

OpenBlocks IoT Family向け Node-REDスターターガイド



Ver.3.2.0

ぷらっとホーム株式会社

■ 商標について

- ・ Linux は、Linus Torvalds 氏の米国およびその他の国における商標あるいは登録商標です。
- ・ 文中の社名、商品名等は各社の商標または登録商標である場合があります。
- ・ その他記載されている製品名などの固有名詞は、各社の商標または登録商標です。

■ 使用にあたって

- ・ 本書の内容の一部または全部を、無断で転載することをご遠慮ください。
- ・ 本書の内容は予告なしに変更することがあります。
- ・ 本書の内容については正確を期するように努めていますが、記載の誤りなどにご指摘がございましたら弊社サポート窓口へご連絡ください。
また、弊社公開の WEB サイトにより本書の最新版をダウンロードすることが可能です。
- ・ 本装置の使用にあたっては、生命に関わる危険性のある分野での利用を前提とされていないことを予めご了承ください。
- ・ その他、本装置の運用結果における損害や逸失利益の請求につきましては、上記にかかわらずいかなる責任も負いかねますので予めご了承ください。

目次

第 1 章 はじめに	4
第 2 章 Node-RED 事前準備.....	5
2-1. Node-RED のインストール.....	5
2-2. Node-RED 起動設定.....	6
2-3. Node-RED のブラウザフィルタについて	7
2-4. パケットフィルタについて.....	7
第 3 章 Node-RED の簡易説明.....	8
3-1. Node-RED 画面構成.....	9
3-2. ノード種類.....	10
3-2. 入力ノード.....	10
3-3. 出力ノード.....	11
3-4. 機能ノード.....	12
3-5. ソーシャルノード.....	13
3-6. ストレージノード.....	13
3-7. 分析ノード.....	14
3-8. その他ノード.....	14
3-9. cloud ノード.....	14
3-10. GatewayKit ノード.....	14
3-11. location ノード.....	15
3-12. dashboard ノード.....	15
3-13. Google ノード.....	16
第 4 章 ノード操作サンプル.....	17
4-1. 位置情報の可視化.....	17
4-2. ビーコンデータへの位置情報付与.....	22
第 5 章 その他.....	29
5-1. Node-RED ログについて.....	29
5-2. Node-RED へのノード追加.....	30
5-3. Node-RED の Config 編集.....	31
5-4. Node-RED のプロセス状況について.....	32
5-5. Node-RED の HTTPS 化.....	33

第 1 章 はじめに

本書は、OpenBlocks IoT Family に搭載可能な Node-RED の使用方法を解説しています。搭載している Node-RED は IoT データ制御機能にて送信先の候補として用意しており、エッジコンピューティングの実装や対応していないクラウドへの対応を想定しております。

第2章 Node-RED 事前準備

2-1. Node-RED のインストール

本製品出荷時では、Node-RED はインストールされておられません。そのため、WEB UI の「メンテナンス」→「機能拡張」タブから Node-RED をインストールしてください。



WEB UI の「メンテナンス」タブを選び、さらに「機能拡張」タブをクリックすると機能拡張用のパッケージを選択することができます。

© 2015 - 2017 PlatHome Co., Ltd. All rights reserved.



インストール機能のリストから「Node-RED」を選択します。

その後、インストールの「実行」ボタンを押し、インストールを行ってください。

尚、インストール完了後には反映を行うため、本体再起動が必要となります。そのため、「メンテナンス」→「停止・再起動」から本製品の再起動を行ってください。

2-2. Node-RED 起動設定

Node-RED のインストールが完了している場合、サービス WEB UI の「サービス」→「基本」タブに Node-RED の項目が表示されます。

この Node-RED のリンクを押して下さい。これにより、「Node-RED」タブへと遷移します。



「Node-RED」→「Node-RED」タブから Node-RED 自体の設定を行えます。

Node-RED

使用設定：

Node-RED の使用設定を行います。Node-RED を使用する場合には、「使用する」を選択してください。

ポート番号：

Node-RED へアクセスするためのポート番号を設定します。通常は、デフォルトの 1880 から変更する必要はありません。

ログイン認証設定：

ログイン認証を使用する場合には“使用する”を選択し、ユーザー名及びパスワードを設定し保存ボタンを押してください。

使用 Config 設定：

Node-RED 自体の起動設定 Config ファイルを本システムまたはユーザー編集したものと切り替えることができます。

「ユーザー定義コンフィグ」を選択した場合、コンフィグ編集用タブが表示されます。

設定完了後、保存ボタンを押すことにより Node-RED が起動・停止します。

※リンクボタンを押すことによって起動中の Node-RED へアクセスすることができます。ただし、AirManage 経由の場合にはアクセスは行えません。AirManage 経由の場合には URI プロキシ機能を用いてアクセスしてください。

2-3. Node-RED のブラウザフィルタについて

Node-RED のインストールが完了している場合、Node-RED 自体への WEB アクセスを行う場合、Node-RED のフィルターを開放する必要があります。そのため、WEB UI の「システム」→「フィルター」タブにてフィルターを開放してください。



デフォルトでは Node-RED のブラウザアクセスはできないようにフィルターが適用されています。

“有効”に設定し、保存ボタンを押してください。

尚、セキュリティの観点上、Node-RED の設定完了後はフィルターを閉鎖してください。

2-4. パケットフィルタについて

Openblocks IoT Family の入力方向のパケットフィルタは、WebUI へのアクセスや時刻同期等、システムの動作に必要となるポートを除き開放されておりません。

TCP 入力ノード 等、リモートからの接続を待ち受けるノードを使用し外部から受信する場合、必要に応じて別途パケットフィルタを開放する必要があります。

パケットフィルタの操作は、`/var/webui/local/bin/iptables-ext.sh` ファイルを作成し WEB UI の「メンテナンス」→「フィルター」タブから、iptables 編集するスクリプトを作成してください。

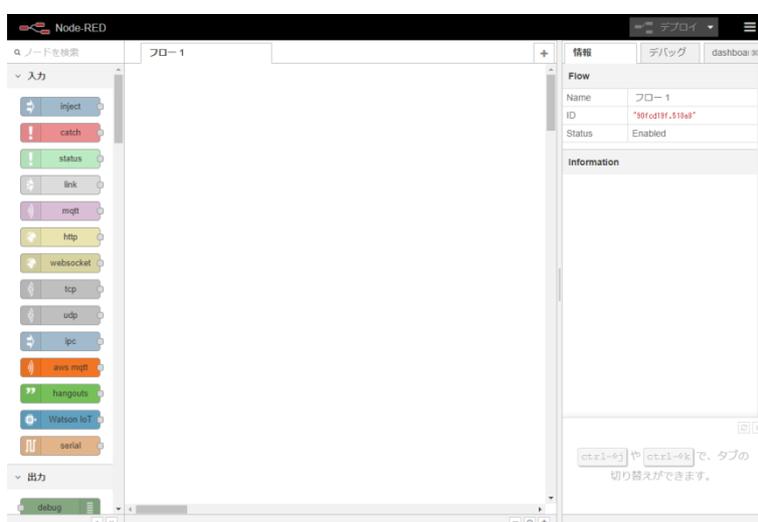
第 3 章 Node-RED の簡易説明

Node-RED はデフォルトで 1880 番ポートを用いてブラウザアクセス行います。

そのため、デフォルトでの Wi-Fi 経由での Node-RED へのアクセスする為の URL は以下となります。

`http://192.168.254.254:1880/`

アクセスした場合、以下のような画面が初期状態では表示されます。(ログイン認証設定をしていない場合)

