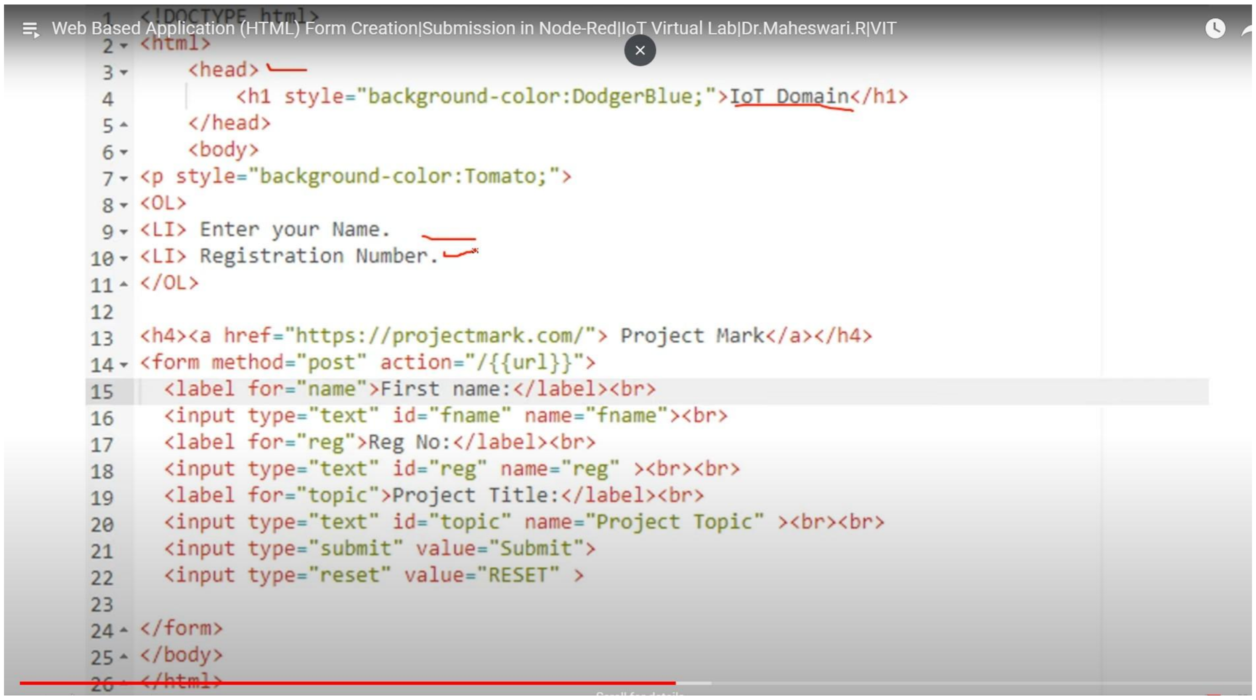
Node-red - elastic search -nodejs


Web Based Application (HTML) Form Creation|Submission in Node-Red|IoT Virtual Lab|Dr.Maheswari.R|VIT

```html
<!DOCTYPE html>
<html>
    <head>
        <h1 style="background-color:DodgerBlue;">IoT Domain</h1>
    </head>
    <body>
<p style="background-color:Tomato;">
<OL>
<LI> Enter your Name.
<LI> Registration Number.
</OL>

<h4><a href="https://projectmark.com/"> Project Mark</a></h4>
<form method="post" action="/{{url}}">
  <label for="name">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <label for="reg">Reg No:</label><br>
  <input type="text" id="reg" name="reg" ><br><br>
  <label for="topic">Project Title:</label><br>
  <input type="text" id="topic" name="Project Topic" ><br><br>
  <input type="submit" value="Submit">
  <input type="reset" value="RESET" >

</form>
</body>
</html>
```

CREATE TABLE RANDOMNUM(TIMESTAMP INT PRIMARY KEY NOT NULL,VALUE INT NOT NULL,BOOL INT NOT NULL)

var randomnum=Math.round(Math.random()*100);
var largebool=(randomnum>50):1:0;
var newmsg={

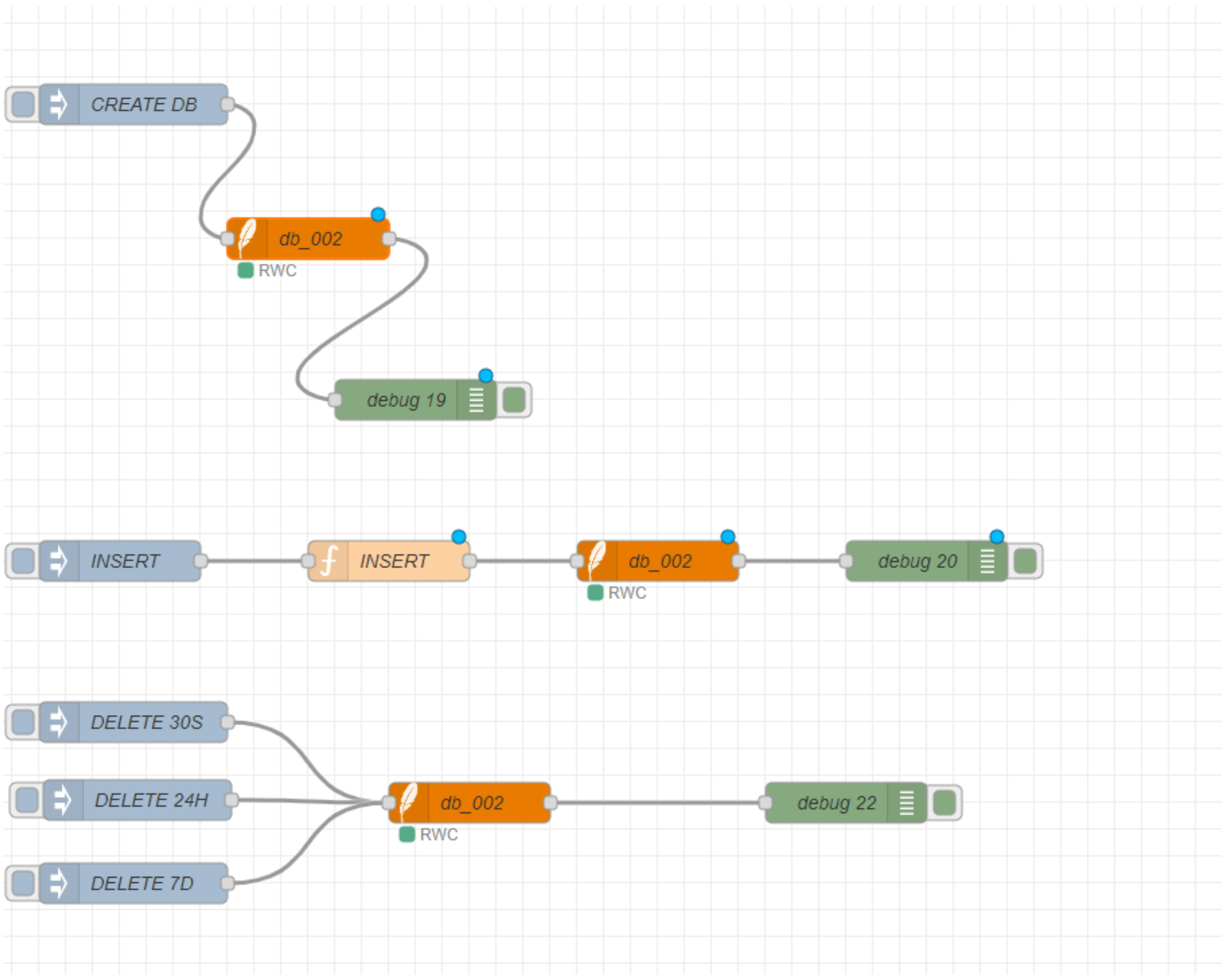
"topic":"INSERT INTO RANDOMNUM VALUES("+msg.payload+", "+randomnum+", "+largebool+")"

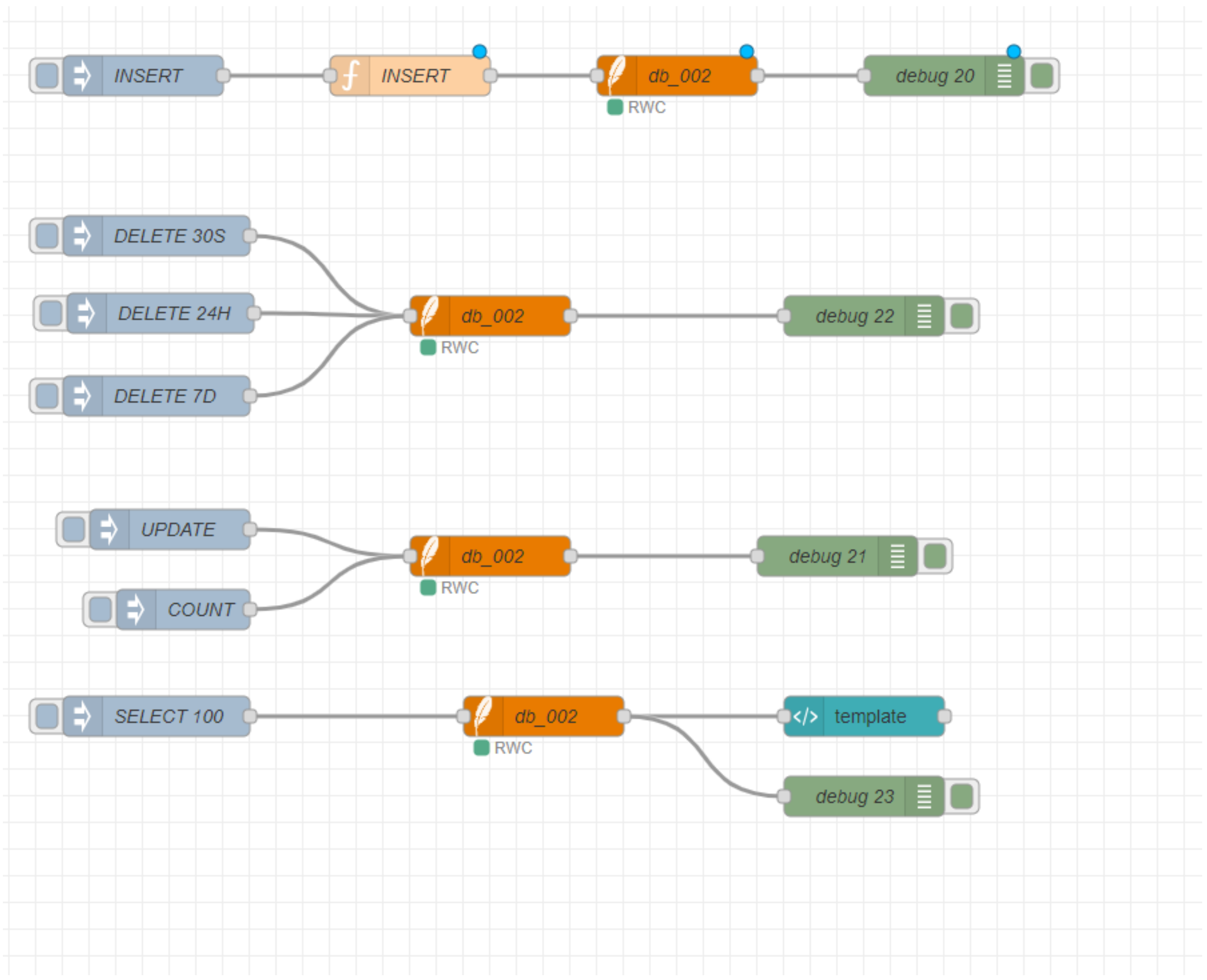
}
return newmsg;

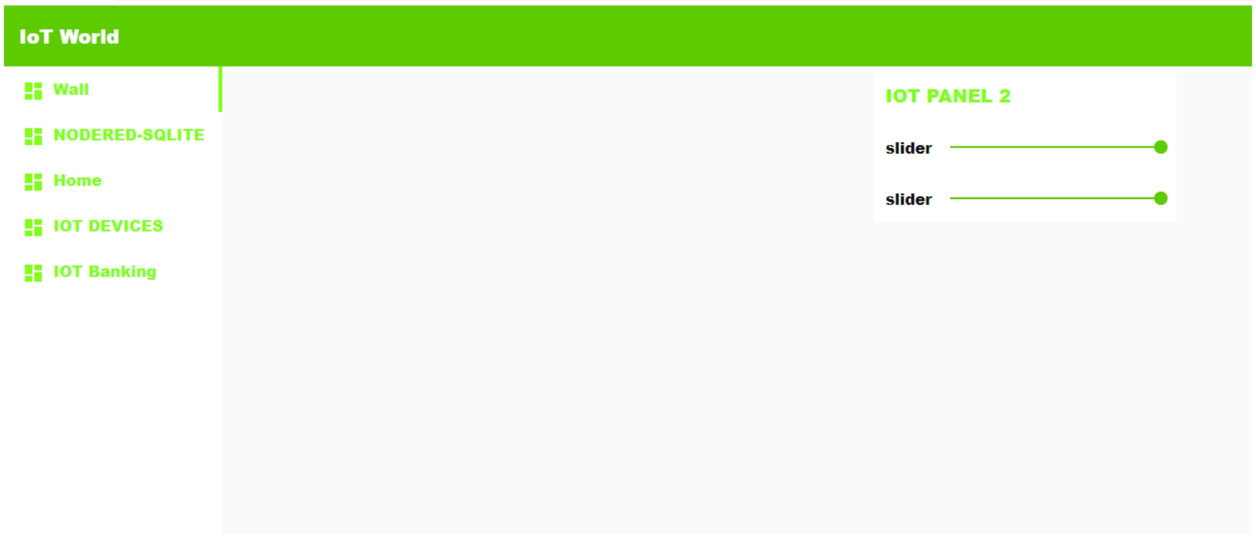
DELETE FROM RANDOMNUM WHERE TIMESTAMP <= strftime('%s','now','-30 seconds')+1000;
DELETE FROM RANDOMNUM WHERE TIMESTAMP <= strftime('%s','now','-24 hours')+1000;
DELETE FROM RANDOMNUM WHERE TIMESTAMP <= strftime('%s','now','-7 days')+1000;

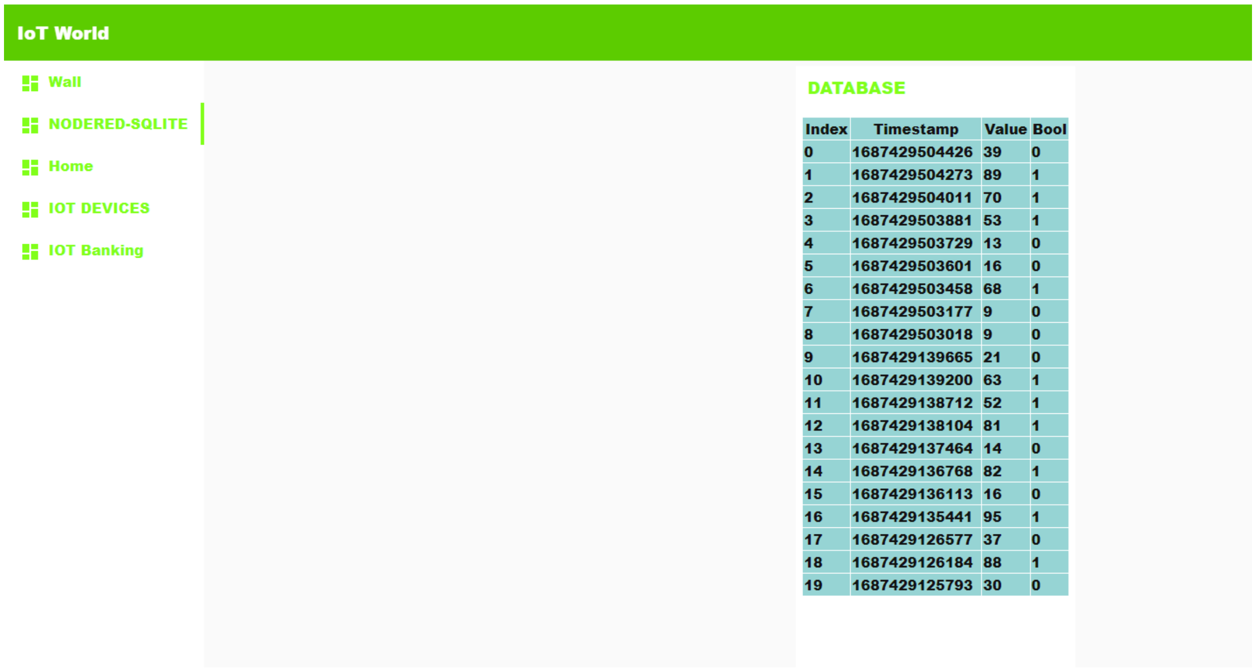
```
UPDATE RANDOMNUM SET BOOL = 1 WHERE VALUE >80 AND BOOL =0;

SELECT COUNT(*) FROM ROUNDNUM;

SELECT * FROM RANDOMNUM ORDER BY TIMESTAMP DESC LIMIT 100;
```

```
<table style="width:100%">
 <tr>
  <th>Index</th>
  <th>Timestamp</th>
  <th>Value</th>
  <th>Bool</th>
 </tr>
 <tr ng-repeat="x in msg.payload | limitTo:20">
  <td>{{$index}}</td>
  <td>{{msg.payload[$index].TIMESTAMP}}</td>
  <td>{{msg.payload[$index].VALUE}}</td>
  <td>{{msg.payload[$index].BOOL}}</td>
 </tr>
</table>
```

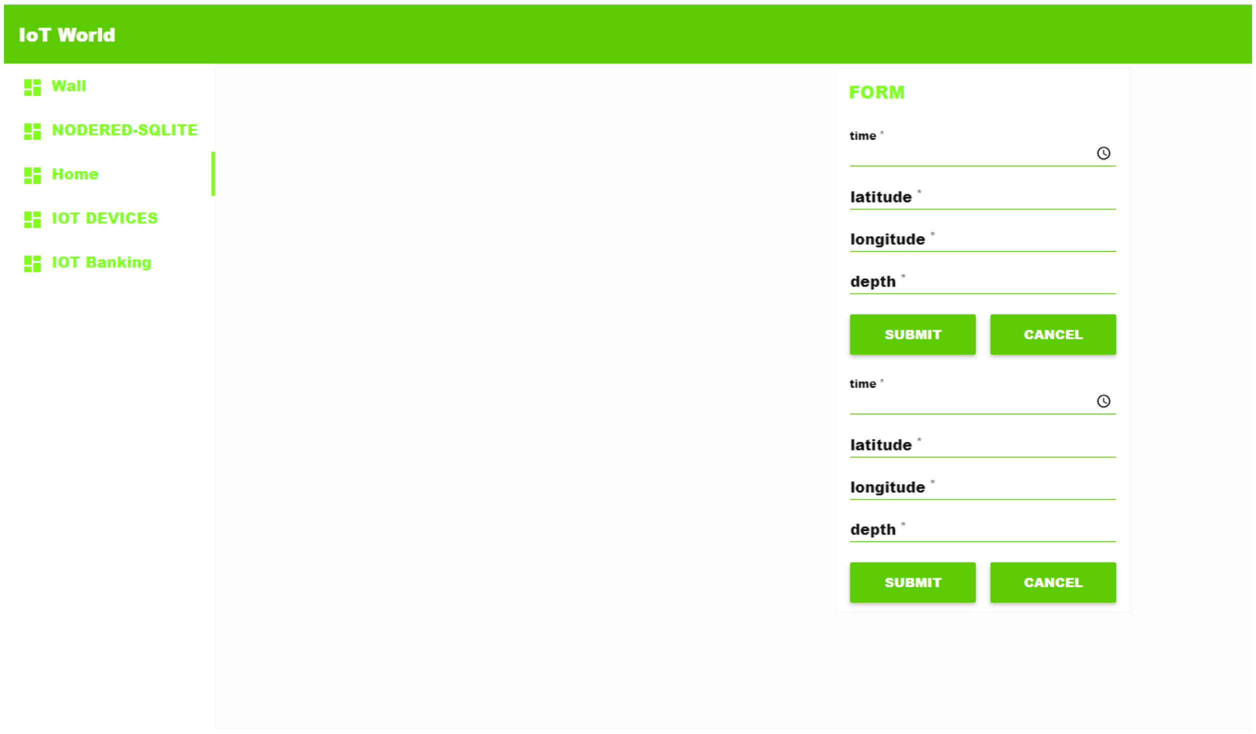

CREATE DB
db_002
RWC
debug 19
INSERT
INSERT
db_002
RWC
debug 20
DELETE 30S
DELETE 24H
DELETE 7D
db_002
RWC
debug 22

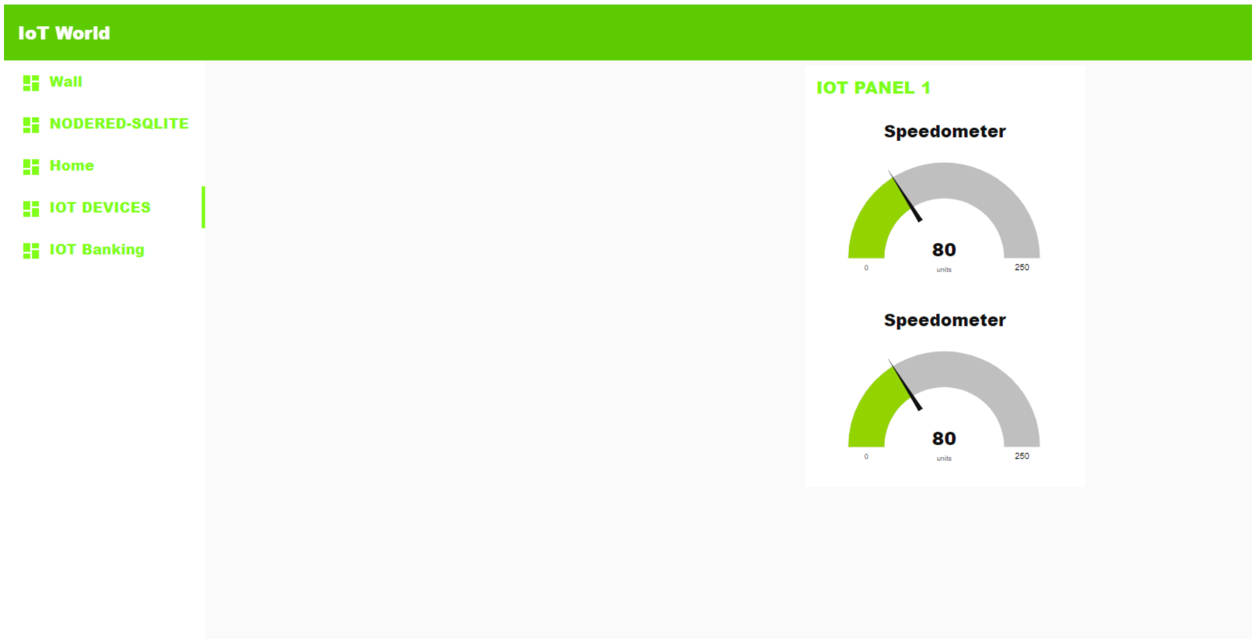

INSERT
INSERT
db_002
RWC
debug 20
DELETE 30S
DELETE 24H
DELETE 7D
db_002
RWC
debug 22
UPDATE
COUNT
db_002
RWC
debug 21
SELECT 100
db_002
RWC
template
debug 23

