Using the Snippet Framework: Prometheus Toolkit PowerPack

PowerPack version 1.0.0

Chapter 1

Introduction to the Snippet Framework: Prometheus Toolkit PowerPack

Overview

This Snippet Framework: Prometheus Toolkit PowerPack provides the ability to create Dynamic Applications that query the Prometheus server without having to write snippet code. This PowerPack includes sample Dynamic Application that can be easily modified and replicated to retrieve metrics from a Prometheus server through PromQL language.

The Snippet Framework: Prometheus Toolkit PowerPack includes the following features:

- A PromQL execution environment.
- A list of required steps for PromQL Dynamic Application development.
- A template PowerPack implementation that demonstrates the use of the available steps.
- Support for these authentication types: Basic, Bearer Token, API Key, and Oauth2.

> The Prometheus Toolkit does not support Histogram and Summary metric types.

Prerequisites

Before building a Dynamic Application with this PowerPack, you should familiarize yourself with the Snippet Framework material. This material provides the necessary workflows and procedures you will use the PowerPack to complete a Dynamic Application build.

Other areas to be familiar with include:

- How to Build Dynamic Applications with the Snippet Framework in the *Snippet Framework* manual.

What is the Snippet Framework: Prometheus Toolkit PowerPack?

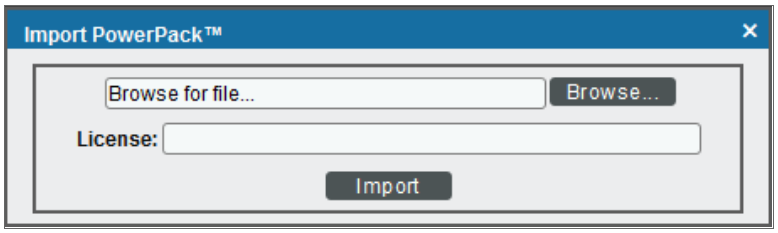
The Snippet Framework: Prometheus Toolkit PowerPack allows users to build Dynamic Applications that monitor Prometheus using PromQL, a functional query language (Prometheus Query Language). PromQL allows users to select and aggregate time series data in real time. The SL1 platform collects data using these Dynamic Applications to monitor, visualize, and forecast information.

Installing the Snippet Framework: Prometheus Toolkit PowerPack

Before you can use the Snippet Framework, you must import and install the latest version of the Snippet Framework: Prometheus Toolkit PowerPack.

To download and install a PowerPack:

1. Download the PowerPack from the ScienceLogic Support Site.
2. Go to the PowerPack Manager page (System > Manage > PowerPacks).
3. In the PowerPack Manager page, click the [Actions] button, then select *Import PowerPack*.
4. The Import PowerPack dialog box appears:



5. Click the [Browse] button and navigate to the PowerPack file.
6. When the PowerPack Installer modal appears, click the [Install] button to install the PowerPack.

NOTE: If you exit the PowerPack Installer without installing the imported PowerPack, the imported PowerPack will not appear in the PowerPack Manager page. However, the imported PowerPack will appear in the Imported PowerPacks page. This page appears when you click the [Actions] menu and select *Install PowerPack*.

Using the Snippet Framework: Prometheus Toolkit PowerPack

To build a successful Dynamic Application with this PowerPack:

- Import and install the Snippet Framework: Prometheus Toolkit PowerPack.
- Determine the correct Authentication Method to use.

- Configure a Credential.
- Define your Dynamic Applications by using the steps available in this Toolkit.

Authentication Types

Authentication is crucial to the collection of data with the Snippet Framework: Prometheus Toolkit PowerPack. The Snippet Framework uses requests to collect data. Meaning, you must ensure that you are using the correct authentication method for data collection. The Snippet Framework: Prometheus Toolkit will not make device data requests until a successful authentication attempt has been made.

This PowerPack supports these types of authentications:

- Basic
- Bearer Token
- API Key
- Oauth2

Troubleshooting

After you build a Snippet Framework Dynamic Application using the steps called out in the Snippet Framework: Prometheus Toolkit, you may want to troubleshoot any Snippet Framework authentication issues that arise.

