inovonics®

Tech note

E*4080 IP Gateway Amazon Web Services MQTT Quick Start

Introduction

This guide will walk through the process of connecting your EE4080 or EN4080 IP gateway to the Amazon Web Services (AWS) Internet of Things (IoT) platform. It will assume that you already have an account with AWS and permissions to access/ change settings in the IoT dashboard.

Contact Information

For questions, contact Inovonics technical support:

- E-mail: support@inovonics.com.
- Phone: (800) 782-2709; (303) 939-9336.

Materials Needed

- An EN4080 or EE4080 IP gateway with a LICMQTT license.
- An AWS account.
- The ability to connect to Amazon servers outside of your firewall.
- Traffic allowed on the following ports:
  - Port 9000 (TCP for OTA upgrades)
  - Port 123 (UDP for NTP sync)
  - Port 8883 (standard MQTT port)
  - Port 443 (HTTPS)
- A USB flash drive (must use MBR partition format, not GPT).
- A computer.
- EchoStream transmitters.

Setup Credentials for the IP Gateway on AWS

This procedure will setup credentials for the gateway on AWS. When completed, you will have the AWS certificate files to put on your IP gateway using the USB flash drive.

1. On your computer, navigate to the AWS IoT dashboard and select “Manage.”

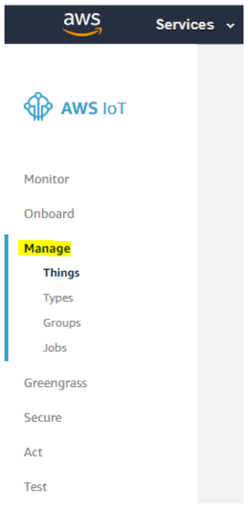


Figure 1 Select “Manage” on the AWS IoT dashboard.

2. Select “Create” in the top right of the AWS IoT dashboard.

3. Select “Create a Single Thing.”

4. Give your thing a name
- No other fields are required.

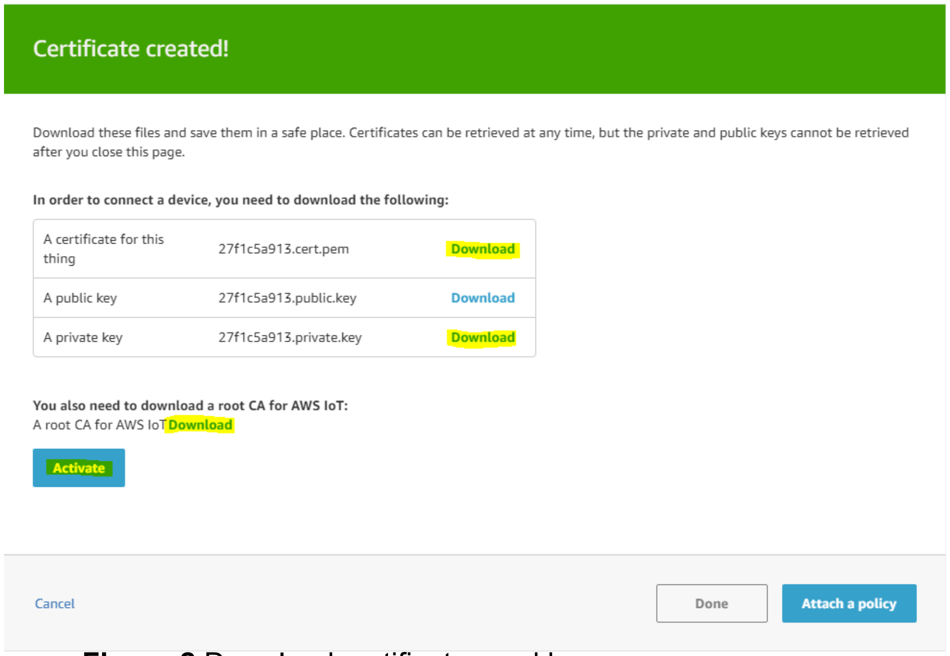
Note: The thing name and client-id must match exactly for monitoring tools within AWS to work correctly.

---

5. Select "Next."

6. Select "One-Click certificate creation."

7. Download all files except for the public key.



Certificate created!

