Lab1 :IoT IT Infrastructure

Assoc. Prof. Ts Dr. Ahmad Shukri Mohd Noor

LAB 1: IoT IT INFRASTRUCTURE

Objective:
In this lab will guide you through understanding functions of some important IT Infrastructure and services to support IoT Application.

Item 1: Structured Cabling

Calculate bill of quantity for passive network equipment for the given network design.

Item 2: IP V4 Address

Calculate the IP addresses for the given network design:

Item 3: Deploy NodeRed IoT Gateway using docker virtualization image available on the Docker Hub.

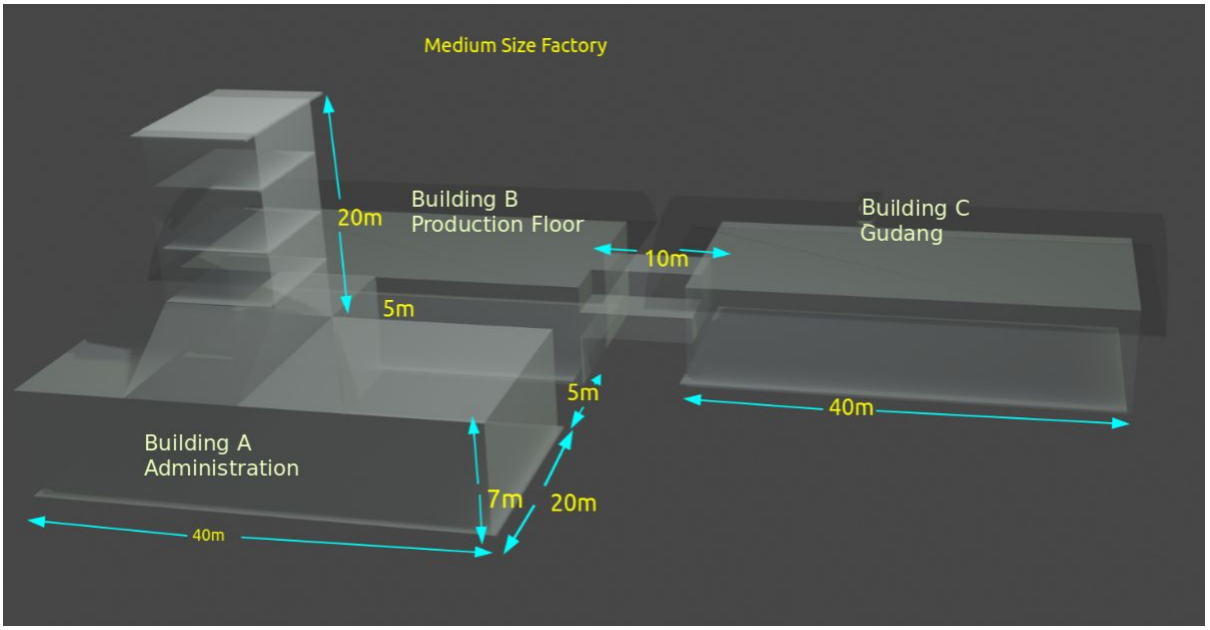
Item 4: Deploy Grafana - InfluxDB database using docker virtualization image available on the Docker Hub.

I**tem 5**: Send data from NodeRed IoT gateway to InfluxDB

Item 6: Visualize data using Grafana server

Item 1:

Structured cabling project for a medium size factory.



Estimated new building dimensions that require structured cabling to be designed and installed.

LAB 1: IoT IT INFRASTRUCTURE

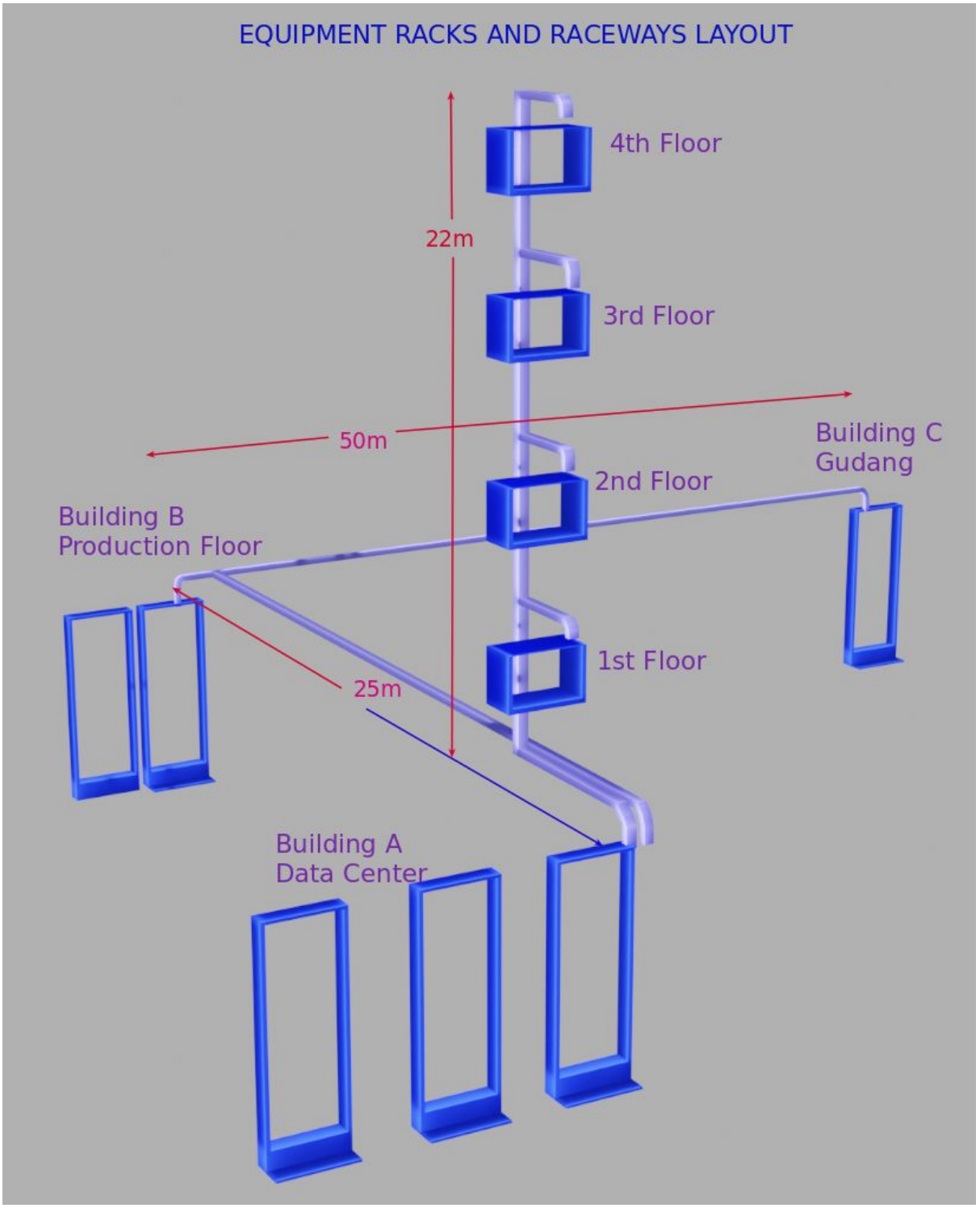
IT Services Requirements by Building Locations

No	Location	Requirements	Qty
1	Administration - Gnd Floor	Work Area Access Point	150 6
2	Administration - 1st Floor	Work Area Access Point	40 2
3	Administration - 2nd Floor	Work Area Access Point	20 2
4	Administration - 3rd Floor	Work Area Access Point	20 2
5	Administration - 4th Floor	Work Area Access Point	20 2
6	Administration - Data Center	Rackmount servers Core Switches Access Switches Internet Routers UPS CCTV Controller	4 2 5 1 4 1
7	Production Floor	Data outlet Access Point	300 15
8	Gudang	Data outlet Access Point	40 4

Overall WireLess Requirements:

1. Bring Your Own Devices (BYOD) 800
2. Notebooks 200
3. Wireless IoT Devices 500

Below is the intended location for equipment racks and the data raceways for the entire building.