To troubleshoot a built Dynamic Application:

1. Click [Manage > Credentials] to access the Credentials page.
2. Assure that your authentication type is selected from the [Authentication Type] drop-down menu.
3. Set Logging Debug as [True]. The debug option displays information from the SL1 collector to the authentication endpoint.
4. Click[[Save & Test]].
5. Click[[Devices]] in the side page menu to navigate to the Devices page.
6. Select your Device from the Devices list and click the [[Collections]] tab.
7. Click the [Ellipses (…)] button to the right of your Dynamic Application and select [[Run Now]].

Known Issues

Some fields may display incorrectly when a user changes a previously selected [Authentication Type] to a new type in the Snippet Framework: Prometheus Toolkit credential. To correct these fields, you must close out the in-progress configuration of the credential without saving. Then, re-open and select the desired [Authentication Type].

Chapter 2

Available Steps

Available Steps

The Prometheus Library can be used as a set of available steps. These available steps, when followed completely, enable low-code development capabilities for Prometheus PowerPacks. Without the need for Python development, the building time for Snippet Framework: Prometheus Toolkit PowerPacks will decrease.

Low Code Syntax

The low_code Argument Parser is a YAML configuration format for specifying your collections. You explicitly provide the steps you want to run, in the order you need them to be executed. There are multiple versions of the low_code Syntax.

Framework Name `low_code`

All versions support specifying configuration. The configuration will be all sub-elements under the step.

If you had a step cool_step and wanted to specify two key values, you would provide the following examples.

Example Usage

Snippet Argument

```
low_code:
  id: eastwood1
  version: 2
  steps:
    - cool_step
      key1: value1
```

```
        key2: value2
```

To provide a list in YAML, you will need to utilize the – symbol.

> NOTE: Notice that, in the example above, `key1` and `key2` are indented one level beyond `cool_step`. Setting them at the same indentation level as `cool_step` will result in an error.

Snippet Argument

For example, if you had a step cool_step and wanted to specify a list of two elements, you would provide the following:

```
low_code:
  id: eastwood1
  version: 2
  steps:
     - cool_step
       key1
       key2
```

Low Code Syntax: Version 1

Version 1 was the original low_code syntax. It allowed for a single Requestor and any number of processors. It lacks support for multiple Requestors so it is not preferred.

Snippet Argument

```
low_code:
  id: my_request
  network:
       static_value:'{"key": "value"}'
  processing:
       - json
       - simple_key: "key"
```

Terms

id: Identification for the request.

version: Specify the version of the low_code Syntax. If not provided, it will default to 1.

network: Section for the data requester step.

processing: Section for listing the steps required to transform your data to the desired output for SL1 to store.

NOTE: If the `id` value is not specified, the Snippet Framework will automatically create one. This allows for easier tracking when debugging when an ID is not required for the collection.

Low Code Syntax: Version 2

Version 2 of the low_code Syntax provides more flexibility when defining the order of step execution. This version can utilize multiple versions of Requestors (if supported) and allows for steps to run before a Requestor executes.

Snippet Argument

```
low_code:
  id: eastwood1
  version: 2
  steps:
     - static_value:'{"key": "value"}'
     - json
     - simple_key: "key"
```

Terms

id: Identification for the request.

version: Specify the version of the low_code Syntax.

steps: Order of the steps for the Snippet Framework to execute.

NOTE: If the `id` value is not specified, the Snippet Framework will automatically create one. This allows for easier tracking when debugging when an ID is not required for the collection.

PromQL

The PromQL syntax is a YAML configuration format for specifying the data to collect from Prometheus. You explicitly provide a query and the labels that will be taken as indices.

Syntax Name `promql`

PromQL Format

Snippet Argument

```
promql:
  id: RequestID1
  query: prometheus_query
```

```
labels:
   - label1
   - label2
```

You can specify which labels you want to use as indices by using the key `labels`.

Terms

id: Identification for the request.

query: Prometheus query.

labes: Order of the labels you would like to get as indices.

> NOTE: If the labels key is not provided, all the labels will be retrieved as you would get them in the Prometheus expression browser. In the case of providing labels that do not define the uniqueness of an index to identify a value, only the first retrieved value will be displayed and a log message will report the collision.