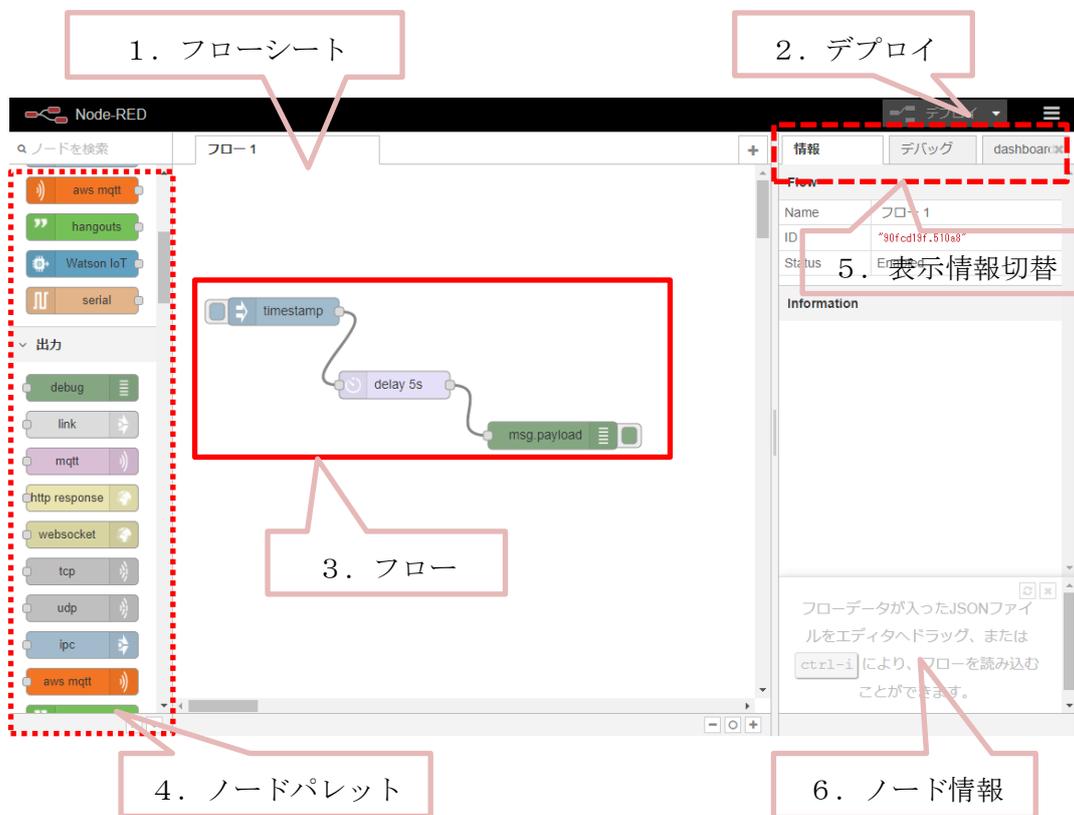


また、本製品向けにデフォルトで用意している入力・出力・処理等のノードは以下となります。



3-1. Node-RED 画面構成

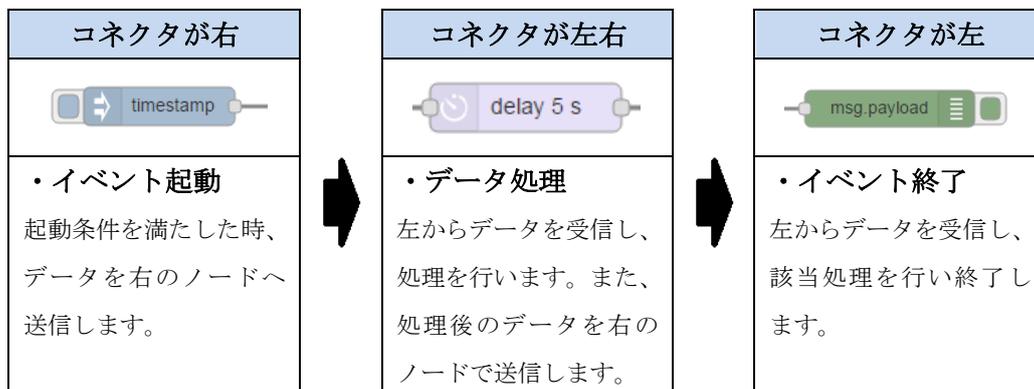
Node-RED の画面は以下のように構成されています。



#	項目	説明
1	フローシート	処理フローを記述するワークスペースです。
2	デプロイ	デプロイボタンをクリックすることでシートに記述した処理フローを有効化します。
3	フロー	ノードを配置し結線することでデータの流れ（処理フロー）を定義します。
4	ノードパレット	処理フローの構成に用いられるノードの一覧です。
5	表示切替	ノード情報・デバック情報の表示を切り替えます。
6	ノード情報	ノード情報、又はデバック情報が表示されます。

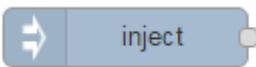
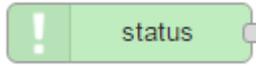
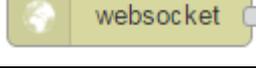
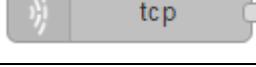
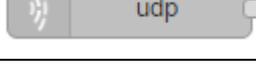
3-2. ノード種類

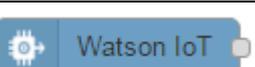
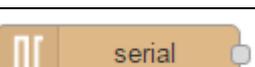
Node-RED では大きく分けて以下のようなコネクタ配置のノードがあります。



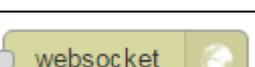
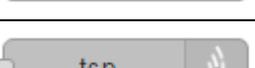
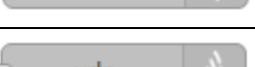
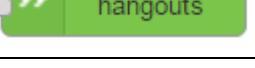
上記のように処理が行われるため、データは左から右へと処理が行われます。

3-2. 入力ノード

	ノードの左部のボタンを押すことでノードに設定された timestamp 等を入力データもしくはイベントとします。
	同じシート上のノードで発生したエラーを入力データもしくはイベントとします。
	同じシート上のノードのステータスを入力データもしくはイベントとします。
	いずれかの link output node の出力を入力データもしくはイベントとします。
	MQTT broker へ subscrib し、 publish message を待ち受け、入力データもしくはイベントとします。
	HTTP リクエストを待ち受け、入力データもしくはイベントとします。
	Websocket による接続を待ち受け、入力データもしくはイベントとします。
	TCP による接続を待ち受け、入力データもしくはイベントとします。
	UDP による接続を待ち受け、入力データもしくはイベントとします。

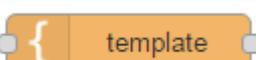
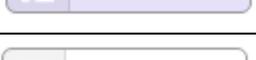
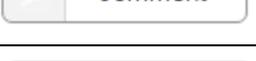
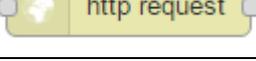
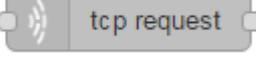
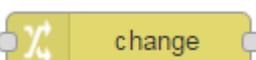
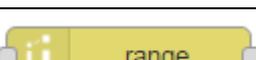
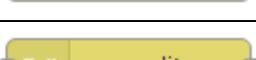
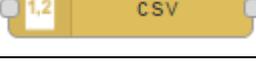
	Unix ドメインソケットによる接続を待ち受け、入力データもしくはイベントとします。
	AWS IoT からのメッセージを待ち受け、入力データもしくはイベントとします。
	Google hangouts からのメッセージを待ち受け、入力データもしくはイベントとします。
	Watson IoT からの device command を待ち受け、入力データもしくはイベントとします。
	シリアルインタフェースからの入力を待ち受け、入力データもしくはイベントとします。

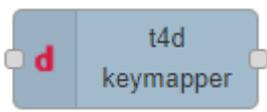
3-3. 出力ノード

	入力データをデバック情報として表示します。
	入力データをいずれかの Input link node へ出力します。
	入力データを MQTT broker へ publish します。
	入力データを http input node への入力に対する応答として出力します。
	入力データを Websocket サーバへ出力します。
	入力データを TCP サーバに出力します。
	入力データを UDP サーバに出力します。
	入力データを Unix ドメインソケットに出力します。
	入力データを AWS IoT に出力します。
	入力データを Google hangouts へ出力します。
	入力データを Watson IoT へ出力します。

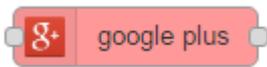
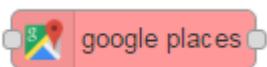
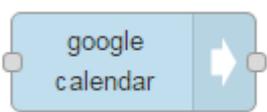
 serial	入力データをシリアルインタフェースへ出力します。
--	--------------------------

3-4. 機能ノード

 function	入力データを JavaScript で処理し出力します。
 template	入力データを整形して出力します。
 delay	入力データを設定された時間後に出力します。
 trigger	入力データに対し、タイムアウトを設け2つのメッセージを出力します。
 comment	フローにコメントを付けます。
 http request	入力データに対し、設定された URL に対し http リクエストを行い、その応答を出力します。
 tcp request	入力データに対し、設定されたサーバに対し TCP 接続を行い、その応答を出力します。
 switch	入力データを設定された分岐条件に応じて異なるノードに出力します。
 change	入力データの属性を設定・変更・削除もしくは移動して出力します。
 range	入力データのスケールを変更して出力します。
 split	入力データを設定される文字で区分して出力します。
 join	入力データを結合して出力します。
 csv	CSV 書式のデータと JavaScript Object を相互変換します。
 html	HTML 書式のデータと JavaScript Object を相互変換します。
 json	JSON 書式のデータと JavaScript Object を相互変換します。

