

# Explore MQTT and the Internet of Things service on IBM Bluemix

Use a sample Java app or the Node-RED editor to learn how it works

Chun Bin Tang (<https://www.ibm.com/developerworks/community/profiles/html/profileView.do?key=67e8725a-7dd8-4119-a337-bf67a3129faa&lang=en&tabid=dwAboutMe>)

Solutions Architect, IBM Innovation Center  
IBM

18 February 2015

For interconnecting devices and applications, the Bluemix Internet of Things (IoT) service is simple but powerful, thanks to MQTT (Message Queue Telemetry Transport). In this tutorial, see how MQTT works effectively behind the IoT service, and follow an easy process to build apps using the IoT service with Java and the Node-RED editor.

The IBM Bluemix Internet of Things (IoT) service provides a simple but powerful capability to interconnect different kinds of devices and applications all over the world. What makes this possible? The secret behind the Bluemix IoT service is MQTT, the Message Queue Telemetry Transport. In this tutorial, you'll see how MQTT works and how you can easily build applications using the IoT service.

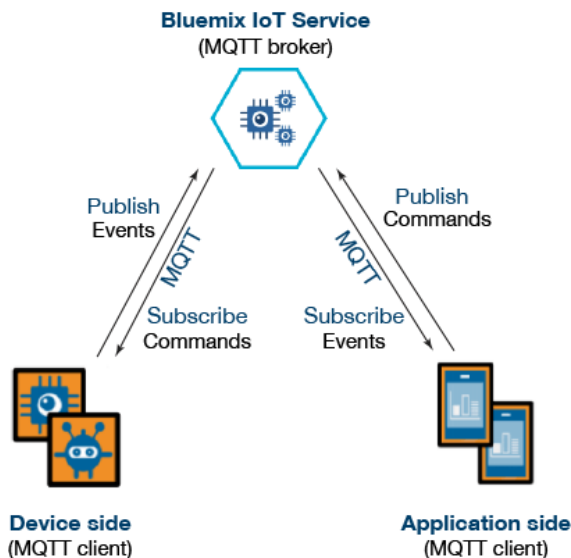
## What you'll need to build your application

- A [Bluemix](#) account
- Familiarity with the Java programming language
- Familiarity with [MQTT](#)
- Optionally, familiarity with [Node-RED](#)

[Run the app](#)  
[Get the code](#)

*“MQTT, a simple, lightweight, publish/subscribe messaging protocol on top of the TCP/IP protocol, is the ideal protocol for the emerging IoT world.”*

In general terms, the Bluemix IoT service acts as the MQTT broker, and is thus responsible for distributing messages to connected clients (devices and applications). *Devices* include machines that publish information they detect, and *applications* are the programs that consume the information received from those devices. Devices and applications communicate with the MQTT broker using the MQTT protocol, as shown:



An app using the Bluemix IoT service usually consists of three parts:

- Bluemix IoT service configuration (device and application registration)
- Device-side programming
- Application-side programming

Ready to give it a try?

## Step 1. Set up the Bluemix IoT service

In this step, you'll register devices and applications with the Bluemix IoT service.

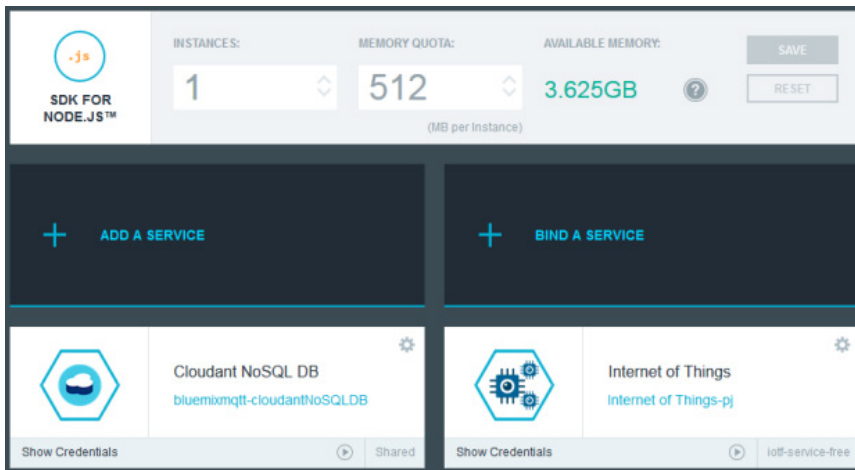
### Create an app on the dashboard

1. Log in to [Bluemix](#) using your Bluemix account.
2. Click **Catalog** in the top menu.
3. From the Boilerplates section, click **Internet of Things Foundation Starter**.
4. In the Name field, specify a unique name of your app. For this tutorial, I've used `bluemixmqtt`.
5. Click **CREATE**. Wait for your application to start.

