SAMSUNG
ARTIK™ Modules

Samsung EBV Training Workshop

TABLE OF CONTENTS

Introduction 2
Hardware set up 4
Lab 1: Stream Telemetry Data from ARTIK055s to ARTIK710 5
1.1 Install ARTIK IDE/SDK 5
1.2 Create an Application for ARTIK055s 5
1.3 Adapt the Application 7
1.4 Stream sensor data to ARTIK710 gateway device via MQTT 8
Lab 2: Integrate with InfluxDB and dashboard 11
2.1 Mosquitto 11
2.2 Write incoming MQTT data to local InfluxDB 13
2.3 Visualize data with Grafana 15
Lab 3: Predictive Analysis with TensorFlow 19
3.1 Use subscriber3.py to predict pressure sensor data 19
3.2 Visualize the predicted data 20
Lab 4: Connect to PTC ThingWorx 22
4.1 Compile ThingWorx SDK for ARTIK 710 22
4.2 Launch ThingWorx server instance 22
4.3 Configure and Run the SteamSensor Example Application 24
4.4 View the steamed data from ThingWorx portal 26

Introduction

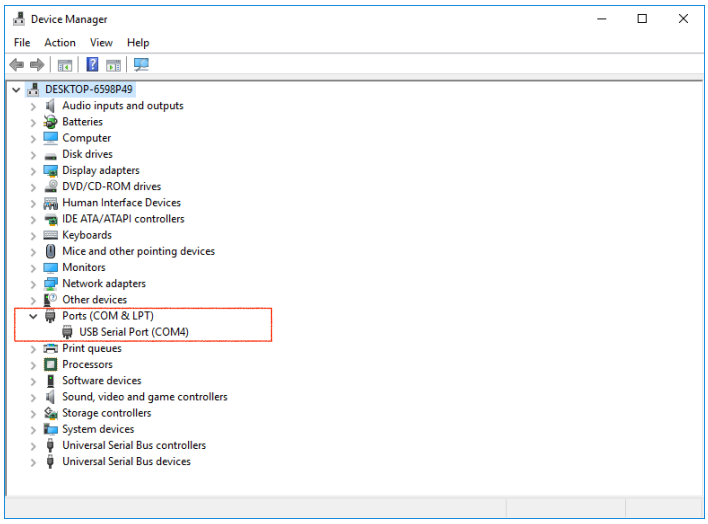
This hands-on lab shows how you can build a smart factory solution by using ARTIK.

In the lab, we explain how to use ARTIK055s to collect sensor data, stream it to ARTIK 710 via MQTT. On ARTIK 710, we use InfluxDB for data storage and Grafana for visualization. We also use TensorFlow running locally on an ARTIK710 for predictive analysis and abnormally detection.

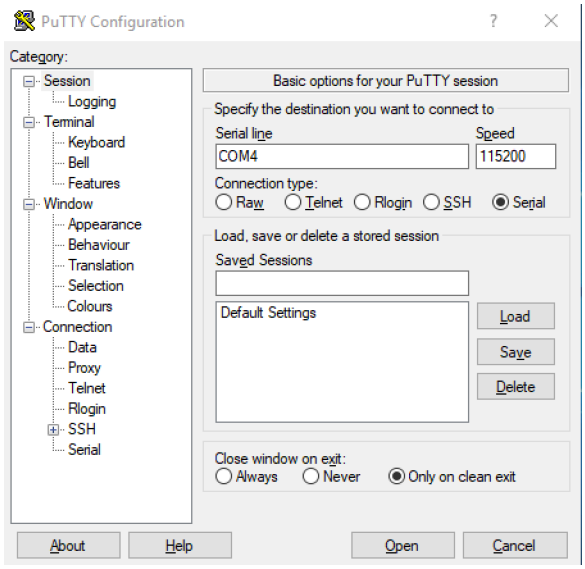
Connect ARTIK 055s development board to your host machine using an USB cable.

On Windows host machine:

1. Find your serial device in your Device Manager / Ports and remember the COM port your ARTIK board is using. In our example, the board uses COM4 port.

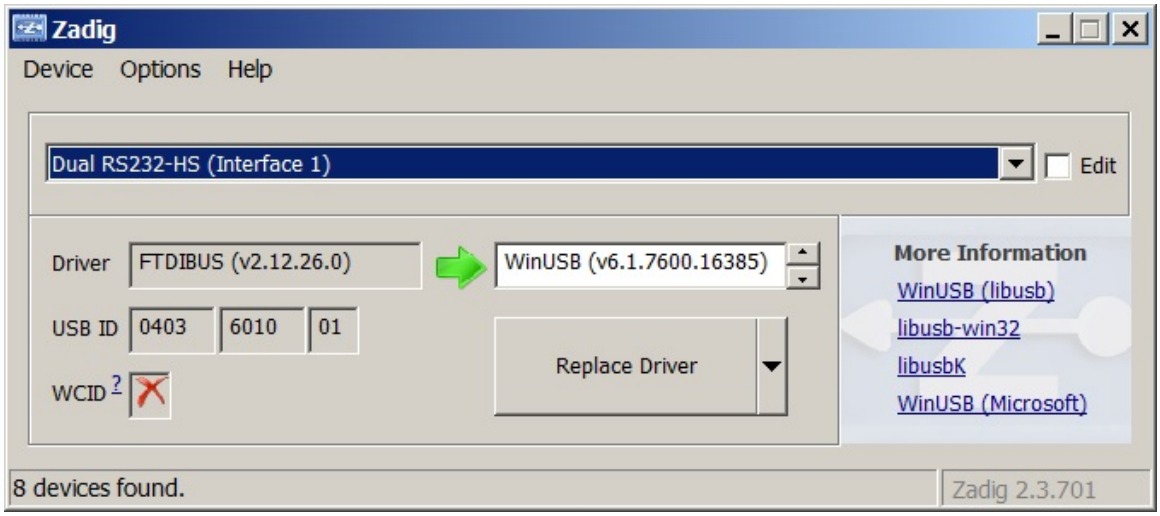


2. Please install and launch a serial console application PuTTY.. Select “Serial” as the Connection type, use the COM port number you find from Device Manager and Speed 115200 for serial access.



3. For Windows users, Please

- Browse to the ARTIK_IDE_Installation_Directory\tools directory.
- Run the zadig-2.3.exe application.
- Under **Options**, select **List All Devices**.
- Under the devices drop-down list, select Interface **1** of the dual COM port
- Update the driver to the WinUSB driver by clicking **Replace Driver**.
- Now select Interface **0** of the FTDI device, and change it to a `libusb` driver.



On Mac host machine:

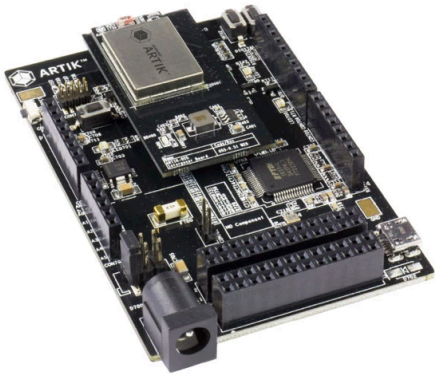
For Mac users, you can use the built-in *screen* utility to access the serial console. Normally the device name ends with `141B` or `142B`.

```
screen /dev/cu.usbserial-XXXXXXX 115200
```

Hardware set up

1. Connect the USB cable to ARTIK 055S board and your host machine.
2. Connect the pressure sensor:
   Yellow wire: A0
   Black wire: GND
   Red wire: 3.3V

