

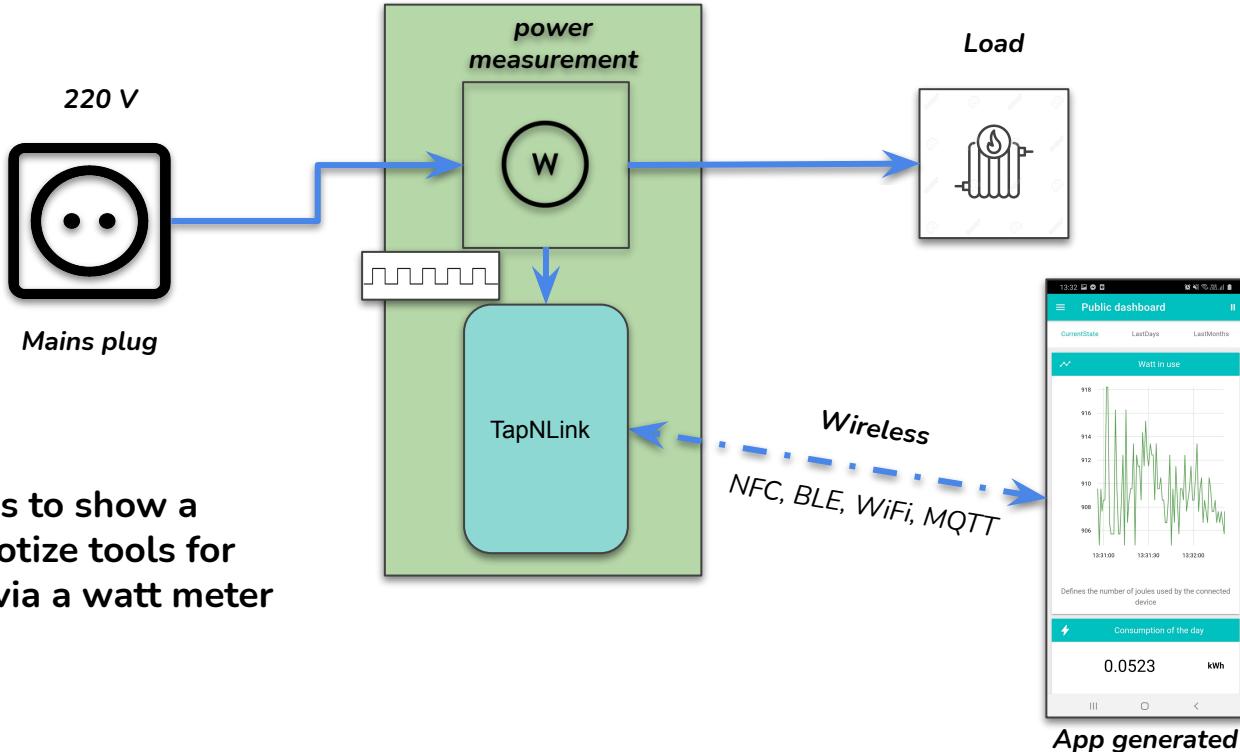
Creating the app generated **WattTap**

The purpose of this documentation is to show how
to create an simple Watt meter based on TapNLink





