

## Simulation of Production

**Difficulty Level:** Medium

### Objective

Implement a web-based UI to simulate 'production' in the smart factory, resulting in an increase in product stock values and decrease in the inventory of the associated parts.

### Achievements

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

- Implementing a client that uses a combination of the GET and POST methods of the web service API
- Checking the inventory for sufficient amount of the parts required for production

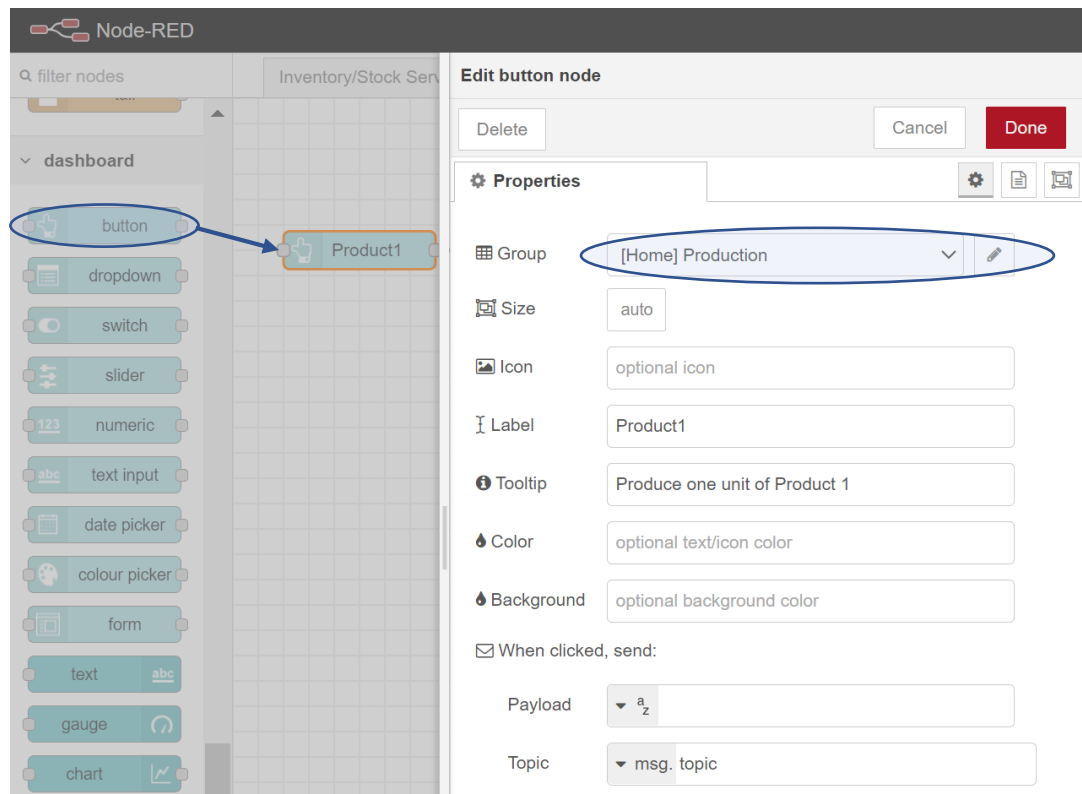
### 1. Using Buttons in the Web UI for Triggering a Specific Action

In this module, we will utilize the Node-RED Dashboard **button** nodes for the user to *trigger an action* via the web UI, in this case the (simulated) production of the products 1 and 2.

Before proceeding to the next steps, make sure that **node-red-dashboard** is installed in your palette manager. If not, refer to the previous IoT-factory exercise module "*Monitoring and Visualization of Sensor Data*" to install it in your Node-RED environment.

Drag and drop a button node into your flow, as depicted in the next figure. The double click on the button node to open and edit its settings.

