



Using SL1 Publisher

SL1 version 10.1.0 Beta

Table of Contents

Introduction to SL1 Publisher	4
What is Publisher?	5
How Does Publisher Work?	5
Data Channels	5
Output Adapter	6
Tools Included with Publisher	6
Prerequisites for Using Publisher	6
Workflow for Using Publisher	7
Enabling SL1 Publisher	8
Enabling Publisher	9
Enabling Publisher During a New SL1 Installation	9
Enabling Publisher on an Existing SL1 System	9
Enabling Collector Pipeline	10
Verifying the Installation	10
Publisher API	12
About the Publisher API	13
Accessing the Publisher API	13
Finding the IP Address of the Publisher API	13
Using the Publisher API Interface	14
Managing Publisher Subscriptions	16
What is a Subscription?	17
Subscription Parameters	17
Supported Data Channels	17
Supported Output Adapters	17
Configuring Communication with Kafka	18
Plaintext Communication with Kafka	18
SSL Communication with Kafka	18
What is SSL?	18
Certificate Files	19
Configuring Kafka Two-Factor Authentication for Subscriptions	19
Creating a Subscription	19
Retrieving Subscriptions	20
Retrieving All Subscriptions	20
Retrieving a Single Subscription	21
Updating a Single Subscription	22
Understanding Subscription Updates	23
Deleting a Single Subscription	24
Managing Publisher Data Channels	25
What is a Data Channel?	26
Retrieving All Data Channels	26
Retrieving a Single Data Channel	26
Updating a Data Channel	27
Viewing Publisher Health	29
What Does the Health Endpoint Report?	30
Retrieving Liveness Status	30
Retrieving Readiness Status	30
sl-pubsub Library	32
Installing the sl-pubsub Library	33
Accessing the sl-pubsub Library Documentation	33
Troubleshooting Publisher	35

Why does my Publisher pod keep crashing or restarting?	36
Why is Publisher failing to send data even though it is not crashing?	36
When trying to change the name or ID or a data type, it does not work	37
When navigating to the URL shown in the Kubernetes ingress, a Not Found error occurs	37
When entering a POST or PUT request, a "Browser (or proxy) sent a request that this server could not understand" error occurs	37

Chapter



1

Introduction to SL1 Publisher

Overview

This chapter describes Publisher.

Use the following menu options to navigate the SL1 user interface:

- To view a pop-out list of menu options, click the menu icon ().
- To view a page containing all of the menu options, click the Advanced menu icon ().

This chapter covers the following topics:

<i>What is Publisher?</i>	5
<i>How Does Publisher Work?</i>	5
<i>Tools Included with Publisher</i>	6
<i>Prerequisites for Using Publisher</i>	6
<i>Workflow for Using Publisher</i>	7

What is Publisher?

Publisher is a service that retrieves near real-time availability and interface performance data from SL1 Data Collectors or the SL1 Agent and delivers the data through an output adapter to a third-party destination for long-term data storage, analysis, or reporting. For example, Publisher can send data to Kafka topics as a helm chart (a collection of files describing a related set of Kubernetes resources).

Publisher supports the concept of *subscriptions*, which allow you to specify what data you want and how you want it presented.

Publisher includes its own API to allow interaction with the service. You can optionally download a library of Python functions (*sl-pubsub*) from the ScienceLogic repository for managing the data received by Publisher.

Publisher is an opt-in service that you can enable in the SL1 Extended architecture. Publisher runs on the Management Node. The client software runs on customer hardware.

How Does Publisher Work?

Publisher listens to a list of **data channels** in SL1 to determine if new data is available.

If new data is available, Publisher creates a binary bundle and sends the bundle to an **output adapter**.

To receive Publisher data, you can create one or more **subscriptions**. Each subscription defines the data channels from which you want to receive data and the output adapter to which you want to send the data. For more information, see [Managing Publisher Subscriptions](#).

Optionally, you can use the *sl-pubsub* library to unpack the binary data bundles on your third-party system. For more information, see [sl-pubsub Library](#).

Data Channels

A *data channel* streams data. The name of the data channel describes the type of data that it streams.

Publisher supports the following data channels:

Name	ID	Description
availability	1	Publisher stream for availability data from SL1
interface	3	Publisher stream for interface data from SL1

When you define a subscription, you supply the name of one or more data channels for the **data_types** key in the JSON file. For more information, see [Managing Data Channels](#).

Output Adapter

An output adapter defines information about the destination of the data from Publisher.

Publisher supports the following output adapters:

Name	Required Parameter(s)	Description
Line printer	None	Publisher will send data to standard output (stdout). This is helpful for debugging. ScienceLogic does not recommend this output adapter for general use.
File writer	<i>filename</i> . The name of the destination file where you want to send the data.	Publisher will send data to the specified file. This is helpful for debugging. ScienceLogic does not recommend this output adapter for general use.
Kafka adapter	<i>topic</i> . String. The name of the Kafka topic. <i>server</i> . String with IP address and port number. The host name:port number of the Kafka server.	Publisher will send data to the specified Kafka topic/server pair. You can specify multiple topics and servers using spaces.

When you define a subscription:

- You supply the name of each output adapter in the **output_name** key in a JSON file.
- You supply parameters for that output adapter in the **output_config** key in a JSON file.

Tools Included with Publisher

- **API**. Publisher includes an API for defining subscriptions and managing subscriptions. A Swagger user interface is provided for ease of use.
- **Metrics**. Publisher communicates with a Prometheus pod to store metrics about Publisher.

Prerequisites for Using Publisher

Before you can use Publisher, you must do the following:

- Deploy SL1 10.1.0 and the SL1 Extended Architecture. You can deploy Publisher when you deploy SL1 10.1.0 and the SL1 Extended Architecture or you can enable and deploy Publisher on an existing SL1 Extended Architecture system. For more information, see [Enabling Publisher](#).
- Enable the Collector Pipeline. For more information, see [Enabling the Collector Pipeline](#).
- Ensure you have SSH or console access to the Management Node so you can access Docker and Kubernetes

Optionally, you can install the following:

- **sl-pubsub Library**. A library of Python functions that enable you to unpack the binary bundles sent by Publisher into JSON messages on your third-party system.

NOTE: The sl-pubsub library requires Python 3.

Workflow for Using Publisher

The following steps represent the general workflow for implementing Publisher. A pre-configuration checklist is available in the next section.

Step	Description	References
1. Enable Publisher.	Enable Publisher during initial SL1 Extended deployment or on an existing SL1 Extended architecture by running deploy again.	Enabling SL1 Publisher
2. Enable the Collector Pipeline.	Collector Pipeline is required for Publisher.	Enabling the Collector Pipeline
3. Access the Publisher API.	The Publisher API is a Swagger UI that lets you manage subscriptions,	Accessing the Publisher API
4. (Optional) Define SSL certificates for Kafka.	If you use two-factor authentication to communicate with your Kafka installation, set up the certificates for use by Publisher.	Configuring Kafka SSL Communication
5. Define subscriptions.	Define subscriptions using the Publisher API.	Creating Publisher Subscriptions
6. (Optional) Install the sl-pubsub library.	Download and install the sl-pubsub library from the ScienceLogic Customer Portal. The documentation for sl-pubsub is contained in the downloadable .zip file.	Installing the sl-pubsub Library
7. Use the sl-pubsub library to unpack the binary data bundles sent from SL1.	See sl-pubsub documentation, included in the sl-pubsub .zip file.	—

Chapter



2

Enabling SL1 Publisher

Overview

This chapter describes how to enable Publisher on an SL1 Extended Architecture.