Lab 1: Stream Telemetry Data from ARTIK055s to ARTIK710

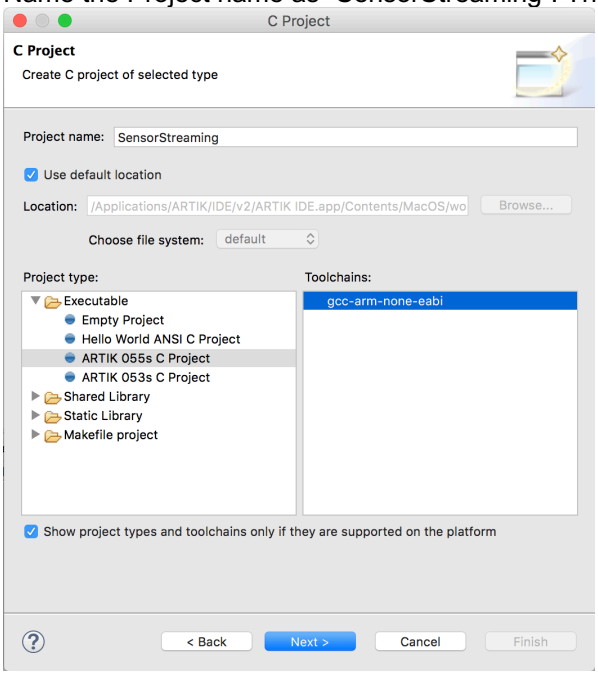


1.1 Install ARTIK IDE/SDK

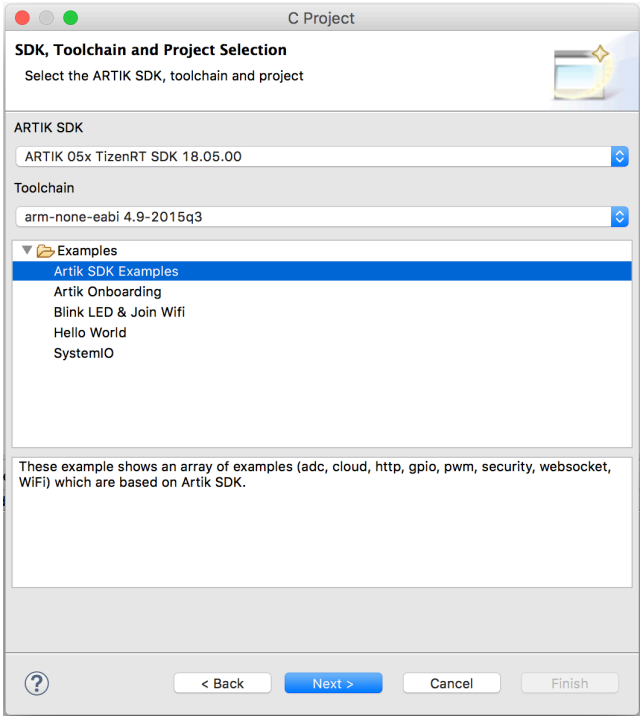
As a prerequisite, you should have installed ARTIK IDE on your host machine. If you haven't done so, please follow the instructions at https://developer.artik.io/documentation/developer-guide/artik-ide/install-ide.html to set up your development environment.

1.2 Create an Application for ARTIK055s

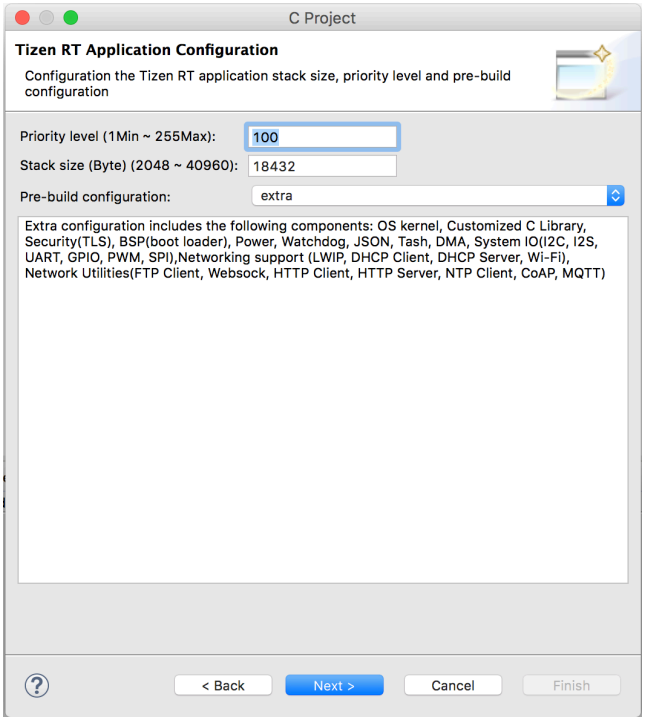
1. Launch your ARTIK IDE, go to File->New->C Project.
2. In the "C Project" dialog, select "ARTIK 055s C project", and choose "gcc-arm-none-eabi" as the default toolchain. Name the Project name as "SensorStreaming". Then, click Next.



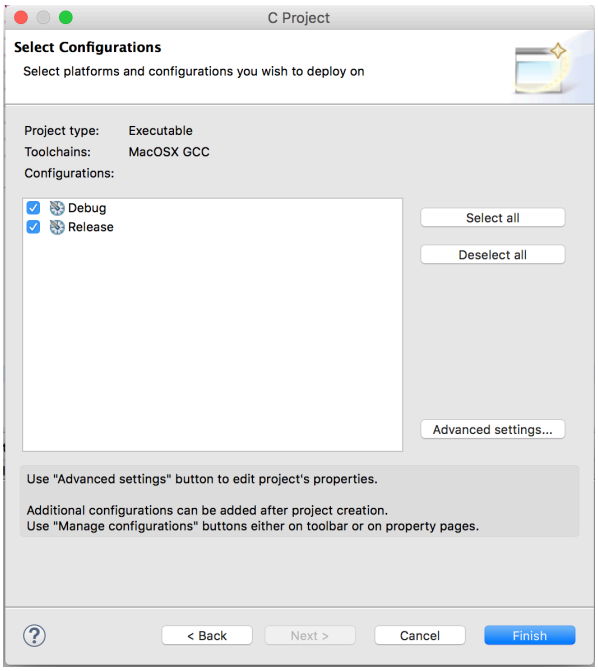
3. In "SDK, Toolchain and Project Selection" dialog, select "ARTIK 05x TizenRT SDK 18.05.00" as the target SDK version. Click on "Artik SDK Examples", we will use this example as our template to generate an application for sensor data collecting. Then, click Next.



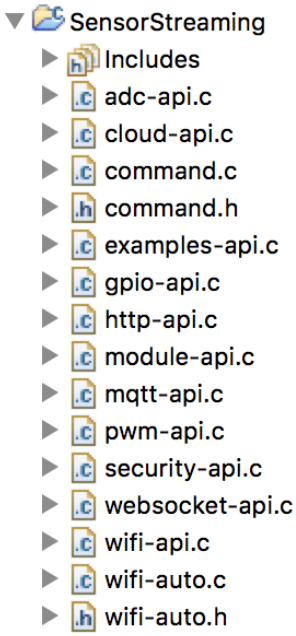
4. In “Tizen RT Application Configuration” dialog, use the default Priority level and Stack Size settings. Change the Pre-build configuration to “extra”, then click Next.



5. In “Select Configurations” dialog, keep the default Debug and Release builds settings, and click Finish.



6. Now you have an application ‘SensorStreaming’ created in your IDE Project Explorer.



1.3 Adapt the Application

1. Open wifi-auto.c, look for WIFI_SSID at the beginning of the file, and configure SSID/PWD by using the credentials of our router.

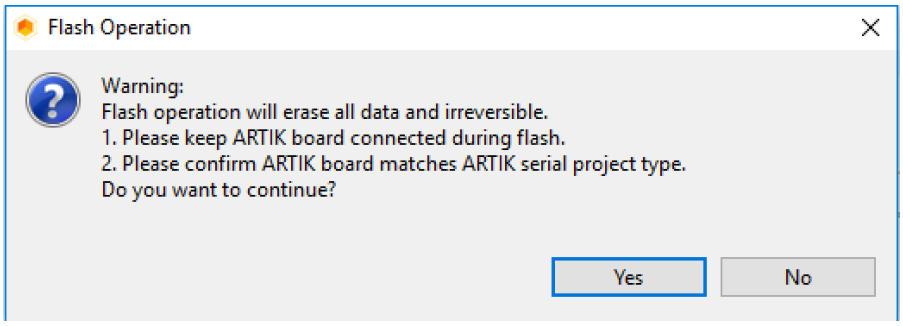
```
#define WIFI_SSID “ARTIK2”
#define WIFI_PASSPHRASE “artik123”
```

2. Download mqtt-api.c from workshop github page. Drag and drop the file to SensorStreaming project. When you are prompted to overwrite existing file, click Yes to proceed.

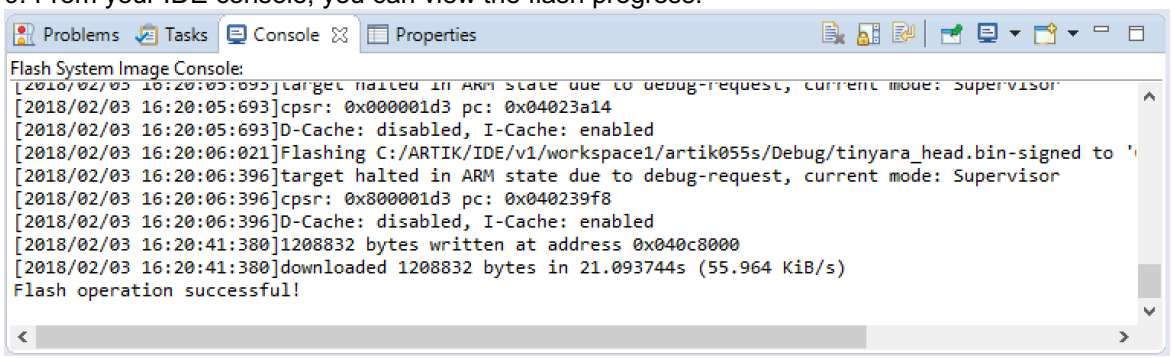
3. Build the application by selecting Project->Build Project. You can see the build log from the IDE Console tab.