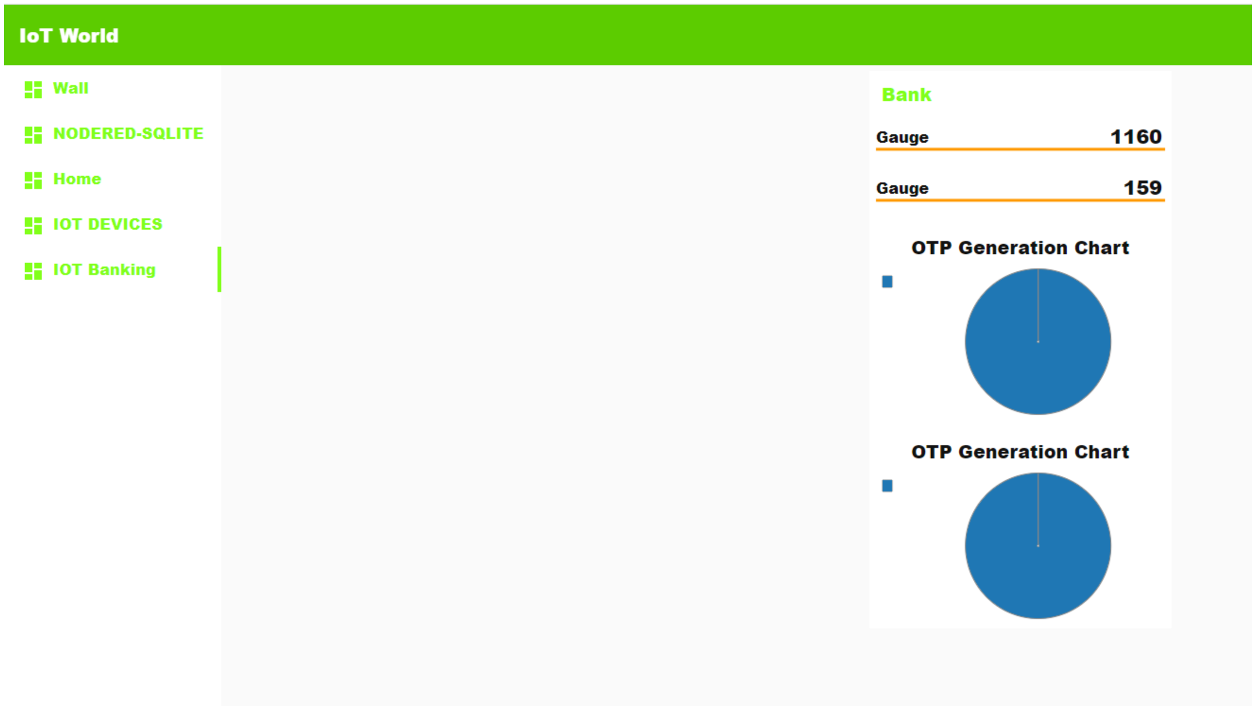


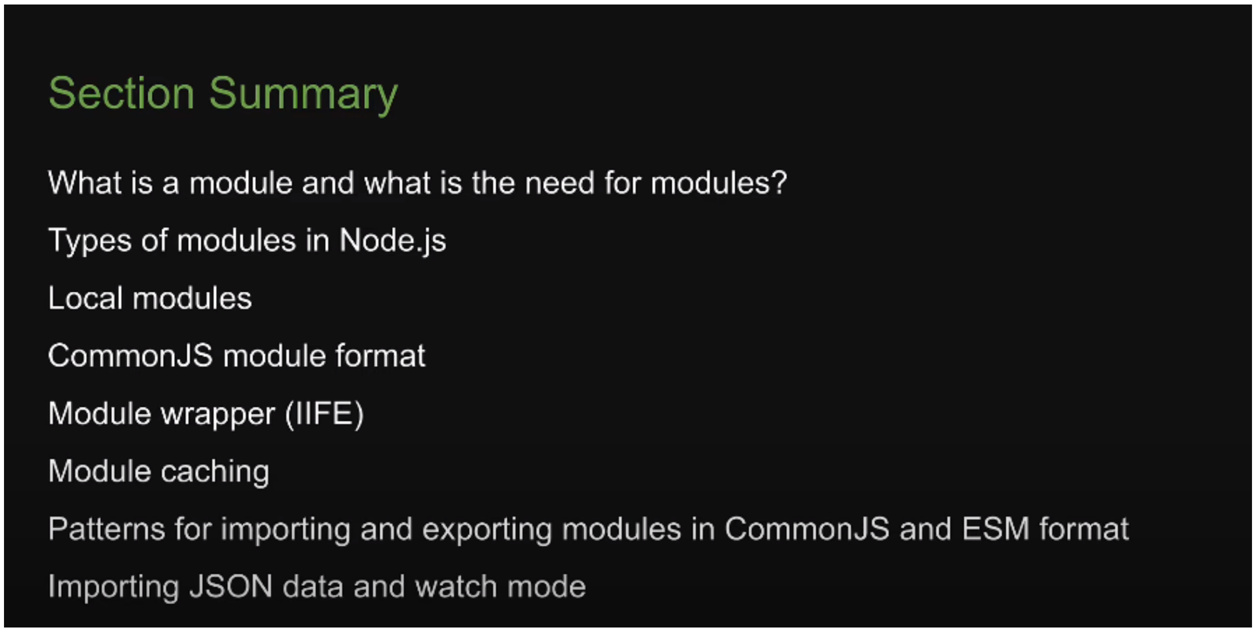


Index	Timestamp	Value	Bool
0	1687429504426	39	0
1	1687429504273	89	1
2	1687429504011	70	1
3	1687429503881	53	1
4	1687429503729	13	0
5	1687429503601	16	0
6	1687429503458	68	1
7	1687429503177	9	0
8	1687429503018	9	0
9	1687429139665	21	0
10	1687429139200	63	1
11	1687429138712	52	1
12	1687429138104	81	1
13	1687429137464	14	0
14	1687429136768	82	1
15	1687429136113	16	0
16	1687429135441	95	1
17	1687429126577	37	0
18	1687429126184	88	1
19	1687429125793	30	0


IoT World
Wall
NODERED-SQLITE
Home
IOT DEVICES
IOT Banking
FORM
time *
latitude *
longitude *
depth *
SUBMIT
CANCEL
time *
latitude *
longitude *
depth *
SUBMIT
CANCEL


IoT World
Wall
NODERED-SQLITE
Home
IOT DEVICES
IOT Banking
IOT PANEL 1
Speedometer
80
units
0
250
Speedometer
80
units
0
250


IoT World
Wall
NODERED-SQLITE
Home
IOT DEVICES
IOT Banking
Bank
Gauge
1160
Gauge
159
OTP Generation Chart
OTP Generation Chart



Section Summary

What is a module and what is the need for modules?

Types of modules in Node.js

Local modules

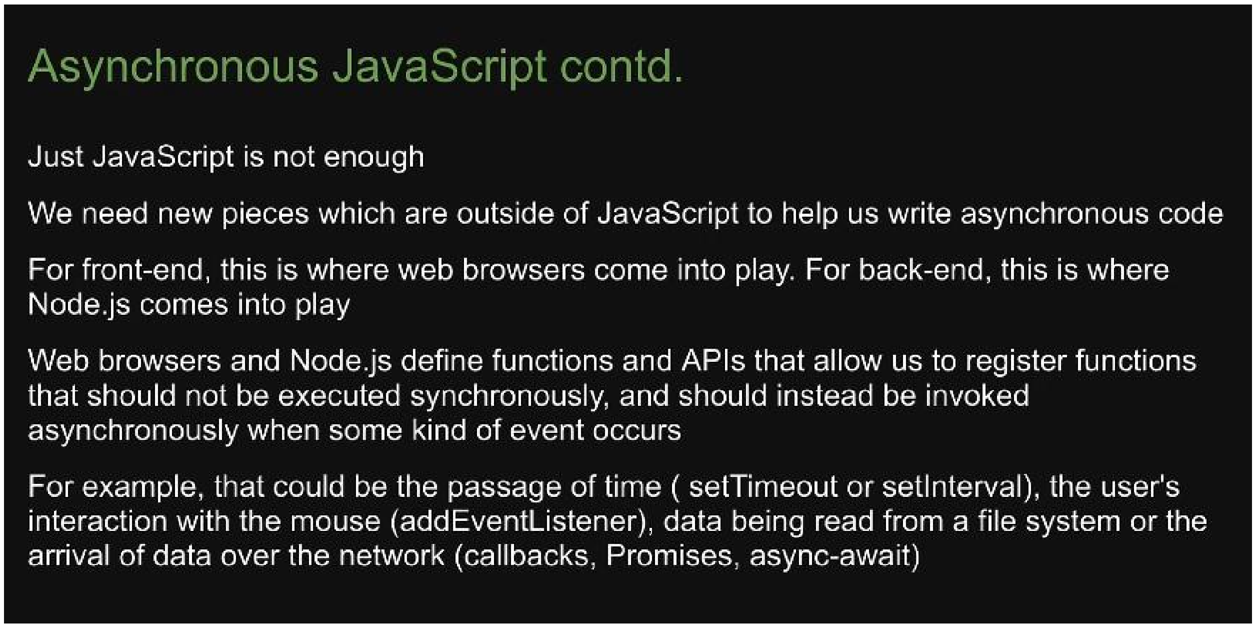
CommonJS module format

Module wrapper (IIFE)

Module caching

Patterns for importing and exporting modules in CommonJS and ESM format

Importing JSON data and watch mode



Asynchronous JavaScript contd.

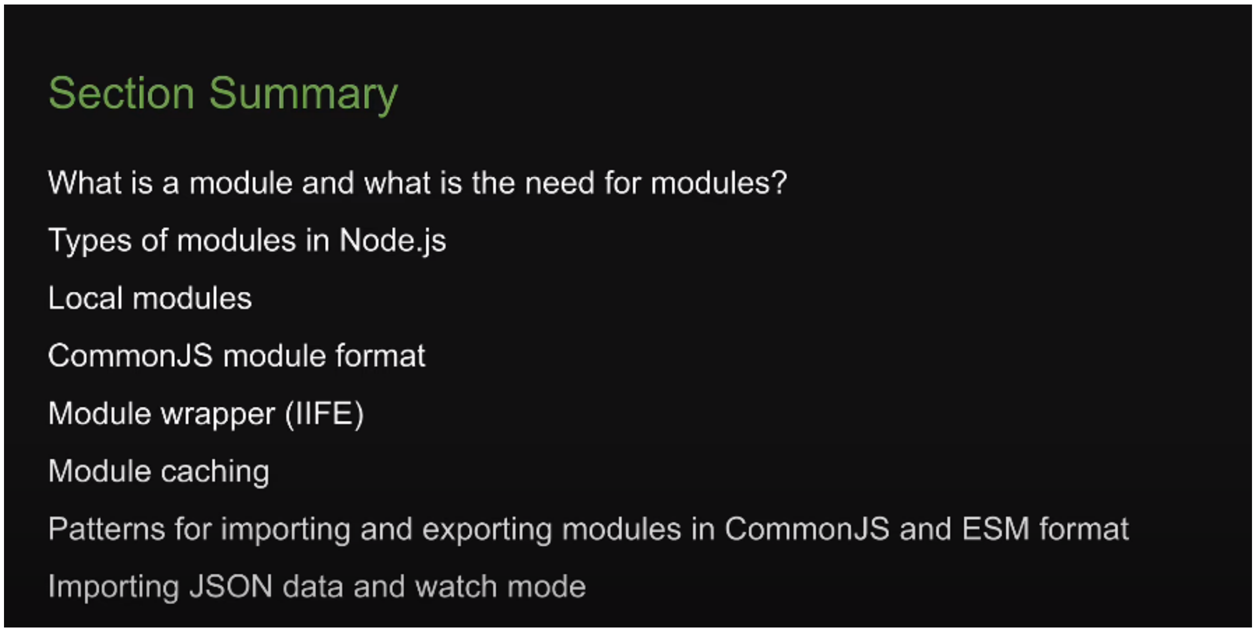
Just JavaScript is not enough

We need new pieces which are outside of JavaScript to help us write asynchronous code

For front-end, this is where web browsers come into play. For back-end, this is where Node.js comes into play

Web browsers and Node.js define functions and APIs that allow us to register functions that should not be executed synchronously, and should instead be invoked asynchronously when some kind of event occurs

For example, that could be the passage of time (setTimeout or setInterval), the user's interaction with the mouse (addEventListener), data being read from a file system or the arrival of data over the network (callbacks, Promises, async-await)



Section Summary

What is a module and what is the need for modules?

Types of modules in Node.js

Local modules

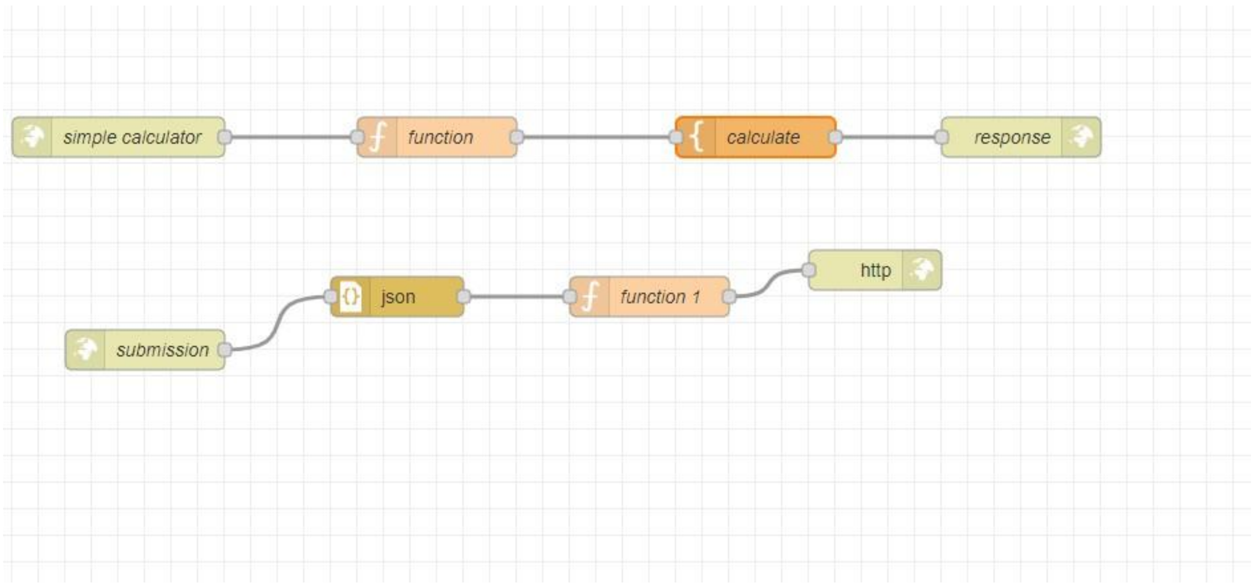
CommonJS module format

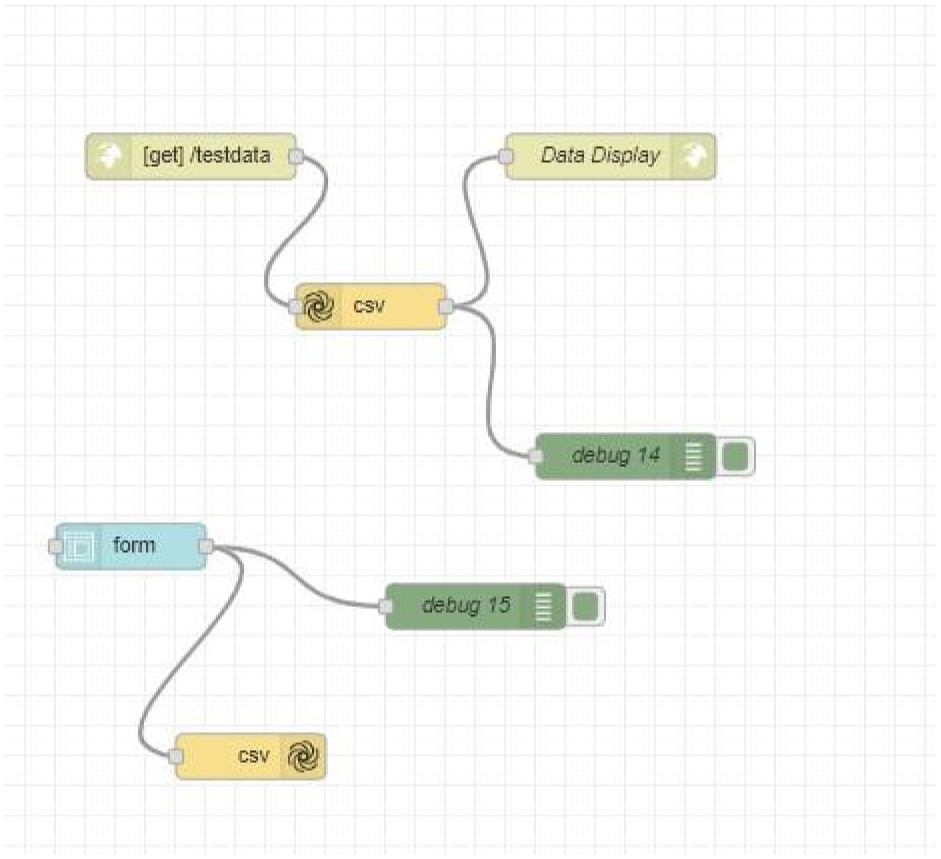
Module wrapper (IIFE)

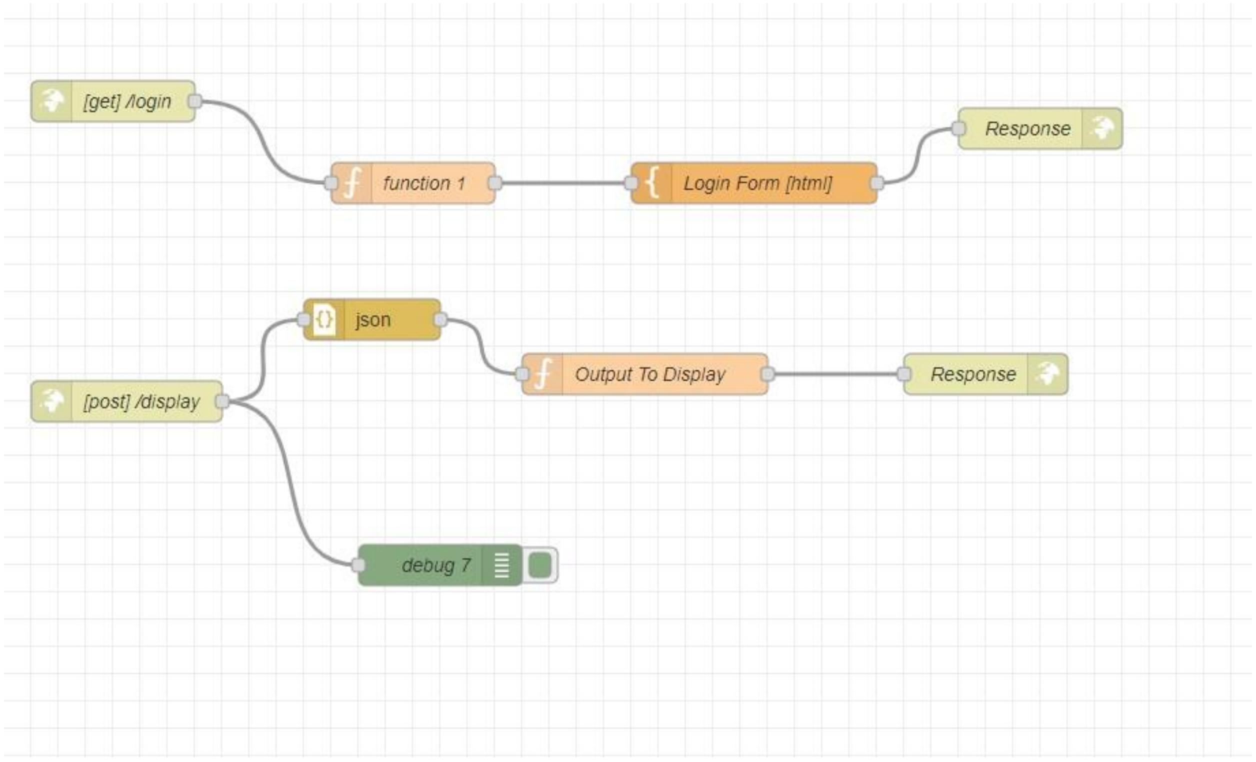
Module caching

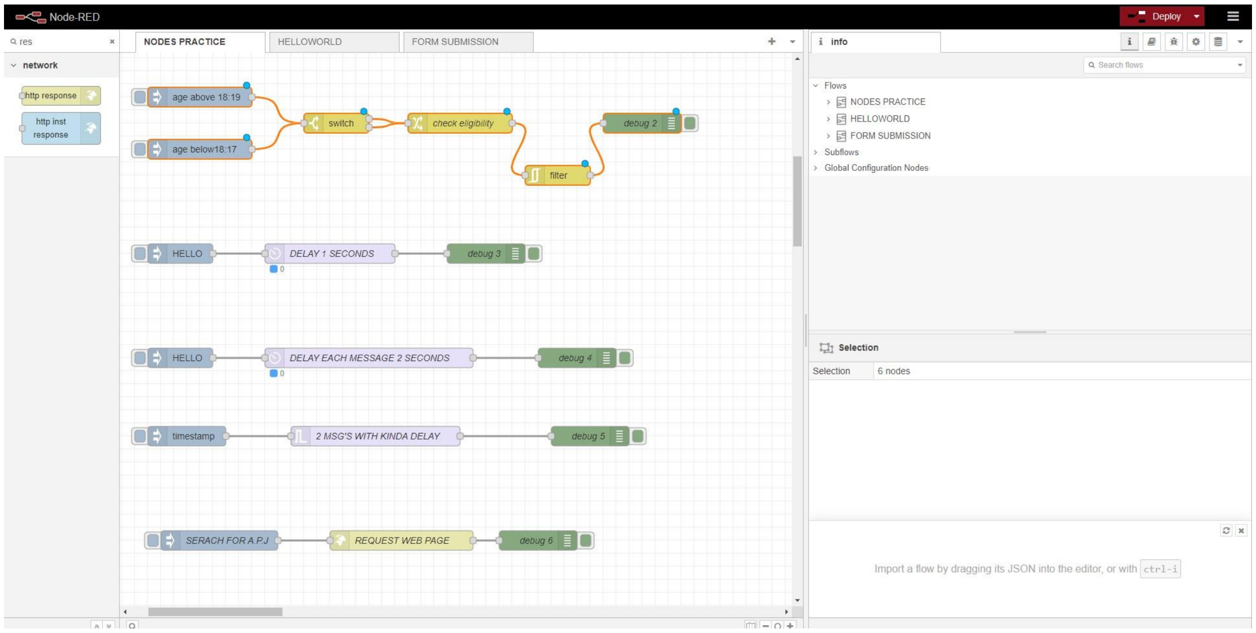
Patterns for importing and exporting modules in CommonJS and ESM format

