fecha de entrega Final:** abril 9 de 2024.
- **Mecanismo de entrega:** Por el buzón de Interactiva virtual se debe realizar la entrega. La entrega debe incluir un enlace a un repositorio en github. A partir de esta fecha y hora, no se puede realizar ningún commit al repositorio.

7. Evaluación

- Se realizará un proceso de sustentación para la verificación de la práctica acorde a lo entregado por el buzón y con lo explicado inicialmente en su video.
- Posteriormente será actualizado y publicado en este documento los criterios de evaluación.

8. Referencias

- https://beej.us/guide/bgnet/
- https://beej.us/guide/bgc/
- https://www.geeksforgeeks.org/tcp-server-client-implementation-in-c/
- https://mqtt.org/mqtt-specification/