### Add the IoT service

1. Click **ADD A SERVICE**.
2. Choose **Internet of Things service** under the Internet of Things section.
3. Click **CREATE**. Click **Restage** if prompted.

An app with IoT service is now created similar to this:

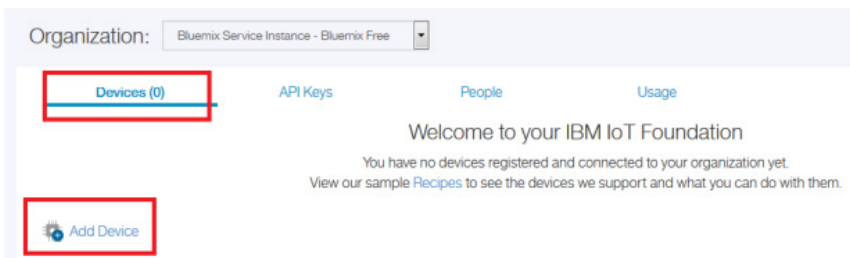


## Launch the IoT service console

1. Navigate to your application Overview page, and click **Internet of Things**.
2. Click **Launch dashboard** to open the IoT service console.

## Register the device

1. Click **Add Device** under the Devices tab.



2. Choose **Create a device type** for the Device Type option.
3. Fill in the Device Type as `MQTTDevice`. We will use that value later.
4. Fill in the Device ID with a unique id, for example, `aabbccdde12`. You may input your own identifier.
- 5.

### Register Device

To help you get the IoT Foundation connection software onto your device, visit our [Recipes](#).  
Let us know your device type and device ID (for example, the MAC address), so the device can be associated with a selected organization.

Device Type:

Organization: Bluemix Service Instance

Device ID:

Click **Continue**.



## 6. Copy the information for the device; we will use it later.

### Connect Your Device

Your devices must be registered with a unique credential to ensure that only people in your organization can see the data.

Your MQTTDevice has been added, but it will not send data to your organization until you add this credential.

- 1 Take note of or copy the following information for device ID  
aabbccdde12

```
org=vqxfzy
type=MQTTDevice
id=aabbccdde12
auth-method=token
auth-token=LZdt7D4yA4plm11A
```

*Authentication tokens are non-recoverable. If you replace this token, you will need to re-register the device to generate a new authentication token.*

- 2 Copy the unique credentials into the device configuration file on your device. [Find out how](#)

Done

## 7. Click **Done**.

Repeat the steps above to register more devices, if you like.

## Register the application

### 1. Under the API Keys tab, click **New API Key**.

Organization: Bluemix Service Instance - Bluemix Free

Devices (1) **API Keys (1)** People Usage

API keys allow you to share your organization's devices with people using the data in external application development

**Add API Key**

Key	Comment
a-vqxfzy-qfvsbak878	Bound to Bluemix Application

2. Copy the information in the New API Key dialog; we will use it later.

### New API Key

Here is your two-part, randomly generated API key. We strongly advise that you use this API key for only one application. If this API key is revoked, all applications using it will be disconnected from Internet of Things Foundation.

Key:	a-vgxfty-bnef55216w
Auth Token:	6W(@abX1EC?1Jskg(N

⚠ This is the only time that the Auth Token will be visible to you. Make sure you note it down now!

OK, I've got it!

3. Click **OK**.

Repeat the steps above to register more applications, if you like.

## Step 2. Create a device-side program

Device-side programming consists of three parts:

- Connecting to the IoT service (MQTT broker)
- Publishing events to applications
- Subscribing commands from applications

In this section, you'll build a simple device-side program in Java with [Eclipse Paho Java](#). For more information, see the [source code](#).

### Connect to the IoT service

The access point for the IoT service (MQTT broker) is `<orgid>.messaging.internetofthings.ibmcloud.com`, where `<orgid>` is created automatically when the IoT service is configured. You may use either TCP or TLS (Transport Layer Security).

```
tcp://<org-id>.messaging.internetofthings.ibmcloud.com:1883
ssl://<org-id>.messaging.internetofthings.ibmcloud.com:8883
```

Before connecting to the IoT service as a device client, you must set the MQTT connection options.

- The client-id property should be in the form `d:<orgid>:<type-id>:<device-id>`. `<orgid>` is the same as above, while `<type-id>` and `<device-id>` were input previously when the devices were registered.

- The authmethod property should be set to **use-token-auth**.
- The authtoken property should be set to the auth-token field from the device information that was copied earlier.

