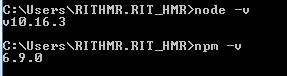
# Exercise 1: Understand Node-RED platform

Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways. It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single- click.

* **Browser-based flow editing**-
* **Built on Node.js**
* **Social Development**

# Installing Node.js

* Download Node.js from the link [https://nodejs.org](https://nodejs.org/) for windows x86 i.e 10.16.3 LTS version
* Install it locally in your system.
* Open the command prompt and display the versions of **nodejs and npm** as shown below.



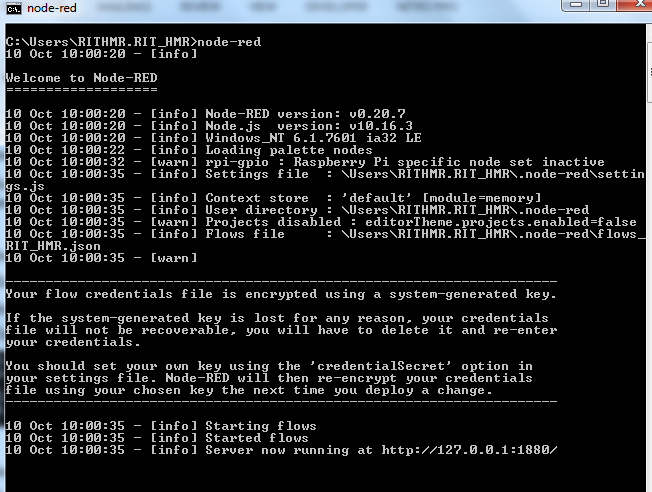
# Installing Node-red:

* Run the below command in the command console.

**npm install –g –unsafe-perm nodered**

* After complete installation run the below command and note down the address given as shown below.

**node-red**





Open the browser and type the above link as **http://localhost:1880**

# Exercise 2: Your First Flow

# The best way to explain “a flow” is by creating one. In this mini flow, we are going to inject a value into our debug window

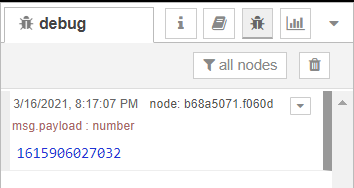
# Open the browser

1. In the address line enter <http://localhost:1880>
2. Drag and drop **“Inject node”** from the nodes library into the flow editor (Once you have chosen the inject node, you should set some general explanation about its functionality in the info pane-no need to read that now).
3. Drag and drop a **“Debug node”** from the nodes library into the flow editor.
4. Crate a pipe between the inject and debug nodes by drawing a connection between their small grey rounded rectangles.
5. Change from the info pane to the debug pane (Upper right).
6. Deploy (=start) your flow.
7. Once deployed, press the left blue rectangle that’s attached to the inject node. Check what’s happening in the debug pane. You should see numbers appear in the sidebar. By default, the Inject node uses the number of milliseconds since January 1st, 1970 as its payload.

**Flow diagram:**



**Output:**



# Exercise 3: Printing date

# Use the flow from Exercise 2

# Add a Function node: The Function node allows you to pass each message though a JavaScript function.

# Delete the existing wire (select it and press delete on the keyboard).

# Wire a Function node in between the Inject and Debug nodes.Double-click on the Function node to bring up the edit dialog.Copy the following code into the function field:

# // Create a Date object from the payload

# var date = new Date(msg.payload);

// Change the payload to be a formatted Date string

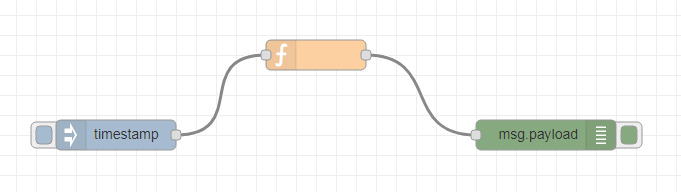
**msg.payload = date.toString();**

// Return the message so it can be sent on

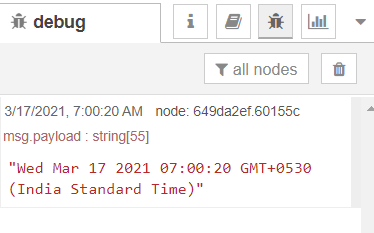
**return msg;**

1. Click done to close the edit dialog and then click the deploy button.
2. Now when you click the Inject button, the messages in the sidebar will now be formatted is readable timestamps.

**Flow Diagram:**



**Output:**



# Exercise 4: To Setup a basic web server in Node-Red

# From the input panel, choose http node

# Change the properties of your http node so that it will respond to GET requests to /mypage from a web browser.

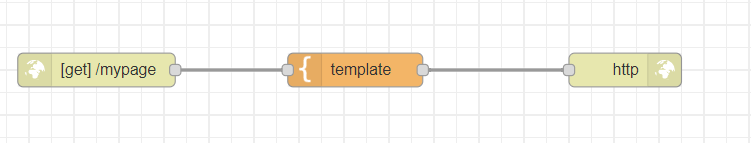
# Add a template node (from functional panel), and add a http response node from the output panel.