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

In order to connect a device, you need to download the following:

A certificate for this thing	27f1c5a913.cert.pem	Download
A public key	27f1c5a913.public.key	Download
A private key	27f1c5a913.private.key	Download

You also need to download a root CA for AWS IoT:
A root CA for AWS IoT Download

Figure 2 Download certificates and keys.

Note: The "Download" link opens a new tab where you select the files.

8. Click "Activate."

9. Click "Done."

- The thing has been created with AWS.

10. Navigate to the "Secure" tab and select "Policies."

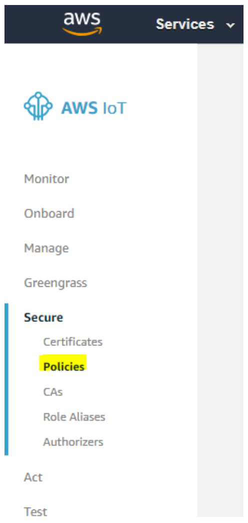


Figure 3 Select "Policies" on the AWS IoT dashboard.

11. Select "Create."

12. Give your policy a name.

13. Set policies according to your standards, or enter the following:

- Action: iot:*
- Resource ARN: *

14. Check “Allow” under “Create.”

15. Click "Create."

16. Navigate to the “Secure” tab and select "Certificates.”

17. Click the three dots on the certificate you generated and select “Attach Policy.”

- The alphanumeric code on your certificate will match the key files you downloaded in step 7.

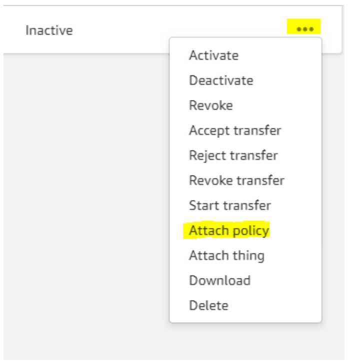


Figure 4 Attach a policy.

18. Select your new policy and click “Attach.”

19. Click the three dots on the certificate you generated and select “Attach thing.”

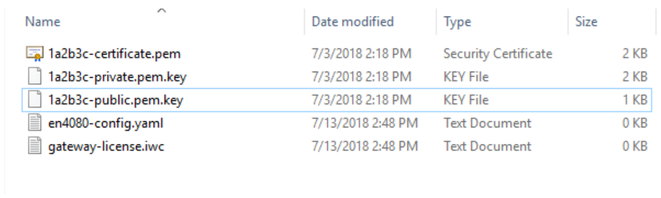
Connect the IP Gateway to AWS

This procedure will establish a connection between your IP gateway and AWS.

1. Referring to the IP gateway’s installation instructions, download the configuration .yaml file and gateway license file onto a blank USB flash drive.

2. Keeping the AWS IoT dashboard open as a reference, copy the three files downloaded earlier onto the USB flash drive.

- All of the files that need to be on the USB flash drive are shown in Figure 5. These include the configuration .yaml file, the IP gateway license file, and three files downloaded in step 7.



Name	Date modified	Type	Size
1a2b3c-certificate.pem	7/3/2018 2:18 PM	Security Certificate	2 KB
1a2b3c-private.pem.key	7/3/2018 2:18 PM	KEY File	2 KB
1a2b3c-public.pem.key	7/3/2018 2:18 PM	KEY File	1 KB
en4080-config.yaml	7/13/2018 2:48 PM	Text Document	0 KB
gateway-license.iwc	7/13/2018 2:48 PM	Text Document	0 KB

Figure 5 The files necessary on your USB flash drive.

3. Open the configuration file on the USB flash drive in any text editor.

- The following is an example of a configuration file, with unrelated lines omitted for clarity. The reporter is specified, and then followed with a section for reporter settings, as in the following configuration file example:

```
radio:
   redundant_message_window: 10.0
   redundant_message_first_hop: exclude
   redundant_message_signal_margin: include
plugin:
   awsmqtt:
      host: acvqxe0o75onu-ats.iot.us-west-
      2.amazonaws.com
      port: 8883
      root_ca: root.pem
      certificate: certificate.pem.crt
      private_key: private.pem.key
      client_id_prefix: inovonics
      client_id_fields: model-number, serial-
      number-unit
      reporting_format: json
      publish_message_topic: messages
      publish_discovery_topic: discovery-messages
      subscribe_conf_topic: settings
      publish_conf_topic: configuration
      connection_status_topic: connection-messages
      batching: false
      qos: 1

network:
   ethernet:
      power: enabled
      address: 192.168.90.50
      netmask: 255.255.254.0
      gateway: 192.168.90.1
      dns: 192.168.1.35
```

```
wifi:
   country: US
   power: disabled
   ssid: Guest
   passphrase: Welcome
   address: dhcp
```

4. Enter configuration information in each field of the configuration file.

Note: Any field marked with an * can be changed later remotely; any field not marked with an * must be edited using the USB flash drive.

