Smart Farmer-IOT Enabled Smart Farming Application

Build A Web Application Using Node -Red

TEAM ID : PNT2022TMID31166

Team Members: SATHIYAPRIYA.S (621519104069)

VIDHYA.A (621519104085)

PRADEEPA.P (621519104058)

PRIYA.P (621519104063)

Node-RED is an open-source visual flow-based programming tool used for wiring not only Internet of Things (IoT) components, but also integrating an ensemble of service APIs, including ones provided by IBM Cloud. A node in Node-RED performs a particular functionality, which typically minimizes the amount of coding that is required to build a given application. If you've never used Node-RED before, you might want to start by reviewing these "Node-RED Essentials videos.

Create a Node-RED starter application

After the status of your application changes to Running, click Visit App URL. Click Go to your Node-RED flow editor. A new flow called Flow 1 is opened in the Node-RED flow editor. If you secured your Node-RED flow editor, you will first be asked to enter the username and password that you just set up

Develop the Application

The web service component

The dashboard component

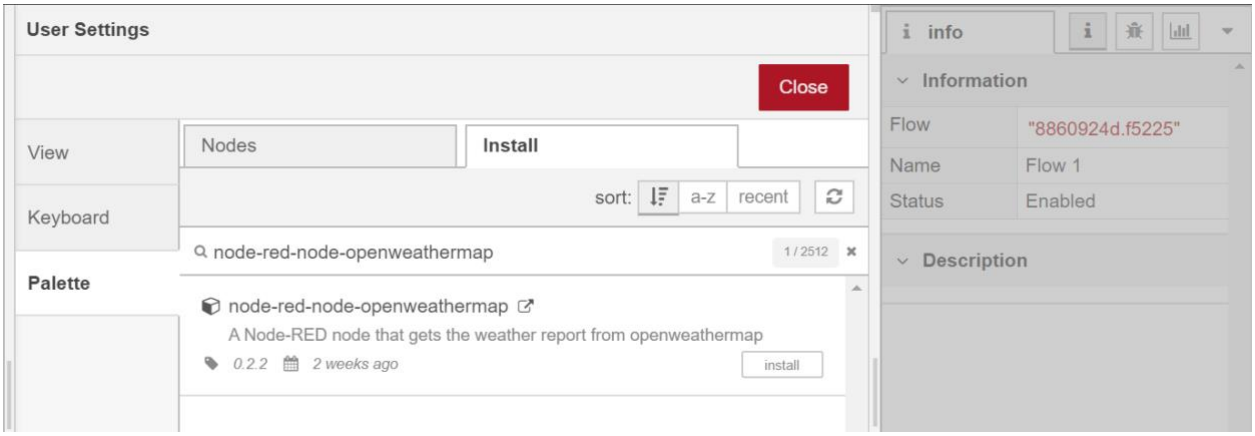
The steps to create these two components in our Node-RED app follow. I encourage you to follow the steps to learn just how easy it is to create apps using Node-RED.

You can also import both of the flows explained in this tutorial. First, download or copy the contents of the flows.json file to the clipboard. Then, go to the hamburger menu in your Node-RED editor, and select Import > Clipboard. Then, paste the contents into the dialog, and click Import. In case you selected to download the file, make sure to select select a file to import, then click Import. You'll still need to follow the steps in this tutorial to configure all the nodes and to make a package available to a function node.

Create the Web Service

Double-click the tab with the flow name, and call it Earthquake Details.

