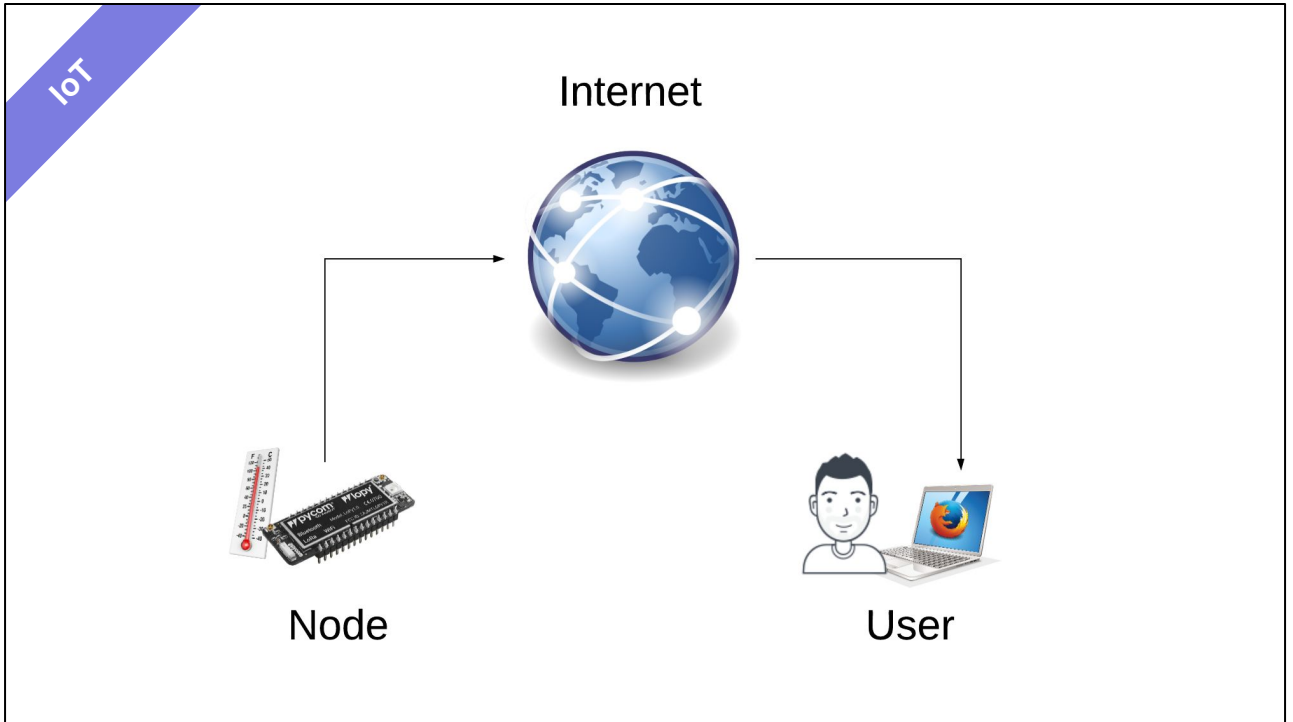




<https://github.com/joarolai/iothack>

Powered by Wireless Trondheim

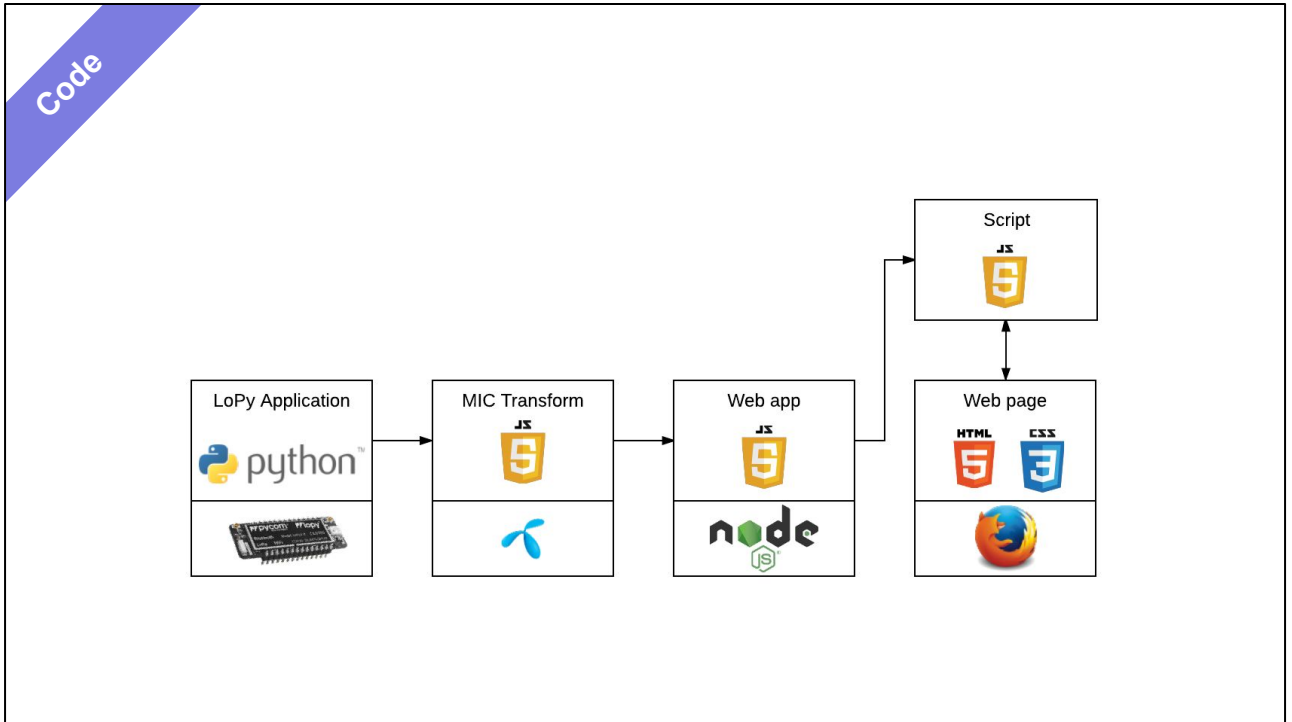
Version 1.1:



In this presentation we are looking at a simple application that is sending sensor data (temperature) via the internet to a web application where the users can observe the temperature change in real time (more or less) via their browsers.

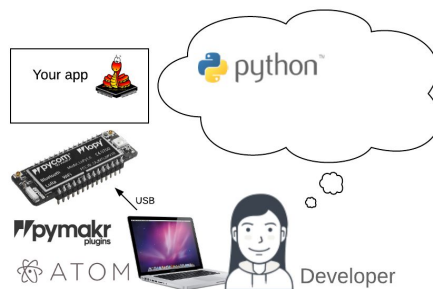
Node = Sensor node operating on a network, also called a mote

MIC = Managed IoT Cloud



A typical IoT application may include several components that each need custom code developed for it. In our example we have the following:

- Code for the LoPy written in Python
- A transform script written in Javascript that shall run in the MIC
- A node.js application written in Javascript that shall run on your web server
- A user interface in the form of a webpage written in HTML, CSS and Javascript



A sensor node usually contain:

- A sensor
- A microcontroller with networking capability
- A microcontroller application