Use the following menu options to navigate the SL1 user interface:

- To view a pop-out list of menu options, click the menu icon ().
- To view a page containing all of the menu options, click the Advanced menu icon ().

This chapter covers the following topics:

<i>Enabling Publisher</i>	9
<i>Enabling Collector Pipeline</i>	10
<i>Verifying the Installation</i>	10

Enabling Publisher

Publisher is an opt-in service that is included with the SL1 Extended architecture but is not enabled by default. When Publisher is enabled, you will see two Kubernetes pods in your SL1 stack:

- publisher- <hash>
- publisher-api- <hash>

NOTE: Kubernetes automatically assigns the hash for the Publisher services.

Enabling Publisher During a New SL1 Installation

For new installations, you can enable Publisher during deployment.

For details on installing the SL1 Extended Architecture and enabling Publisher during install, see the *Installation* manual.

Enabling Publisher on an Existing SL1 System

You can enable Publisher on an existing SL1 Extended Architecture by editing the `sl1x-inv.yml` file and running `deploy` again, as described in this section.

To enable Publisher on an existing SL1 Extended Management Node:

1. Either go to the console of the Management Node or use SSH to access the Management Node. Open a shell session on the server. Log in with the system password.
2. At the shell prompt, navigate to your deploy directory, as follows:
`cd sl1x-deploy`
3. Edit the file `sl1x-inv.yml`, as follows:
`vi sl1x-inv.yml`

CAUTION: Do not remove colons when editing this file.

4. In the `all: vars:` section of the file, set **`install_publisher`** to `true`.
`install_publisher: true`
5. Save and exit the file.
6. Run the deploy script.
`docker-compose -f docker-compose.internal.yml run --rm deploy shell`
7. (Optional) If you want to view the Publisher configuration, you can `cat` the `publisher-values.yml` file. For example:
`cat output-files/publisher-values.yml`

Enabling Collector Pipeline

Collector Pipeline is a platform feature that allows horizontal scaling (adding more Data Collectors and Agent installations) without data loss or performance loss.

Collector Pipeline also support the new beta feature, Publisher and the new beta feature, Anomaly Detection.

Currently, Collector Pipeline supports availability data, interface data, and data from Performance Dynamic Applications. SL1 will add more data types in future releases.

NOTE: If you want to use Anomaly Detection, enable Collector Pipeline with data from Performance Dynamic Applications.

To enable Collector Pipeline for availability data, interface data, and anomaly detection:

1. Either go to the console of the Database Server or use SSH to access the Database Server. Open a shell session on the server. Log in with the system password you defined in the ISO menu.
2. To view information about the command, enter the following at the shell prompt:

```
/opt/em7/backend/set_cpl.py -help
```

3. To enable Collector Pipeline for availability data, interface data, and anomaly detection, enter the following at the shell prompt:

```
/opt/em7/backend/set_cpl.py -d availability ENABLE
```

```
/opt/em7/backend/set_cpl.py -d interface ENABLE
```

```
/opt/em7/backend/set_cpl.py -d da_perf ENABLE
```

4. To disable Collector Pipeline for availability data, interface data, and anomaly detection, enter the following at the shell prompt:

```
/opt/em7/backend/set_cpl.py -d availability DISABLE
```

```
/opt/em7/backend/set_cpl.py -d interface DISABLE
```

```
/opt/em7/backend/set_cpl.py -d da_perf DISABLE
```

Verifying the Installation

To verify the installations:

1. Either go to the console of the Management Node or use SSH to access the Management Node. Open a shell session on the server. Log in with the **System Password** you defined in the ISO menu.
2. At the shell prompt, navigate to your deploy directory, as follows:
`cd s11x-deploy`
3. To verify the Publisher pod is up and running, run the following command:
`kubectl get pods | grep publisher`

The command should return two pods:

- publisher-<hash>
- publisher-api-<hash>

The pods will report "1/1 Ready" when they are operational.

3. To verify the publisher-api service is up:
`kubectl get services | grep publisher`
4. To verify the publisher-api endpoint is up:
`kubectl get endpoint | grep publisher`
5. To verify the publisher-api ingress is up:
`kubectl get ingress | grep publisher`

Chapter



3

Publisher API

Overview

This chapter describes how to access and interact with the Publisher API through the user interface.

Use the following menu options to navigate the SL1 user interface:

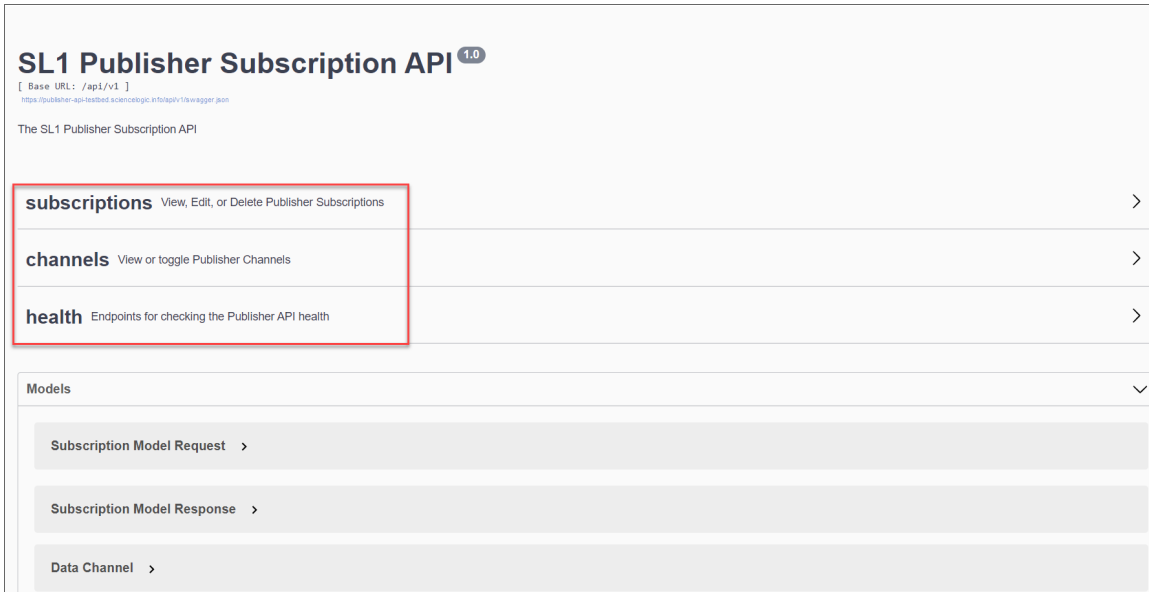
- To view a pop-out list of menu options, click the menu icon ().
- To view a page containing all of the menu options, click the Advanced menu icon ().