- radio: Configure options for redundant message suppression:
  - redundant_message_window: A message with the same payload as a recent (in seconds) message will be considered redundant and not buffered. This is set to 10 seconds by default.
  - redundant_message_first_hop: Consider different first hops are non-redundant messages if include; first hop ignored if exclude (default).
  - redundant_message_signal_margin: Consider a strong signal margin is a non-redundant message if include (default); signal margin ignored if exclude.
  - To receive the fewest (if any) redundant messages, disable the two parameters above with exclude. Enable the former to learn all repeaters that heard a message; enable the latter to learn the strong signal margin heard.
- host: this will be a url, which can be found in the IoT dashboard.
  - Under the "Manage" tab, select "Things", then your new thing, then "Interact."
  - Locate the url ending in ".amazonaws.com" which on the top of the page. This is your host.
- port: This indicates which port your IP gateway communicates on and will be 8883 in almost all cases.
- root_ca: This is the name of the root cert file placed on the USB flash drive.
- certificate: This is the name of the device’s certificate file placed on the USB flash drive.
- private_key: This is the name of the private key placed on the USB flash drive.
- client_id_prefix: This is the string for your client id to start with.
- client_id_fields: This is a list of values which will be appended to the base client_id_prefix with dashes. Possible values: “mac-wired”, “model-number”, “serial-number-board” and “serial-number-unit”. See the example on page 5 and the following table:

Note: Do not type information about your gateway here.

client-id-prefix	client-id-fields	Resulting client ID
inovonics	serial-number-unit	inovonics-VP1234567
my-gateway		my-gateway
inovonics	serial-number-unit, mac-wired	inovonics-VP1234567-F8:DC:7A:13:6D:1A

- *publish_message_topic: This is the topic your device will push its messages to; it will be used to receive messages in AWS.
- subscribe_conf_topic: This is the topic your device will receive messages from, and is used to change settings remotely. If you will not be configuring settings remotely this can be left blank.
- publish_conf_topic: The IP gateway will send an up-to-date copy of your configuration settings after each modification. This can be the same as subscribe_conf_topic.
- *discovery_mode: If true, this will activate discovery mode and send messages on your defined publish_discovery_topic. This feature times out after 30 minutes and must be reactivated. Discovery mode cannot be activated by USB; it must activated over MQTT.
- *publish_discovery_topic: Messages from unregistered devices will transmit on this topic.
- connection_status_topic: The topic that receives messages on a connection status change. All devices can use the same topic for this. Each message also includes a client_id.
- qos: This stands for quality of service and is a part of MQTT. It will be an integer (0 or 1). A setting of 1 is recommended in most cases. This impacts most messages, but discovery mode messages always use a QOS of 0. A QOS of 2 is not currently supported by AWS, and cannot be used here.
- *txids: This is a list of txids to collect messages from.
- batching: If true, this sends messages in groups to save resources. This is recommended, as otherwise each message will be transmitted separately.
- reporting_format: This can be JSON or bytearray. Most of the time it will be JSON.

Note: We suggest making a copy of this file once it is complete and updating the file as changes are made remotely. This will give you a backup of all settings.

5. Enter the network configuration information in the network section.

Note: If the network section is not present, wired Ethernet is configured for DHCP IP address assignment by default, and the WiFi is disabled.

- Configure the wired Ethernet interface in the ethernet section:

  - power: Controls if wired Ethernet is to be enabled or disabled.
  - address: This is the static IP address the gateway will use for its wired Ethernet port. To use DHCP for IP address assignment and network interface configuration, specify a value of dhcp; in this case, the following three settings can be omitted.
  - netmask: The network mask for the attached network.
  - gateway: The gateway of the attached network.
  - dns: The address of the desired DNS server.
- Configure the WiFi interface in the wifi section:
  - country: The two character ISO country code in which the gateway resides; e.g. US for the United States.