Importing JSON data and watch mode

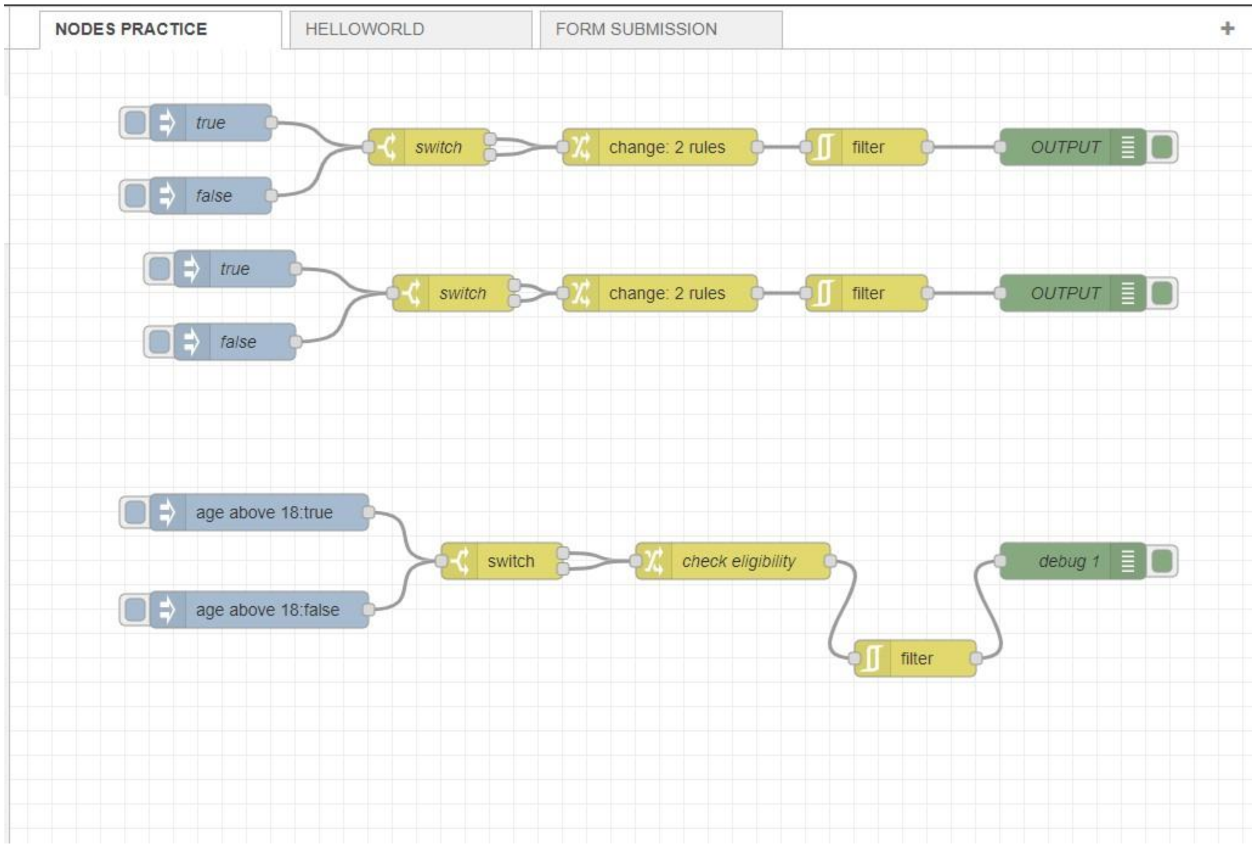



[get] /testdata
Data Display
csv
debug 14
form
debug 15
csv


[get] /login
function 1
Login Form [html]
Response
json
[post] /display
Output To Display
Response
debug 7



Import a flow by dragging its JSON into the editor, or with `ctrl-i`



About featured snippets • Feedback

GitHub
https://github.com › mpolinowski › nodered-alasql

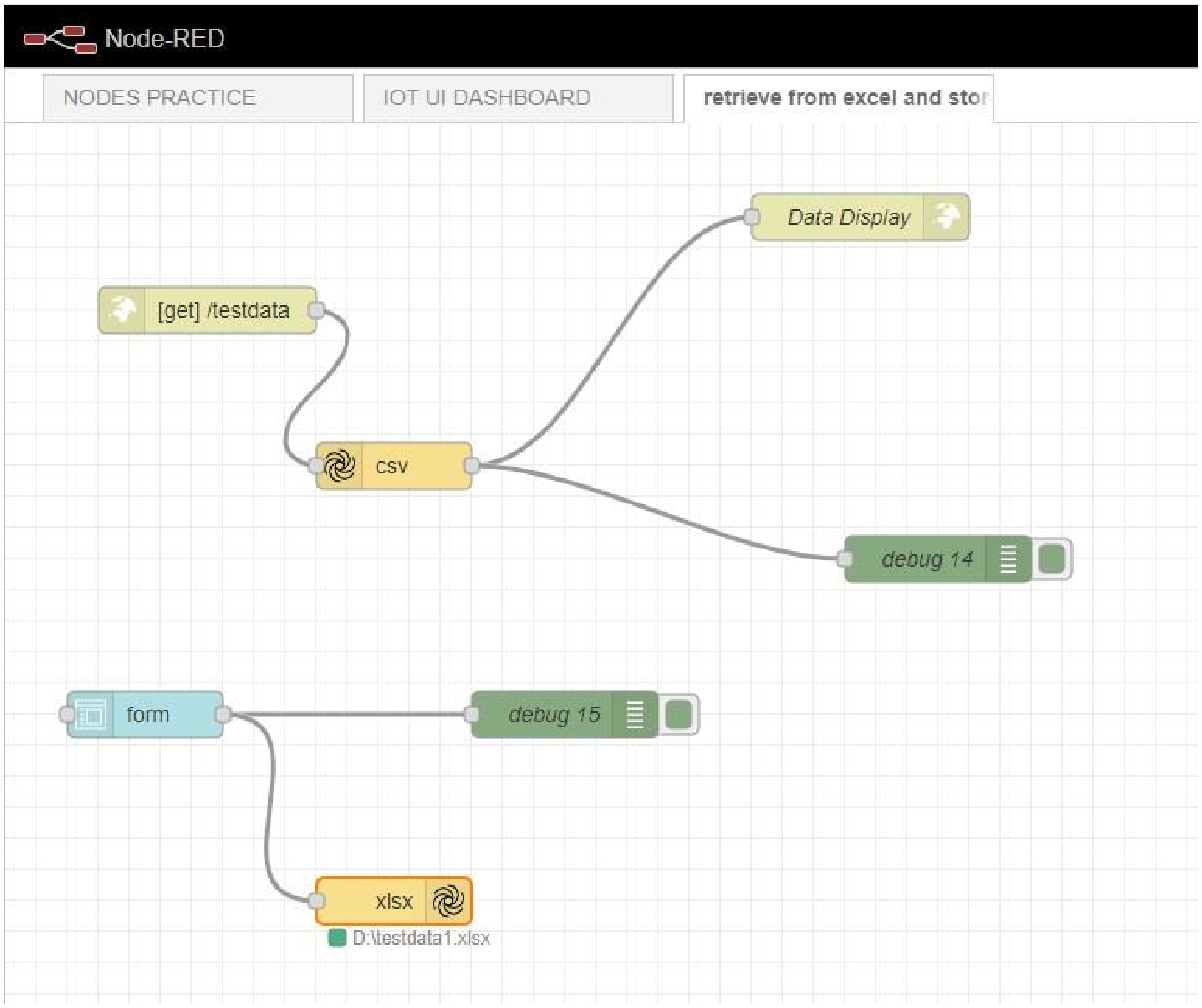
Working with AlaSQL in Node-RED

It can be added to **Node**-RED by installing the **node-red-contrib-alasql node**. The following is a basic flow that adds DROP, CREATE, INSERT, ...

People also ask

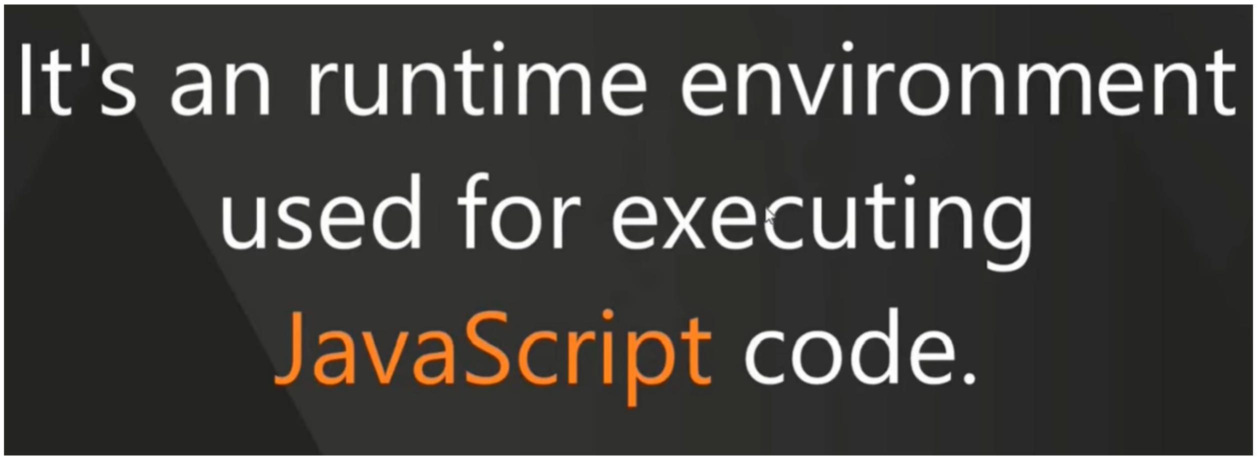
What is the purpose of Node-RED?

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.

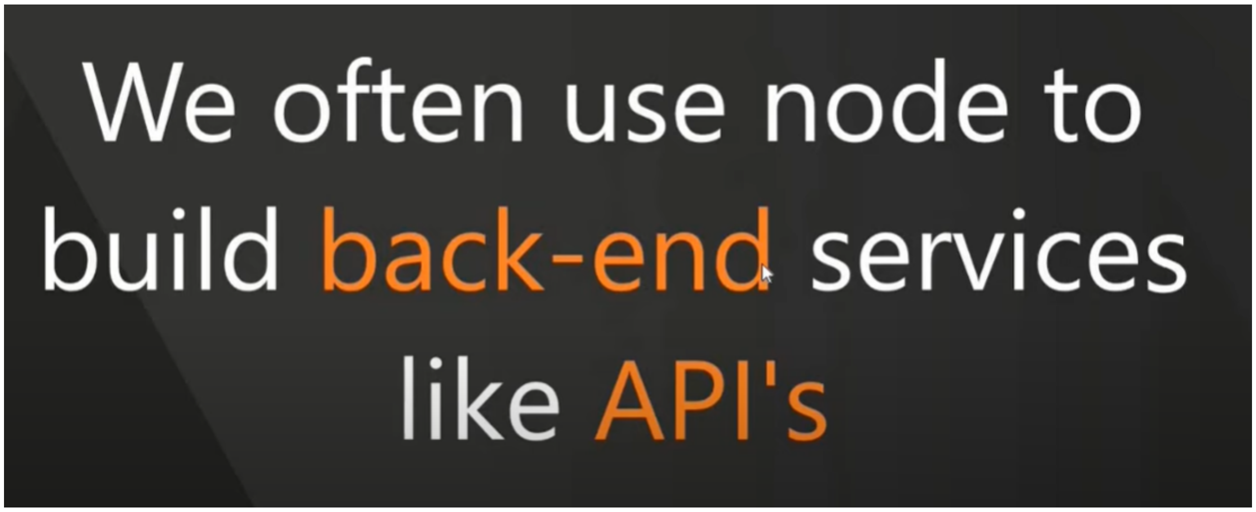


Senior anna GFG akshith44reddy@gmail.com

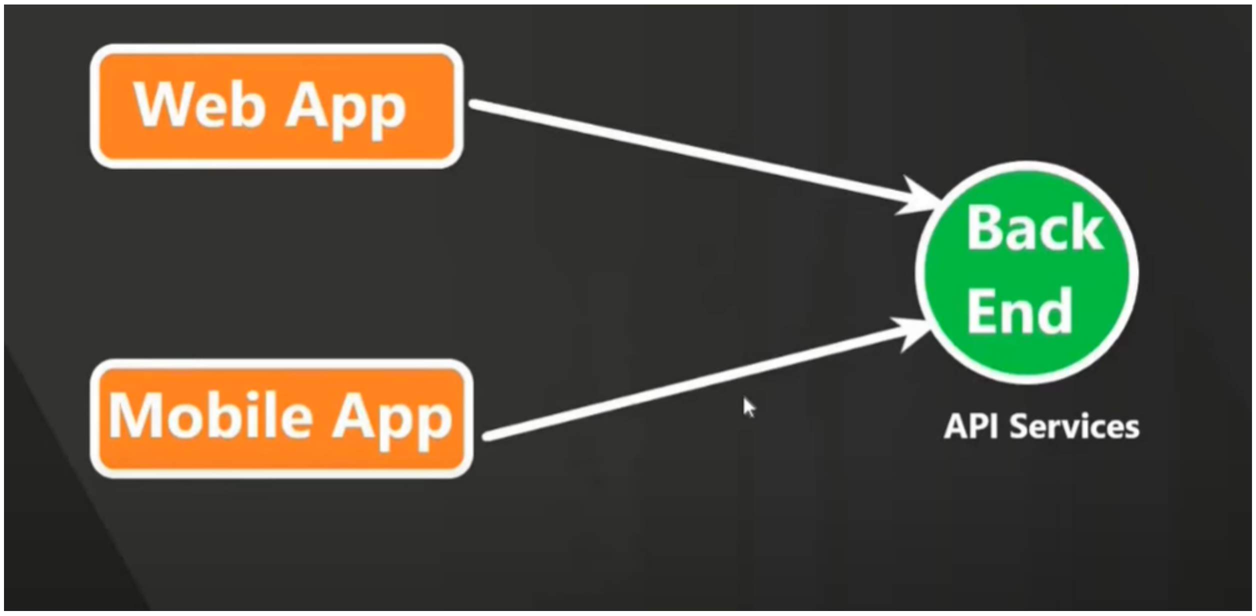
NODE.JS

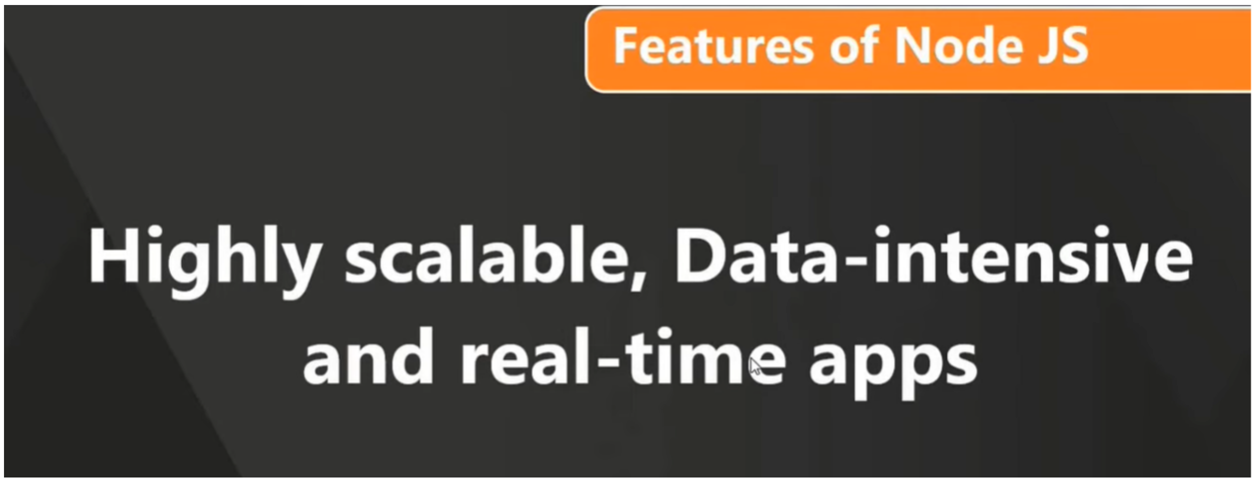


It's an runtime environment used for executing JavaScript code.



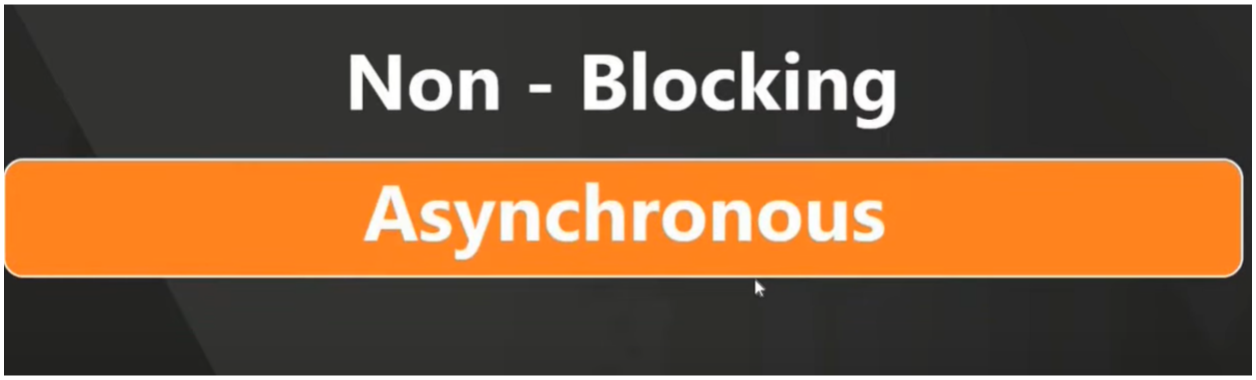
We often use node to build back-end services like API's