This chapter covers the following topics:

<i>About the Publisher API</i>	13
<i>Accessing the Publisher API</i>	13

About the Publisher API

The Publisher API provides a user interface to the available endpoints (subscriptions, channels, and health) using Swagger. You can click on each endpoint to see the available actions for that endpoint.



Accessing the Publisher API

Finding the IP Address of the Publisher API

To find the IP address of the Publisher API:

1. Either go to the console of the Management Node or use SSH to access the Management Node.
2. Open a shell session on the server.
3. Change to the deployment directory. For example:
`cd sl1x-deploy/`
4. Run the following command:
`docker-compose -f docker-compose.internal.yml run --rm deploy shell`
5. At the command prompt, enter the following:
`$ kubectl get ingress | grep publisher`

The command outputs a Kubernetes ingress called `publisher-api-ingress`.

6. Note the URL for the publisher-api in the **HOSTS** section of the output. The actual URL for the publisher-api will be `<URL returned by kubectl>/api/v1`.

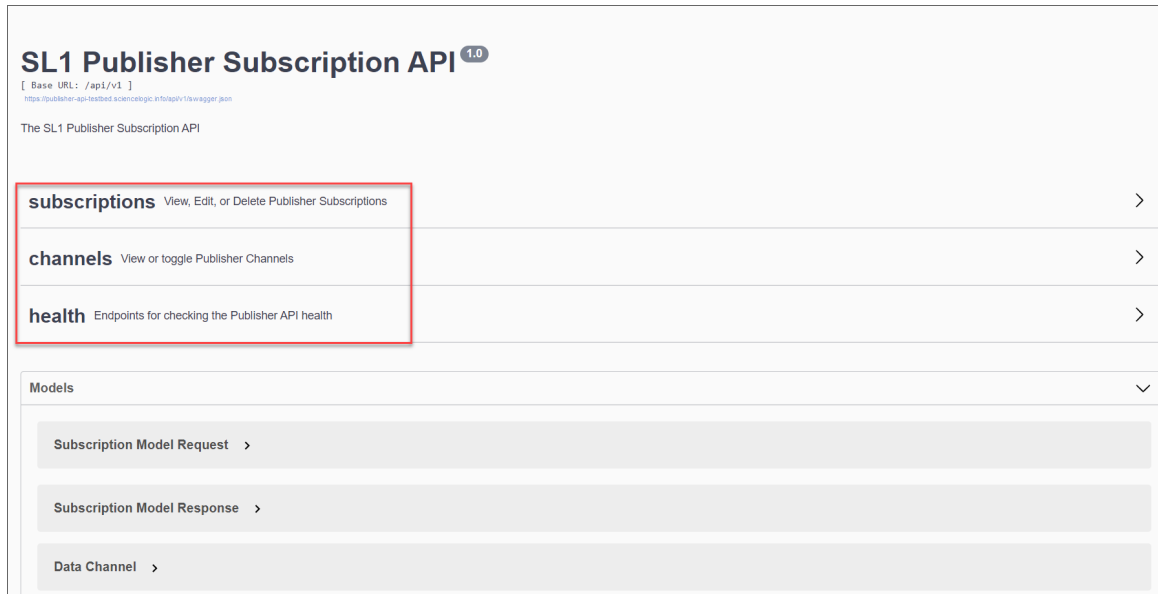
Using the Publisher API Interface

To access the Publisher API in a web browser:

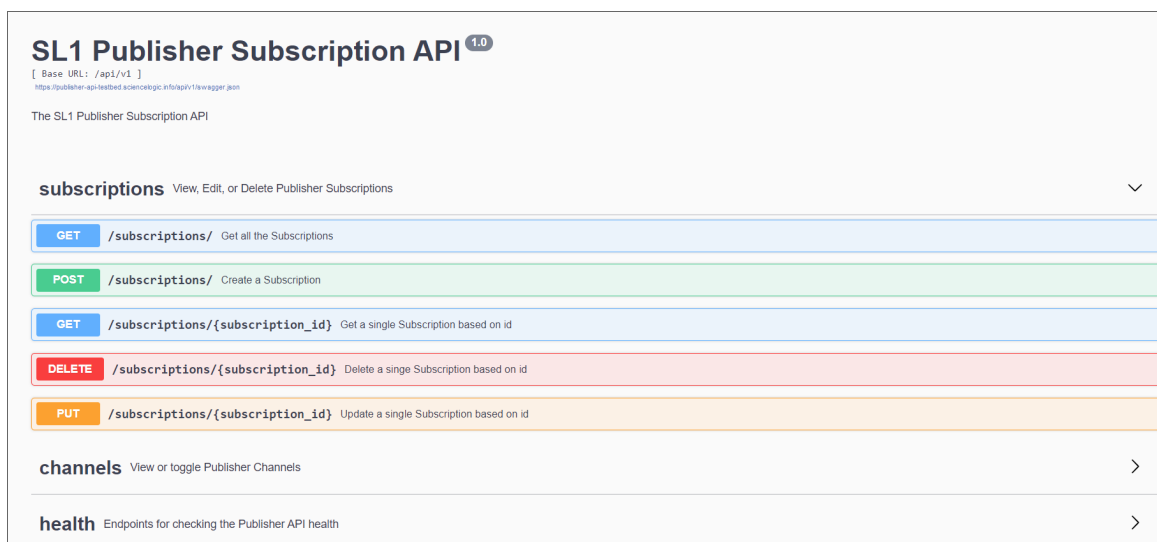
1. Open a browser and enter the following URL:

`<URL returned by kubectl>/api/v1`

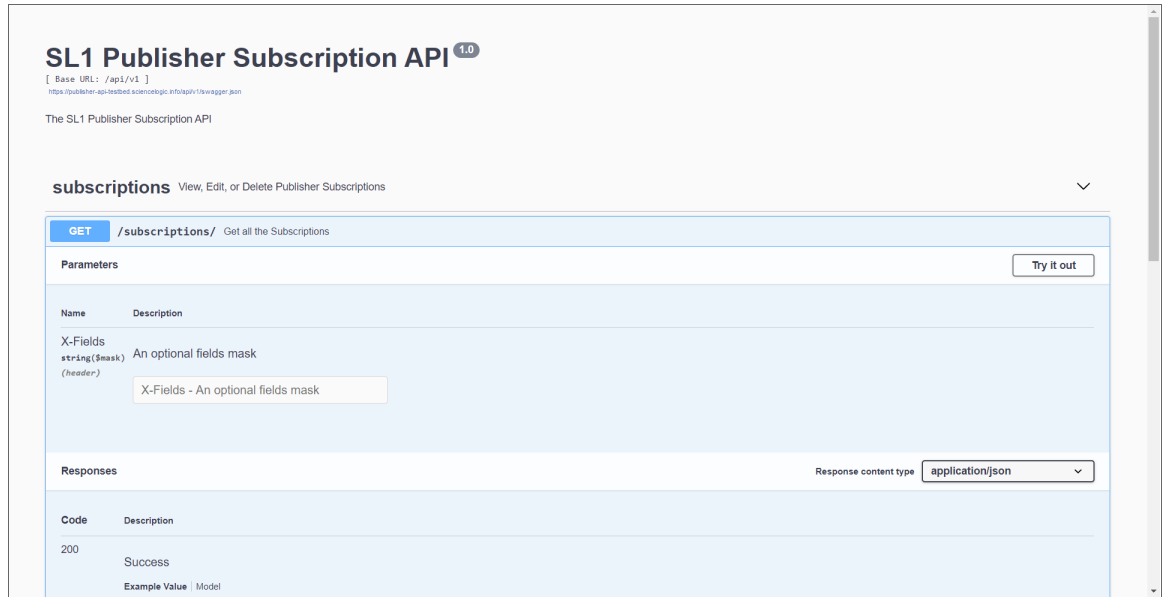
The browser will display a Swagger page for the Publisher API.