WattMeter Project Diagram

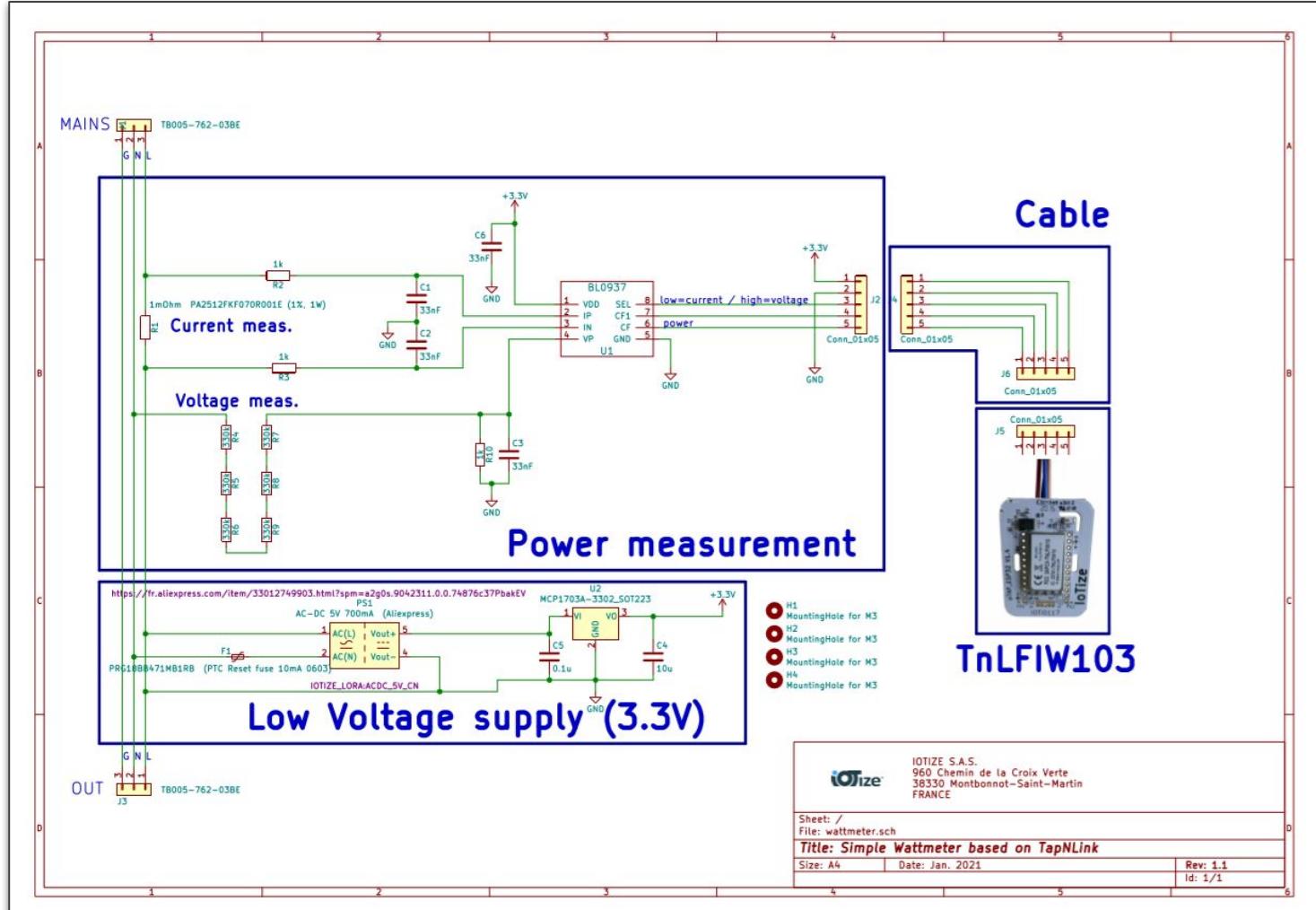




Here we can see that the tap is connected via five pads to the wattmeter circuit.

We connect the wattmeter system to a 1000W load.

TapNLink wattmeter schematic

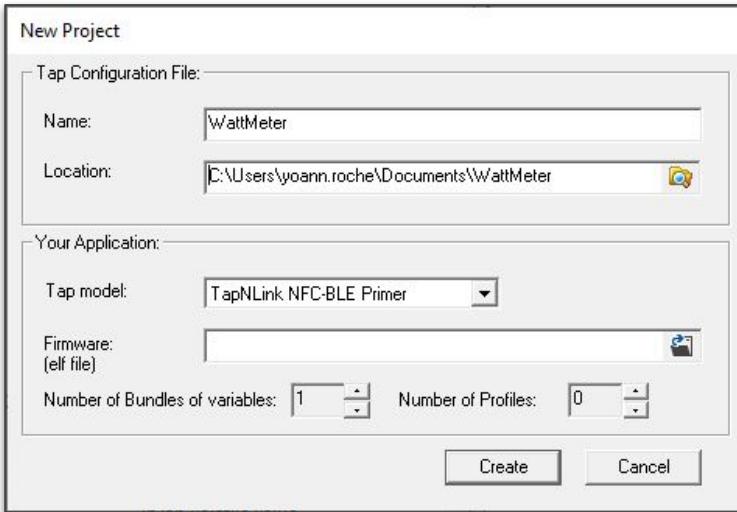


Project management from IoTize Studio

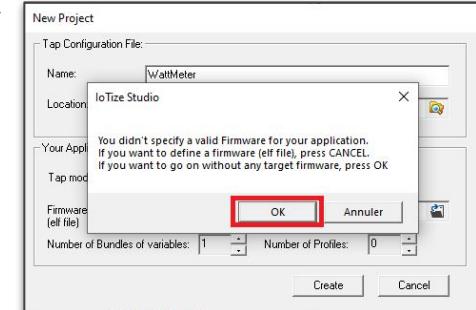
Find the IoTize Studio file on GitHub here : [WattMeterDemo](#)

1. Implementation of the project

- First we will create a new project on IoTize Studio : File/New (Ctrl + N).

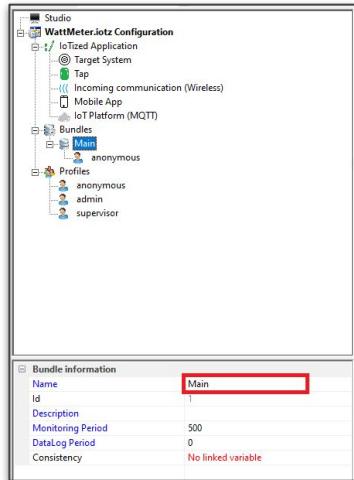


- Define the name of the project.
- Choose the location of the project on your computer.
- Select your tap model.
- Click on **Create** button.
- Press ok when the firmware warning appears ->

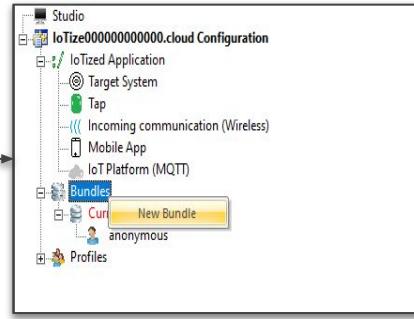


2. Implementation of Bundle

- Here we will create several Bundles to contain the variables

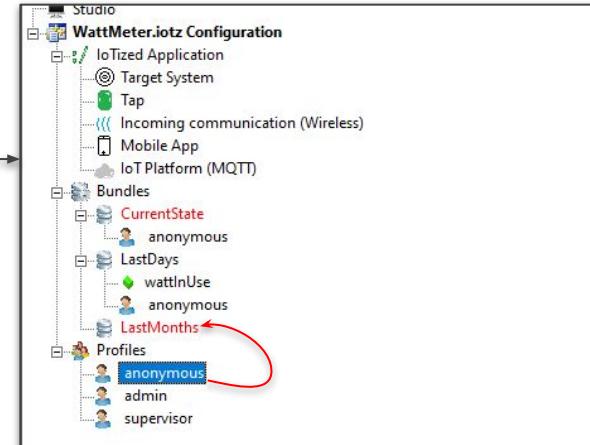


Rename the already existing bundle.



Create a bundle with a right click on the part of the **Bundle** branch.

Create a bundle for each variable group you want.