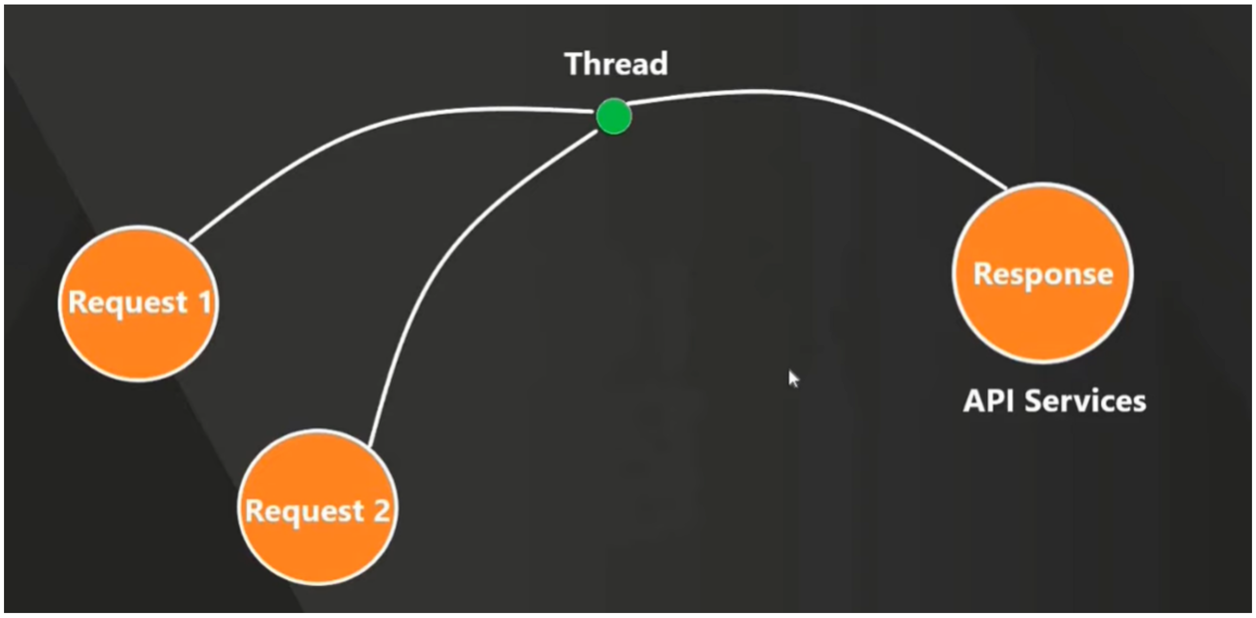
Features of Node JS

Highly scalable, Data-intensive and real-time apps



Non - Blocking

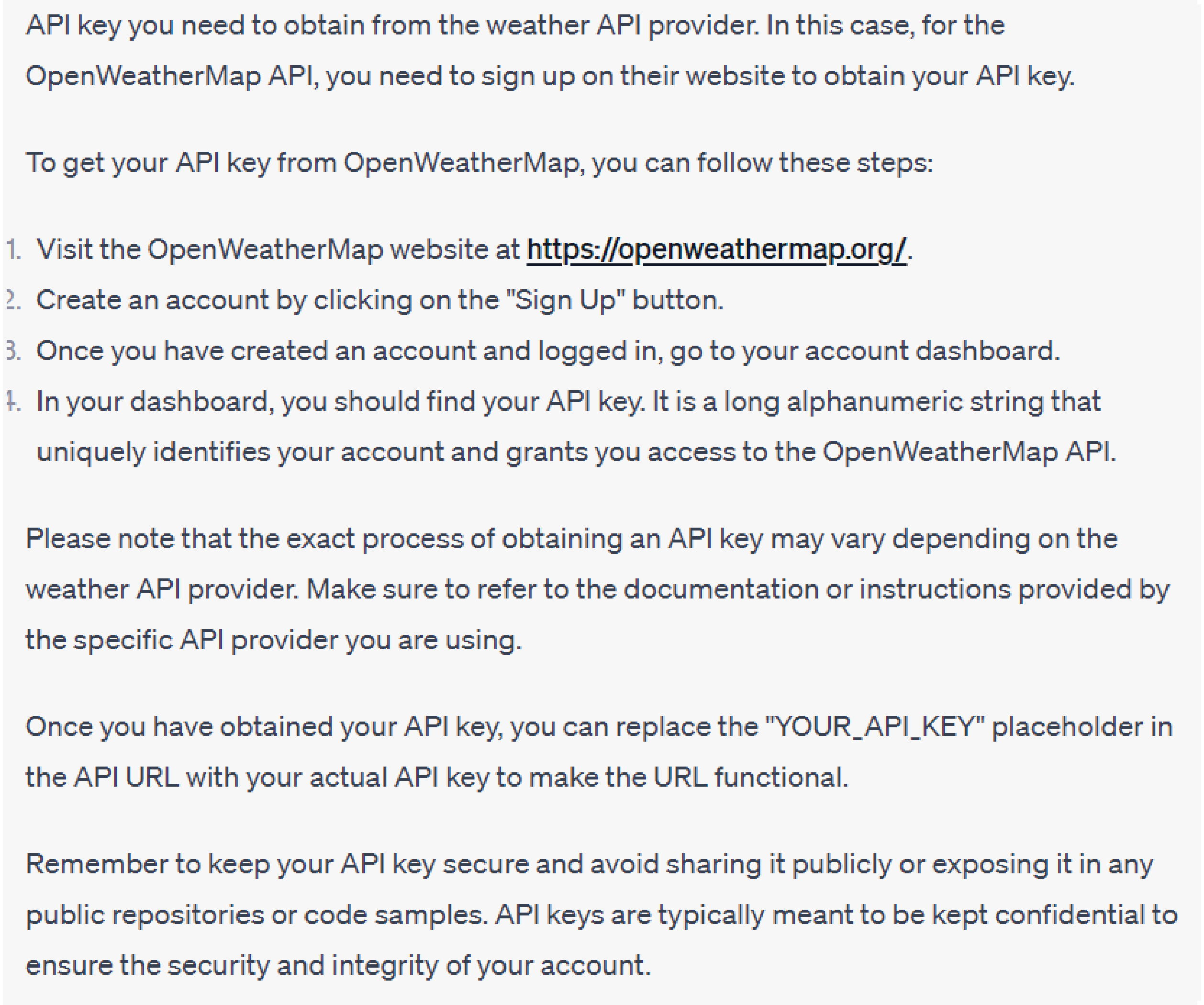
Asynchronous



Collecting weather data continuously into text files using node red?

Node-RED is a visual programming tool that allows you to create flows by connecting nodes together. You can use Node-RED to fetch weather data from an API and store it in text files on your local system or a remote server.

Here's a step-by-step guide on how to collect weather data continuously into text files using Node-RED:



API key you need to obtain from the weather API provider. In this case, for the OpenWeatherMap API, you need to sign up on their website to obtain your API key.

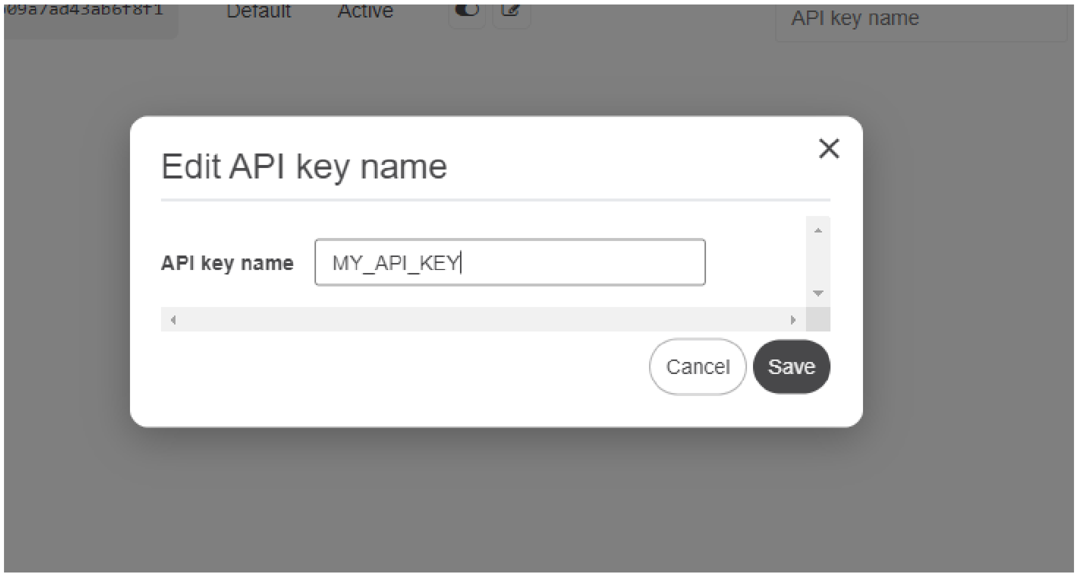
To get your API key from OpenWeatherMap, you can follow these steps:

1. Visit the OpenWeatherMap website at https://openweathermap.org/.
2. Create an account by clicking on the "Sign Up" button.
3. Once you have created an account and logged in, go to your account dashboard.
4. In your dashboard, you should find your API key. It is a long alphanumeric string that uniquely identifies your account and grants you access to the OpenWeatherMap API.

Please note that the exact process of obtaining an API key may vary depending on the weather API provider. Make sure to refer to the documentation or instructions provided by the specific API provider you are using.

Once you have obtained your API key, you can replace the "YOUR_API_KEY" placeholder in the API URL with your actual API key to make the URL functional.

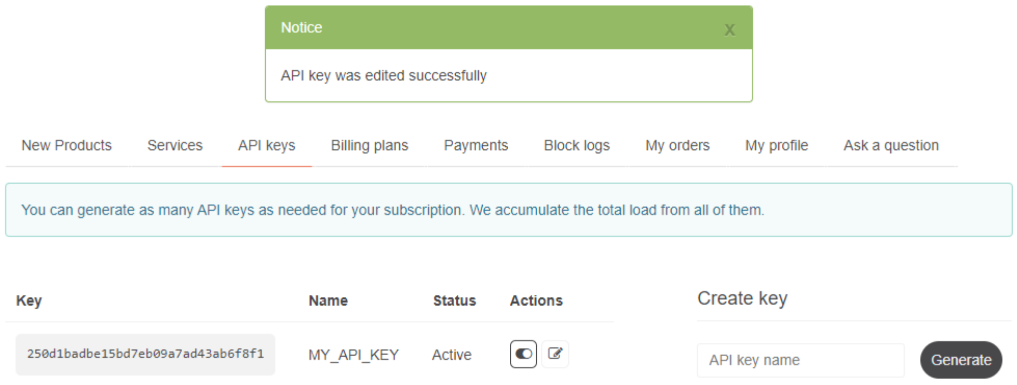
Remember to keep your API key secure and avoid sharing it publicly or exposing it in any public repositories or code samples. API keys are typically meant to be kept confidential to ensure the security and integrity of your account.



Edit API key name

API key name: MY_API_KEY

`250d1badbe15bd7eb09a7ad43ab6f8f1`



Notice

API key was edited successfully

You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them.

Key	Name	Status	Actions
250d1badbe15bd7eb09a7ad43ab6f8f1	MY_API_KEY	Active	

https://api.openweathermap.org/data/2.5/weather?q=New%20York&appid=250d1badbe15bd7eb09a7ad43ab6f8f1

Working url
https://samples.openweathermap.org/data/2.5/weather?q=London,uk&appid=b6907d289e10d714a6e88b30761fae22
Get json data in browser and copy it.

Goto

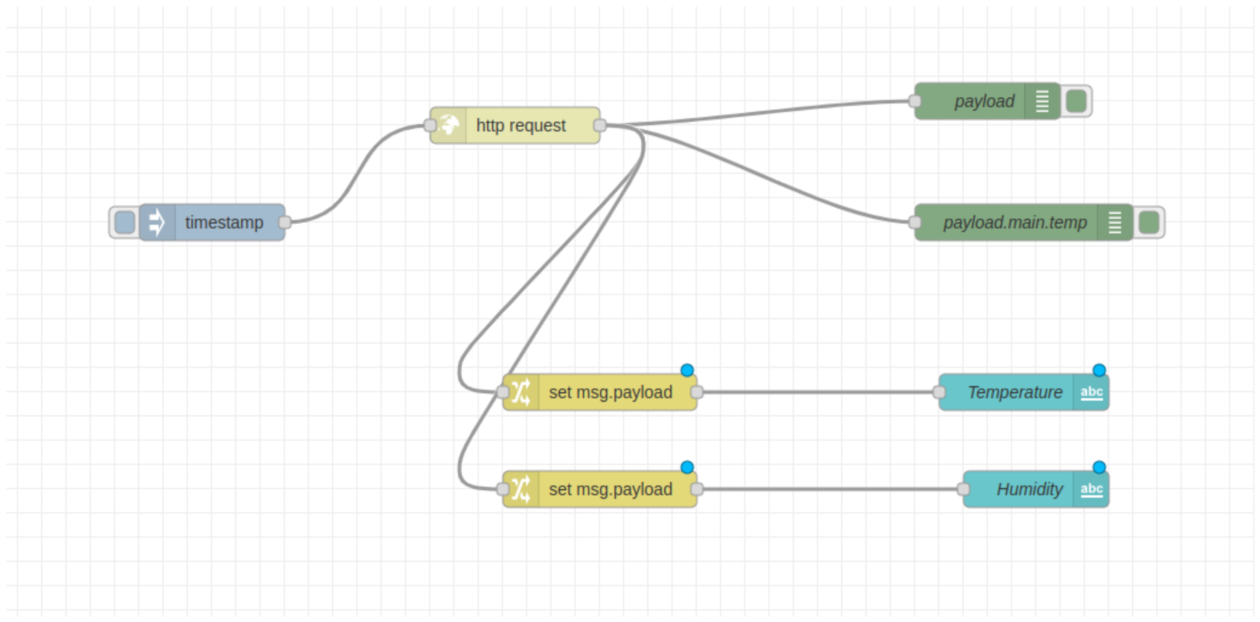
https://jsonformatter.curiousconcept.com/

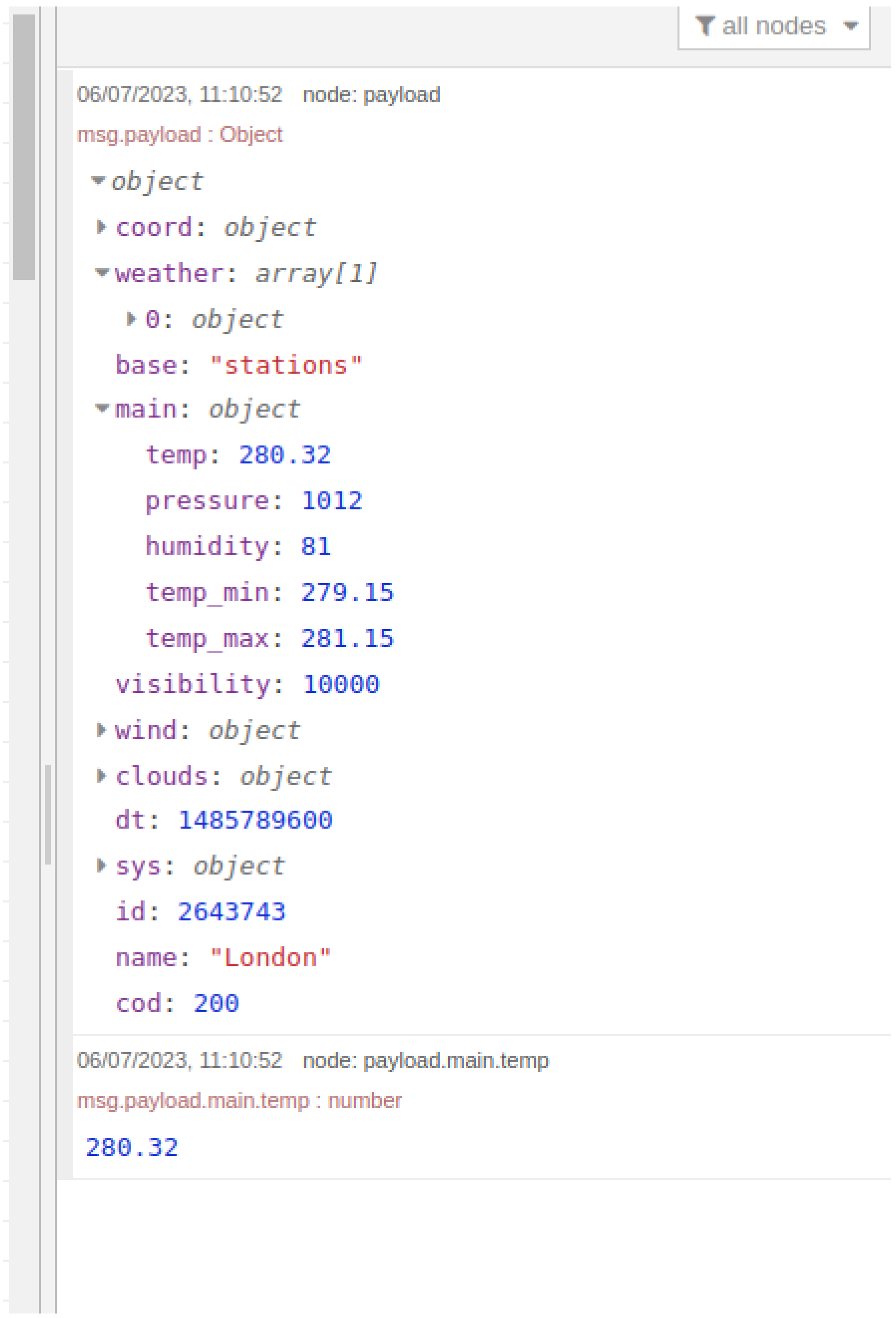
{"coord":{"lon":-0.13,"lat":51.51},"weather":[{"id":300,"main":"Drizzle","description":"light intensity drizzle","icon":"09d"}],"base":"stations","main":{"temp":280.32,"pressure":1012,"humidity":81,"temp_min":279.15,"temp_max":281.15},"visibility":10000,"wind":{"speed":4.1,"deg":80},"clouds":{"all":90},"dt":1485789600,"sys":{"type":1,"id":5091,"message":0.0103,"country":"GB","sunrise":1485762037,"sunset":1485794875},"id":2643743,"name":"London","cod":200}

Paste it here.

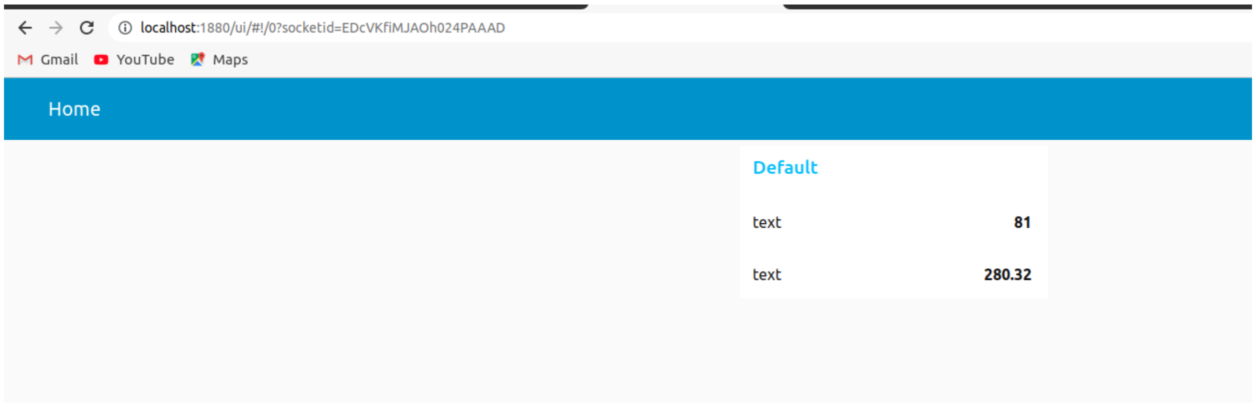
Json beautifier and validator

```
{
  "coord":{
    "lon":-0.13,
    "lat":51.51
  },
  "weather":[
    {
      "id":300,
      "main":"Drizzle",
      "description":"light intensity drizzle",
      "icon":"09d"
    }
  ],
  "base":"stations",
  "main":{
    "temp":280.32,
    "pressure":1012,
    "humidity":81,
    "temp_min":279.15,
    "temp_max":281.15
  },
  "visibility":10000,
  "wind":{
    "speed":4.1,
    "deg":80
  },
  "clouds":{
    "all":90
  },
  "dt":1485789600,
  "sys":{
    "type":1,
    "id":5091,
    "message":0.0103,
    "country":"GB",
    "sunrise":1485762037,
    "sunset":1485794875
  },
  "id":2643743,
  "name":"London",
  "cod":200
}
```


timestamp
http request
payload
payload.main.temp
set msg.payload
set msg.payload
Temperature
Humidity
abc
abc



```
06/07/2023, 11:10:52   node: payload
msg.payload : Object
▾object
 ▸coord: object
 ▾weather: array[1]
   ▸0: object
  base: "stations"
 ▾main: object
    temp: 280.32
    pressure: 1012
    humidity: 81
    temp_min: 279.15
    temp_max: 281.15
  visibility: 10000
 ▸wind: object
 ▸clouds: object
  dt: 1485789600
 ▸sys: object
  id: 2643743
  name: "London"
  cod: 200
06/07/2023, 11:10:52   node: payload.main.temp
msg.payload.main.temp : number
280.32
```



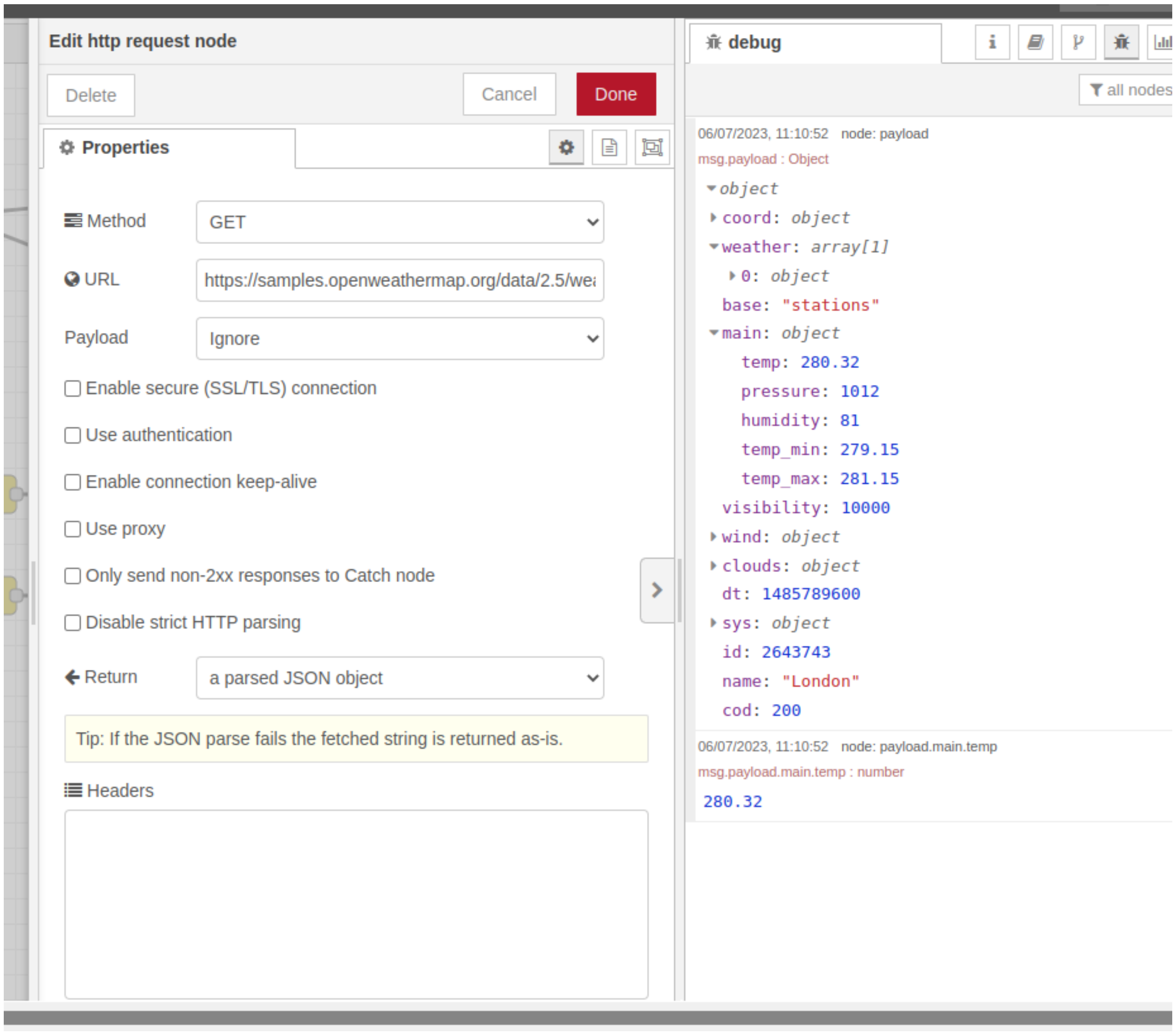
localhost:1880/ui/#!/0?socketid=EDcVKfiMJAOh024PAAAD

