PrometheusInterface

v. 0.8.0

Holger Queckenstedt

21.10.2024

Contents

Chapter 1

Introduction

The interface library **PrometheusInterface** provides **Robot Framework** keywords to communicate with the monitoring system **Prometheus**. With the help of this interface it is possible to use **Prometheus** to monitor data provided by **Robot Framework** tests.

T.B.C.

Chapter 2

Description

2.1 Test setup

How many tests are passed? How many tests are failed? On which test benches do these tests run? Did any resets occur during test execution? How about the temporary unavailability of required test external components?

To monitor all these informations, a test system setup is necessary consisting of at least the following components:

1. A test framework that executes the test and provides the data to be monitored (here: the **Robot Framework**). This includes the possibility that more than one test framework runs in parallel.
2. A component that collects and stores data from all executed test frameworks (here: the monitoring system **Prometheus**).
3. A component that visualizes all data that have been collected (here: **Grafana**).

This is not the only way to establish such a test system, but in this document we concentrate on **Robot Framework**, **Prometheus** and **Grafana**.

To be able to collect values, **Prometheus** requires an http based counterpart. And the **Robot Framework** must be enabled to support this counterpart. In pure Python this is realized by a **Prometheus Python client library**. On **Robot Framework** level this is done by this component **PrometheusInterface** that is a mapping between the interface of the **Prometheus Python client library** and **Robot Framework** keywords.

Or in other words: **PrometheusInterface** uses the **Prometheus Python client library** to provide values to **Prometheus** to enable the data visualization in **Grafana**.

The **Prometheus Python client library** is part of the installation dependencies of **PrometheusInterface**. **Prometheus** and **Grafana** are components that have to be downloaded, installed and configured separately.

2.2 Installations

1. Prometheus

To install **Prometheus** please visit the homepage. This homepage also contains a *getting started* section containing useful hints about how to configure the application.

Further informations can also be found here.

2. Grafana

The advantage of using **Grafana** for data visualization is to have a *ready to use* interface to **Prometheus** available. Other possible solutions are not in focus here.

To install **Grafana** please visit the homepage.

How to create a Prometheus data source in Grafana, is described here.

3. Prometheus Python client library

This library belongs to the installation dependencies of **PrometheusInterface**. A separate installation is not required. In case you want to learn more about this client library please visit the following web pages: [1], [2], [3].

2.3 Configuration

Prometheus can be configured with a configuration file in YAML format. This includes the used port numbers. The example files in the **robotframework-prometheus** repository use the port numbers 8000, 8001 and 8002. Therefore the configuration file of **Prometheus** requires the following entry:

```
- targets: ["localhost:8000","localhost:8001","localhost:8002"]
```

2.4 Library import

After installation (see README), the interface library can be found within the `site-packages` that is the usual place for installed Python modules. It is recommended to introduce an environment variable to store the `site-packages` path, e.g. `ROBOTPYTHONSITEPACKAGESPATH` .

Now within robot files the interface library can be imported in the following way:

```
*** Settings ***
Library    %{ROBOTPYTHONSITEPACKAGESPATH}/PrometheusInterface/prometheus_interface.py
```

If nothing else is specified, the default port 8000 is used. It can be required to run several **Robot Framework** instances in parallel. In this case every instance needs it's own port number. Within the **Prometheus** YAML file we already have three port numbers defined (like described above).

Let's assume now we want to execute testsuite *A* parallel to testsuite *B*. Then we can assign port number 8001 to testsuite *A*:

```
*** Settings ***
Library    %{ROBOTPYTHONSITEPACKAGESPATH}/PrometheusInterface/prometheus_interface.py    ↩
    ↪ port_number=${8001}
```

and we assign port number 8002 to testsuite *B*:

```
*** Settings ***
Library    %{ROBOTPYTHONSITEPACKAGESPATH}/PrometheusInterface/prometheus_interface.py    ↩
    ↪ port_number=${8002}
```

Every testsuite needs it's own robot or resoure file in which this import happens!

To support a better readibility of the test code we recommend to import the interface library with a certain name:

```
*** Settings ***
Library    %{ROBOTPYTHONSITEPACKAGESPATH}/PrometheusInterface/prometheus_interface.py    ↩
    ↪ port_number=${8001}    WITH NAME    rf.prometheus_interface
```

With `rf` is the abbreviation of **Robot Framework**.