2. Click on the endpoint you are interested in to see all available actions. For example, if we click the "subscriptions" endpoint, it expands to show the following available actions:



3. Click an action to modify or execute it. In the example below, we clicked **GET /subscriptions**. This action will retrieve a list of all available subscriptions. For more information about this action, see [Retrieving All Subscriptions](#).



The screenshot displays the API documentation for the SL1 Publisher Subscription API (version 1.0). The base URL is /api/v1. The main heading is "subscriptions" with a sub-heading "View, Edit, or Delete Publisher Subscriptions". The selected endpoint is "GET /subscriptions/ Get all the Subscriptions".

Parameters

Name	Description
X-Fields string(\$mask) (header)	An optional fields mask

X-Fields - An optional fields mask

Responses

Response content type: application/json

Code	Description
200	Success

Example Value | Model

For more information about the API endpoints, refer to the following sections:

- [Managing Publisher Subscriptions](#)
- [Managing Data Channels](#)
- [Viewing Publisher Health](#)

Chapter



4

Managing Publisher Subscriptions

Overview

This chapter describes how to create and manage subscriptions for Publisher using the *subscriptions* endpoint.

Use the following menu options to navigate the SL1 user interface:

- To view a pop-out list of menu options, click the menu icon ().
- To view a page containing all of the menu options, click the Advanced menu icon ().

This chapter covers the following topics:

<i>What is a Subscription?</i>	17
<i>Configuring Communication with Kafka</i>	18
<i>Creating a Subscription</i>	19
<i>Retrieving Subscriptions</i>	20
<i>Updating a Single Subscription</i>	22
<i>Deleting a Single Subscription</i>	24

What is a Subscription?

Publisher listens to a list of **data channels** in SL1 to determine if new data is available and sends it to a destination based on its active subscriptions. A **subscription** is an object that specifies how and where to send the data. To receive Publisher data, you can create one or more subscriptions.

Subscription Parameters

A subscription is defined by the following fields:

- **name**. Specifies a name for the subscription.
- **enabled**. Specifies whether the subscription is active (enabled) or inactive (disabled).
- **output_name**. Specifies the type of subscription destination (for example, kafka).
- **cafile, certfile, keyfile**. Fields used in Kafka with two-factor authentication to specify the keyfile. For more information, see [SSL Communication with Kafka](#).
- **output_config**. A JSON structure containing output configuration options.
- **data_types**. List of all the data_types this subscription to which this subscription should listen.

Supported Data Channels

Currently, Publisher supports two data channels, specified in the subscription as **data_types**:

- **Availability**. Publisher stream for availability data from SL1.
- **Interface**. Publisher stream for interface performance data from SL1.

Supported Output Adapters

Publisher sends the data to an **output adapter**, specified in the subscription as **output_name**. The following output adapters are supported:

- **Line printer**. Publisher will send data to standard output (stdout). No additional configuration is required when sending to the line printer. This option is helpful for debugging.

JSON declaration:

```
"output_config" : {}
```

- **File writer**. Publisher will send data to the specified file. This option is helpful for debugging.

Example JSON declaration:

```
"output_config" : {  
  "filename" : /some_location/some_directory/some_file  
}
```

- **Kafka adapter**. Publisher will send data to the specified kafka topic/server pair.

Configuring Communication with Kafka

To enable communication with Kafka, you can either use plaintext or SSL. This section describes the parameters you must set in the "output_config" section of the subscription to enable this communication.

Plaintext Communication with Kafka

Publisher accepts all configuration types for Kafka and passes them through to the third-party Kafka library included in SL1. When "output_name" is set to "kafka", a minimum of two fields are expected: a *topics* key and a *servers* key.

Example JSON declaration for Kafka plaintext configuration:

```
"output_config" : {  
  "topics": "testTopic",  
  "servers": "KafkaServer:9092"  
}
```

Note the following:

- You can specify more than one topic by using a space to separate topics in a list.
- You can specify more than one server by using a space to separate servers in a list.
- If you do not specify a host in the servers field, Publisher will default to localhost.
- If you do not specify a port in the servers field, Publisher will default to port 9092.

For more information about available configurations, see the Kafka documentation at <https://kafka-python.readthedocs.io/en/master/index.html>.

SSL Communication with Kafka

Publisher can communicate with Kafka over SSL when you configure two-factor authentication.

What is SSL?

SSL is an acronym for Secure Sockets Layer. SSL is a protocol for securely transmitting data via the internet. SSL uses a private key to encrypt data to be transferred over the Internet connection. Usually, URLs that include "HTTPS" are using SSL for security.

To implement SSL, an SSL certificate resides on the web server and is used to encrypt the data and to identify the website. The SSL certificate contains information about the certificate holder, the domain for which the certificate was issued, the name of the Certificate Authority who issued the certificate, and the root and the country in which the certificate was issued.

There are two ways to acquire an SSL certificate:

- You can purchase a certificate from a vendor (called a "certificate authority"), such as VeriSign or GeoTrust.
- You can "self-sign" your own certificate. Using available tools (both open source and proprietary), you can create and sign your own SSL certificate instead of purchasing from a certificate authority.

Certificate Files

To configure Publisher for SSL communication with Kafka, you need at least three certificate files:

- **cafile**. A Certificate Authority (CA) file used in certificate verification. This corresponds to the `ssl_cafile` parameter in Kafka.
- **certfile**. A client certificate file (.pem format) and any files needed to verify the certificate's authenticity. This corresponds to the `ssl_certfile` parameter in Kafka.
- **keyfile**. Client private key. This corresponds to the `ssl_keyfile` parameter in Kafka.

Publisher ingests the certificate files as base64-encoded file contents. Prior to moving to the next step, use a command such as the following to encode and output the contents of each file. You will copy and paste these contents in the next procedure.

```
cat <file_path>/<file_name> | base64 -w 0
```

NOTE: Publisher does not support the use of newline in any of the certificate files. The command above encodes the file without newlines, making it one continuous string.

Configuring Kafka Two-Factor Authentication for Subscriptions

To enable two-factor authentication in a subscription, you must include:

- The declaration `"auth_protocol": "two_way_auth"` in the `output_config` section, and
- The filenames of the `cafile`, `certfile`, and `keyfile` must be included in the subscription.