	XML 書式のデータと JavaScript Object を相互変換します。
	YAML 書式のデータと JavaScript Object を相互変換します。
	入力データを Toami for DOCOMO 用の入力データへと変換します。
	AWS IoT の Thing Shadow 制御設定を行います。
	入力データが変化した場合のみ出力します。

3-5. ソーシャルノード

	電子メールを待ち受け、入力データもしくはイベントとします。
	Twitter からのメッセージを待ち受け、入力データもしくはイベントとします。
	入力データを設定された電子メールアドレスに送付します。
	入力データを Twitter へ出力します。
	Google plus に対する入出力を提供します。
	Google places に対する入出力を提供します。
	Google calendar に登録されている次のイベントを返します。

3-6. ストレージノード

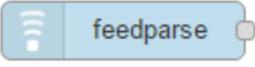
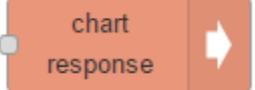
	設定されたファイルの末尾を入力データとします。
	入力データに示されるファイルを開き、その内容を出力します。

	設定されたファイルにデータを出力します。
---	----------------------

3-7. 分析ノード

	入力データを AFINN-111 単語リストを用いて感情分析(肯定的/否定的/中立) を行い出力します。
---	--

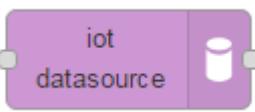
3-8. その他ノード

	ディレクトリまたはファイルの更新を監視し、入力イベントとします。
	RSS/Atom を監視し、Web コンテンツの更新を入力イベントとします。
	システムのコマンドを実行し、その出力を返します。
	Google chart にグラフ描画をリクエストします。
	Google chart のグラフ描画を出力します。

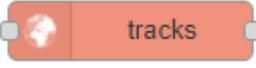
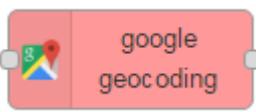
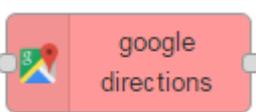
3-9. cloud ノード

	入力データを Azure IoT Hub へ出力します。
	入力データに示されるデバイスを Azure IoT Hub へ登録します。

3-10. GatewayKit ノード

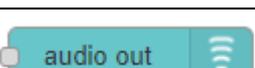
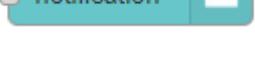
	入力データをダッシュボードアプリケーションへ出力します。
---	------------------------------

3-11. location ノード

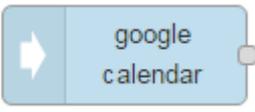
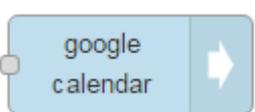
	入力データを WEB 上にプロットします。(インターネット接続が必要です。)
	Worldmap の出力 Web ページからイベントを受信します。
	指定した数の前の場所に基づいてトラックを作成します。
	Google geocoding を用いて入力データをジオコーディング（住所情報と地理的座標の相互変換）し、出力します。
	Google directions を用いて入力された出発地点と目的地点の経路を出力します。

3-12. dashboard ノード

	ユーザーインターフェースにボタンを追加します。
	ユーザーインターフェースにドロップダウン選択ボックスを追加します。
	ユーザーインターフェースにスイッチを追加します。
	ユーザーインターフェースにスライダーウィジェットを追加します。
	ユーザーインターフェースに数値入力ウィジェットを追加します。
	ユーザーインターフェースにテキスト入力ウィジェットを追加します。
	ユーザーインターフェースに日付選択ウィジェットを追加します。
	ユーザーインターフェースに色選択ウィジェットを追加します。
	ユーザーインターフェースにフォームを追加します。

	ユーザーインターフェースに編集不可能なテキストフィールドを表示します。
	ユーザーインターフェースにゲージタイプのウィジェットを追加します。
	ユーザーインターフェースに入力値をグラフにプロットします。
	ダッシュボードの音声またはテキストを音声で再生します。
	<code>msg.payload</code> をポップアップ通知または OK / Cancel ダイアログメッセージとしてユーザーインターフェイスに表示します。
	ダッシュボードを動的に制御できます。
	テンプレートウィジェットには、任意の有効な <code>html</code> および角度/角度 - 材料指示を含めることができます。