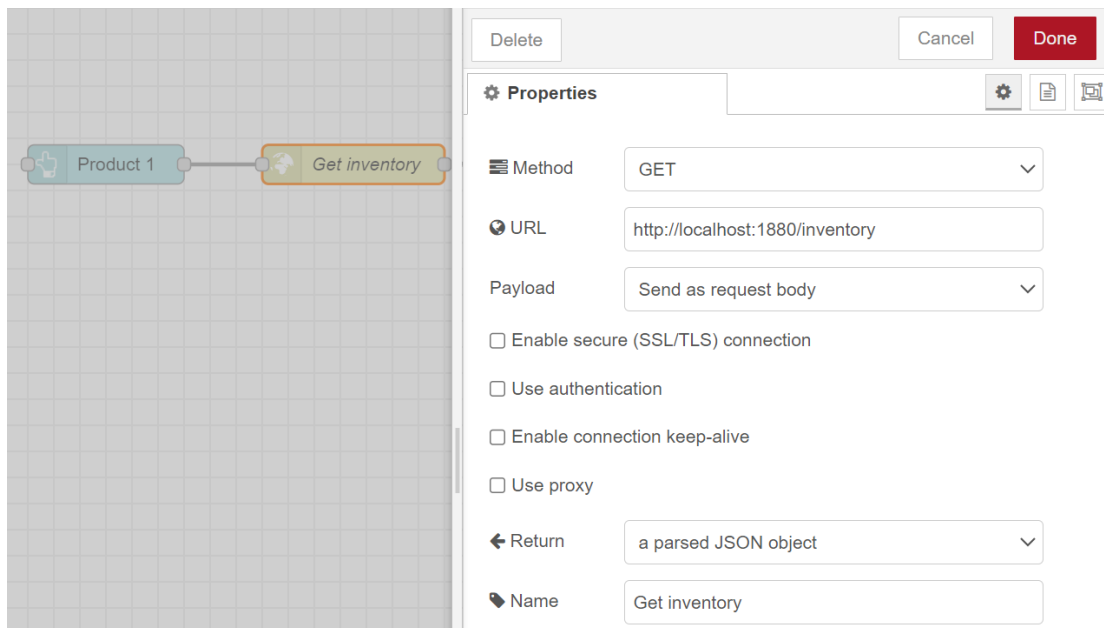
- As with all dashboard / UI nodes, we also need to specify the **Group** for our *form* node. This serves for the organization of different UI elements / nodes on the web page.
- We can create a new group or edit an existing group by clicking on the pen icon, as depicted in the figure below. (For details, refer to the former IoT-factory exercise module "*Monitoring and Visualization of Sensor Data*"). Here, we would like to create a new group called "Production" under the "Home" *tab/page* of the web UI.
- Let us give this first button the *label* of "Product 1", as it will serve for the simulated production of product-1 by making use of the necessary parts from the inventory.
- There is also a "Tooltip" setting, which helps for providing extra information when the user hovers over the button on the web UI. The remaining settings can be left empty or as the default values.
- Note that we would just like to use this button as a trigger whenever the user clicks on it, so we do not need to send any specific payload data to the next node in our flow.



## 2. Simulated Production of Product 1

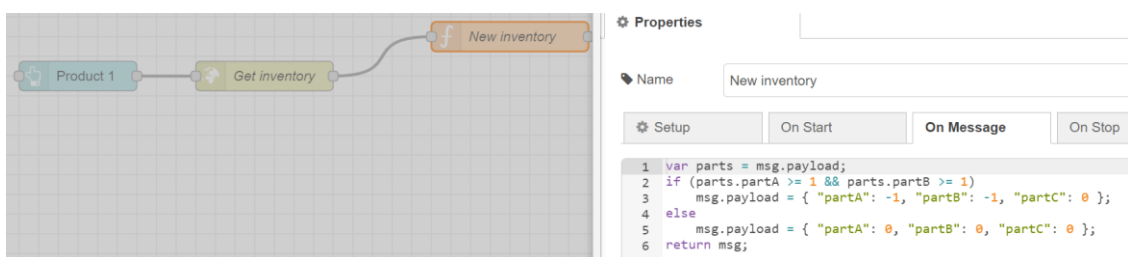
Whenever the user clicks on our “Product 1” button, the amount of the corresponding product will be incremented by 1. However, before producing any of the products, a request to the inventory endpoint needs to be sent, in order to check if there are sufficient parts in the inventory. (Recall that product-1 uses 1 unit of part-A and 1 unit of part-B.)

Therefore, we attach an http request node to the button and configure it as follows:



After we retrieve the parts' availability information by using the inventory service, we attach a function node to the http request node, as depicted below, in order to check if sufficient parts are available (line 2) and then to prepare the `msg.payload` for the HTTP POST request to the inventory service endpoint to store the new parts values.

- If there are indeed sufficient parts in the inventory, both part-A and part-B values should be decremented by 1 (line 3), otherwise they remain intact (line 5) since we cannot produce product-1 in this case.



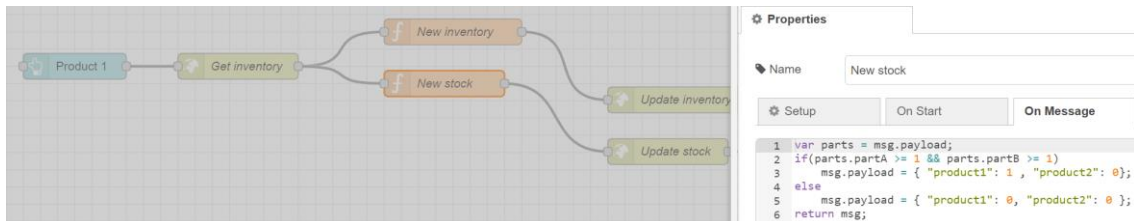
- Finally, we attach an http request node to send the prepared `msg.payload` to the *inventory* service endpoint via an HTTP POST request:



- Now that we are done checking the parts' availability and updating the inventory, we should just increase the product-1 stock value by 1 in case there indeed were enough parts for production. We do this by adding another function node and attaching another http request

node to call the POST method of the *stock* service, in a very similar way to the previous steps for the parts inventory.