# Write your flow together as shown below:

# Deploy your flow

# Once deployed, open a separate browser window enter localhost:1880/mypage

**Flow Diagram:**



**Output:**

# 

# Exercise 5: Printing custom messages on web server

# From the input panel, choose http node

# Change the properties of your http node so that it will respond to GET requests to /test1 from a web browser.

# Add a template node (from functional panel), and add html code

# <htmL>

<head>

<title> Node red custom site</title>

</head>

<body>

<h1>Hello world!</h1>

<p>Your text goes here</p>

</body>

</htmL>

# Add a http response node from the output panel.

# Write your flow together as shown below:

# Flow diagram:

# 

# Deploy your flow

# Once deployed, open a separate browser window enter <localhost:1880/test1>

# Output:

# 

# Experiment-06: Retrieve information from a website at a regular interval, Convert that information into a useful form and Display the result in the Debug sidebar

# Add an Inject node, set the repeat interval to every 5 minutes.

# Add an HTTP Request node

* + The HTTP Request node can be used to retrieve a web-page when triggered.
  + After adding one to the workspace, edit it to set the URL property to: <https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/significant_week.csv>
  + This URL is a feed of significant earthquakes in the last 7 days from the US Geological Survey web site.
  + The site offers a number of other options that you may want to play around with after completing this tutorial.

# Add a CSV node, Enable option for ‘First row contains column names’.

# Add a Debug node

# Wire them all together, Add wires connecting:

* + The Inject node output to the HTTP Request node input.
  + The HTTP Request node output to the CSV node input.
  + The CSV node output to the Debug node input.

# Add a Switch node

* + Edit its properties and configure it to check the property msg.payload.mag with a test of
  + >= and the value 7.
  + Add a second wire from the CSV node to this Switch node.

# Add a Change node

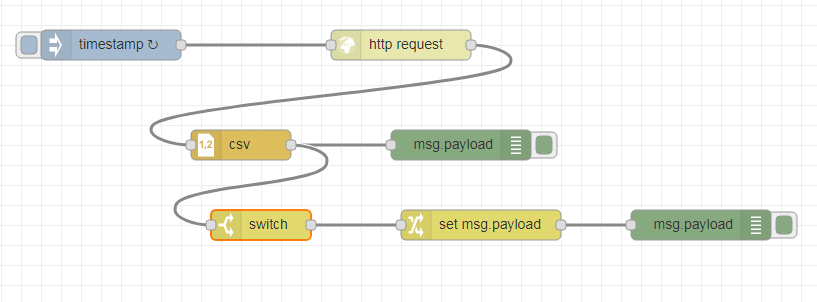
* + Add a Change node, wired to the output of the Switch node.
  + Configure it to set **msg.payload** to the string **PANIC!.**

# Add a Debug node

* + Wire a new Debug node to the output of the Change node

1. Deploy
   * Deploy the flow to the runtime by clicking the Deploy button.
   * With the Debug sidebar tab open click the Inject button.

**Flow Diagram:**



# Output:

# 

# 

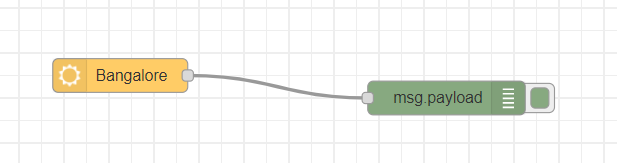
# Conclusion:

This flow is automatically triggered every 1 minutes and retrieves data from a url. It parses the data and displays in the Debug sidebar. It also checks the magnitude value in the data and branches the flow for any messages with a magnitude greater than, or equal to, 7. The payloads of such messages are modified and displayed in the Debug sidebar.

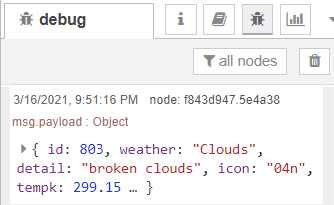
**Exercise 07:** Using openweathermap API to get details of a particular location

1. use **openweathermap** node
2. goto **manage palette** and install openweathermap
3. goto openweathermap.org and sign up and **generate API key**
4. provide **API key in the openweathermap pallete**

**Flow Diagram:**



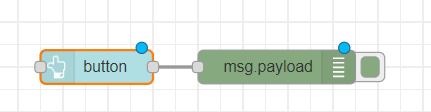
**Output:**



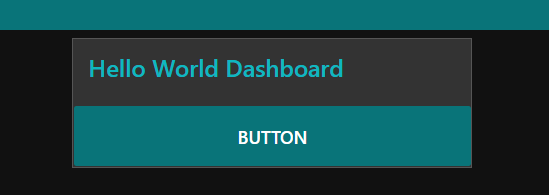
**Exercise 08: Creating dashboard**

1. Install **node red dashboard** from manage palette
2. Add **butto**n and **debug node**
3. Double click on button, **Add to a group**, we dont have a **tab**, create it and call it as Hello world and group as Hello World dash board.
4. Deploy it.
5. Access the dashboard in the URL: [http://localhost:1880](http://localhost:1880/)/ui