PromQL Examples

Example 1

Input Parameters

If we want to collect the Kafka Exporter metrics for the number of in-sync replicas for a topic partition, the query should be: `kafka_topic_partition_in_sync_replica`

Incoming Data

The response is:

```
{
  'kafka_topic_partition_in_sync_replica{instance="kafka-service-
  metrics.default.svc:9308", job="kubernetes-services", kubernetes_
  namespace="default", partition="0", port_name="http-metrics", service_
  name="kafka-service-metrics", topic="__consumer_offsets"}': '3',
  'kafka_topic_partition_in_sync_replica{instance="kafka-service-
  metrics.default.svc:9308", job="kubernetes-services", kubernetes_
  namespace="default", partition="0", port_name="http-metrics", service_
  name="kafka-service-metrics", topic="apl.prod.agent.avail.trigger"}':
  '3',
  'kafka_topic_partition_in_sync_replica{instance="kafka-service-
  metrics.default.svc:9308", job="kubernetes-services", kubernetes_
  namespace="default", partition="0", port_name="http-metrics", service_
  name="kafka-service-metrics", topic="apl.prod.app.agg.trigger"}': '3'
  ...
}
```

Snippet Argument

The Snippet Argument will be this:

```
promql:
  id: InSyncReplicas
  query: kafka_topic_partition_in_sync_replica
```

The Snippet Argument (if we want to index by topic):

```
promql:
  id: InSyncReplicas
  query: kafka_topic_partition_in_sync_replica
  labels:
      - topic
```

Data Collected

Data collected:

```
  {
    '{topic="__consumer_offsets"}': '3',
    '{topic="apl.prod.agent.avail.trigger"}': '3',
    '{topic="apl.prod.app.agg.trigger"}': '3'
    ...
  }
```

Promql syntax takes the query, puts it in the `http` step as param, and sends it to the Prometheus server as a REST API request. Then, it takes the response and parses it using the `json` step. Finally, it takes the parsed response and indexes it by the labels using the `promql_vecto`r step.

Requestors

HTTP

The HTTP Data Requester provides HTTP request functionality to gather information for the framework. The full URI is built from the credential and argument of the requester. The credential contains the base URL and the port.

Static Value

The Static Value Data Requester is used to mock network responses from a device for testing purposes or when a step needs a static name.

> TIP: To see step and example details of these requestors, see the *Requestors* section in the Snippet Framework: REST Toolkit PowerPack manual.

Processors: Data Parsers

CSV

The CSV Data Parser converts a string into the format requested by the Args.

JSON

The JSON Data Parser converts a JSON string into a python dictionary.

Regex

The Regex step uses regular expressions to extract the required information and returns a dictionary.

String Float

The String Float parses a string to float. If the string contains multiple floats, it returns a list of them.

TIP: To see step and example details of these data parsers, see the *Data Parsers*section in the Snippet Framework: REST Toolkit PowerPack manual.

Selectors

JMESPath

JMESPath is a query language for JSON data. The JMESPath step can be used on data after being JSON parsed. It uses a path expression as a parameter to specify the location of the desired element (or a set of elements). Paths use the dot notation.

The JMESPath step accepts one path expression. Additionally, the JMESPath step provides a capability to build custom indexable output. This requires the expression path to be a multiselect hash with defined index and value keys.

JSONPath

The JSONPath is a query language for JSON data. The JSONPath selector can be used on any data once it has been parsed. It uses path expressions as parameters to specify a path to the element (or a set of elements). Paths use the dot notation.

The JSONPath parser can accept one or two paths. If one is given, that is the path to the data. If two are given, that provides the path to the index and the path to the data.

promql_vector

The PromQL Vector selector processes responses in a similar format as the ones provided by the expression browser at a Prometheus server. This step returns dictionaries where the keys are built by the labels. It also allows you to only show the labels of interest by providing a list of labels as part of the arguments.