Gmail YouTube Maps

Home

Default

text	81
text	280.32



Edit http request node

Delete Cancel Done

Properties

- Method: GET
- URL: https://samples.openweathermap.org/data/2.5/wea
- Payload: Ignore
- ☐ Enable secure (SSL/TLS) connection
- ☐ Use authentication
- ☐ Enable connection keep-alive
- ☐ Use proxy
- ☐ Only send non-2xx responses to Catch node
- ☐ Disable strict HTTP parsing
- Return: a parsed JSON object

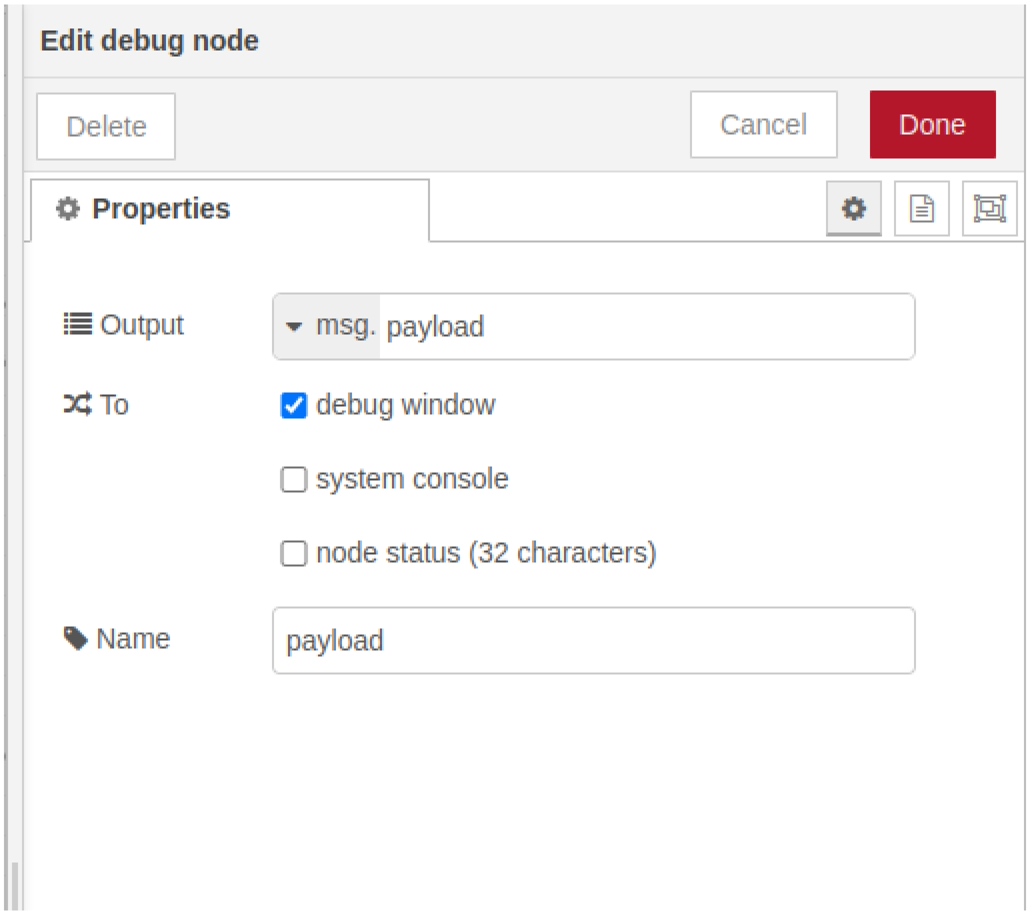
Tip: If the JSON parse fails the fetched string is returned as-is.

Headers

debug

all nodes

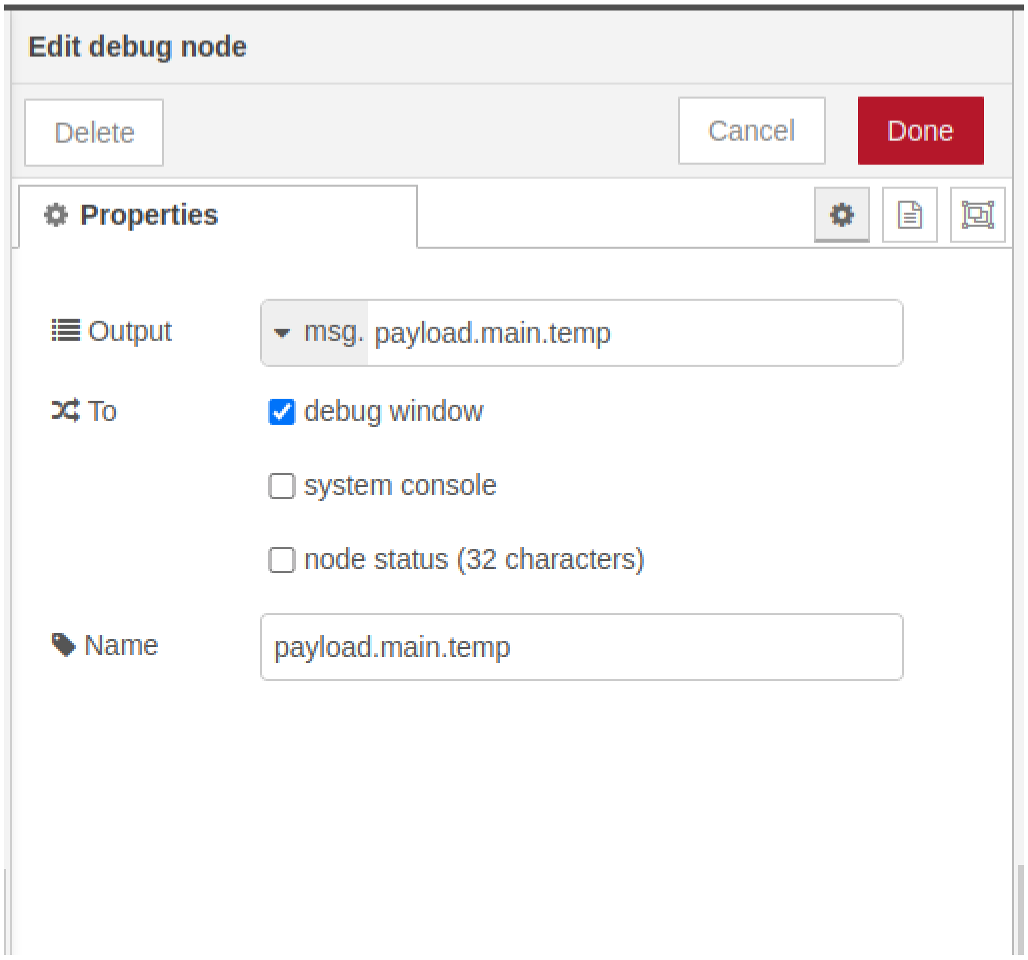
```
06/07/2023, 11:10:52   node: payload
msg.payload : Object
▾object
  ▸coord: object
  ▾weather: array[1]
    ▸0: object
  base: "stations"
  ▾main: object
     temp: 280.32
     pressure: 1012
     humidity: 81
     temp_min: 279.15
     temp_max: 281.15
  visibility: 10000
  ▸wind: object
  ▸clouds: object
  dt: 1485789600
  ▸sys: object
  id: 2643743
  name: "London"
  cod: 200

06/07/2023, 11:10:52   node: payload.main.temp
msg.payload.main.temp : number
280.32
```



Output: msg. payload

To: [✓ debug window] system console node status (32 characters)

Name: payload

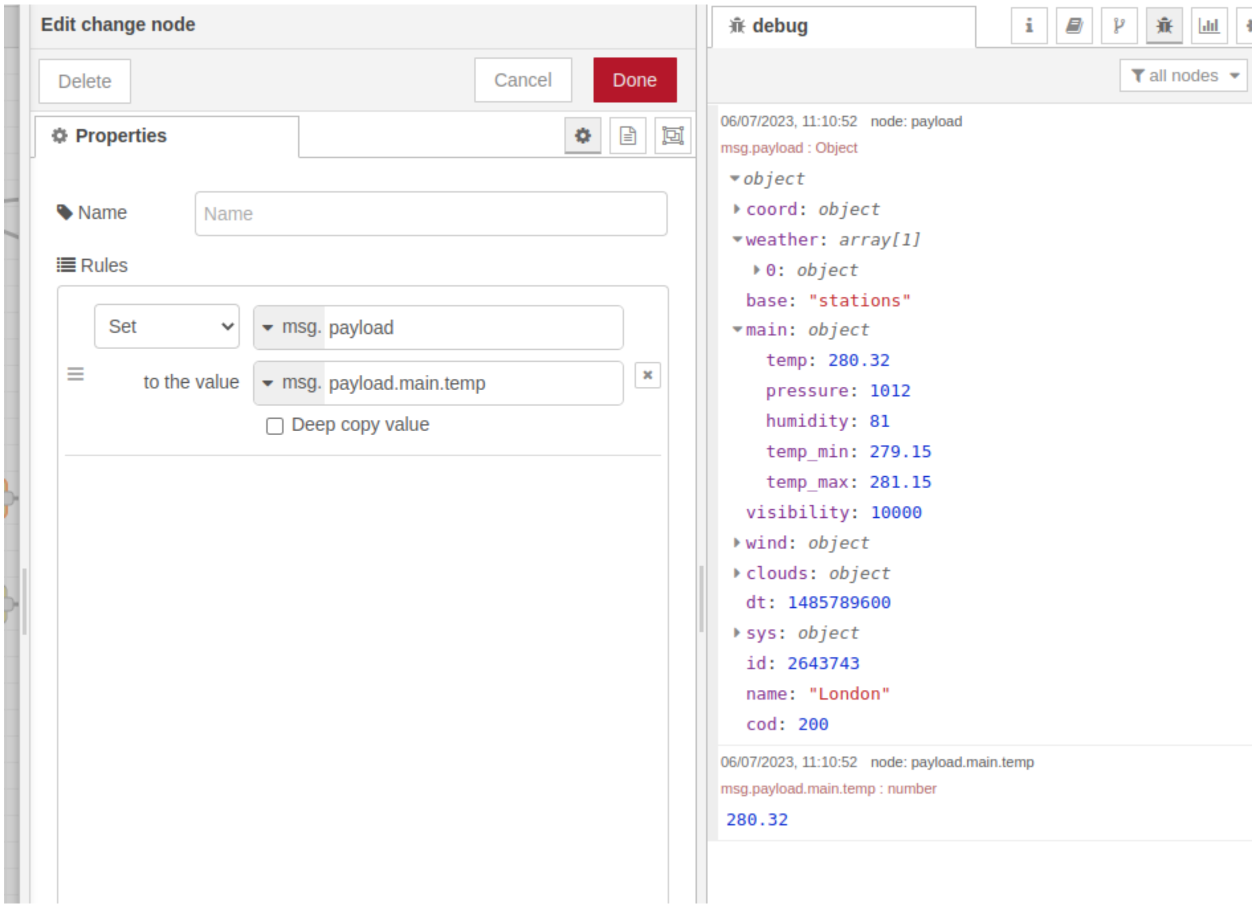

Edit debug node

Output: msg. payload.main.temp

To: [✓ debug window] system console / node status (32 characters)

Name: payload.main.temp

```
// Create a new Date object
const currentDate = new Date();

// Get the current timestamp
const timestamp = currentDate.getTime();

// Set the timestamp as a query parameter in the msg.url
msg.url += `&timestamp=${timestamp}`;

// Return the modified message
return msg;
```



Name: Name

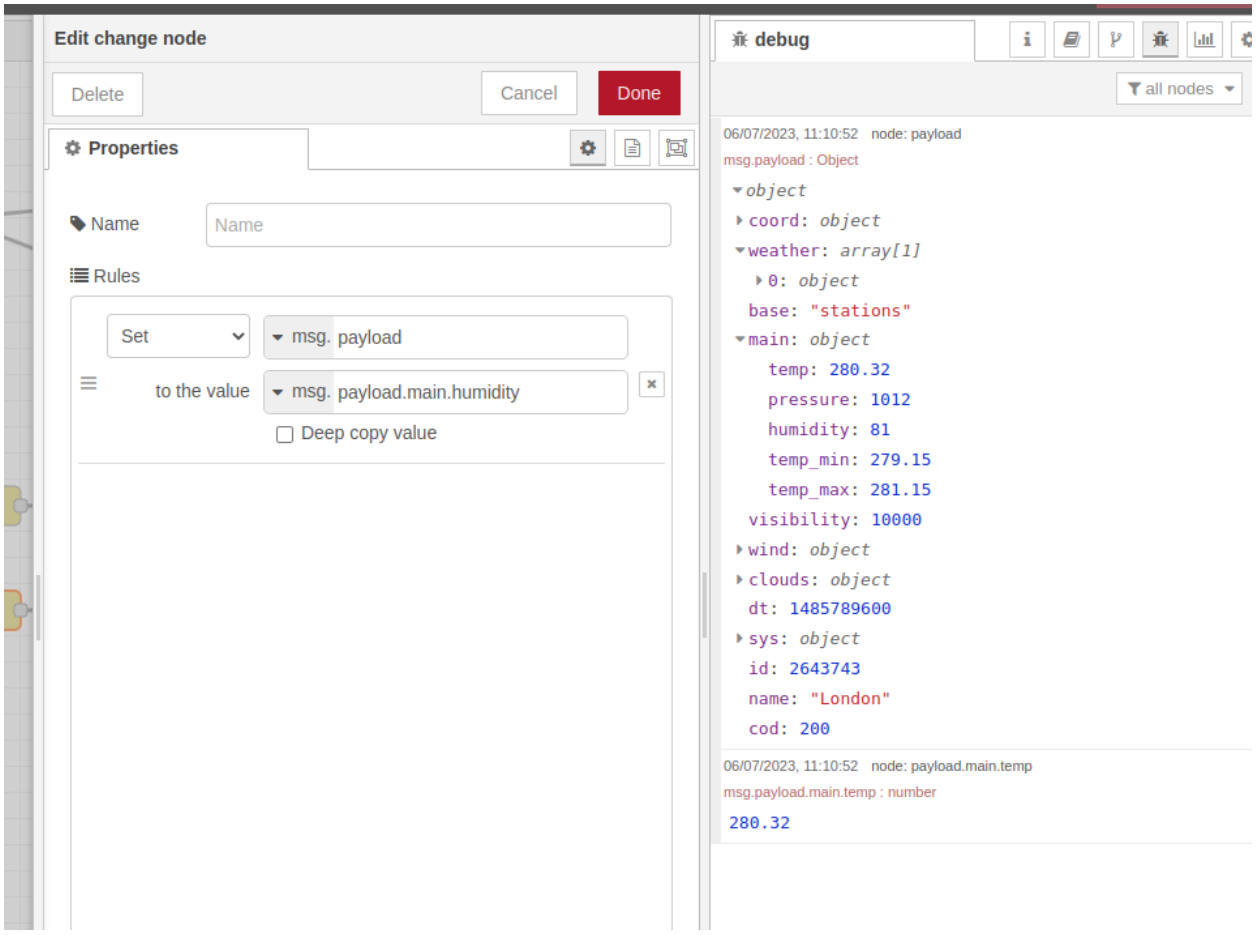
Rules

Set msg. payload

to the value msg. payload.main.temp

Deep copy value

```
06/07/2023, 11:10:52   node: payload
msg.payload : Object
▾object
 ▸coord: object
 ▾weather: array[1]
   ▸0: object
  base: "stations"
 ▾main: object
    temp: 280.32
    pressure: 1012
    humidity: 81
    temp_min: 279.15
    temp_max: 281.15
  visibility: 10000
 ▸wind: object
 ▸clouds: object
  dt: 1485789600
 ▸sys: object
  id: 2643743
  name: "London"
  cod: 200
06/07/2023, 11:10:52   node: payload.main.temp
msg.payload.main.temp : number
280.32
```



Name: Name

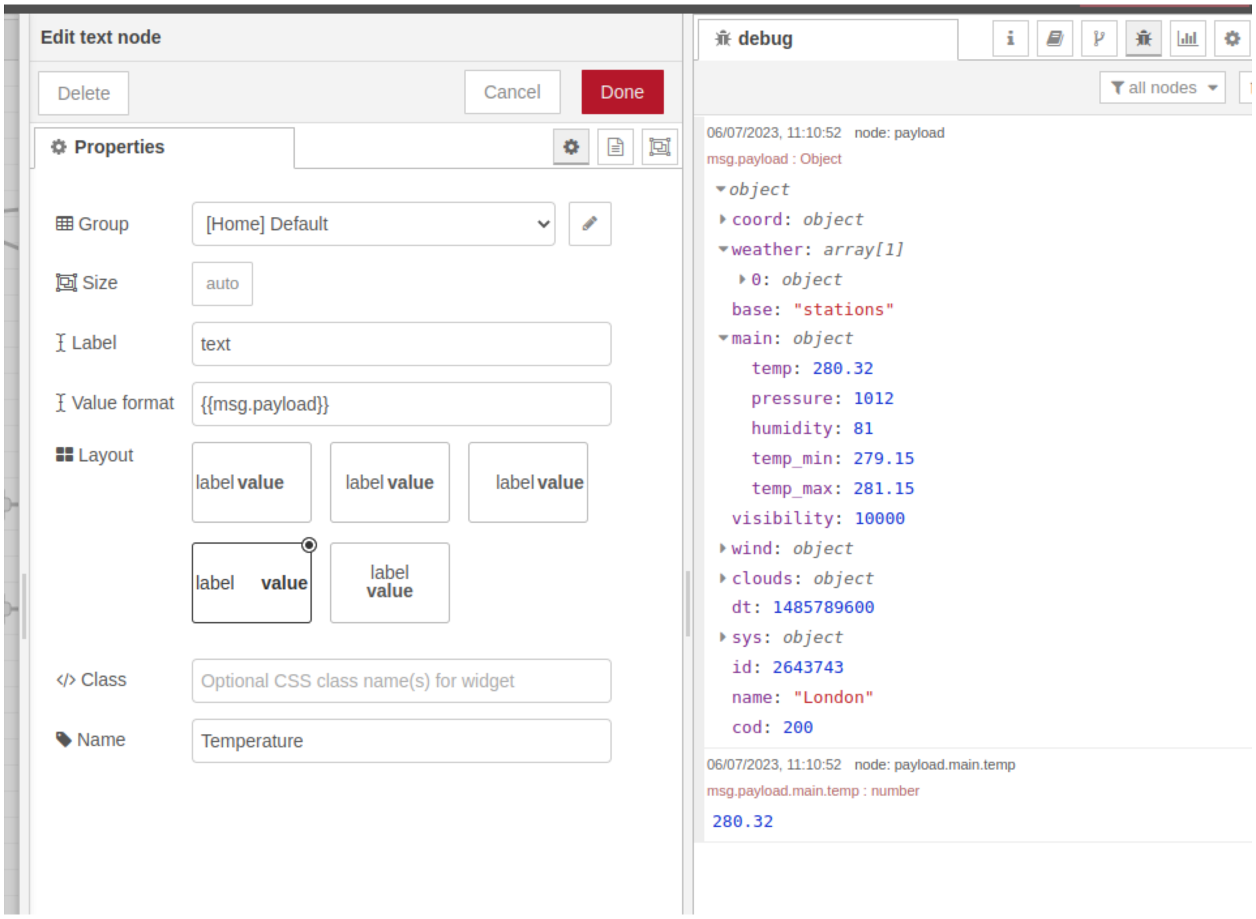
Rules

Set msg. payload

to the value msg. payload.main.humidity

Deep copy value

```
06/07/2023, 11:10:52   node: payload
msg.payload : Object
▾object
  ▸coord: object
  ▾weather: array[1]
    ▸0: object
  base: "stations"
  ▾main: object
    temp: 280.32
    pressure: 1012
    humidity: 81
    temp_min: 279.15
    temp_max: 281.15
  visibility: 10000
  ▸wind: object
  ▸clouds: object
  dt: 1485789600
  ▸sys: object
  id: 2643743
  name: "London"
  cod: 200

06/07/2023, 11:10:52   node: payload.main.temp
msg.payload.main.temp : number
280.32
```