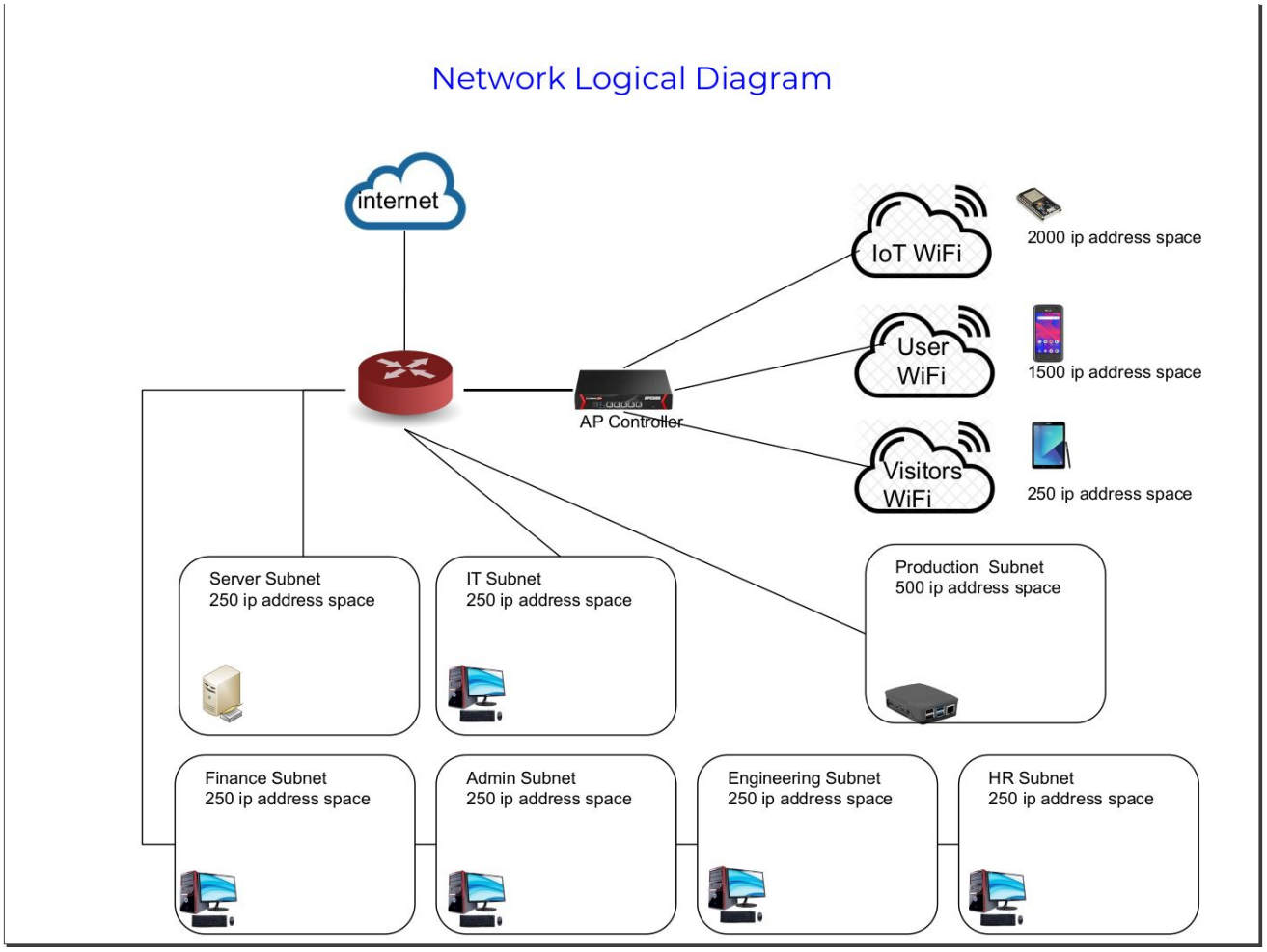


Dat

With the above information please calculate the bill of material of the components needed to complete the structure cabling.

LAB 1: IoT IT INFRASTRUCTURE

Medium Size LAN LogicalDiagram for A Factory



Item 2: Calculate the IPv4 addresses to be used in DHCP scopes for all the subnets shown.

1. Network Number
2. Starting IP address
3. Ending IP address
4. Subnet Mask
5. Gateway IP address

Note:

Please use Class B for IoT subnet

Please use Class A for all office and Production subnets

Please use Class C for Visitors subnet

Item 3: Deploying NodeRED IoT Gateway using Docker Container.

Create a directory to place docker-compose.yml and setting.js files.

Use the following docker-compose.yml

```
version: '3.1'
services:
    nodered:
        image: nodered/node-red-docker
        container_name: noderedsecure
        volumes:
            -
"./settings.js:/usr/src/node-red/node_modules/node-red/settings.js
"
        ports:
            - "1880:1880"
            - "1883:1883"
```

Create “settings.js” file in the directory and enter the following content and save the file.

This setting file will make your node-red application secured by enabling admin password:

node admin password is “adminpwd”

```
 * Copyright JS Foundation and other contributors,
http://js.foundation
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the
License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
software
 * distributed under the License is distributed on an "AS IS"
BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 * See the License for the specific language governing permissions
and
 * limitations under the License.
 **/
```

```
// The `https` setting requires the `fs` module. Uncomment the
following
// to make it available:
//var fs = require("fs");

module.exports = {
    // the tcp port that the Node-RED web server is listening on
    uiPort: process.env.PORT || 1880,

    // By default, the Node-RED UI accepts connections on all IPv4
interfaces.
    // To listen on all IPv6 addresses, set uiHost to "::",
    // The following property can be used to listen on a specific
interface. For
    // example, the following would only allow connections from
the local machine.
    //uiHost: "127.0.0.1",

    // Retry time in milliseconds for MQTT connections
    mqttReconnectTime: 15000,

    // Retry time in milliseconds for Serial port connections
    serialReconnectTime: 15000,

    // Retry time in milliseconds for TCP socket connections
    //socketReconnectTime: 10000,

    // Timeout in milliseconds for TCP server socket connections
    //  defaults to no timeout
    //socketTimeout: 120000,

    // Maximum number of messages to wait in queue while
attempting to connect to TCP socket
    //  defaults to 1000
    //tcpMsgQueueSize: 2000,

    // Timeout in milliseconds for HTTP request connections
    //  defaults to 120 seconds
    //httpRequestTimeout: 120000,

    // The maximum length, in characters, of any message sent to
the debug sidebar tab
    debugMaxLength: 1000,

    // The maximum number of messages nodes will buffer internally
as part of their
    // operation. This applies across a range of nodes that
operate on message sequences.
    //  defaults to no limit. A value of 0 also means no limit is
applied.
```

```
    //nodeMessageBufferMaxLength: 0,

    // To disable the option for using local files for storing
keys and certificates in the TLS configuration
    //  node, set this to true
    //tlsConfigDisableLocalFiles: true,

    // Colourise the console output of the debug node
    //debugUseColors: true,

    // The file containing the flows. If not set, it defaults to
flows_<hostname>.json
    //flowFile: 'flows.json',

    // To enabled pretty-printing of the flow within the flow
file, set the following
    //  property to true:
    //flowFilePretty: true,

    // By default, credentials are encrypted in storage using a
generated key. To
    // specify your own secret, set the following property.
    // If you want to disable encryption of credentials, set this
property to false.
    // Note: once you set this property, do not change it - doing
so will prevent
    // node-red from being able to decrypt your existing
credentials and they will be
    // lost.
    //credentialSecret: "a-secret-key",

    // By default, all user data is stored in a directory called
`.node-red` under
    // the user's home directory. To use a different location, the
following
    // property can be used
    //userDir: '/home/nol/.node-red/',

    // Node-RED scans the `nodes` directory in the userDir to find
local node files.
    // The following property can be used to specify an additional
directory to scan.
    //nodesDir: '/home/nol/.node-red/nodes',

    // By default, the Node-RED UI is available at
http://localhost:1880/
    // The following property can be used to specify a different
root path.
    // If set to false, this is disabled.
    //httpAdminRoot: '/admin',
```

```
    // Some nodes, such as HTTP In, can be used to listen for
incoming http requests.
    // By default, these are served relative to '/'. The following
property
    // can be used to specifiy a different root path. If set to
false, this is
    // disabled.
    //httpNodeRoot: '/red-nodes',

    // The following property can be used in place of
'httpAdminRoot' and 'httpNodeRoot',
    // to apply the same root to both parts.
    //httpRoot: '/red',

    // When httpAdminRoot is used to move the UI to a different
root path, the
    // following property can be used to identify a directory of
static content
    // that should be served at http://localhost:1880/.
    //httpStatic: '/home/nol/node-red-static/',

    // The maximum size of HTTP request that will be accepted by
the runtime api.
    // Default: 5mb
    //apiMaxLength: '5mb',

    // If you installed the optional node-red-dashboard you can
set it's path
    // relative to httpRoot
    //ui: { path: "ui" },

    // Securing Node-RED
    // -----------------
    // To password protect the Node-RED editor and admin API, the
following
    // property can be used. See
http://nodered.org/docs/security.html for details.
    adminAuth: {
        type: "credentials",
        users: [{
            username: "admin",
            password:
"$2a$08$sd0ZuGsa1G6TyC.VTE7SCet5TMSISz0ZW0l/b4AhBudWThNhNS6VK",
            permissions: "*"
         }]
    },

    // To password protect the node-defined HTTP endpoints
(httpNodeRoot), or
```

```
    // the static content (httpStatic), the following properties
can be used.
    // The pass field is a bcrypt hash of the password.
    // See
http://nodered.org/docs/security.html#generating-the-password-hash
    //httpNodeAuth:
{user:"user",pass:"$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dNbM6tl8sJo
gENOMcxWV9DN."},
    //httpStaticAuth:
{user:"user",pass:"$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dNbM6tl8sJo
gENOMcxWV9DN."},

    // The following property can be used to enable HTTPS
    // See
http://nodejs.org/api/https.html#https_https_createserver_options_
requestlistener
    // for details on its contents.
    // See the comment at the top of this file on how to load the
`fs` module used by
    // this setting.
    //
    //https: {
    //    key: fs.readFileSync('privatekey.pem'),
    //    cert: fs.readFileSync('certificate.pem')
    //},

    // The following property can be used to cause insecure HTTP
connections to
    // be redirected to HTTPS.
    //requireHttps: true,

    // The following property can be used to disable the editor.
The admin API
    // is not affected by this option. To disable both the editor
and the admin
    // API, use either the httpRoot or httpAdminRoot properties
    //disableEditor: false,

    // The following property can be used to configure
cross-origin resource sharing
    // in the HTTP nodes.
    // See
https://github.com/troygoode/node-cors#configuration-options for
    // details on its contents. The following is a basic
permissive set of options:
    //httpNodeCors: {
    //    origin: "*",
    //    methods: "GET,PUT,POST,DELETE"
    //},
```

```
    // If you need to set an http proxy please set an environment
variable
    // called http_proxy (or HTTP_PROXY) outside of Node-RED in
the operating system.
    // For example - http_proxy=http://myproxy.com:8080
    // (Setting it here will have no effect)
    // You may also specify no_proxy (or NO_PROXY) to supply a
comma separated
    // list of domains to not proxy, eg -
no_proxy=.acme.co,.acme.co.uk

    // The following property can be used to add a custom
middleware function
    // in front of all http in nodes. This allows custom
authentication to be
    // applied to all http in nodes, or any other sort of common
request processing.
    //httpNodeMiddleware: function(req,res,next) {
    //    // Handle/reject the request, or pass it on to the http
in node by calling next();
    //    // Optionally skip our rawBodyParser by setting this to
true;
    //    //req.skipRawBodyParser = true;
    //    next();
    //},

    // The following property can be used to pass custom options
to the Express.js
    // server used by Node-RED. For a full list of available
options, refer
    // to http://expressjs.com/en/api.html#app.settings.table
    //httpServerOptions: { },

    // The following property can be used to verify websocket
connection attempts.
    // This allows, for example, the HTTP request headers to be
checked to ensure
    // they include valid authentication information.
    //webSocketNodeVerifyClient: function(info) {
    //    // 'info' has three properties:
    //    //   - origin : the value in the Origin header
    //    //   - req : the HTTP request
    //    //   - secure : true if req.connection.authorized or
req.connection.encrypted is set
    //    //
    //    // The function should return true if the connection
should be accepted, false otherwise.
    //    //
    //    // Alternatively, if this function is defined to accept
a second argument, callback,
```

```
    //    // it can be used to verify the client asynchronously.
    //    // The callback takes three arguments:
    //    //   - result : boolean, whether to accept the
connection or not
    //    //   - code : if result is false, the HTTP error status
to return
    //    //   - reason: if result is false, the HTTP reason
string to return
    //},

    // The following property can be used to seed Global Context
with predefined
    // values. This allows extra node modules to be made available
with the
    // Function node.
    // For example,
    //    functionGlobalContext: { os:require('os') }
    // can be accessed in a function block as:
    //    global.get("os")
    functionGlobalContext: {
        // os:require('os'),
        // jfive:require("johnny-five"),
        // j5board:require("johnny-five").Board({repl:false})
    },
    // `global.keys()` returns a list of all properties set in
global context.
    // This allows them to be displayed in the Context Sidebar
within the editor.
    // In some circumstances it is not desirable to expose them to
the editor. The
    // following property can be used to hide any property set in
`functionGlobalContext`
    // from being list by `global.keys()`.
    // By default, the property is set to false to avoid
accidental exposure of
    // their values. Setting this to true will cause the keys to
be listed.
    exportGlobalContextKeys: false,


    // Context Storage
    // The following property can be used to enable context
storage. The configuration
    // provided here will enable file-based context that flushes
to disk every 30 seconds.
    // Refer to the documentation for further options:
https://nodered.org/docs/api/context/
    //
    //contextStorage: {
    //    default: {
```

```
    //        module:"localfilesystem"
    //    },
    //},

    // The following property can be used to order the categories
in the editor
    // palette. If a node's category is not in the list, the
category will get
    // added to the end of the palette.
    // If not set, the following default order is used:
    //paletteCategories:
['subflows','flow','input','output','function','parser','social','
mobile','storage','analysis','advanced'],

    // Configure the logging output
    logging: {
        // Only console logging is currently supported
        console: {
            // Level of logging to be recorded. Options are:
            // fatal - only those errors which make the
application unusable should be recorded
            // error - record errors which are deemed fatal for a
particular request + fatal errors
            // warn - record problems which are non fatal + errors
+ fatal errors
            // info - record information about the general running
of the application + warn + error + fatal errors
            // debug - record information which is more verbose
than info + info + warn + error + fatal errors
            // trace - record very detailed logging + debug + info
+ warn + error + fatal errors
            // off - turn off all logging (doesn't affect metrics
or audit)
            level: "info",
            // Whether or not to include metric events in the log
output
            metrics: false,
            // Whether or not to include audit events in the log
output
            audit: false
        }
    },

    // Customising the editor
    editorTheme: {
        projects: {
            // To enable the Projects feature, set this value to
true
            enabled: false
        }
```

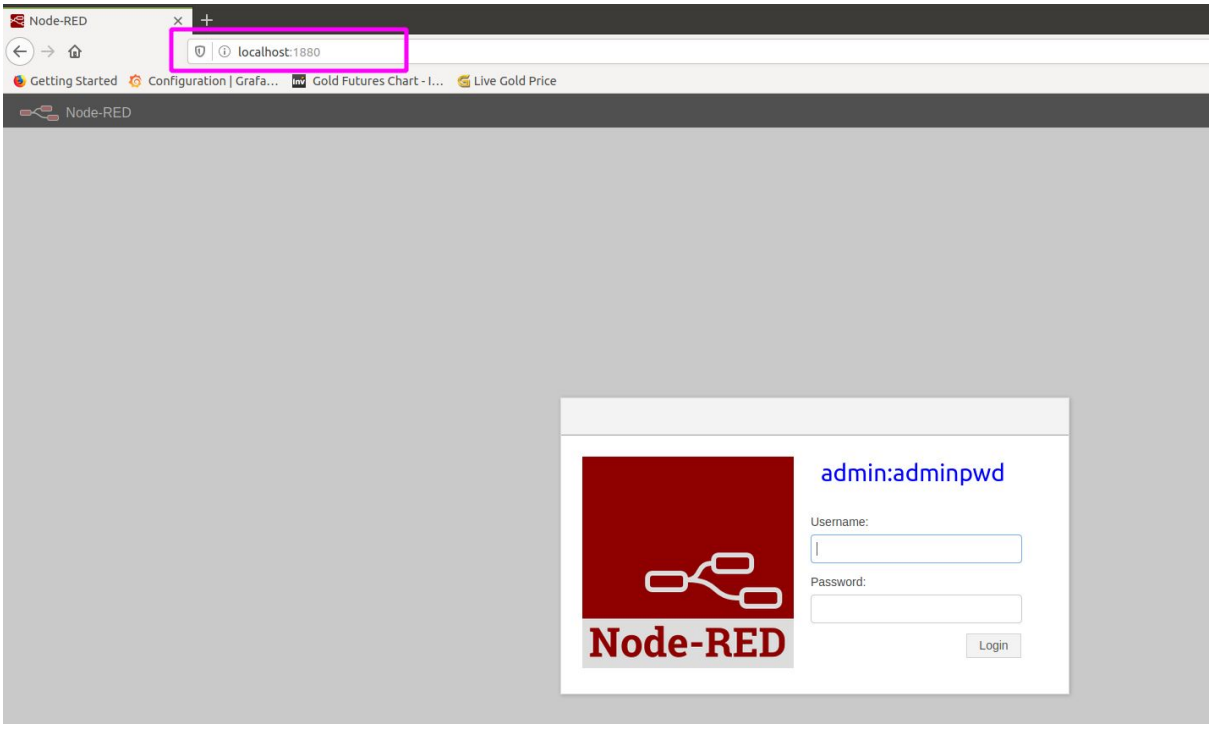
```
    }
}
```

Issue command to build and compose docker-nodered:

