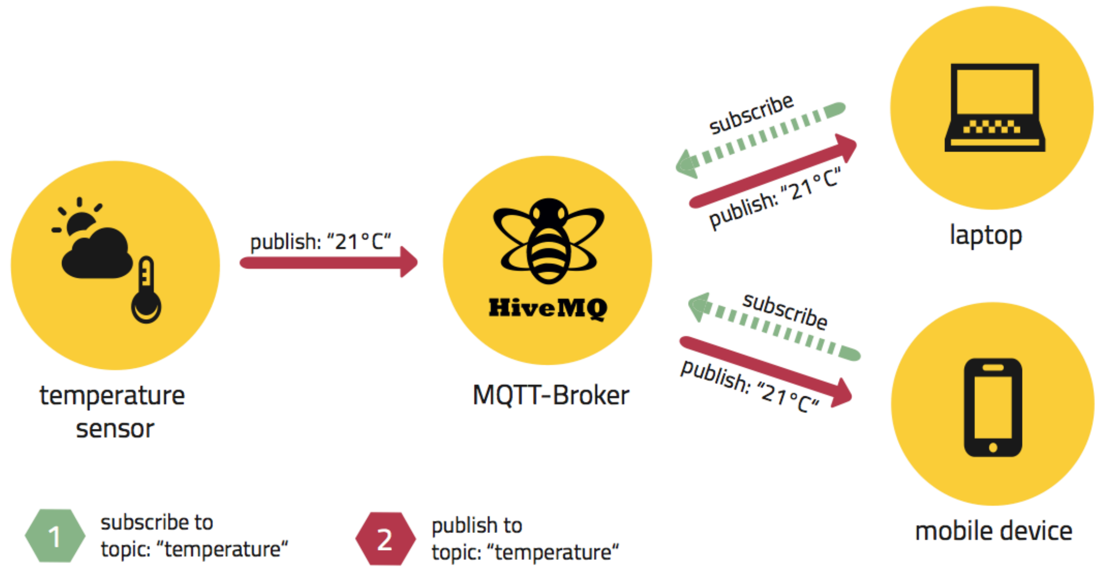
A Gentle Introduction to Eclipse Paho MQTT Client



Tosin D. Oyetoyan

Why MQTT?

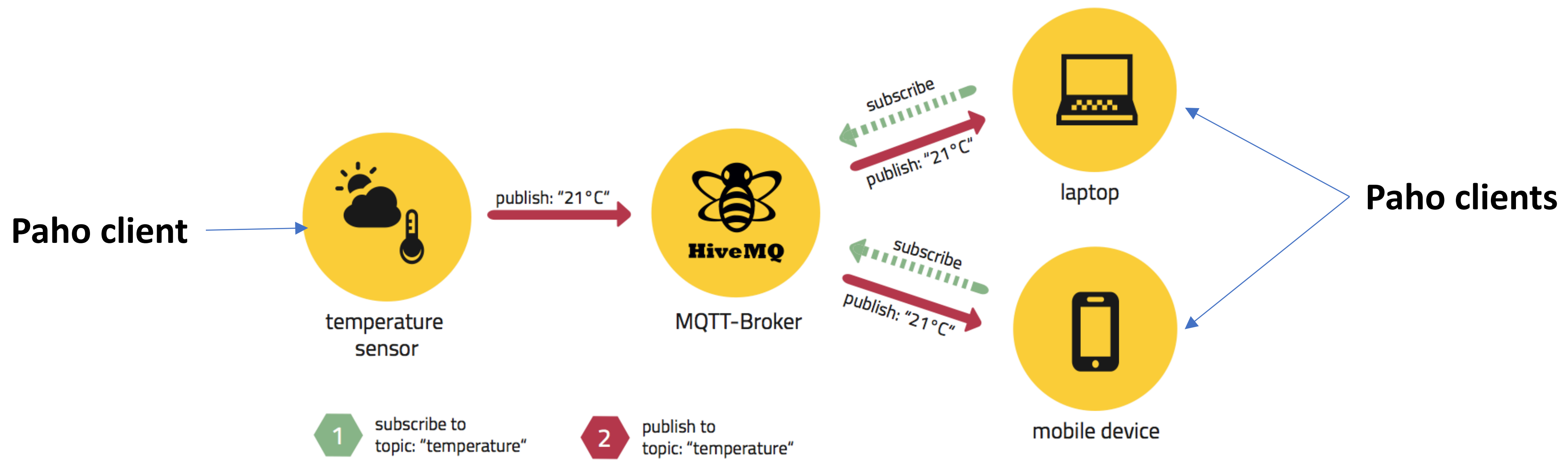
- MQTT comes from the world of M2M (Machine to Machine) and the Internet of Things.
  - devices can be as small as a sensor and controller connected over a wireless system.
  - needs for protocol implementation to be lightweight in terms of code footprint and system load, and connection problem.