**Flow Diagram:**



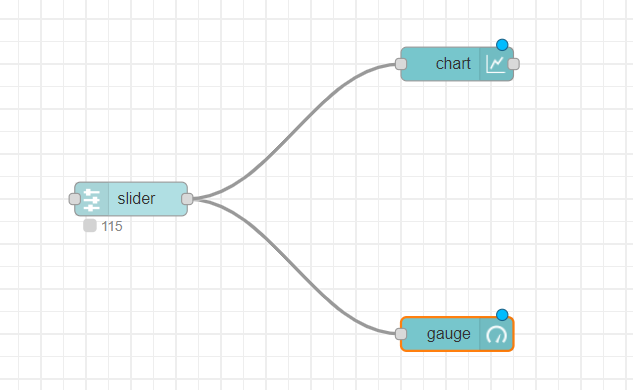
**Output:**



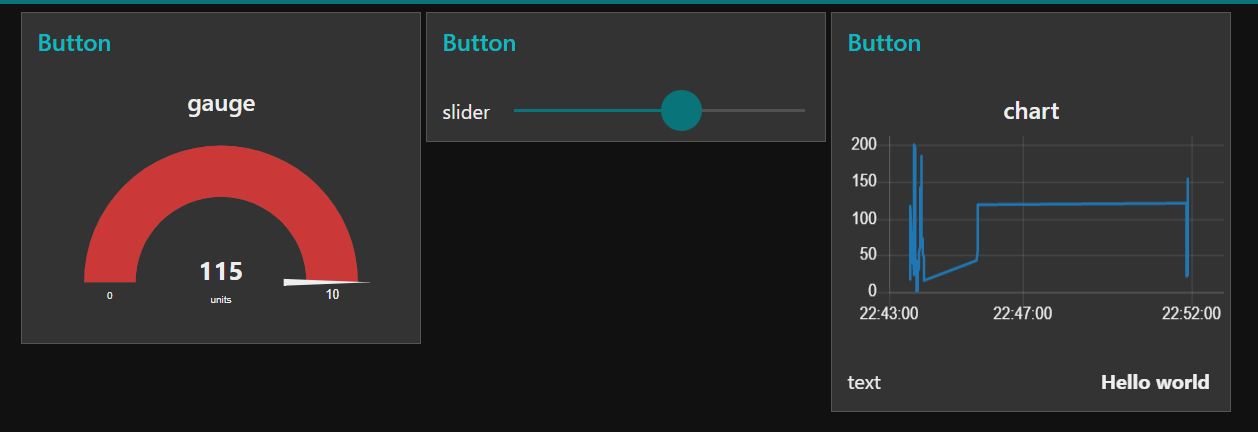
**Exercise 09: Creating dashboard-charts**

1. Add **slider, gauge and chart nodes**
2. Double click on all the three nodes and to the same group, **Add to a group**, we dont have a **tab**, create it and call it as Button and group as Button.
3. Deploy it.
4. Access the dashboard in the URL: <http://localhost:1880/ui>
5. Play around the properties of gauge, chart and slider

**Flow Diagram:**



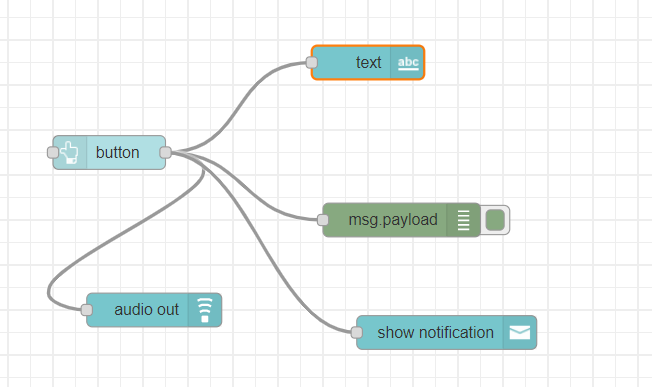
**Output:**



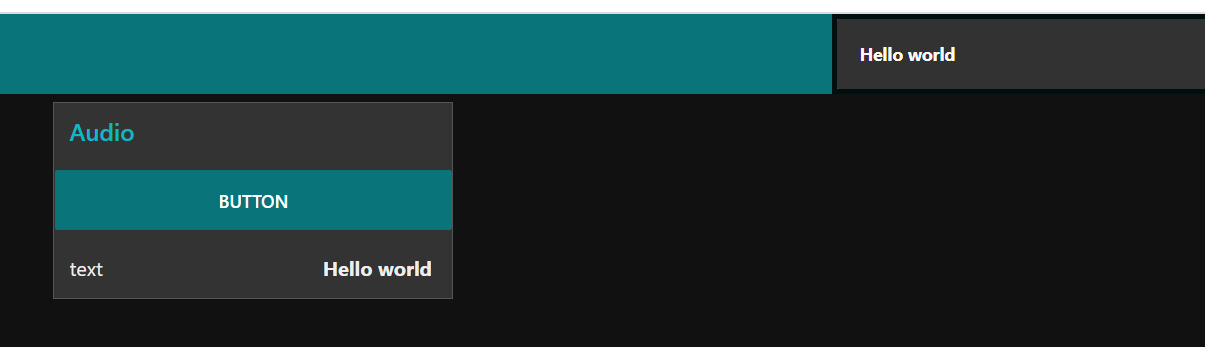
**Exercise 10: Creating dashboard-audio out**