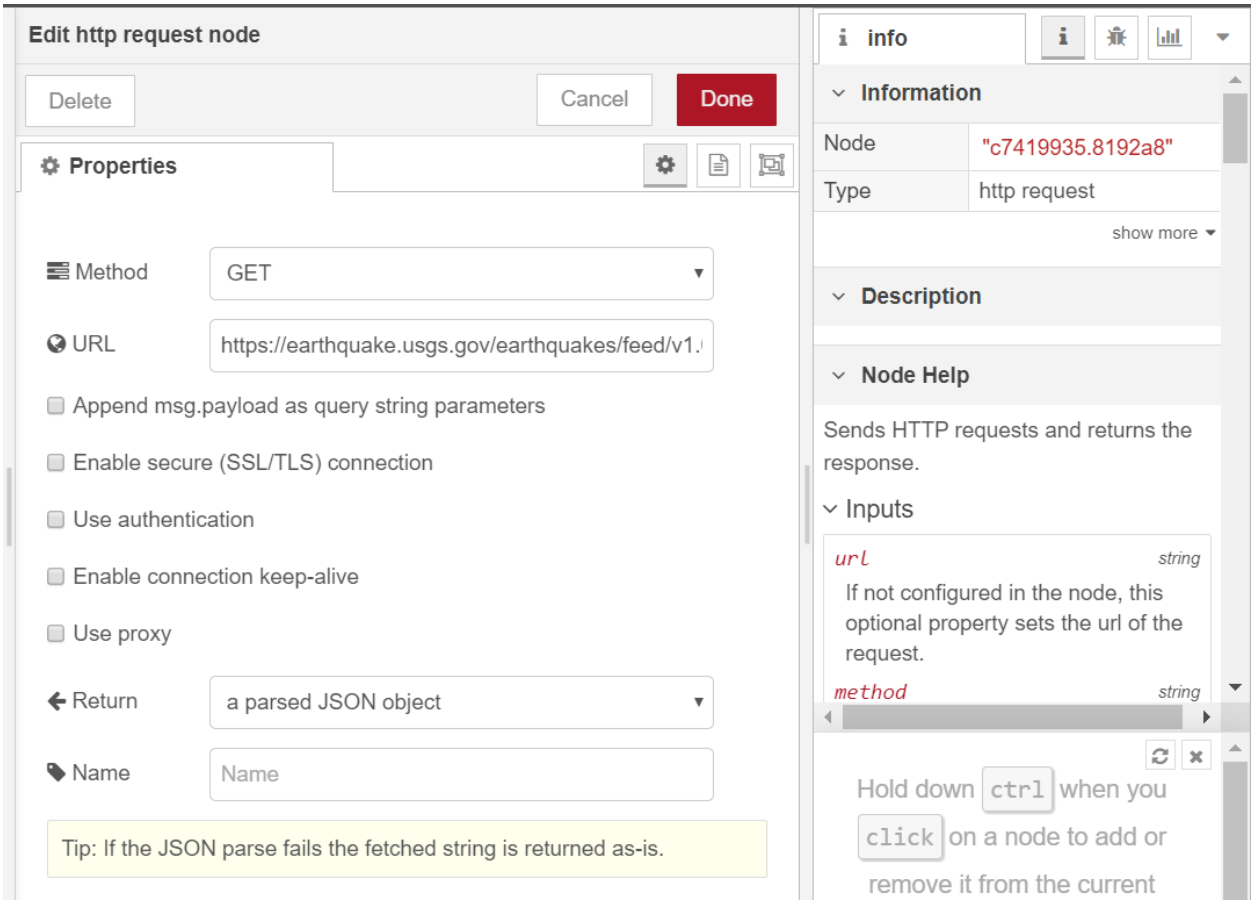
- Click the hamburger menu, and then click Manage palette. Look for node-red-node-openweathermap to install these additional nodes in your palette.
- Screen capture of Node-RED palette settings



- Add an HTTP input node to your flow.
- Double-click the node to edit it. Set the method to GET and set the URL to /earthquakeinfo-hr.
- Add an HTTP response node, and connect it to the previously added HTTP input node. All other nodes introduced in this sub-section is to be

added between the HTTP input node and the HTTP response node.