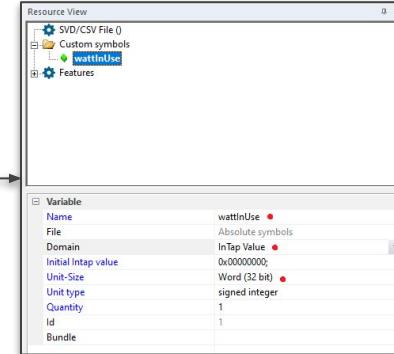
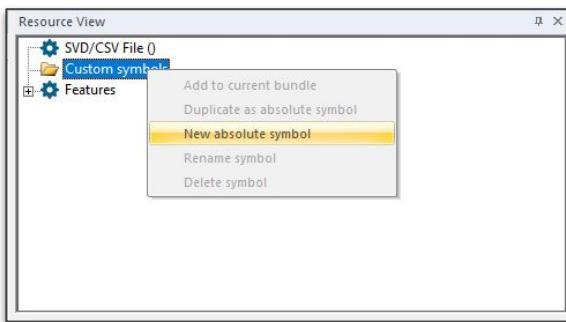
For each bundle created you must select them a profile. Drag'n drop a profile on a bundle.



On this demo we use only VarInTap value (volatile or not) but you can access to GPIO value/config or Target Variable.

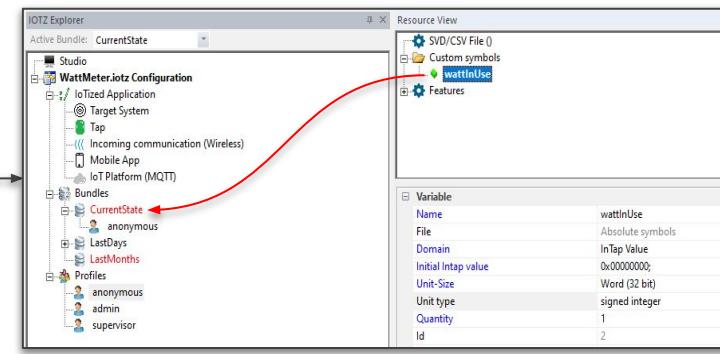
3. Implementation of VarInTap

- We will set up a variable for each information that must be retrieved in the generated application or in the java code



In the resource view frame, right-click on **Custom symbol** then **New absolute symbol**.

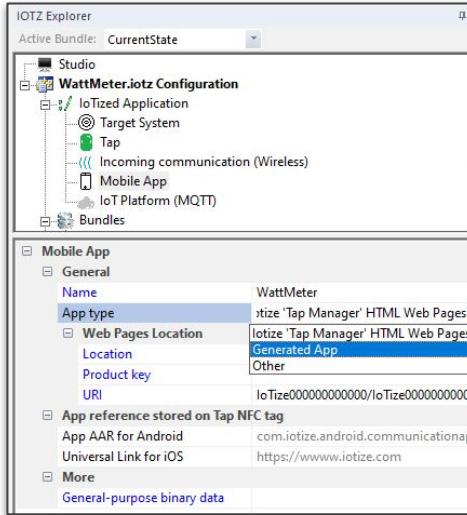
Once the item is created, it is necessary to modify this information as desired.



You can then assign the variable to a **Bundle** using drag n drop on the bundle.

Warning! The id may be different in the assigned bundle than resource view.

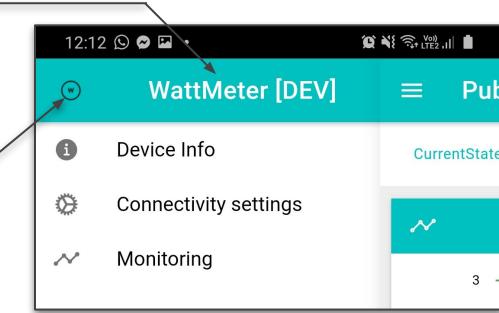
4. Implementation of the generated app



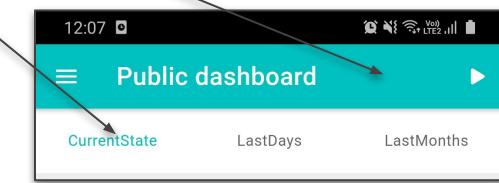
To generate an application, go to the **Mobile App** directory. In **General/App Type** select **Generated App**.

The screenshot shows the 'Generated App Settings' tab in the Mobile Studio. It includes sections for General (Name: WattMeter, App type: Generated App), Layout and Display (Columns per page: 2, App icon pathname: app/resources/icon.png, Splash screen pathname: app/resources/splash.png, Primary Color: #00c0c0, Secondary Color: #0cd1e8, Tertiary Color: #7044ff), and Communication Protocols (NFC Enabled: Yes, Energy harvesting: No; BLE Enabled: Yes; Socket Enabled: Yes, Remember last devices: Yes). Arrows from the 'Generated App' section in the IOTZ Explorer point to the corresponding settings in the Mobile Studio.

You can change the general appearance of the application via the **Generated App Settings** tab.



Splash screen is the loading screen during application opening



5. Improve the visual of the application

- For each created variable it is possible to define its visual aspect on the generated application

The screenshot shows a configuration interface for a variable named 'consumption'. The left panel displays the variable's properties under several sections: Variable information, Monitoring page generation, HTML Generation, and App generation.

Variable information:

- Name: consumption
- File: Absolute symbols
- Domain: IntTap Value
- Initial Intap value: 0.00000;
- Unit-Size: Word (32 bit)
- Unit type: float
- Quantity: 1
- Id: 1
- Bundle: CurrentState

Monitoring page generation:

- Display: Yes
- Editable: No
- Alias: Consumption of the day
- Byte order: B3_B2_B1_B0
- Unit: kWh

HTML Generation:

- Display as Floating Point: Yes
- Digits for float: 4
- Expression:

App generation:

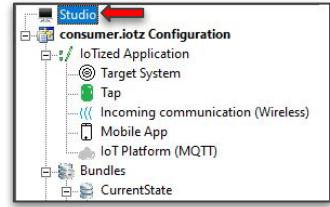
- Mode: Text
- Value formatting: 2em
- Card options:
 - Show in card: Yes
 - Header icon: flash
 - Header color name: primary
 - Additional text: Cumulates the consumption of the day. Starts from scratch on each new day.

