

REAL-TIME ALERTING WITH RULES ENGINE

The Rules Engine makes it easy to trigger alerts in the cloud when important events happen in the physical world. This is a very common component of almost any IoT product.

In this tutorial, we'll start by creating a device that measures water depth. Then, using the Rules Engine, when the water level gets too high, we can:

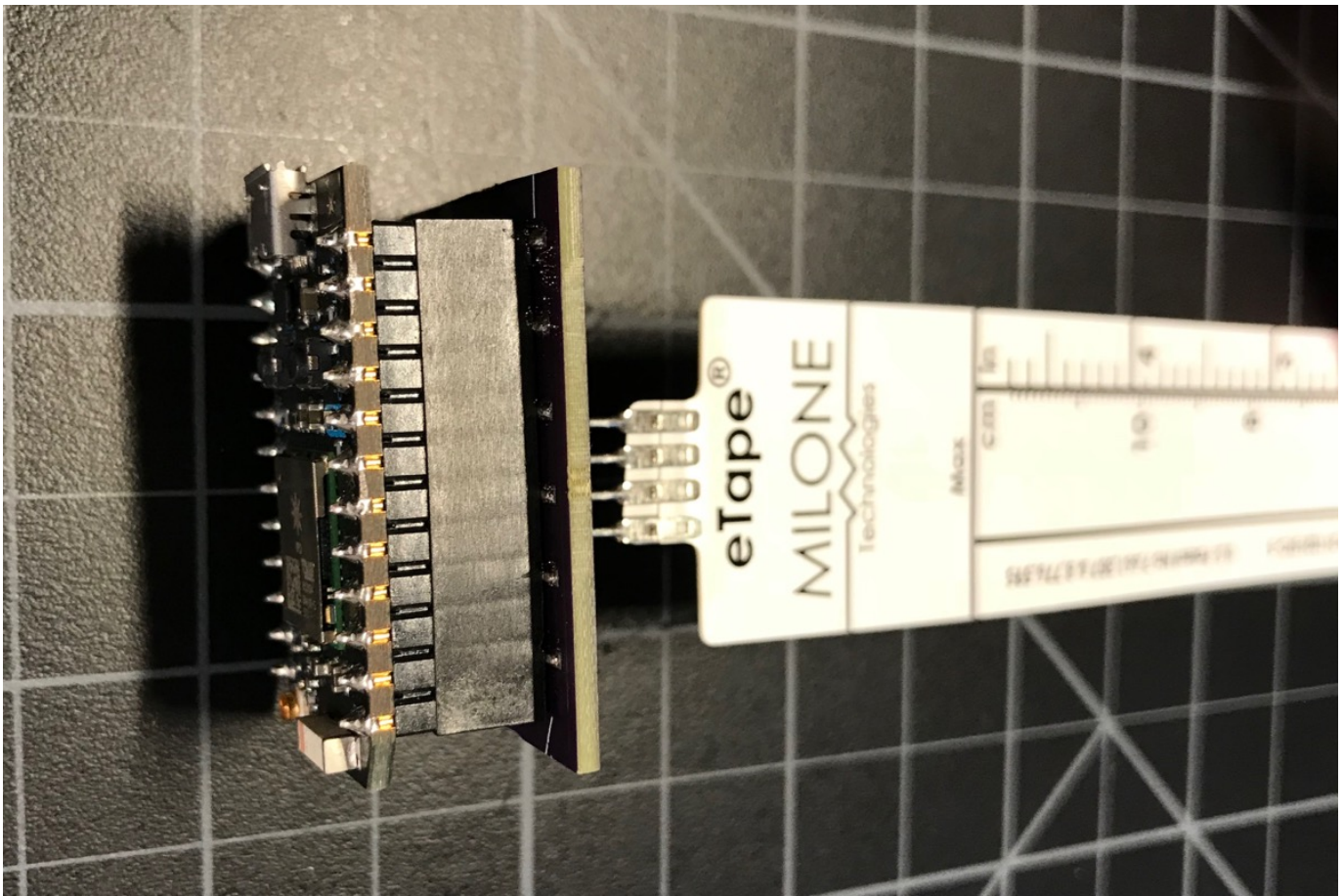
- Alert by SMS using Twilio
- Alert by email
- Alert by Slack

Using the Rules Engine makes it easy to customize the message you send and the recipients, and switch between a variety of notification methods. You can even combine them.

Tutorial Hardware

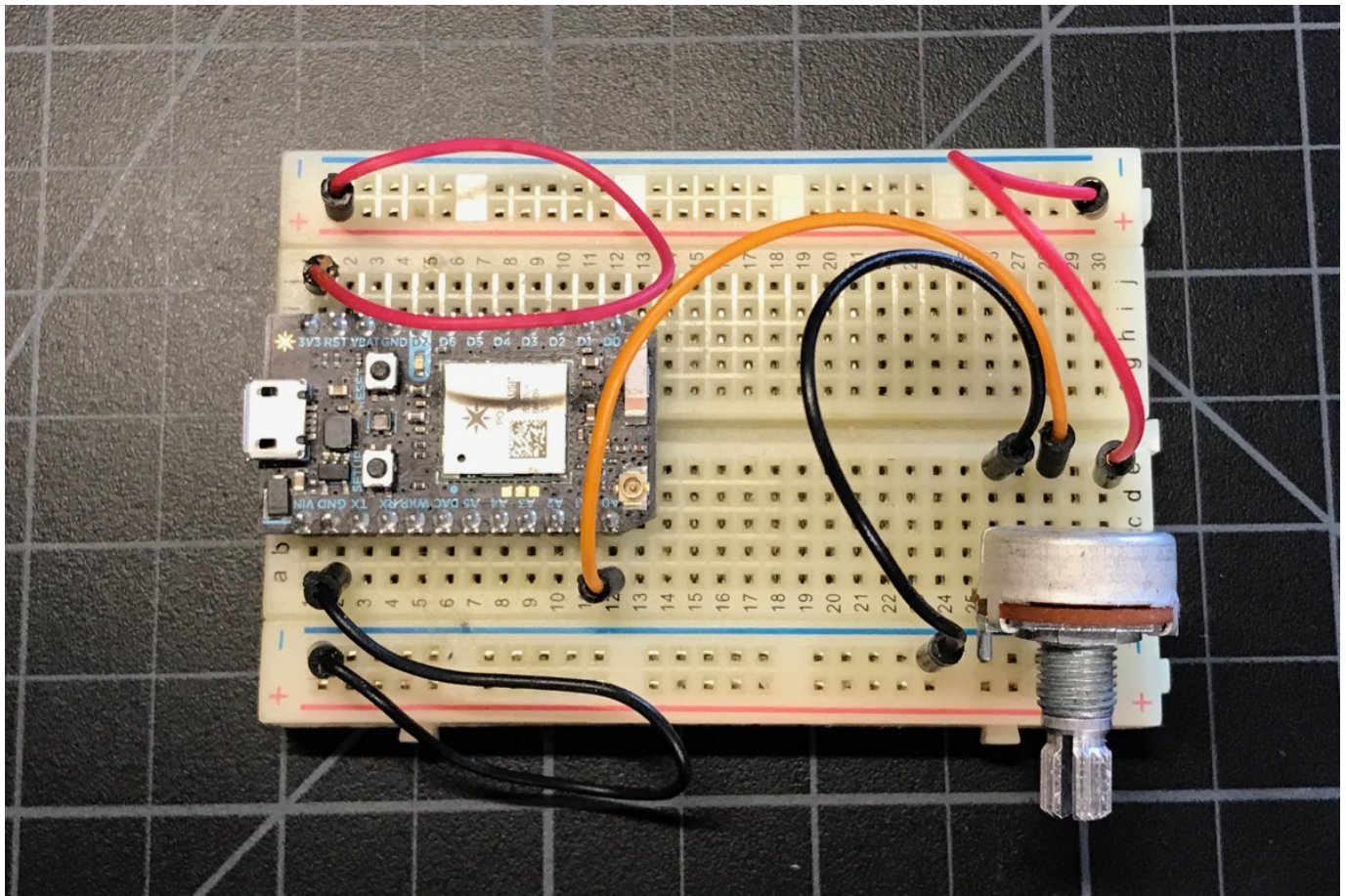
For the hardware side of this project we're using a [eTape Liquid Level Sensor](#) and a Particle Photon. The device firmware reads this sensor continuously and:

- Publishes an alert notification immediately if the level exceeds 2"
- Every minute, checks to see if the level changed, and if so, publishes the new level.



However, for ease of testing with ordinary parts you probably have on hand, you can simulate this using a potentiometer.

Connect the outer legs to 3V3 and GND, and the center tap to A0.

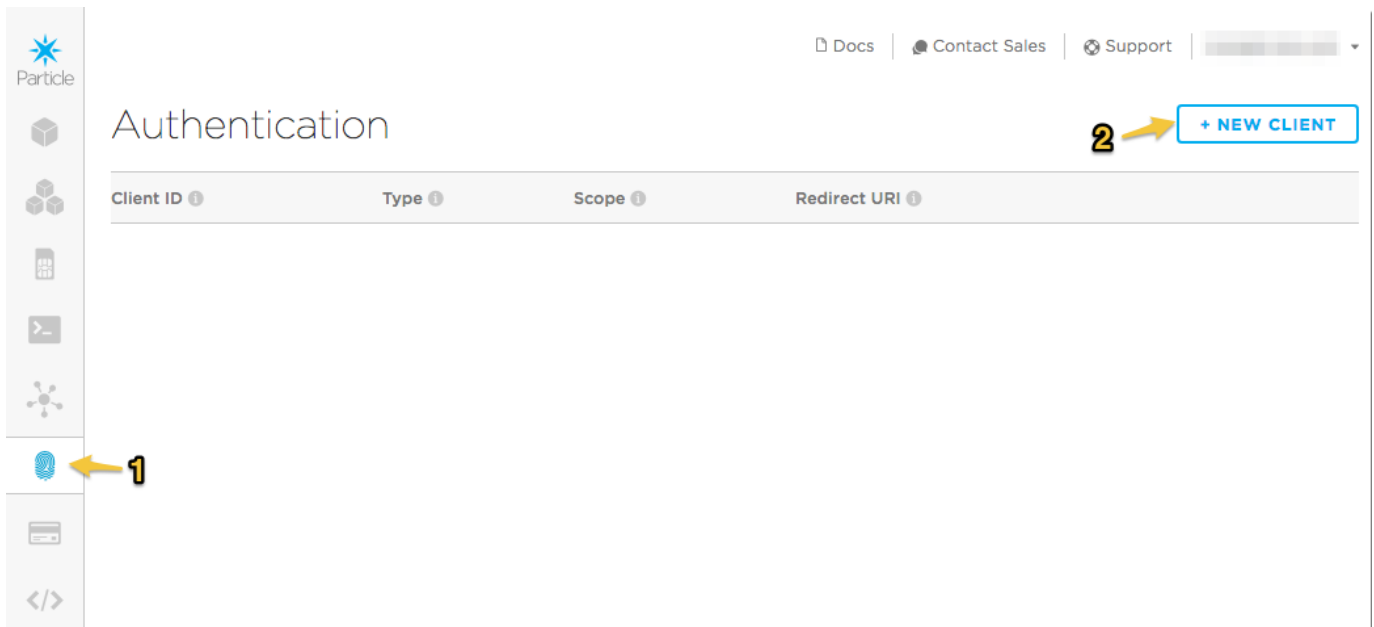


The device firmware is included at the end of this tutorial.

Setting up Authentication

The Rules Engine will need access to your Particle account in order to interact with your devices.

- Log into [the console](#).
- Select **Authentication** (1)
- Click **New Client** (2)



- In the **New OAuth Client** window, select **Two-Legged Auth (Server)**
- Enter a name. I called mine **Rules Engine**.

New OAuth Client

Your Particle project will likely involve interaction with the Particle cloud via a web or mobile application. You can use an OAuth *Client ID* and *Client Secret* to securely communicate with the Particle cloud from an application or a server ([Learn more about clients](#)).

Particle has created sensible defaults for client configuration based on what kind of application you will be building. [Visit the Product Creator Guide](#) to learn more about which client type is right for you.

CLIENT TYPE

☒ Two-Legged Auth (Server) ☐ Simple Auth (Web App)

☐ Simple Auth (Mobile App) ☐ Custom

Name ?

RulesEngine

GET CLIENT ID AND SECRET

- Copy the Client ID (rulesengine-2316 in my example)
- Copy the Client Secret. Note that this should be kept secret, and this is the only chance you have to copy it. Once you close this window you can't get the secret back!

✓ Client Created

You're all set! The client credentials below can now be used to authenticate your app with the Particle cloud. For instructions on adding your credentials to your app, [Visit the Product Creator Guide](#).

For security purposes, this is the only time your client secret will be shown. Please copy it for your records, and store it in a safe place.