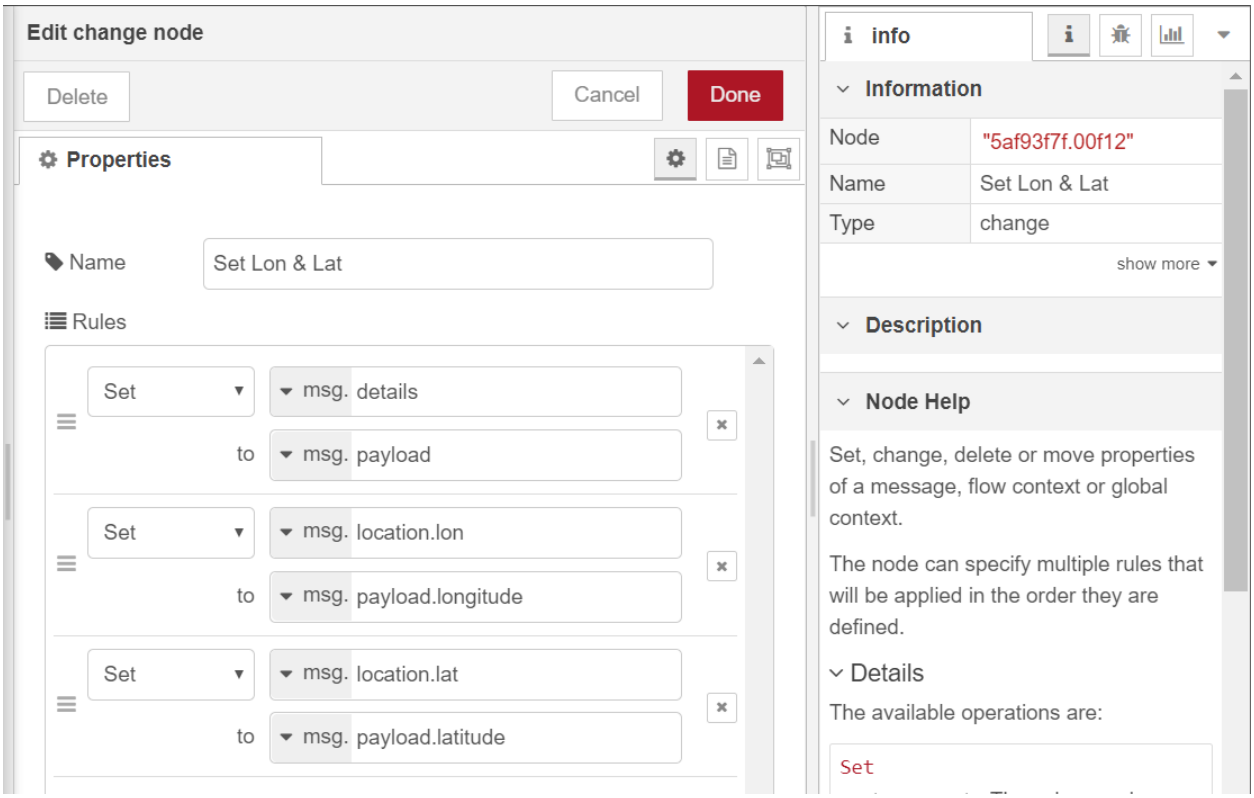
- Add an HTTP request node and set the URL to https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_hour.geojson, the Method to GET and the Return to a parsed JSON object. This will allow extracting all earthquakes that occurred within the last hour. Name this node Get Earthquake Info from USGS.



- Add a change node. Double-click the node to modify it. Name this node Set Earthquake Info. In the Rules section, add rules to Delete msg.topic, msg.headers, msg.statusCode, msg.responseUrl and msg.redirectList and Set msg.payload to the following JSONata expression.

```
Payload.features.{
  “type”:properties.type,
  “magnitude”: properties.mag,
  “location”: properties.place,
  “longitude”:geometry.coordinates[0],
  “latitude”:geometry.coordinates[1],
  “depth”:geometry.coordinates[2],
  “timestamp”: $fromMillis(
    Properties.time,
    ‘[H01]:[m01]:[s01] [z]’,
    ‘+0400’
  ),
  “source”: properties.net
}
```

- Add a **split** node, which will split the earthquake information points into different messages based on the location.
- Add a **change** node to set the longitude and latitude to the right properties, which will get fed into the **openweathermap** node. In the **Rules** section, you will *Set* `msg.details` to `msg.payload`, `msg.location.lon` to `msg.payload.longitude`, and `msg.location.lat` to `msg.payload.latitude`.Name this node `Set Lon & Lat`



- Add an **openweathermap** node to which you will be adding the API key from the OpenWeatherMap site. (Log in to the site with the account you created.)
- Add another **change node** that will *Set* the `msg.payload` to the output of a JSONata expression that will format the messages correctly. Name the node `Add Weather Data`. The JSONata expression is as follows.

```
msg.{
   "name": parts.index,
   "Place": details.place,
   "Location":details.location,
   "Country":location.country,
   "mag":details.magnitude,
   "Source":details.source,
  "Timestamp":details.timestamp,
  "lon": location.lon,
      "lat":location.lat,
  "Type":details.type,
   "Temperature":data.main.temp & '
Kelvin',
   "Pressure":data.main.pressure & '
hPa',
   "Humidity":data.main.humidity & '
%',
```