Three cards are generated based on these settings:

- Card 1:** Consumption of the day (kWh) - Shows 0.0384 kWh. Description: Cumulates the consumption of the day. Starts from scratch on each new day.
- Card 2:** Consumption of the day (kWh) - Shows 0.0384 kWh. Description: Cumulates the consumption of the day. Starts from scratch on each new day.
- Card 3:** Consumption of the day (kWh) - Shows 0.0384 kWh. Description: Cumulates the consumption of the day. Starts from scratch on each new day.

6. Tap Connection

- To connect to the tap, go to the **Studio** branch and select the **Protocol**.



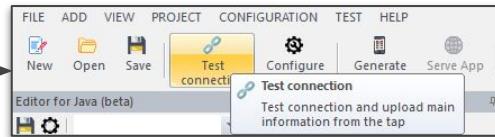
Connection to Tap	
Tap Id (serial number)	IoTize00610000101F
Encryption	Mid
Protocol	Socket
Socket host name/IP	192.168.20.999
Service name/Port	2000

The tap must be connected to the same wifi as the computer. Once connected to the wifi the tap, retrieve the ip of the tap connected to the wifi

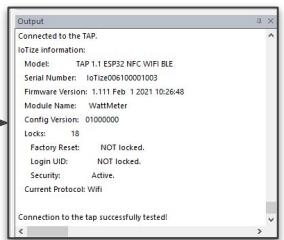
Connection to Tap	
Tap Id (serial number)	IoTize00610000101F
Encryption	Mid
Protocol	BLE
BLE device name/address	"consumer_01003 (70:b3:d5:9c:f0:04)"

Click on **BLE device name/address** to launch the available tap search.
Warning! Only one connection is possible at the same time.

To test the connection, click on the **Test connection** button



If the connection is good the **Output** tab should return this



Implementation of the Java code

Find the Java code on GitHub here : [WattMeterDemo](#)

7. Prepare a Java class for the JVM

- Below we find the body of a Java class with the strict minimum to be build

```
1 import com.iotize.jvm.*;  
2 import com.iotize.jvm.hal.*;  
3  
4  
5  
6 public class WattMeter {  
7  
8  
9     WattMeter() { ←  
10    }  
11  
12    public void check(final int id) { ←  
13    }  
14  
15    public void onException(int errorcode, int par1, int par2) { ←  
16    }  
17  
18    }  
19  
20 }
```

The class constructor. It is run only once at tap start. It can be used to instantiate variables.

The check method is called for each TapNLinkVar at the frequency defined in the variable constructor. The passed id is that of the TapNLinkVar which makes its check call.

The onException method is called if there is a serious error during the JVM process. If this method is called the call of the check methods is interrupted.

Warning ! If no TapNLinkVar is instantiated in the Java class, then the check method will never run.

8. Setting up the MQTT in the Java code

TapNLinkSys

We prepare an array of bytes to recover the topic.

```
6 public class WattMeter {  
7  
8     private TapNLinkSys system;  
9     private String topic;  
10  
11    WattMeter() {  
12        byte[] topicPrefixPayload = new byte[256];  
13        system.engineCommand(TapNLinkSys.GET, 1024, 0, 58, topicPrefixPayload );  
14        this.topic = new String(topicPrefixPayload) + "/result";  
15        system.sendMQTTMessage(topic, "initClass()", 0);  
16    }  
17  
18    public void check(final int id) {  
19    }  
20  
21  
22    public void onException(int errorcode, int par1, int par2) {  
23        system.sendMQTTMessage(topic, String.format("Error code : " + errorcode, errorcode, par1, par2), 0);  
24    }  
25 }
```

The system's engineCommand is used to retrieve the mqtt topic defined in the tap configuration. Pass the byte array that will be transformed and contain the topic.

One transforms the array of byte in String to recover the topic.

We use the sendMQTTMessage method of the system interface to signal that the class is well instantiated.

Here we send a mqtt message if the onException method is called, the message contains information about the error.

9. Definition of TapNLinkVar in Java code

TapNLinkVar

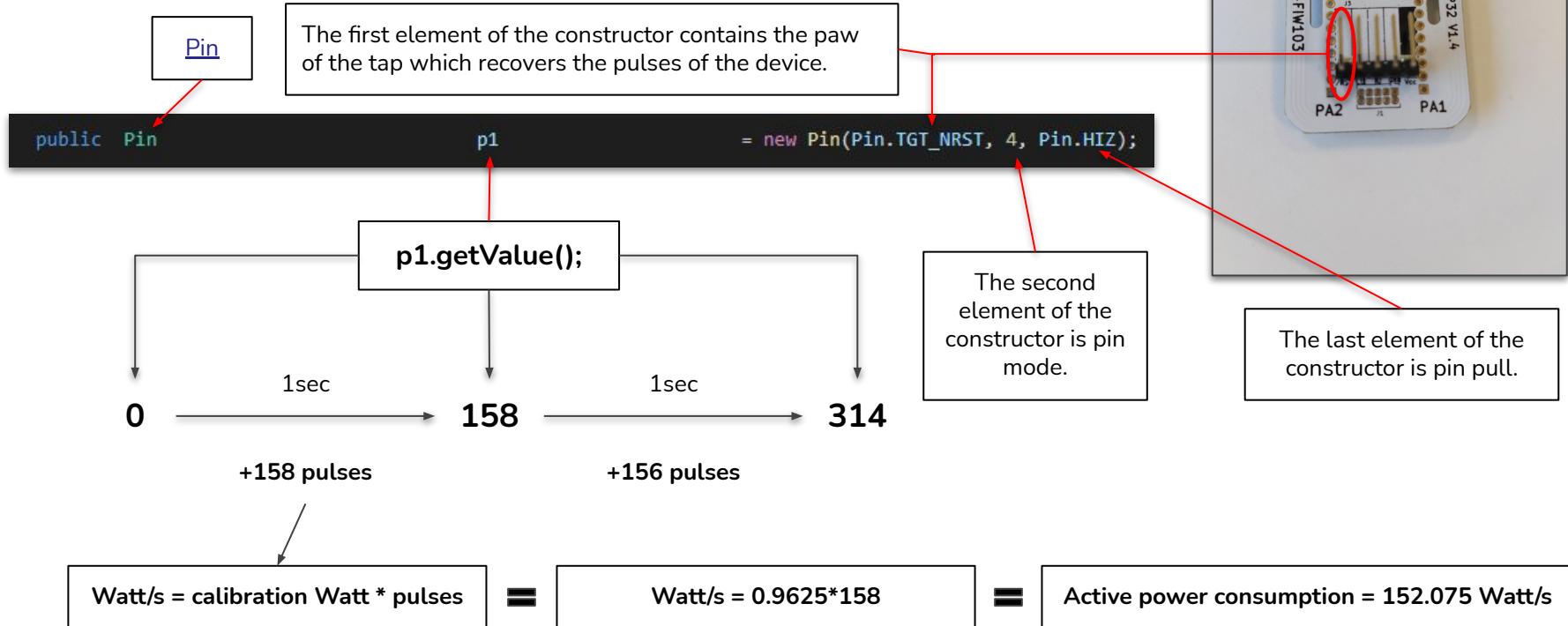
This value defines the call frequency of the check method with the id of this variable. The value is in milliseconds.

```
public TapNLinkVarInt higherWatt = new TapNLinkVarInt((short) 0x07, VariableType.INT16, "higherWatt", 1000);
```