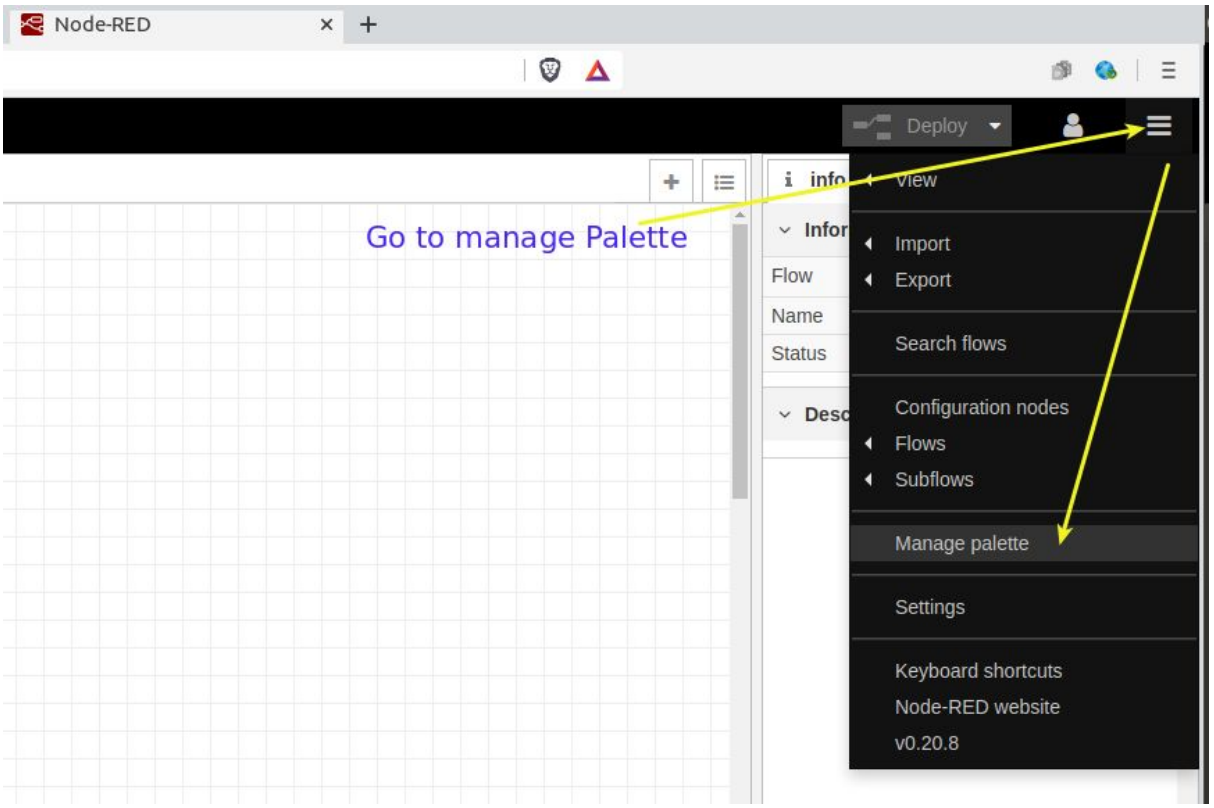
```
(base) fuzis@fsvivo:~/dockers/noderedockersecure$ sudo docker-compose up --build
ERROR: The Compose file './docker-compose.yml' is invalid because:
Unsupported config option for services.nodered: 'volume' (did you mean 'volumes'?)
(base) fuzis@fsvivo:~/dockers/noderedockersecure$ sudo docker-compose up --build
Creating network "noderedockersecure_default" with the default driver
Creating noderedsecure ... done
Attaching to noderedsecure
noderedsecure |
noderedsecure | > node-red-docker@1.0.0 start /usr/src/node-red
noderedsecure | > node $NODE_OPTIONS node_modules/node-red/red.js -v $FLOWS "--userDir" "/data"
noderedsecure |
noderedsecure | 2 Jul 05:45:13 - [info]
noderedsecure |
noderedsecure | Welcome to Node-RED
noderedsecure | ===================
noderedsecure |
noderedsecure | 2 Jul 05:45:13 - [info] Node-RED version: v0.20.8
noderedsecure | 2 Jul 05:45:13 - [info] Node.js  version: v8.16.1
noderedsecure | 2 Jul 05:45:13 - [info] Linux 5.3.0-59-generic x64 LE
noderedsecure | 2 Jul 05:45:14 - [info] Loading palette nodes
noderedsecure | 2 Jul 05:45:14 - [warn] rpi-gpio : Raspberry Pi specific node set inactive
noderedsecure | 2 Jul 05:45:14 - [warn] rpi-gpio : Cannot find Pi RPi.GPIO python library
noderedsecure | 2 Jul 05:45:14 - [info] Settings file  : /data/settings.js
noderedsecure | 2 Jul 05:45:14 - [info] Context store  : 'default' [module=memory]
noderedsecure | 2 Jul 05:45:14 - [info] User directory : /data
noderedsecure | 2 Jul 05:45:14 - [warn] Projects disabled : editorTheme.projects.enabled=false
noderedsecure | 2 Jul 05:45:14 - [info] Flows file     : /data/flows.json
noderedsecure | 2 Jul 05:45:14 - [info] Creating new flow file
noderedsecure | 2 Jul 05:45:14 - [warn]
noderedsecure |
noderedsecure | ---------------------------------------------------------------------
noderedsecure | Your flow credentials file is encrypted using a system-generated key.
noderedsecure |
noderedsecure | If the system-generated key is lost for any reason, your credentials
noderedsecure | file will not be recoverable, you will have to delete it and re-enter
noderedsecure | your credentials.
noderedsecure |
noderedsecure | You should set your own key using the 'credentialSecret' option in
```

```
noderedsecure | your settings file. Node-RED will then re-encrypt your credentials
noderedsecure | file using your chosen key the next time you deploy a change.
noderedsecure | ---------------------------------------------------------------------
noderedsecure |
noderedsecure | 2 Jul 05:45:14 - [info] Server now running at http://127.0.0.1:1880/
noderedsecure | 2 Jul 05:45:14 - [info] Starting flows
noderedsecure | 2 Jul 05:45:14 - [info] Started flows
```

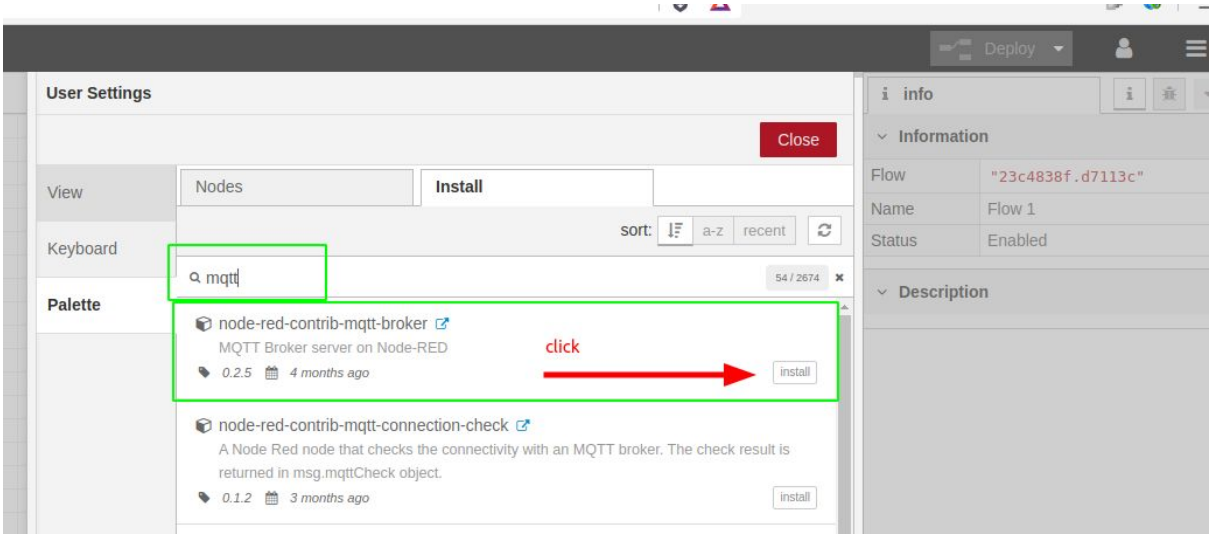
Open a browser and access to the url given:**http://127.0.0.1:1880**

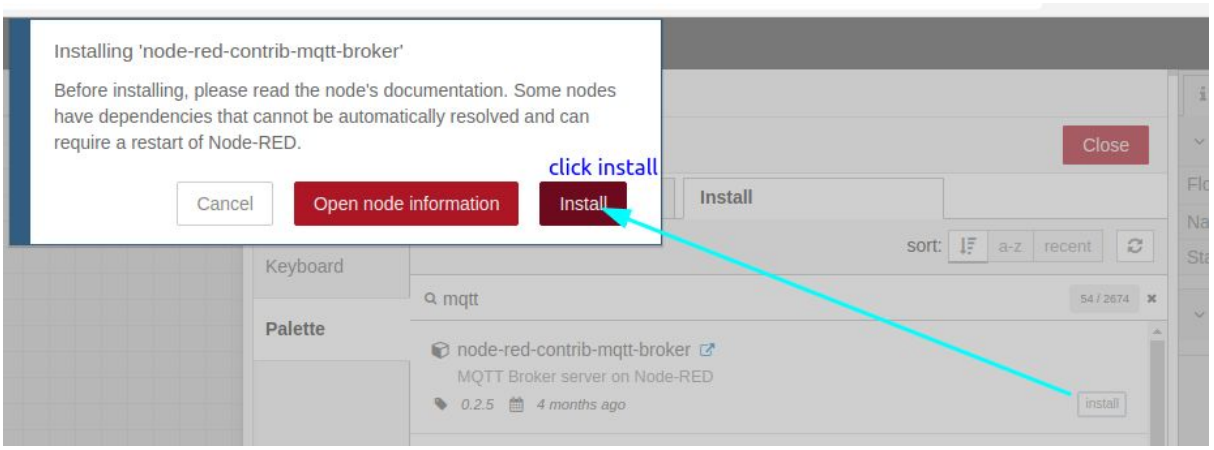


Once you login, go to the manage palette and Install MQTT broker as follows:

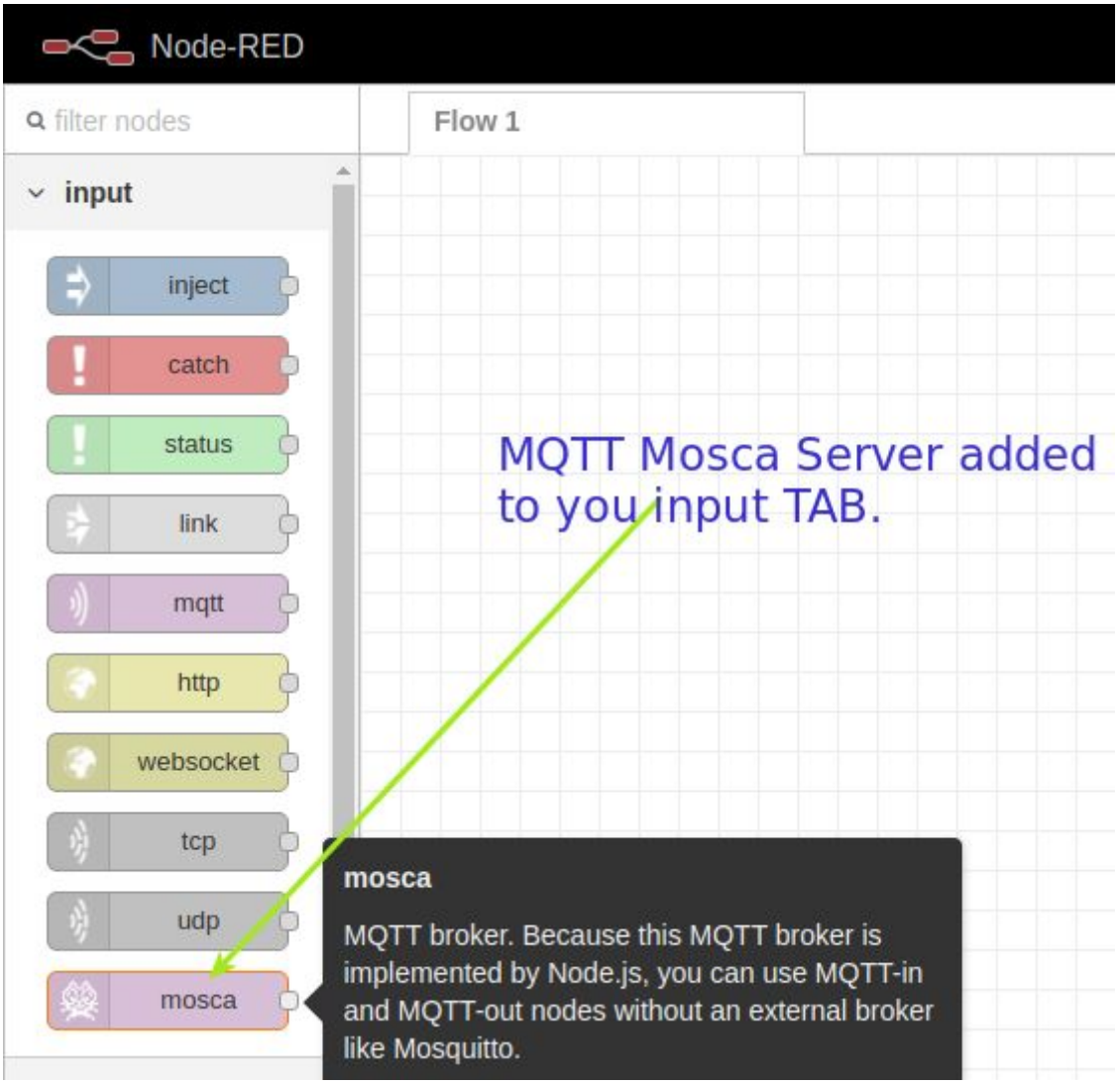


click tab install and go to search box key in “mqtt”





Once installation completed, the Installed MQTT server name MOSCA can be found in the input TAB as shown below:



Note: MQTT serve port can be accessed from localhost:1883

Item4: Spin up Grafana + InfluxDB using docker image:

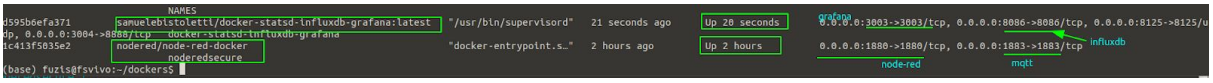
Use docker image available built by another user from docker-hub.

Run the image as follows:

```
docker pull samuelebistoletti/docker-statsd-influxdb-grafana
```

```
docker run --ulimit nofile=66000:66000 \
  -d \
  --name docker-statsd-influxdb-grafana \
  -p 3003:3003 \
  -p 3004:8888 \
  -p 8086:8086 \
  -p 8125:8125/udp \
  samuelebistoletti/docker-statsd-influxdb-grafana:latest
```

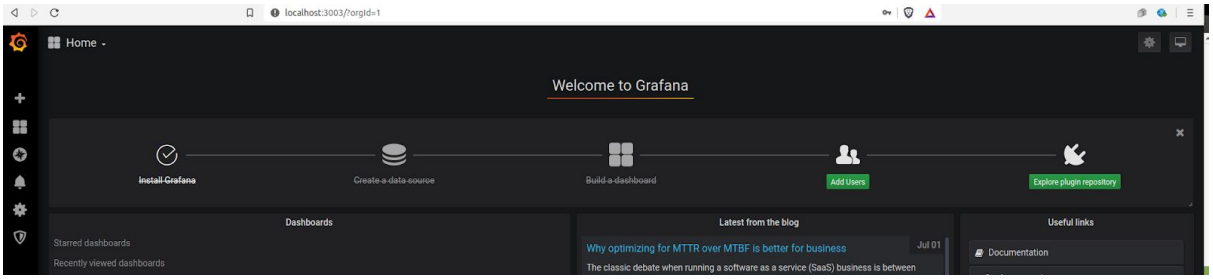
run docker ps to check the status.



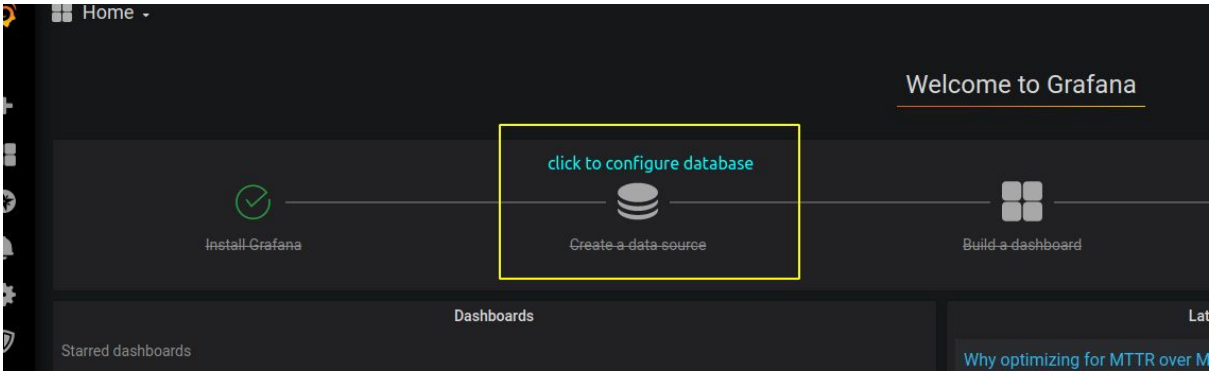
Now access to Grafana IoT visualization server:

http://localhost:3003

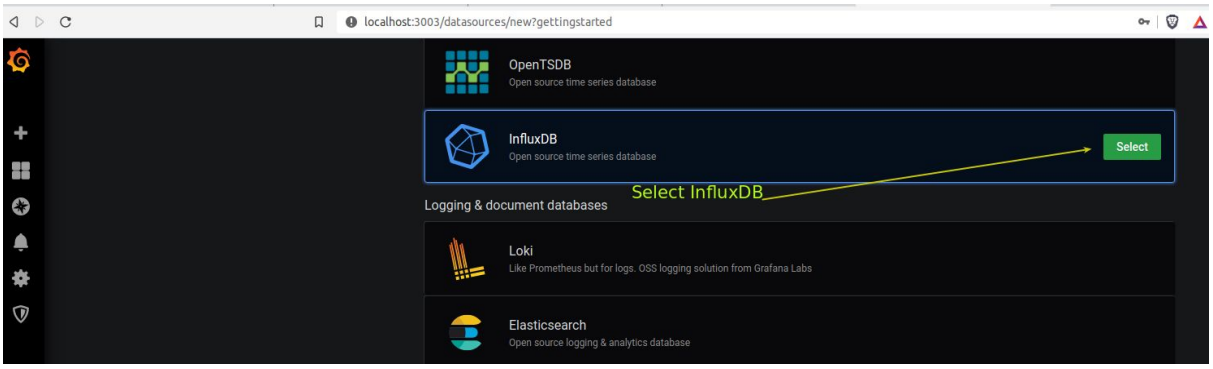
You should be able to see the Grafana Selcome Screen:

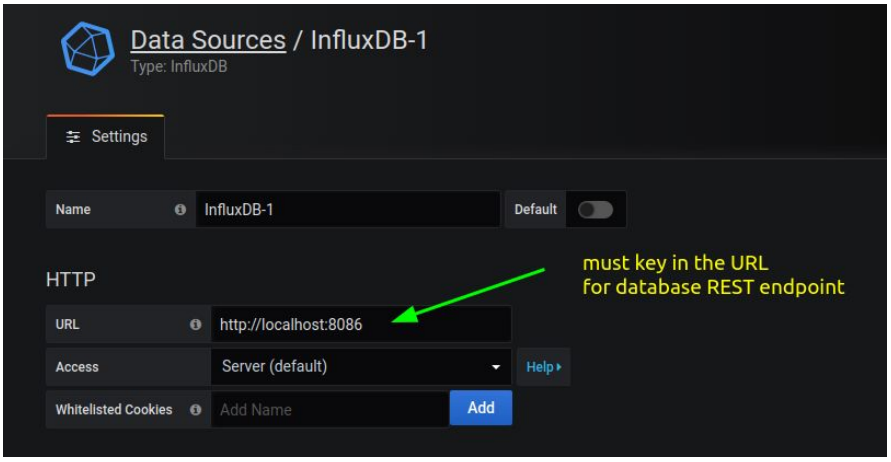


Now, configure the database for Grafana server.



Select InfluxDB

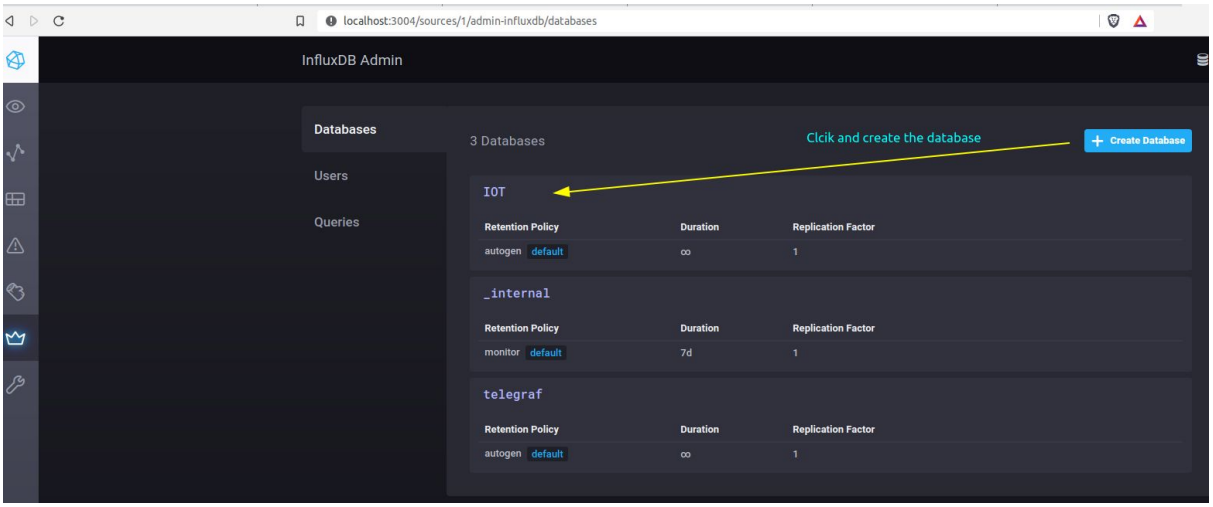




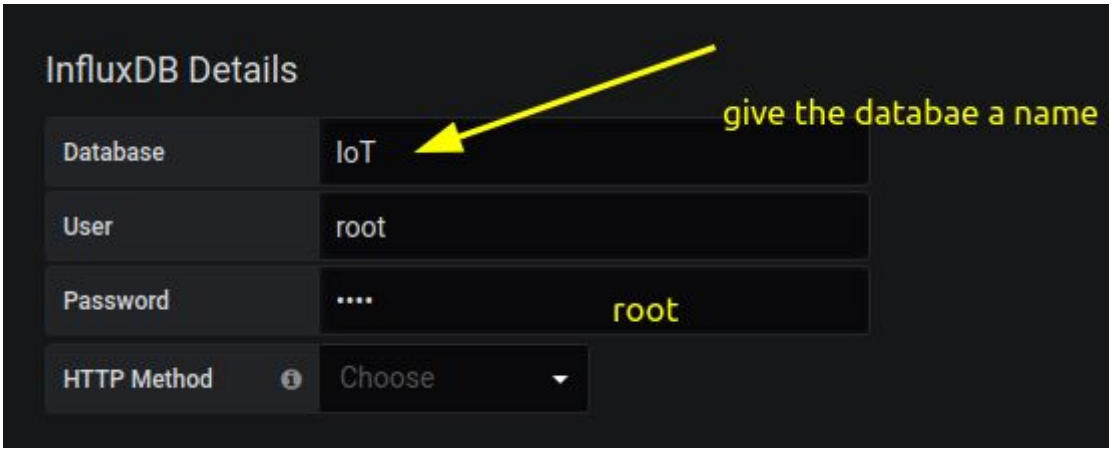
The database is not available yet on the InfluxDB server.

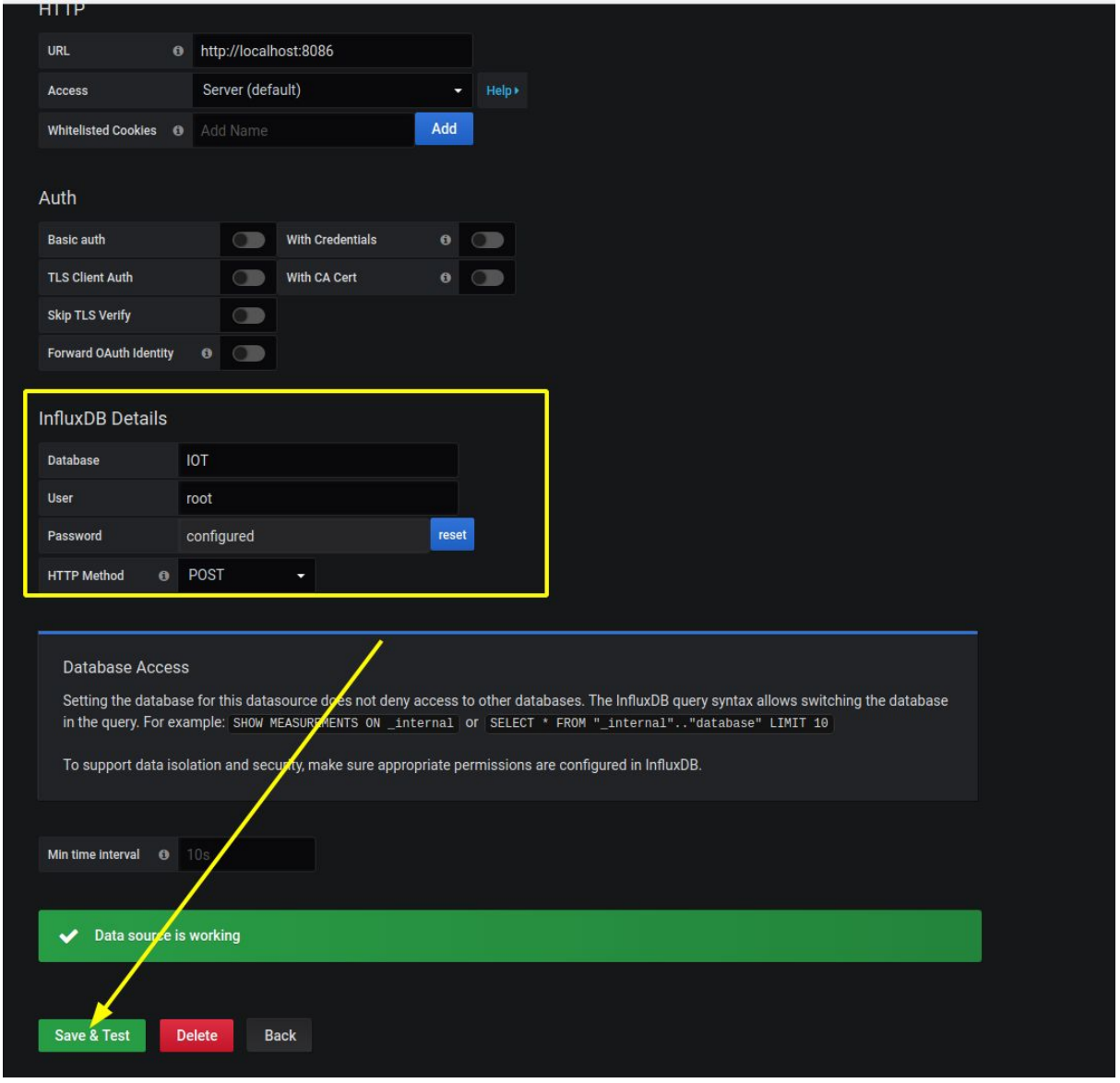
You need to create it first using InfluxDB admin page:

access to http://localhost:3004 using anthe browser tab:



No go back to your Grafana configuration page and continue database setup.