CLIENT ID

rulesengine-2316

CLIENT SECRET

I'VE COPIED MY SECRET

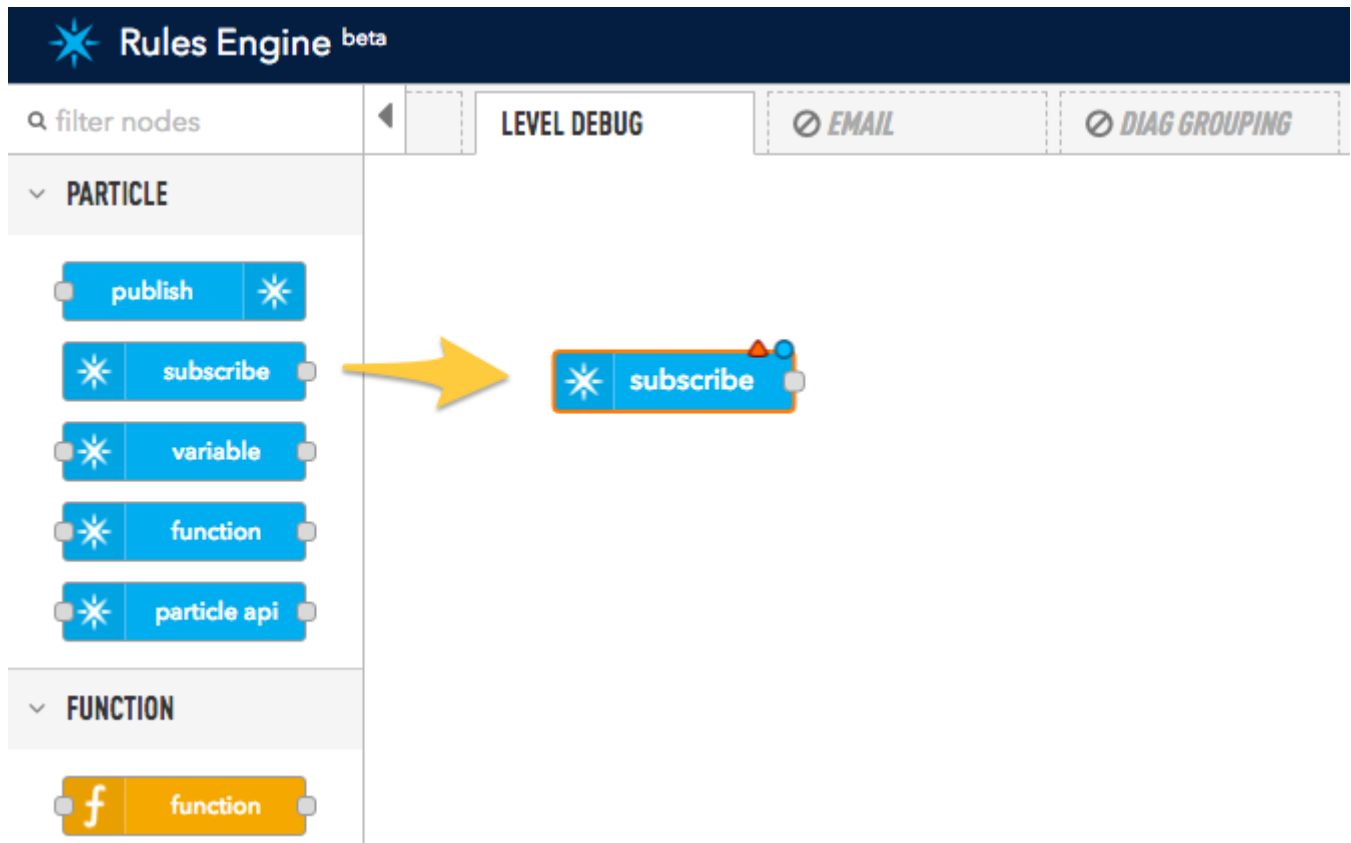
Setting up the flow

We'll be using a subscribe node. This allows the Rules Engine to listen for events posted by devices in your account. The firmware above publishes events periodically with the level (LevelValue), when an alarm occurs (LevelAlarm) and when it stops (LevelClear).

This is the flow we'll be creating in this section:



- In the Rules Engine, from the **Particle** section of the palette, drag a **subscribe** node to the flow. You'll notice it has an red triangle, so it needs to be configured.





- Click the pencil icon to **Add Particle config**.

Edit subscribe node

DELETE CANCEL DONE

▼ **NODE PROPERTIES**

Name

Auth  

Event

Device

Scope ☒ User ☐ Product

- Enter your Client ID and Client Secret from the console into the Particle config window.

Edit subscribe node > Add new particle-config config node

CANCEL ADD

1. Create a Particle OAuth client

Follow the instructions from the [authentication guide](#).

Make sure that you choose "Two-Legged Auth (Server)" as the client type to ensure sufficient permissions.

2. Copy the OAuth client credentials here:

Client ID

Client Secret

- Fill in the rest of the subscribe node configuration.

- Set **Name** and **Event** to "LevelAlarm".
- Leave the **Device** field blank.
- The **Scope** should be left the default of **User**.

Edit subscribe node

DELETE CANCEL DONE

▼ **NODE PROPERTIES**

Name LevelAlarm

Auth rulesengine-2316

Event LevelAlarm

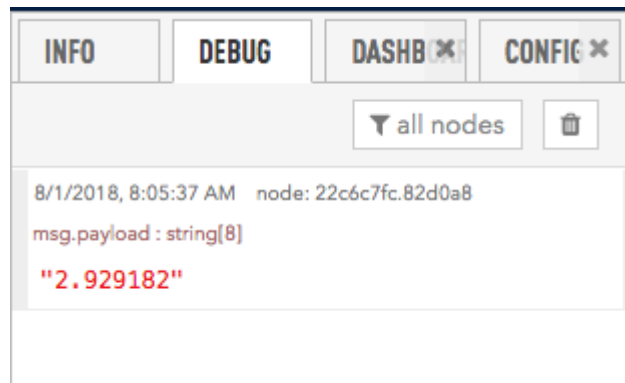
Device Device name or ID

Scope ☒ User ☐ Product

- From the **Output** section of the palette, drag a **debug** node next to your **subscribe** node.
- Then click on one of the handles and drag to the other to connect them.

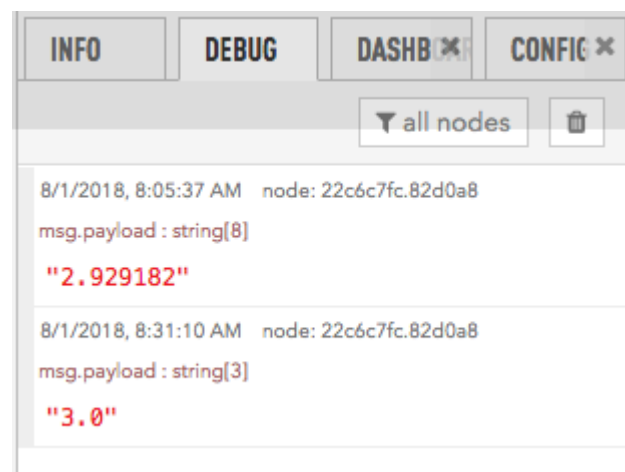


- Click the Deploy button to start your flows running.
- View the **Debug** tab on the right hand side of the Rules Engine.
- Cause an alarm condition in the Photon sensor.



- If you didn't set up the circuit, you can simulate it using the Particle CLI:

```
particle publish "LevelAlarm" "3.0" --private
```



The debug log isn't very interesting or all that useful, so let's send an SMS.

Configure Twilio

This part of the example uses Twilio. There are some more examples below if you want to use other services.