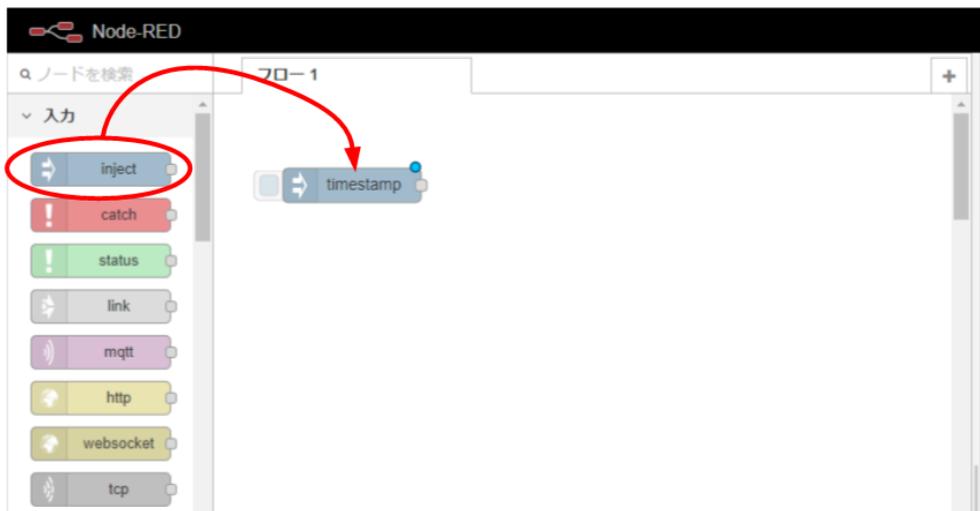
3-13. Google ノード

	Google calendar のイベント（予定の通知）の発生を待ち受けます。
	Google calendar に新しい予定（イベント）を登録します。

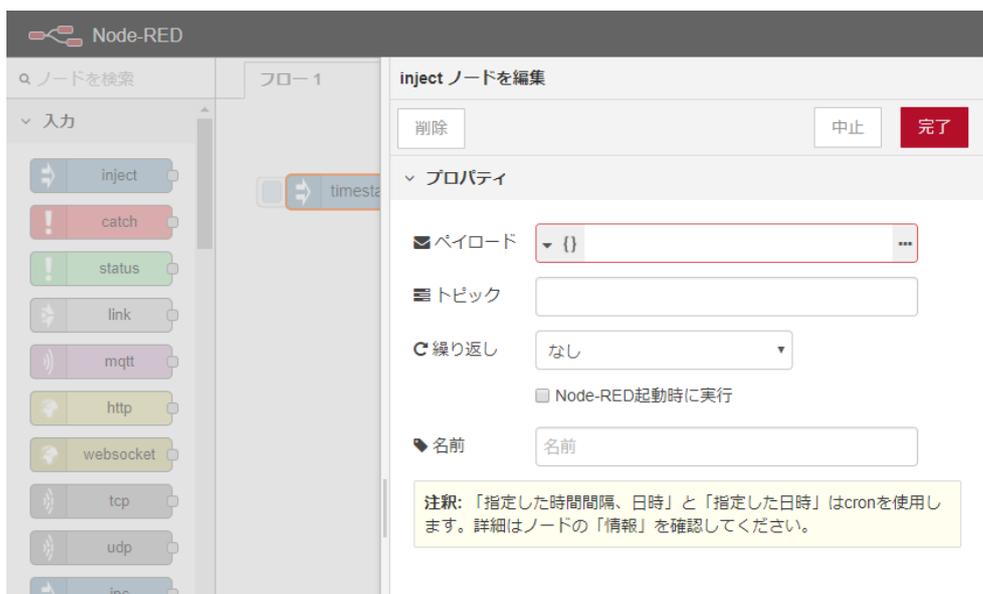
第4章 ノード操作サンプル

4-1. 位置情報の可視化

1. Node-RED にアクセスします。(第3章参照)
`http://192.168.254.254:1880/`
2. 入力ノードパレットから Inject ノードをドラッグしてシートにドロップします。



3. 配置した Inject ノードをダブルクリックし、ノードの編集を行います。

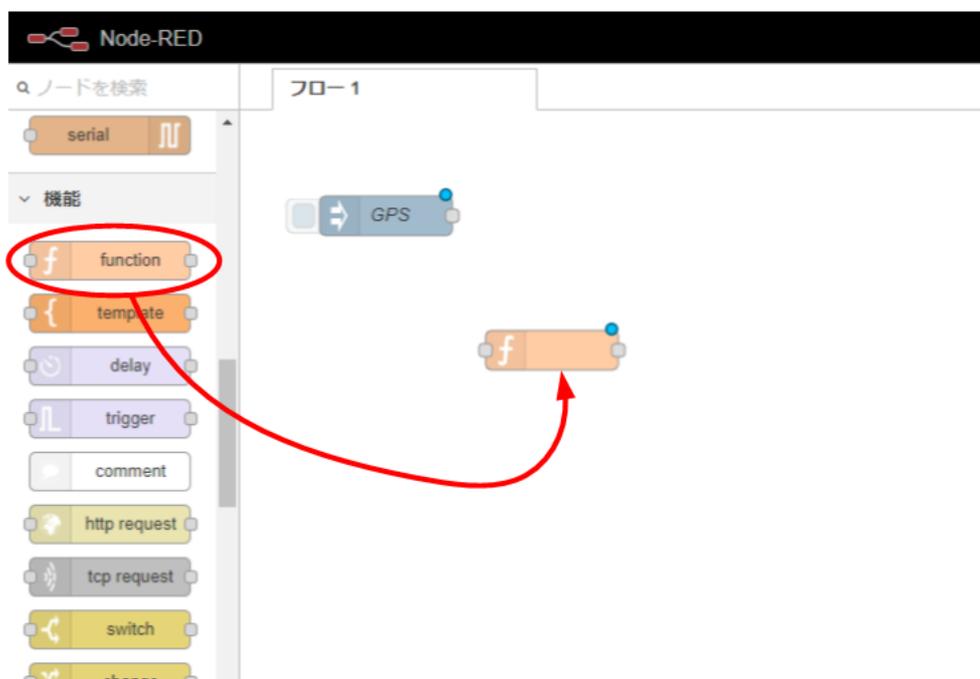


名前欄に GPS と入力します。ペイロードの種類に JSON を選択します。また、ペイロード部の『...』を選択し、表示されるウィンドウに以下を設定します。

```
{  
  "latitude": 35.681167,  
  "longitude": 139.767052  
}
```

設定後、「完了」ボタンを押します。

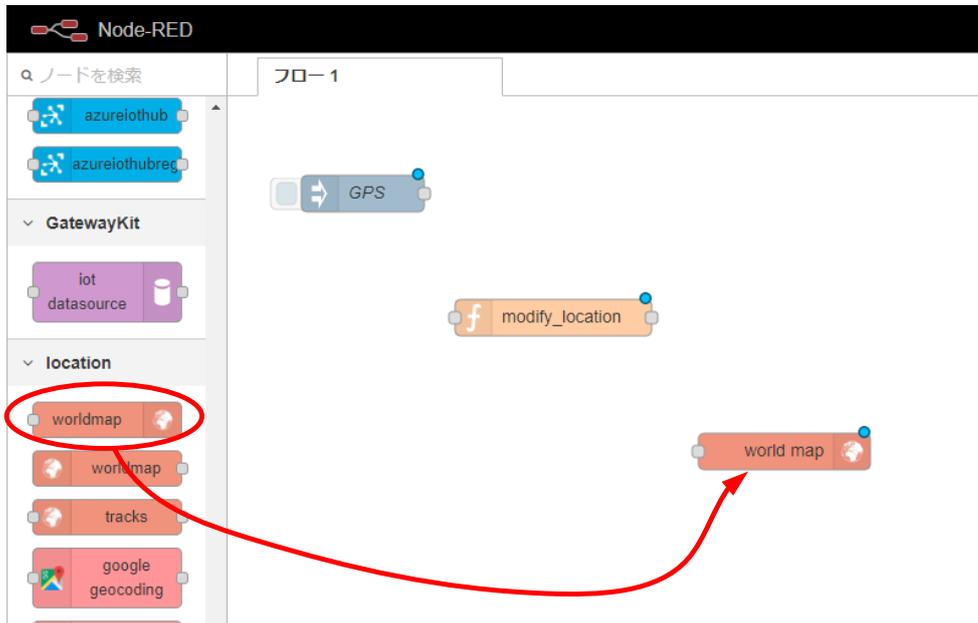
- 機能ノードから function ノードをドラッグしてシートにドロップします。



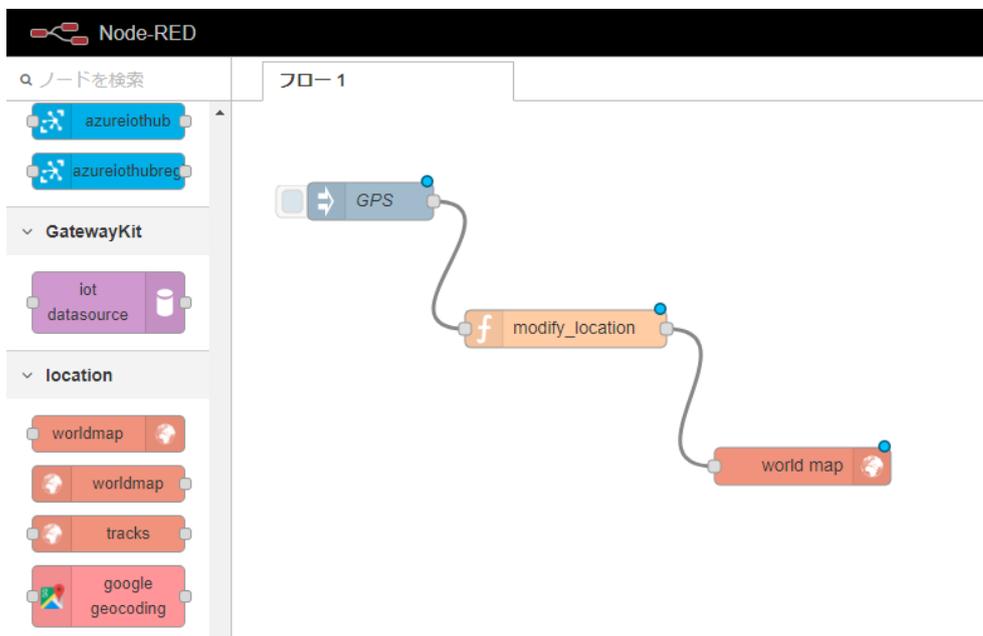
- 配置した function ノードをダブルクリックし、名前欄に”modify_location”を設定します。また、コード部に以下を設定します。設定完了後、『完了』ボタンを押します。

```
//var res = JSON.parse(msg.payload);  
var res = msg.payload;  
  
msg.payload = {  
  name: "Tokyo sta.",  
  lat: res.latitude,  
  lon: res.longitude  
};  
  
return msg;
```

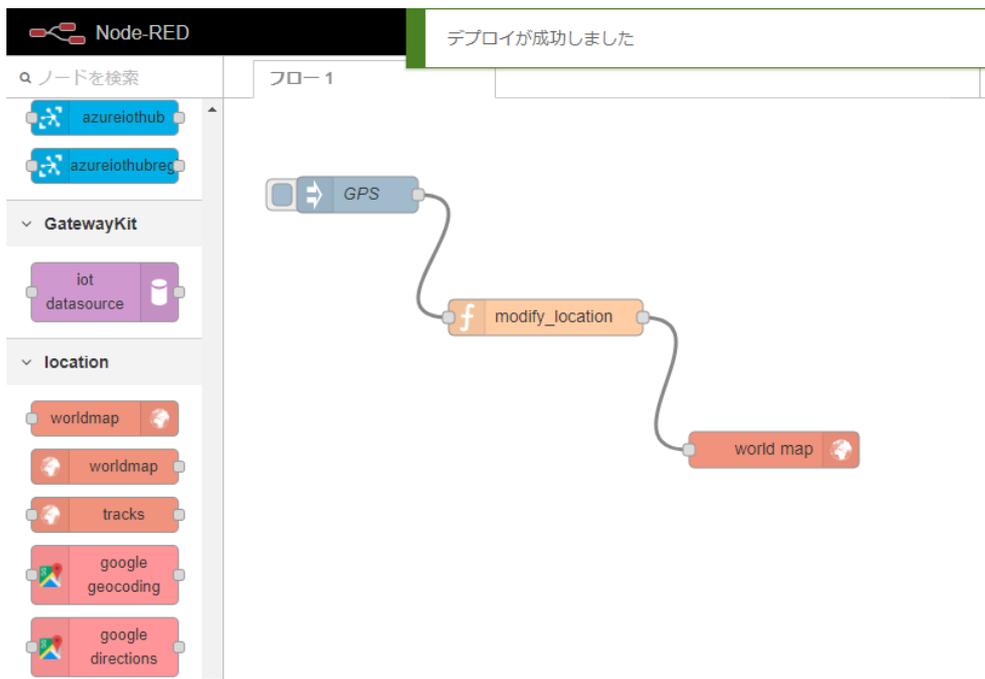
- location ノードから worldmap ノードをドラッグしてシートにドロップします。