Framework Name	`promql_vector`
Parameters	labels - list of labels to retrieve

promql_vector Examples

Example 1

Incoming Data

```
{
  "data": {
    "result": [
           {
             "metric": {"consumergroup": "AIML_anomaly_detection.alert"},
             "value": [1658874518.797, "0"],
           },
           {
             "metric": {"consumergroup": "AIML_anomaly_
    detection.autoselector"},
             "value": [1658874518.797, "0"],
           },
           {
             "metric": {"consumergroup": "AIML_anomaly_
    detection.storage"},
             "value": [1658874518.797, "3"],
           },
           {
             "metric": {"consumergroup": "AIML_anomaly_detection.train"},
             "value": [1658874518.797, "1"],
           },
           {
             "metric": {"consumergroup": "sl_event_storage"},
             "value": [1658874518.797, "0"]
           },
    ],
    "resultType": "vector",
  },
  "status": "success",
  }
```

Input Parameters

If we wanted to provide these input parameters: `labels: ["consumergroup"]`

Output

The output of this step will be:

```
  {
    '{consumergroup="AIML_anomaly_detection.alert"}':"0",
    '{consumergroup="AIML_anomaly_detection.autoselector"}':"0",
    '{consumergroup="AIML_anomaly_detection.storage"}':"3",
    '{consumergroup="AIML_anomaly_detection.train"}':"1",
    '{consumergroup="sl_event_storage"}': "0",

  }
```

Snippet Argument

The Snippet Argument of this step will be:

```
promql:
  id: my_request
  query: <query>
  labels:
     - consumer group
```

Example 2

Incoming Data

If the incoming data to the step is:

```
  {
    "data": {
      "result": [
        {
          "metric": {"consumergroup": "AIML_anomaly_detection.alert"},
          "value": [1658874518.797, "0"],
        },
        {
          "metric": {"consumergroup": "AIML_anomaly_
          detection.autoselector"},
          "value": [1658874518.797, "0"],
        },
        {
          "metric": {"consumergroup": "AIML_anomaly_detection.storage"},
          "value": [1658874518.797, "3"],
        },
        {
          "metric": {"consumergroup": "AIML_anomaly_detection.train"},
          "value": [1658874518.797, "1"],
        },
        {
          "metric": {"consumergroup": "sl_event_storage"},
          "value": [1658874518.797, "0"]
        },
      ],
      "resultType": "vector",
    },
    "status": "success",

  }
```

Output

The output of this step will be:

```
  {
    '{consumergroup="AIML_anomaly_detection.alert"}':"0",
    '{consumergroup="AIML_anomaly_detection.autoselector"}':"0",
    '{consumergroup="AIML_anomaly_detection.storage"}':"3",
    '{consumergroup="AIML_anomaly_detection.train"}':"1",
    '{consumergroup="sl_event_storage"}': "0",
  }
```

Snippet Argument

The Snippet Argument should be:

```
promql:
  id: my_request
  query: <query>
```

Example 3

Incoming Data

If the incoming data to the step is:

```
  {
    "data": {
      "result": [
        {
          "metric": {
            "__name__": "kafka_topic_partition_in_sync_replica",
            "instance": "kafka-service-metrics.default.svc:9308",
            "job": "kubernetes-services",
            "kubernetes_namespace": "default",
            "partition": "0",
            "port_name": "http-metrics",
            "service_name": "kafka-service-metrics",
            "topic": "__consumer_offsets",
          }
          "value": [1658944573.158, "3"],
        },
        {
          "metric": {
            "__name__": "kafka_topic_partition_in_sync_replica",
            "instance": "kafka-service-metrics.default.svc:9308",
            "job": "kubernetes-services",
            "kubernetes_namespace": "default",
            "partition": "0",
            "port_name": "http-metrics",
            "service_name": "kafka-service-metrics",
            "topic": "apl.prod.app.agg.trigger",
          }
          "value": [1658944573.158, "3"],
        },
        {
          "metric":
            "__name__": "kafka_topic_partition_in_sync_replica",
            "instance": "kafka-service-metrics.default.svc:9308",
            "job": "kubernetes-services",
            "kubernetes_namespace": "default",
            "partition": "9",
            "port_name": "http-metrics",
```

```
            "service_name": "kafka-service-metrics",
            "topic": "swap.data",
          }
          "value": [1658944573.158, "3"]
        },
      ],
      "resultType": "vector",
    },
    "status": "success",
  }
```

Input Parameters

If we wanted to provide these input parameters: `labels: ["service_name", "topic"]`