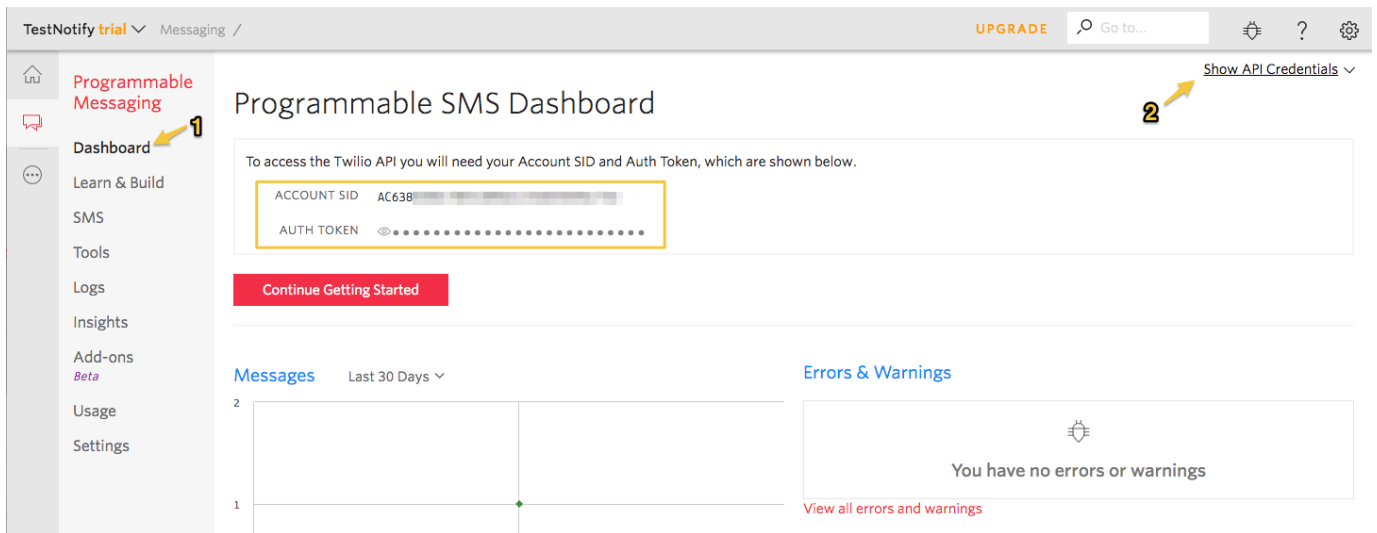
You will need:

1. An active **Twilio account**
2. A project with **Programmable SMS enabled**

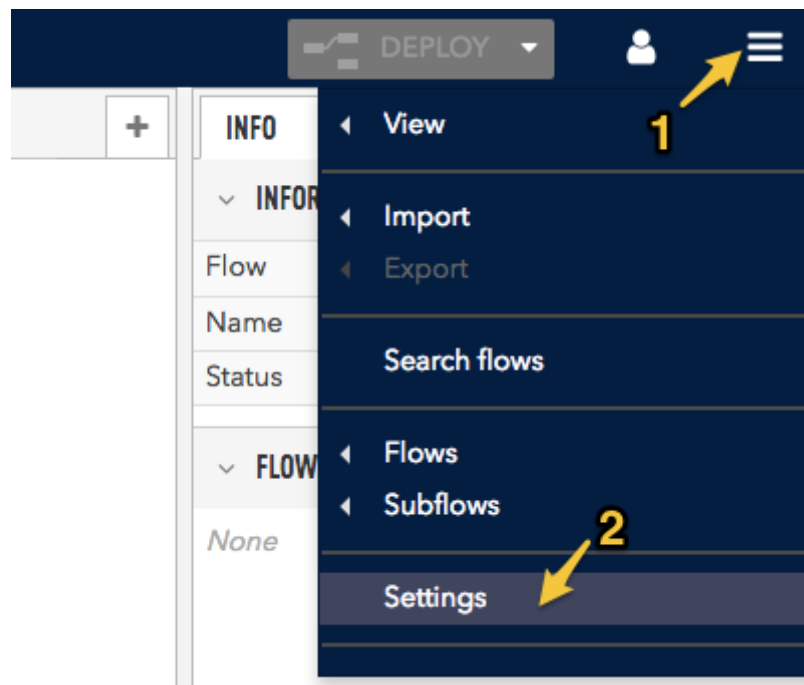
3. A Twilio **phone number** to send SMS from
4. Your account's **SID** and **Auth token**

This can all be configured quite easily using the [Twilio Console](#).

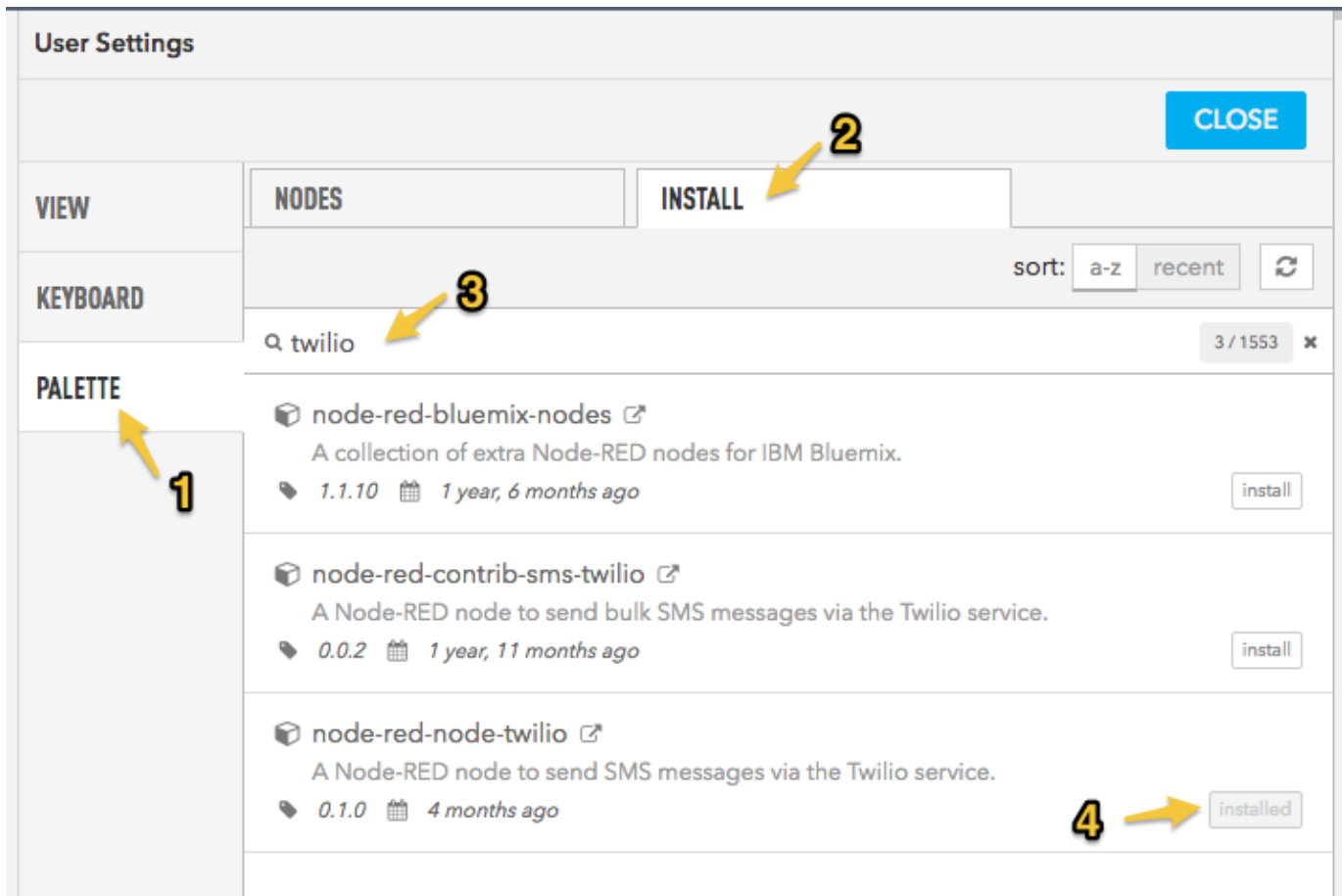
- From the Twilio console, select **Programmable Messaging** and **Dashboard** (1). Then click **Show API Credentials** (2) in the upper right. This is where you can get your Account SID and Auth Token. You'll need these later.