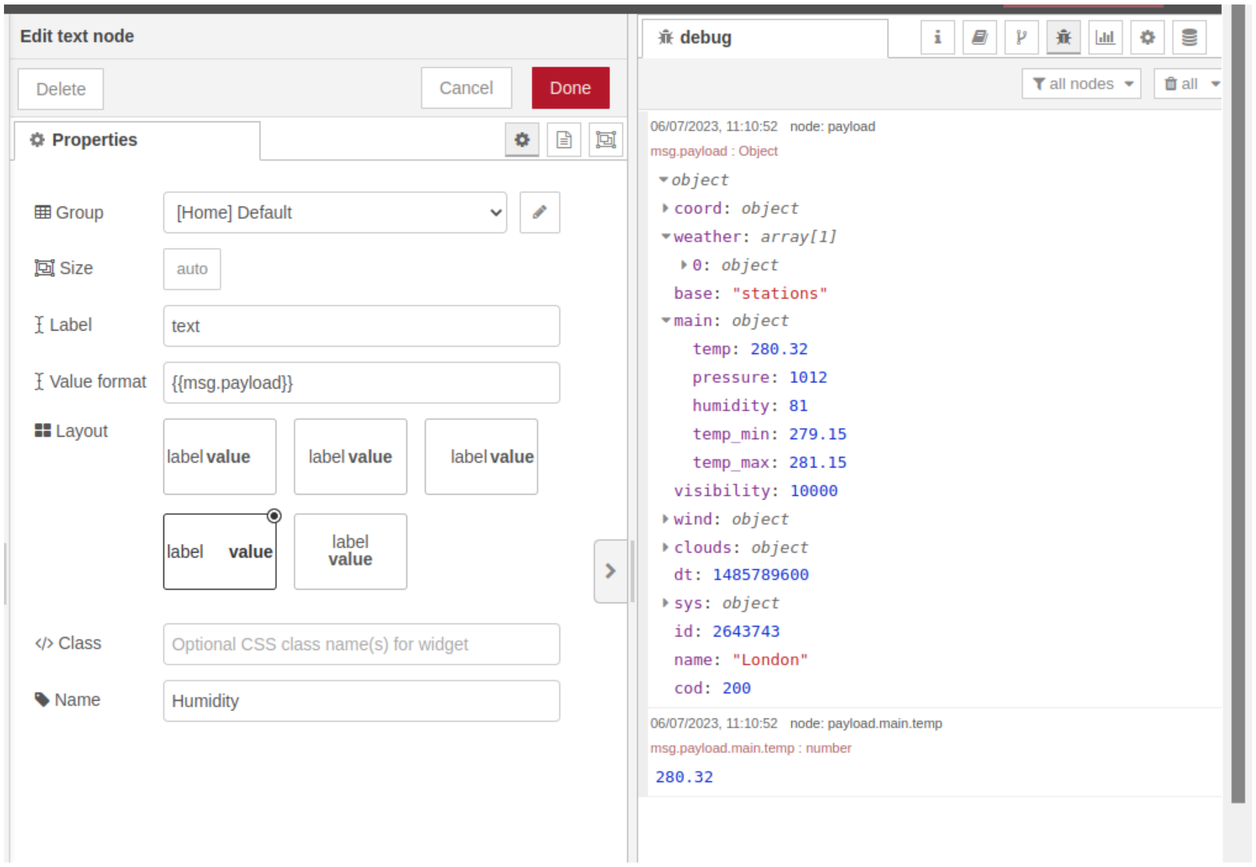
Edit text node

Delete | Cancel | Done

Properties

Group	[Home] Default
Size	auto
Label	text
Value format	{{msg.payload}}
Layout	label value
Class	Optional CSS class name(s) for widget
Name	Temperature

```
06/07/2023, 11:10:52   node: payload
msg.payload : Object
▾object
 ▸coord: object
 ▾weather: array[1]
   ▸0: object
  base: "stations"
 ▾main: object
    temp: 280.32
    pressure: 1012
    humidity: 81
    temp_min: 279.15
    temp_max: 281.15
  visibility: 10000
 ▸wind: object
 ▸clouds: object
  dt: 1485789600
 ▸sys: object
  id: 2643743
  name: "London"
  cod: 200
06/07/2023, 11:10:52   node: payload.main.temp
msg.payload.main.temp : number
280.32
```


Edit text node

Delete | Cancel | Done

Properties

Group: [Home] Default

Size: auto

Label: text

Value format: {{msg.payload}}

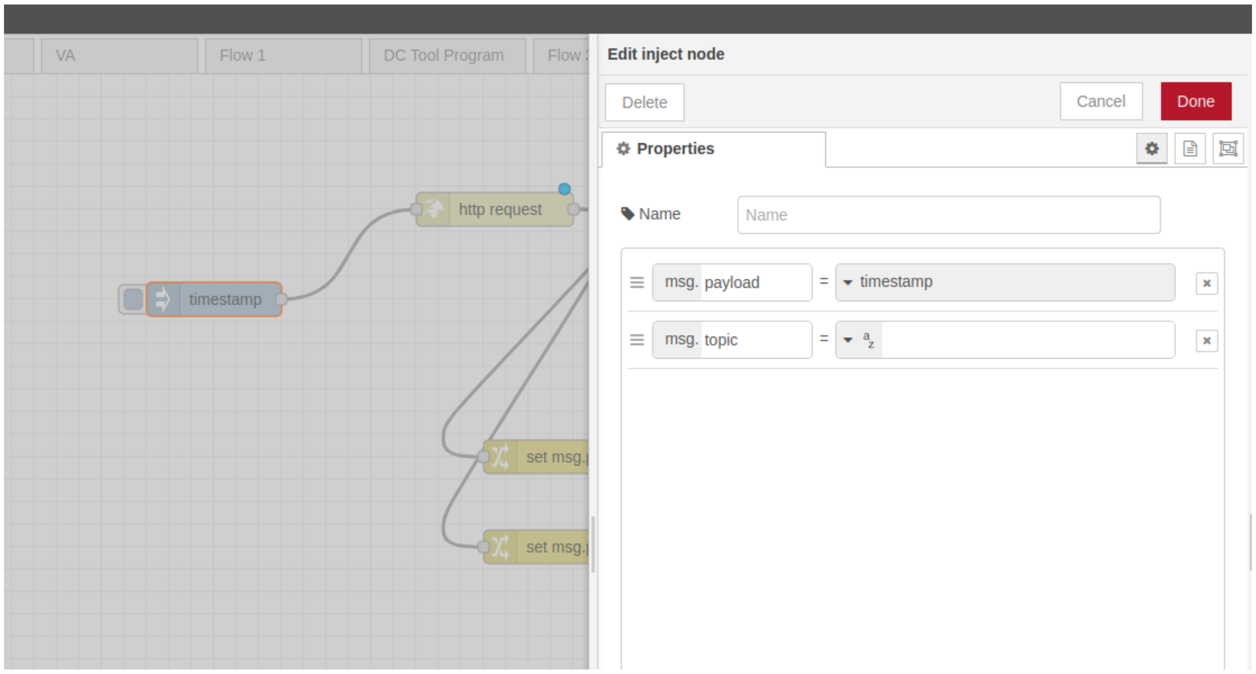
Layout: label value | label value | label value | [✓ label value] | label value

Class: Optional CSS class name(s) for widget

Name: Humidity

debug — all nodes — all

```
06/07/2023, 11:10:52   node: payload
msg.payload : Object
▾object
 ▸coord: object
 ▾weather: array[1]
   ▸0: object
  base: "stations"
 ▾main: object
   temp: 280.32
   pressure: 1012
   humidity: 81
   temp_min: 279.15
   temp_max: 281.15
  visibility: 10000
 ▸wind: object
 ▸clouds: object
  dt: 1485789600
 ▸sys: object
  id: 2643743
  name: "London"
  cod: 200
06/07/2023, 11:10:52   node: payload.main.temp
msg.payload.main.temp : number
280.32
```



Edit inject node

Delete | Cancel | Done

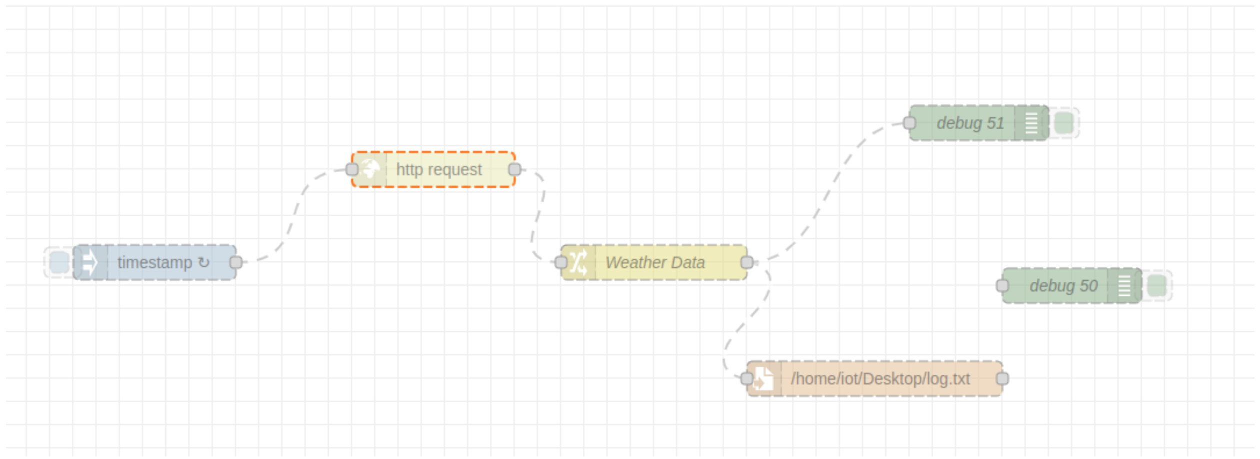
Properties

Name: Name

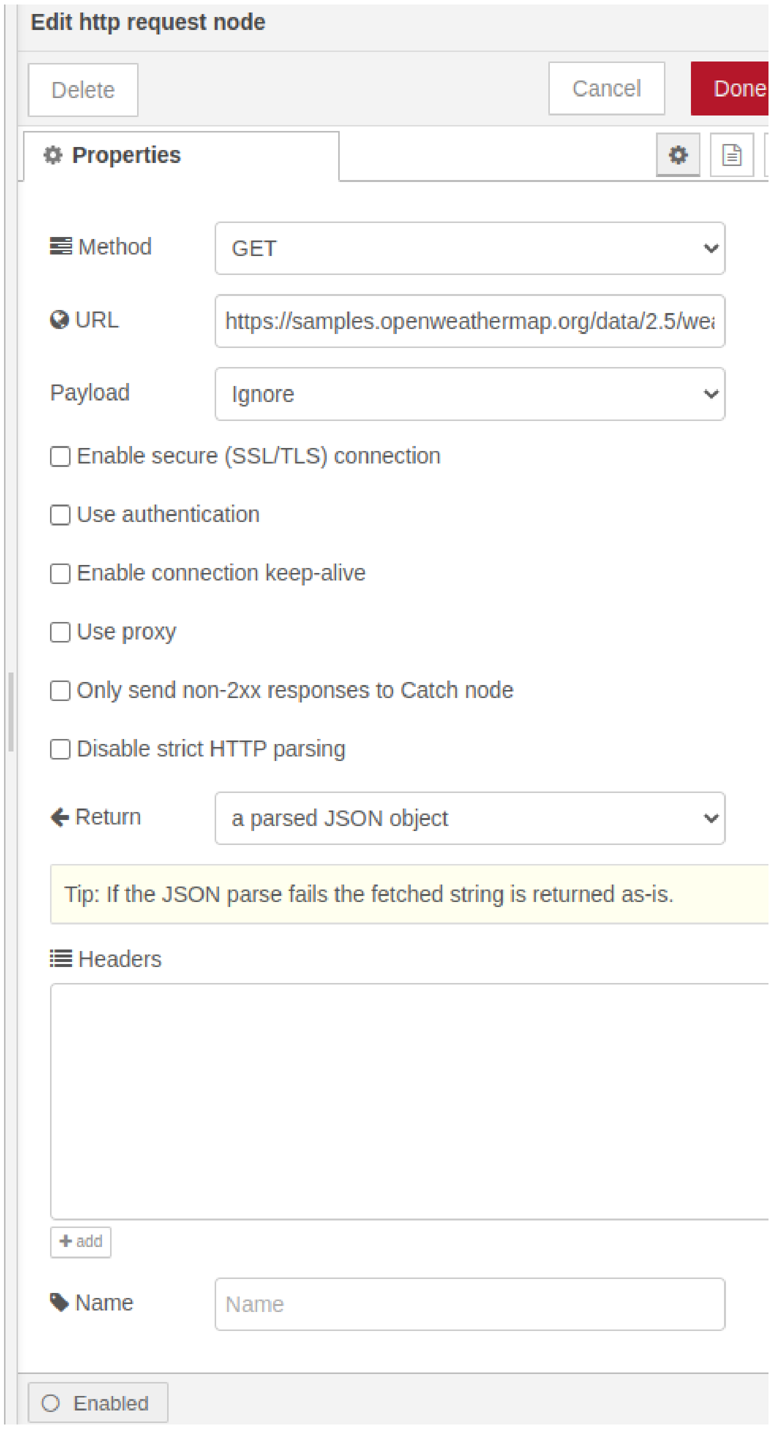
msg. payload = timestamp

msg. topic =

```
// Create a new Date object
const currentDate = new Date();

// Get the current timestamp
const timestamp = currentDate.getTime();

// Set the timestamp as a query parameter in the msg.url
msg.url += `&timestamp=${timestamp}`;

// Return the modified message
return msg;
```



Http request node:


Edit http request node

Delete | Cancel | Done

Properties

Method: GET

URL: https://samples.openweathermap.org/data/2.5/we

Payload: Ignore

☐ Enable secure (SSL/TLS) connection

☐ Use authentication

☐ Enable connection keep-alive

☐ Use proxy

☐ Only send non-2xx responses to Catch node

☐ Disable strict HTTP parsing

Return: a parsed JSON object

Tip: If the JSON parse fails the fetched string is returned as-is.

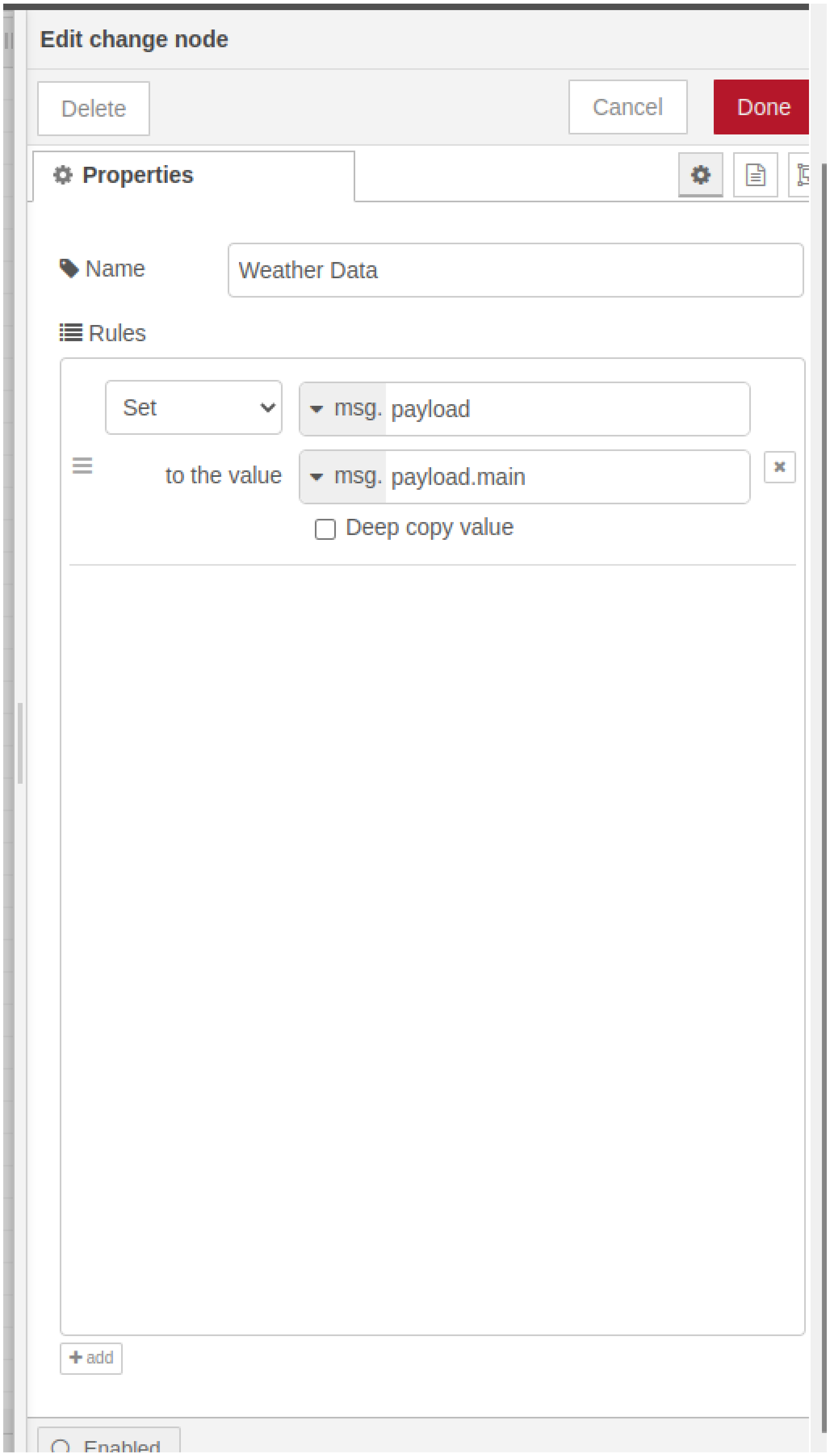
Headers

+ add

Name: Name

Enabled

Change node:weathr data ……. To just get the main parameters(tempereature,pressure,humidity,.)


Edit change node

Name: Weather Data

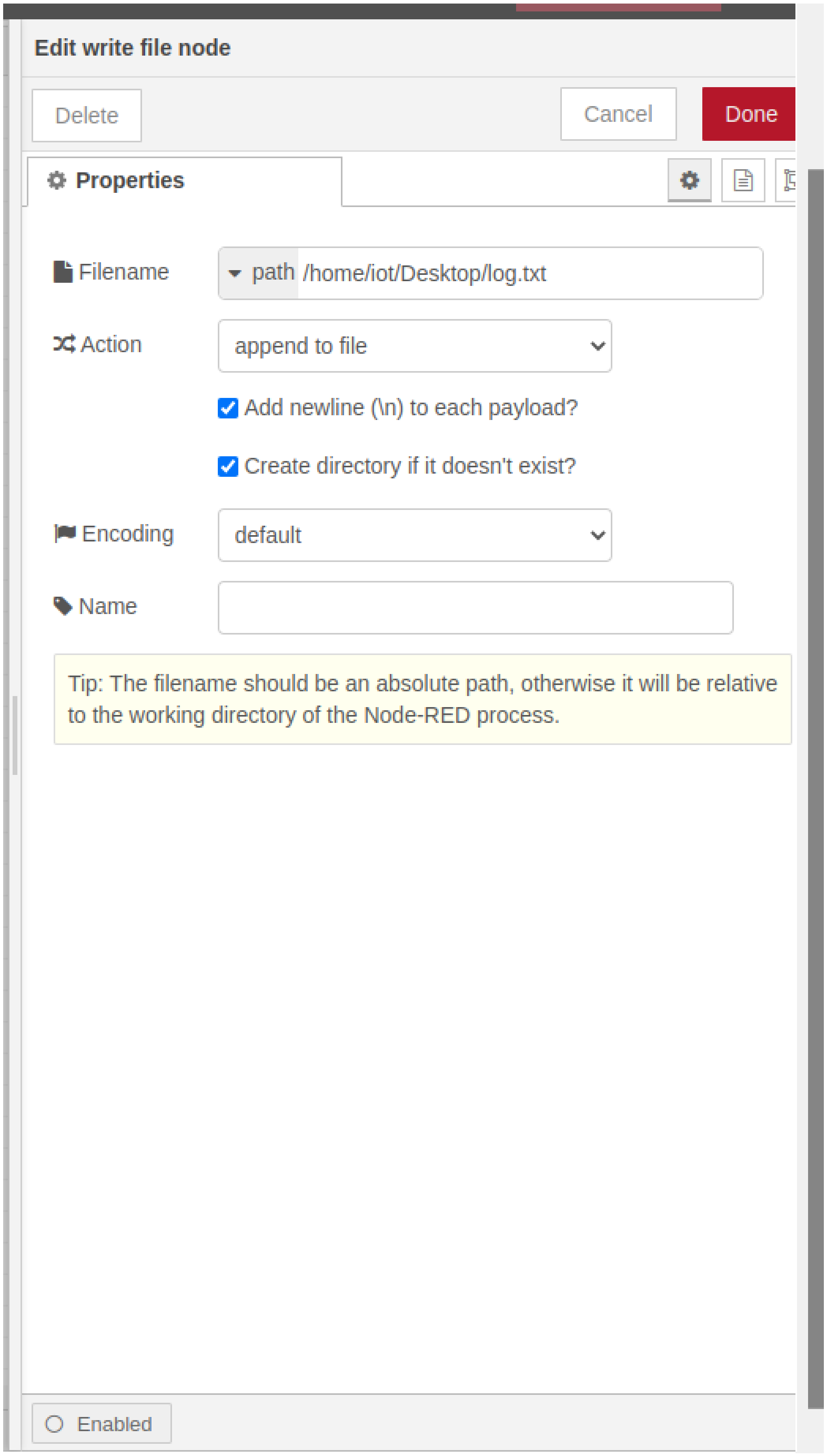
Rules

Set msg. payload

to the value msg. payload.main

Deep copy value

Write file node:
 To append the log which comes from the api key of open weather into a separate file created named log.txt



Filename: path /home/iot/Desktop/log.txt

Action: append to file

☑ Add newline (\n) to each payload?

☑ Create directory if it doesn't exist?

Encoding: default

Name:

Tip: The filename should be an absolute path, otherwise it will be relative to the working directory of the Node-RED process.