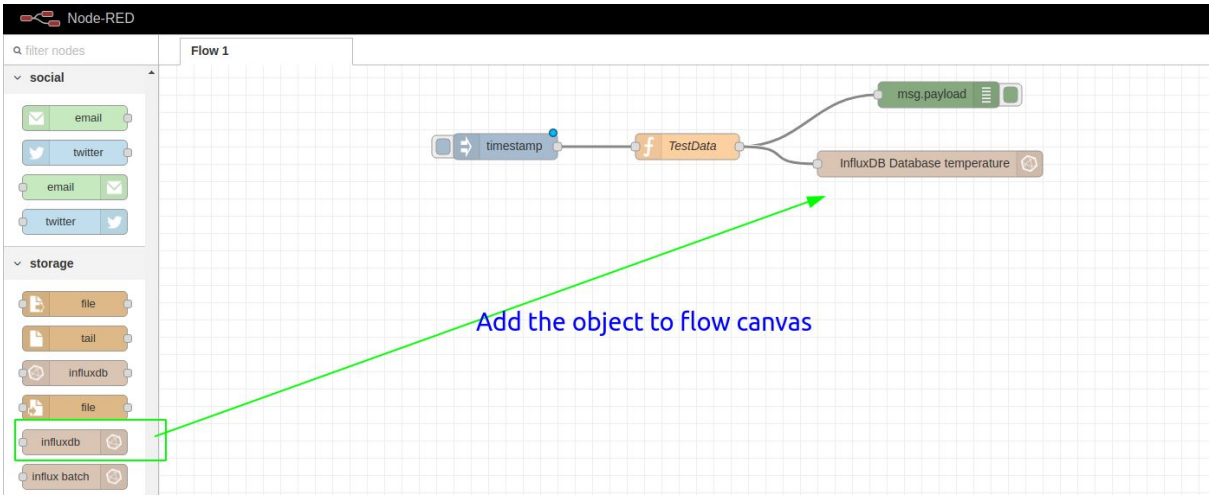
Save and test the database connection.

You should be able to see the green alert bar “Data source is working”

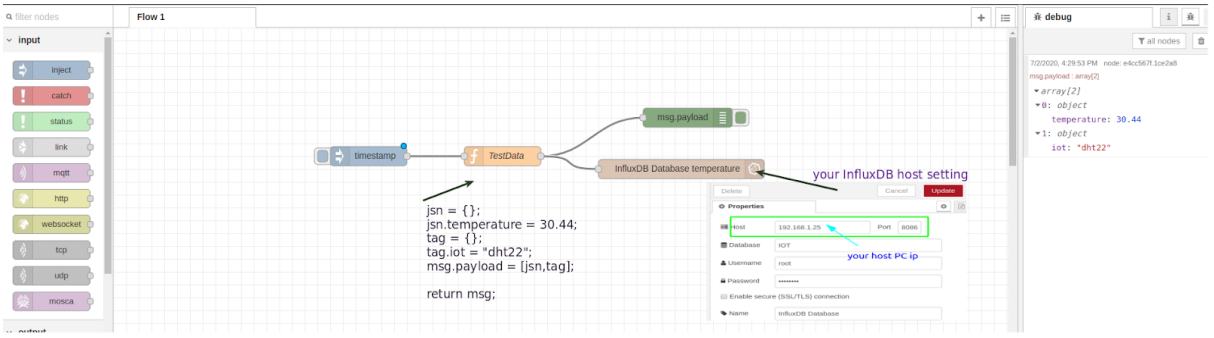
Item 5:

Now go to your NodeRED admin page and Install InfluxDB from Mage Palette.

Create a flow:

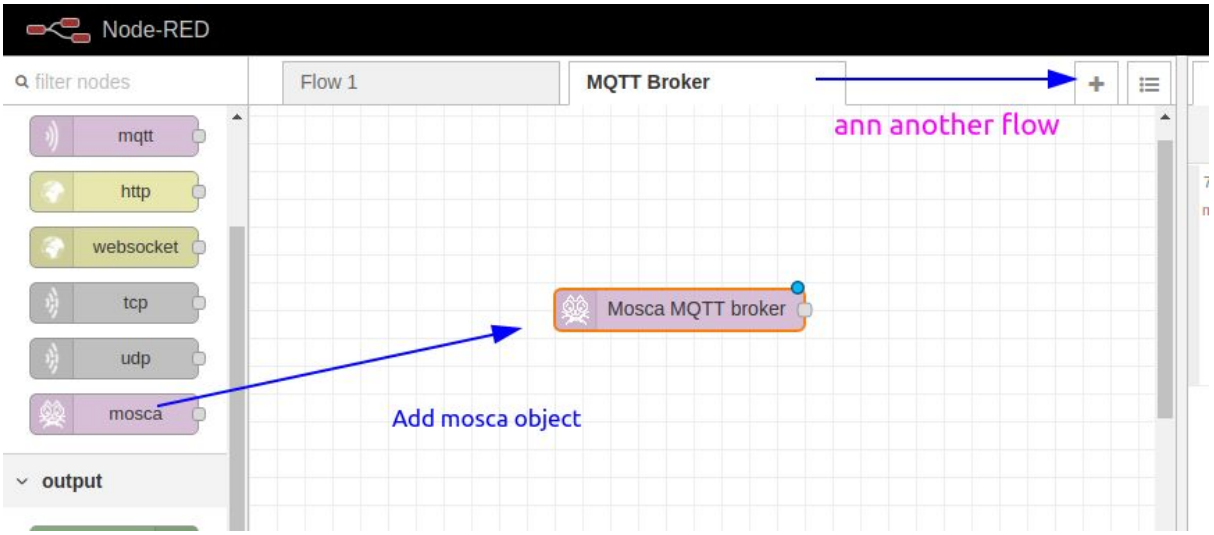


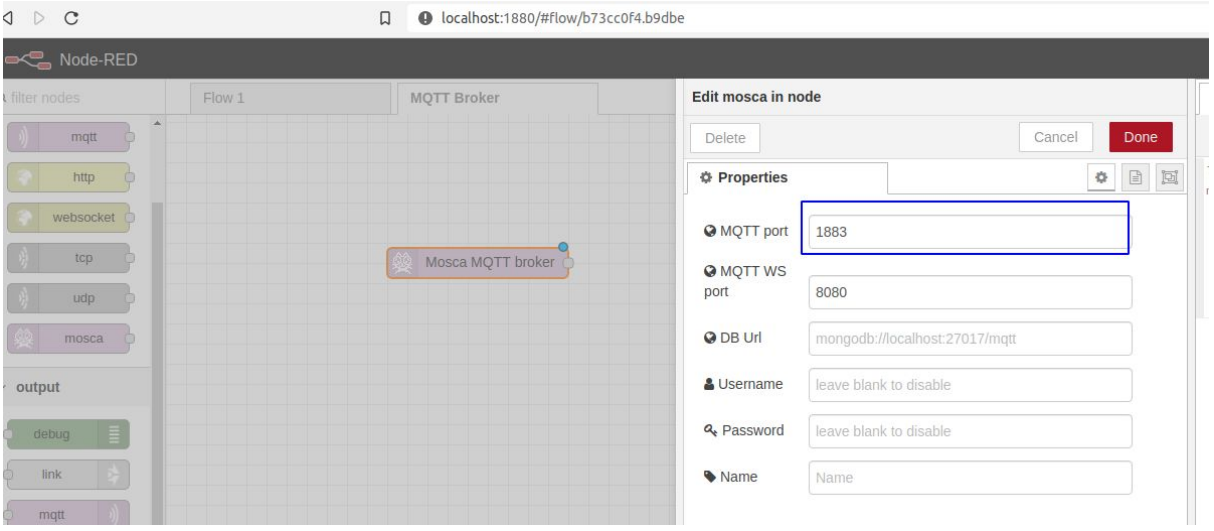
configure the database and add the function script to send test data.




```
jsn = {};
jsn.temperature = 30.44;
tag = {};
tag.iot = "dht22";
msg.payload = [jsn,tag];

return msg;
```

Now enable MQTT broker by adding another flow on node-red.