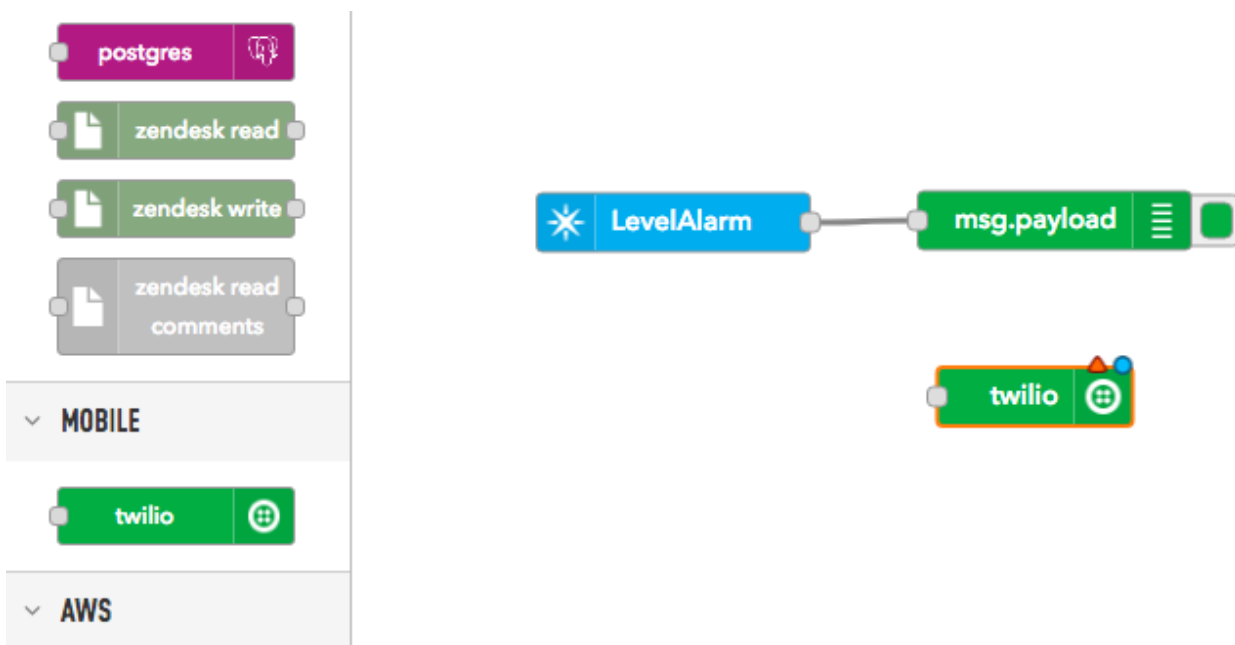
- Back in the Rules Engine, click the "hamburger icon" in the upper right of the Rules Engine window (1) then **Settings** (2).



- Click **Palette** (1).
- Then **Install** (2).
- Type **twilio** in the search box (3).
- Install the item **node-red-node-twilio** (4).



- There will be a new section **mobile** in the palette with **twilio** in it.
- Drag the **twilio** node to your flow.



- Double click to configure your Twilio node.
- Click the pencil icon to create a new configuration.
- Enter your Account SID, Twilio Phone Number, and Auth Token.
- The name is just for display purposes and you can set it to anything.
- Click Add.

Edit twilio out node > Add new twilio-api config node

CANCEL ADD

Account SID AC638!

From +1802

Token

Name TestNotify

- Then configure the **twilio out node**.
- Make sure **Output** is **SMS**
- Enter the phone number in the **To** field. Note that for US phone numbers, its "+1" then the phone number with area code. For other countries, the "1" would be replaced by the country code.

Edit twilio out node

DELETECANCELDONE

▼ NODE PROPERTIES

Twilio

TestNotify

Output

SMS

To

+1802

Name

Notify Rick

- Now drag a connection from the **LevelAlarm** to the twilio node **Notify Rick**.

```
graph LR; LevelAlarm[LevelAlarm] --> msg_payload[msg.payload]; msg_payload --> NotifyRick[Notify Rick]
```

The diagram illustrates a flow in a visual programming environment. It starts with a blue node labeled 'LevelAlarm' with a star icon. A line connects its output to a green node labeled 'msg.payload' with a list icon. Another line connects the output of 'msg.payload' to a green node labeled 'Notify Rick' with a plus icon. The 'Notify Rick' node has a small blue dot above it, indicating it is active or has data.

- Deploy your flow.
- Trigger an alarm condition and you should receive an SMS!

<https://docs.particle.io/tutorials/iot-rules-engine/real-time-alerting/>

14/35



Making the output more readable

There are two problems we want to fix first.

1. Limit the number of SMS messages to at most one every 5 minutes.
2. Make the output a bit more readable than just the number of inches.

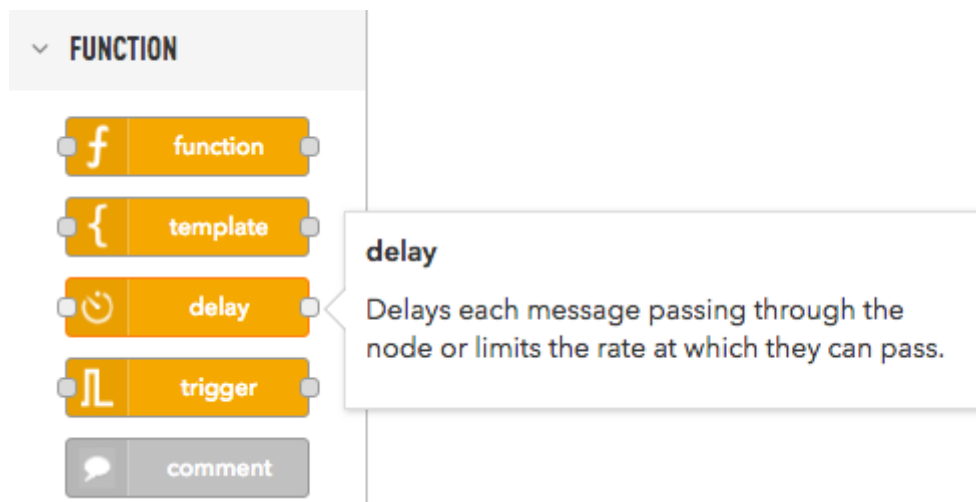
Drag the Copy Rules button into the Rules Engine window to create the flow automatically, or you can create the flow from scratch with the steps below.

