Using SL1 Publisher

SL1 version 10.2.0

Table of Contents

Chapter

1

Introduction to SL1 Publisher

Overview

This chapter describes SL1 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 the menu options, click the Advanced menu icon (⋯).

This chapter covers the following topics:

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 to a third-party destination for long-term data storage, analysis, or reporting. For example, Publisher can send data to Kafka topics.

Publisher supports the concept of *data models*, which specify where you want to retrieve ingested data from, and *subscriptions*, which allow you to specify what data to push and where to send it.

A library of Python functions (sl-schema-registry) is provided in the ScienceLogic Customer Portal for deserializing messages sent to Kafka topics by Publisher.

Publisher is installed and enabled by default in the SL1 Extended Architecture, however you must configure Publisher before it will send your data to a destination. Client software, such as Kafka, must be configured on your own hardware.

How Does Publisher Work?

Publisher ingests data specified by the data models and publishes this data to your Kafka topics as specified in your subscriptions. The Publisher service listens for Custom Resource Definitions (CRDs) that you establish in YAML files, discussed below.

Publisher requires the following two types of YAML files to collect and push data:

- *DataModel*. Specifies where in the ingestion pipelines to extract data from.
- *Subscription*. Specifies which Data Models to publish and the endpoint to which to publish the data.

You must create and apply these files using the examples shown in the next chapter.

If new data is available from the data model, Publisher creates a binary bundle and sends the bundle as specified in the subscription.

When your data is published to your Kafka topics, you can use the sl-schema-registry library to unpack the binary data bundles on your third-party system.

Prerequisites for Using Publisher

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

- Deploy SL1 version 10.2.0 and the SL1 Extended Architecture.
- **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.
- Install `sl-schema-registry` on the system where you will unpack and consume the messages sent to your Kafka topic by Publisher. For more information, see *Installing the sl-schema-registry Library*.

Workflow for Using Publisher

The following steps represent the general workflow for implementing Publisher.

Step	Description	References
1. Enable the Collector Pipeline.	Collector Pipeline is required for Publisher.	*Enabling the Collector Pipeline*
2. Define data models.	Define data models in YAML files.	*Adding Supported Data Models*
3. Define subscriptions.	Define subscriptions in YAML files.	*Adding a Subscription*
4. Install the sl-schema-registry library.	Download the sl-schema-registry from the ScienceLogic Customer Portal and install it on the system where you will consume the messages sent from Publisher. The documentation for the library is contained in the .zip file with the library.	*Installing the sl-schema-registry Library*
5. Use the sl-schema-registry library to unpack the binary data bundles sent from SL1.	See the sl-schema-registry library documentation, included in the .zip file.	—

Enabling the 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 supports Publisher, and the new beta feature, Anomaly Detection.

Currently, Collector Pipeline supports availability data, network 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.

> **NOTE**: SL1 Publisher does not currently support data from Performance Dynamic Applications.

To enable Collector Pipeline for availability data, network 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, network 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
```

NOTE: Enabling `da_perf` has no impact for SL1 Publisher as of SL1 version 10.2.0. Collection from Dynamic Applications is currently unsupported for Publisher.

4. To disable Collector Pipeline for availability data, network 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
```

Chapter

2

Creating the Publisher Custom Resource Definitions

Overview

This chapter describes how to create the Custom Resource Definitions (CRD) for the data models and subscriptions to which Publisher listens during operation. These CRDs are YAML files that you must create and apply before using 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 the menu options, click the Advanced menu icon ( ).

This chapter covers the following topics:

Adding Supported Data Models

To publish data from the data streams, you must create a YAML file for each data stream you want to publish. The supported data models, as well as the format for the YAML files, are covered in this section.

Supported Data Models

Currently, Publisher supports two data streams, specified in a subscription as *dataModels*:

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

Required Fields for Data Model File

The following fields must be specified in your Data Model YAML file:

- `name`. Specifies the unique name of the data model (for example, "availability"). Subscriptions will specify the data model to listen for, by name.
- `address`. Specifies the Kafka topic from which to extract data.
- `config`. This section lets you define Kafka connection options for the source and the sink.
    - `source`. Specifies the configuration for the Kafka topic being consumed. This is the Kafka Broker containing the topic defined by the address field.
    - `sink`. Specifies the configuration for the Kafka topic to which you will publish the data. This is the internal Kafka Broker containing the topics for subscriptions created by Publisher.