We choose the class according to the variable created in the **Bundle**

Variable information	
Name	higherWatt
File	Absolute symbols
Domain	InTap Value
Initial Intap value	0x0000;
Unit-Size	Half-Word (16 bit)
Unit type	signed integer
Quantity	1
Id	7
Bundle	CurrentState

10. Reading pulses on Pin in Java code



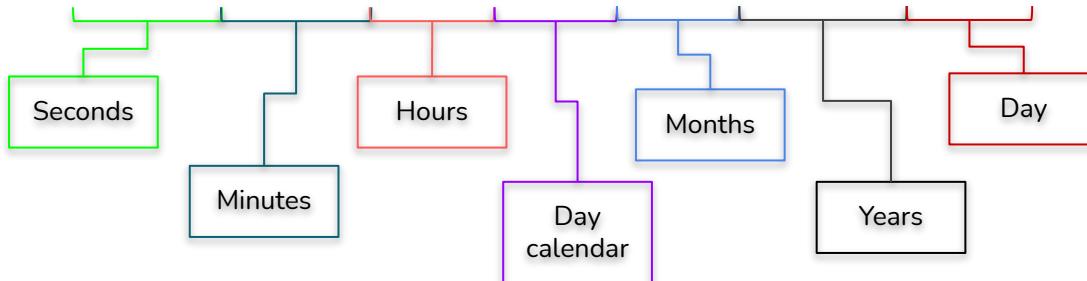
11. Absolute time recovery

```
public void currentTime() {
    try {
        system.engineCommand(TapNLinkSys.GET, RESSOURCE_ID, 0x00, 79, date);
        String month = "" + date[16] + date[17] + date[18] + date[19];
        String day = "" + date[24] + date[25] + date[26] + date[27];
        currentDay = (int) Integer.valueOf(day);
        currentMonth = (int) Integer.valueOf(month);
    } catch (TapNLinkException e) {
        system.sendMQTTMessage(topic, "Error on method currentTime(), error code : " + e.getCode(), 0);
    }
}
```

```
private byte[] date = new byte[36];
private int currentDay, currentMonth;
```

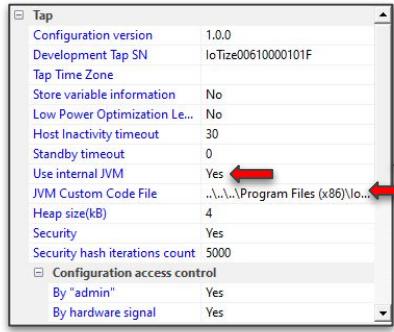
The system get command on 79 will recover the absolute time and transform the date object. In our example of the WattMeter only the day and the month are necessary.

date[] = 0.0.0.39.0.0.0.33.0.0.0.8.0.0.0.3.0.0.0.1.0.0.0.121.0.0.0.3.0.0.0.33.0.0.0



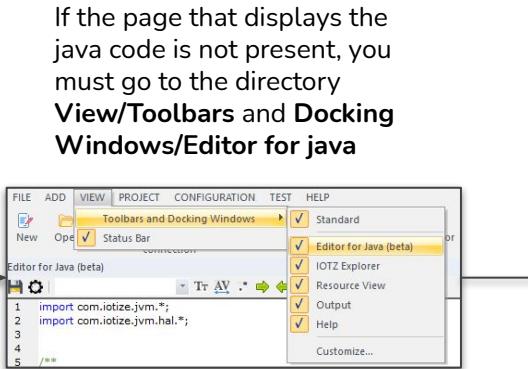


12. Set up the Java code in IoTizeStudio



Pass the **Use internal JVM** parameter to Yes.
In **JVM Custom Code File** set the path of your java file.