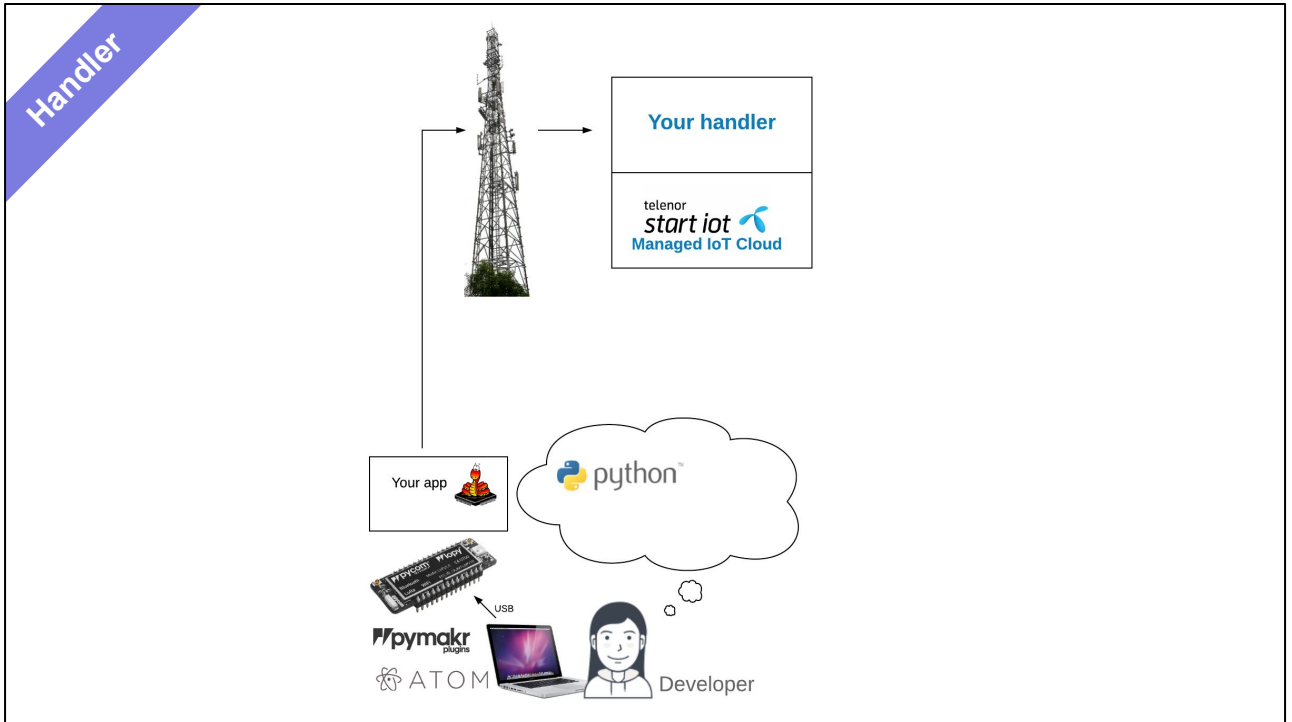
For our scenario the LoPy is our microcontroller, and our application is written in a variant of Python called Micropython.

Our application can be created using the Atom editor, and the LoPy can be programmed directly from Atom, via USB, after installing the Pymakr plugin.

<https://atom.io/>
<https://docs.pycom.io/chapter/gettingstarted/installation/pymakr.html>

You should also consider going through the complete getting started guide from Pycom: <https://docs.pycom.io/chapter/gettingstarted/>

You are not required to install python on your computer since micropython runs inside the LoPy.



After you have your development environment up and running, the next step is to try to make a real IoT application that can send data to the Telenor MIC.

Have a look at this tutorial from Telenor:

<https://startiot.telenor.com/learning/quick-start-guide/> , however, ignore the stuff in section: "Access Data Directly From Device", we will look at that in the next slides.