4. Once you have a successful build, click the “Flash System Image” button  to flash the newly built image to the board.

As a precaution, a warning dialog will pop up to confirm you want to flash the connected ARTIK device. Click “Yes” to proceed.



9. From your IDE console, you can view the flash progress.



1.4 Stream sensor data to ARTIK710 gateway device via MQTT

1. Push the Reset button.

```
U-Boot 2017.01-00013-g814fa7d (Jan 08 2018 - 15:17:26 +0900)

CPU:   Exynos200 @ 320 MHz
Model: ARTIK-05x based on Exynos T20
DRAM:  1.3 MiB
WARNING: Caches not enabled
BL1 released at 2017-3-13 15:00
SSS released at 2017-09-12
WLAN released at 2017-12-21
Flash: 8 MiB
*** Warning - bad CRC, using default environment

In:    serial@80180000
Out:   serial@80180000
Err:   serial@80180000
Autoboot in 50 milliseconds
gpio: pin gpg16 (gpio 46) value is 1
## Starting application0at 0x040C8020 ...
s5j_sflash_init: FLASH Quad Enabled
i2c_uioregister: Registering /dev/i2c-0
i2c_uioregister: Registering /dev/i2c-1
System Information:
        Board: ARTIK053S
        Version: 1.0.12
        Commit Hash: d1dfb56208b5897ac852a713b6f760ce4efdbd7a
        Build User: ARTIK@Samsung
        Build Time: 2018-07-17 15:58:54
        System Time: 01 Jan 2010, 00:00:00 [s] UTC Hardware RTC
Support
        ARTIK SDK version: 1.8.0
TASH>>
```

2. From your serial console, you should be able to see the message below when the board boots up.

3. From the ARTIK 055s TASH shell, connect to the MQTT broker running on ARTIK710. Use your ARTIK710's IP address to replace the one below. Press <Enter> to continue.

```
TASH>>mqtt connect <YOUR_ARTIK710_IP_ADDRESS>
Starting supplicant in foreground...
Connected to ARTIK
Client mosq/;L<pQ^K@gIIb3Xt11u sending CONNECT
TASH>>Client mosq/;L<pQ^K@gIIb3Xt11u received CONNACK
MQTT connection result: OK
```

4. Publish sensor telemetry data to ARTIK710 by running 'mqtt publish'.

```
TASH>>mqtt publish
ADC0=268
MQTT: publish 268 on Machine
Client mosq/W0MtF?rTGxm<u<TJ]v sending PUBLISH (d0, q0, r0, m1,
'Machine', ... (3 bytes))
TASH>>Assage published: 1
ADC0=1601
MQTT: publish 1601 on Machine
Client mosq/W0MtF?rTGxm<u<TJ]v sending PUBLISH (d0, q0, r0, m2,
'Machine', ... (4 bytes))
MQTT message published: 2
ADC0=1611
MQTT: publish 1611 on Machine
Client mosq/W0MtF?rTGxm<u<TJ]v sending PUBLISH (d0, q0, r0, m3,
'Machine', ... (4 bytes))
MQTT message published: 3
ADC0=582
MQTT: publish 582 on Machine
Client mosq/W0MtF?rTGxm<u<TJ]v sending PUBLISH (d0, q0, r0, m4,
'Machine', ... (3 bytes))
MQTT message published: 4
```

Lab 2: Integrate with InfluxDB and dashboard

This lab is for ARTIK710. In this lab, we will subscribe to incoming MQTT sensor data on ARTIK710, write it to InfluxDB and use Grafana for visualization.

2.1 Mosquitto

The MQTT broker mosquitto runs as a daemon on your ARTIK 710, and listens for incoming MQTT messages from your ARTIK055s. We will spin up a MQTT Client on the 710, which subscribes to the pressure sensor data coming from ARTIK 055s, then writes the data to local InfluxDB.

Under /root directory, you can find a *subscriber1.py* script, which is a MQTT client that subscribes to MQTT messages from the local broker.

```
import paho.mqtt.client as mqtt
import datetime
import time

def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
    client.subscribe("Machine")

def on_message(client, userdata, msg):
    # Use utc as timestamp
    receiveTime=datetime.datetime.utcnow()
    message=msg.payload.decode("utf-8")
    isfloatValue=False

    try:
        # Convert the string to a float so that it is stored as a number in the
database
        val = float(message)
        isfloatValue=True
    except:
        print("Could not convert " + message + " to a float value")
        isfloatValue=False

    if isfloatValue:
        print(str(receiveTime) + ": " + msg.topic + " " + str(val))

# Initialize the MQTT client that should connect to the Mosquitto broker
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
connOK=False
while(connOK == False):
    try:
        client.connect("localhost", 1883, 60)
        connOK = True
    except:
        connOK = False
    time.sleep(2)

# Blocking loop to the Mosquitto broker
client.loop_forever()
```

Launch your MQTT client on ARTIK710 which your ARTIK 055s is publishing, you can see the incoming MQTT messages as below.

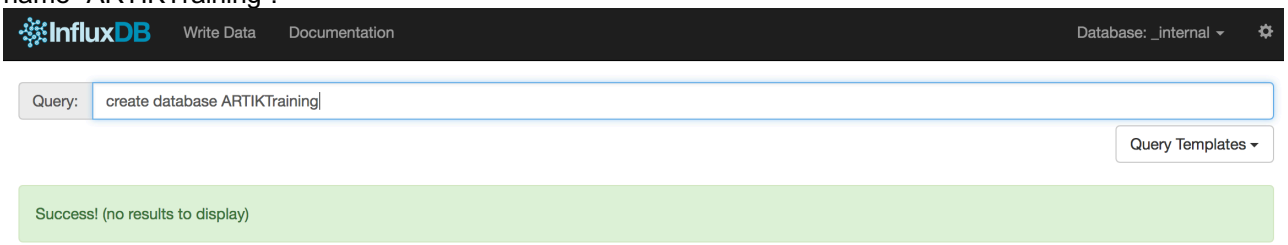
```
[root@artik ~]# python subscriber1.py
Connected with result code 0
2018-09-09 07:04:44.453532: Machine 1632.0
2018-09-09 07:04:49.454750: Machine 1634.0
2018-09-09 07:04:54.450260: Machine 1632.0
2018-09-09 07:04:59.451163: Machine 555.0
2018-09-09 07:05:04.451638: Machine 697.0
2018-09-09 07:05:09.453303: Machine 1332.0
2018-09-09 07:05:14.454881: Machine 1635.0
```

2.2 Write incoming MQTT data to local InfluxDB

On your ARTIK 710, we have pre-installed InfluxDB and Grafana for data storage and visualization.

1. Make sure your host machine is on the same network as your target ARTIK 7x device. Launch a browser and visit <YOUR_ARTIK710_IP_ADDRESS>:8083. Port 8083 is the default admin interface of your local InfluxDB.

2. In the Query box, enter “create database ARTIKTraining”, then press Enter. This creates a database with the name “ARTIKTraining”.



3. In the python script subscriber2.py under your /root, it not only listens for incoming MQTT messages with the topic “Machine”, but also writes the sensor data included in the message to InfluxDB. The database is the “ARTIKTraining” we just created, and measurement name we use is *ketchupfactory*.