If the page that displays the java code is not present, you must go to the directory
View/Toolbars and Docking Windows/Editor for java



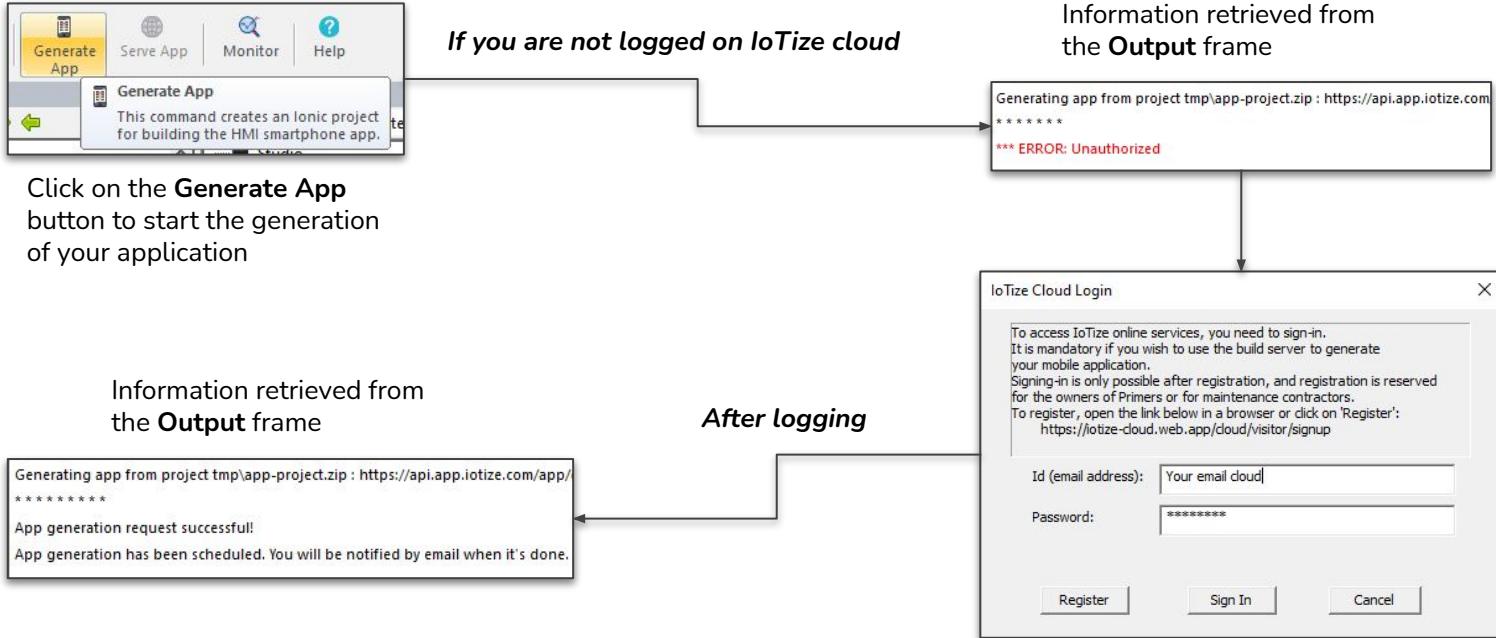
Run the build via the engravings button

```
import com.iotize.jvm.*;
import com.iotize.jvm.hal.*;
```

```
Start Building
C:\User\yann.roche\Documents>javac -version >classpath "C:\Program Files (x86)\IoTize\IoTize Studio\JVM"
javac 1.8.0_265
C:\User\yann.roche\Documents>IOT_JVM.exe -m4 >C:\Program Files (x86)\IoTize\IoTize Studio\Examples\Se
-----
IoT_JVM class to b2b converter
Copyright IOTIZE 2020.
Version 1.11
-----
Analyzing file C:\Program Files (x86)\IoTize\IoTize Studio\Examples\Sensors_STM32_Demo\java_src\WattMeter
Java class version: 52.0
Constant pool: 235 items
List of Methods (embedded byte code):
[280B] CODE: WattMeter.currentTime()V (range=[3773-3969])
[280Q] CODE: WattMeter.initState()V (range=[3715-3772])
[281Z] CODE: WattMeter.refreshTapValue()V (range=[3663-3696])
[281S] CODE: WattMeter.refreshFilter()V (range=[3697-3714])
[281C] CODE: WattMeter.<init>()V (range=[3040-3329])
[281D] CODE: WattMeter.check()V (range=[3330-3654])
[281E] CODE: WattMeter.onException()V (range=[3978-4040])
< This byte code requires at least version V1.109 for the JVM (TapNLUk version) >
< and cannot run with a JVM more recent than V1.255 >
Output file size (.bcb): 5292 bytes.
***** Build done.
```

Generation of the application

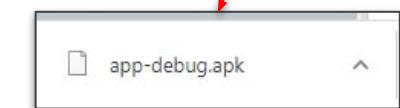
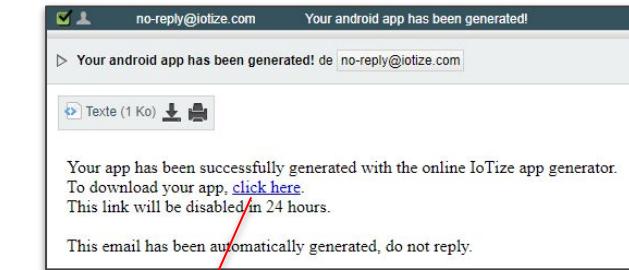
13. Generate App with IoTizeStudio



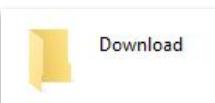


14. Retrieve the generated app on your mobile

After running the app generation you must have received an email containing a link to click to download the generated app.



Drop the app file on your mobile **Download** folder.