Device EUI

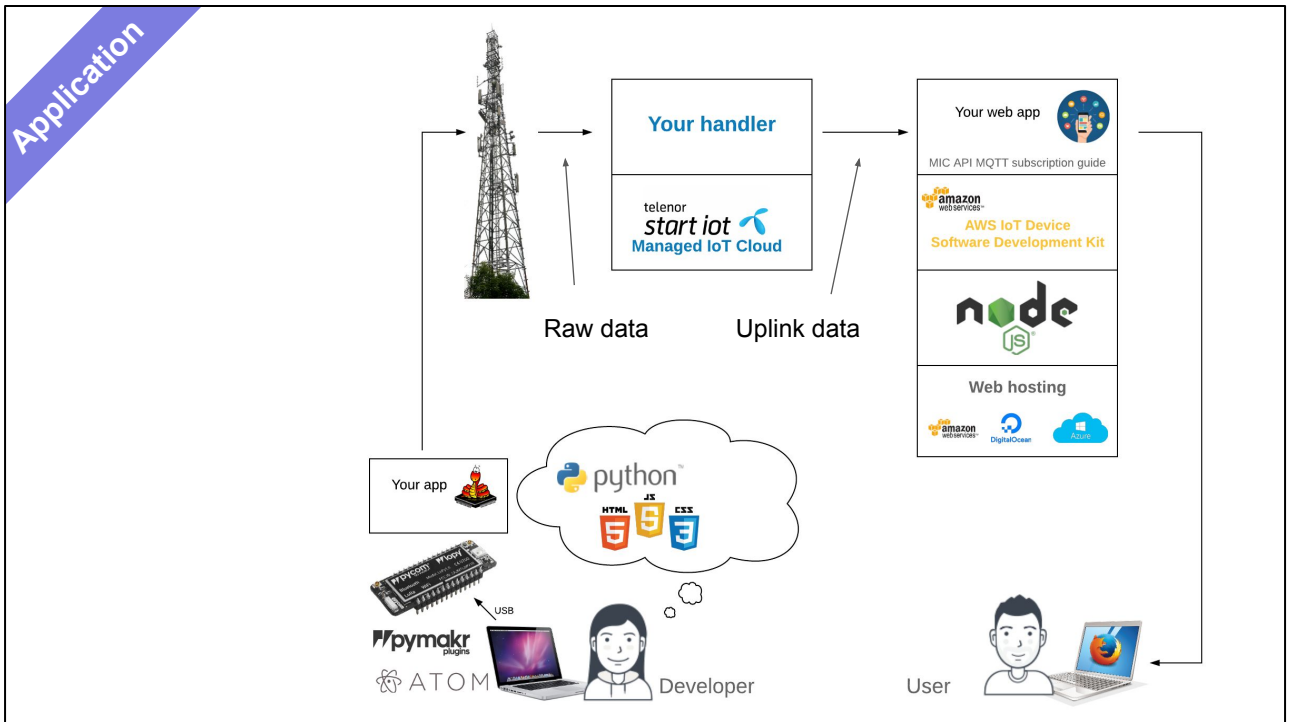
```
import network
import binascii
binascii.hexlify(network.LoRa().mac())
```

Use REPL for this

Before you can add the File Widget to your Thing in MIC, you must first update the Thing and specify the Device EUI of your LoPy.

Every individual LoRa module, including the ones from LoPy, has a unique EUI (MAC address).

After you have created the File widget, and downloaded the files, you should be able to send data to “Your Handler” in MIC.



When you have been able to send data to the MIC from your LoPy, the next step is to present the data in a web application that your users can access.

The following tutorial is explaining many of the steps that are required to accomplish this : <https://startiot.telenor.com/learning/managed-iot-cloud-api-mqtt-subscription/>

The code shown in the next slide, and the transform shown in the slide after that is needed to get the application in the tutorial to work.

If you are hosting the node.js server on your own computer then you do not need to change anything in the **index.html** file

If you want to install the node.js server on a hosting service like Digital Ocean, you must change line 34 in **index.html** to match the IP/domain of your server instead of **localhost**

LoPy code

```
from startiot import Startiot
import pycom
import time

pycom.heartbeat(False)      # disable the blue blinking
iot = Startiot()

print("Connecting...")
pycom.rgbled(0x0F0000)      # Red light when not connected
iot.connect()
pycom.rgbled(0x00000F)      # Blue light when connected

count = 0


while True:
    print("Send data...", count)
    data = "TEMP,%s" % (count)
    count = count + 1
    iot.send(data)
    time.sleep(60)
```

You can change the default code generated by MIC and use the one above instead.

Transform

Name is predefined in MIC.

This is the raw data sent from LoPy



```
var variables = payload.toString('ascii').split(',');  
return {  
  'temperature' : variables[1]  
};
```

Transform

Raw data (string)