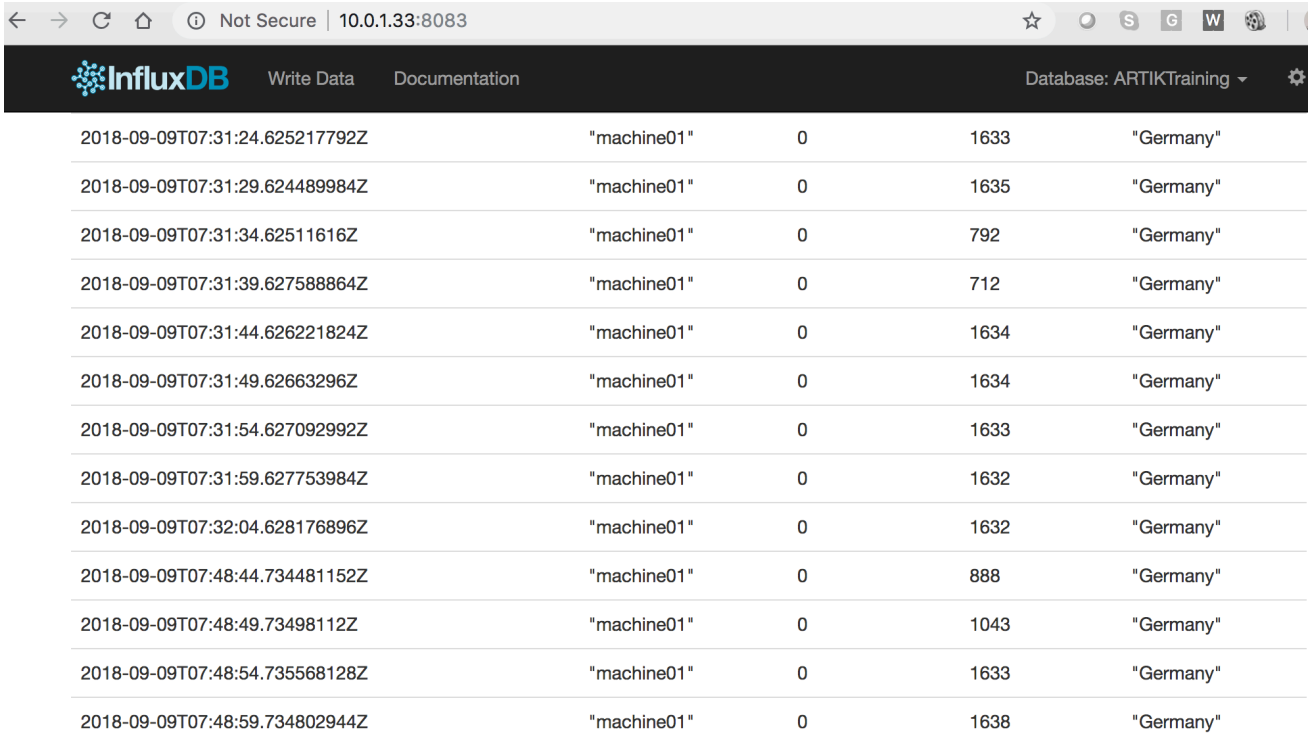
   The code below extends the Python script from the previous exercise, generates a JSON message for InfluxDB.

```
dbname = 'ARTIKTraining'
dbclient = InfluxDBClient('localhost', 8086, 'root', 'root',dbname)

json_body = [
        {
            "measurement": "ketchupfactory",
            "tags": {
               "host": "machine01",
               "region": "Germany"
            },
            "time": receiveTime,
               "fields": {
                 "pressure": val,
                 "predicted": predicted_value
               }
            }
        ]

dbclient.write_points(json_body)
```

4. Run *subscriber2.py* on your ARTIK710, while your ARTIK055s publishes data to it.

```
[root@artik ~]# python subscriber2.py
2018-09-09 07:31:09.661484: Machine 1633.0
2018-09-09 07:31:14.623046: Machine 1635.0
2018-09-09 07:31:19.625288: Machine 1630.0
2018-09-09 07:31:24.625218: Machine 1633.0
2018-09-09 07:31:29.624490: Machine 1635.0
2018-09-09 07:31:34.625116: Machine 792.0
2018-09-09 07:31:39.627589: Machine 712.0
```

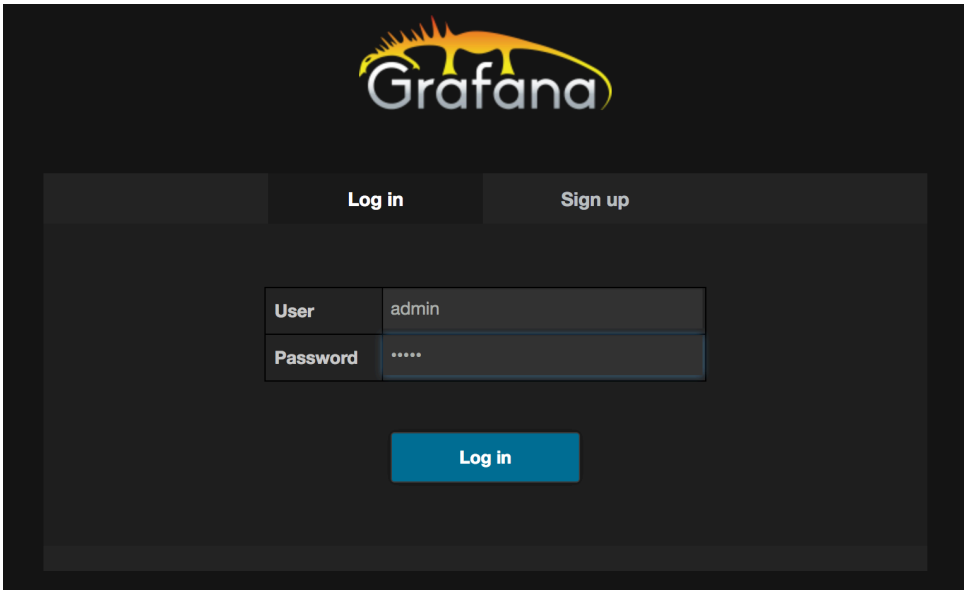
5. Now, if you go back to your InfluxDB portal. Select ARTIKTraining from the dropdown list, and in Query box, enter “select * from ketchupfactory”, and press Enter. You will be able to see the streamed sensor data from the portal.



2018-09-09T07:31:24.625217792Z	"machine01"	0	1633	"Germany"
2018-09-09T07:31:29.624489984Z	"machine01"	0	1635	"Germany"
2018-09-09T07:31:34.62511616Z	"machine01"	0	792	"Germany"
2018-09-09T07:31:39.627588864Z	"machine01"	0	712	"Germany"
2018-09-09T07:31:44.626221824Z	"machine01"	0	1634	"Germany"
2018-09-09T07:31:49.62663296Z	"machine01"	0	1634	"Germany"
2018-09-09T07:31:54.627092992Z	"machine01"	0	1633	"Germany"
2018-09-09T07:31:59.627753984Z	"machine01"	0	1632	"Germany"
2018-09-09T07:32:04.628176896Z	"machine01"	0	1632	"Germany"
2018-09-09T07:48:44.734481152Z	"machine01"	0	888	"Germany"
2018-09-09T07:48:49.73498112Z	"machine01"	0	1043	"Germany"
2018-09-09T07:48:54.735568128Z	"machine01"	0	1633	"Germany"
2018-09-09T07:48:59.734802944Z	"machine01"	0	1638	"Germany"

2.3 Visualize data with Grafana

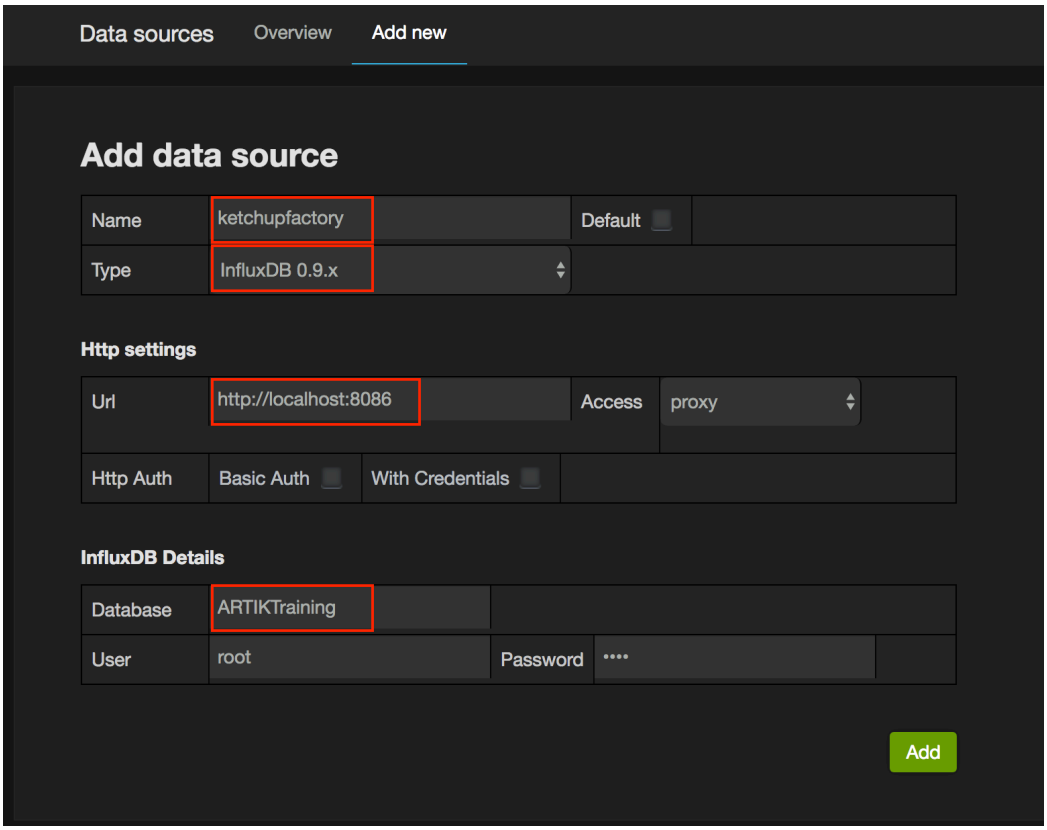
From your browser, Grafana can be accessed from your ARTIK710's IP address, port 3000. Use default username/password "admin"/"admin" to log in.