Then

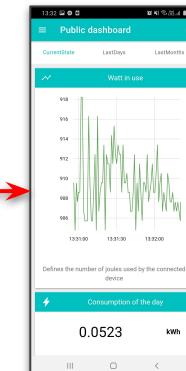


Install app-debug

5



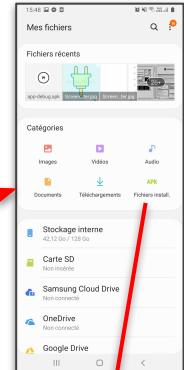
6



7



1



2

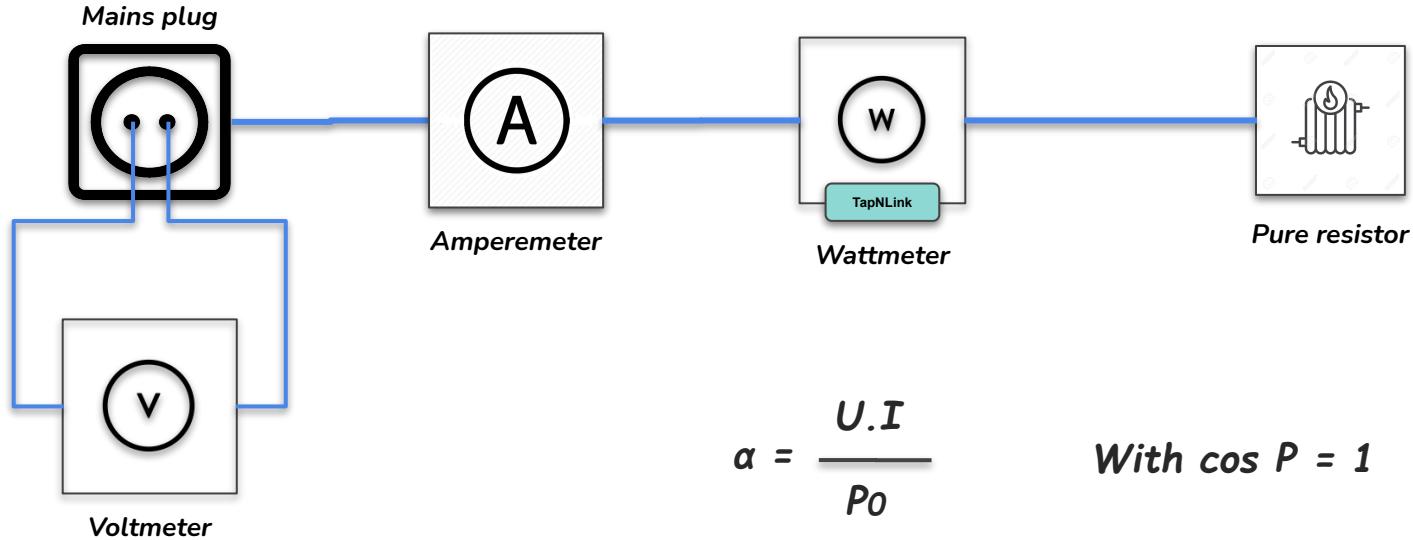


3

Optimize your results



15. Define the calibration



Implementation of AWS

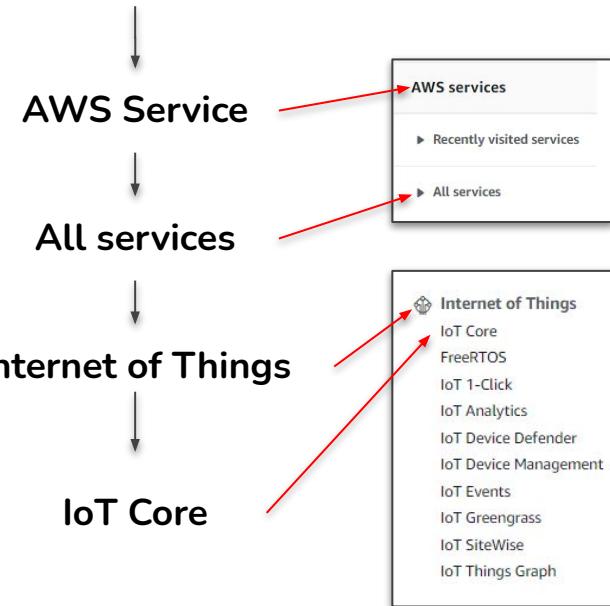
Find more information on doc IoTize here : [AWS IoT](#)



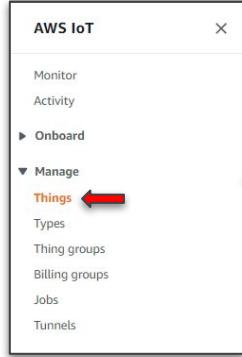
16. Create your AWS account

The screenshot shows the AWS Free Tier landing page. At the top, there's a navigation bar with links like Products, Solutions, Pricing, Documentation, Learn, Partner Network, AWS Marketplace, Customer Enablement, Events, Explore More, Contact Sales, Support, English, My Account, and Sign In to the Console. Below the navigation is a search bar. The main content area has a dark background with a geometric pattern. It features a large orange button labeled "Create a Free Account". Below this, there's a section titled "AWS Free Tier" with the sub-section "Types of offers". It lists three options: "Always free" (represented by a person icon), "12 months free" (represented by a calendar icon), and "Trials" (represented by a stopwatch icon). Each option has a brief description and a note about its availability. At the bottom of the page is a blue button labeled "Create AWS account".

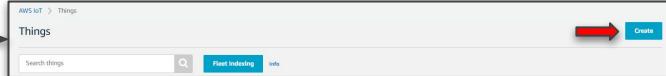
Once your account is created, go to the **AWS Management Console** page.
Then connect to **AWS** and go to main menu



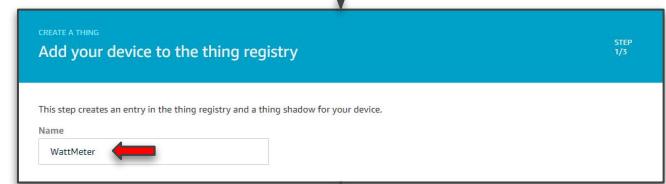
17. Create things on AWS



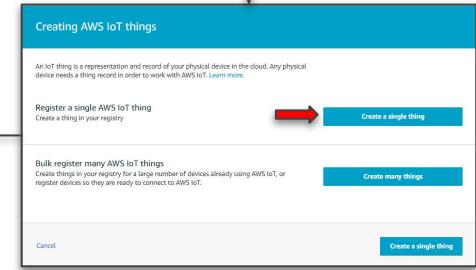
Click on **Things** in the left bar on tab **Manage**



Click on **Create** to access on **Creating AWS IoT things**



Set the name of the thing to **WattMeter**



Click on **Create a single thing**

In the WattMeter example the other information in the form is not necessary.
Press **Next** at the bottom of the form. sd





18. Create certificate for the thing

CREATE A THING
Add a certificate for your thing
STEP 2/2
A certificate is used to authenticate your device's connection to AWS IoT.
One-click certificate creation (recommended)
This will generate a certificate, public key, and private key using AWS IoT's certificate authority.
Create with CSR
Upload your own certificate signing request (CSR) based on a private key you own.
Get started
Use my certificate
Register your CA certificate and use your own certificates for one or more devices.
Create thing without certificate
Skip certificate and create thing
You will need to add a certificate to your thing later before your device can connect to AWS IoT.