For example:

```
"output_config" : {
  "topics": "testTopic",
  "servers": "testKafka:testPort",
  "auth_protocol": "two_way_auth"
  "security_protocol": "SSL",
  "cafile": "<insert_file_contents>",
  "certfile": "<insert_file_contents>",
  "keyfile": "<insert_file_contents>"
}
```

Creating a Subscription

To create a subscription:

1. Navigate to the API URL. For example: **10.10.10.10/api/v1**.
2. Click on **POST /subscriptions** in the *subscriptions* section.
3. Click **[Try it out]**.

4. Fill in **Payload** with the desired subscription configuration. The following example shows a new subscription called "test_subs", which is a subscription to the Kafka availability data channel.

The screenshot shows a REST client interface for the endpoint `POST /subscriptions/` with the sub-header "Create a Subscription". The "Parameters" section is visible, and the "payload" field is required. The payload is an object with the following JSON structure:

```
{
  "name": "test_subs",
  "enabled": true,
  "output_name": "kafka",
  "output_config": {
    "key": "value"
  },
  "data_types": [
    "availability"
  ]
}
```

Below the payload field, there is a "Parameter content type" dropdown menu set to `application/json`. At the bottom, there is an "X-Fields" section with a text input field containing `X-Fields - An optional fields mask`.

5. Click **[Execute]**.
6. Verify in the response that the subscription was created successfully. An example response for our new subscription is shown below.

The screenshot shows the "Server response" section of the REST client. The status code is `200`. The "Response body" is displayed in a dark-themed code editor with the following JSON structure:

```
{
  "results": {
    "id": 1,
    "name": "test_subs",
    "enabled": true,
    "output_name": "kafka",
    "output_config": {
      "key": "value"
    },
    "conf": {},
    "certfile": "",
    "keyfile": ""
  },
  "data_types": [
    "availability"
  ]
}
```

Below the response body, the "Response headers" are listed:

```
content-length: 178
content-type: application/json
date: Tue, 29 Sep 2020 19:17:08 GMT
server: nginx/1.15.6
status: 200
strict-transport-security: max-age=15724800; includeSubDomains
```

NOTE: The data type ID value is auto-generated and cannot be modified.

Retrieving Subscriptions

This section describes how to retrieve one or more subscriptions from the API.

Retrieving All Subscriptions

To retrieve all subscriptions:

1. Navigate to the API URL. For example: **10.10.10.10/api/v1**.
2. Click on **GET /subscriptions** in the *subscriptions* section.
3. Click **[Try it out]**.

The screenshot shows an API client interface for the endpoint `GET /subscriptions/`. The interface includes a **Parameters** section with a table:

Name	Description
X-Fields string (mask) (header)	An optional fields mask <input type="text" value="X-Fields - An optional fields mask"/>

Below the parameters is an **Execute** button and a **Clear** button. The **Responses** section shows the response content type set to `application/json`. The **Curl** section displays the command:

```
curl -s GET "http://localhost:5000/api/v1/subscriptions/" -H "accept: application/json"
```

The **Request URL** is `http://localhost:5000/api/v1/subscriptions/`. The **Server response** section shows a **Code** of `200` and a **Response body** containing the following JSON:

```
{
  "results": [
    {
      "id": 1,
      "name": "testing",
      "machine": "lrv",
      "output_name": "kafka",
      "output_config": {
        "engine": "testing",
        "servers": "kafka:9092"
      }
    }
  ],
  "data_types": [
    {
      "availability": true
    }
  ]
}
```

The **Response headers** section shows:

```
content-length: 167
content-type: application/json
date: Wed, 04 Sep 2019 16:26:47 GMT
server: Werkzeug/0.15.0 Python/3.6.9
```

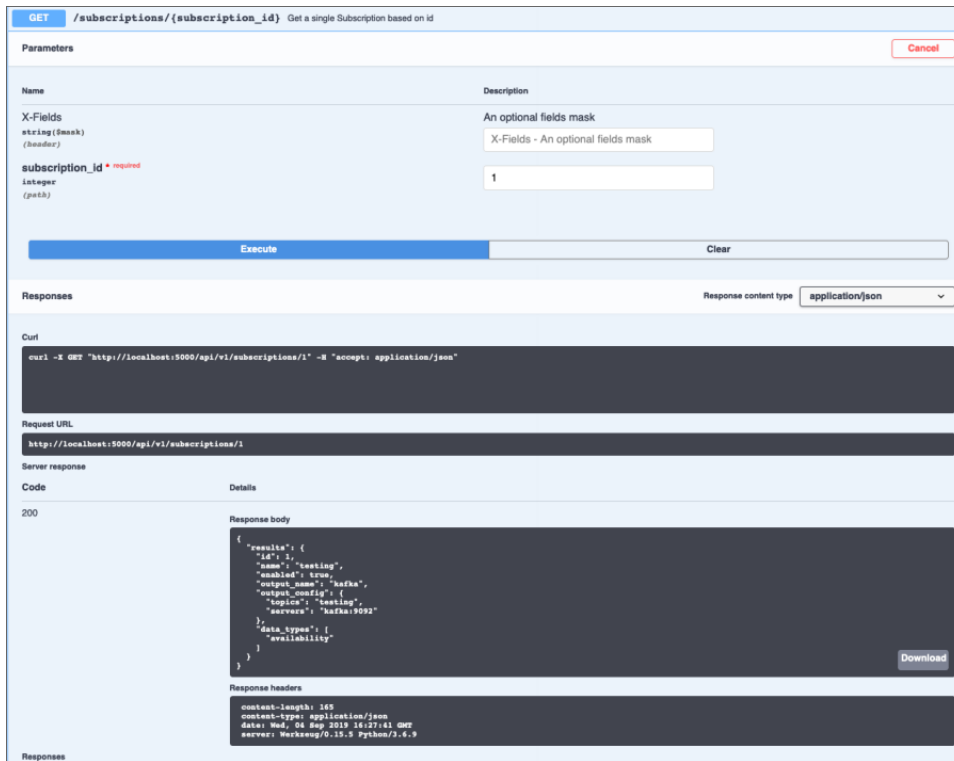
4. Click **[Execute]**.
5. View the subscriptions returned in the response.

Retrieving a Single Subscription

To retrieve a specific subscription by subscription ID:

1. Navigate to the API URL. For example: **10.10.10.10/api/v1**.
2. Click on **GET /subscriptions/{subscription_id}** in the *subscriptions* section.
3. Click **[Try it out]**.