Copy Rules ►

This is the flow we'll be building:



- In the **function** portion of the palette, select **delay**.



- Drag it to your flow and double click to configure it.
- Click on **Action** and change it from **Delay** to **Rate Limit**
- Select **All messages**
- Select **1 msg(s) per 5 Minutes**
- Select **drop intermediate messages**
- Set the name to **Rate limit** (or something else of your choosing).

Edit delay node

DELETECANCELDONE

▼ NODE PROPERTIES

Action

Rate Limit

All messages

⌚ Rate

1

msg(s) per

5

Minutes

☒ drop intermediate messages

🔖 Name

Rate Limit

- Find **function** in the palette in the **Function** section. Note that this is not the function in the Particle section.
- Drag it into your flow.

```
graph LR; LevelAlarm[LevelAlarm] --> msg_payload[msg.payload]; msg_payload --> RateLimit[Rate Limit]; RateLimit --> function[f]; function --> NotifyRick[Notify Rick]
```

- Double click the **function** node to configure it.
- Set the name. I made mine **Make Readable Message**.
- In the function box, set the function to:

```
msg.payload = "Level alert! Level is " + msg.payload;  
return msg;
```

<https://docs.particle.io/tutorials/iot-rules-engine/real-time-alerting/>

17/35

Edit function node

DELETE

CANCEL

DONE


▼

NODE PROPERTIES

🔑

Name

Make Readable Message

 ▼

🔑

Function

```
1 msg.payload = "Level alert! Level is " + msg.payload;
2 return msg;
```

- Connect your nodes together by dragging between the handles.



- Deploy your flow.
- Trigger an alert level
- And you should receive a much more readable SMS message!



Sending Email

Drag the Copy Rules button into the Rules Engine window to create the flow automatically, or you can create the flow from scratch with the steps below.

Copy Rules ►

This is the flow we'll be building:



In the **social** group of the palette is the **email** (out) node that you can use for email notifications.

- Drag the **email** (out) icon to your flow.
- Double click to configure it.
- **To** is the email address you're sending to
- **Server** is the SMTP email server to use. The default **smtp.gmail.com** is appropriate for gmail.
- **Port** 465 and **Use secure connection** are appropriate for gmail.
- **Userid** is your username (just the username, not the @gmail.com part)
- **Password** may be your password, but if you have Google two-factor authentication enabled, it's [an app-specific password](#) instead.

Edit email node

DELETE CANCEL DONE

▼ **NODE PROPERTIES**

✉ To rick...

🌐 Server smtp.gmail.com

🔌 Port 465 ☒ Use secure connection.

👤 Userid ...

🔒 Password

🏷 Name Mail Rick

- Drag the handles to connect the email node to your flow.
- Deploy your flow
- Trigger an alert.
- And you should receive an email!

☆ [redacted]@gmail.com
Message from Node-RED
To: rickl [redacted]

Level alert! Level is 2.677656


Posting to Slack

It's easy to post your alert in Slack using [a slack incoming webhook](#).

Configure Slack

- At the link above, click the green button: **Create your slack app**.

Create a Slack App




Interested in the next generation of apps?
We're improving app development and distribution. Join the API Preview period for workspace tokens and the Permissions API.

App Name

Don't worry; you'll be able to change this later.

Development Slack Workspace

 Particle

Your app belongs to this workspace—leaving this workspace will remove your ability to manage this app. Unfortunately, this can't be changed later.

By creating a Web API Application, you agree to the [Slack API Terms of Service](#).

CancelCreate App

- Click **Incoming webhooks**.

The screenshot shows the Particle Rules Engine interface. On the left is a sidebar with a dropdown menu set to 'RulesEngineTest'. Below this are sections for 'Settings' (with 'Basic Information' selected), 'Features', and 'Slack'. The 'Basic Information' section is titled 'Building Apps for Slack' and contains a description: 'Create an app that's just for your workspace (or build one that can be used by any workspace) by following the steps below.' Below this is a section 'Add features and functionality' with a dropdown arrow. It lists six features: 'Incoming Webhooks' (highlighted with a yellow arrow), 'Interactive Components', 'Slash Commands', 'Event Subscriptions', 'Bots', and 'Permissions'. Each feature has a brief description of its functionality.

RulesEngineTest ▼

Settings

- Basic Information
- Collaborators
- Install App
- Manage Distribution

Features

- Incoming Webhooks
- Interactive Components
- Slash Commands
- OAuth & Permissions
- Event Subscriptions
- Bot Users
- User ID Translation

Slack ♥

- Help
- Contact
- Policies
- Our Blog

Basic Information

Building Apps for Slack

Create an app that's just for your workspace (or build one that can be used by any workspace) by following the steps below.

Add features and functionality ▼

Choose and configure the tools you'll need to create your app (or review all [our documentation](#)).

Incoming Webhooks

Post messages from external sources into Slack.

Interactive Components

Add buttons to your app's messages, and create an interactive experience for users.

Slash Commands

Allow users to perform app actions by typing commands in Slack.

Event Subscriptions

Make it easy for your app to respond to activity in Slack.

Bots

Add a bot to allow users to exchange messages with your app.

Permissions

Configure permissions to allow your app to interact with the Slack API.

- Click the slider (1) to **Activate Incoming Webhooks**.
- The click **Add New Webhook to Workspace** (2).

RulesEngineTest

Settings

Basic Information

Collaborators

Install App

Manage Distribution

Features

Incoming Webhooks

Interactive Components

Slash Commands

OAuth & Permissions

Event Subscriptions

Bot Users

User ID Translation

Slack

Help

Contact

Policies

Our Blog

Incoming Webhooks

1

Activate Incoming Webhooks

On

Incoming webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details. You can include [message attachments](#) to display richly-formatted messages.

Each time your app is installed, a new Webhook URL will be generated.

If you deactivate incoming webhooks, new Webhook URLs will not be generated when your app is installed to your team. If you'd like to remove access to existing Webhook URLs, you will need to [Revoke All OAuth Tokens](#).

Webhook URLs for Your Workspace

To dispatch messages with your webhook URL, send your [message](#) in JSON as the body of an `application/json` POST request.

Add this webhook to your workspace below to activate this curl example.

Sample curl request to post to a channel:

```
curl -X POST -H 'Content-type: application/json' --data '{"text":"Hello, World!"}'  
YOUR_WEBHOOK_URL_HERE
```

Webhook URL	Channel	Added By
No webhooks have been added yet.		
<div>2</div> <div>Add New Webhook to Workspace</div>		

- Confirm your identify and select the channel to post to. I just posted to slackbot for testing, but you would normally select a real channel.



On Particle, RulesEngineTest would like to:

Confirm your identity on Particle

Post to

Slackbot, which is private to you ▼

Cancel

Authorize

- Copy the Slack URL, you'll need it later.

RulesEngineTest

Settings
Basic Information
Collaborators
Install App
Manage Distribution

Features
Incoming Webhooks
Interactive Components
Slash Commands
OAuth & Permissions
Event Subscriptions
Bot Users
User ID Translation

Slack
Help
Contact
Policies
Our Blog

Incoming Webhooks

Activate Incoming Webhooks

On

Incoming webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details. You can include [message attachments](#) to display richly-formatted messages.