- 配置した Inject ノード - function ノード、function ノード - worldmap ノード間を接続します。



- Deploy ボタンをクリックします。

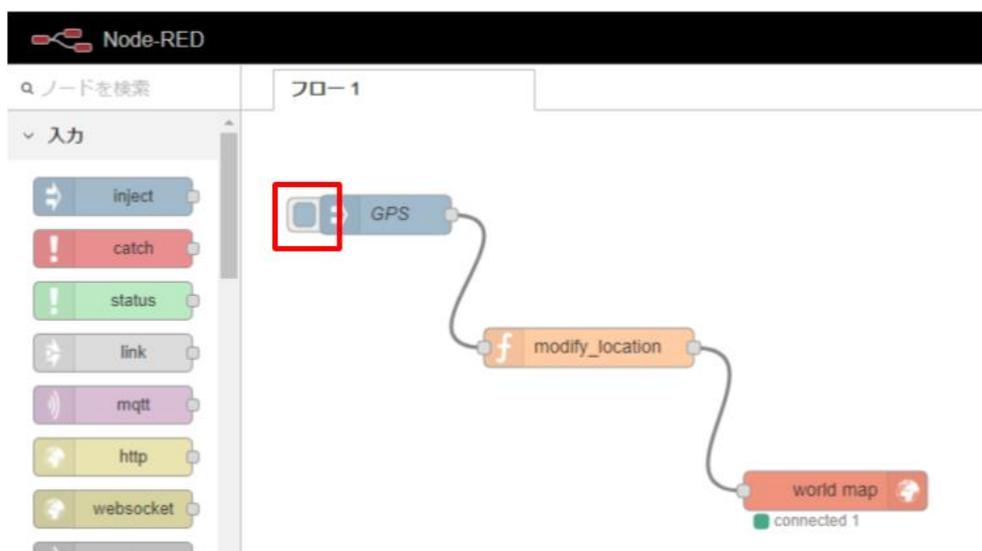


Deploy が完了すると、Deploy ボタンの背景が赤から黒に変わります。

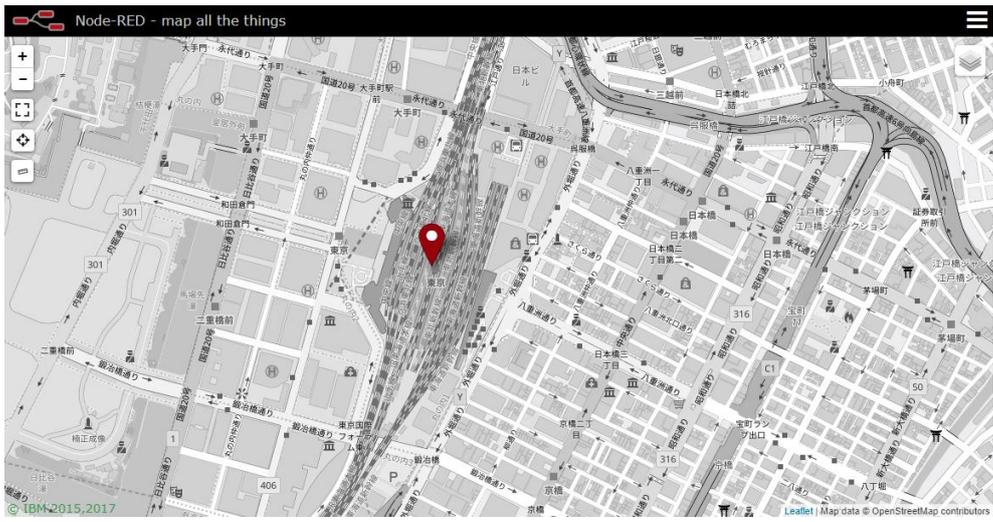
9. ブラウザの別タブにて以下にアクセスを行います。

<http://192.168.254.254:1880/worldmap/>

10. Inject ノードの左部のボタンを押します。



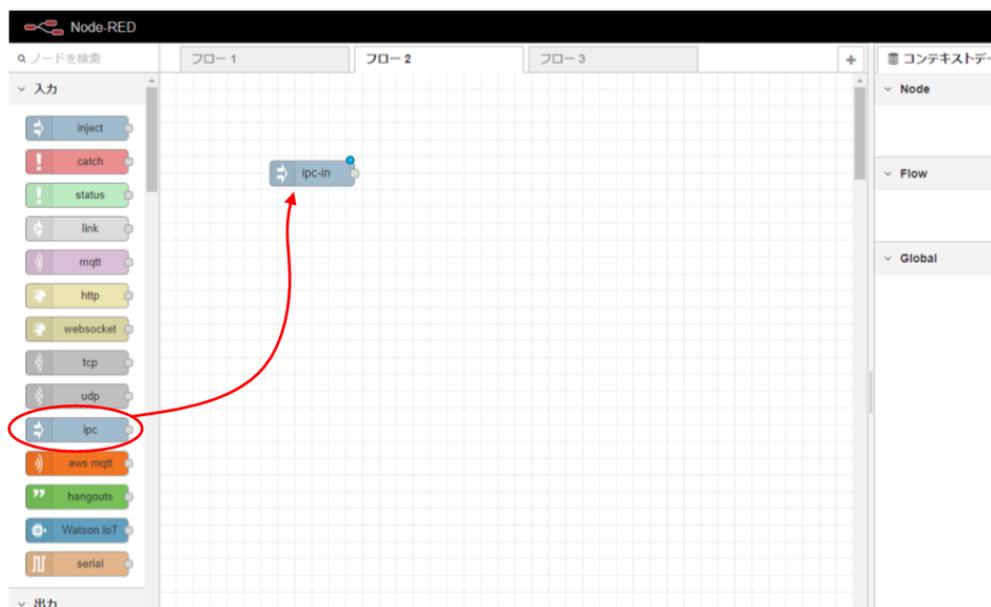
11. 別タブにて表示した地図の日本の東京駅にプロットされます。



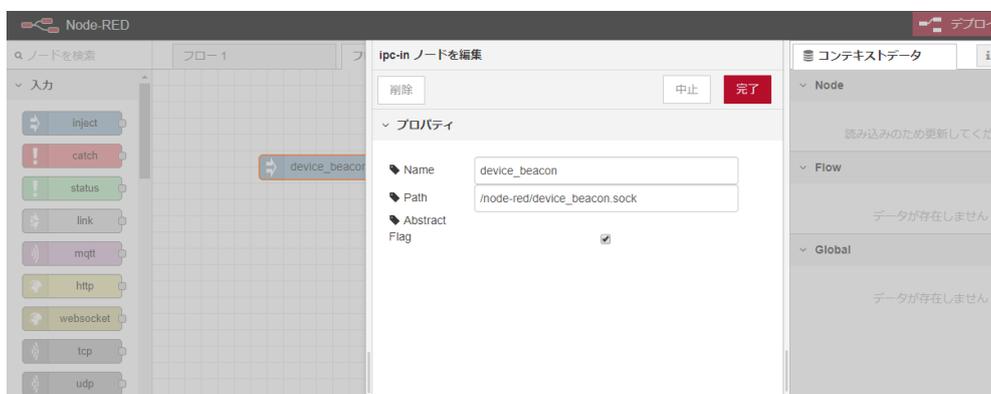
4-2. ビーコンデータへの位置情報付与

本項ではIoTデータ制御機能にて収集したビーコンデータに対して、LTE モジュール(NTTドコモ/KDDI)または BWA モジュールを用いて取得した GPS 情報を付与するサンプルとなります。