4. Insert the `subscription_id` you want to view (for example, 1).



The screenshot shows a REST client interface for a GET request to `/subscriptions/{subscription_id}`. The parameters section includes `X-Fields` (string(mask), header) and `subscription_id` (integer, path) with the value `1`. The response section shows a 200 status code and a JSON response body:

```
{
  "results": {
    "id": 1,
    "name": "testing",
    "enabled": true,
    "output_name": "sdfta",
    "output_config": {
      "topics": "testing",
      "servers": "herka1000"
    },
    "data_type": [
      "availability"
    ]
  }
}
```

The response headers are:

```
content-length: 169
content-type: application/json
date: Wed, 04 Sep 2019 16:27:41 GMT
server: Werkzeug/0.15.5 Python/3.6.9
```

5. Click **[Execute]**.
6. View the subscription returned in the response.

Updating a Single Subscription

To update a subscription:

1. Navigate to the API URL. For example: `10.10.10.10/api/v1`.
2. Click on **PUT /subscription/{subscription_id}** in the `subscriptions` section.
3. Click **[Try it out]**
4. Insert the `subscription_id` you want to update (for example, 1).

5. Edit the fields you want to reconfigure. You do not need to edit each field, only the ones you want to update.

The screenshot shows a REST client interface for a PUT request to `/subscriptions/{subscription_id}`. The request body is a JSON object: `{ "name": "new name", "output_config": { "topics": "new_topic" } }`. The `subscription_id` path parameter is set to `1`. The interface includes an `Execute` button and a `Responses` section showing the curl command: `curl -X PUT "http://localhost:9000/api/v1/subscriptions/1" -H "accept: application/json" -H "Content-Type: application/json" -d '{ "name": "new name", "output_config": { "topics": "new_topic" } }` and the request URL: `http://localhost:9000/api/v1/subscriptions/1`.

6. Click **[Execute]**.
7. View the updated subscription returned in the response.

Understanding Subscription Updates

Updating a subscription applies your input to the adapter. For example, given the following subscription:

```
{ "id": 1,
  "name": "string",
  "enabled": true,
  "output_name": "kafka",
  "output_config": {
    "topics": "testTopic",
    "servers": "10.10.10.10" },
  "data_types": [ "availability" ] }
```

And we apply the following in an update request to ID 1:

```
{
  "output_config": {
    "topics": "newTopic"
  }
}
```

The subscription then becomes:

```
{ "id": 1,
  "name": "string",
```

```
"enabled": true,  
"output_name": "kafka",  
"output_config": {  
  "topics": "newTopic",  
"data_types": [ "availability" ] }
```

Notice that the **servers** field has disappeared. You can think of this as replacing the contents of a given field with new contents.

NOTE: If you get no update or server response when you execute the PUT, check your JSON format for any trailing commas or brackets that should not appear. Use a JSON validator, if necessary, to check your format.

Deleting a Single Subscription

To delete a subscription:

1. Navigate to the API URL. For example: **10.10.10.10/api/v1**.
2. Click on **DELETE /subscription/{subscription_id}** in the **subscriptions** section.
3. Insert the *subscription_id* of the subscription you want to delete.
4. Click **[Execute]**.
5. View the deleted subscription returned in the response.

Chapter



5

Managing Publisher Data Channels

Overview

This chapter describes how to manage data channels with Publisher using the *channels* endpoint.

Use the following menu options to navigate the SL1 user interface:

- To view a pop-out list of menu options, click the menu icon ().
- To view a page containing all of the menu options, click the Advanced menu icon ().

This chapter covers the following topics:

<i>What is a Data Channel?</i>	26
<i>Retrieving All Data Channels</i>	26
<i>Retrieving a Single Data Channel</i>	26
<i>Updating a Data Channel</i>	27

What is a Data Channel?

Publisher listens to a list of **data channels** in SL1 to determine if new data is available and sends it to a destination based on its active subscriptions.

Currently, Publisher supports two data channels, specified in the subscription as **data_types**:

- **Availability**. Publisher stream for availability data from SL1.
- **Interface**. Publisher stream for interface performance data from SL1.

Retrieving All Data Channels

This section describes how to retrieve the full list of data channels from the API.

To get information about all data channels:

1. Navigate to the API URL. For example: **10.10.10.10/api/v1**.
2. Click on **GET /channels** in the **channels** section.
3. Click **[Try it out]**.
4. Click **[Execute]**.
5. View the data channels contained in the server response.



The screenshot displays a 'Server response' window with a 'Code' tab selected, showing a 200 status code. The 'Details' tab shows the 'Response body' as a JSON array of three data channel objects. The first object is 'availability' (id: 1, enabled: true), the second is 'agent_pipeline' (id: 2, enabled: false), and the third is 'interface' (id: 3, enabled: true). Below the response body, the 'Response headers' are listed, including content-length, content-type, date, server, status, and strict-transport-security.

```
Server response
Code    Details
200     Response body
{
  "results": [
    {
      "id": 1,
      "name": "availability",
      "enabled": true
    },
    {
      "id": 2,
      "name": "agent_pipeline",
      "enabled": false
    },
    {
      "id": 3,
      "name": "interface",
      "enabled": true
    }
  ]
}
Response headers
content-length: 152
content-type: application/json
date: Thu, 08 Oct 2020 16:27:14 GMT
server: nginx/1.15.6
status: 200
strict-transport-security: max-age=15724800; includeSubDomains
```

NOTE: The data type ID value is auto-generated and cannot be modified.

Retrieving a Single Data Channel

This section describes how to retrieve a given data channel from the API when you know the channel ID.

To retrieve a data channel with the channel ID:

1. Navigate to the API URL. For example: **10.10.10.10/api/v1**.
2. Click on **GET /channels/{channel_id}** in the *channels* section.
3. Click **[Try it out]**.
4. Insert the *channel_id* you want to view (for example, 1).
5. Click **[Execute]**.
6. View the channel information returned in the server response.



Server response

Code	Details
200	<p>Response body</p> <pre>{ "results": { "id": 1, "name": "availability", "enabled": true } }</pre> <p>Response headers</p> <pre>content-length: 58 content-type: application/json date: Thu, 08 Oct 2020 16:34:08 GMT server: nginx/1.15.6 status: 200 strict-transport-security: max-age=15724800; includeSubDomains</pre>

Updating a Data Channel

This section describes how to retrieve a given data channel from the API when you know the channel ID. You would use this procedure if you wanted to modify the name of the channel or to enable or disable the channel.

NOTE: The data type ID value is auto-generated and cannot be modified.

To update a data channel:

1. Navigate to the API URL. For example: **10.10.10.10/api/v1**.
2. Click on **PUT /channels/{channel_id}** in the *channels* section.
3. Click **[Try it out]**
4. Insert the *channel_id* you want to update (for example, 1).

5. Edit the fields you want to reconfigure. You do not need to edit each field, only the ones you want to update. In the following example, we will disable channel 1 by changing the "enabled" parameter to "false".

The screenshot shows a 'Parameters' configuration window with a 'Cancel' button in the top right corner. The window contains a table with columns 'Name' and 'Description'. The 'payload' parameter is highlighted, with a red asterisk and the word 'required' next to it. The 'payload' is an object (body) with a value of a JSON object: { "id": 0, "name": "string", "enabled": false }. Below the payload field is a 'Cancel' button and a 'Parameter content type' dropdown menu set to 'application/json'. The 'X-Fields' parameter is a string (header) with a value of 'X-Fields - An optional fields mask'. The 'channel_id' parameter is an integer (path) with a value of '1'.