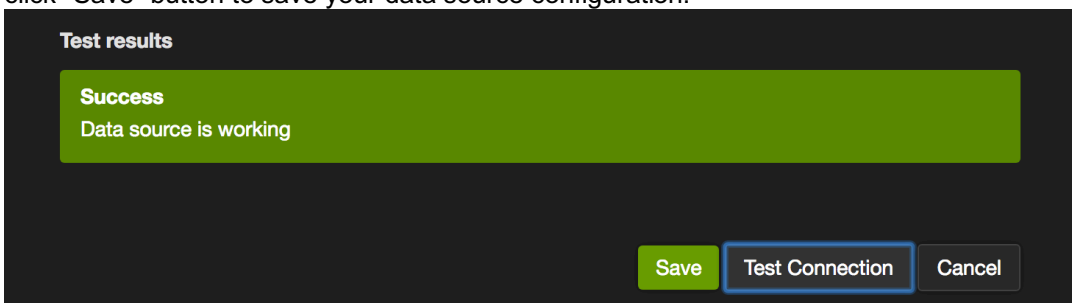
Click Data Source from left pane, and "Add new".

1. Add a datasource. Enter the data source details as below, then click "Add".
   - In the Name field, enter measurement name "ketchupfactory".
   - Select "InfluxDB 0.9.x" from dropdown list as the data source.
   - InfluxDB runs http service at port 8086. In Grafana http settings, configure Url as http://localhost:8086 in order to connect InfluxDB with Grafana.

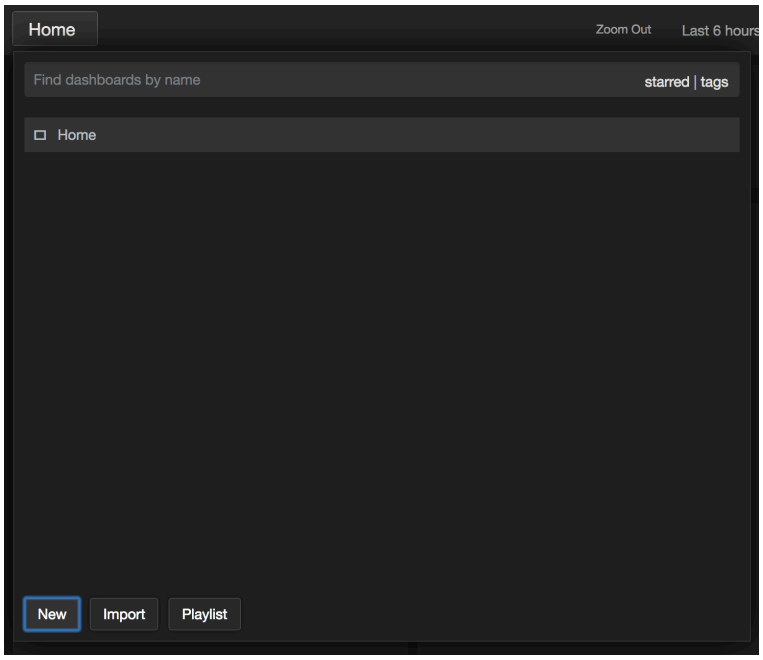
- Database name is the “ARTIKTraining” we just created.

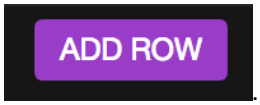


2. Click the “Test Connection” button to verify your InfluxDB connection. Once you verify your Data source is working, click “Save” button to save your data source configuration.

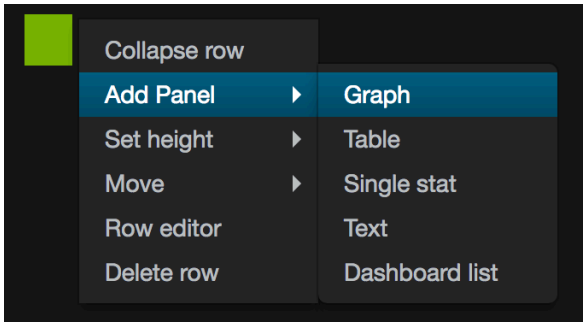


3. As the next step, we will add a dashboard for visualization. Click the Home button next to Grafana icon on the top, and select “New”.

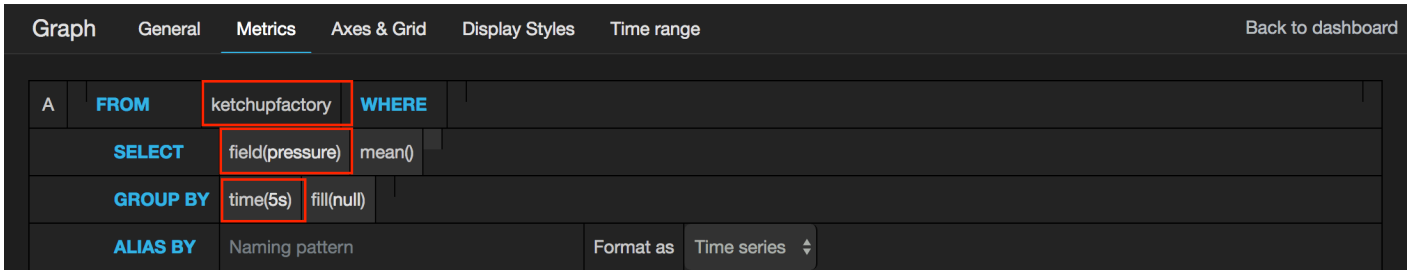


Click 

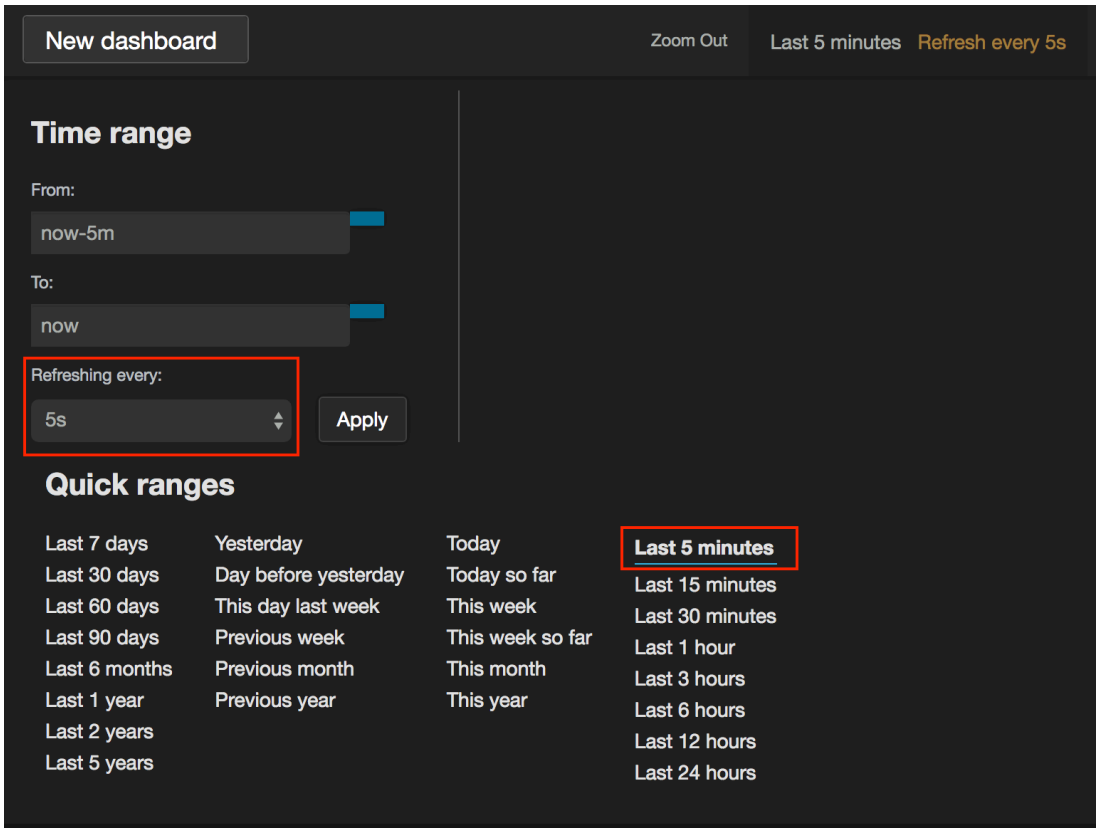
. You will see a floating green tab on the left side. Click on it, then select “Add Panel->Graph”



4. Configure your Query like below. If you have configured your data source correctly, measurement name, data fields should automatically pop up.