TEMP,49

Uplink data (data structure)

```
{
  state: {
    reported: {
      txn: {
        connection_status: 2,
        cellular: {
          rssi: 16
        }
      },
      lsnr: -0.5,
      lating: '63.4184,10.4002',
      temperature: 49
    }
  },
  preventMessageRepublish: true
}
```

```
var awsIot = require('aws-iot-device-sdk');
var io = require('socket.io')(3000);

var thingName = '00000901'; // Replace with your own thing name

var device = awsIot.device({
  keyPath: './certs/privkey.pem',
  certPath: './certs/cert.pem',
  caPath: './certs/ca.pem',
  clientId: thingName,
  host: 'a31ovqfkmg1ev8.iot.eu-west-1.amazonaws.com'
});

device.on('connect', function() {
  console.log('Client connected');
  device.subscribe('$aws/things/' + thingName + '/shadow/update');
});

device.on('message', function(topic, payload) {
  console.log('Message: ', topic, payload.toString());

  // Broadcast the message to any connected socket clients
  io.emit('broadcast', {topic, message: payload.toString()});
});
```

```
<html>
  <head>
    <script src="https://code.jquery.com/jquery-1.12.0.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.0.3/socket.io.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.6.0/Chart.min.js"></script>
  </head>
  <body>
    <canvas id="output" width="600" height="250"></canvas>

    <script> // Include script from Script slide
    </script>
  </body>
</html>
```

Script

```
// Create the Chartjs element
var ctx = document.getElementById('output').getContext('2d');

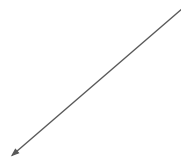
var myChart = new Chart(ctx, {
  type: 'line',
  data: {
    labels: [],
    datasets: [{
      label: 'Temperature',
      data: []
    }]
  }
});

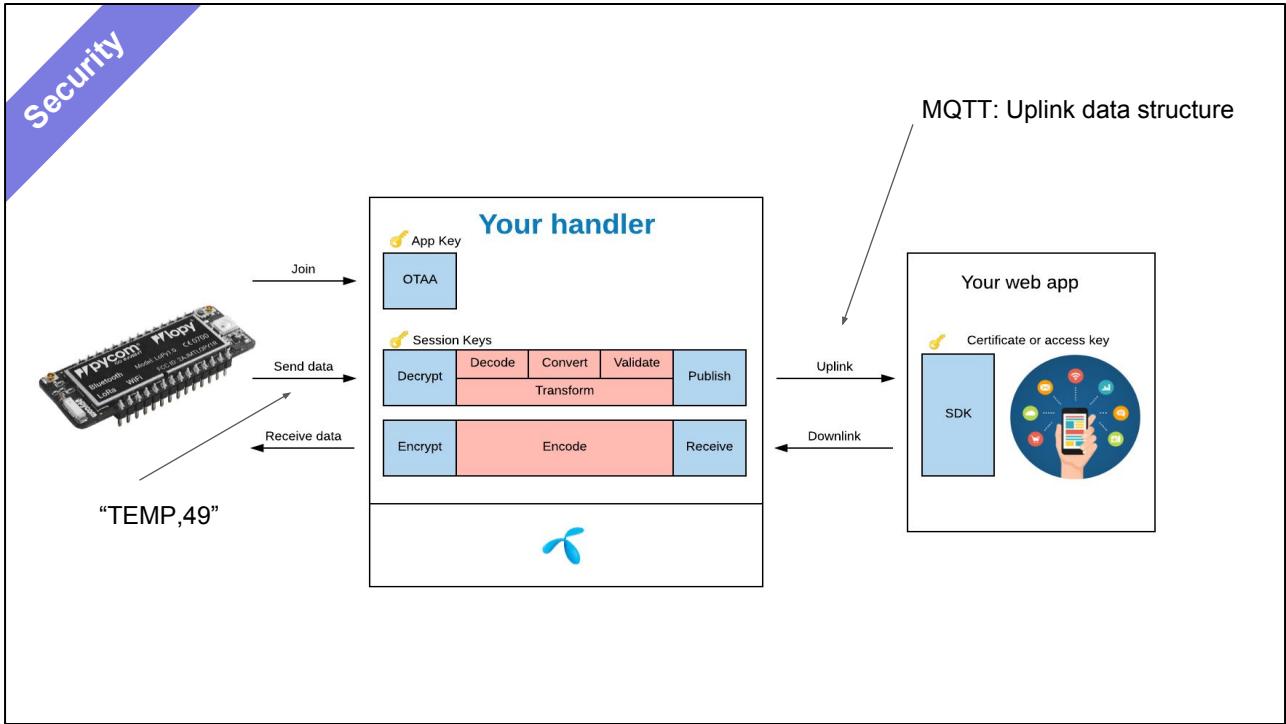
// Function to dynamically add data to the chart and update it
function addData(chart, label, data) {
  chart.data.labels.push(label);
  chart.data.datasets.forEach((dataset) => {
    dataset.data.push(data);
  });
  chart.update();
}

// Init Socket.io and add data to chart when broadcasted
var socket = io('http://localhost:3000');

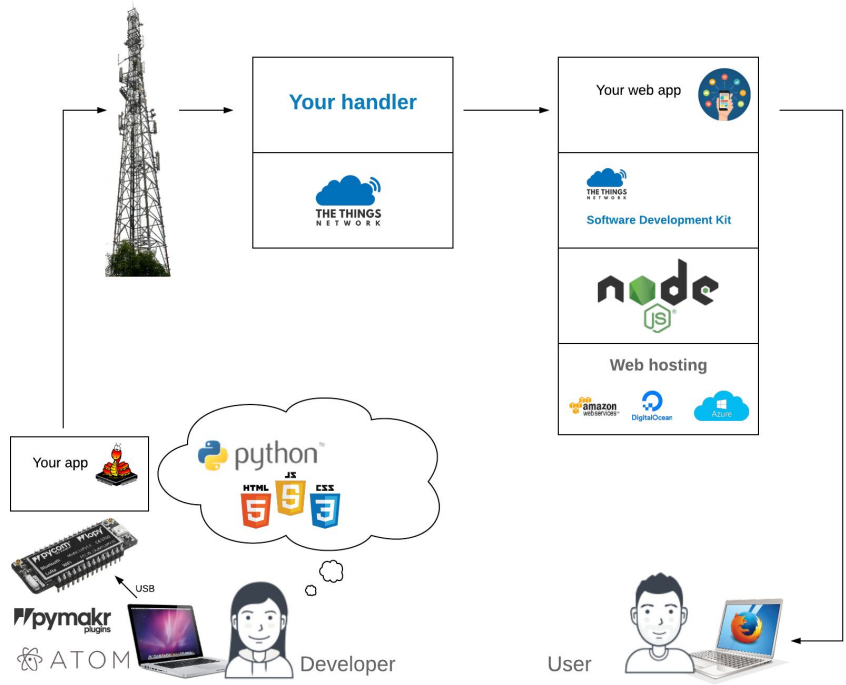
socket.on('broadcast', function(data) {
  var payload = JSON.parse(data.message);
  addData(myChart, new Date(), payload.state.reported.temperature);
});
```