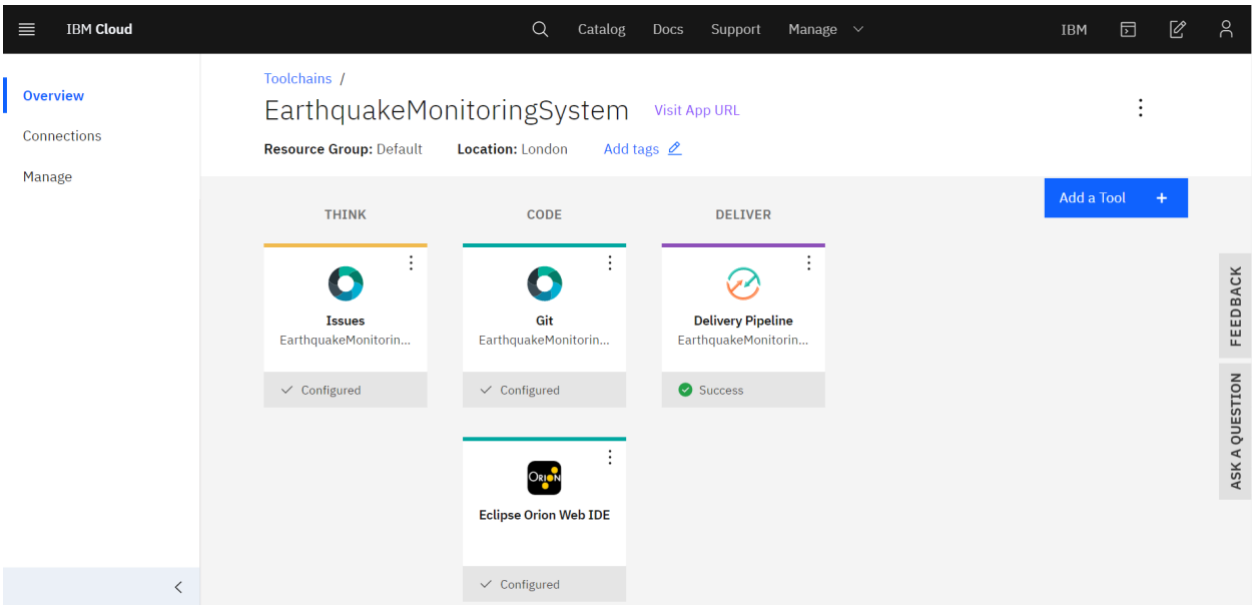
```
   "Wind Speed":data.wind.speed & '
meter(s)/sec',
   "Wind Direction":data.wind.deg & '
degree(s)',
   "Cloud Coverage":data.clouds.all & '
%',
   "icon":'earthquake',
   "intensity":details.magnitude / 10
}
```

- Click Deploy for all changes to take effect.