Then, you will be able to see the pressure data in the dashboard. By default, the dashboard shows the data collected in the last 6 hours. You can click the “Last 6 hours” on top right corner, and make it show the data in the last 5 minutes, and change the refresh rate to every 5s.



You will be able to see the graph as below.

Lab 3: Predictive Analysis with TensorFlow

Now, we will further extend our Python code to run the sensor reading against a pre-trained TensorFlow model for predictive analysis.

We already loaded a pre-trained model and will feed a window of 4 pressure sensor readings to the model for prediction. The model is trained by using the pattern that rotates between low pressure and high pressure. To simulate the model, you need to release the pressure sensor for 10 secs, then press it for 10 secs.

3.1 Use subscriber3.py to predict pressure sensor data

In subscriber3.py, we retrieve the last 3 pressure sensor readings from the database, append the latest sensor reading to it, and feed these data to our TensorFlow model.

```
results = dbclient.query("SELECT pressure FROM ketchupfactory ORDER BY time
DESC LIMIT 3")
points = results.get_points()
items = []
X = []
Items.insert(0, val)
for item in points:
  items.insert(0, item['pressure'])
X.append(items)
X=np.array(X)
print("Historical Data")
print(X)
classes = model.predict(X)
predicted_value = int(classes[0][0])
print("Predicted Data")
print(predicted_value)
print("\n")
```

Launch the python script

```
[root@artik ~]# python subscriber3.py
Using TensorFlow backend.
Keras model loaded
Connected with result code 0
Received 348.0 at 2018-09-10 04:07:51.404227
Historical Data
[[1059. 1633. 1634.  348.]]
Predicted Data
845

Received 1202.0 at 2018-09-10 04:07:56.402263
Historical Data
[[1633. 1634.  348. 1202.]]
Predicted Data
1629

Received 1635.0 at 2018-09-10 04:08:01.404101
Historical Data
[[1634.  348. 1202. 1635.]]
Predicted Data
1646
```

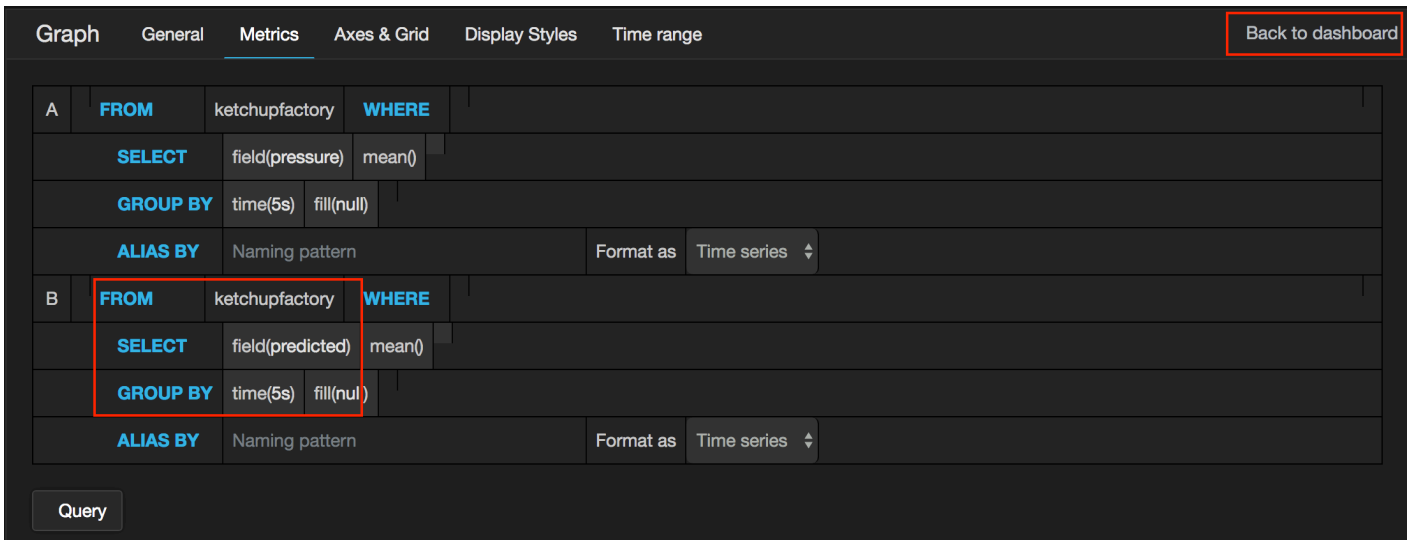
3.2 Visualize the predicted data

From your Grafana portal, click the 'Panel Title' , and select Edit button.

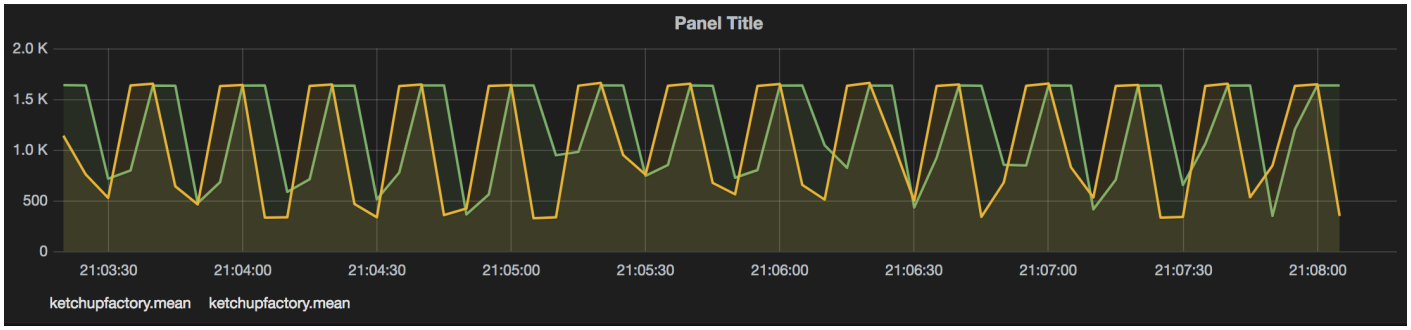
You should be able to see the view as shown below. Click 'Metrics', then Query button to add a new query to the graph.

Select ketchupfactory measurement, select predicted field and update it every 10 secs.

Click Back to dashboard.



Overtime, you will be able to see the charts of predicted data and actual data. In the graph below, the green line represents the actual sensor data, and yellow line is the predicted data.



Bonus Question:

Now, we know the predicted data and actual data, can you further extend your Python script for anomaly detection?

Lab 4: Connect to PTC ThingWorx

4.1 Compile ThingWorx SDK for ARTIK 710

Download ThingWorx C SDK c-sdk-2.1.1.636-master to your host machine, then scp it to your ARTIK710 /root/ directory, unzip it.

1. From your ARTIK710, Change to your ThingWorx C SDK directory.

```
[root@artik /]# unzip c-sdk-2.1.1.636-master.zip
[root@artik /]# cd /root/c-sdk-2.1.1.636-master/
[root@artik c-sdk-2.1.1.636-master]#
```

2. Replace the CMakeLists.txt file under c-sdk-2.1.1.636-master by using the version from our workshop page.
3. Create a `/bin` directory under C-SDK to build in.

```
[root@artik c-sdk-2.1.1.636-master]# mkdir bin
[root@artik c-sdk-2.1.1.636-master]# cd bin
```

4. Run cmake to generate Makefiles. CMake will produce a set of project files that are compatible with your development environment.