1. Add **button, text, audio out, notification and debug nodes**
2. Double click on all the nodes and to the same group, **Add to a group**, we dont have a **tab**, create it and call it as Audio and group as Audio.
3. For notification node, choose top right option
4. Deploy it.
5. Access the dashboard in the URL: <http://localhost:1880/ui>

**Flow Diagram:**



**Output:**



**Exercise 11: Creating dashboard-Generating random number and display on widgets**

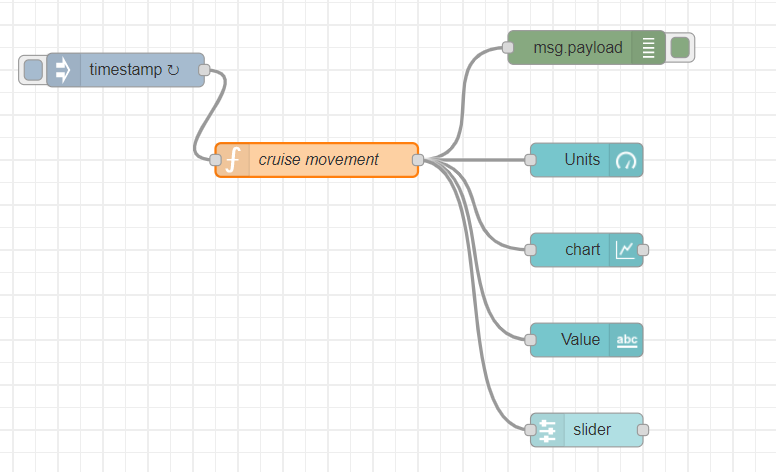
1. Add **inject, function, debug, gauge, chart, text and slider nodes.**
2. Double click on all the nodes and to the same group, **Add to a group**, we dont have a **tab**, create it and call it as Movement and group as Movement.
3. For the function node, add this code to generate random numbers

msg.payload = Math.round(Math.random(50)\*30);

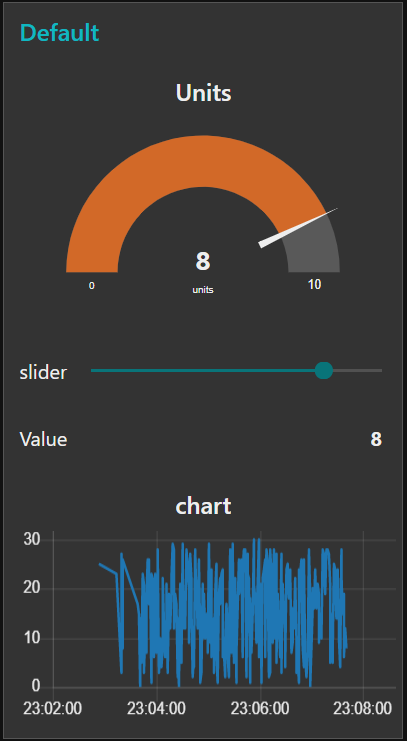
return msg;

