

Implementing an Inventory Service API – POST method

Difficulty Level: Medium

Objective

Continue with the implementation of inventory database service from the previous module, so that our Web API allows updating the number of products in the stock or the parts inventory.

Achievements

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

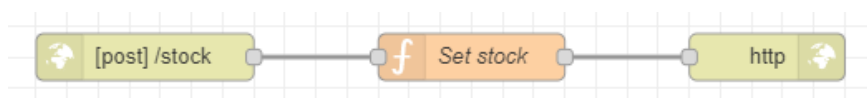
- Implementing a POST method for external software modules to update the inventory/stock states
- Observing the context data and variable values to debug or confirm the application functionality

1. Implementing a POST Method for the Product Stock

In the previous module, we started implementing a web service with an API that uses the GET method to allow other software entities to **read/retrieve** the inventory and stock values from our inventory tracking service. This module will complement this API with the **POST** method in order to allow other software modules also to **change/update** the inventory and stock values.

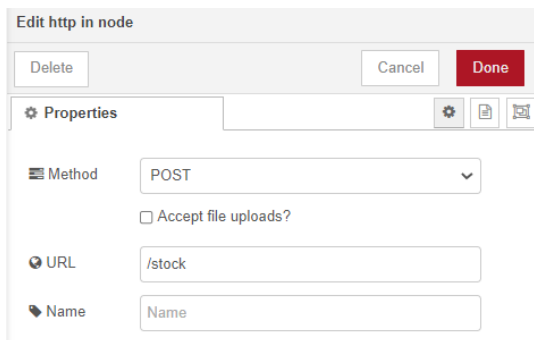
The nodes we will use in this module are actually the same as in the previous module:

- An **http in** node to accept and handle incoming **POST** requests,
- A **function** node to process the request and update the stock/inventory values,
- An **http response** node to send back a confirmation to the requester.

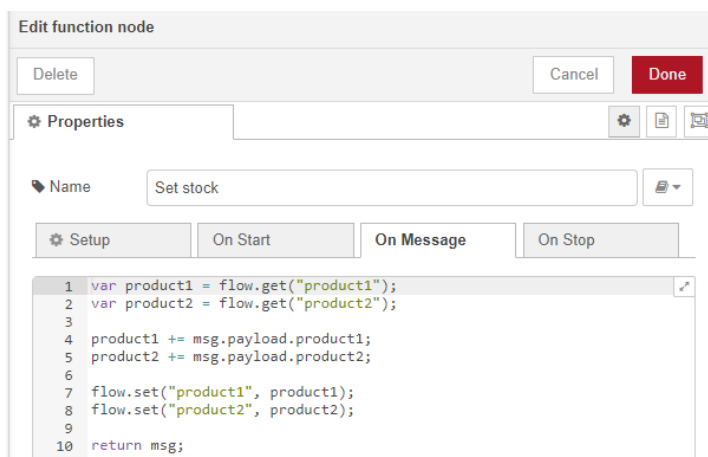


We will see next how to add and configure each of those.

- Recall that the http nodes are available under the Network group in Node-RED palette. Drag and drop an **http in** node to the flow editor and edit the node settings as follows:



- With this, we are adding a new **POST** method to our “**API endpoint**” called “/stock”.
- As in the previous module for the GET method, the URL field in the settings shown above is relative to the base URL. So, if NodeRED application is running on the default localhost:1880, then our API endpoint becomes “localhost:1880/stock”.
- In the function node that follows the http in node (“Set stock”), we will process the incoming data with the POST request message. Drag and drop a function node, connect it to the output of the http in node, and edit it as follows:

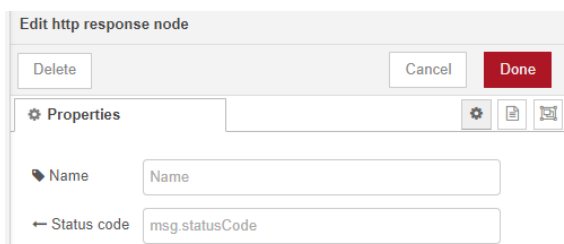


- Note that we are adding the code to the “On Message” section of the function node.
- Lines 1-2 above read the flow variables (created in the previous module to keep track of the product stock values) and assign them to the local (temporary) variables of the same name.
- On lines 4 and 5, we are updating these local variables *product1* and *product2* based on the incoming data. (As a side note, the “+=” operator here adds up both sides of the operator and assigns it to the variable on the left side. So, A+=B is just a compact form of A=A+B.)
- Note that **msg.payload** is a JSON object that is automatically transferred from the output of *http in* node whenever there is an incoming POST message to our API endpoint (via localhost:1880/stock). So, *msg.payload.product1* on line 4 refers to the product1 value in the (JSON) data that comes with the http request (as part of the POST message).