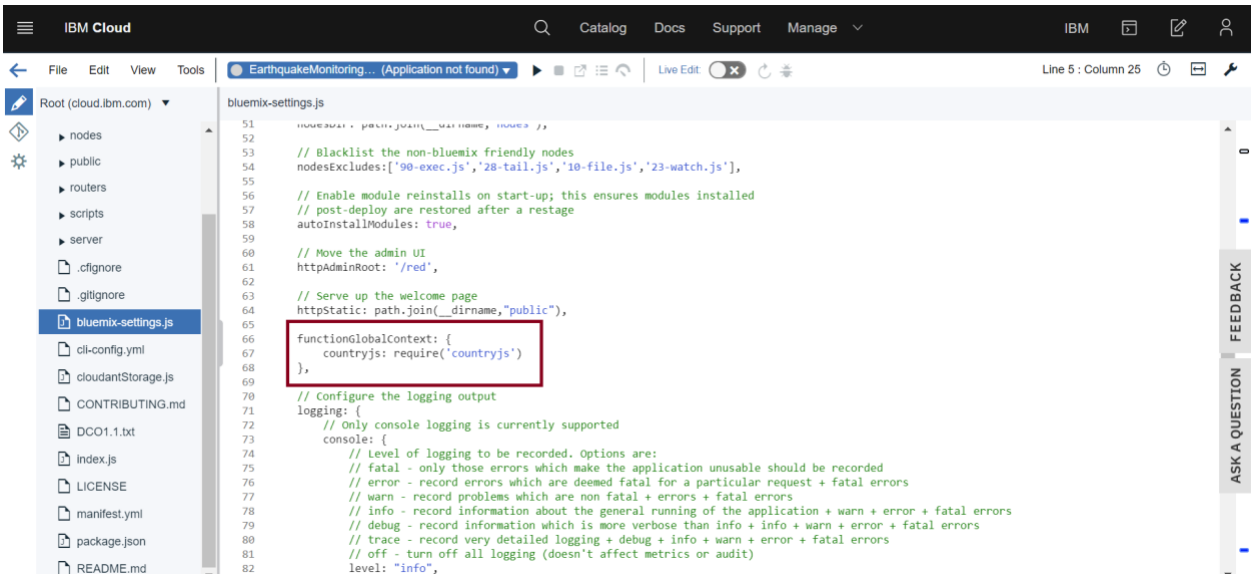
Adding the countryjs Package

- Before being able to finish the web service component, you need to make the countryjs package available to the function node, which will allow us to set the region and country per earthquake point.
- Go back to the IBM Cloud application that you created.
  Click Overview. In the Continuous delivery section, click View toolchain.
- Now, you need to configure the files to make the countryjs package available to the function node. In the Toolchains window, do one of the following steps:

- Clone the repository from the Git action, and make the edits to the files locally.
  Open the Eclipse Orion Web IDE to edit the files in the IBM Cloud.



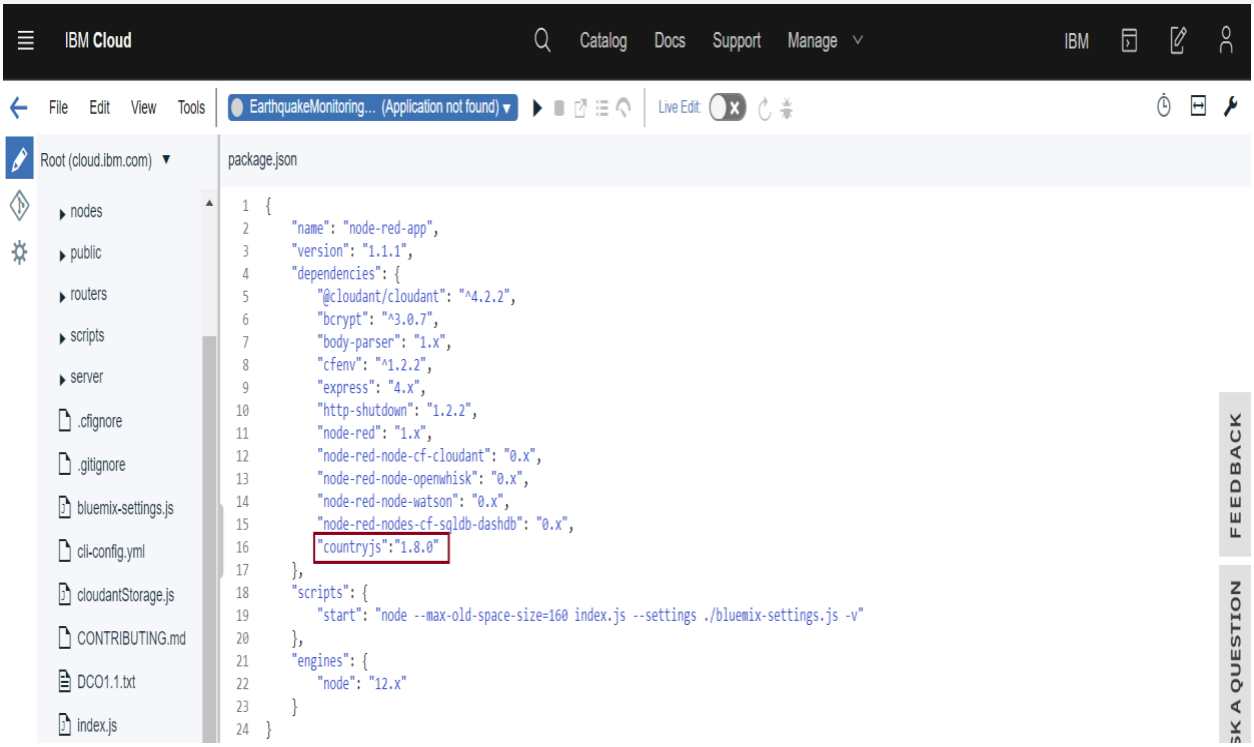
- Edit the `bluemix-settings.js` file



- Find the definition of the functionGlobalContext object, and add countryjs.
  functionGlobalContext: {
  countryjs:require('countryjs')
  },