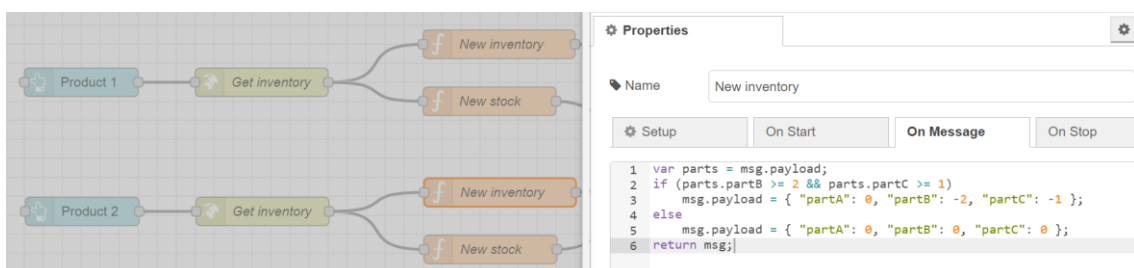
- The following shows the resulting flow with the “New stock” function node (and its properties), followed by the http request node to update the stock.



### 3. Simulated Production of Product 2

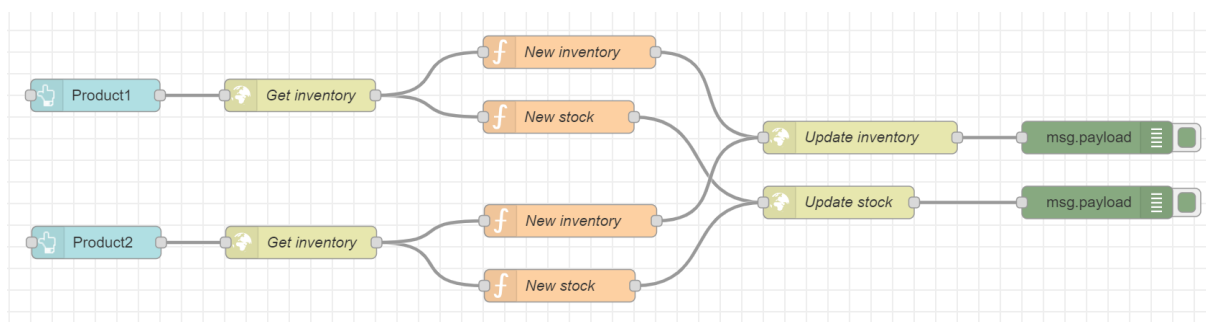
The steps for simulating the production of product-2 is very much similar to the previous steps for product-1. Therefore we can also duplicate the nodes above as a block and then reconfigure those, rather than creating and connecting new nodes from scratch.

- Similar to the above, we need a button with the label “Product 2”, followed by an http request node to retrieve the part inventory values and then function nodes for checking their availability.
- An important point to note here is that product-2 uses 2 units of part-B and 1 unit of part-C. Therefore, in the new function nodes, the *if statement* (line 2) and preparation of inventory/stock update values (line 3) are changed accordingly.



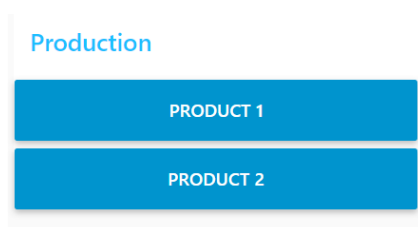
- The code block for the function node “New Stock” for product-2 is not shown here, but a similar reconfiguration is needed for the *if statement* and *msg.payload* updates.
- We can now reuse the same “Update inventory” and “Update stock” nodes from Section 2, i.e. we do not need to create new nodes for those.
- We can also add some debug nodes at the end of each http request (POST) nodes, in order to double check if correct inventory/stock update messages are sent to the http service endpoints.

The final flow would look like the following:



Let us go ahead and deploy our flow. Please keep in mind that the *inventory* and *stock* services should be running on the localhost, as covered in the previous modules.

Once the flow is deployed, you can go to the “Dashboard” web UI by pointing your browser to <http://localhost:1880/> (assuming the default settings in Node-RED environment). You should then be able to see a web page with the two buttons “Product 1” and “Product 2”, as depicted below. You can try the buttons and observe the debug view in Node-RED to see their effect.



In the next and final module, we will develop additional UI elements to monitor the current inventory and stock values, allowing us to conduct integrated testing and validation of the delivery, production, and sales processes, all through the web UI.