1. Deploy it.
2. Access the dashboard in the URL: <http://localhost:1880/ui>

**Flow Diagram:**



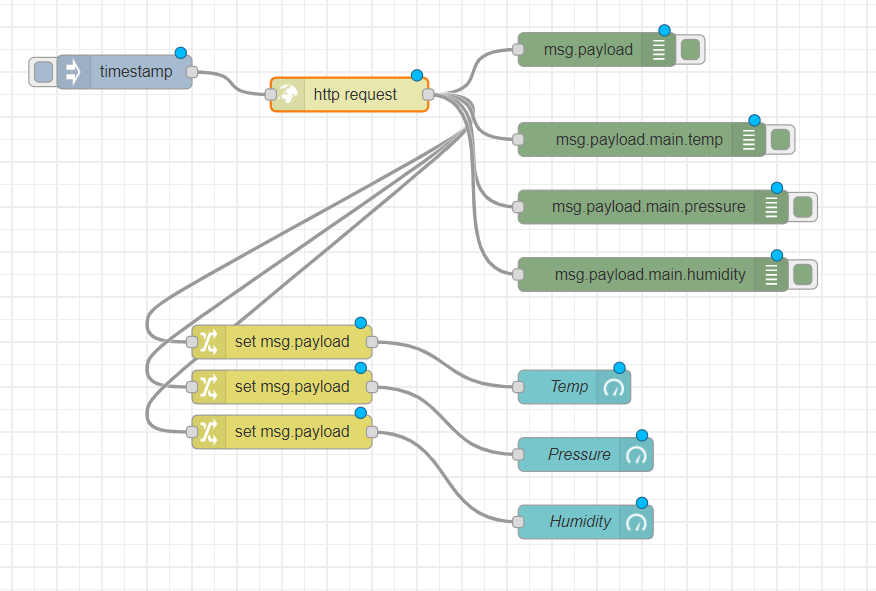
**Output:**



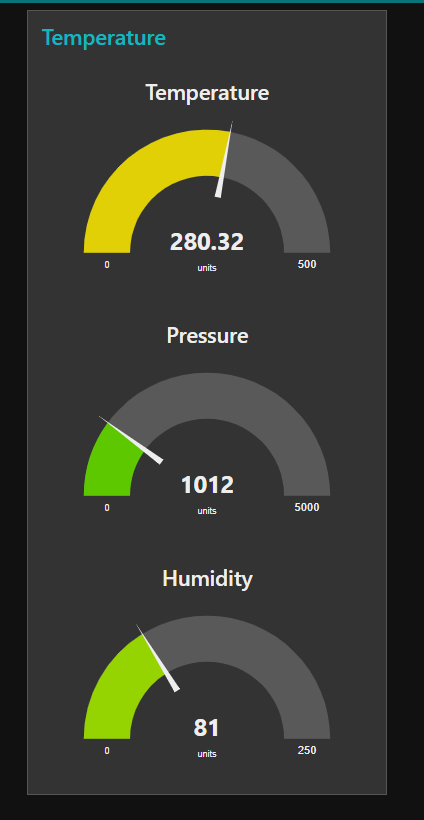
**Exercise 12: Creating dashboard-displaying temperature, humidity, pressure of a location**

1. Add **inject, change, debug, and gauge nodes.**
2. Double click on all the nodes and to the same group, **Add to a group**, we dont have a **tab**, create it and call it as Temperature and group as Temperature.
3. For the http request node, make a GET request to the URL: [http://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b6907d289e10d714a6e88b30761fae22](http://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b6907d289e10d714a6e88b30761fae22%20)
4. For the change nodes, set the property to payload.main.temp, payload.mai.pressure and payload.main.humidity respectively.
5. Deploy it.
6. Access the dashboard in the URL: <http://localhost:1880/ui>
7. Play around the properties of gauge, chart and slider

**Flow Diagram:**



**Output:**



**Exercise 13: Creating dashboard-Error checking criterion**

1. Add **inject, function, debug, gauge, chart, slider nodes an audio out nodes.**
2. Double click on all the nodes and to the same group, **Add to a group**, we dont have a **tab**, create it and call it as OSParameters and group as OSParameters.
3. For the function node, add this code to generate random numbers

msg.payload = Math.round(Math.random(50)\*30);

return msg;

1. For the Second function node, add this code

if(msg.payload >= "150")

{

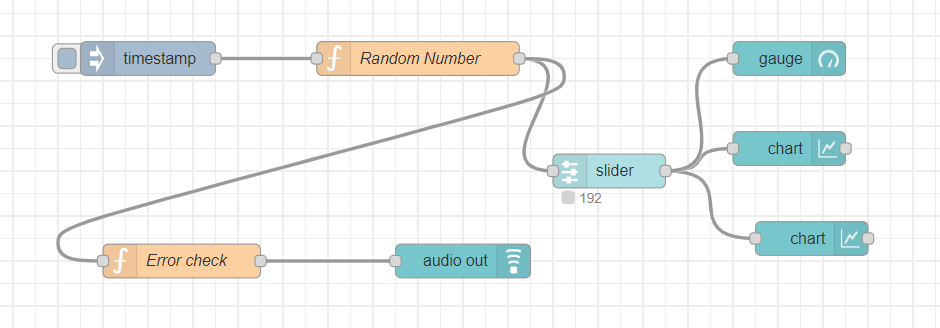
msg.payload = "Too High!";

}

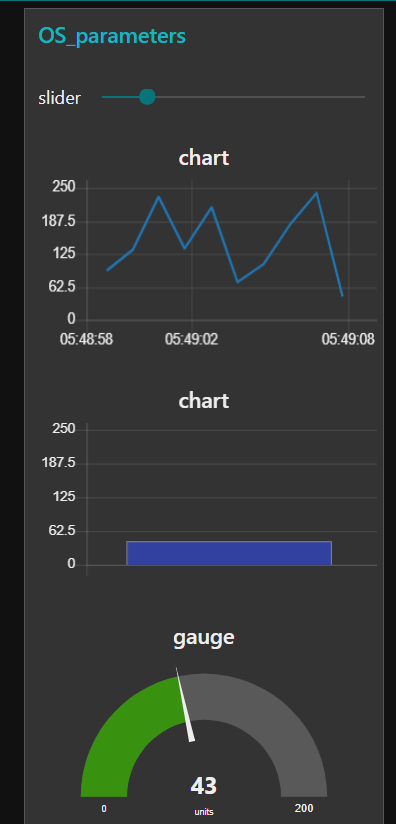
return msg;

1. If random number is greater than 150, TOO high message is popped out from an audio node.
2. Deploy it.
3. Access the dashboard in the URL: <http://localhost:1880/ui>

