

## Dashboard for Monitoring Inventory/Stock status

**Difficulty Level:** Easy

### Objective

Implement a visual dashboard to monitor the number of available parts in the warehouse and products in the stock, utilizing the inventory/stock web service API from the previous modules.

### Achievements

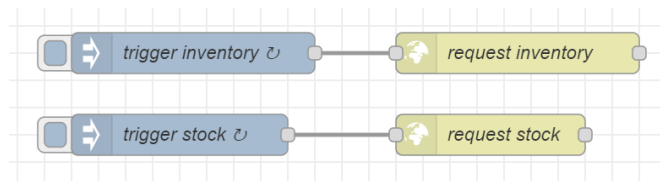
The skills to be acquired at the end of this module:

- Creating web UI with multiple data series in a single plot
- Using bar charts for data visualization

### 1. Periodically Retrieving the Inventory/Stock Status

In this final module of the smart factory scenario, we are going to visually present the current status of the parts inventory and product stock. For this, we have to utilize the web service API that was developed in Module 6 (“Implementing an Inventory Service API – GET Method”).

We will start by adding two “inject” nodes, in order to trigger the request for the inventory and stock values periodically. We will then connect these inject nodes to the http request nodes, one for the “inventory” API endpoint and one for the “stock” API endpoint.



The two **inject** nodes are configured with an interval of 1 sec. to have frequent update of the inventory-stock plots. To avoid simultaneous http requests, the initial inject values are set slightly different.

Name: trigger inventory

msg. payload = timestamp

msg. topic = a\_z

☒ Inject once after 0.1 seconds, then

Repeat: interval

every 1 seconds

Name: trigger stock

msg. payload = timestamp

msg. topic = a\_z

☒ Inject once after 0.2 seconds, then

Repeat: interval

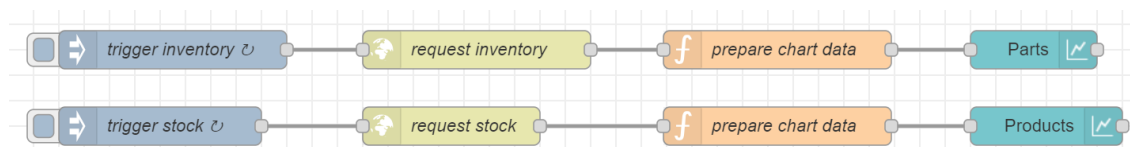
every 1 seconds

The two **http request** nodes are configured as standard GET requests to the web services running on the localhost (<http://localhost:1880/inventory> and <http://localhost:1880/stock>, respectively).

Method	GET	Method	GET
URL	<a href="http://localhost:1880/inventory">http://localhost:1880/inventory</a>	URL	<a href="http://localhost:1880/stock">http://localhost:1880/stock</a>
Payload	Send as request body	Payload	Send as request body
<input type="checkbox"/> Enable secure (SSL/TLS) connection		<input type="checkbox"/> Enable secure (SSL/TLS) connection	
<input type="checkbox"/> Use authentication		<input type="checkbox"/> Use authentication	
<input type="checkbox"/> Enable connection keep-alive		<input type="checkbox"/> Enable connection keep-alive	
<input type="checkbox"/> Use proxy		<input type="checkbox"/> Use proxy	
Return	a UTF-8 string	Return	a UTF-8 string
Name	request inventory	Name	request stock

We can now continuously retrieve the current inventory and stock values every 1 second. As the next step, we would like to plot those data on the web UI. For this purpose, we can use the *chart* node of the Dashboard package, but we first need to prepare the data in a proper format, so that our charts provide a nice visualization of the multi-series data. We will use *function* nodes for the data preparation as an input to the *chart* nodes.

Here is how the final flow should look like:



Let us first look at the function nodes, starting with the one for inventory data preparation:

```

1 var parts = JSON.parse(msg.payload);
2 msg.payload = [{
3   "series": ["Parts"],
4   "data": [parts.partA, parts.partB, parts.partC],
5   "labels": ["Part A", "Part B", "Part C"]
6 }];
7 return msg;

```

- In the first line of code above, *msg.payload*, which is the output of the “*request inventory*” *http request* node, is parsed and converted into a JSON object, named “*parts*”.
- In lines 2-6, we are constructing the new *msg.payload* object to be passed on to the *chart* node to plot the parts’ inventory values dynamically. As you can notice, this is structured also as a JSON object with three *string* keys; namely, “*series*”, “*data*”, and “*labels*”, as expected by a *chart* node. In line 3, we simply set the series name to “*Parts*”, which can be any indicative name of our choice.