Template for Data Model File

To add a Data Model, create one in YAML using the DataModel CRD. This is a template for creating the file:

```
apiVersion: publisher.sl1.io/v1alpha1
kind: DataModel
metadata:
  name: <data model name>
spec:
  address: <kafka "url" to receive schema registry models from. ex: kafka://broker-
address:port/topic_name>
  config:
    sink:  # Any needed Kafka Client config for publishing messages
      <kafka_config_variable>: <config value>
    source:  # Any needed Kafka Client config for consuming messages
      <kafka_config_variable>: <config value>
```

Example

The following example shows a YAML file for publishing availability data:

```
apiVersion: publisher.sl1.io/v1alpha1
```

```
kind: DataModel
metadata:
  name: availability
spec:
  address: kafka://kafka.kafka.svc.cluster.local:9092/avail.data
```

This example shows a YAML file for publishing interface data and adding a configuration when reading from the source topic:

```
apiVersion: publisher.sl1.io/v1alpha1
kind: DataModel
metadata:
  name: interface
spec:
  address: kafka://kafka.kafka.svc.cluster.local:9092/interface.data
  config:
    source:
      retry_backoff_ms: 100
```

Applying the Data Model File

After creating the Data Model YAML file, you must apply it.

To apply the Data Model file:

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, enter the following command, substituting the name of your Data Model file:
   `kubectl apply -f <data_model_file>`

Adding a Subscription

To subscribe to data from the data streams, you must create a Subscription YAML file. You can subscribe to any of the supported data models you have created (see *Adding Supported Data Models*). The format for the YAML file is covered in this section.

Required Fields for Subscription File

The following fields must be specified in your Subscription YAML file:

- `dataModels`. Contains a list of data models from which to retrieve data, by name.

- `address`. Defines the Kafka Broker and Topic to which to publish the data.

- `config`. This section lets you define Kafka connection options for the source and the sink.

  - `source`. Specifies the Kafka Topic being consumed. This is the internal Kafka Broker containing the topic subscriptions created by Publisher.

- ○ `sink`. Specifies the Kafka Topic to which you will publish the data. This is the external Kafka Broker defined by the address field.

Template for Subscription File

To add a Subscription, create one in YAML using the Subscription CRD. This is a template for creating the file:

```
apiVersion: publisher.sl1.io/v1alpha1
kind: Subscription
metadata:
  name: <subscription name>
spec:
  address: <destination; possible values: kafka://broker_address/topic_name >
  dataModels:
    - <datamodels to listen for>
    - <possible are the names of the dataModels>
  config:
    sink:  # Any needed Kafka Client config for publishing messages
      kafka_config_variable: <config value>
    source:  # Any needed Kafka Client config for consuming messages
      kafka_config_variable: <config value>
```

Example

This example shows a YAML file for subscribing to availability and interface data:

```
apiVersion: publisher.sl1.io/v1alpha1
kind: Subscription
metadata:
  name: data-lake
spec:
  address: kafka://192.10.14.37:9092/sl1_sub_topic
  dataModels:
    - availability
    - interface
```

Applying the Subscription File

After creating the Subscription YAML file, you must apply it.

To apply the Subscription file:

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, enter the following command, substituting the name of your Subscription file:
   `kubectl apply -f <subscription_file>`

Configuring an Authenticated Connection for Subscriptions

This section describes how to configure authentication for your subscriptions. Currently, only the Secure Sockets Layer (SSL) authentication method is supported.

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.

Subscription Example

The following is an example of a subscription configured to connect to a Kafka server through an SSL connection:

```
apiVersion: publisher.sl1.io/v1alpha1
kind: Subscription
metadata:
  name: <subscription_name>
spec:
  address: kafka://<kafka_address>:<kafka:port>/<dest_topic>
  dataModels:
    - <desired datamodels>
  config:
    sink:
      security_protocol: "SSL"
      ssl_check_hostname: false
      ssl_cafile: "LS0t…tLS0K"
      ssl_certfile: "LS0t…Qo="
      ssl_keyfile: "Qm…LS0K"
```

The fields `ssl_cafile`, `ssl_certfile`, and `ssl_keyfile` are Base64 encoded strings of the respective files.

For example, if we have a file called `CARoot.pem` that stores our Certificate Authority root certificate, we can convert that into a Base64 encoded string (such as, `LS0t...tLSOK` shown in the example subscription above) and use that Base64 encoded string in the `ssl_cafile` field in the `sink` configuration for our subscription.

> **NOTE:** The `ssl_check_hostname` parameter might need to be set to `true`, depending on how the destination Kafka server and its respective certificates are configured.

Chapter

3

Consuming Messages from a Publisher Subscription

Overview

This chapter describes how to read the messages sent from SL1 Publisher through a subscription to a Kafka topic.

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 the menu options, click the Advanced menu icon ( ).