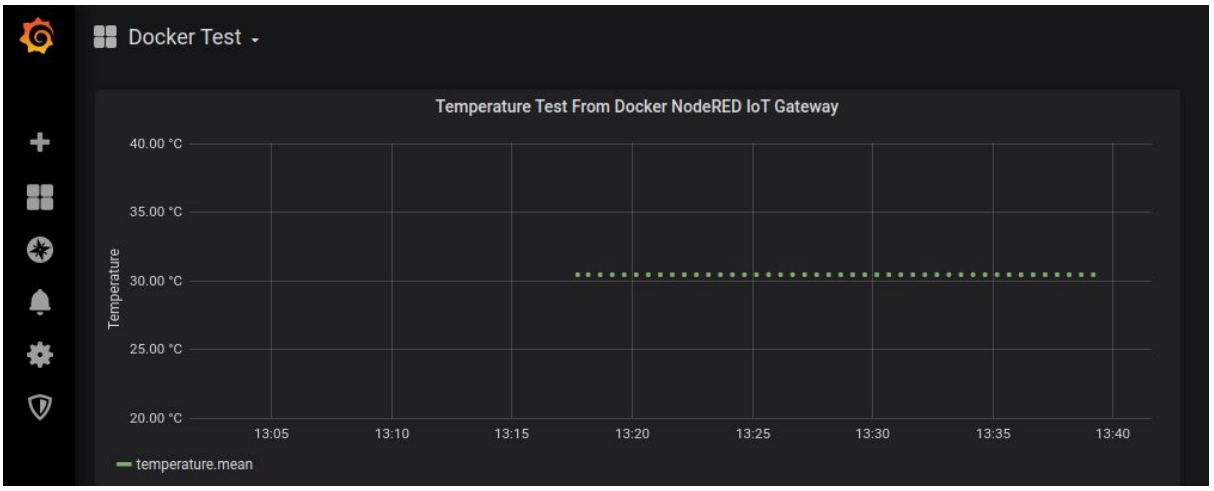




Click Done and enable Flow.

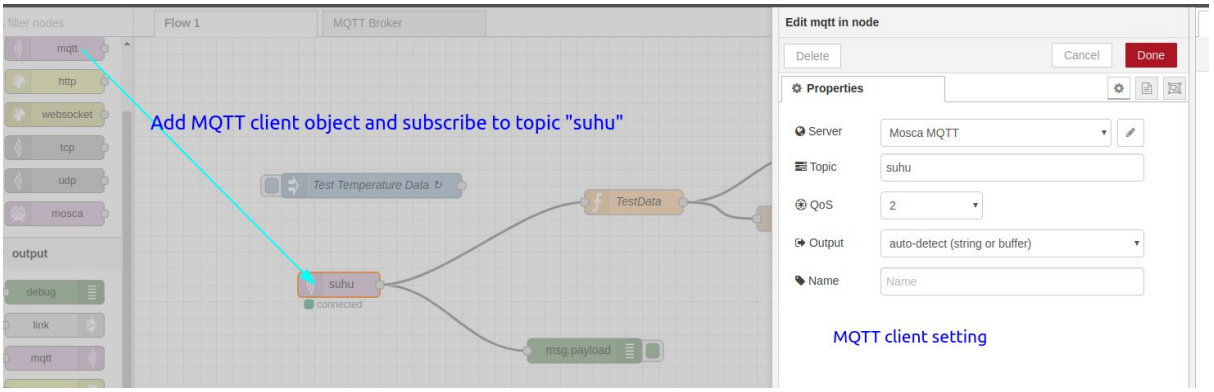
Your PC now is able to receive messages from MQTT Clients like your ESP32 Iot device.

Try send temperature data and see the result in Grafana Server:



Now you can try to send Temperature data using MQTT client by publishing the data to Mosca server and subscribe back the data using NodeRed MQTT client. The data is then can be written to InfluxDB

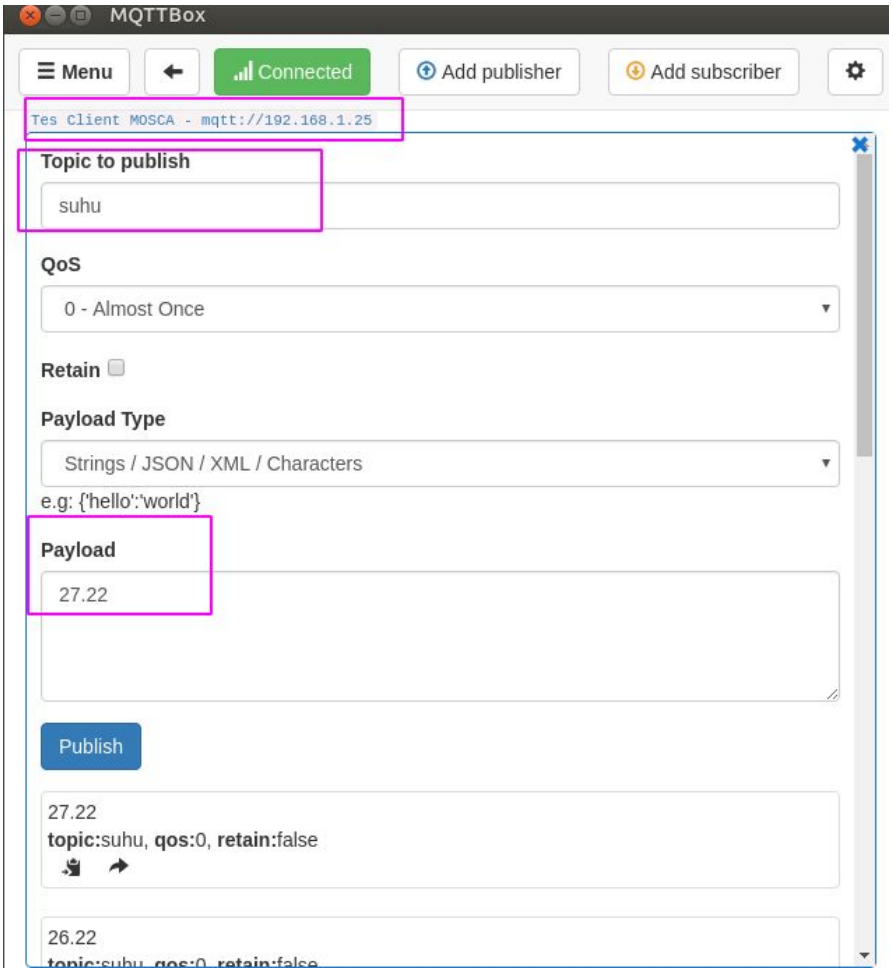
Here is the updated Flow:



You need to change the JS script function to process temeprature data arrives from MQTT client.

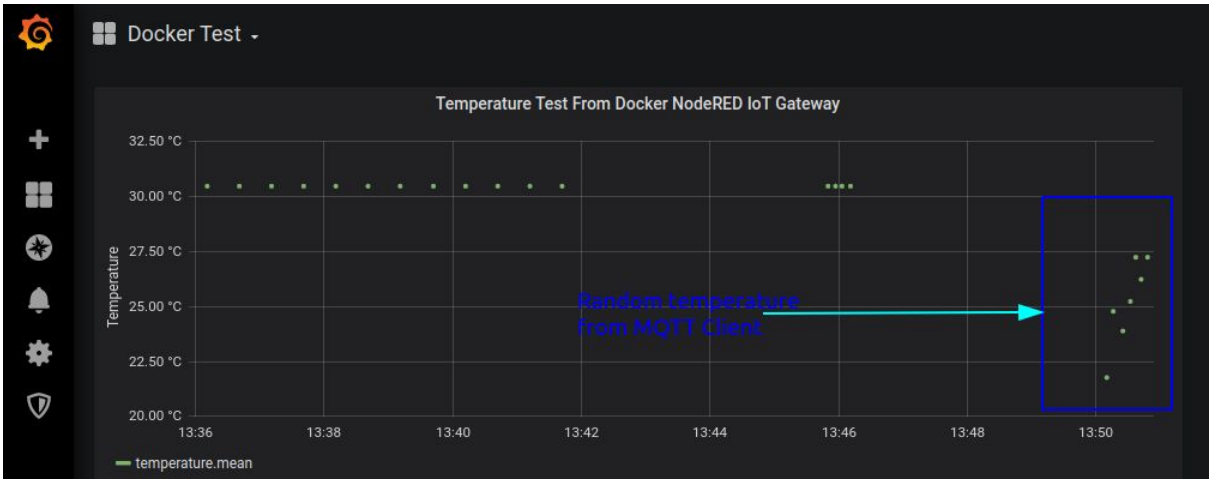
```
t = parseFloat(msg.payload);
jsn = {};
jsn.temperature = t;
tag = {};
tag.iot = "dht22";
msg.payload = [jsn,tag];

return msg;
```

Now you can use MQBOX Client to send test data reandomly:



Item 6

Your test data will be displayed immediately by the Grafana Visualization server.



Now your IoT gateway that supports MQTT, InfluxDB and Grafana visualization server is complete.