- Finally, lines 7 and 8 assign the updated values of the local variables *product1* and *product2* back to the *flow variables* for persistent usage across all nodes in the flow.
- Line 10 relays the same *msg* object to the output of the function node, although we do not need to use this in the following *http response* node for the purposes of this exercise.

With the function node above, we are processing the incoming POST messages and updating the stock values accordingly. Unlike the GET method, a POST method does not require sending any specific data back to the requester. Normally, the response would be just an *HTTP status code* (e.g., “200 OK” indicating success of the operation), but we leave those details outside of the scope of this module.

So, here we simply add an **http response** node and connect it to the output of our function node (without changing any setting in the *http response* node).



Once we deploy the flow in Node-RED, our API endpoint “/stock” should be accepting both POST and GET method requests.

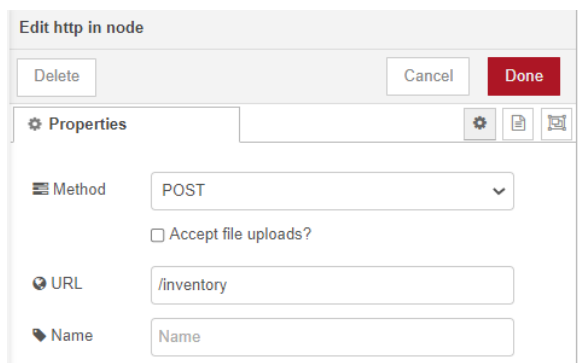


2. Implementing a POST Method for the Parts Inventory

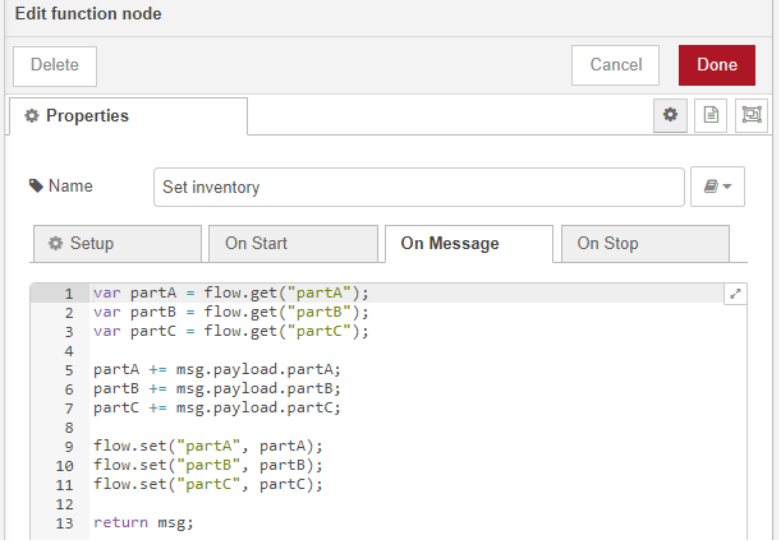
We need a similar POST method for our API endpoint called “/inventory” in order to update the number of available parts in the warehouse.

The process is quite straightforward, following the steps of the previous section:

- We will add a new **http in** node with the **POST** method and URL “/inventory”:



- Again, we will now connect a **function** node to the output of the **http in** node, in order to process the incoming HTTP requests:

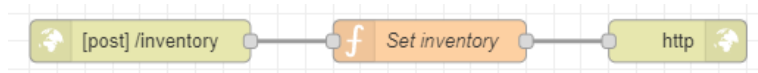


```

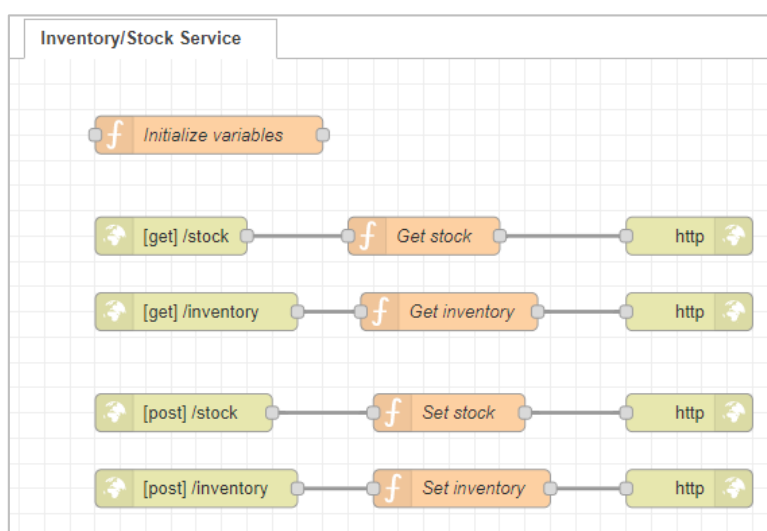
1 var partA = flow.get("partA");
2 var partB = flow.get("partB");
3 var partC = flow.get("partC");
4
5 partA += msg.payload.partA;
6 partB += msg.payload.partB;
7 partC += msg.payload.partC;
8
9 flow.set("partA", partA);
10 flow.set("partB", partB);
11 flow.set("partC", partC);
12
13 return msg;