Why Paho?

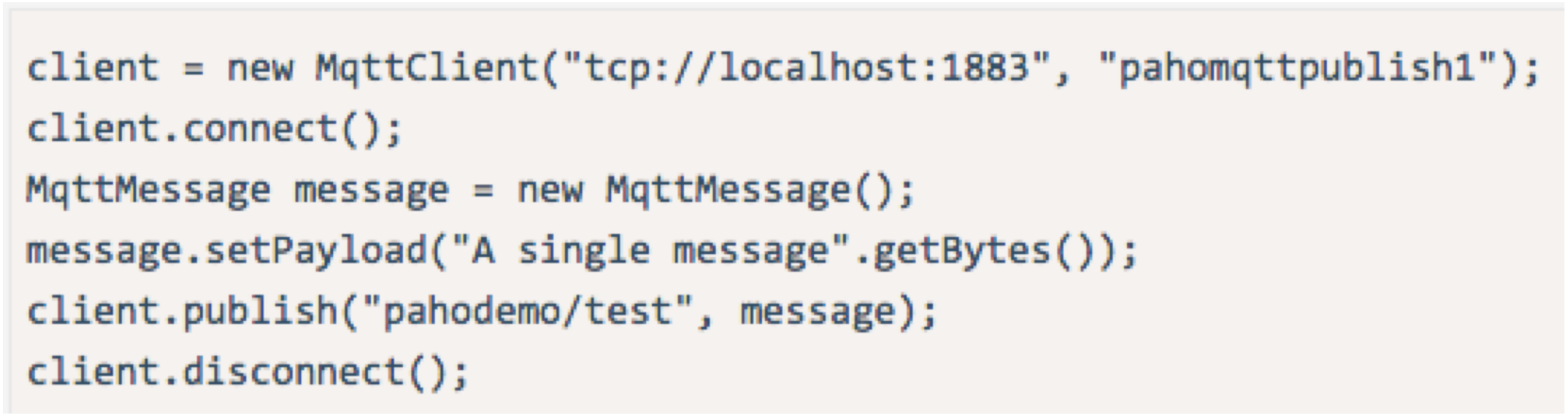
- MQTT is a protocol and protocols need client implementations.
- Eclipse Paho project
    - part of the Eclipse Foundation's M2M mission to provide high quality implementations of M2M libraries and tools.
- Libraries in
    - C, Java, C++, Python, Javascript

Paho client

- Allows a client to publish messages to a broker
- Allows a client to subscribe and receive messages on topics from a broker



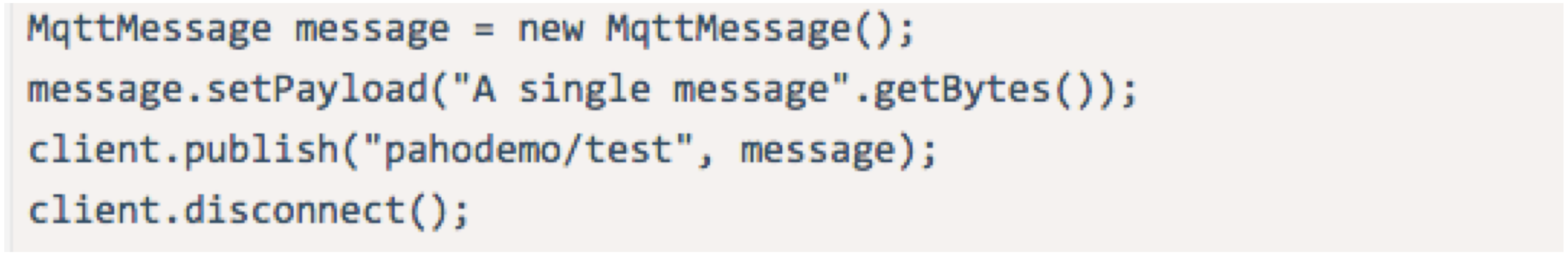
Connect



```
client = new MqttClient("tcp://localhost:1883", "pahomqttpublish1");
client.connect();
MqttMessage message = new MqttMessage();
message.setPayload("A single message".getBytes());
client.publish("pahodemo/test", message);
client.disconnect();
```