2.5 Support of Prometheus metric types

2.5.1 Counters and gauges

The difference between a counter and a gauge is: A counter can be incremented only, whereas a gauge can be incremented, decremented and set to a certain value explicitly. Both counters and gauges have to be added before.

A counter is added in this way:

```
rf.prometheus_interface.add_counter    name=(name of counter)    description=(description of counter)    labels=(label names)
```

`name` and `decription` are required, `labels` is optional.

Example:

We want to count *passed*, *failed* and *unknown* tests. A posssible definition of counter can look like this:

```
rf.prometheus_interface.add_counter    name=num_passed    description=number of passed tests    labels=room;testbench;testname
rf.prometheus_interface.add_counter    name=num_failed    description=number of failed tests    labels=room;testbench;testname
rf.prometheus_interface.add_counter    name=num_unknown    description=number of unknown tests    labels=room;testbench;testname
```

In this example we assume that several different tests are executed in several rooms and on several testbenches. It is not necessary to add a separate counter for every test on every testbench in every room. Only one counter (counting *passed* tests) is required, but with several labels. Every label will be a separate series of measurements of the corresponding counter. Every label is also a filter criteria in **Grafana** when configuring metrics.

During the lifetime of a testsuite a counter can be added only once. Therefore we suggest to place the adding into an `__init__.robot` file.

A counter can be incremented (by default value 1) in this way:

```
rf.prometheus_interface.inc_counter    name=(name)    labels=(label values)
```

A counter can also be incremented by a user defined value:

```
rf.prometheus_interface.inc_counter    name=(name)    value=(user defined increment)    labels=(label values)
```

In `add_counter` , `labels` is a semicolon separated list of label *names*. In `inc_counter` , `labels` is a semicolon separated list of label *values*. Label names and label values must fit together in `add_counter` and `inc_counter` .

Example:

```
rf.prometheus_interface.inc_counter    name=num_passed    labels=Room_1;Testbench 2;Suite-A-Test-01
```

The same with gauges. A gauge is added in this way:

```
rf.prometheus_interface.add_gauge    name=(name of gauge)    description=(description of gauge)    labels=(label names)
```

`name` and `decription` are required, `labels` is optional.

A gauge can e.g. be set to a certain value

```
rf.prometheus_interface.set_gauge    name=(name of gauge)    value=(value of gauge)    labels=(label values)
```

As with counters, `labels` is a semicolon separated list of names or values.

Chapter 3

prometheus_interface.py

3.1 Class: prometheus_interface

Imported by:

```
from PrometheusInterface.prometheus_interface import prometheus_interface
```

The class 'prometheus_interface' provides to communicate with the monitoring system Prometheus. For this purpose the 'Prometheus Python client library' is used.

3.1.1 Method: convert_to_int_or_float

Little helper to convert a string value to an integer or a float

3.1.2 Keyword: get_version

Returns the version of this interface library

3.1.3 Keyword: who_am_i

Returns the full name of this interface library

3.1.4 Keyword: where_am_i

Returns path to this interface library

3.1.5 Keyword: get_port_number

Returns the port number assigned to this instance of the library

3.1.6 Keyword: add_info

This keyword adds a new info. The content of an existing info can be defined with `set_info`.

Arguments:

- `name`
  The name of the new info
  / *Condition*: required / *Type*: str /
- `description`
  The description of the new info
  / *Condition*: required / *Type*: str /

- `labels`
  A semicolon separated list of label names assigned to the new info
  / *Condition*: optional / *Type*: str / *Default*: None /

Returns:

- `success`
  / *Type*: bool /
  Indicates if the computation of the keyword was successful or not
- `result`
  / *Type*: str /
  The result of the computation of the keyword

3.1.7 Keyword: set_info

This keyword defines the content of an info. The info has to be added with '`add_info`' before.

Arguments:

- `name`
  The name of the info
  / *Condition*: required / *Type*: str /
- `info`
  The info itself (every info is a key-value information).
  / *Condition*: required / *Type*: dict /
- `labels`
  A semicolon separated list of labels assigned to the info. The order of labels must fit to the order of label names like defined in `add_info`.
  / *Condition*: optional / *Type*: str / *Default*: None /

Returns:

- `success`
  / *Type*: bool /
  Indicates if the computation of the keyword was successful or not
- `result`
  / *Type*: str /
  The result of the computation of the keyword

3.1.8 Keyword: add_counter

This keyword adds a new counter. The values of existing counters can be changed with `inc_counter`.

