Presentación:

Instituto Tecnológico de las Américas

Grupo:

#9

Integrantes:

Julianny Tejeda 2019-7818

Laura Cabral 2019-7799

Robert Pérez 2019-8369

Francisco Lora 2019-7800

Materia/Sección:

Electiva 2: Desarrollo de IoT / G3

Día/hora:

Jueves / 2:00 PM – 5:00 PM

Maestro:

Willis Polanco

Fecha de entrega:

Viernes 23 de julio del 2021

Tema:

Tecnologías para proyecto IoT

Tecnologías para proyecto IoT

MQTT: The Standard for IoT Messaging

MQTT es un protocolo estandar de mensajeria de OASIS para el internet de las cosas (IoT). Está diseñado como un transporte de mensajería de publicación / suscripción extremadamente liviana que es ideal para conectar dispositivos remotos con una huella de código pequeña y un ancho de banda de red mínimo. Actualmente, MQTT se utiliza en una amplia variedad de industrias, como la automotriz, la fabricación, las telecomunicaciones, el petróleo y el gas, etc.

Ventajas de MQTT

- Ligero y eficiente: Los clientes MQTT son muy pequeños, requieren recursos mínimos, por lo que se pueden utilizar en microcontroladores pequeños. Los encabezados de los mensajes MQTT son pequeños para optimizar el ancho de banda de la red.
- Comunicaciones bidireccionales: MQTT permite la mensajería entre dispositivo a nube y de nube a dispositivo. Esto facilita la transmisión de mensajes a grupos de cosas.

Escala a millones de cosas: MQTT puede escalar para conectarse con millones de dispositivos IoT.

- Entrega de mensajes confiable: La confiabilidad de la entrega de mensajes es importante para muchos casos de uso de IoT. Es por eso que MQTT tiene 3 niveles de calidad de servicio definidos: 0 - como máximo una vez, 1- al menos una vez, 2 - exactamente una vez
- Soporte para redes no confiables: Muchos dispositivos de IoT se conectan a través de redes celulares poco confiables. El soporte de MQTT para sesiones persistentes reduce el tiempo para volver a conectar al cliente con el corredor.
- Seguridad habilitada: MQTT facilita el cifrado de mensajes mediante TLS y la autenticación de clientes mediante protocolos de autenticación modernos, como OAuth.

RabbitMQ

Con decenas de miles de usuarios, RabbitMQ es uno de los corredores de mensajes de código abierto más populares. Desde T-Mobile hasta Runtastic, RabbitMQ se utiliza en todo el mundo en pequeñas empresas emergentes y grandes.

RabbitMQ es liviano y fácil de implementar en las instalaciones y en la nube. Admite múltiples protocolos de mensajería. RabbitMQ se puede implementar en configuraciones distribuidas y federadas para cumplir con los requisitos de alta disponibilidad y gran escala.

RabbitMQ se ejecuta en muchos sistemas operativos y entornos de nube, y proporciona una amplia gama de herramientas de desarrollo para los idiomas más populares.

Ventajas de RabbitMQ:

- Mensajería asincrónica: Admite múltiples protocolos de mensajería, cola de mensajes, reconocimiento de entrega, enrutamiento flexible a colas, tipo de intercambio múltiple.
- Experiencia de desarrollador: Implemente con BOSH, Chef, Docker y Puppet. Desarrolle mensajería en varios idiomas con los lenguajes de programación favoritos como: Java, .NET, PHP, Python, JavaScript, Ruby, Go y muchos otros.
- Despliegue distribuido: Implementar como clústeres para alta disponibilidad y rendimiento; federar en múltiples zonas y regiones de disponibilidad.
- Preparado para la nube y la empresa: Autenticación y autorización conectables, soporta TLS y LDAP. Ligero y fácil de implementar en nubes públicas y privadas.
- Herramientas y complementos: Variedad de herramientas y complementos que admiten la integración continua, las métricas operativas y la integración con otros sistemas empresariales. Enfoque de complemento flexible para ampliar la funcionalidad de RabbitMQ.
- Gestión y seguimiento: HTTP-API, herramienta de línea de comandos y UI para administrar y monitorear RabbitMQ.

Apache Kafka

Es una plataforma de transmisión de eventos distribuida open-source con el objetivo de canalizar datos de alto rendimiento, análisis de transmisión, integración de datos y aplicaciones de misión crítica.

Componentes

- Topic: Flujo de datos sobre algún tema en particular.
- Particiones: Division de topics.
- Mensajes: Elementos que se agregan al topic.
- Brokers: Cluster identificado por ID, con servidores cuales contienen múltiples servidores, replicando y particionando los topics para la generación de tolerancia a fallos.