- Recall that the GET /inventory API call returns the values of the parts in the form of key-value pairs, such as “partA: 1”, “partB: 2”, etc. We can then access each of the individual part values from the JSON object *parts*, as given in line 4.
- In lines 4 and 5, we are providing the values and labels of the three plots, to be used by the *chart* node. If this sounds complex now, it should be clearer once we complete and deploy our flow and then test the behavior on the web UI.
- Finally in line 7, we return the *msg* object, so that it is sent to the next node in our Node-RED flow.

The code for the other function node (for stock data preparation) is very similar in logic to the first one:

Name

prepare chart data

Setup

On Start

On Message

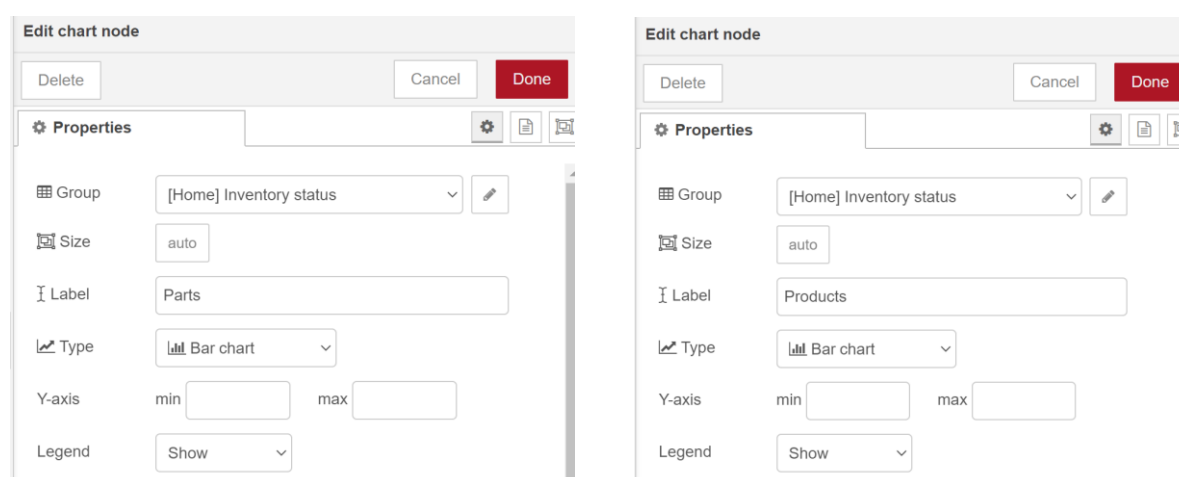
On Stop

```

1 var products = JSON.parse(msg.payload);
2 msg.payload = [{
3   "series": ["Products"],
4   "data": [products.product1, products.product2],
5   "labels": ["Product 1", "Product 2"]
6 }];
7 return msg;

```

For the chart nodes, we add a new “dashboard group” named “*Inventory status*” and set the Group property of both chart nodes (*Parts* and *Products*) to this group, as depicted below. (See Module 5 “Monitoring and Visualization of Sensor Data” for the details on the chart node and dashboard group configurations.)



Our Node-RED flow is now ready for the dynamic visualization of the parts inventory and product stock values.

We can go ahead and deploy the flow. Assuming that your NodeRED is running at the address <http://localhost:1880>, simply add a “/ui” at the end of that to reach the NodeRED Dashboard web UI, i.e., <http://localhost:1880/ui/>.

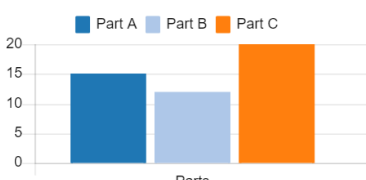
The final result of the Web UI may look like the following, where the last three training modules are actually brought together in our case. The left most column below corresponds to the inventory status for parts and products that we implemented in this module, the “Production” section represents the outcome of the previous module (module 9), and the “Parts delivery” and “Product sales” sections represent the outcome of the module 8 of the “smart factory” scenario.

We can now simulate parts delivery, production, and product sales all in one screen and dynamically observe their impact on the parts and products’ availability.

Home

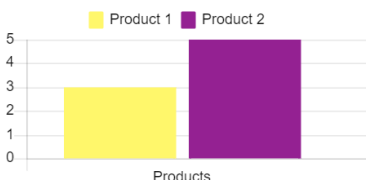
### Inventory status

#### Parts



Part	Inventory
Part A	15
Part B	12
Part C	20

#### Products



Product	Inventory
Product 1	3
Product 2	5

### Parts delivery

Part A

Part B

Part C

SUBMIT

CANCEL

### Production

PRODUCT 1

PRODUCT 2

### Product sales

Product 1

Product 2

SUBMIT

CANCEL