Name	Description
payload * required object (body)	Edit Value Model <pre>{ "id": 0, "name": "string", "enabled": false }</pre>
X-Fields string(\$mask) (header)	An optional fields mask <input type="text" value="X-Fields - An optional fields mask"/>
channel_id * required integer (path)	<input type="text" value="1"/>

6. Click **[Execute]**.
7. View the updated channel returned in the server response.

Chapter



6

Viewing Publisher Health

Overview

This chapter describes how to view Publisher health using the **health** endpoint.

Use the following menu options to navigate the SL1 user interface:

- To view a pop-out list of menu options, click the menu icon ().
- To view a page containing all of the menu options, click the Advanced menu icon ().

This chapter covers the following topics:

What Does the Health Endpoint Report?	30
Retrieving Liveness Status	30
Retrieving Readiness Status	30

What Does the Health Endpoint Report?

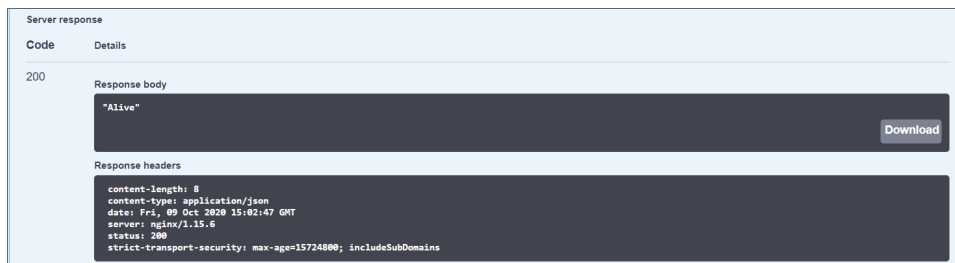
Publisher's health endpoint reports liveness and readiness. The reports are for informational purposes only and do not provide direct interactivity with the Publisher services.

Retrieving Liveness Status

This section describes how to retrieve the Publisher liveness status from the API. The liveness tests ensures that Publisher can receive requests. If the liveness status reports anything other than "Alive" in the server response, contact ScienceLogic Customer Support.

To retrieve Publisher liveness status:

1. Navigate to the API URL. For example: **10.10.10.10/api/v1**.
2. Click on **GET /health/liveness** in the *health* section.
3. Click **[Try it out]**.
4. Click **[Execute]**.
5. View the status contained in the server response.



Retrieving Readiness Status

This section describes how to retrieve Publisher readiness status. The readiness test ensures that Publisher is ready and can reach all necessary services such as MariaDB and Kafka. If the readiness status reports anything other than "Ready" in the server response, contact ScienceLogic Customer Support.

To retrieve a data channel with the channel ID:

1. Navigate to the API URL. For example: **10.10.10.10/api/v1**.
2. Click on **GET /** in the *channels* section.
3. Click **[Try it out]**.
4. Insert the *channel_id* you want to view (for example, 1).
5. Click **[Execute]**.

6. View the channel information returned in the server response.

Server response

Code	Details
200	<p>Response body</p> <pre>{ "results": { "id": 1, "name": "availability", "enabled": true } }</pre> Download
	<p>Response headers</p> <pre>content-length: 58 content-type: application/json date: Thu, 08 Oct 2020 16:34:08 GMT server: nginx/1.15.6 status: 200 strict-transport-security: max-age=15724800; includeSubDomains</pre>

Chapter



7

sl-pubsub Library

Overview

This chapter describes the sl-pubsub library. After you have created subscriptions and Publisher is either reporting the messages being sent and Kafka is receiving the binary data bundles, you can use the sl-pubsub library to unpack the data bundles and receive useful messages from them.

Use the following menu options to navigate the SL1 user interface:

- To view a pop-out list of menu options, click the menu icon ().
- To view a page containing all of the menu options, click the Advanced menu icon ().

This chapter covers the following topics:

Installing the sl-pubsub Library	33
Accessing the sl-pubsub Library Documentation	33

Installing the sl-pubsub Library

To install the sl-pubsub library on your third-party system:

1. Log on to the ScienceLogic Customer Portal and download the *sl-pubsub_whl_and_docs.zip* file, which contains the sl-pubsub library, as well as the sl-pubsub library documentation.
2. Copy the compressed file to the third-party system where you will unpack the binary data bundles sent by Publisher.
3. At the command line, enter the following command:

```
pip3 install sl_pubsub
```

If you have *virtualenvwrapper* installed, enter the following command instead:

```
mkvirtualenv sl_pubsub  
(sl_pubsub) pip3 install sl_pubsub
```

4. The installation process will complete, and the license will be displayed.

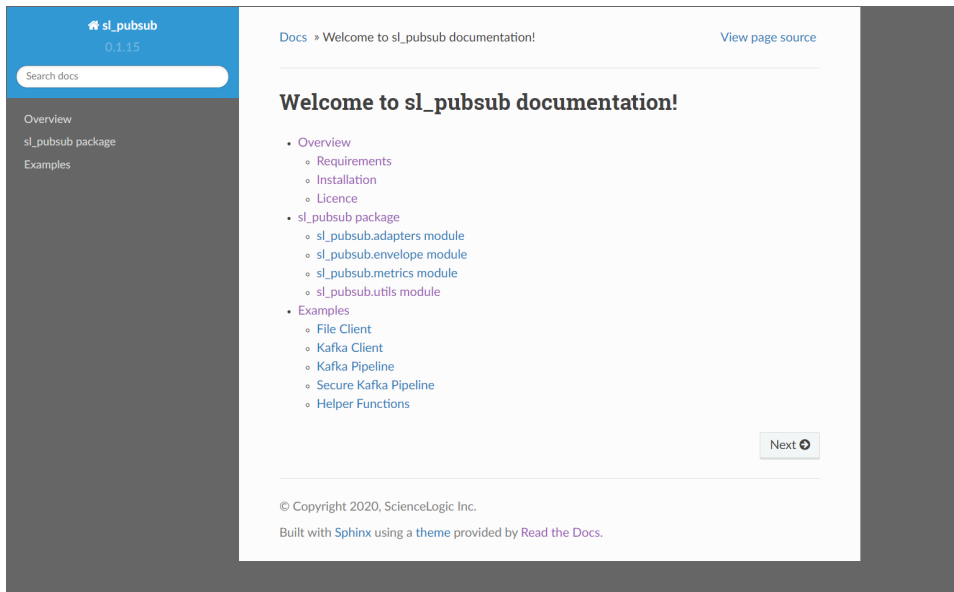
Accessing the sl-pubsub Library Documentation

When you install the sl-pubsub library, the sl-pubsub_doc directory will install in the location where you extract the files.

To view the sl-pubsub documentation:

1. Navigate to the sl-pubsub_doc directory.
2. Open the html sub-directory.

3. Open the index.html file. The documentation will open in your default browser.



The screenshot shows a web browser displaying the documentation for the `sl_pubsub` package. The page has a dark blue header with the package name and version (0.1.15). A search bar is located below the header. The main content area is white and features a navigation sidebar on the left with links for Overview, sl_pubsub package, and Examples. The main content area displays a welcome message and a list of links to various sections of the documentation, including Overview, Requirements, Installation, Licence, and several modules and examples. A 'Next' button is visible at the bottom right of the main content area. The footer contains copyright information for ScienceLogic Inc. and mentions the use of Sphinx and Read the Docs.

sl_pubsub
0.1.15

Search docs

Overview
sl_pubsub package
Examples

Docs » Welcome to sl_pubsub documentation! [View page source](#)

Welcome to sl_pubsub documentation!