Each time your app is installed, a new Webhook URL will be generated.

If you deactivate incoming webhooks, new Webhook URLs will not be generated when your app is installed to your team. If you'd like to remove access to existing Webhook URLs, you will need to [Revoke All OAuth Tokens](#).

Webhook URLs for Your Workspace

To dispatch messages with your webhook URL, send your [message](#) in JSON as the body of an `application/json` POST request.

Add this webhook to your workspace below to activate this curl example.

Sample curl request to post to a channel:

```
curl -X POST -H 'Content-type: application/json' --data '{"text":"Hello, World!"}' https://hooks.slack.com/services/T0.../B.../...
```

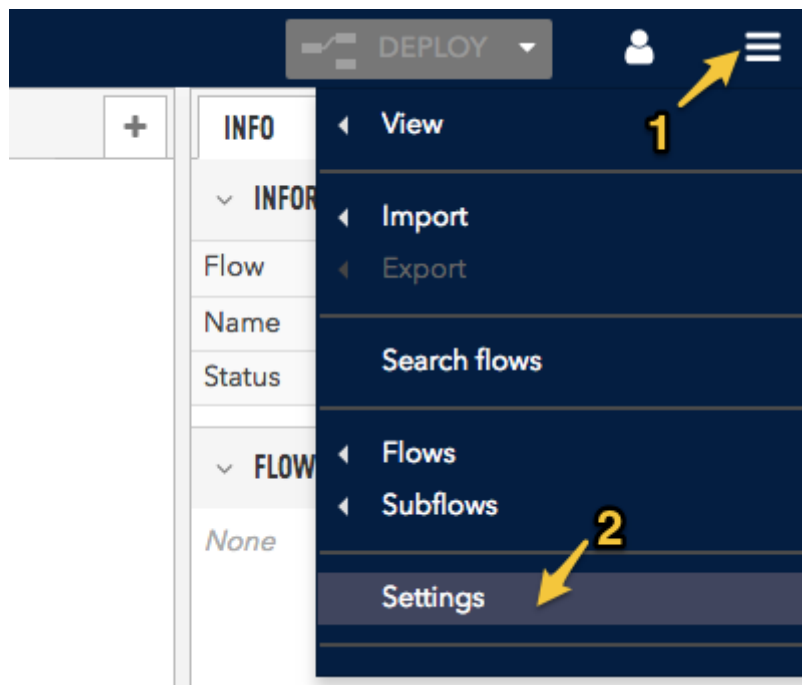
Copy

Webhook URL	Channel	Added By
https://hooks.slack.com/services/T0.../B.../... <div>Copy</div>	slackbot	<div></div> Aug 2, 2018 <div></div>

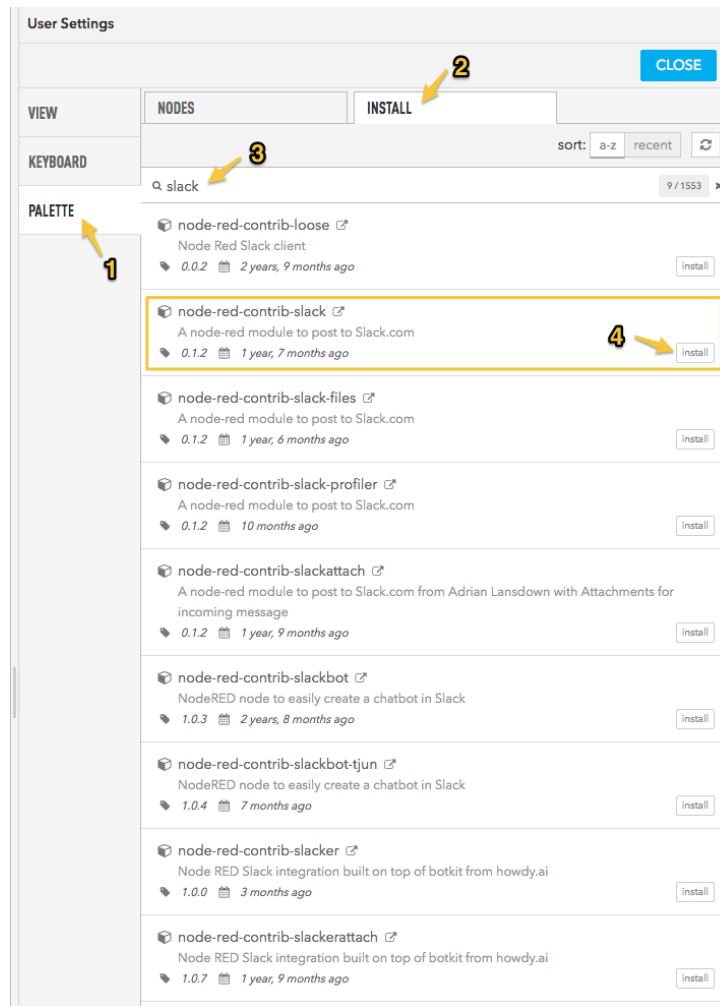
Add New Webhook to Workspace

Add Slack to the Rules Engine

- Back in the Rules Engine, click the "hamburger icon" in the upper right of the Rules Engine window (1) then **Settings** (2).



- Click **Palette** (1).
- Then ****Install** (2).
- Type **slack** in the search box (3).
- Install the item **node-red-contrib-slack** (4).



Building the slack flow

Drag the Copy Rules button into the Rules Engine window to create the flow automatically, or you can create the flow from scratch with the steps below.

[Copy Rules](#) ►

This is the flow we'll be building:




- This flow reuses the **Level Alarm** and **Make Readable Messages** from the previous tutorial. You can either reuse that flow, or copy and paste them into a new flow.
- From the **Social** section of the palette, drag a **slack** (out) node to your flow.
- Double click to configure it.
- Set the **WebHook URL** to the webhook URL you got from Slack.
- Set the other fields as desired.

Edit slack node


DELETE CANCEL DONE

▼ NODE PROPERTIES

⚙ WebHook URL

https://hooks.slack.com/services/T02-

👤 Posting UserName



😊 Emoji Icon

:emojicon:

🔗 Channel

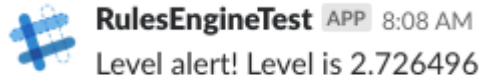
optional channel override

🏷 Name

Post To Slack

- Connect up the nodes in your flow.

- I added a debug node, but that's not required.
- Deploy.
- Cause an alarm condition, and you should see a message in Slack!



Posting to Slack when a device stops reporting

This tutorial expands on the previous tutorial.