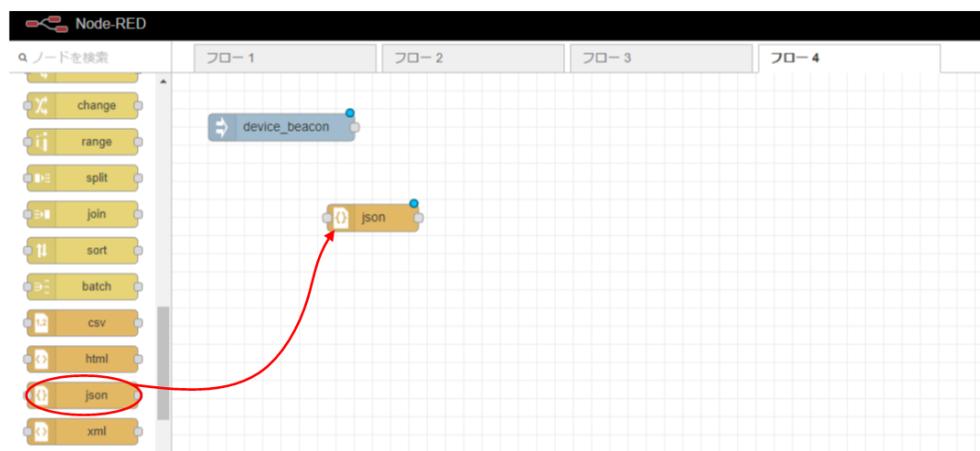
1. 入力ノードパレットから ipc ノードをドラッグしてシートにドロップします。



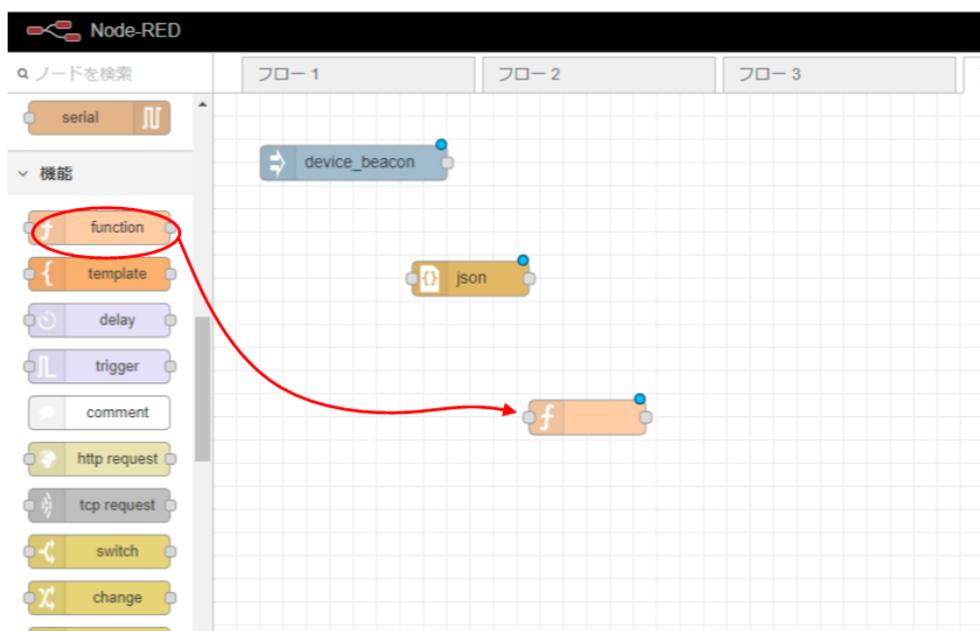
2. 配置した ipc ノードをダブルクリックし、ノードの編集を行います。
Name に”device_beacon”と入力し、Path に”/node-red/device_beacon.sock”と入力します。尚、Abstract Flag をチェックしたままとします。
入力完了後、完了ボタンを押します。



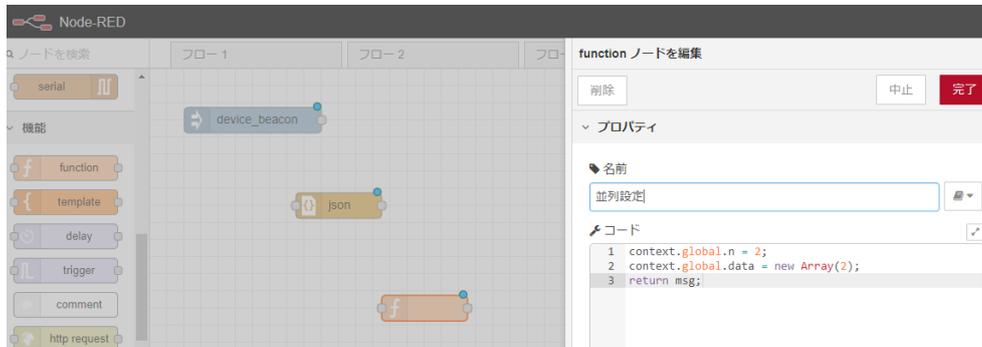
- 機能ノードパレットから `json` ノードをドラッグしてシートにドロップします。



- 機能ノードパレットから `function` ノードをドラッグしてシートにドロップします。



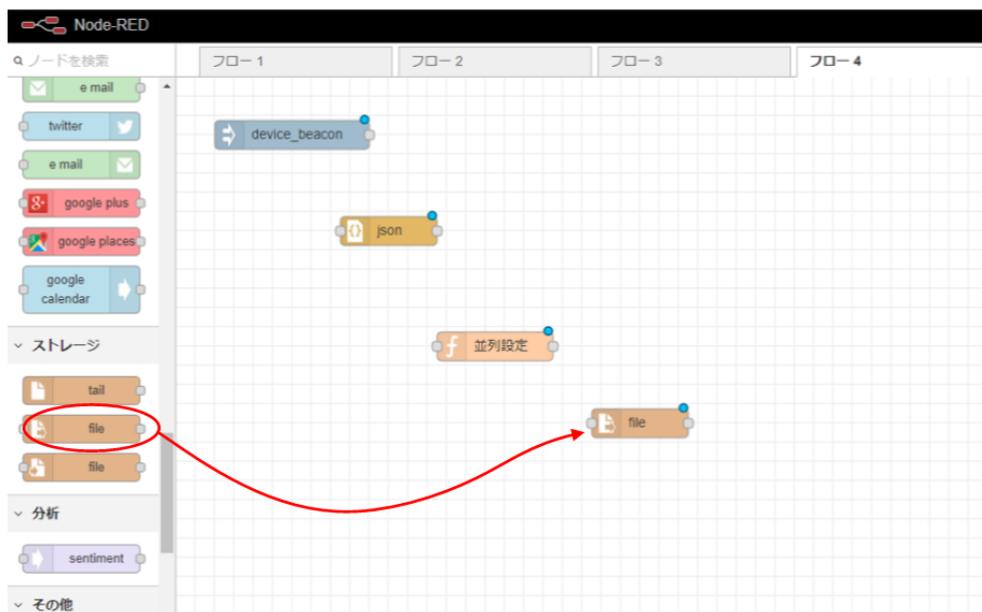
5. 配置した **function** ノードをダブルクリックし、ノードの編集を行います。
名前はビーコンデータと GPS 情報の 2 種類を扱う為、”並列設定”とします。
後述のコードを入力後、完了ボタンを押します。



また、コードは以下を設定します。

```
context.global.n = 2;  
context.global.data = new Array(2);  
return msg;
```