Note: See https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2 for a full list of ISO country codes.

  - power: Controls if 2.4GHz WiFi is to be enabled or disabled.
  - ssid: This is the name of the WiFi network to connect to.
  - passphrase: The passphrase of the WPA/2 Wifi network to connect to. Leave empty for an open network.
  - address: The static IP address the gateway will use for its wired Ethernet port. To use DHCP for IP address assignment and network interface configuration, specify a value of dhcp; in this case, the following three settings can be omitted.
  - netmask: The network mask for the attached network.
  - gateway: The gateway of the attached network.
  - dns: The address of the desired DNS server.

6. Referring to the IP gateway’s installation instructions, use the USB flash drive to boot the IP gateway. The IP gateway should now be connected to AWS.

- The orange connection light on the IP gateway should light and remain illuminated after booting completes.

Test Connection

1. Select “Test” on the IoT dashboard.

2. Input the topic specified in your configuration file for “pub_msg_topic” in the “Subscription topic” field.

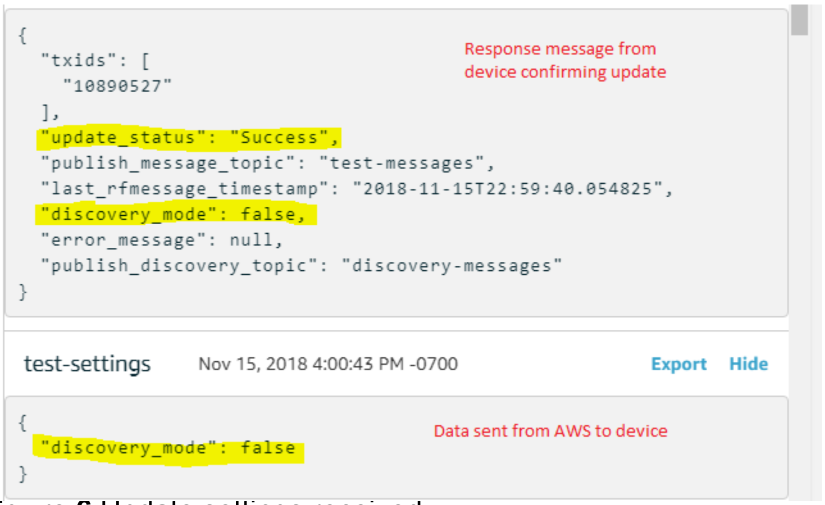
3. Click “Subscribe to topic.” Messages should begin to appear, as long as transmitters are sending messages to the IP gateway.

Change Settings

This section describes how to change settings on an IP gateway that is already in use.

Your device can receive settings via messages over the subscribe_conf_topic specific in the config file. This can be done in a variety of ways, but we'll showcase the simplest method here.

1. Navigate to the "Test" tab of AWS IoT dashboard.
2. Enter the name of your subscribe_conf_topic.
3. Click "Subscribe to topic"
   - This will use your configured QOS, so be sure to match your configured QOS setting with the QOS setting here.
4. Add the amended settings in valid JSON.
5. Click "Publish to topic."



```
{
  "txids": [
    "10890527"
  ],
  "update_status": "Success",
  "publish_message_topic": "test-messages",
  "last_rfmessage_timestamp": "2018-11-15T22:59:40.054825",
  "discovery_mode": false,
  "error_message": null,
  "publish_discovery_topic": "discovery-messages"
}
```

Response message from device confirming update
test-settings
Nov 15, 2018 4:00:43 PM -0700
Export
Hide
Data sent from AWS to device

```
{
  "discovery_mode": false
}
```

Figure 6 Update settings received

The change should take effect almost instantly. Each time a modification is made to settings via MQTT, a message will be sent on your publish_conf_topic that shows the current state of each editable setting, including the most recent change.

update-status can be:

- Success: Configuration was updated successfully.
- Warning: Configuration was updated but with potential problems.
- Failure: Configuration failed to adopt new settings.

Note: Settings changes overwrite existing settings, they do not append. For example, if you're adding a txid, send the complete list of txids, not just the new addition.

Note: Minimize topic changes as they can interrupt connectivity with the cloud and create a large queue on the IP gateway.