**Flow Diagram:**



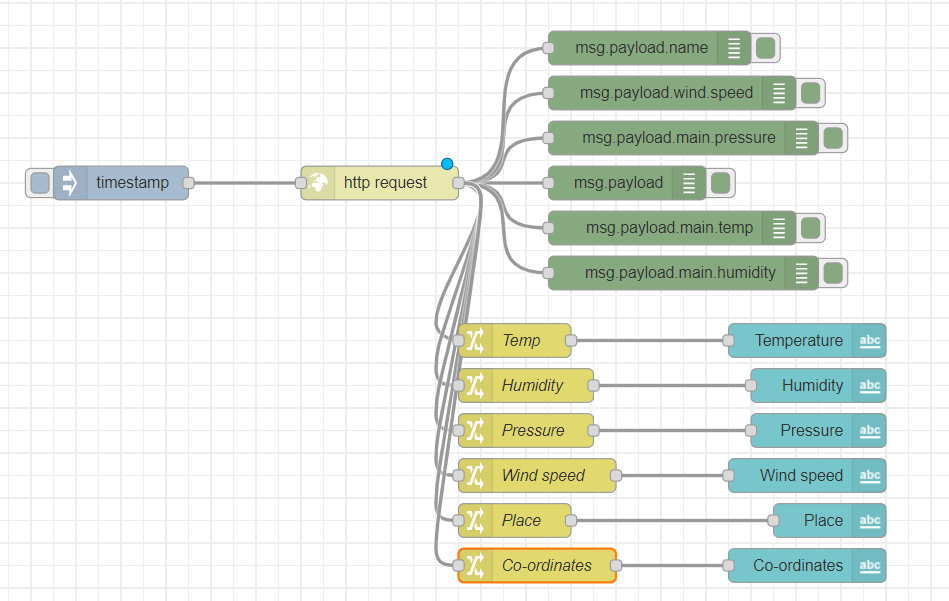
**Output:**



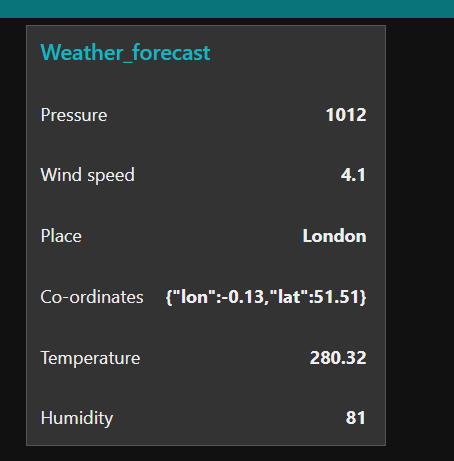
**Exercise 14: Creating dashboard-displaying temperature, humidity, pressure of a location**

1. Add **inject, change, debug, and text nodes.**
2. Double click on all the nodes and to the same group, **Add to a group**, we dont have a **tab**, create it and call it as Weather\_forecast and group as Weather\_forecast.
3. For the http request node, make a GET request to the URL: [https://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b6907d289e10d714a6e88b30761fae22](https://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b6907d289e10d714a6e88b30761fae22%20)
4. For the change nodes, set the property to payload.main.temp, payload.main.pressure, payload.main.humidity,payload.main.wind.speed,paload.mai.cord and payload.main.name respectively.
5. Deploy it.
6. Access the dashboard in the URL: <http://localhost:1880/ui>
7. Play around the properties of gauge, chart and slider

**Flow Diagram:**



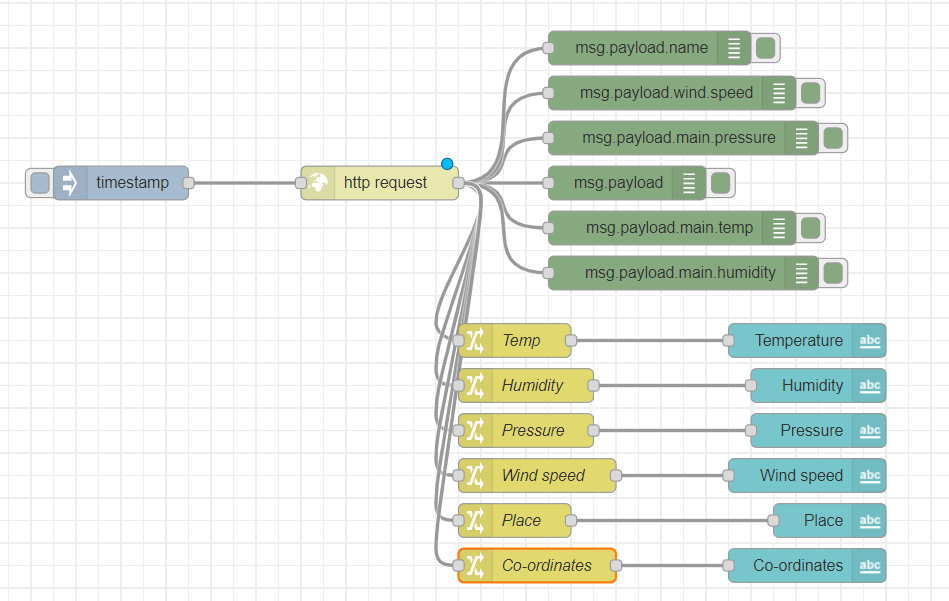
**Output:**



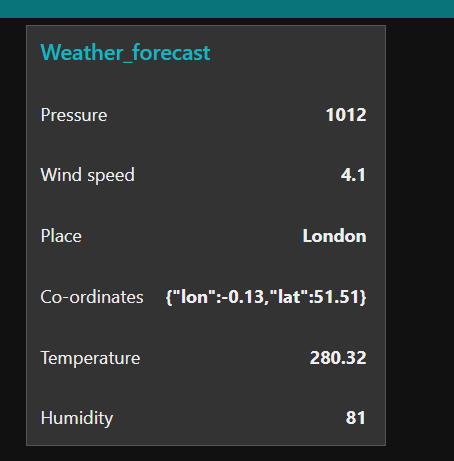
**Exercise 15: Creating dashboard-displaying temperature, humidity, pressure of a location**