- Overview
 - Requirements
 - Installation
 - Licence
- sl_pubsub package
 - sl_pubsub.adapters module
 - sl_pubsub.envelope module
 - sl_pubsub.metrics module
 - sl_pubsub.utils module
- Examples
 - File Client
 - Kafka Client
 - Kafka Pipeline
 - Secure Kafka Pipeline
 - Helper Functions

[Next](#)

© Copyright 2020, ScienceLogic Inc.
Built with [Sphinx](#) using a [theme](#) provided by [Read the Docs](#).

Chapter



8

Troubleshooting Publisher

Overview

This chapter describes some common problems you might encounter while using Publisher, as well as troubleshooting solutions.

Use the following menu options to navigate the SL1 user interface:

- To view a pop-out list of menu options, click the menu icon (.
- To view a page containing all of the menu options, click the Advanced menu icon ().

This chapter covers the following topics:

<i>Why does my Publisher pod keep crashing or restarting?</i>	36
<i>Why is Publisher failing to send data even though it is not crashing?</i>	36
<i>When trying to change the name or ID or a data type, it does not work</i>	37
<i>When navigating to the URL shown in the Kubernetes ingress, a Not Found error occurs</i>	37
<i>When entering a POST or PUT request, a "Browser (or proxy) sent a request that this server could not understand" error occurs</i>	37

Why does my Publisher pod keep crashing or restarting?

Numerous reasons may exist that cause a pod to restart. The following is a short list of possible issues and solutions:

Issue. If an *sl1-mdb* secret, service, or endpoint is not established, Publisher will not work. Publisher looks for each of those specifically to connect to a MariaDB server.

Solution. These items should be created during deployment. Contact ScienceLogic Customer Support for assistance.

Issue. You can connect but get a message saying the table is not created.

Solution. Check the mysql database and ensure that the Publisher database and its corresponding tables exist. If the tables do not exist, contact ScienceLogic Customer Support.

Issue. The address Publisher is trying to reach is not reachable from the pod. This should not cause Publisher to crash or restart, but rather Publisher will be unable to send data.

Solution. The problem could be in network connectivity or in your configuration. Contact ScienceLogic Customer Support for assistance.

Why is Publisher failing to send data even though it is not crashing?

Issue. If you check the logs of the pod and see this message, what has likely happened is that a misconfigured subscription exists, and Publisher is constantly trying to reconnect.

```
::ERROR::sl_pubsub.utils.280::Reached maximum (11) attempts
```

To view logs for a pod:

1. Either go to the console of the Management Node or use SSH to access the Management Node. Open a shell session on the server. Log in with the system password.
2. Run the following command to enter the Docker container:

```
docker-compose -f docker-compose.external.yml run --rm deploy shell
```

3. Enter the following command:

```
kubectl get pods
```

4. Locate the pod for which you want to view logs.
5. Enter the following command for the pod you want to view.

```
kubectl logs -f pod_name
```

6. Review the logs.

Solution. Find the misconfigured subscription using the API, and fix the misconfiguration. Publisher might take some time to pick up the change.

Checking the logs. If you check the pod logs, and you see messages indicating that an exchange is not happening, two likely scenarios could exist.

1. If the queue that is causing the error is a data type queue, such as "avail.publisher" or "interface.publisher", it likely means that the Connector Pipeline is not set up yet, or is not set up as Publisher expects. Investigate the Connector Pipeline and the exchanges it creates.
2. If the queue that is causing the error is "publisher.process", it is likely that something is wrong with the Publisher API, since this message means that the exchange was not created by the API. This could be due to a timing issue, if Publisher started before the API. Investigating the Publisher API in general is recommended.

Issue. Another common error you might see in the logs is a subscription having "topic" or "server" rather than "topics" or "servers". The error in the logs will indicate that "topics" is "None" or "servers" is "None".

Solution. Correct the subscription to use the proper term for topics or servers.

When trying to change the name or ID or a data type, it does not work

Issue. Currently, SL1 does not support changing the name or ID of a data type. The only information you can change is whether the data channel is enabled or disabled.

When navigating to the URL shown in the Kubernetes ingress, a Not Found error occurs

Issue. The URL must include the /api/v1 parameter.

Solution. Go to `https://<ip_address>/api/v1`.

When entering a POST or PUT request, a "Browser (or proxy) sent a request that this server could not understand" error occurs

This issue might also return no errors but likewise produce no results.

Issue. The most likely reason for the issue is that the data set in the PUT request is not formatted correctly.

Solution. The PUT request accepts JSON-formatted data. Check for trailing commas, missing braces, and so on.

© 2003 - 2020, ScienceLogic, Inc.

All rights reserved.

LIMITATION OF LIABILITY AND GENERAL DISCLAIMER

ALL INFORMATION AVAILABLE IN THIS GUIDE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED. SCIENCELOGIC™ AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT.

Although ScienceLogic™ has attempted to provide accurate information on this Site, information on this Site may contain inadvertent technical inaccuracies or typographical errors, and ScienceLogic™ assumes no responsibility for the accuracy of the information. Information may be changed or updated without notice. ScienceLogic™ may also make improvements and / or changes in the products or services described in this Site at any time without notice.

Copyrights and Trademarks

ScienceLogic, the ScienceLogic logo, and EM7 are trademarks of ScienceLogic, Inc. in the United States, other countries, or both.

Below is a list of trademarks and service marks that should be credited to ScienceLogic, Inc. The ® and ™ symbols reflect the trademark registration status in the U.S. Patent and Trademark Office and may not be appropriate for materials to be distributed outside the United States.

- ScienceLogic™
- EM7™ and em7™
- Simplify IT™
- Dynamic Application™
- Relational Infrastructure Management™

The absence of a product or service name, slogan or logo from this list does not constitute a waiver of ScienceLogic's trademark or other intellectual property rights concerning that name, slogan, or logo.

Please note that laws concerning use of trademarks or product names vary by country. Always consult a local attorney for additional guidance.

Other

If any provision of this agreement shall be unlawful, void, or for any reason unenforceable, then that provision shall be deemed severable from this agreement and shall not affect the validity and enforceability of any remaining provisions. This is the entire agreement between the parties relating to the matters contained herein.

In the U.S. and other jurisdictions, trademark owners have a duty to police the use of their marks. Therefore, if you become aware of any improper use of ScienceLogic Trademarks, including infringement or counterfeiting by third parties, report them to Science Logic's legal department immediately. Report as much detail as possible about the misuse, including the name of the party, contact information, and copies or photographs of the potential misuse to: legal@sciencelogic.com



ScienceLogic

800-SCI-LOGIC (1-800-724-5644)

International: +1-703-354-1010