Output

The output of this step would be:

```
  {
    '{service_name="kafka-service-metrics", topic="__consumer_offsets"}':
    "3",
    '{service_name="kafka-service-metrics",
    topic="apl.prod.app.agg.trigger"}': "3",
    '{service_name="kafka-service-metrics", topic="swap.data"}': "3",
  }
```

Snippet Argument

The Snippet Argument should be:

```
promql:
  id: my_request
  query: <query>
  labels:
     - service_name
     - topic
```

Example 4

Incoming Data

If the incoming data to the step is:

```
  {
    "data": {
      "result": [
        {
          "metric": {
            "__name__": "kafka_topic_partition_in_sync_replica",
            "instance": "kafka-service-metrics.default.svc:9308",
            "job": "kubernetes-services",
            "kubernetes_namespace": "default",
```

```
            "partition": "0",
            "port_name": "http-metrics",
            "service_name": "kafka-service-metrics",
            "topic": "__consumer_offsets",
          }
          "value": [1658944573.158, "3"],
        },
        {
          "metric": {
            "__name__": "kafka_topic_partition_in_sync_replica",
            "instance": "kafka-service-metrics.default.svc:9308",
            "job": "kubernetes-services",
            "kubernetes_namespace": "default",
            "partition": "0",
            "port_name": "http-metrics",
            "service_name": "kafka-service-metrics",
            "topic": "apl.prod.app.agg.trigger",
          }
          "value": [1658944573.158, "3"],
        },
        {
          "metric":
            "__name__": "kafka_topic_partition_in_sync_replica",
            "instance": "kafka-service-metrics.default.svc:9308",
            "job": "kubernetes-services",
            "kubernetes_namespace": "default",
            "partition": "9",
            "port_name": "http-metrics",
            "service_name": "kafka-service-metrics",
            "topic": "swap.data",
          }
          "value": [1658944573.158, "3"]
        },
      ],
      "resultType": "vector",
    },
    "status": "success",
  }
```


Output

The output of this step will be:


```
{
  'kafka_topic_partition_in_sync_replica{instance="kafka-service-
  metrics.default.svc:9308", job="kubernetes-services", kubernetes_
  namespace="default", partition="0", port_name="http-metrics", service_
  name="kafka-service-metrics", topic="__consumer_offsets"}': "3",
  'kafka_topic_partition_in_sync_replica{instance="kafka-service-
  metrics.default.svc:9308", job="kubernetes-services", kubernetes_
  namespace="default", partition="0", port_name="http-metrics", service_
  name="kafka-service-metrics", topic="apl.prod.app.agg.trigger"}': "3",
  'kafka_topic_partition_in_sync_replica{instance="kafka-service-
  metrics.default.svc:9308", job="kubernetes-services", kubernetes_
  namespace="default", partition="9", port_name="http-metrics", service_
  name="kafka-service-metrics", topic="swap.data"}': "3",
}
```

Snippet Argument

The Snippet Argument should be:

```
promql:
  id: my_request
  query: <query>
```

Example 5

Incoming Data

If the incoming data to the step is:

```
  {
    "data": {'result': [{'metric': {},
                         'value': [1659022840.388, '5.100745223340136']}],
             'resultType': 'vector'},
    'status': 'success'}
  }
```

Output

The output of this step will be:

```
  {"{}":"5.100745223340136"}
```

Snippet Argument

The Snippet Argument should be:

```
promql:
  id: my_request
  query: <query>
```

Example 6

Incoming Data

If the incoming data to the step is:

```
  {
    "data": {
      "result": [
        {
          "metric": {
            "__name__": "kafka_topic_partition_in_sync_replica",
            "instance": "kafka-service-metrics.default.svc:9308",
            "job": "kubernetes-services",
            "kubernetes_namespace": "default",
            "partition": "0",
            "port_name": "http-metrics",
            "service_name": "kafka-service-metrics",
            "topic": "__consumer_offsets",
          }
```

```
          "value": [1658944573.158, "1"],
        },
        {
          "metric": {
            "__name__": "kafka_topic_partition_in_sync_replica",
            "instance": "kafka-service-metrics.default.svc:9308",
            "job": "kubernetes-services",
            "kubernetes_namespace": "default",
            "partition": "0",
            "port_name": "http-metrics",
            "service_name": "kafka-service-metrics",
            "topic": "apl.prod.app.agg.trigger",
          }
          "value": [1658944573.158, "3"],
        },
        {
          "metric":
            "__name__": "kafka_topic_partition_in_sync_replica",
            "instance": "kafka-service-metrics.default.svc:9308",
            "job": "kubernetes-services",
            "kubernetes_namespace": "default",
            "partition": "9",
            "port_name": "http-metrics",
            "service_name": "kafka-service-metrics",
            "topic": "swap.data",
          }
          "value": [1658944573.158, "3"]
        },
      ],
      "resultType": "vector",
    },
    "status": "success",

  }
```