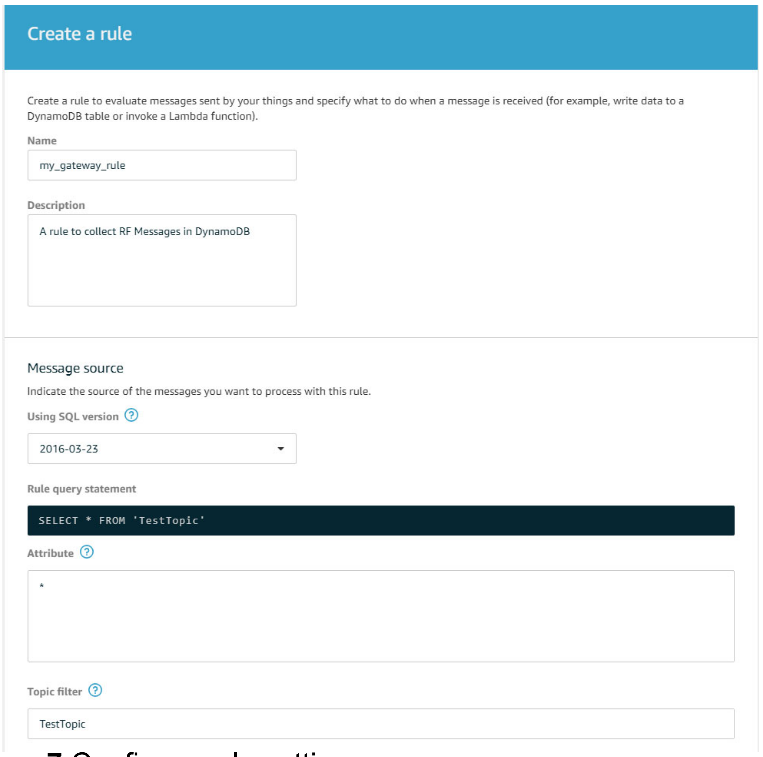
Note: Sending only “{}” to the device will return the current settings without any modification.

Store Message Data

This section will show a basic use case to store message data in DynamoDB. This is only an example of a possible next step. If you're familiar with using IoT devices in AWS, this section can be skipped.

1. Select the “Act” tab on the AWS IoT dashboard and click “Create” to start a new rule.

2. Configure settings as desired. Following are the minimum settings required.



Create a rule

Create a rule to evaluate messages sent by your things and specify what to do when a message is received (for example, write data to a DynamoDB table or invoke a Lambda function).

Name: my_gateway_rule

Description: A rule to collect RF Messages in DynamoDB

Message source

Indicate the source of the messages you want to process with this rule.

Using SQL version: 2016-03-23

Rule query statement:

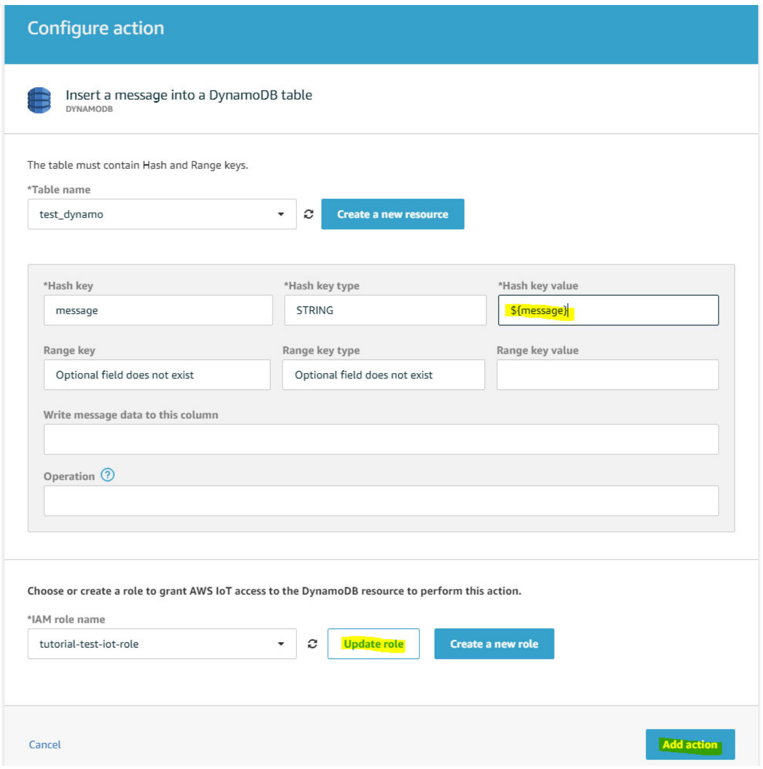
```
SELECT * FROM 'TestTopic'
```

Attribute: *

Topic filter: TestTopic

Figure 7 Configure rule settings.