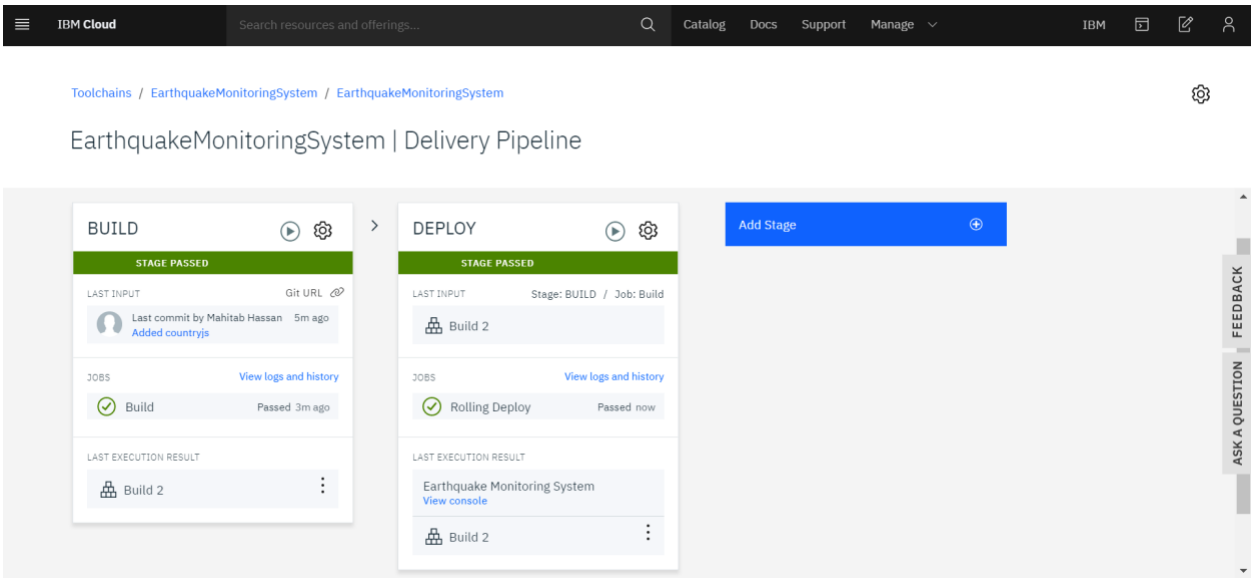
- Edit the `package.json` file, and define `countryjs` as a dependency.

```
"dependencies": {
    ...,
    "countryjs":"1.8.0"
},
```



- Use Git to commit and push all changes.

- Go back to your application, and wait for your application to finish deploying. You can check that by looking at the Delivery Pipeline card.



Finishing the Web Service

Add a function node after the change node that you previously added. Name the function node Set Region & Country using countryjs. Add the following javascript code to the node.