- Zookeeper: Gestiona los brokers bajo servicios centralizados.
- Topic Replication: Permite que los topics tengan replicación para servir los datos.
- Producers: Permite la asincronía de la publicación de mensajes de un topic, para que en caso de que el broker se caiga, el productor se recupere y siga escribiendo.
- Consumers: Se subscribe a un topic y consume sus mensajes para interpretarlos con el ecosystem.
- Stream Processors: Librería para la creación de app que consuman datos del topic. Manipulable con Java.
- Connectors: Se integra fácilmente con sistemas externos y el clúster de Kafka.

Protocolos: Se basa en TCP, cual proporciona un método rápido, escalable y duradero para los intercambios de datos utilizando arquitecturas de TI con el fin de separar procesos de los mensajes de las aplicaciones que lo generan. Los desarrolladores disponen un Clientes para lenguajes como PHP, Java, Python C/C++, Ruby, Perl o Go.

Librerías

- Producer API – Publica mensajes a los topics en el clúster Kafka.
- Consumer API – Consume los mensajes de los topics en el cluster Kafka.
- Connect API – Se conecta directamente del cluster a la fuente sin codificar, pudiendo ser un archivo, una base relacional, etc.
- Streams API – Consume mensajes de los topics y procede a transformarlos en otros topics en el cluster Kafka, pudiendo filtrarlos, integrarlos, mapearlos, agruparlos, etc.
- Admin API – Maneja e inspecciona topics y brokers en el cluster.
- Kafka-Python – Permite el envio de mensajes de manera asincrónica aumentando la velocidad de los brokers.
- Confluent-kafka – Colabora bastante con las velocidades, similar a Kafka-Python, pudiendo utilizarse sincrónicamente.
- PyKafka – Su interfaz cubre tanto la parte Producer API como Consumer API.

Redis

Remote Dictionary Server (servidor de diccionarios remoto), es un rápido almacén de datos clave-valor en memoria de código abierto que se puede utilizar como base de datos, caché, agente de mensajes y cola. incluye estructuras de datos versátiles, alta disponibilidad, datos geoespaciales, scripts Lua, transacciones, persistencia en disco y soporte de clúster, lo que simplifica la creación de aplicaciones a escala de Internet en tiempo real.

Componentes

- Cadenas: datos de texto o binarios de hasta 512 MB de tamaño
- Listas: una colección de cadenas en el orden en que se agregaron
- Conjuntos: una colección desordenada de cadenas con la capacidad para intercalarse, unirse y diferenciarse de otros tipos de conjuntos
- Conjuntos ordenados: conjuntos ordenados por un valor
- Hashes: una estructura de datos para almacenar una lista de campos y valores
- Mapas de bits: un tipo de datos que ofrece operaciones a nivel de bits
- HyperLogLogs: una estructura de datos probabilísticos para estimar los elementos únicos en un conjunto de datos

Protocolos: El protocolo de REPS (Protocolo de serialización de Redis) puede utilizarse tanto en Redis como para proyectos de software cliente-servidor, siendo un compromiso en las siguientes cosas:

- Sencillo de implementar
- Rapido de analizar
- Totalmente legibles

Puede serializar diferentes tipos de datos como enteros, cadenas y matrices, enviando solicitudes desde el cliente al servidor bajo matrices de cadenas que representan argumentos del comando a ejecutar, respondiendo al tipo de dato específico.

Este protocolo es binario seguro y no requiere el procesamiento de datos masivos transferidos de proceso a otro, por la longitud prefijada para la transferencia masiva de datos.

Librerías

- PhpRedis
- Predis

Eclipse Mosquitto

Es un agente de mensajes de código abierto (con licencia EPL / EDL) que implementa las versiones 5.0, 3.1.1 y 3.1 del protocolo MQTT. Mosquitto es liviano y adecuado para su uso en todos los dispositivos, desde computadoras de placa única de baja potencia hasta servidores completos.

El protocolo MQTT proporciona un método ligero para realizar mensajes mediante un modelo de publicación / suscripción. Esto lo hace adecuado para la mensajería de Internet de las cosas, como con sensores de baja potencia o dispositivos móviles como teléfonos, computadoras integradas o microcontroladores.

El proyecto Mosquitto también proporciona una biblioteca C para implementar clientes MQTT, y los muy populares clientes MQTT de línea de comando mosquitto_pub y mosquitto_sub.

Mosquitto es parte de la Fundación Eclipse, es un proyecto de iot.eclipse.org y está patrocinado por cedalo.com.

Descarga y seguridad: Mosquitto es altamente portátil y está disponible para una amplia gama de plataformas. Vaya a la página de descarga dedicada para encontrar la fuente o los binarios para su plataforma.