3. Navigate to “Set one or more actions.”


Configure action

Insert a message into a DynamoDB table
DYNAMODB

The table must contain Hash and Range keys.

*Table name	test_dynamo	
*Hash key: message	*Hash key type: STRING	*Hash key value: ${message}
Range key: Optional field does not exist	Range key type: Optional field does not exist	Range key value:
Write message data to this column:		
Operation:		

Choose or create a role to grant AWS IoT access to the DynamoDB resource to perform this action.

*IAM role name: tutorial-test-iot-role

Figure 8 Add an action.

4. Select “Insert a message into a DynamoDB table.”
5. Set a Hash key value.
6. Select a role and update it.
7. Click “Add action.”After that, you can create the rule.
8. Check the AWS "Act" tab and ensure your rule is listed as enabled.

Connection Status

If a connection status topic has been configured (connection_status_topic), then the IP gateway will publish messages to this topic when its connection status has changed. The message is a brief description of the connection status. For example:

```
{
   "client_id": "inovonics-EN4080-VP123456789",
   "connection_status": "Connected",
   "message": "The gateway has established a
   connection to the broker."
}
```

- client_id is the IP gateway's client id as configured by the client.
- connection_status is the new status for the connection.
- Connected if the IP gateway is now connected to the broker.
- Disconnected if the IP gateway has cleanly disconnected from the broker.
- Dropped if the IP gateway has unexpectedly lost connection to the broker (using the MQTT Last Will & Testament functionality).

Inovonics Transmitter and Repeater Messages

RF messages are published on the configured topics in JSON format as a list of zero or more message dictionaries. It is recommended to use a JSON library to parse them into a more friendly data structure for your programming language. If using environmental transmitters, ensure your JSON library supports floating point numbers, including Infinity, as those values may be included. The Pythonjson module and the jq command are known to support floating point numbers correctly. In the following sections are examples of security, submetering, environmental, and repeater messages. Each message will include at least these fields:

- client_id is the IP gateway's client id as configured by the client.
- dispatch-type is one of the following:
  - Historical: Message was received while the IP gateway was offline, sent once reconnected.
  - Realtime: Message was sent to the MQTT broker as soon as it was received from the EchoStream network.
  - Batch: Message was received recently but held for the batch interval (15 minutes) before being sent with other received messages.
  - Discovery: Message for potentionally unregistered device was sent as soon as it was received from the EchoStream network.
- format is one of the following:
  - json: The message has been parsed into a JSON dictionary.
  - bytearray: The message is left as a list of byte values, unparsed (see the *EchoStream Developers Guide* for parsing).
- market is the kind of device that sent the message; either security, submetering, environmental, or repeater.
- message is the RF message itself.
- network-type is the type of RF network the message was received from; usually Echostream.
- timestamp is the when the message was received from the RF network.

- txid is the serial number of the device that sent the message.

Security Transmitter RF Message Example

```
[{"client-id": "inovonics-EN4080-VP123456789",
"dispatch-type": "Realtime",
   "format": "json", "market": "Security",
   "message": {"header": "InboundComplete",
   "metadata": {"firstHopUid": {"market": {"name":
   "Repeater"}, "serialNumber": 2204027},
   "hopCount": 1,
   "originatorUid": {"market": {"name":
   "Security"}, "serialNumber": 2685902},
   "traceCount": 0, "traceUidList": []},
   "payload": {"mcb": 62,
   "name": "SecurityApplicationPayload", "pti": 40,
   "status": {"alarm0": false, "alarm1": false,
   "alarm2": false, "alarm3": false, "cleanMe":
   false, "eolTamper": false, "lowBattery": false,
   "reset": false, "sleep": false, "stat0bit0":
   false, "stat0bit1": false, "stat0bit7": false,
   "stat1bit4": false, "stat1bit5": false,
   "supervision": true, "tamper": true}},
   "signal": {"level": 26, "margin": 26}},
   "network-type":  "echostream", "timestamp":
   "2019-08-22T20:28:44.632521", "txid":
   "2685902"},
...]
```