Select **Create certificate** to generate AWS IoT's certificate authority.

Certificate created!
Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.
In order to connect a device, you need to download the following:
A certificate for this thing [REDACTED] cert.pem Download
A public key [REDACTED] publickey Download
A private key [REDACTED] private.key Download
You also need to download a root CA for AWS IoT:
A root Ca for AWS IoT Download
Activate

It is important to click **Activate** for the root CA for AWS IoT. Once activated click on **download** above to be taken to the download page of Amazons root CA.

Download **A certificate for this thing** and **Private key**. Place them in a file reserved for your WattMeter project.

Done Attach a policy

To complete the creation, click **Done** to return to the list of things page with your new thing



19. Create a policy for the thing



Click on
Policies in the
left bar on tab
Secure

Create a policy

Create a policy to define a set of authorized actions. You can authorize actions on one or more resources (things, topics, topic filters). To learn more about IoT policies go to the [AWS IoT Policies documentation page](#).

Name

Add statements

Policy statements define the types of actions that can be performed by a resource.

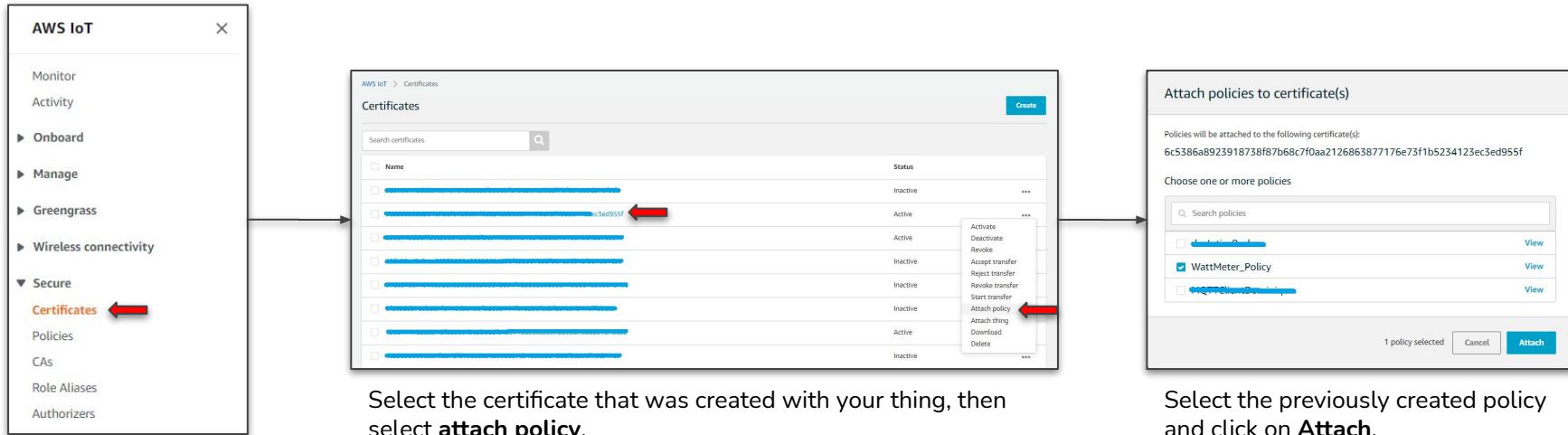
Action

Effect Allow Deny

Complete the policy creation form as shown in the picture.
Once completed click on
Create.



20. Define a policy for certificate of the thing



Select the certificate that was created with your thing, then select **attach policy**.

Click on **Certificates** in the left bar on tab **Secure**

Select the previously created policy and click on **Attach**.



21. Define IoT Platform (MQTT) on IoTize Studio

IoT Platform (MQTT)

Enable Relay Yes

Enable Cloud Yes

Platform

IoT Platform AWS IoT

Select the **IoT Platform** with **AWS IoT**

IoT Platform (MQTT)

Enable Relay Yes

Enable Cloud Yes

Platform

IoT Platform AWS IoT

AWS information.

Organization Custom Endpoint [REDACTED]

Thing Name WattMeter

Certificate [REDACTED]

Private Key [REDACTED]

AWS IoT root CA certificate ..\\AmazonRootCA1.pem

Recover previously uploaded files in your WattMeter project folder

THING WattMeter

WATTMETER_TYPE

Details Security Thing groups Billing Groups Shadows Interact Activity Jobs Violations Defender metrics

This thing already appears to be connected.

HTTPS

Update your Thing Shadow using this Rest API Endpoint. Learn more

https://wattmeter-1-west-1.amazonaws.com

MQTT

Use topics to enable applications and things to get, update, or delete the state information for a Thing (Thing Shadow). Learn more

Go back to your created thing and go to the **Interact** tab.



22. Implementation of the MQTT message to AWS on Java code

```
system.sendMQTTMessage("$aws/things/WattMeter/shadow/name/inUse/update",
  "{\"state\": {\"reported\": {\"ID\": \"WattMeter\", \"CurrentDayConsumption\": " + consumptionDay + ", \"wattInUse\": " + wattInUse.getValue() + \"}}}", 0);
```

This command will send a mqtt message on the aws topic that will define a new shadow if it is not existing. The message contains two values that will be implemented in the **inUse** shadow