```

- The code above in the “On Message” block of the function node executes whenever the **http in** node is triggered by a new incoming POST request. As was done for the product stock in the previous section, it processes the incoming data for updating the part inventory values and saves them back in the flow variables.
- Finally, we again connect an “**http response**” node to the output of our function node:



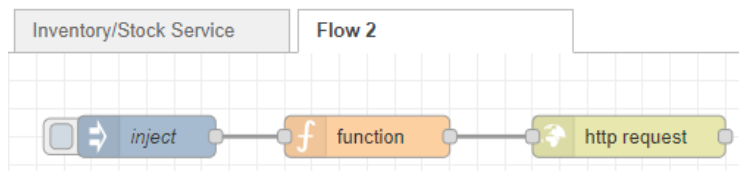
After completing both the GET and POST methods of our web service API in this module and in the previous module, the final flow should look like the following:



3. Testing the API Endpoints

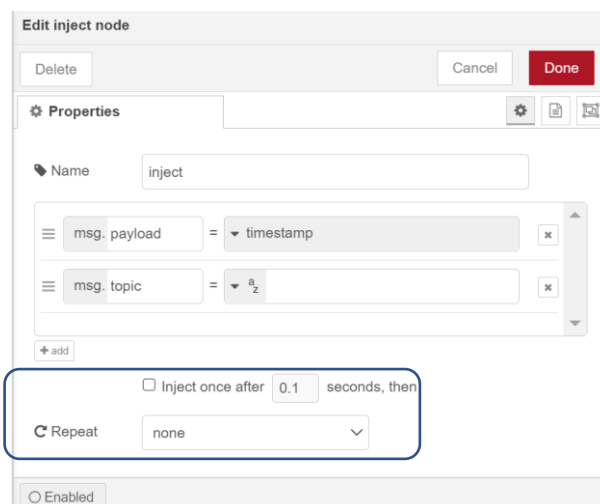
We have actually completed the implementation of our web service, but in this section we will try to test the API functionality, before using it in the next modules.

For this, we will add a new flow, which will serve as a simple client that uses our web API:



Here, we use an *inject* node to trigger API GET/POST requests, then a *function* node to prepare the data for API requests, and finally an **http request** node to transmit the request to our API endpoint.

- For the inject node, leave the node settings as default, i.e., leave the “inject once after ...” field unchecked and set the Repeat to “None”:



Properties

Name: inject

msg. payload = timestamp

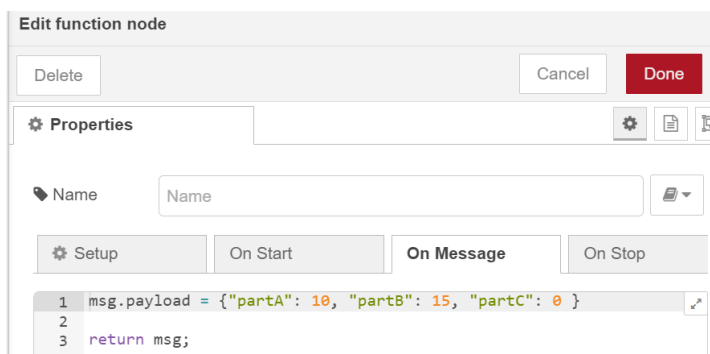
msg. topic = a_z

☐ Inject once after 0.1 seconds, then

☒ Repeat: none

☐ Enabled

- The function node creates a JSON string to update the parts inventory, and sends this out to the next node (http response) to be sent to our API endpoint with the POST method:



Properties

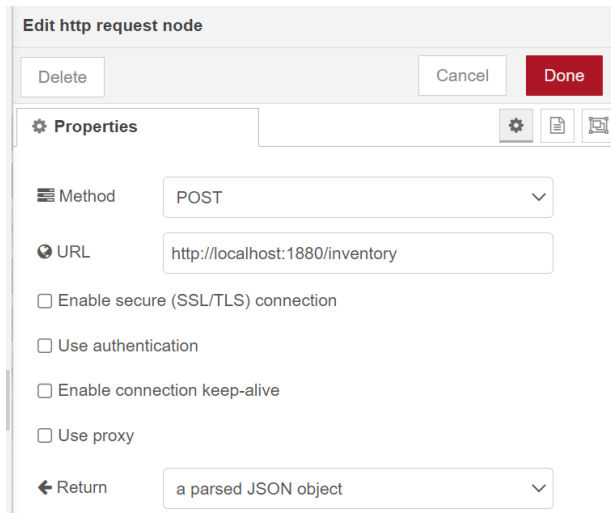
Name: Name

Setup | On Start | **On Message** | On Stop

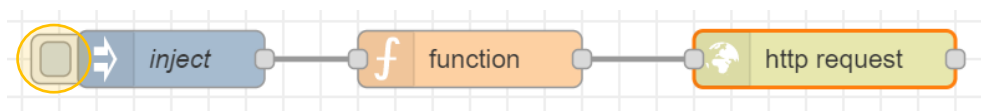
```

1 msg.payload = {"partA": 10, "partB": 15, "partC": 0 }
2
3 return msg;
  
```

- The aim of the JSON string above is to tell our inventory API that it wants to add 10 new units of *Part A* and 15 new units of *Part B* to the parts inventory.
- Finally we connect the output of the function node to the input of an http request node, which is set to use the POST method and the URL “http://localhost:1880/inventory”.

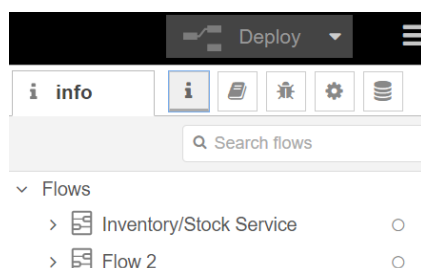



Once we deploy this flow, we can click on the **trigger node's button** (the square on the left side of the trigger node), as shown below, in order to initiate a POST method request from this client to our 'inventory' API endpoint.



In order to see the effect of this on our web service, we will switch back to the first flow (“Inventory/Stock Service” tab), and then use the “context data” feature of Node-RED.

Under the Deploy button of Node-RED, notice the various icons for different views. The default view is the “Info” page:

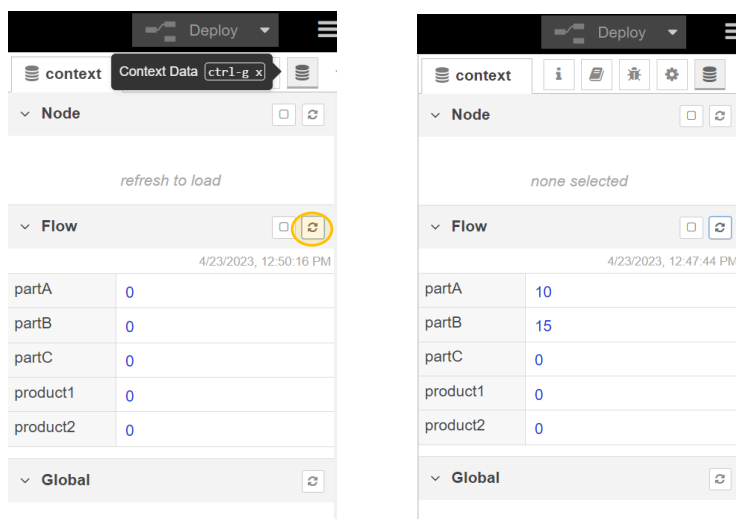


The right-most icon  is for the “**Context Data**”, which allows us to observe the values of application variables. These values can be at the *node*, *flow* or *global* levels.

We click on the “Context Data” view. (Make sure that you are in the first flow for the web service and not in the second flow for the simple client we just implemented.)

Then find the “**flow**” section in this context data view and click on the “refresh” icon next to it. It should then show you the flow variables for parts and products, all initialized to zero (as shown on the **left figure below**).

Now switch to second flow (**Flow 2**) and click on the **inject node’s button** to trigger a POST request and then come back to the first flow to observe the flow variable values by clicking again on the “refresh” icon. As shown on the **right figure below**, the inventory is updated!



Variable	Initial Value (Left)	Updated Value (Right)
partA	0	10
partB	0	15
partC	0	0
product1	0	0
product2	0	0

Now, it is your turn to change the function node and the http request node slightly to test both API endpoints of our web service with both GET and POST methods.

Keep in mind that you need to focus only on the client flow when testing the GET method, since this does not change anything on the server side, but retrieves some data. So, in this case, you can use the *Context Data* view on the second flow and not on the first flow, or just use a *debug* node in Node-RED to print out the retrieved values to the debug console.