Submetering Transmitter RF Message Example

```
[{"client-id": "inovonics-EN4080-VP123456789",
"dispatch-type": "Realtime",
   "format": "json", "market": "Submetering",
   "message": {"header": "InboundComplete",
   "metadata": {"firstHopUid": {"market": {"name":
   "NetworkCoordinator"}, "serialNumber":
   12177055},
   "hopCount": 0,
   "originatorUid": {"market": {"name":
   "Submetering"}, "serialNumber": 12910813},
   "traceCount": 0, "traceUidList": []},
   "payload": {"count": 0,
   "leakDetectCount": 0,
   "mcb": 61,
   "name": "SubmeteringApplicationPayload", "pti":
   1,
   "status": {"deltaTotalizer": false,
   "lowBattery": false, "rapidTransmissionMode":
   false, "reset": false,
   "shipping": false, "stat0bit0": false,
   "stat0bit1": false, "stat0bit7": false,
   "stat1bit0": false, "stat1bit1": false,
   "stat1bit2": false, "stat1bit3": false,
   "stat1bit4": false, "stat1bit5": false,
   "supervision": false, "tamper": true}},
   "signal": {"level": 67, "margin": 66}},
   "network-type":  "echostream", "timestamp":
   "2019-08-22T21:20:28.944177", "txid":
   "12910813"},
...]
```

Environmental Transmitter RF Message Example

```
[{"client-id": "inovonics-EN4080-VP123456789",
"dispatch-type": "Realtime",
   "format": "json", "market": "Environmental",
   "message": {"header": "InboundComplete",
   "metadata": {"firstHopUid": {"market": {"name":
   "Repeater"}, "serialNumber": 10360932},
   "hopCount": 1,
   "originatorUid": {"market": {"name":
   "Environmental"}, "serialNumber": 4148924},
   "traceCount": 0, "traceUidList": []},
   "payload": {"data": [75.63200378417969,
   45.04424285888672],
   "mcb": 60,
   "name": "EnvironmentalApplicationPayload",
   "pti": 21,
   "status": {"alarm0": false, "alarm1": false,
   "alarm2": false, "alarm3": false, "binaryInput":
   false, "deltaSensor1": false, "deltaSensor2":
   true, "lowBattery": false, "reset": false,
   "stat0bit0": false, "stat0bit1": false,
   "stat0bit2": false, "stat0bit7": false,
   "stat1bit4": false, "supervision": false,
   "tamper": false}},
   "signal": {"level": 25, "margin": 25}},
   "network-type":  "echostream", "timestamp":
   "2019-08-22T20:28:46.040172", "txid":
   "4148924"},
...]
```

Repeater RF Message Example

```
[{"client-id": "inovonics-EN4080-VP182120006",
"dispatch-type": "Realtime",
    "format": "json", "market": "Repeater",
    "message": {"header": "InboundComplete",
    "metadata": {"firstHopUid": {"market": {"name":
    "NetworkCoordinator"}, "serialNumber":
    13033338},
    "hopCount": 0,
    "originatorUid": {"market": {"name":
    "Repeater"}, "serialNumber": 16004699},
    "traceCount": 0, "traceUidList": []},
    "payload": {"mcb": 65,
    "name": "RepeaterApplicationPayload",
    "repeaterPayload": {"name":
    "RepeaterStatusWithBroadcastRepeater", "pti": 0,
    "status": {"jammed": false, "layer": 0,
    "limitedScanningRemotes": false,
    "lossOfLinePower": true, "lowBattery": false,
    "neighborListAssigned": false, "noChange":
    false, "onlyOneNeighbor": false, "reset": false,
    "stat0bit0": false, "stat0bit2": false,
    "statusAck": false, "tamper": false}}},
    "signal": {"level": 66, "margin": 62}},
    "network-type":  "echostream", "timestamp":
    "2019-08-22T21:23:22.509580", "txid":
    "16004699"},
...]
```