1. Add **inject, change, debug, and text nodes.**
2. Double click on all the nodes and to the same group, **Add to a group**, we dont have a **tab**, create it and call it as Weather\_forecast and group as Weather\_forecast.
3. For the http request node, make a GET request to the URL: [https://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b6907d289e10d714a6e88b30761fae22](https://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b6907d289e10d714a6e88b30761fae22%20)
4. For the change nodes, set the property to payload.main.temp, payload.main.pressure, payload.main.humidity,payload.main.wind.speed,paload.mai.cord and payload.main.name respectively.
5. Deploy it.
6. Access the dashboard in the URL: <http://localhost:1880/ui>
7. Play around the properties of gauge, chart and slider

**Flow Diagram:** **https://www.slideshare.net/BorisAdryan/node-redcamjampibirthdayfeb2015**



**Output:**



**Exercise 16: Crypto currencies and Node-Red**

Node-RED is a perfect tool for rapid development of crypto currency applications. There are a wide range of uses, starting with simple access to exchange data (e.g. prices, trends) right up to developing your own block chain!

# Getting and displaying basic cryptocurrency data using Node-RED

## The Binance node

[Binance](https://www.binance.com/) is one of the largest crypto trading platform (perhaps largest by volume), and importantly has a comprehensive and sophisticated API. The resident cryptocurrency expert, Ted, over at [Sense Tecnic Systems](http://www.sensetecnic.com/) (STS) recently decided to jump in and develop a new Node for accessing and using the Binance cryptocurrency platform.

### Simple flow to get all market data

* For this demo we’ll use the Binance getPrice node. The Binance getPrice node takes a simple ticker pair as input and returns the current price for that ticker. For this example, we’ll use the Bitcoin to Tether USD ticker pair, which is BTCUSDT. We’ll use the inject node to send that to the getPrice node, so double click on the inject node, set the payload to an empty string, and set it’s topic  to **BTCUSDT**
* The getPrice node looks for a ticker pair on the incoming message topic and uses that to make an API call to the Binance platform, which returns the current price. Our simple flow then passes that to a debug node, which will show the results in the debug pane on the right side of Node-RED window.

## Binance node main features

The Binance node set covers two main areas of functionality – accessing market data and account related functionality

* **getPrice:**Gets the latest price of a symbol. Takes a ticker pair as input and returns the current price
* **getAllPrices**: Gets the latest price of all symbols. No input and returns an {object} map of all available ticker pairs and their current prices
* **getBookTicker**: Gets the bid/ask prices for a symbol. Takes a ticker pair as input and returns an {object} info on latest book price
* **getDayStats**: Get the 24hr ticker price change statistics for a symbol. Takes a tickerPair as input and returns the latest 24h stats
* **getCandlesticks**: Get Kline/candlestick data for a symbol. Takes a tickerPair, time interval and start/ed times and returns an {array} of candlesticks/kline data specified by parameters

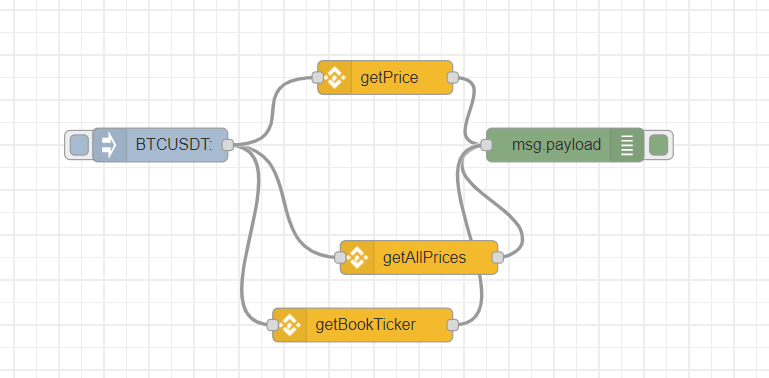
Go ahead and wire up your flow as described, deploy it and then click on the inject node and you’ll see the current price in the debug pane as shown below.

Let’s continue our simple example shown above but extend it to get all market info. For that we’ll need the getAllPrices node which, as you’d expect, returns all the prices that Binance handles. It’s a long list and is returned as an object. However, we can display it in the debug pane, so let’s modify our original flow and replace the getPrice node with a getAllPrices node as shown below. Note that the inject node doesn’t need to set the topic – although its ignored if you do.