1. **Connect to a MQTT broker running on e.g. localhost**
2. **Port = 1883 (default)**
3. **Clientid = pahomqttpublish1**
    1. **A unique identifier for each client**
4. **Call connect()**

Publish



```
MqttMessage message = new MqttMessage();
message.setPayload("A single message".getBytes());
client.publish("pahodemo/test", message);
client.disconnect();
```

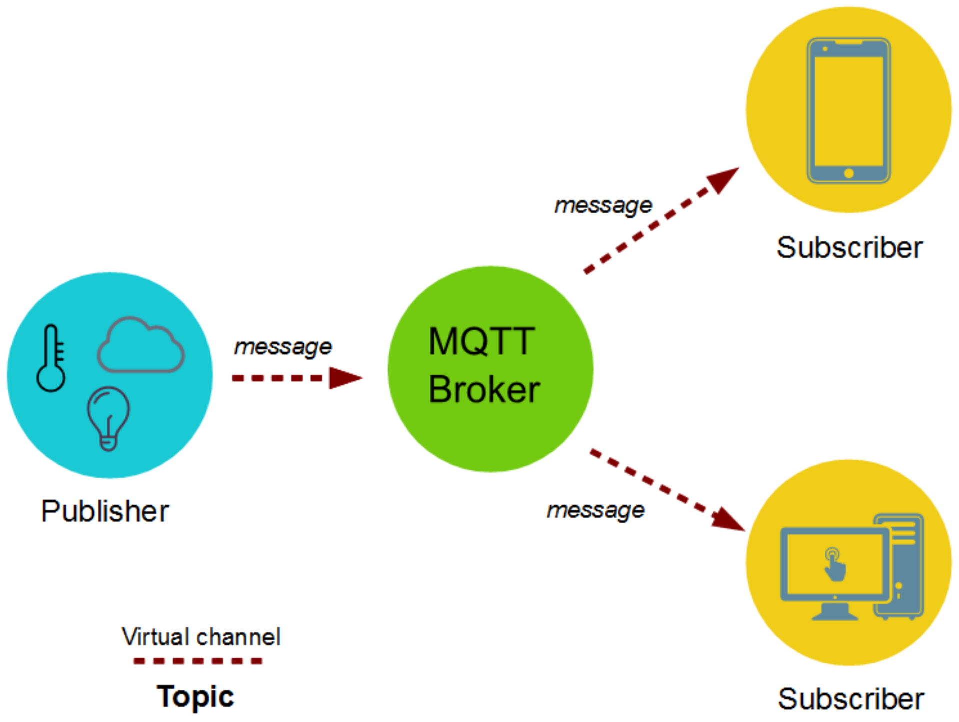
- To publish
- Create a message
  - MqttMessage
    - convert the string to bytes
- MQTT messages are published with what are called topics
  - avoid the problem of every client getting every message published by every other client
- A topic is a structured string that defines a location in a namespace
  - with "/" used to delimit levels of that namespace's hierarchy.
  - e.g. "/pumpmonitor/pumps/1/level»
- Client can only publish to absolute topic
- disconnect() is called to disconnect from the broker when MQTT session is completed

Subscribe

- Client can subscribe to a topic using wildcards to aggregate messages.
- A "+" represents one level of the implied hierarchy,
- A "#" represents all the tree from that point on.
- Example
  - "pumpmonitor/pumps/1/level" for pump 1's level
  - "pumpmonitor/pumps/+/level" for all pump levels
  - "pumpmonitor/pumps/#" for all pump activity

Broker

- A broker in MQTT handles receiving published messages and sending them on to any clients who have subscribed.
  - e.g. Mosquitto
  - CloudMQTT (probably uses mosquitto)