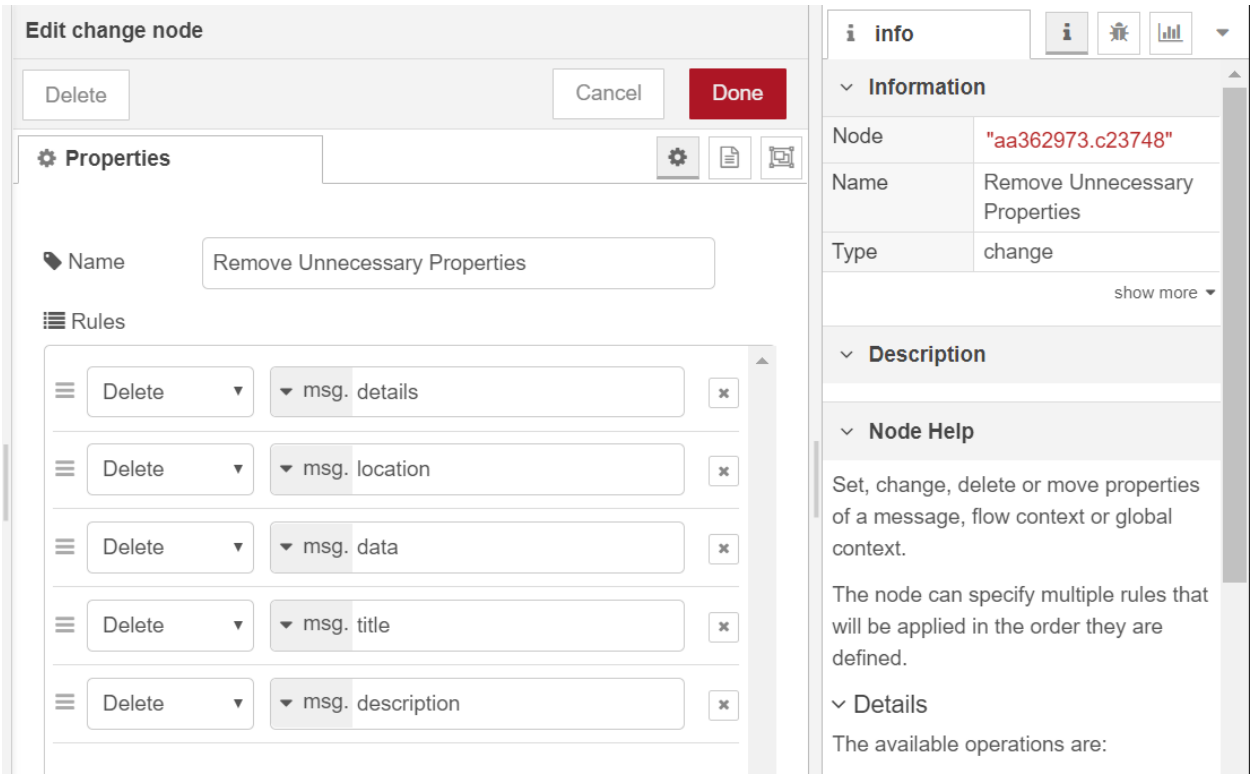
Var countryjs = global.get('countryjs');

Msg.payload.Country = countryjs.name(msg.location.country)

Msg.payload.Region = countryjs.region(msg.location.country)

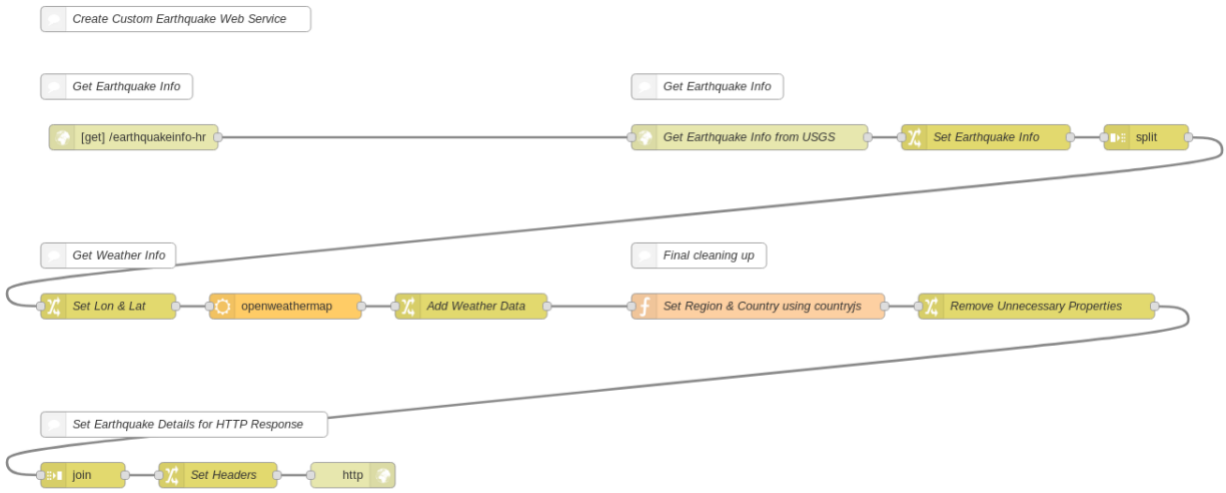
Return msg;

- Add a change node to remove any redundant properties. In the Rules section, add rules to Delete msg.details, msg.location, msg.data, msg.title and msg.description. Name this node Remove Unnecessary Properties.



- Add a join node to join the previous split message into a single one again, which will be returned when an HTTP request is submitted.

- Add a change node and name it Set Headers. In the Rules section, add a rule to Set msg.headers to { “Content-Type”: “application/JSON” } to define that HTTP request to the web service will be returned in a JSON format.
  After cleaning up the flow, your flow should look similar to the following.



- Click Deploy for all changes to take effect.