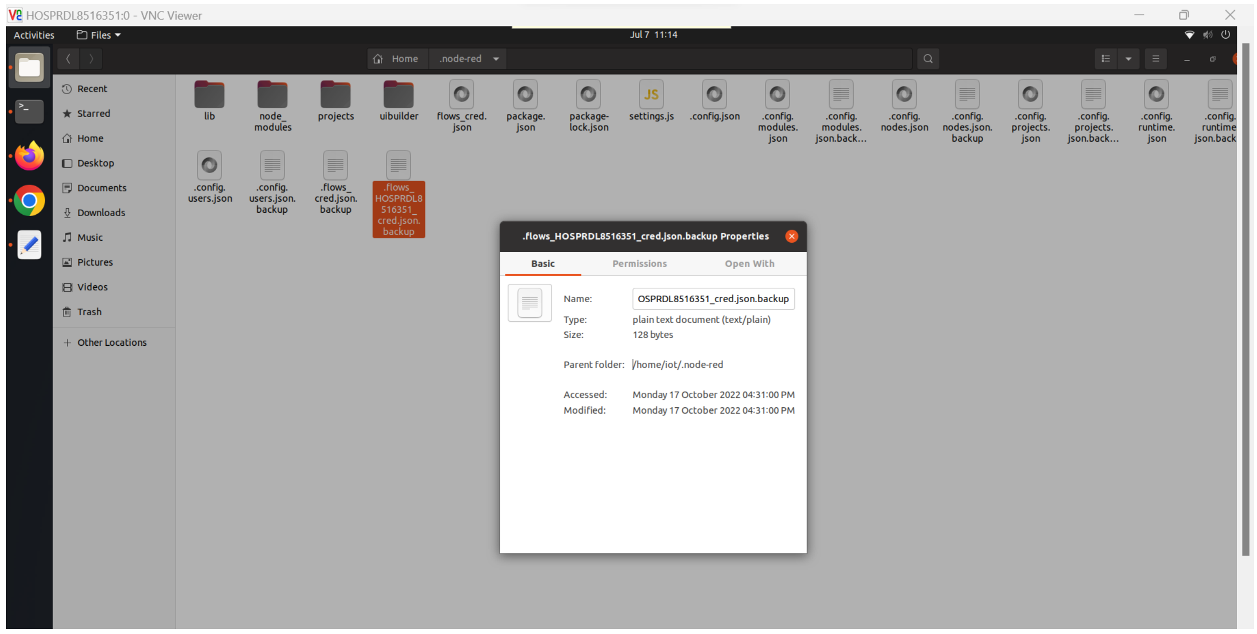
https://github.com/SuryaMahesh789/Data-Ingestion-with-Node.js-on-Elasticsearch-Service.git

Yes, it is possible to use your existing Node.js code to ingest data into Elasticsearch from Node-RED. Node-RED allows you to execute custom Node.js code within its flows using the "function" node. Here's how you can achieve it:

Import the required dependencies: If your existing Node.js code uses any external dependencies, make sure to import them in your Node-RED flow. This can be done by opening the Node-RED editor, clicking on the menu icon, selecting "Manage palette," and installing the necessary packages.
Create a function node: Drag and drop a "function" node from the Node-RED palette onto your flow canvas.
Configure the function node: Double-click on the function node to open its configuration dialog. Here, you can write your existing Node.js code inside the function's code editor. Ensure that your code includes the necessary Elasticsearch client libraries and functions for data ingestion.
Modify the code if needed: Depending on your existing code and how it interacts with Elasticsearch, you may need to make adjustments to align it with the Node-RED environment. This could involve modifying input/output variables, adapting data formatting, or incorporating Node-RED context or message objects.
Connect the function node: Connect the input and output nodes around the function node to define the flow of data. The preceding nodes should provide the necessary input data, while the subsequent nodes can handle the result or perform further operations on the data.
Deploy and test: After making the necessary configurations, click on the "Deploy" button to deploy your Node-RED flow. Test the flow by injecting data into the

function node and verifying that it successfully ingests the data into Elasticsearch.

By utilizing the function node in Node-RED, you can seamlessly integrate your existing Node.js code for data ingestion into Elasticsearch within the visual flow-based environment of Node-RED.



.flows_HOSPRDL8516351_cred.json.backup Properties

Name: OSPRDL8516351_cred.json.backup
Type: plain text document (text/plain)
Size: 128 bytes
Parent folder: /home/iot/.node-red
Accessed: Monday 17 October 2022 04:31:00 PM
Modified: Monday 17 October 2022 04:31:00 PM

```
{
    "elastic": {
      "cloudID":
"b72908efb0dd4f3c8e60bb0941c0bb57:dXMtY2VudHJhbDEuZ2NwLmNsb3VkLmVzLmlvJGZk
MDBjNjM3MGM2OTQxNzdhY2ZjZjNmOGQ2NjNmYjdkJGIxOGFhYjgwOTJkYjQyMzhhYmUyYTAzYm
ZjMWYzNWM5",
      "username": "elastic",
      "password": "xhLARutkh41NExTC3iLODkuT"
    }
}
const { Client } = require('@elastic/elasticsearch');
const client = new Client({
  node:
'https://fd00c6370c694177acfcf3f8d663fb7d.us-central1.gcp.cloud.es.io',
  auth: {
    username: 'elastic',
    password: 'xhLARutkh41NExTC3iLODkuT'
  },
  ssl: {
    rejectUnauthorized: false
  }
});

client.info()
  .then(response => console.log(response))
  .catch(error => console.error(error))



async function run() {
  await client.index({
    index: 'game-of-thrones',
    body: {
      character: 'Ned Stark',
    quote: 'Winter is coming.'
    }
  })

  await client.index({
    index: 'game-of-thrones',
    body: {
      character: 'Daenerys Targaryen',
    quote: 'I am the blood of the dragon.'
    }
  })

  await client.index({
    index: 'game-of-thrones',
    body: {
      character: 'Tyrion Lannister',
    quote: 'A mind needs books like a sword needs whetstone.'
    }
  })

  await client.indices.refresh({index: 'game-of-thrones'})
```

```
}

run().catch(console.log)
```

```
const { Client } = require('@elastic/elasticsearch');
const client = new Client({
  node:
'https://fd00c6370c694177acfcf3f8d663fb7d.us-central1.gcp.clo
ud.es.io',
  auth: {




    username: 'elastic',
    password: 'xhLARutkh41NExTC3iLODkuT'
  },
  ssl: {
    rejectUnauthorized: false
  }
});


client.info()
  .then(response => console.log(response))
  .catch(error => console.error(error))
```

```
async function run() {
  await client.index({
    index: 'game-of-thrones',
    body: {
      character: 'Ned Stark',
    quote: 'Winter is coming.'
    }
  })


  await client.index({
    index: 'game-of-thrones',
    body: {
      character: 'Daenerys Targaryen',
    quote: 'I am the blood of the dragon.'
    }
  })


  await client.index({
    index: 'game-of-thrones',
    body: {
      character: 'Tyrion Lannister',
    quote: 'A mind needs books like a sword needs whetstone.'
    }
  })


  await client.indices.refresh({index: 'game-of-thrones'})
}


run().catch(console.log)
```

```
[
    {
        "id": "fd9d58d8d8b2ec7f",
        "type": "tab",
        "label": "logging data to elastic search",
        "disabled": false,
        "info": "",
        "env": []
    },
    {
        "id": "df37ac34e1dd5840",
        "type": "inject",
        "z": "fd9d58d8d8b2ec7f",
        "name": "",
        "props": [
            {
                "p": "payload"
            },
        {
            "p": "topic",
            "vt": "str"
        }
    ],
    "repeat": "4",
    "crontab": "",
    "once": false,
    "onceDelay": 0.1,
    "topic": "",
    "payload": "",
    "payloadType": "date",
    "x": 390,
    "y": 320,
    "wires": [
        [
            "367e6eb4263ddd4c"
        ]
    ]
},
{
    "id": "367e6eb4263ddd4c",
    "type": "http request",
    "z": "fd9d58d8d8b2ec7f",
    "name": "",
    "method": "GET",
    "ret": "obj",
    "paytoqs": "ignore",
        "url":
"https://samples.openweathermap.org/data/2.5/weather?q=Lond
on,uk&appid=b6907d289e10d714a6e88b30761fae22",
        "tls": "",
        "persist": false,
        "proxy": "",
        "insecureHTTPParser": false,
        "authType": "",
        "senderr": false,
        "headers": [],
        "x": 610,
        "y": 320,
        "wires": [
          [
            "a4dec287db646f06",
            "2ee191ab6da45458",
            "945a15118bbfeaa6",
            "b095e5b5639baabd",
            "cb3b27d5e7468648",
            "25ad8c1446391bd0"
          ]
        ]
    },
    {
        "id": "a4dec287db646f06",
        "type": "change",
        "z": "fd9d58d8d8b2ec7f",
        "name": "Weather Data",
        "rules": [
        {
            "t": "set",
            "p": "payload",
            "pt": "msg",
            "to": "payload.main",
            "tot": "msg"
        }
    ],
    "action": "",
    "property": "",
    "from": "",
    "to": "",
    "reg": false,
    "x": 800,
    "y": 400,
    "wires": [
        [
            "d301ab0242ac1254"
        ]
    ]
},
{
    "id": "ce8cc3ec292e872c",
    "type": "file",
    "z": "fd9d58d8d8b2ec7f",
    "name": "",
    "filename": "/home/iot/Desktop/weather_data.txt",
    "filenameType": "str",
    "appendNewline": true,
        "createDir": true,
        "overwriteFile": "false",
        "encoding": "none",
        "x": 1060,
        "y": 460,
        "wires": [
            []
        ]
    },
    {
        "id": "5984ada26391d2ad",
        "type": "debug",
        "z": "fd9d58d8d8b2ec7f",
        "name": "output",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "payload",
        "targetType": "msg",
        "statusVal": "",
        "statusType": "auto",
        "x": 910,
        "y": 700,
        "wires": []
    },
    {
        "id": "d301ab0242ac1254",
        "type": "function",
        "z": "fd9d58d8d8b2ec7f",
        "name": "current weather conditions",
        "func": "//Current Date\nvar timestamp = new Date();\n\n//
Format the time using desired options\nvar formattedTime =
timestamp.toLocaleTimeString('en-US', { hour12: true, hour:
'numeric', minute: 'numeric', second: 'numeric'
})\n\n\n//msg.payload IS THE MAIN PARAMETERS FROM THE
WEATHER DATA Collected\n//Split the parameters
accordingly\n\n\nvar temp = msg.payload.temp;\nvar pressure =
msg.payload.pressure;\nvar humidity =
msg.payload.humidity;\nvar temp_max =
msg.payload.temp_max;\nvar temp_min =
msg.payload.temp_min;\n\ntemp = Math.random() * 123 + temp *
Math.random() -
Math.random()*20;\ntemp=temp.toFixed(2);\ntemp =
parseInt(temp);\n\npressure = Math.random() * 678 + pressure *
Math.random() - Math.random() * 33;\npressure =
Math.round(pressure);\n\nhumidity = Math.random() * 45 +
humidity * Math.random() - Math.random() * 22;\nhumidity =
Math.round(humidity);\n\ntemp_max = Math.random() * 131 +
temp_max * Math.random() - Math.random() * 55;\ntemp_max =
temp_max.toFixed(2);\ntemp_max =
parseInt(temp_max);\n\n\ntemp_min = Math.random() * 34 +
temp_min * Math.random() - Math.random() * 20;\ntemp_min =
temp_min.toFixed(2);\ntemp_min =
parseInt(temp_min);\n\n\nmsg.payload = { temp: temp, pressure:
pressure, humidity: humidity, temp_max: temp_max, temp_min:
temp_min, time: formattedTime}\nreturn msg;\n\n\n\n",
        "outputs": 1,
    "noerr": 0,
    "initialize": "",
    "finalize": "",
    "libs": [],
    "x": 500,
    "y": 460,
    "wires": [
      [
        "ce8cc3ec292e872c",
        "68bba3be84906ca6"
      ]
    ]
  },
  {
    "id": "244cf30fe0e45b60",
    "type": "ui_text",
    "z": "fd9d58d8d8b2ec7f",
    "group": "0623e9795635cab0",
    "order": 5,
    "width": 0,
    "height": 0,
    "name": "temperature",
    "label": "temperature",
    "format": "{{msg.payload}}",
    "layout": "row-spread",
    "className": "",
    "x": 1350,
    "y": 120,
    "wires": []
```

```
},
{
    "id": "2ee191ab6da45458",
    "type": "change",
    "z": "fd9d58d8d8b2ec7f",
    "name": "",
    "rules": [
        {
            "t": "set",
            "p": "payload",
            "pt": "msg",
            "to": "payload.main.temp",
            "tot": "msg"
        }
    ],
    "action": "",
    "property": "",
    "from": "",
    "to": "",
    "reg": false,
    "x": 1120,
    "y": 120,
    "wires": [
        [
            "244cf30fe0e45b60"
        ]
    ]
},
{
    "id": "945a15118bbfeaa6",
    "type": "change",
    "z": "fd9d58d8d8b2ec7f",
    "name": "",
    "rules": [
        {
            "t": "set",
            "p": "payload",
            "pt": "msg",
            "to": "payload.main.humidity",
            "tot": "msg"
        }
    ],
    "action": "",
    "property": "",
    "from": "",
    "to": "",
    "reg": false,
    "x": 1120,
    "y": 200,
    "wires": [
        [
            "48528a761fb14a6a"
        ]
    ]
},
{
    "id": "48528a761fb14a6a",
    "type": "ui_text",
        "z": "fd9d58d8d8b2ec7f",
        "group": "0623e9795635cab0",
        "order": 4,
        "width": 0,
        "height": 0,
        "name": "humidity",
        "label": "humidity",
        "format": "{{msg.payload}}",
        "layout": "row-spread",
        "className": "",
        "x": 1340,
        "y": 200,
        "wires": []
    },
    {
        "id": "f91bae0c8200f09c",
        "type": "ui_text",
        "z": "fd9d58d8d8b2ec7f",
        "group": "0623e9795635cab0",
        "order": 2,
        "width": 0,
        "height": 0,
        "name": "pressure",
        "label": "pressure",
        "format": "{{msg.payload}}",
        "layout": "row-spread",
        "className": "",
        "x": 1340,
        "y": 260,
    "wires": []
},
{
    "id": "2c475fcb66bb519a",
    "type": "ui_text",
    "z": "fd9d58d8d8b2ec7f",
    "group": "0623e9795635cab0",
    "order": 3,
    "width": 0,
    "height": 0,
    "name": "temp_max",
    "label": "temp_max",
    "format": "{{msg.payload}}",
    "layout": "row-spread",
    "className": "",
    "x": 1350,
    "y": 320,
    "wires": []
},
{
    "id": "b095e5b5639baabd",
    "type": "change",
    "z": "fd9d58d8d8b2ec7f",
    "name": "",
    "rules": [
        {
            "t": "set",
            "p": "payload",
            "pt": "msg",
            "to": "payload.main.pressure",
            "tot": "msg"
        }
    ],
    "action": "",
    "property": "",
    "from": "",
    "to": "",
    "reg": false,
    "x": 1120,
    "y": 260,
    "wires": [
        [
            "f91bae0c8200f09c"
        ]
    ]
},
{
    "id": "cb3b27d5e7468648",
    "type": "change",
    "z": "fd9d58d8d8b2ec7f",
    "name": "",
    "rules": [
        {
            "t": "set",
            "p": "payload",
            "pt": "msg",
            "to": "payload.main.temp_max",
            "tot": "msg"
        }
    ],
    "action": "",
    "property": "",
    "from": "",
    "to": "",
    "reg": false,
    "x": 1120,
    "y": 320,
    "wires": [
        [
            "2c475fcb66bb519a"
        ]
    ]
},
{
    "id": "25ad8c1446391bd0",
    "type": "change",
    "z": "fd9d58d8d8b2ec7f",
    "name": "",
    "rules": [
        {
            "t": "set",
            "p": "payload",
            "pt": "msg",
            "to": "payload.main.temp_min",
            "tot": "msg"
        }
    ],
    "action": "",
    "property": "",
    "from": "",
    "to": "",
    "reg": false,
    "x": 1120,
    "y": 380,
    "wires": [
        [
            "6e7daf87110da508"
        ]
    ]
},
{
    "id": "6e7daf87110da508",
    "type": "ui_text",
    "z": "fd9d58d8d8b2ec7f",
    "group": "0623e9795635cab0",
    "order": 6,
    "width": 0,
    "height": 0,
    "name": "temp_min",
    "label": "temp_min",
    "format": "{{msg.payload}}",
    "layout": "row-spread",
    "className": "",
    "x": 1340,
    "y": 380,
    "wires": []
    },
    {
        "id": "68bba3be84906ca6",
        "type": "function",
        "z": "fd9d58d8d8b2ec7f",
        "name": "elastic search",
        "func": "const payload = msg.payload;\nconst elasticsearch =
global.get('elasticsearch');\nconst client=new
elasticsearch.Client({\n    node:
'https://fd00c6370c694177acfcf3f8d663fb7d.us-central1.gcp.clo
ud.es.io',\n    auth:{\n        username:'elastic',\n
password:'xhLARutkh41NExTC3iLODkuT'\n    },\n    ssl:{\n
rejectUnauthorized:false\n    }\n})\n\nclient.index({\n
index:'weather-information',\n    body:payload\n})\n\n
.then(response => {\n        // Handle the response as needed\n
console.log('Document indexed:', response);\n        // ...\n        //
Send the output message or perform other actions\n
msg.payload = response;\n        return msg;\n    })\n    .catch(error
=> {\n        // Handle the error\n        console.error('Error indexing
document:', error);\n        // ...\n        // Send an error message or
perform other actions\n        msg.payload = { error: error.message
};\n        return msg;\n    });\n\n\n\n\n\nmsg.payload =
msg.payload;\n\nreturn msg;",
        "outputs": 1,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 660,
        "y": 580,
        "wires": [
            [
                "5984ada26391d2ad"
            ]
        ]
    },
    {
        "id": "0623e9795635cab0",
        "type": "ui_group",
        "name": "Default",
        "tab": "62d45b6a18a1b70d",
        "order": 1,
        "disp": true,
        "width": "6",
        "collapse": false,
        "className": ""
    },
    {
        "id": "62d45b6a18a1b70d",
        "type": "ui_tab",
        "name": "Home",
        "icon": "dashboard",
        "disabled": false,
        "hidden": false
    }
]
```

—---

Module 1: Introduction to Elasticsearch

This module is designed to provide an introduction to Elasticsearch, a powerful search and analytics engine widely used for data exploration, log analysis, and real-time search applications. In this module, you will learn the basics of Elasticsearch, including how to install and configure the software, index data, and create mappings.
You will also explore the different types of data that can be indexed in Elasticsearch, such as text, numerical, and geospatial data. Additionally, you will gain a basic understanding of Elasticsearch's architecture, including its distributed nature and its use of sharding and replication to ensure high availability and fault tolerance.

Module 2: Search Based Query Optimisation

This module is designed to teach you how to optimize search-based queries in Elasticsearch for improved performance and relevance. You will start by learning how to use Elasticsearch's Explain API to understand how queries are executed and identify potential bottlenecks.
You will then explore different techniques for improving query performance, such as using filters instead of queries, limiting the number of fields returned, and caching query results.

Module 3: Advanced Querying Mechanism in Elasticsearch

This module is designed to teach you advanced query mechanisms in Elasticsearch, enabling you to build complex queries that deliver highly relevant results. You will start by learning about the different types of query mechanisms available in Elasticsearch, such as Boolean queries, fuzzy queries, and proximity queries.
You will then explore advanced query mechanisms such as nested queries, script queries, and join queries. Additionally, you will learn how to use Elasticsearch's search templates to build and reuse complex queries.

how to execute full-text queries with Elasticsearch, a powerful search and analytics engine widely used for data exploration, log analysis, and real-time search applications. In this course, you will learn the basics of Elasticsearch, including how to install and configure the software, index data, and create mappings. You will then explore the various query types supported by Elasticsearch, such as term queries, match queries, and phrase queries.
You will also learn how to use Elasticsearch's aggregation framework to analyze and summarize your data. Additionally, you will gain hands-on experience using Elasticsearch's search API to execute queries and retrieve search results.

Data ingestion into elastic search using node.js

Some Introduction To ELK stack.

Node.js with ELK Stack
1. Ingest data with Node.js on Elasticsearch Service
2. This guide tells you how to get started with:
3. Securely connecting to Elasticsearch Service with Node.js
4. Ingesting data into your deployment from your application
5. Searching and modifying your data on Elasticsearch Service

If you are an Node.js application programmer who is new to the Elastic Stack, this content helps you get started more easily.

1. Get Elasticsearch Service
2. Get a free trial.
3. Log into Elastic Cloud.
4. Select Create deployment.

Give your deployment a name. You can leave all other settings at their default values.
Select Create deployment and save your Elastic deployment credentials. You need these credentials later on.

When the deployment is ready, click Continue and a page of Setup guides is displayed. To continue to the deployment homepage click I'd like to do something else.
Prefer not to subscribe to yet another service? You can also get Elasticsearch Service through AWS, Azure, and GCP marketplaces.

Set up your application
These steps are applicable to your existing application. If you don't have one, use the example included here to create one.

Create the npm package.json file

```
npm init
```

Get the elasticsearch and config packages

```
npm install @elastic/elasticsearch
npm install config
```

The config package is not required if you have your own method to keep your configuration details private.

Create a configuration file

```
mkdir config
vi config/default.json
```

The example here shows what the config package expects. You need to update config/default.json with your deployment details in the following sections:

```
{
  "elastic": {
    "cloudID": "DEPLOYMENT_NAME:CLOUD_ID_DETAILS",
    "username": "elastic",
    "password": "LONGPASSWORD"
  }
}
```

Find your Cloud ID by going to the Kibana main menu and selecting Management > Integrations, and then selecting View deployment details.

About connecting securely
When connecting to Elasticsearch Service use a Cloud ID to specify the connection details. You must pass the Cloud ID that is found in Kibana or the cloud console.

To connect to, stream data to, and issue queries with Elasticsearch Service, you need to think about authentication. Two authentication mechanisms are supported, API key and basic authentication. Here, to get you started quickly, we'll show you how to use basic authentication, but you can also generate API keys as shown later on. API keys are safer and preferred for production environments.