Utilice la página de seguridad para averiguar cómo informar vulnerabilidades o respuestas a problemas de seguridad pasados.

Prueba: Puede hacer que su propia instancia de Mosquitto se ejecute en minutos, pero para facilitar aún más las pruebas, el Proyecto Mosquitto ejecuta un servidor de prueba en test.mosquitto.org donde puede probar a sus clientes de varias formas: MQTT simple, MQTT sobre TLS , MQTT sobre TLS (con certificado de cliente), MQTT sobre WebSockets y MQTT sobre WebSockets con TLS.

Comunidad

- Informar errores o enviar cambios en el repositorio de Github
- Habla con otros usuarios en la lista de correo de Mosquitto o en Slack.
- Obtenga ayuda de los foros.
- Cite Mosquitto en su trabajo académico.

JMS (Java Message Service)

La API Java Message Service (en español servicio de mensajes Java), también conocida por sus siglas JMS, es la solución creada por Sun Microsystems para el uso de colas de mensajes. Este es un estándar de mensajería que permite a los componentes de aplicaciones basados en la plataforma Java2 crear, enviar, recibir y leer mensajes. También hace posible la comunicación confiable de manera asíncrona.

El servicio de mensajería instantánea también es conocido como Middleware Orientado a Mensajes (MOM por sus siglas en inglés) y es una herramienta universalmente reconocida para la construcción de aplicaciones empresariales.

Dicha API es parte integral de la versión 2 de Java.

Existen dos modelos de la API JMS, los cuales son:

- Modelo Punto a Punto (point to point) (P2P): Este modelo cuenta con solo dos clientes, uno que envía el mensaje y otro que lo recibe. Este modelo asegura la llegada del mensaje ya que si el receptor no está disponible para aceptar el mensaje o atenderlo, de cualquier forma se le envía el mensaje y este se agrega en una cola del tipo FIFO para luego ser recibido según haya entrado
- Modelo Publicador/Suscriptor (Publish/subscribe): Este modelo cuenta con varios clientes, unos que publican temas o eventos, y los que ven estos temas, a diferencia del modelo punto a punto este modelo tiende a tener más de un consumidor.

Ambos modelos pueden ser síncronos mediante el método receive y asíncronos por medio de un MessageListener.

Para enviar o recibir mensajes los clientes tienen que conectarse por medio de los objetos administrados, este nombre lo reciben porque son creados por el administrador (j2eeadmin). Estos implementan las interfaces JMS y se sitúan en el espacio de nombres JNDI (Java Naming and Directory Interface) para que así los clientes puedan solicitarlos.

Hay dos tipos de objetos administrados en JMS:

- ConnectionFactory: Se usa para crear una conexión al proveedor del sistema de mensajes.
- Destination: Son los destinos de los mensajes que se envían y el recipiente de los mensajes que se reciben.

Mensajes: Estos son la base de los JMS, se componen de tres elementos.

Header: Es la cabecera del mensaje, le sirve a los clientes y a los proveedores para poder identificarlos.

Properties: Son propiedades en general para personalizar y/o hacer más específico un mensaje.

Body: Es el mensaje en sí mismo, hay varios tipos de “cuerpos” que puede llevar un mensaje:

1. StreamMessage: Contiene un flujo (stream) de datos que se escriben y leen de manera secuencial.
2. MapMessage: Contiene pares nombre-valor.
3. TextMessage: Contiene un String.
4. ObjectMessage: Contiene un objeto que implemente la interfaz Serializable.
5. BytesMessage: Contiene un stream de bytes.

Clientes JMS: Son clientes todos aquellos que envíen mensajes, como aquellos que los reciban, en algunos textos a los que envían mensajes son llamados proveedores y los que reciben consumidores o clientes. Para enviar o recibir mensajes tienen que tener una serie de pasos :

- Conseguir un destino, mediante el objeto Destination a través de JNDI.
- Usar ConnectionFactory para conseguir un objeto Connection.
- Usar Destination para crear un objeto Session.

Ejemplos de sistemas comerciales de cola de mensajes

- WebSphere MQ de IBM (antes MQ*Series)
- Message Queue de Microsoft (MSMQ)
- Java Message Service de Sun (JMS)
- Data Distribution Service del OMG (DDS)
- Enterprise Message Service de WTF (EMS)
- Amazon Simple Queue Service de Amazon Web Services (SQS

Paho mqtt

Eclipse Paho es una implementación MQTT. Paho está disponible en varias plataformas y lenguajes de programación: Java C# Vamos C Pitón JavaScript

MQTT y MQTT-SN son transportes de mensajería de publicación / suscripción livianos para TCP / IP y protocolos sin conexión (como UDP) respectivamente. El proyecto Eclipse Paho proporciona implementaciones de código abierto, principalmente del lado del cliente, de MQTT y MQTT-SN en una variedad de lenguajes de programación.