The following code snippet is from class `com.ibm.bluemixmqtt.MqttHandler.java`. You may choose to use TCP or TLS. If you are using TLS, you should set property `com.ibm.ssl.protocol` to **TLSv1.2**; otherwise, the connection may fail.

```
public void connect(String serverHost, String clientId, String authmethod,
    String authtoken, boolean isSSL) {
    // check if client is already connected
    if (!isMqttConnected()) {
        String connectionUri = null;

        //tcp://<org-id>.messaging.internetofthings.ibmcloud.com:1883
        //ssl://<org-id>.messaging.internetofthings.ibmcloud.com:8883
        if (isSSL) {
            connectionUri = "ssl://" + serverHost + ":" + DEFAULT_SSL_PORT;
        } else {
            connectionUri = "tcp://" + serverHost + ":" + DEFAULT_TCP_PORT;
        }

        if (client != null) {
            try {
                client.disconnect();
            } catch (MqttException e) {
                e.printStackTrace();
            }
            client = null;
        }

        try {
            client = new MqttClient(connectionUri, clientId);
        } catch (MqttException e) {
            e.printStackTrace();
        }

        client.setCallback(this);

        // create MqttConnectOptions and set the clean session flag
        MqttConnectOptions options = new MqttConnectOptions();
        options.setCleanSession(true);

        options.setUserName(authmethod);
        options.setPassword(authtoken.toCharArray());

        //If SSL is used, do not forget to use TLSv1.2
        if (isSSL) {
            java.util.Properties sslClientProps = new java.util.Properties();
            sslClientProps.setProperty("com.ibm.ssl.protocol", "TLSv1.2");
            options.setSSLProperties(sslClientProps);
        }

        try {
            // connect
            client.connect(options);
            System.out.println("Connected to " + connectionUri);
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }
}
```

The following code snippet is from class `com.ibm.bluemixmqtt.DeviceTest.java`. This code reads from the configuration file and sets the MQTT connection options with the values from the configuration file in the correct formats.

```
//Read properties from the conf file
Properties props = MqttUtil.readProperties("device.conf");

String org = props.getProperty("org");
String id = props.getProperty("deviceid");
String authmethod = "use-token-auth";
String authtoken = props.getProperty("token");
//isSSL property
String sslStr = props.getProperty("isSSL");
boolean isSSL = false;
if (sslStr.equals("T")) {
    isSSL = true;
}

System.out.println("org: " + org);
System.out.println("id: " + id);
System.out.println("authmethod: " + authmethod);
System.out.println("authtoken: " + authtoken);
System.out.println("isSSL: " + isSSL);

String serverHost = org + MqttUtil.SERVER_SUFFIX;

//Format: d:<orgid>:<type-id>:<device-id>
String clientId = "d:" + org + ":" + MqttUtil.DEFAULT_DEVICE_TYPE + ":"
    + id;
handler = new DeviceMqttHandler();
handler.connect(serverHost, clientId, authmethod, authtoken, isSSL);
```

**Note:** To connect to the Bluemix IoT service, you must use MQTT at the Version 3.1 level at a minimum; MQTT 3.1.1 is recommended for richer functionality.

## Publish events

Event topics should be published in this form: `iot-2/evt/<event-id>/fmt/<format>`. The `<event-id>` is set to categorize different event types; you may choose your own value. The `<format>` is set to **json**. The message should be encoded in **JSON**, and it must contain a single top-level property called "d".

The following code snippet is from class `com.ibm.bluemixmqtt.DeviceTest.java`. It encodes the message into JSON format and publishes it to the correct topic.

```

        //Format the Json String
JSONObject contObj = new JSONObject();
JSONObject jsonObj = new JSONObject();
try {
    contObj.put("count", count);
    contObj.put("time", new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
        .format(new Date()));
    jsonObj.put("d", contObj);
} catch (JSONException e1) {
    e1.printStackTrace();
}

System.out.println("Send count as " + count);

//Publish device events to the app
//iot-2/evt/<event-id>/fmt/<format>
handler.publish("iot-2/evt/" + MqttUtil.DEFAULT_EVENT_ID
    + "/fmt/json", jsonObj.toString(), false, 0);

```

In our sample device program, we publish events with the count value incrementing from 0 every 15 seconds. Below is a sample message. Note that top-level property "d" is required in the event JSON message.

```

{
  "d": {
    "count": 3,
    "time": "2014-12-30 16:14:59"
  }
}

```

## Subscribe commands

The topic of subscribing commands should be in the form: `iot-2/cmd/<cmd-type>/fmt/<format-id>`. The `<cmd-type>` is set to categorize different command types; you may choose your own value. `<cmd-type>` can be set using the plus symbol ("+") as a wildcard, so that it can subscribe various types of commands. The `<format-id>` is set to **json**.