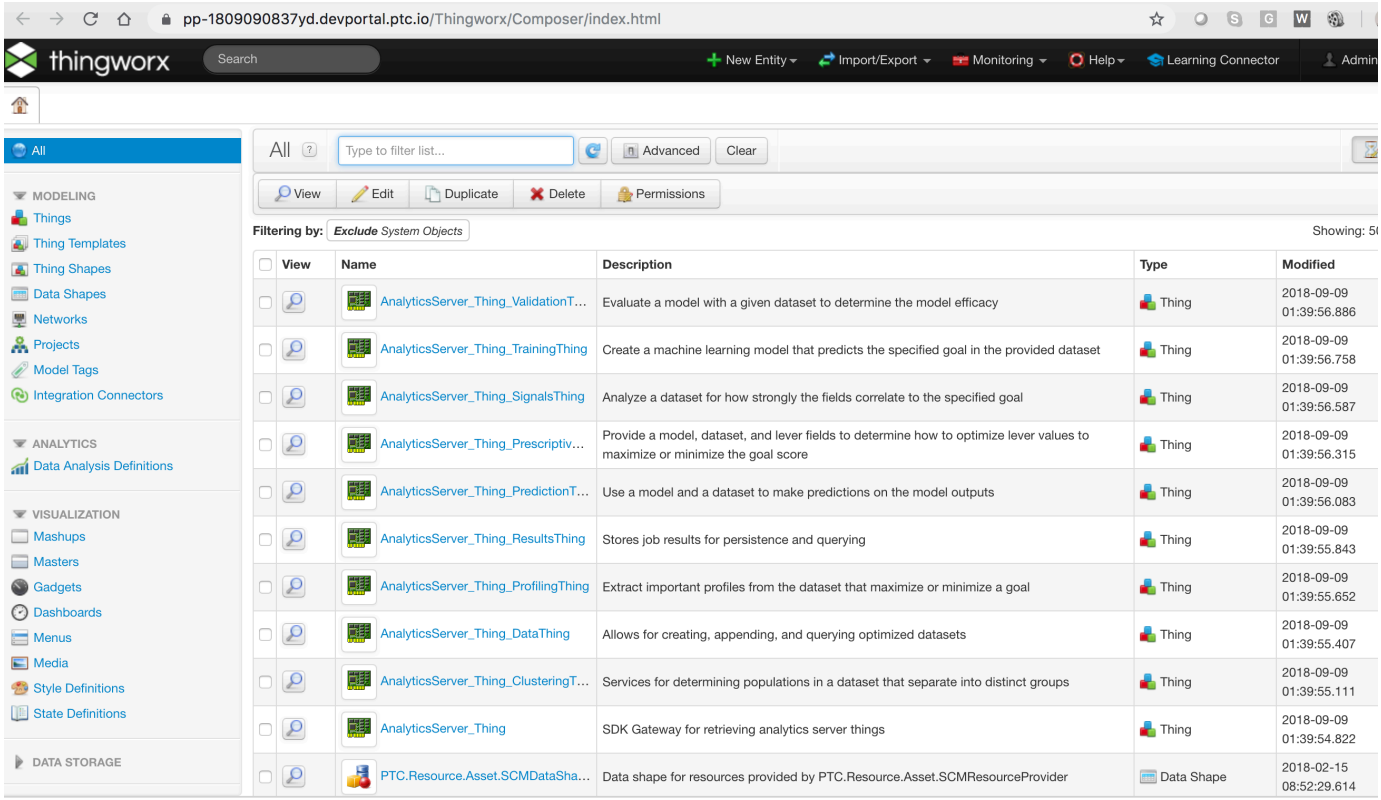
```
[root@artik bin]# cmake ..
```

5. Run make to compile the SDK and sample files. Executables will be generated under the bin directory.

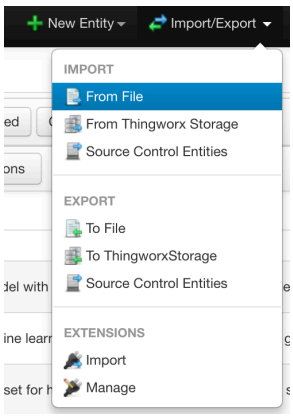
```
[root@artik bin]# make
```

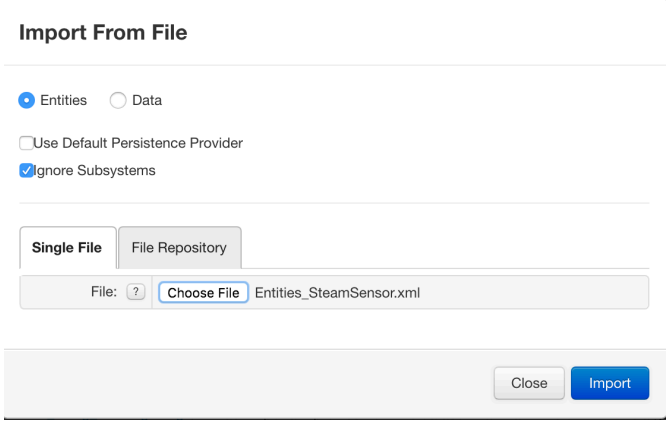
4.2 Launch ThingWorx server instance

From your host machine, launch a browser and go to https://developer.thingworx.com/profile/hosting to apply for your ThingWorx trial server instance if you haven't. Click "Start Server" to start the server instance and then "Launch" the instance.

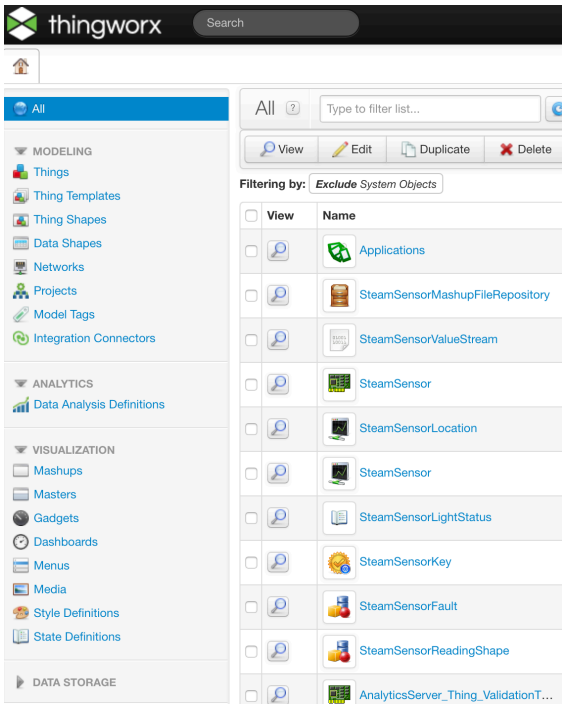


Import the provided Entities_SteamSensor.xml file to your ThingWorx Composer by selecting Import->From File-> Choose Entities_StreamSensor.xml.





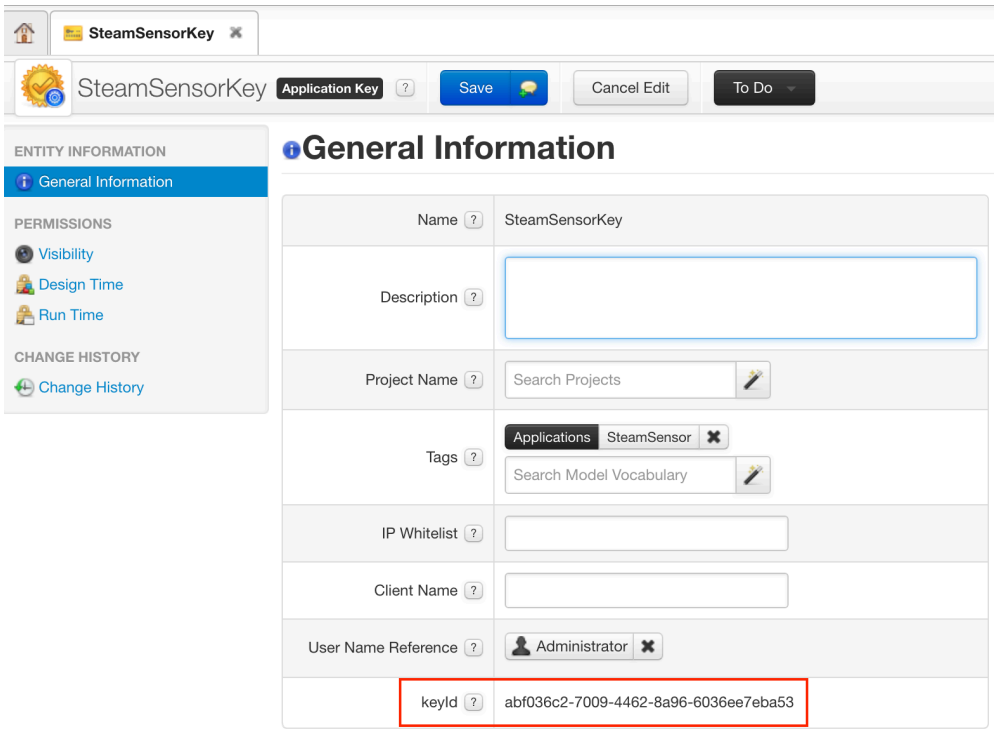
Once you are done, you should be able to see some newly created entities, e.g, SteamSensor etc.