MQTT es un protocolo de mensajería de publicación / suscripción liviana, creado originalmente por IBM y Arcom (que luego se convertiría en parte de Eurotech) alrededor de 1998. MQTT es un estándar OASIS. La última versión es la 5.0 y está disponible en una variedad de formatos.

MQTT 3.1.1 también es un estándar ISO (ISO / IEC 20922).

Paho Java Client es una biblioteca cliente MQTT escrita en Java para desarrollar aplicaciones que se ejecutan en la JVM u otras plataformas compatibles con Java, como Android.

Paho Java Client proporciona dos API: MqttAsyncClient proporciona una API totalmente asíncrona donde la finalización de las actividades se notifica a través de devoluciones de llamada registradas. MqttClient es un contenedor sincrónico alrededor de MqttAsyncClient donde las funciones aparecen sincrónicas con la aplicación.

Características

- MQTT 3.1
- MQTT 3.1.1
- MQTT 5.0
- LWT
- SSL / TLS
- Persistencia del mensaje
- Reconexión automática
- Almacenamiento en búfer sin conexión
- Soporte WebSocket
- Soporte TCP estándar
- API sin bloqueo
- API de bloqueo
- Alta disponibilidad

Descripción del Proyecto:

El proyecto Paho ha sido creado para proporcionar implementaciones confiables de código abierto de protocolos de mensajería abiertos y estándar dirigidos a aplicaciones nuevas, existentes y emergentes para Machine-to-Machine (M2M) e Internet of Things (IoT). Paho refleja las limitaciones físicas y de costes inherentes a la conectividad de los dispositivos. Sus objetivos incluyen niveles efectivos de desacoplamiento entre dispositivos y aplicaciones, diseñados para mantener abiertos los mercados y fomentar el rápido crecimiento de aplicaciones y middleware Web y Enterprise escalables.

Apache NiFi

Es un proyecto de software de Apache Software Foundation diseñado para automatizar el flujo de datos entre sistemas de software. Aprovechando el concepto de Extraer, transformar, cargar, se basa en el software " NiagaraFiles " desarrollado anteriormente por la Agencia de Seguridad Nacional de los Estados Unidos (NSA), que también es la fuente de una parte de su nombre actual: NiFi. Fue de código abierto como parte del programa de transferencia de tecnología de la NSA en 2014.

El diseño del software se basa en el modelo de programación basado en flujo y ofrece características que incluyen de manera destacada la capacidad de operar dentro de clústeres, seguridad mediante cifrado TLS, extensibilidad (los usuarios pueden escribir su propio software para ampliar sus capacidades) y características de usabilidad mejoradas como un portal. que se puede utilizar para ver y modificar el comportamiento visualmente.

Componentes: NiFi es un programa Java que se ejecuta dentro de una máquina virtual Java que se ejecuta en un servidor. [10] Los componentes destacados de Nifi son:

- Servidor web: el componente basado en HTTP que se utiliza para controlar visualmente el software y monitorear los eventos que ocurren dentro
- Controlador de flujo: sirve como el cerebro del comportamiento de NiFi. Controla la ejecución de las extensiones de Nifi y programa la asignación de recursos para que esto suceda.
- Extensiones: varios complementos que permiten a Nifi interactuar con varios tipos de sistemas
- Repositorio de FlowFile: utilizado por NiFi para mantener y rastrear el estado del FlowFile actualmente activo o la información que NiFi está ayudando a moverse entre sistemas.
- Repositorio de contenido: los datos en tránsito se mantienen aquí
- Repositorio de procedencia: aquí se conservan los datos relacionados con la procedencia de los datos que fluyen a través del sistema.

Ventajas

- Facilidad de uso mediante UI
- Escalable horizontalmente
- Gran cantidad de componentes out-of-the-box (processors y conectores)
- Es posible implementar nuevos componentes y procesadores (programando con la API de Java)
- Se encuentra en constante evolución y con una gran comunidad
- Incorpora auditoría del dato
- Tiene integrada la validación de configuraciones
- Política de Usuarios (LDAP)
- Software multiplataforma
- Linaje de datos integrada, de cara a cumplir regulaciones.
- Uso para enrutar mensajes a microservicios
- Integrado en Cloudera Data Platform (CDP) – Cloudera Flow Management (CDF)

Inconvenientes

- Las versiones por debajo de 1.x no disponen de HA
- Consumo de recursos de hardware muy elevado en función de la carga de procesamiento
- Otras herramientas como Apache Flume son más ligeras y adecuadas para realizar transformaciones de datos simples