This chapter covers the following topics:

Configuring Your System to Consume Messages from a Publisher Subscription

After you create a subscription in SL1 Publisher, you will want to verify the messages are being sent. The easiest way to ensure that messages are being sent is to check the logs for the subscription's pods and to verify that they report messages numbering greater than 0.

After you verify that messages are being sent, you can then read the messages from the destination Kafka topic. Two things are required to read the messages:

- A Kafka Python library that allows you to connect and consume content from a Kafka topic

- The sl-schema-registry library provided by ScienceLogic

The messages sent from SL1 Publisher to the Kafka topic are encoded by the sl-schema-registry library, so you need to install this library to unpack and decode the messages.

Installing the sl-schema-registry Library

The sl-schema-registry library is provided as a wheel (`.whl` file) available from the Customer Portal.

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

1. Log on to the ScienceLogic Customer Portal and download the *sl_schema_registry_whl_and_docs.zip* file, which contains the library, as well as the documentation.
2. Copy the compressed file to the third-party system where you will unpack the binary data bundles sent by Publisher.
3. Unzip the compressed file to access the wheel and documentation files.
4. (Option 1) To deploy in a standard environment, enter the following at the command prompt:

   ```
   pip3 install sl_schema_registry-1.5.13-py2.py3-none-any.whl
   ```

4. (Option 2) To deploy in a virtual environment, if you have *virtualenvwrapper* installed, enter the following at the command prompt:

   ```
   mkvirtualenv sl_schema_registry
   (sl_schema_registry) pip3 install sl_schema_registry-1.5.13-py2.py3-none-any.whl
   ```

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

Using a Kafka Python Library to Read Subscription Messages

You can use any Python library that is Python 3.6 or higher to read the messages sent to the Kafka topic. The example below uses the `kafka-python` library found here: https://pypi.org/project/kafka-python/

Example Script

The following is a sample script that you can use to read and print messages from a Kafka topic using the `kafka-python` library and the sl-schema-registry library. This script takes messages from the topic you specify on the Kafka server you specify, converts them to JSON, and prints the resulting JSON messages.

To run this example script, save it to a file (for example, `kafka_consumer.py`), and change the `topic` and `kafka_servers` variables to a valid topic and server in your environment. Run the script as you would run any Python 3 code on that appliance (for example, `python3.6 kafka_consumer.py`).

```
import datetime
import json

from json import JSONEncoder
from contextlib import closing

from kafka import KafkaConsumer
from sl_schema_registry import registry, objectgraph
```

```python
class DateTimeEncoder(JSONEncoder):
    """
    Handles decoding dates and datetimes into a JSON compatible format.
    """
    def default(self, obj):
        if isinstance(obj, (datetime.date, datetime.datetime)):
            return obj.isoformat()
        return None


def get_messages(topic, kafka_servers):
    """
    Generator that yields JSON-formated messages from the a Kafka Topic.
    """
    with closing(KafkaConsumer(topic, bootstrap_servers=kafka_servers)) as consumer:
        for message in consumer:
            env = registry.loads(message.value)
            json_msg = json.dumps(env, cls=DateTimeEncoder)
            yield json_msg


# Establish the Schema Registry to output in Python Dictionaries and Lists, which
are
# the most compatible with JSON.
objectgraph.set_auto_registration_hook(registry)


if __name__ == "__main__":
    topic = "destination-topic" # destination topic where the subscription is pop-
ulating
    kafka_servers = ["kafka:9092"] # kafka server address where the destination
topic can be consumed from

    for json_msg in get_messages(topic, kafka_servers):
        print(json_msg) # place operational logic here
```

Chapter

4

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 the menu options, click the Advanced menu icon ( ).

This chapter covers the following topics:

Why do my Publisher pods keep crashing or restarting?

Numerous reasons may exist that cause a pod to restart. The following is one possible issue and solution:

Issue. If a Publisher service is unable to connect to Kafka, it will not work.

Solution. Ensure connectivity between the service, the internal Kafka server, and the external Kafka service, is possible.

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

Issue. 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 topic that is causing the error is a data type topic, such as "avail.data" or "interface.data", it likely means that the Collector Pipeline is not set up yet, or is not set up as Publisher expects. Investigate the Collector Pipeline, the exchanges it creates, and the data models associated with them.

2. If the topic that is causing the error is "publisher.subscription.name", it is likely that something is incorrect with the Kafka configuration or the Operator. Reviewing the logs of the Operator or the Publisher service is recommended.

Why is my subscription hanging or not being deleted properly?

Issue. When deleting a subscription, sometimes the "kubectl" command will hang and not return, or the subscription object will fail to be deleted.

Solution. Correct the subscription metadata using the Kubernetes command line to address the problem, as follows:

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 patch sub/subscription-name -p '("metadata":{"finalizers":[]}}' --type=merge
   ```