Input Parameters

If we wanted to provide these input parameters: `labels: ["port_name"]`

Output

The output of the step will be:


```
  {
    {port_name="http-metrics"}': "1",

  }
```


Snippet Argument

The Snippet Argument should be:

```
promql:
  id: my_request
  query: <query>
  labels:
     - port_name
```

NOTE: In the example above, the label `port_name` has the same value (http-metrics) for all elements. However, only the first entry will be returned. That is emphasized with a info log message like this: `The following labels were duplicated: {port_name="http-metrics"}. Only the first entry is being displayed.`

Simple Key

The Simple Key selector is for dictionaries, lists, etc. It returns the specified key by using periods as separators.

TIP: To see step and example details of these selectors, see the *Selectors*section in the Snippet Framework: REST Toolkit PowerPack manual.

Cachers

cache_writer

The cache_writer step enables user defined caching (read and write) to SL1's database instance.

Step details:

Framework Name	`cache_write`
Parameters	• key: Metadata key that the DB will use for cache R/W(default: request_id) • reuse_for:Time in minutes that specifies valid cache entry duration times to be Read. (default: 5) • cleanup_after:Time in minutes that specifies when caches will expire and should be removed from the DB (default: 15)

NOTE: reuse_for is defined as 0 minutes, the cache_writer will not allow a fast-forward in the pipeline execution.

NOTE: cleanup_after is defined to be smaller than reuse_for, the cache_writer will fail to find valid data and run through the step execution up to the point of cache_writer.

Example Usage

The following is an example network request, process the data (json->dict) and then select a subset of that data.

Snippet Argument

The Snippet Argument should look like this:

```
low_code:
  id: my_request
  version: 2
  steps:
    - <network_request>
    - json
    - simple_key: "id"
```

Let's assume that the network request and json processing are steps that we want to cache and possibly reuse in another collection and/or dynamic application.

The following code uses a custom key, with a cache reuse time, reuse_for, of 5 minutes and a clean up, cleanup_after, on cache entries after 15 minutes.

Snippet Argument

```
low_code:
  id: my_request
  version: 2
  steps:
    - <network_request>
    - json
    - cache_writer: {"key": "here", "reuse_for": 5, "cleanup_after": 15}
    - simple_key: "id"
```

If there is a cache entry that is 5 minutes or newer since the start of the collection cycle the step will read the cached value and fast forward to the simple_key step.

Chapter 3

Prometheus Development Best Practices

Best Practices

There are a series of developer best practices recommended to ensure the highest efficiency.

Data collection, snippet capability, connection, and code simplicity are some of the areas that benefit from the following Development Best Practices:

- Prometheus supports many binary operators (like arithmetic, trigonometric, comparison, etc) and aggregation operations (like sum, min, max, avg, etc). Before defining a custom step to perform these operations with the collected data, consider using the operators as part of your query.
- Prometheus supports several functions to operate on data. Before you define a custom step to do something similar, consider using the function as part of your query.
- The Snippet Framework truncates indices if they are longer than 512 characters. As a result of this truncation, a new index will be generated, with the first original 50 characters of the index followed by a unique identifier. If possible, try to reduce the index by using the key labels of the PromQL syntax.

Design Checklist for Prometheus Dynamic Applications

Before you start to write your Dynamic Applications, think through your process.

1. Model
    - What does the data that you want to collect look like?
2. Collection Objects
    - What data do you want to collect?

3. Query

    - What metrics do you need to collect?
    - Ensure to use appropriate operators and functions to get desired data.

4. Labels

    - If you find an opportunity to reduce the labels, use the key labels to select only the labels of interest.

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

International: +1-703-354-1010