Message Options

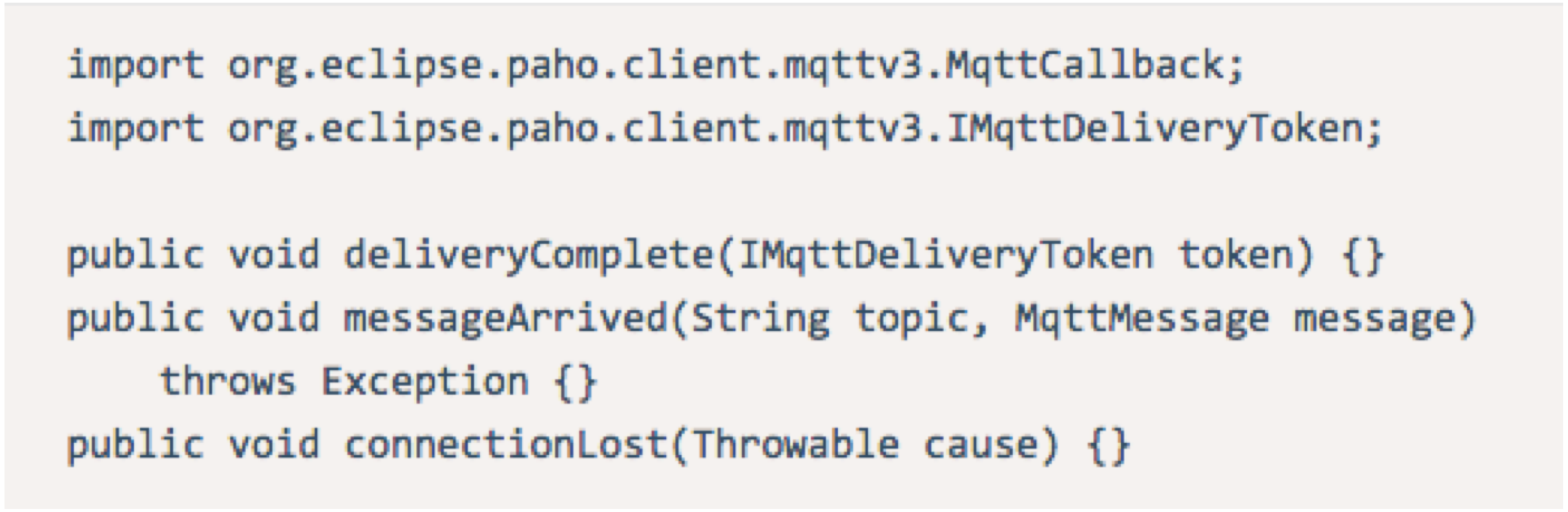
- Each message in MQTT can have its quality of service and retain flag set.
  - The quality of service advises the code if and how it should ensure the message arrives.
  - 0 - At Most Once
    - the fastest mode, where the client doesn't wait for an acknowledgement
    - if there’s a disconnection or server failure, a message may be lost
  - 1 - At Least Once
    - the sender will deliver the message at least once and, if there's no acknowledgement of it, it will keep sending it with a duplicate flag set until an acknowledgement turns up, at which point the client removes the message from its persisted set of messages.
  - 2 - Exactly Once
    - uses two pairs of exchanges, first to transfer the message and then to ensure only one copy has been received and is being processed.
    - slower
    - but most reliable QoS setting
- Retain flag for an MqttMessage.
  - If the retain flag is set to false (default), a broker will not hold onto the message so that any subscribers arriving after the message was sent will not see the message.
  - If the retain flag is set to true, the message is held onto by the broker, so when the late arrivers connect to the broker or clients create a new subscription they get all the relevant retained messages.

Connection options

- setAutomaticReconnect(boolean automaticReconnect)
  - Sets whether the client will automatically attempt to reconnect to the server if the connection is lost.
- setCleanSession(boolean cleanSession)
  - Sets whether the client and server should remember state across restarts and reconnects.
- setConnectionTimeout(int connectionTimeout)
  - Sets the connection timeout value.
- setKeepAliveInterval(int keepAliveInterval)
  - Sets the "keep alive" interval.
- setPassword(char[] password)
  - Sets the password to use for the connection.
- setSSLProperties(java.util.Properties props)
  - Sets the SSL properties for the connection.
- setUserName(java.lang.String userName)
  - Sets the user name to use for the connection.
- setWill(MqttTopic topic, byte[] payload, int qos, boolean retained)
  - Sets the "Last Will and Testament" (LWT) for the connection..

Delivery callback & Subscription

- The core of listening to MQTT activity in the Java API is the MqttCallback interface.
- allows the API to call code we have specified when
  - a message arrives (**messageArrived(String topic, MqttMessage message)**)
  - delivery of a message is completed (**deliveryComplete(IMqttDeliveryToken token)**)
  - connection is lost (**connectionLost(Throwable cause)**)



```
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;

public void deliveryComplete(IMqttDeliveryToken token) {}
public void messageArrived(String topic, MqttMessage message)
    throws Exception {}
public void connectionLost(Throwable cause) {}
```

Resources

- https://www.infoq.com/articles/practical-mqtt-with-paho
- https://www.hivemq.com/blog/mqtt-client-library-encyclopedia-eclipse-paho-java