Basic authentication
For basic authentication, use the same deployment credentials (username and password parameters) and Cloud ID you copied down earlier when you created your deployment. (If you did not save the password, you can reset the password .)
Refernces:

https://www.elastic.co/guide/en/cloud/current/ec-getting-started
-node-js.html

b72908efb0dd4f3c8e60bb0941c0bb57:dXMtY2VudHJhbDEuZ2N
wLmNsb3VkLmVzLmlvJGZkMDBjNjM3MGM2OTQxNzdhY2ZjZjN
mOGQ2NjNmYjdkJGIxOGFhYjgwOTJkYjQyMzhhYmUyYTAzYmZj
MWYzNWM5

To reset the password and get the username and pasword from
elastic search.
Refernces:
https://www.elastic.co/guide/en/cloud/current/ec-password-rese
t.html

username
elastic
password
xhLARutkh41NExTC3iLODkuT

elastic search endpoint
https://fd00c6370c694177acfcf3f8d663fb7d.us-central1.gcp.clou
d.es.io

Create a sample application

The sample application connects to Elasticsearch, creates an index, inserts some records, performs a search, and updates a record.

Read the configuration created earlier, and connect to Elasticsearch:

```
const { Client } = require('@elastic/elasticsearch')
const config = require('config');
const elasticConfig = config.get('elastic');

const client = new Client({
  cloud: {
    id: elasticConfig.cloudID
  },
  auth: {
    username: elasticConfig.username,
    password: elasticConfig.password
  }
})
```

Now confirm that you are connected to the deployment by returning some information about the deployment:

```
client.info()
  .then(response => console.log(response))
  .catch(error => console.error(error))
```

app.js

```
const { Client } = require('@elastic/elasticsearch');
const client = new Client({
  node:
'https://fd00c6370c694177acfcf3f8d663fb7d.us-central1.gcp.clo
ud.es.io',
  auth: {
    username: 'elastic',
    password: 'xhLARutkh41NExTC3iLODkuT'
  },
  ssl: {
    rejectUnauthorized: false
  }
});

client.info()
  .then(response => console.log(response))
  .catch(error => console.error(error))


async function indexDocument(index, document) {
    const { body } = await client.index({
      index,
      body: document
    });

    console.log('Document indexed:', body);
  }
```

```
const index = 'my_index';
const document = { name: 'John Doe', age: 30 };

indexDocument(index, document);

const index2 = 'my_index2';
const document2 = { name: 'Surya Mahesh', age: 30 };

indexDocument(index2, document2);


const index3 = 'my_index3';
const document3 = { name: 'Pemma Koushik', age: 30 };

indexDocument(index3, document3);

const index4 = 'my_index4';
const document4 = { name: 'John Doe', age: 30 };

indexDocument(index4, document4);

const index5 = 'my_index5';
const document5 = { name: 'Surya Mahesh', age: 30 };

indexDocument(index5, document5);


const index6 = 'my_index6';
const document6 = { name: 'Pemma Koushik', age: 30 };
```

```
  indexDocument(index6, document6);
```

deafult.json

```
{
   "elastic": {
     "cloudID":
"b72908efb0dd4f3c8e60bb0941c0bb57:dXMtY2VudHJhbDEuZ2
NwLmNsb3VkLmVzLmlvJGZkMDBjNjM3MGM2OTQxNzdhY2ZjZj
NmOGQ2NjNmYjdkJGIxOGFhYjgwOTJkYjQyMzhhYmUyYTAzYmZj
MWYzNWM5",
     "username": "elastic",
     "password": "xhLARutkh41NExTC3iLODkuT"
   }
 }
```

https://www.elastic.co/guide/en/cloud/current/ec-getting-started-node-js.html#ec_search_and_modify_data

connection the Node.js with elastic search platform and perform operations like creating index kinda things...

1) Indexin Documents

```
const { Client } = require('@elastic/elasticsearch');
const client = new Client({
  node:
'https://fd00c6370c694177acfcf3f8d663fb7d.us-central1.gcp.clo
ud.es.io',
  auth: {
    username: 'elastic',
    password: 'xhLARutkh41NExTC3iLODkuT'
  }
});

client.info()
  .then(response => console.log(response))
  .catch(error => console.error(error))


async function indexDocument(index, document) {
    const { body } = await client.index({
      index,
      body: document
    });

    console.log('Document indexed:', body);
  }

  const index = 'my_index';
  const document = { name: 'John Doe', age: 30 };

  indexDocument(index, document);
```

explaination:

By running the code to index a document in Elasticsearch using your Node.js application, you have achieved the following:

1. Index Creation: If the specified index (`my_index` in the example) did not exist before running the code, Elasticsearch will create the index for you. The index is a logical container that holds your documents.

2. Document Indexing: The provided document (in the example, `{ name: 'John Doe', age: 30 }`) is stored in Elasticsearch. The document is added to the specified index, making it searchable and retrievable.

3. Successful Connection: Since you mentioned that the connection was successful, it means your Node.js application was able to communicate with Elasticsearch and perform the indexing operation without encountering any errors.

By indexing the document, you have effectively stored data in Elasticsearch, which allows you to perform various operations like searching, aggregating, and retrieving the document based on different criteria.

To verify that the document has been indexed, you can use Elasticsearch APIs or tools like Kibana to explore the data in your Elasticsearch cluster. You can execute search queries against the index to retrieve the indexed document and validate its presence.

Congratulations on setting up the connection and indexing your first document! If you have any further questions or need assistance with other Elasticsearch operations, feel free to ask.

2) Searching & Retrieving Data

syntax

```
async function searchDocuments(index, query) {
  const { body } = await client.search({
    index,
    body: {
      query: {
        match: {
          '<field_name>': query
        }
      }
    }
  });
  console.log('Search results:', body.hits.hits);
}
```

example

```
async function searchDocuments(index, query) {
  const { body } = await client.search({
    index,
    body: {
      query: {
        match: {
          'title': query
        }
      }
    }
  });
  console.log('Search results:', body.hits.hits);
}
```

search_query.js

search by checking name match

```
const { Client } = require('@elastic/elasticsearch');
// node is elastic search endpoint
const client = new Client(
    {
        node:
'https://fd00c6370c694177acfcf3f8d663fb7d.us-central1.gcp.clo
ud.es.io',
        auth: {
                username: 'elastic',
                password: 'xhLARutkh41NExTC3iLODkuT'
              },
        ssl: {
            rejectUnauthorized: false
            }

    }
    );




//   searching and retrieving data
```

```
  async function searchDocuments(index, query) {
    const { body } = await client.search({
      index,
      body: {
        query: {
          match: {
            'name': query
          }
        }
      }
    });
    console.log('Search results:', body.hits.hits);
  }

const index = 'my_index';
const query = 'John Doe';
searchDocuments(index, query);
```

Search by age query match

```
async function search_by_age(index2, query2) {
  const { body } = await client.search({
    index,
    body: {
      query: {
        match: {
          'age': query2
        }
      }
    }
  });
  console.log('Search results:', body.hits.hits);
}


const index2 = 'my_index';
const query2 = 32;
search_by_age(index2, query2);
```

explaination

Certainly! Let's break down the search query we used earlier and provide an explanation for each part:

index: This is the parameter passed to the searchDocuments function and represents the name of the index in which the search will be performed. You can specify the index you want to search within your Elasticsearch cluster.

query: This is also a parameter passed to the searchDocuments function and represents the search query string that you want to match against the documents in the specified index.

client.search: This method is called on the Elasticsearch client instance to perform the search operation. It takes an object as a parameter with various options, including the index to search and the query body.

body: The body object is the request body of the search operation. In this case, we specify a query field that contains the specific query we want to execute.

match: The match query is used to search for documents that contain a specific value in a specific field. In the example code, <field_name> represents the name of the field you want to match against, and query represents the search query string.

body.hits.hits: This is the response from the Elasticsearch server, and it contains the search results. The hits object contains an array of matched documents (hits.hits) that you can access and work with.

So, in summary, the search query we used is a simple match query that searches for documents in the specified index where a specific field (<field_name>) matches the provided query string.

Remember to customize the <field_name> according to your index's field structure, and adjust the query string to match your search criteria.

Ingest data
After connecting to your deployment, you are ready to index and search data. Let's create a new index, insert some quotes from our favorite characters, and refresh the index so that it is ready to be searched. A refresh makes all operations performed on an index since the last refresh available for search.

```
const { Client } = require('@elastic/elasticsearch');
const client = new Client({
  node:
'https://fd00c6370c694177acfcf3f8d663fb7d.us-central1.gcp.clo
ud.es.io',
  auth: {
    username: 'elastic',
    password: 'xhLARutkh41NExTC3iLODkuT'
  },
  ssl: {
    rejectUnauthorized: false
  }
});

client.info()
  .then(response => console.log(response))
  .catch(error => console.error(error))

async function run() {
  await client.index({
    index: 'game-of-thrones',
    body: {
      character: 'Ned Stark',
    quote: 'Winter is coming.'
    }
  })

  await client.index({
    index: 'game-of-thrones',
    body: {
      character: 'Daenerys Targaryen',
    quote: 'I am the blood of the dragon.'
    }
  })

  await client.index({
    index: 'game-of-thrones',
    body: {
      character: 'Tyrion Lannister',
    quote: 'A mind needs books like a sword needs whetstone.'
    }
  })

  await client.indices.refresh({index: 'game-of-thrones'})
}

run().catch(console.log)
```

Search_Ingest.js

After creating a new index and ingesting some data, you are now ready to search. Let’s find what characters have said things about winter:

```
const { Client } = require('@elastic/elasticsearch');
const client = new Client({
  node:
'https://fd00c6370c694177acfcf3f8d663fb7d.us-central1.gcp.clo
ud.es.io',
  auth: {
    username: 'elastic',
    password: 'xhLARutkh41NExTC3iLODkuT'
  },
  ssl: {
    rejectUnauthorized: false
  }
});

client.info()
  .then(response => console.log(response))
  .catch(error => console.error(error))

async function run() {
  await client.index({
    index: 'game-of-thrones',
    body: {
      character: 'Ned Stark',
    quote: 'Winter is coming.'
    }
  })

  await client.index({
    index: 'game-of-thrones',
    body: {
      character: 'Daenerys Targaryen',
    quote: 'I am the blood of the dragon.'
    }
  })
  await client.index({
    index: 'game-of-thrones',
    body: {
      character: 'Tyrion Lannister',
    quote: 'A mind needs books like a sword needs whetstone.'
    }
  })

  await client.indices.refresh({index: 'game-of-thrones'})
}

run().catch(console.log)
```

The search request returns content of documents containing 'winter' in the quote field, including document IDs that were automatically generated. You can make updates to specific documents using document IDs

3)Aggregations & Analytics
Elasticsearch allow you to perform advanced data analysis on your indexed data. Aggregations provide you with the ability to compute metrics, create histograms, generate summaries, and gain valuable insights from your data. Here's a step-by-step guide to get started:

Understand Aggregations:

Familiarize yourself with the concept of aggregations in Elasticsearch. Aggregations are similar to SQL's GROUP BY clause and can be used to perform calculations and statistical operations on your data.

Update Mapping:
Ensure that your index mapping includes the appropriate data types for the fields you want to aggregate. If necessary, update your index mapping to define fields as numeric or date types to enable accurate aggregations.

Perform Aggregations:
Choose the type of aggregation you want to perform based on your requirements. Elasticsearch provides various types of aggregations, including:

Metric Aggregations: Calculate metrics like sum, average, min, max, etc. Examples include sum, avg, min, max, stats, etc.
Bucket Aggregations: Group data into buckets based on specific criteria. Examples include terms, date_histogram, histogram, range, etc.
Pipeline Aggregations: Perform calculations on the output of other aggregations. Examples include derivative, moving_avg, cumulative_sum, etc.
Implement Aggregations in Node.js:
Update your Node.js application (app.js or any other file) to include the code for performing aggregations. Here's an example:

Syntax

```
async function performAggregation(index) {
  const { body } = await client.search({
    index,
    body: {
      size: 0,
      aggs: {
        <aggregation_name>: {
          <aggregation_type>: {
            field: '<field_name>'
          }
        }
      }
    }
  });
  console.log('Aggregation results:', body.aggregations);
}
```

Example

Here's an example query that demonstrates how to perform a metric aggregation to calculate the average price for products in an Elasticsearch index:

```
async function performAggregation(index) {
  const { body } = await client.search({
    index,
    body: {
      size: 0,
      aggs: {
        average_price: {
          avg: {
            field: 'price'
          }
        }
      }
    }
  });
  console.log('Average price:',
body.aggregations.average_price.value);
}
```
Explaination

The index parameter represents the name of the Elasticsearch index you want to perform the aggregation on.

The aggregation is a metric aggregation of type avg (average).

The field parameter is set to 'price', indicating that the aggregation will be performed on the 'price' field of the indexed documents.

The size parameter is set to 0 to indicate that no documents need to be returned in the search response. We are only interested in the aggregation result.

The result of the aggregation is logged to the console using console.log.

To use this example, make sure to replace 'price' with the actual field name in your index that represents the price of the products.

Feel free to modify this example based on your specific requirements and experiment with different aggregations, fields, and parameters.

If you have any further questions or need assistance with specific aggregations or queries, please let me know.

Create a new index names - products stores all the super market data into the products index as documents
super_market_ingestion.js

```
const { Client } = require('@elastic/elasticsearch');
const client = new Client({
  node:
'https://fd00c6370c694177acfcf3f8d663fb7d.us-central1.gcp.cloud.es.io',
  auth: {
    username: 'elastic',
    password: 'xhLARutkh41NExTC3iLODkuT'
  },
  ssl: {
    rejectUnauthorized: false
  }
});

client.info()
  .then(response => console.log(response))
  .catch(error => console.error(error))
```

```
async function indexDocument(index, document) {
    const { body } = await client.index({
      index,
      body: document
    });

    console.log('Document indexed:', body);
  }

  const index = 'products';

  const document1 = { name: 'Chocolates', price:
100,count_per_packet:4 };

  const document2 = { name: 'biscuits', price:
183,count_per_packet:20 };

  const document3 = { name: 'cakes', price:
201,count_per_packet:6 };

  const document4 = { name: 'ice_cream', price:
165,count_per_packet:7 };

  const document5 = { name: 'chips', price:
192,count_per_packet:50 };

  const document6 = { name: 'cookies', price:
199,count_per_packet:5 };
```

```
  indexDocument(index, document1);
  indexDocument(index, document2);
  indexDocument(index, document3);
  indexDocument(index, document4);
  indexDocument(index, document5);
  indexDocument(index, document6);
```

super_market_aggregations.js

```
const { Client } = require('@elastic/elasticsearch');
// node is elastic search endpoint
const client = new Client(
    {
        node:
'https://fd00c6370c694177acfcf3f8d663fb7d.us-central1.gcp.clo
ud.es.io',
        auth: {
                username: 'elastic',
                password: 'xhLARutkh41NExTC3iLODkuT'
              },
        ssl: {
            rejectUnauthorized: false
            }
    }
    );


async function performAggregation(index) {
  const { body } = await client.search({
    index,
    body: {
      size: 0,
      aggs: {
        average_price: {
          avg: {
            field: 'price'
          }
        }
      }
    }
  });
  console.log('Average price:',
body.aggregations.average_price.value);
}

performAggregation('products').catch(console.log);
```

syntax for function call

```
// Example usage of the performAggregation() function
performAggregation('your_index_name');
```

Aggregations for super_market data (index:products)
Min price,Max price,Avg price of the documents in the products index

```
const { Client } = require('@elastic/elasticsearch');
// node is elastic search endpoint
const client = new Client(
  {
     node:
'https://fd00c6370c694177acfcf3f8d663fb7d.us-central1.gcp.clo
ud.es.io',
     auth: {
          username: 'elastic',
          password: 'xhLARutkh41NExTC3iLODkuT'
         },
     ssl: {
        rejectUnauthorized: false
        }

  }
  );


async function performAggregation(index) {
 const { body } = await client.search({
  index,
  body: {
   size: 0,
      aggs: {
        average_price: {
          avg: {
            field: 'price'
          }
        }
      }
    }
  });
  console.log('Average price:',
body.aggregations.average_price.value);
}

performAggregation('products').catch(console.log);

async function performAggregation2(index) {
    const { body } = await client.search({
      index,
      body: {
        size: 0,
        aggs: {
          min_price: {
            min: {
              field: 'price'
            }
          }
        }
      }
    });
    console.log('Minimum price:',
body.aggregations.min_price.value);
  }

  performAggregation2('products').catch(console.log);

  async function performAggregation3(index) {
    const { body } = await client.search({
      index,
      body: {
        size: 0,
        aggs: {
          max_price: {
            max:{
              field: 'price'
            }
          }
        }
      }
    });
    console.log('Maximum price:',
body.aggregations.max_price.value);
  }

  performAggregation3('products').catch(console.log);
```

getting document id by query

```
async function getDocumentIdByQuery(index, query) {
  const { body } = await client.search({
    index,
    body: {
      query
    }
  });

  const documentId = body.hits.hits[0]._id;
  return documentId;
}
const documentId = await
getDocumentIdByQuery('your_index_name', { term: { field: 'value' }
});
```

Getting document by ID and update the documents in the index by query

```
async function getDocumentIdByQuery(index, query) {
  const { body } = await client.search({
    index,
    body: {
      query
    }
  });

  if (body.hits.total.value === 0) {
    throw new Error('Document not found');
  }
  const documentId = body.hits.hits[0]._id;
  return documentId;
}

async function updateDocumentByQuery(index, query,
updateFields) {
  const documentId = await getDocumentIdByQuery(index, query);

  const { body } = await client.update({
    index,
    id: documentId,
    body: {
      doc: updateFields
    }
  });

  console.log('Document updated:', body.result);
}
call the function
const index = 'your_index_name';
const query = { term: { field: 'value' } };
const updateFields = { title: 'New Title', price: 99.99 };
updateDocumentByQuery(index, query, updateFields);
```

Certainly, here's an example of how you can retrieve the document_id using a search query and then use it to update a specific document:

javascript

Copy code
```
async function getDocumentIdByQuery(index, query) {
  const { body } = await client.search({
    index,
    body: {
      query
    }
  });

  if (body.hits.total.value === 0) {
    throw new Error('Document not found');
  }

  const documentId = body.hits.hits[0]._id;
  return documentId;
}

async function updateDocumentByQuery(index, query,
updateFields) {
  const documentId = await getDocumentIdByQuery(index, query);

  const { body } = await client.update({
    index,
    id: documentId,
    body: {
      doc: updateFields
    }
  });
  console.log('Document updated:', body.result);
}
```

In this example, we have two functions:

getDocumentIdByQuery: This function takes the index parameter, which represents the name of the Elasticsearch index, and the query parameter, which represents the search query to find the document. It retrieves the _id of the first matching document from the search results.

updateDocumentByQuery: This function takes the index parameter, query parameter, and updateFields parameter. It calls the getDocumentIdByQuery function to retrieve the document_id based on the provided query and then performs the update operation using the obtained document_id.

To use these functions, you can call updateDocumentByQuery with the appropriate parameters:
Replace 'your_index_name' with the name of your Elasticsearch index, { term: { field: 'value' } } with a valid search query to match the desired document, and { title: 'New Title', price: 99.99 } with the fields you want to update and their new values.

The updateDocumentByQuery function retrieves the document_id based on the provided query and then updates the document with the specified fields.

get_document_by_id_and_update.js

```
const { Client } = require('@elastic/elasticsearch');
// node is elastic search endpoint
const client = new Client(
   {
      node:
'https://fd00c6370c694177acfcf3f8d663fb7d.us-central1.gcp.clo
ud.es.io',
      auth: {
            username: 'elastic',
            password: 'xhLARutkh41NExTC3iLODkuT'
           },
      ssl: {
         rejectUnauthorized: false
         }

   }
   );


   // getting document id with the query within the index and
updating its name and price
   async function getDocumentIdByQuery(index, query) {
      const { body } = await client.search({
        index,
        body: {
          query
        }
      });
```

```
    if (body.hits.total.value === 0) {
      throw new Error('Document not found');
    }

    const documentId = body.hits.hits[0]._id;
    return documentId;
  }

  async function updateDocumentByQuery(index, query,
updateFields) {
    const documentId = await getDocumentIdByQuery(index,
query);

    const { body } = await client.update({
      index,
      id: documentId,
      body: {
        doc: updateFields
      }
    });

    console.log('Document updated:', body.result);
  }

//   index is products index
  const index = 'products';
//   getting all the documents named 'Chocolates' and updating
its name to choco chips and price from 100 to 99.99
  const query = { term: { name: 'Chocolates' } };
  const updateFields = { name: 'choco-chips', price: 99.99 };

  updateDocumentByQuery(index, query, updateFields);
```

Send Mail Using Js CODE

```
module.exports = function (RED) {
    function NodemailerNode(config) {
      RED.nodes.createNode(this, config);

      const nodemailer = require("nodemailer");

      this.on("input", function (msg) {
        const transporter = nodemailer.createTransport({
            service: "Gmail",
            auth: {
              user: "mailsurya366@gmail.com", // Replace with your Gmail
email address
              pass: "usdllwcuzganudnq", // Replace with your Gmail
password or an App Password if you have 2-step verification enabled
            },
          });

          // Example usage:
const apiResponse = {
    status: 500,
    data: "Some error occurred.",
  };

const htmlContent = `
  <h1>API Failure Notification</h1>
  <p>Dear User,</p>
  <p>The API call was not successful. The status code received was:
${apiResponse.status}</p>
  <p>Please take appropriate action to resolve the issue.</p>
  <p>Best regards,</p>
  <p>Development Team</p>
`;

          const mailOptions = {
            from: "mailsurya366@gmail.com",
            to: "sukusuku366@gmail.com",
            subject: "API Failure Notification",
            html: htmlContent,
          };

        transporter.sendMail(mailOptions, function (error, info) {
          if (error) {
            console.log("Error sending email:", error);
          } else {
            console.log("Email sent:", info.response);
          }
        });
      });
    }

    RED.nodes.registerType("nodemailer-node", NodemailerNode);
  };
```

```
{
  "name": "node-red-contrib-nodemailer",
  "version": "0.1.0",
  "description": "Custom Node-RED node for sending emails using
Nodemailer",
  "keywords": ["node-red", "nodemailer", "email"],
  "author": "mailsurya366@gmail.com",
  "dependencies": {
    "nodemailer": "^7.3.0"
  },
  "node-red": {
    "nodes": {
      "nodemailer-node": "nodemailer-node.js"
    }
  }
}
```