The following code snippet is from class `com.ibm.bluemixmqtt.DeviceTest.java`. It demonstrates how to subscribe the command message.

```

        //Subscribe the Command events
//iot-2/cmd/<cmd-type>/fmt/<format-id>
handler.subscribe("iot-2/cmd/" + MqttUtil.DEFAULT_CMD_ID + "/fmt/json",
    0);

```

Once a command event is received, the callback function `messageArrived` is executed. The code snippet below is from `com.ibm.bluemixmqtt.DeviceTest.java`. In our sample application, when the command is received from the application side, the count is reset to the value extracted from the command.



```

        public void messageArrived(String topic, MqttMessage mqttMessage)
            throws Exception {
            super.messageArrived(topic, mqttMessage);

            //Check whether the event is a command event from app
            if (topic.equals("iot-2/cmd/" + MqttUtil.DEFAULT_CMD_ID
                + "/fmt/json")) {
                String payload = new String(mqttMessage.getPayload());
                JSONObject jsonObject = new JSONObject(payload);
                String cmd = jsonObject.getString("cmd");
                //Reset the count
                if (cmd != null && cmd.equals("reset")) {
                    int resetcount = jsonObject.getInt("count");
                    count = resetcount;
                    System.out.println("Count is reset to " + resetcount);
                }
            }
        }
    }
}

```

**Note:** For more information about device-side programming, see <https://developer.ibm.com/iot/recipes/improvise-connect-quickstart/>.

## Step 3. Create an application-side program

As with device-side programming, application-side programming consists of three parts:

- Connecting to the IoT service (MQTT broker)
- Subscribing events from devices or from the MQTT broker
- Publishing commands to devices

In this section, you'll build a simple application-side program in Java with [Eclipse Paho Java](#). For more information, see the [source code](#).

### Connect to the IoT service

The access point for IoT service is the same as that of the device side, while there are some differences between the MQTT connection option formats:

- The client-id property should be in the form a:<orgid>:<app-id>. <orgid> is the same as in the device side, and <app-id> is the unique id set for your application; you may choose your own.
- The authmethod and the authtoken property should be set to relevant fields (Key and Auth Token) from the application information copied earlier.

The following code snippet is from class `com.ibm.bluemixmqtt.APPTest.java`. It reads from the configuration file and sets the MQTT connection properties with the values from the configuration file in the correct formats.

```

        //Read properties from the conf file
        Properties props = MqttUtil.readProperties("app.conf");

        String org = props.getProperty("org");
        String id = props.getProperty("appid");
        String authmethod = props.getProperty("key");
        String authtoken = props.getProperty("token");
    }
}

```

```
//isSSL property
String sslStr = props.getProperty("isSSL");
boolean isSSL = false;
if (sslStr.equals("T")) {
    isSSL = true;
}

System.out.println("org: " + org);
System.out.println("id: " + id);
System.out.println("authmethod: " + authmethod);
System.out.println("authtoken" + authtoken);
System.out.println("isSSL: " + isSSL);

//Format: a:<orgid>:<app-id>
String clientId = "a:" + org + ":" + id;
String serverHost = org + MqttUtil.SERVER_SUFFIX;

handler = new AppMqttHandler();
handler.connect(serverHost, clientId, authmethod, authtoken, isSSL);
```

## Subscribe events

The application side can subscribe two types of events: (1) status messages from the device side and (2) system-generated connection monitor messages.

- The topic for subscribing device-side events should be in the form: `iot-2/type/<type-id>/id/<device-id>/evt/<event-id>/fmt/<format-id>`. It should be consistent with the event form on the device side. A plus symbol ("+") can be used as a wildcard, which would match exactly one level in the topic tree.
- The topic for subscribing system events should be in this form: `iot-2/type/<type-id>/id/<device-id>/mon`. In this case, too, a plus symbol ("+") can be used as a wildcard. By subscribing system events, you can receive the messages every time a device connects or disconnects from the IoT service.

The following code snippet is from class `com.ibm.bluemixmqtt.APPTest.java`. Both the system and device events are subscribed.

```
handler.subscribe("iot-2/type/" + MqttUtil.DEFAULT_DEVICE_TYPE
    + "/id+/mon", 0);

//Subscribe Device Events
//iot-2/type/<type-id>/id/<device-id>/evt/<event-id>/fmt/<format-id>
handler.subscribe("iot-2/type/" + MqttUtil.DEFAULT_DEVICE_TYPE
    + "/id+/evt/" + MqttUtil.DEFAULT_EVENT_ID + "/fmt/json", 0);
```