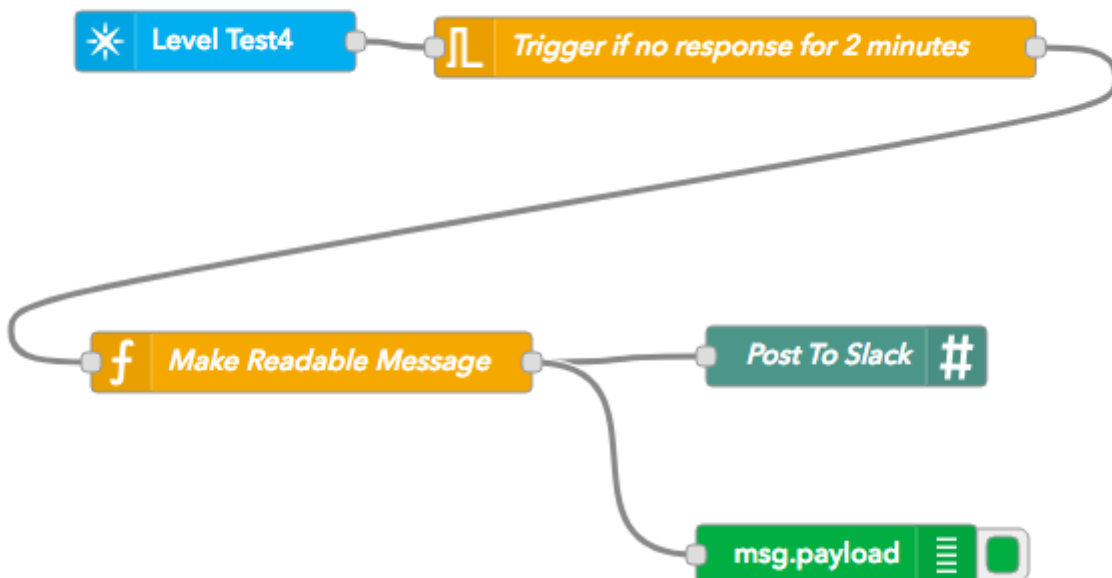
However, the technique for reporting when a device stops responding could easily be changed to email, Twilio SMS, or any number of other notification methods.

Drag the Copy Rules button into the Rules Engine window to create the flow automatically, or you can create the flow from scratch with the steps below.

Copy Rules



This is the flow we'll be creating in this section:



- From the **Particle** section of the palette, drag a **subscribe** node to a new flow.
- Set the **Name** to **Level Test4** (or anything else).
- Set the **Auth** to the Particle authorization you created earlier in the **Real-time Alerting** tutorial.
- Set the **Event** to **Level**. This will trigger on any of the level reporting events.
- Set the **Device** to the name of the device that's reporting. In all of the other examples, we left that field blank, but we're interested in when a specific device stops reporting here.
- Leave the **Scope** as **User**.

Edit subscribe node

DELETE CANCEL DONE

▼ **NODE PROPERTIES**

Name

Auth

Event

Device

Scope ☒ User ☐ Product

- From the **Function** section of the palette, drag **trigger** to your flow.
- Double-click to configure it.
- Set **Send** to **nothing**.
- Set **then** to **wait for 125 seconds**.
- Make sure **extend delay if new message arrive** is checked.
- Set **Handling** to **all messages**
- Set the **Name** to **trigger if no response for 2 minutes** (or anything else).

Edit trigger node

DELETE
CANCEL
DONE

▼ NODE PROPERTIES

Send

▼ nothing

then

wait for

125

Seconds

☒ extend delay if new message arrives

then send

▼ the latest msg object

Reset the trigger if:

• msg.reset is set

• msg.payload equals optional

Handling

all messages

Name

Trigger if no response for 2 minutes

- From the **Function** section of the palette, drag **function** to your flow. Note that this is not the function in the Particle section of the palette.
- Double-click to configure it.
- Set the **Name** to **Make Readable Message** (or anything else).
- Set the **Function** to:

```

msg.payload = 'No response from ' + msg.device_id + ', last level was ' +
msg.payload + ' at ' + msg.published_at;
return msg;

```

Edit function node

DELETE CANCEL DONE

▼ NODE PROPERTIES

Name
Make Readable Message

Function

```
1 msg.payload = 'No response from ' + msg.device_id + ', la:
2 return msg;
```

- Copy and paste the **Post to Slack** node from the previous tutorial.
- Connect your nodes together into a flow.
- I added a debug of the payload for easier debugging.
- Deploy your flow.
- When an event has been received and it's in the two-minute timeout, a blue dot will appear in the bottom left of the **trigger** node.



- If you turn off the publishing device and wait 2 minutes, there should be a message in Slack.

**RulesEngineTest** APP 9:18 AM

No response from 1e[REDACTED], last level was 0.102564 at 2018-08-02T13:16:11.795Z

- You can easily expand this to monitor more than one device by adding more **subscribe** and **trigger** nodes. They can just feed into the existing **Make Readable Message**.

Device firmware

The Photon is programmed with the following code. You can also use [this link](#) to open it in the Particle Web IDE.

```
#include "Particle.h"

SerialLogHandler logHandler;

// This is the pin the sensor is connected to
const int SENSOR_PIN = A0;

// How often to poll the sensor (in milliseconds) to see if it's in alert state
const unsigned long POLL_INTERVAL_MS = 1000;

// How often to publish the sensor (in milliseconds) if the value changes
const unsigned long PUBLISH_INTERVAL_MS = 60000;

// Used to note the last time the value was polled (value from millis())
unsigned long lastPollMs = 0;

// Used to note the last time the value was published (value from millis())
// The initial value means it will publish 3000 milliseconds after startup
unsigned long lastPublishMs = 3000 - PUBLISH_INTERVAL_MS;

// Set to true once we've alerted; flag is cleared when the level drops below ALERT_LEVEL
bool hasAlerted = false;

// The current level (read every second) that's exposed by a Particle.variable
double currentLevel = 0.0;

// The level to alert at
double alertLevel = 2.0;

// Function to read the level in inches
double readLevelInches();
```

```
void setup() {
    Serial.begin();

    // In addition to publishing the level, allow it to be retrieved as a variable
    Particle.variable("level", currentLevel);
}

void loop() {
    if (millis() - lastPollMs >= POLL_INTERVAL_MS) {
        lastPollMs = millis();

        // This block is executed once per second

        currentLevel = readLevelInches();
        if (currentLevel >= alertLevel) {
            if (!hasAlerted) {
                Particle.publish("LevelAlarm", String(currentLevel), PRIVATE);
                Log.info("Level %lf published (alarm)", currentLevel);
                hasAlerted = true;
            }
        }
        else {
            // Once level drops below the alert level, clear the hasAlerted flag

            if (hasAlerted) {
                Particle.publish("LevelClear", String(currentLevel), PRIVATE);
                Log.info("Level %lf published (alarm cleared)", currentLevel);
                hasAlerted = false;
            }
        }
    }

    if (millis() - lastPublishMs >= PUBLISH_INTERVAL_MS) {
        lastPublishMs = millis();

        // This block is executed once per minute

        double level = readLevelInches();

        Particle.publish("LevelValue", String(level), PRIVATE);
        Log.info("Level %lf published (periodic)", level);
    }
}
```

```
    }  
}  
  
double readLevelInches() {  
    //  
  
    double value = (double) analogRead(SENSOR_PIN);  
  
    // Temporary: connect potentiometer outer pins to 3V3 and GND. Center to  
    A0.  
    // Far left = 0V = 0 = 0"  
    // Far right = 3.3V = 4096 = 5.0"  
  
    return (value * 5.0) / 4095.0;  
}
```

The code above uses the testing potentiometer.