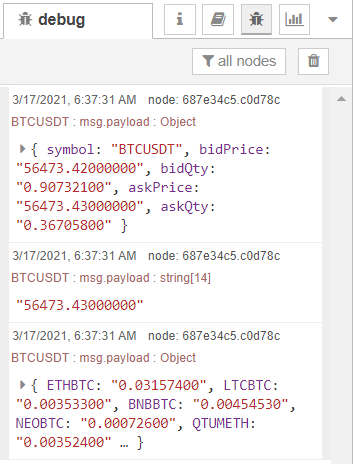
### Getting bid/ask prices

As a final example, let’s return to the **BTCUSDT**ticker pair and get the full book info for it using the getBookTicker node. Using the first flow that injects BTCUSDT as a message topic, replace the getPrices node with a getBookTicker

**Flow Diagram:**

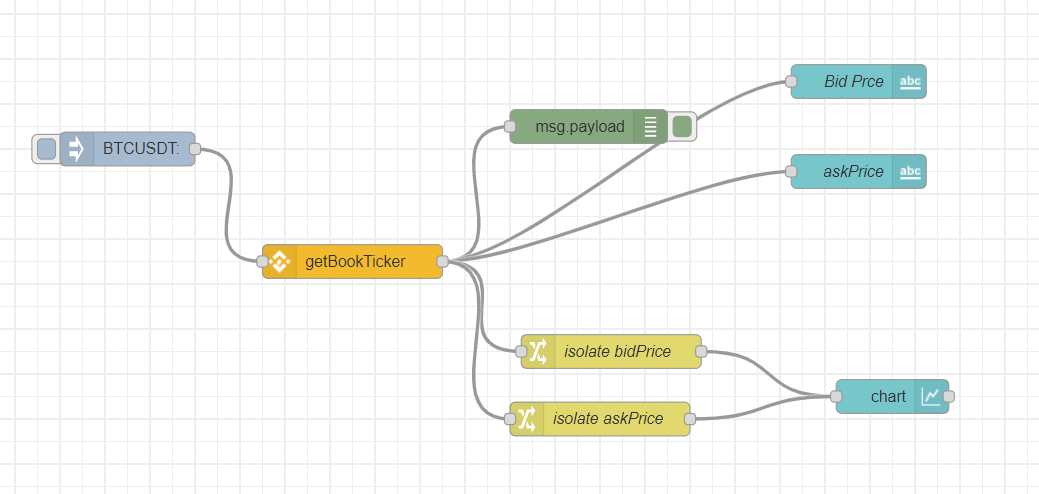


**Output:**

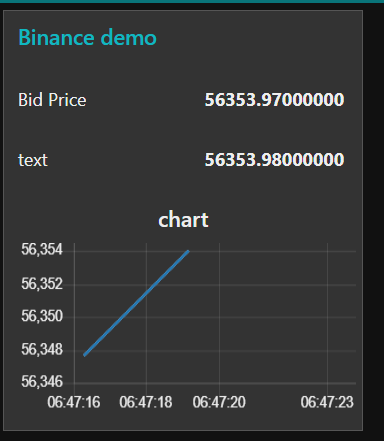


**Exercise 17:** Develop a simple dashboard showing bid and ask prices for symbol BTCUSDT

**Flow Diagram:**

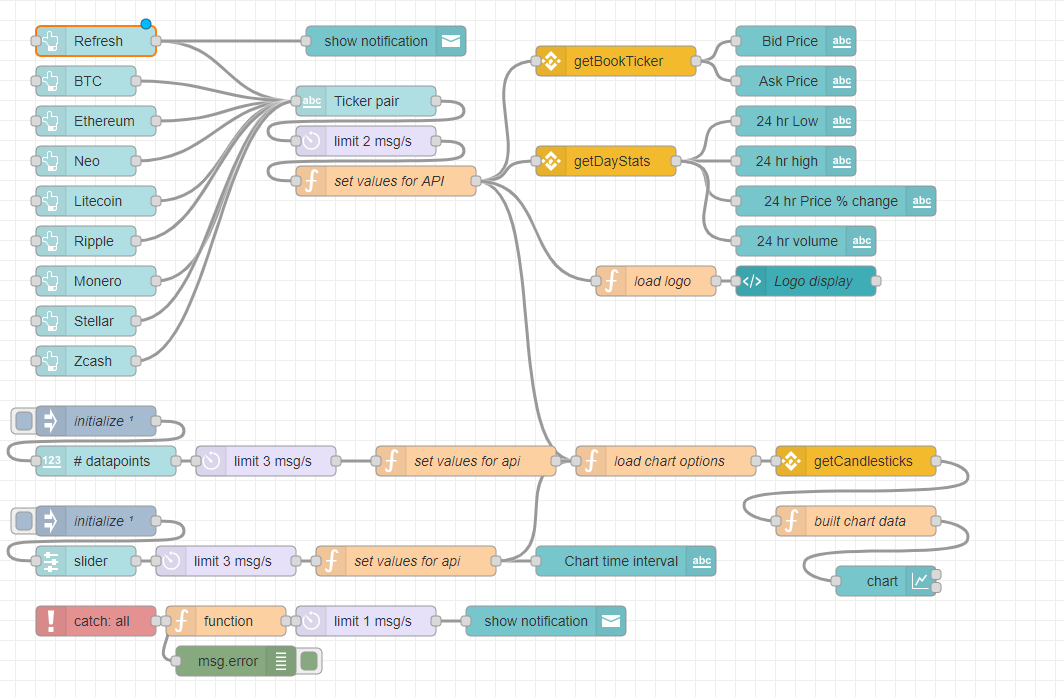


**Output:**



**Exercise 17:** Develop a dashboard showing bid and ask prices for different types of crypto currencies

**Flow diagram:**



**Output:**