Once an event is received, the callback function `messageArrived` is executed. In our sample application, we check the count value in the message. If the count value reaches 4, we start a new thread to publish a command to the corresponding device to reset the count value to 0. For more information, see the class `com.ibm.bluemixmqtt.APPTest.java` file.

```
public void messageArrived(String topic, MqttMessage mqttMessage)
    throws Exception {
    super.messageArrived(topic, mqttMessage);

    Matcher matcher = pattern.matcher(topic);
    if (matcher.matches()) {
```

```
String deviceid = matcher.group(1);
String payload = new String(mqttMessage.getPayload());

//Parse the payload in Json Format
JSONObject jsonObject = new JSONObject(payload);
JSONObject contObj = jsonObject.getJSONObject("d");
int count = contObj.getInt("count");
System.out.println("Receive count " + count + " from device "
    + deviceid);

//If count >=4, start a new thread to reset the count
if (count >= 4) {
    new ResetCountThread(deviceid, 0).start();
}
}
}
```

## Publish commands

The topic for publishing commands should be in this form: `iot-2/cmd/<cmd-type>/fmt/<format-id>`. It should be consistent with the command form on the device side.

The following code snippet is from class `com.ibm.bluemixmqtt.APPTest.java`. It publishes the reset command to the corresponding device once the count value reaches 4. Of course, the command is encoded into JSON format as well.

```
                JSONObject jsonObj = new JSONObject();
try {
    jsonObj.put("cmd", "reset");
    jsonObj.put("count", count);
    jsonObj.put("time", new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
        .format(new Date()));
} catch (JSONException e) {
    e.printStackTrace();
}
System.out.println("Reset count for device " + deviceid);

//Publish command to one specific device
//iot-2/type/<type-id>/id/<device-id>/cmd/<cmd-id>/fmt/<format-id>
handler.publish("iot-2/type/" + MqttUtil.DEFAULT_DEVICE_TYPE
    + "/id/" + deviceid + "/cmd/" + MqttUtil.DEFAULT_CMD_ID
    + "/fmt/json", jsonObj.toString(), false, 0);
```

**Note:** For more information about application-side programming, see <https://developer.ibm.com/iot/recipes/improvise-application-development/>.

## Step 4. Run the app

### Build the project

1. Fork the code from the [source code](#).
2. Build the project using Eclipse locally and export it as a jar archive, for example, "bluemixmqtt.jar". You can find a compiled example and other referenced libraries under the MyData folder, in case you do not want to compile one yourself.

**Note:** The project requires Java SDK 7.0 or later.

## Update the application-side configuration file

The application-side configuration file is at MyData/app.conf. Update it according to the properties you copied while registering the application previously.

```
#Configuration files for App Side Applications

#The org field is the same org field as the Device side
org=<org>

# A unique id you choose it by yourself, maybe, abcdefg123456
appid=<appid>

# The key field from App Keys info you copied previously
key=<Key>

# The Auth Token field from App Keys info you copied previously
token=<Auth_Token>

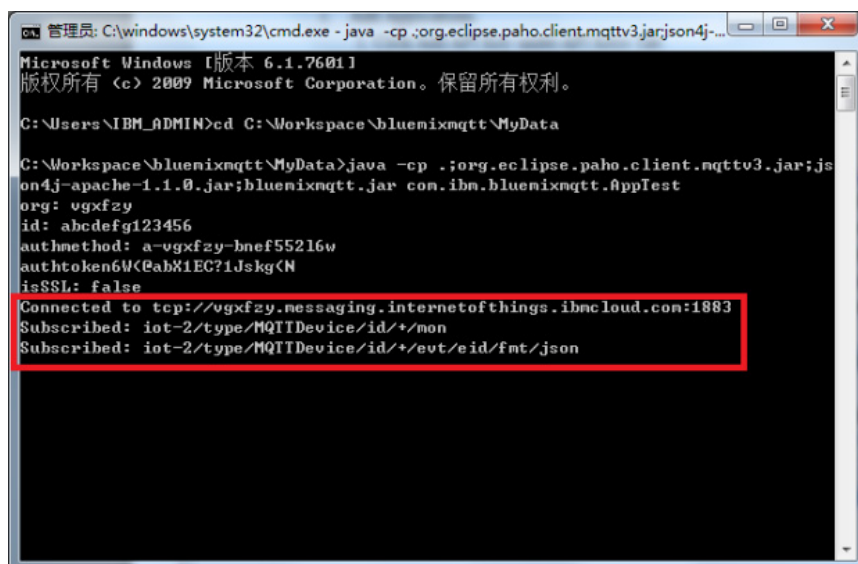
#T or F, T means using SSL, while F means not
isSSL=F
```

## Run the application-side program

After the configuration file is updated, open a command line, go to the MyData folder, and execute the following command to run the application-side program:

```
java -cp .;org.eclipse.paho.client.mqttv3.jar;json4j-apache-1.1.0.jar;bluemixmqtt.jar
com.ibm.bluemixmqtt.AppTest
```