MIC data structure





TTN



MIC data structure

```
{
  state: {
    reported: {
      tcxn: {
        connection_status: 2,
        cellular: {
          rssi: 16
        }
      },
      lsnr: -0.5,
      latlng: '63.4184,10.4002',
      temperature: 49
    }
  },
  preventMessageRepublish: true
}
```

TTN data structure

```
{
  app_id: 'kakemonster',
  dev_id: 'malopy',
  hardware_serial: '70B3D5499A4CE82A',
  port: 2,
  counter: 5892,
  payload_raw: <Buffer 54 45 4d 50 2c 35 38 39 31>,
  payload_fields: {
    temperature: '49'
  },
  metadata: {
    time: '2017-10-10T11:34:50.160318598Z',
    frequency: 867.9,
    modulation: 'LORA',
    data_rate: 'SF7BW125',
    coding_rate: '4/5',
    gateways: [ [Object] ]
  }
}
```

TTN gateways

```
[{
  gtw_id: 'trt-samf-loragw01',
  gtw_trusted: true,
  timestamp: 3869682171,
  time: '2017-10-10T11:37:47Z',
  channel: 4,
  rssi: -118,
  snr: -9.25,
  rf_chain: 0,
  latitude: 63.422485,
  longitude: 10.395755,
  altitude: 20
}, {
  gtw_id: 'eui-008000000000bc6c',
  timestamp: 4068030371,
  time: '2017-10-10T11:33:13.36681Z',
  channel: 4,
  rssi: -115,
  snr: -4.2,
  rf_chain: 0,
  latitude: 63.42883,
  longitude: 10.3857,
  altitude: 21
}]
```