4.3 Configure and Run the SteamSensor Example Application

From your host machine, launch a browser and go to https://developer.thingworx.com/profile/hosting. Apply for your ThingWorx trial server instance if you haven't. If you already have an instance provisioned, click "Launch" to launch the instance.

1. From ThingWorx portal, in the left pane, select Security->Application Keys-> SteamSensorKey. Click on SteamSensorKey Application Key and make a note of the keyId, keyId is the application key to be used in the local application.



2. We will start looking in the *<sdk>/examples/SteamSensor/src* directory.

```
[root@artik bin]# cd ../examples/SteamSensor/src
[root@artik src]# ls
SteamThing.c  SteamThing.h  gps-route.h  main.c
```

The *main.c* file contains the program to create and update properties on the ThingWorx server.

You can view the file by typing the command "vi main.c".

```
#define TW_HOST “localhost”
#define TW_APP_KEY “eld78abf-cfd2-..”
```

Replace localhost with the hostname of your server (for the online trial instance, it is something like pp-180207193531.devportal.ptc.io)..

Replace AppKey by using the application key you obtained from step above.

3. By default, the sample application enforces command line parameters parsing. This requires you to enter hostname, port, application key etc. parameters every time when you start the application. We can disable it by comment out the lines below in *main.c* main() function.

```
/* Parse command line parameters */
/* if (argc == 1){
     printf("Syntax: %s <hostname> <port> <appKey> <thingname>\n",argv[0]);
     exit(0);
} */
```

4. Compile the SteamSensor example.

To incorporate our code changes, recompile the program. Go back to the <sdk>/bin directory to run make.

```
[root@artik src]# cd ../../../bin
[root@artik bin]# make
```

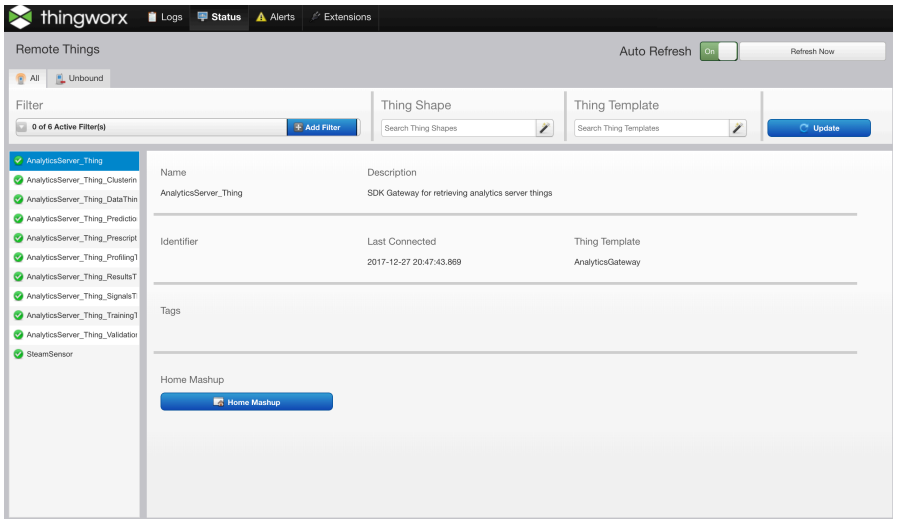
Now that this is complete, the finished program can be found in the *./bin/examples/SteamSensor* directory

```
[root@artik bin]# ./examples/SteamSensor/SteamSensor
[FORCE] 2017-12-28 06:13:28,598: Starting up...
[WARN ] 2017-12-28 06:13:28,599: SDK Version: 2.0.2
[WARN ] 2017-12-28 06:13:28,599: TLS Library: OpenSSL
[WARN ] 2017-12-28 06:13:28,599: TLS Library Version: 1.0.2l
[DEBUG] 2017-12-28 06:13:28,599: twWs_Create: Initializing Websocket Client
for PP-1712280444BL.devportal.ptc.io:443//Thingworx/WS
[DEBUG] 2017-12-28 06:13:28,599: twTlsClient_Create: Initializing TLS Client
[DEBUG] 2017-12-28 06:13:28,622: twApi_Initialize: Websocket Established
after 0 tries
[DEBUG] 2017-12-28 06:13:28,623: subscribedPropsMgr_Initialize: Initializing
subscribed properties manager
[TRACE] 2017-12-28 06:13:28,624: twFileManager_AddVirtualDir: Adding
__staging__ to vdir list
[TRACE] 2017-12-28 06:13:28,625: twFileManager_AddVirtualDir: Adding logs to
vdir list
[TRACE] 2017-12-28 06:13:28,625: twSubscribedPropsMgr_SetPropertyVTQ:
Property TemperatureLimit pushType is ALWAYS
```

```
[TRACE] 2017-12-28 06:13:28,625: twSubscribedPropsMgr_QueueValueForSending:
Updating saved property value. Property: TemperatureLimit. Folding is OFF
[TRACE] 2017-12-28 06:13:28,625: twSubscribedPropsMgr_QueueValueForSending:
Creating property list for entity SteamSensor
[TRACE] 2017-12-28 06:13:28,625: twSubscribedPropsMgr_QueueValueForSending:
Adding property TemperatureLimit to entity SteamSensor list. Source: RAM
[TRACE] 2017-12-28 06:13:28,625: twSubscribedProps_Write: Property
TemperatureLimit being queued to be sent to server
[TRACE] 2017-12-28 06:13:28,625: twSubscribedPropsMgr_SetPropertyVTQ:
Property Logfile pushType is ALWAYS
[TRACE] 2017-12-28 06:13:28,625: twSubscribedPropsMgr_QueueValueForSending:
Updating saved property value. Property: Logfile. Folding is OFF
[TRACE] 2017-12-28 06:13:28,626: twSubscribedPropsMgr_QueueValueForSending:
Adding property Logfile to entity SteamSensor list. Source: RAM
[TRACE] 2017-12-28 06:13:28,626: twSubscribedProps_Write: Property Logfile
being queued to be sent to server
[DEBUG] 2017-12-28 06:13:28,626: added SteamSensor to boundList
[DEBUG] 2017-12-28 06:13:28,626: twApi_BindThings: not currently connected,
only binding things to api
[DEBUG] 2017-12-28 06:13:28,626: twMessage_Delete: Deleting BIND Message: 1
[TRACE] 2017-12-28 06:13:28,626: twApi_Connect: Delaying 1 milliseconds
before connecting
[DEBUG] 2017-12-28 06:13:28,627: twTlsClient_Reconnect: Re-establishing SSL
context
[DEBUG] 2017-12-28 06:13:28,632: twTlsClient_Connect: Connecting to server
[DEBUG] 2017-12-28 06:13:28,932: twTlsClient_Connect: TLS connection
established
[TRACE] 2017-12-28 06:13:28,933: twWs_Connect: Connected to PP-
1712280444BL.devportal.ptc.io:443
[TRACE] 2017-12-28 06:13:28,933: twWs_Connect: Sent request:
GET /Thingworx/WS HTTP/1.1
User-Agent: ThingWorx C SDK
Upgrade: websocket
Connection: Upgrade
Host: PP-1712280444BL.devportal.ptc.io
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: AcSnChI/N/h07tk7IWzpnQ==
Max-Frame-Size: 8192
appKey: 9b5b577e-2379-4ca1-ba0c-d5cd4c8da7ad
```

4.4 View the steamed data from ThingWorx portal

From your ThingWorx Composer, click on SteamSensor Thing and select the **Properties** tab. Here you should see the *isConnected* property set to **true**, indicating that the connection is successful.
SteamSensor is bound for the duration of the application run time. To verify, go the **Monitoring** dropdown at the top right of the Composer, then select **Remote Things**. The monitoring page of Remote Things should show a green check with information about *SteamSenso*



Also, From *SteamSensor* **Properties** tab, you can see the streamed properties data, including temperature, pressure etc. telemetry data from your ARTIK530.