When the application-side program has been successfully connected to the IoT service, you should receive a confirmation message similar to this:



```
管理员: C:\windows\system32\cmd.exe - java -cp .;org.eclipse.paho.client.mqttv3.jar;json4j-apache-1.1.0.jar;bluemixmqtt.jar com.ibm.bluemixmqtt.AppTest
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\IBM_ADMIN>cd C:\Workspace\bluemixmqtt\MyData

C:\Workspace\bluemixmqtt\MyData>java -cp .;org.eclipse.paho.client.mqttv3.jar;json4j-apache-1.1.0.jar;bluemixmqtt.jar com.ibm.bluemixmqtt.AppTest
org: vgxzfzy
id: abcdefg123456
authmethod: a-vgxfzy-bnef55216w
authtoken6W@abX1EC?1Jskg<N
isSSL: false
Connected to tcp://vgxfzy.messaging.internetofthings.ibmcloud.com:1883
Subscribed: iot-2/type/MQTTDevice/id/+mon
Subscribed: iot-2/type/MQTTDevice/id/+evt/eid/Fmt/json
```

## Update the device-side configuration file

The device-side configuration file is at MyData/device.conf. Update it according to the properties you copied while registering the device previously.

```
#Configuration files for Device Side Applications

#The org field from Device info you copied previously
org=<org>

#The id field from Device info you copied previously
deviceid=<id>

#The auth-token field from Device info you copied previously
token=<auth-token>

#T or F, T means using SSL, while F means not
isSSL=F
```

## Run the device-side program

After the configuration file is updated, open a command line, go to the MyData folder, and then execute the following command to run the device-side program.

```
java -cp .;org.eclipse.paho.client.mqttv3.jar;json4j-apache-1.1.0.jar;bluemixmqtt.jar
com.ibm.bluemixmqtt.DeviceTest
```

When the device-side program has been successfully connected to the IoT service, you should receive a confirmation message similar to this:

```
管理员: C:\windows\system32\cmd.exe - java -cp .;org.eclipse.paho.client.mqttv3.jar;json4j-apache-1.1.0.jar;bluemixmqtt.jar com.ibm.bluemixmqtt.DeviceTest
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\IBM_ADMIN>cd C:\Workspace\bluenixmqtt\MyData

C:\Workspace\bluenixmqtt\MyData>java -cp .;org.eclipse.paho.client.mqttv3.jar;json4j-apache-1.1.0.jar;bluenixmqtt.jar com.ibm.bluemixmqtt.DeviceTest
org: vgxfzy
id: aabbccddeei2
authmethod: use-token-auth
authtoken: LZDt?D4yAy4plm1lA
isSSL: false
Connected to tcp://vgxfzy.messaging.internetofthings.ibmcloud.com:1883
Subscribed: iot-2/cmd/cid/fmt/json
Send count as 0
.deliveryComplete(<> entered)
```

## Check running applications

Device-side programs publish status event messages (with the count value incrementing from 0 every 15 seconds). You can find relevant logs (including event and command messages details) from the command line.

In addition, you can follow the steps below to view the event details on the IoT service console.

1. Navigate to the IoT service console.

2. On the Devices tab, observe what happens to the relevant devices. In this case, the device id is `aabbccdde12`.

Device ID	Device Type	Last Event	Message Rate	Date Added	Added By
<code>aabbccdde12</code>	MQTTDevice	Just now	Every 14.2 seconds	Tuesday, December 30, 2014	chunbint@cn.ibm.com

Latest 10 Inbound Events  
Click on a row to get more detailed message information.

Event Type	Event	Timestamp
Message published	eid	Tuesday, December 30, 2014 4:12:59 PM
Device status	Connect	Tuesday, December 30, 2014 4:13:06 PM
Message published	eid	Tuesday, December 30, 2014 4:13:14 PM
Message published	eid	Tuesday, December 30, 2014 4:13:29 PM
Message published	eid	Tuesday, December 30, 2014 4:13:44 PM
Message published	eid	Tuesday, December 30, 2014 4:13:59 PM
Message published	eid	Tuesday, December 30, 2014 4:14:14 PM
Message published	eid	Tuesday, December 30, 2014 4:14:29 PM
Message published	eid	Tuesday, December 30, 2014 4:14:44 PM
Message published	eid	Tuesday, December 30, 2014 4:14:59 PM

3. Click one event to view the details of the message.  
Pay attention to the count field. As mentioned earlier, the value of the count field is incrementing from 0. Once it reaches 4, the application-side program publishes a command to the corresponding device to reset the value to 0. In that way, the count value will always be from 0 to 4, but never outside the domain.