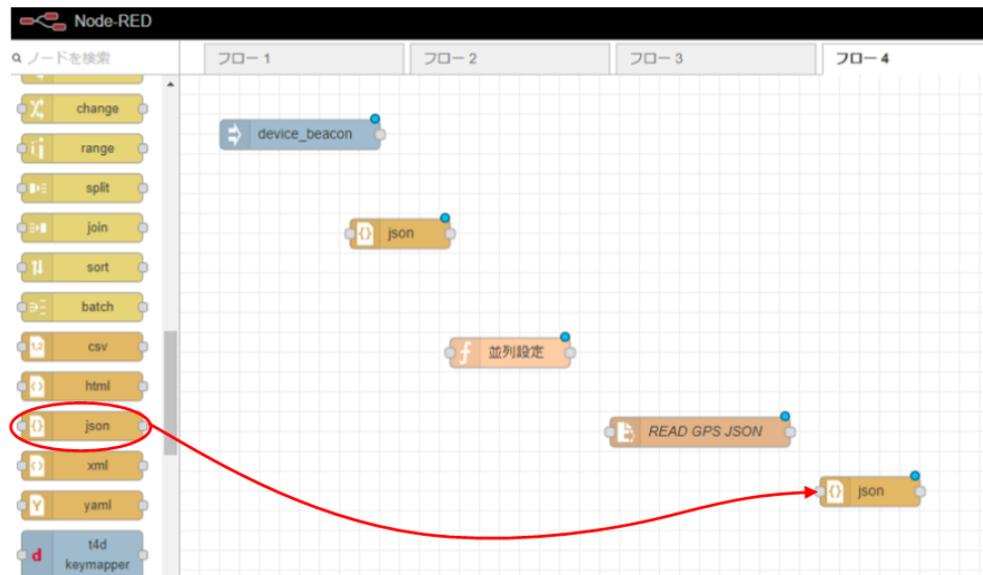
6. ストレージノードパレットから **file** ノードをドラッグしてシートにドロップします。



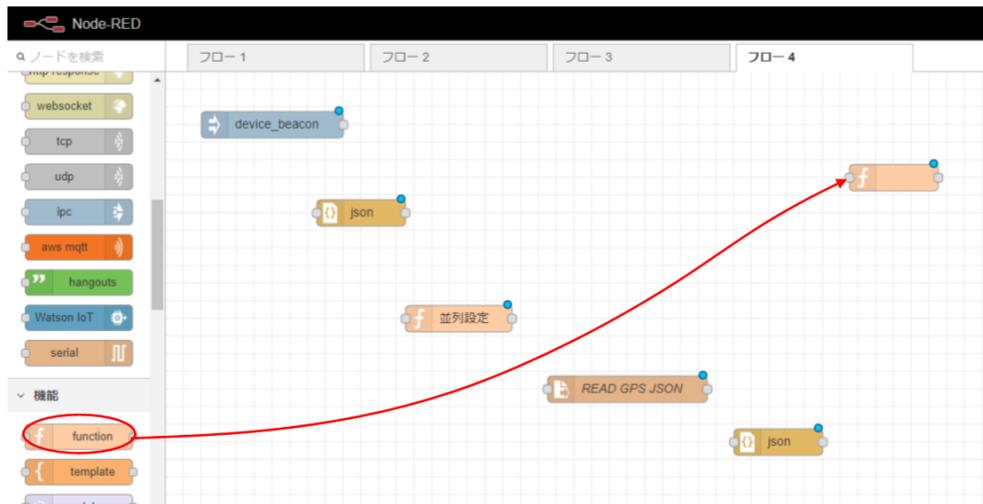
7. 配置した file ノードをダブルクリックし、ノードの編集を行います。
ファイル名は”/tmp/gps_posi.json”とします。また、名前は”READ GPD JSON”とします。尚、出力形式は文字列からの変更はありません。
各項目入力後、完了ボタンを押します。



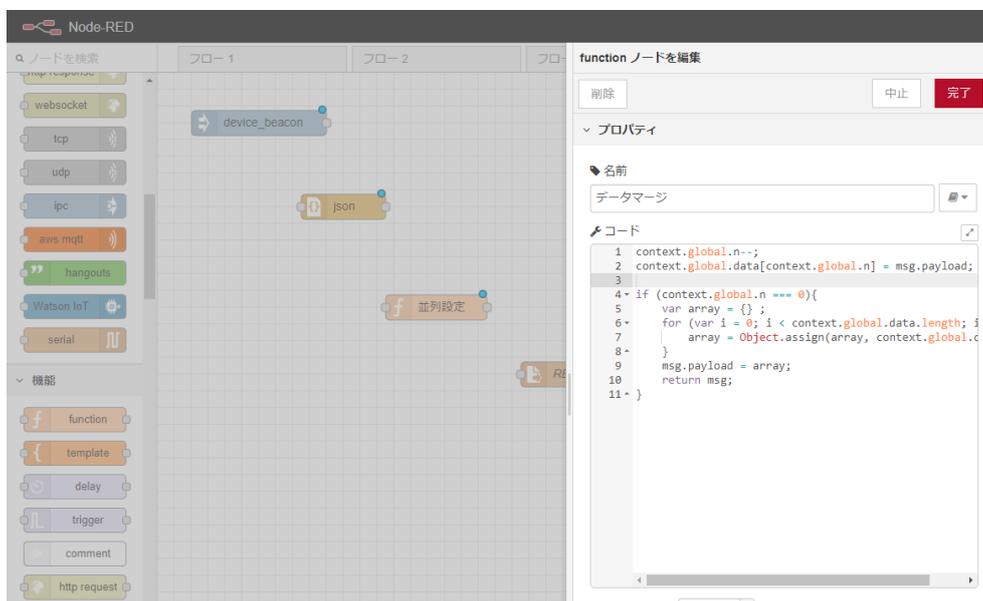
8. 機能ノードパレットから json ノードをドラッグしてシートにドロップします。



9. 機能ノードパレットから **function** ノードをドラッグしてシートにドロップします。



10. 配置した **function** ノードをダブルクリックし、ノードの編集を行います。
名前はビーコンデータと GPS 情報の 2 種類のデータをマージする為、”データマージ”
とします。
後述のコードを入力後、完了ボタンを押します。

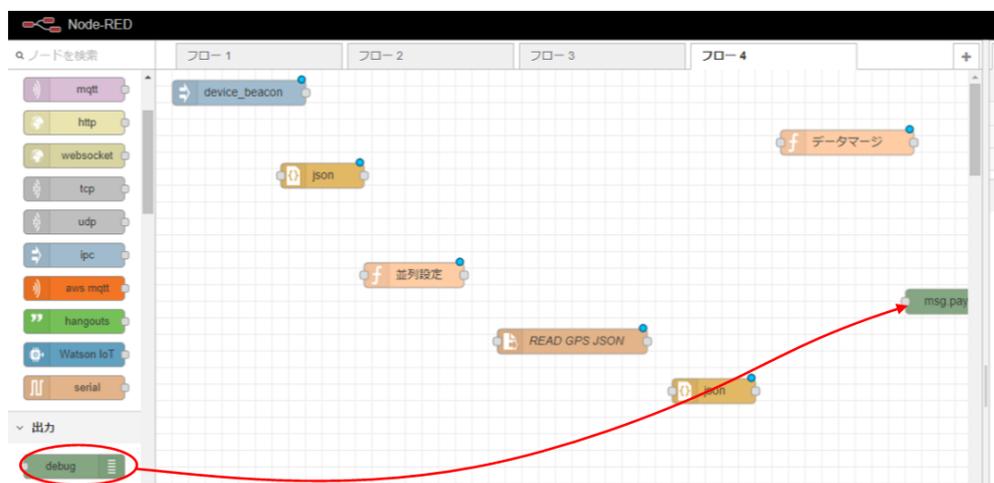


また、コードは以下を設定します。

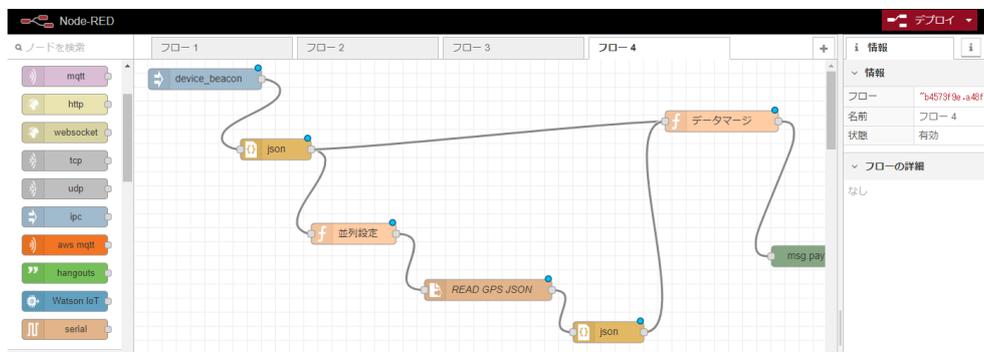
```
context.global.n--;
context.global.data[context.global.n] = msg.payload;

if (context.global.n === 0){
  var array = {};
  for (var i = 0; i < context.global.data.length; i++) {
    array = Object.assign(array, context.global.data[i]);
  }
  msg.payload = array;
  return msg;
}
```