### View Payload



Dec 30, 2014 4:14:59 PM

```
{"d":{"time":"2014-12-30 16:14:59","count":3}}
```

## Learn more

By following the steps above, you can build an app using the Bluemix IoT service via the MQTT protocol. If you are familiar with MQTT client programming, you may find there are no significant differences, except that when using the Bluemix IoT service you must make properties — such as the MQTT connection options, publishing and subscription subjects form, and so on — conform to the correct format.

It is easier to build solutions using the Bluemix IoT service when they are based on an existing MQTT client library. The good news is that there are quite a number of [open source libraries](#) for different platforms, such as C, C++, Java, JavaScript, Ruby, Go, and more.

Bluemix IoT services support a variety of connected smart devices, such as Arduino Uno, Raspberry Pi, and so on. See the [Bluemix documentation](#) for detailed info. If your device is in

the list, study the sample code first. The previously discussed MQTT mechanism should help you understand and modify the source code. If the device is not in the list, don't worry. Just as I implemented the device-side program in Java in my sample app, you can easily build your own program using an MQTT client library.

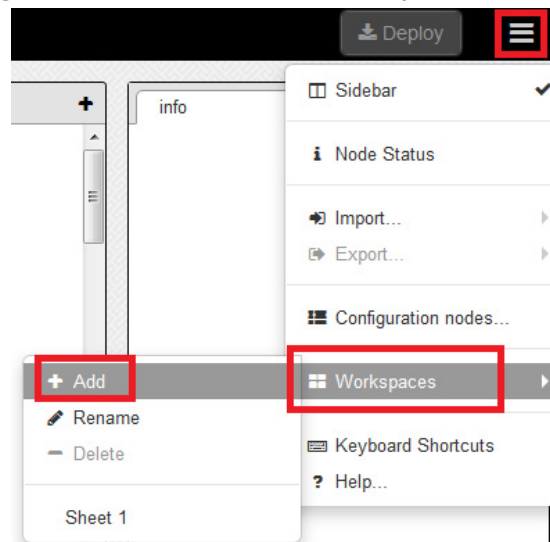
I implemented my application-side program in Java, but you may choose another programming language, such as Node.js or Ruby. To ease the application-side programming, you can use the browser-based flow editor [Node-RED](#) instead of coding it locally. Whether you are building it yourself or via Node-RED, the MQTT flow under the Bluemix IoT service works the same way.

## Application-side programming via Node-RED (Optional)

If you are interested in using the Node-RED tool to build your application-side program, follow these steps.

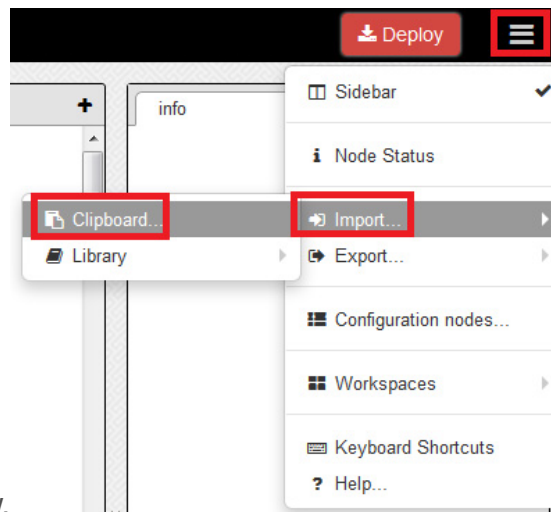
Node-RED is based on the Node.js runtime, and it is included in the Internet of Things Foundation Starter boilerplate that we created earlier.

1. Navigate to the app URL, which is usually `http://<app_name>.mybluemix.net`. In this case, it is <http://bluemixmqtt.mybluemix.net>.
2. Click the **Go to your Node-RED flow editor** button. You may need to scroll down a little to see the button.
3. Click the menu symbol in the top-right section. With the mouse pointer over **Workspaces**,



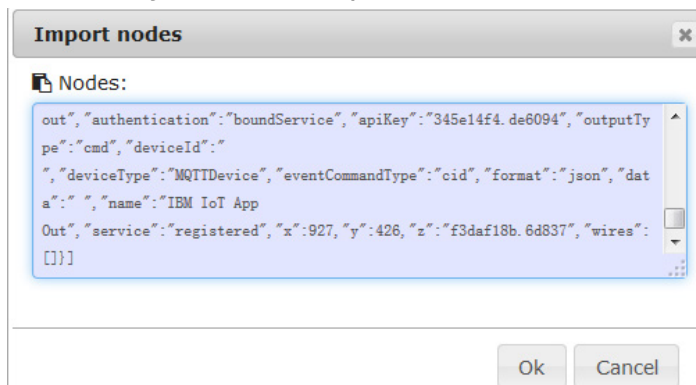
click **Add** to add a new workspace.

4. Click the menu symbol in the top-right section. With the mouse pointer over **Import...**, click



**Clipboard** to import the flow.

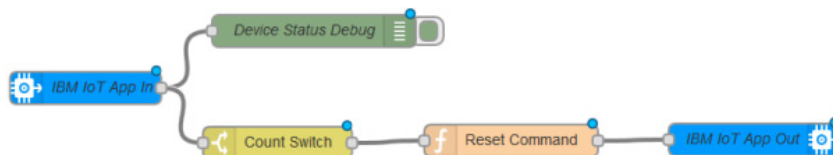
5. Copy the content from the MyData/redsampler.txt in the [source code](#) to the Nodes text field,



and then click **OK**.

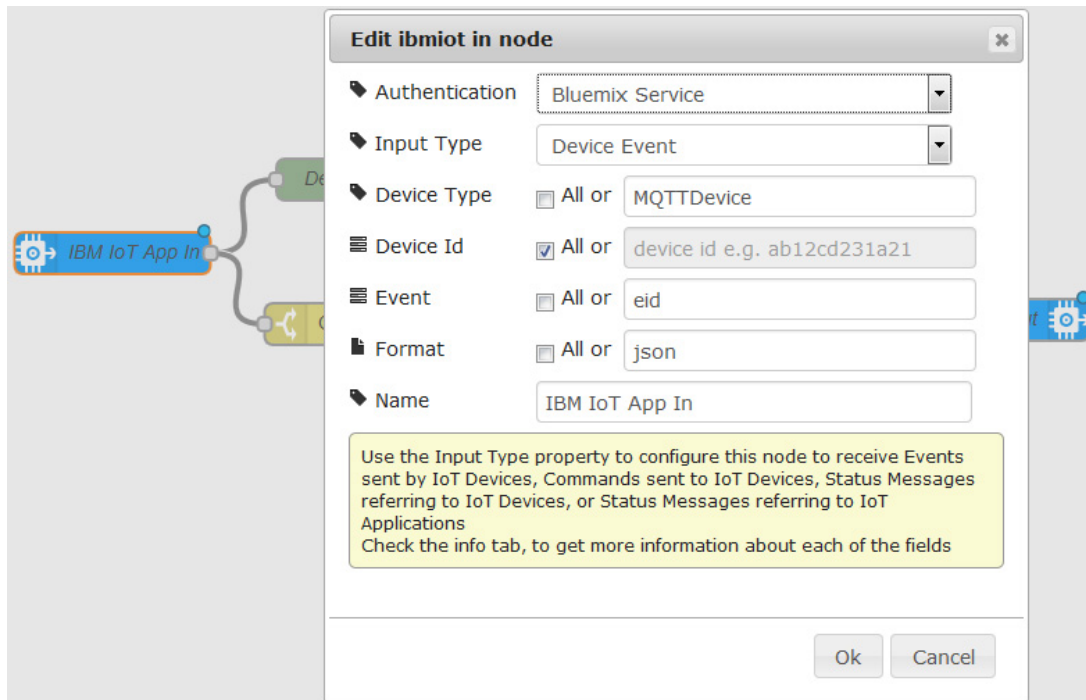
6. A confirmation indicates that the flow has been

imported.





7. Double-click each flow block. You will see its configuration dialog. These parameters may seem familiar as they are similar to those used in building the application-side program in



Java.

8. Click the **Deploy** button in the top-right section to deploy the application-side program.
9. You may run the device-side program again, and you will get similar results. On the right pane of the editor, you can view the messages from the device side as well.

It is more user friendly to build your application-side program using the Node-RED GUI editor. Node-RED provides many built-in flow blocks so that you can wire them together easily and deploy them immediately. For more information about using Node-RED on IoT, see the [Bluemix documentation](#).

**Note:** You do not need to register the application in the IoT service console manually, as it will be registered automatically when using the Node-RED editor. Choose the **Bluemix Service for Authentication** option when prompted.

## Conclusion

It is both simple and interesting to build applications with the Bluemix Internet of Things (IoT) service. In this tutorial, we explored the MQTT world behind the IoT service and explained how it works. A step-by-step tutorial showed how to build a sample solution using MQTT with either Java or the Node-RED editor. Because the IoT concept can be used in many different scenarios, you can use the sample app as a reference and compose your own solutions using the Bluemix IoT service to help interconnect the whole world.

The Internet of Things service <http://www.ibm.com/developerworks/topics/internet-of-things-service> provides simple but powerful application access to IoT devices and data.

RELATED TOPICS: [IoT](#) [MQTT](#) [Eclipse Paho Java](#)

© Copyright IBM Corporation 2015

([www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml))

[Trademarks](#)

([www.ibm.com/developerworks/ibm/trademarks/](http://www.ibm.com/developerworks/ibm/trademarks/))