- 出力結果を確認する為、出力ノードパレットから debug ノードをドラッグしてシートにドロップします。



12. 以下の図のように各ノード間を接続します。接続完了後、デプロイボタンを押します。
これにより、ビーコンデータを Node-RED 側で受信した際に GPS 情報ファイルのデータをマージし、debug ノードにマージした情報が出力されます。



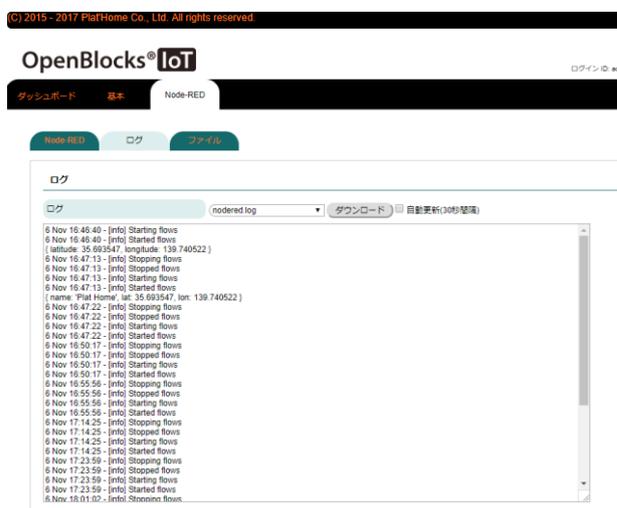
第5章 その他

5-1. Node-RED ログについて

Node-RED 自体のログについては、「Node-RED」→「ログ」タブから確認が行えます。



閲覧したいログファイルをログ欄から選択します。



ログファイルの選択を行うと、対象ログファイルの末尾が表示されます。

また、ダウンロードボタンを押すことによりログファイルのダウンロードが行えます。

5-2. Node-RED へのノード追加

Node-RED に対して対応するファイルをアップロードすることにより、Node-RED に対してノードを追加することができます。

ノード用のファイルアップロードは「Node-RED」→「ファイル」から行えます。



node ファイル

アップロード:

Node-RED のアップロード用に単体のファイルをアップロードが行えます。

アップロードするファイルを選択後、対応するアップロードボタンを押してください。

一括アップロード:

Node-RED のアップロード用に複数のファイルをアップロードが行えます。複数のファイルをアップロードする場合、tar 形式に圧縮したファイルを指定してください。

アップロードするファイルを選択後、対応するアップロードボタンを押してください。

アップロードしたノードファイルを Node-RED に反映させる場合には、Node-RED 自体の再起動が必要となります。後述のダッシュボードからプロセスの再起動を推奨します。

Node-RED のノードの作成方法については以下を参照してください。

<https://nodered.jp/docs/creating-nodes/first-node>

また、本項によるノード追加の場合、ノードの公開は行われられないため package.json の作成等は不要です。

尚、フォーマットエラー等のファイル(html 及び js ファイル)をアップロードした場合、読み込みがスキップされ、対象ノードは表示されません。

5-3. Node-RED の Config 編集

「Node-RED」タブにて使用 Config 設定にてユーザー定義コンフィグを選択している場合、「編集」タブが表示されます。

この「編集」タブから Node-RED 自体のコンフィグ情報を編集できます。



編集

編集対象ファイル：

“setting.js”を選択してください。

ファイルを選択した段階で、現状使用するコンフィグファイルの内容がロードされます。

最終保存時のユーザーコンフィグを読み込みたい場合は、「Load(ユーザーコンフィグ)」ボタンを押してください。

また、システム作成のコンフィグを読み込みたい場合は、「Load(システムコンフィグ)」ボタンを押してください。

※本項目の編集については十二分に理解がある方を前提としています。

コンフィグファイルの編集後、保存ボタンを押すことにより反映されます。

また、編集中のコンフィグファイルの内容をエクスポートボタンを押すことで保存することが可能です。

5-4. Node-RED のプロセス状況について

Node-RED を使用設定にしている場合、ダッシュボードにて Node-RED 自体のプロセス状況を確認することができます。

プロセス状況に応じて、「起動」・「停止」・「初期化」ボタン等が表示されます。

「初期化」ボタンは Node-RED のフロー状況等が削除されますので、ご注意ください。

OpenBlocks® IoT

ダッシュボード

サービス

システム

ネットワーク

メンテナンス

拡張

AirManage

システム全体の概要 [更新](#)

ハードウェアリソース

メインメモリ: 370 MB / 880 MB
ストレージ: 1093 MB / 5337 MB

ネットワーク [設定](#)

FQDN: obsiot.example.org
ゲートウェイ: 172.16.7.1
IPアドレス (eth0): 172.16.7.135
IPアドレス (wlan0): 192.168.254.254

プロセス状況 (Node-RED) [停止](#)

Node-RED: 稼働中 (PID: 11358)

5-5. Node-RED の HTTPS 化

Node-RED のダッシュボードへのアクセスは通常では HTTP 通信となります。

セキュリティ強化のため HTTPS 通信へと変更を行いたい場合には、以下の方法を実施することで切り替え可能となります。

※HTTPS 通信に切り替えた場合、WEB UI の Node-RED タブ内のリンクボタンからのリンクは HTTP 前提となっている為、リンク切れが発生します。

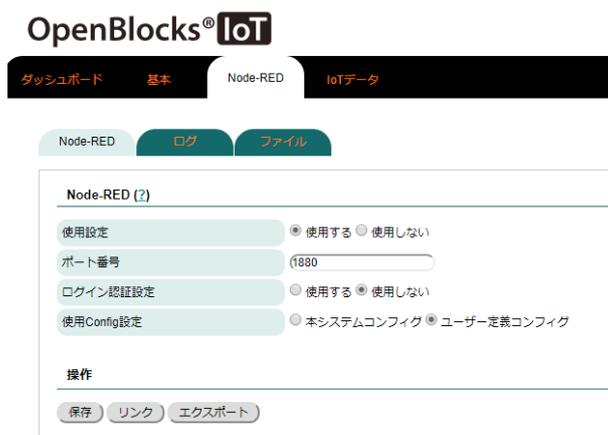
1. 証明書の作成

本項では/var/tmp ディレクトリ配下に自己署名証明書を作成しています。正式なものとして使用するには推奨いたしません。

```
# cd /var/tmp
# openssl genrsa 2048 > privatekey.pem
# openssl req -new -key privatekey.pem 2> /dev/null > server.csr <<!
JP
Tokyo

!
# openssl x509 -days 365 -req -signkey privatekey.pem < server.csr 2> /dev/null >
certificate.pem
```

2. WEB UI の Node-RED タブ内の使用 Config 設定をユーザー定義コンフィグに設定し保存します。



3. Node-RED カテゴリの編集タブにてコンフィグを修正します。

編集対象ファイルを **setting.js** を選択し、テキストボックス内を変更します。

変更内容は”var fs”部のコメント解除及び **https** ディレクティブのコメント解除・パス修正となります。

※変更前

```

. . . . .
// The `https` setting requires the `fs` module. Uncomment the following
// to make it available:
//var fs = require("fs");
. . . . .
// The following property can be used to enable HTTPS
// See http://nodejs.org/api/https.html#https_https_createserver_options_requestlistener
// for details on its contents.
// See the comment at the top of this file on how to load the `fs` module used by
// this setting.
//
//https: {
//  key: fs.readFileSync('privatekey.pem'),
//  cert: fs.readFileSync('certificate.pem')
//},
. . . . .
```

4. ※変更後

```

. . . . .
// The `https` setting requires the `fs` module. Uncomment the following
// to make it available:
var fs = require("fs");
. . . . .
// The following property can be used to enable HTTPS
// See http://nodejs.org/api/https.html#https_https_createserver_options_requestlistener
// for details on its contents.
// See the comment at the top of this file on how to load the `fs` module used by
// this setting.
//
https: {
  key: fs.readFileSync('/var/tmp/privatekey.pem'),
  cert: fs.readFileSync('/var/tmp/certificate.pem')
},
. . . . .
```

5. 変更完了後、保存ボタンを押すことで HTTPS 化は完了となります。

OpenBlocks IoT Family 向け Node-RED スターターガイド
(2018/08/30 第 2 版)

ぷらっとホーム株式会社

〒102-0073 東京都千代田区九段北 4-1-3 日本ビルディング九段別館 3F