Arguments:

- `name`
  The name of the new counter
  / *Condition*: required / *Type*: str /
- `description`
  The description of the new counter
  / *Condition*: required / *Type*: str /

- `labels`
  A semicolon separated list of label names assigned to the new counter
  / *Condition*: optional / *Type*: str / *Default*: None /

Returns:

- `success`
  / *Type*: bool /
  Indicates if the computation of the keyword was successful or not
- `result`
  / *Type*: str /
  The result of the computation of the keyword

3.1.9 Keyword: inc_counter

This keyword increments a counter. The counter has to be added with '`add_counter`' before.

Arguments:

- `name`
  The name of the counter
  / *Condition*: required / *Type*: str /
- `value`
  The value of increment. If not given, the value of the counter is incremented by value 1.
  / *Condition*: optional / *Type*: int / *Default*: None /
- `labels`
  A semicolon separated list of labels assigned to the counter. The order of labels must fit to the order of label names like defined in `add_counter`.
  / *Condition*: optional / *Type*: str / *Default*: None /

Returns:

- `success`
  / *Type*: bool /
  Indicates if the computation of the keyword was successful or not
- `result`
  / *Type*: str /
  The result of the computation of the keyword

3.1.10 Keyword: add_gauge

This keyword adds a new gauge. The values of existing gauges can be changed with `set_gauge`, `inc_gauge` and `dec_gauge`.

Arguments:

- `name`
  The name of the new gauge
  / *Condition*: required / *Type*: str /
- `description`
  The description of the new gauge
  / *Condition*: required / *Type*: str /

- `labels`
  A semicolon separated list of label names assigned to the new gauge
  / *Condition*: optional / *Type*: str / *Default*: None /

Returns:

- `success`
  / *Type*: bool /
  Indicates if the computation of the keyword was successful or not
- `result`
  / *Type*: str /
  The result of the computation of the keyword

3.1.11 Keyword: set_gauge

This keyword sets the value for a gauge. The gauge has to be added with '`add_gauge`' before.

Arguments:

- `name`
  The name of the gauge
  / *Condition*: required / *Type*: str /
- `value`
  The new value of the gauge.
  / *Condition*: optional / *Type*: int / *Default*: None /
- `labels`
  A semicolon separated list of labels assigned to the gauge. The order of labels must fit to the order of label names like defined in `add_gauge`.
  / *Condition*: optional / *Type*: str / *Default*: None /

Returns:

- `success`
  / *Type*: bool /
  Indicates if the computation of the keyword was successful or not
- `result`
  / *Type*: str /
  The result of the computation of the keyword

3.1.12 Keyword: inc_gauge

This keyword increments a gauge. The gauge has to be added with '`add_gauge`' before.

Arguments:

- `name`
  The name of the gauge
  / *Condition*: required / *Type*: str /
- `value`
  The value of increment. If not given, the value of the gauge is incremented by value 1.
  / *Condition*: optional / *Type*: int / *Default*: None /

- `labels`

  A semicolon separated list of labels assigned to the gauge. The order of labels must fit to the order of label names like defined in `add_gauge`.

  / *Condition*: optional / *Type*: str / *Default*: None /

Returns:

- `success`

  / *Type*: bool /

  Indicates if the computation of the keyword was successful or not

- `result`

  / *Type*: str /

  The result of the computation of the keyword

3.1.13 Keyword: dec_gauge

This keyword decrements a gauge. The gauge has to be added with '`add_gauge`' before.

Arguments:

- `name`

  The name of the gauge

  / *Condition*: required / *Type*: str /

- `value`

  The value of decrement. If not given, the value of the gauge is decremented by value 1.

  / *Condition*: optional / *Type*: int / *Default*: None /

- `labels`

  A semicolon separated list of labels assigned to the gauge. The order of labels must fit to the order of label names like defined in `add_gauge`.

  / *Condition*: optional / *Type*: str / *Default*: None /

Returns:

- `success`

  / *Type*: bool /

  Indicates if the computation of the keyword was successful or not

- `result`

  / *Type*: str /

  The result of the computation of the keyword

3.1.14 Keyword: add_summary

This keyword adds a new summary. The values of existing summaries can be set with `observe_summary`‘.

Arguments:

- `name`

  The name of the new summary

  / *Condition*: required / *Type*: str /

- `description`

  The description of the new summary

  / *Condition*: required / *Type*: str /

- `labels`
  A semicolon separated list of label names assigned to the new summary
  / *Condition*: optional / *Type*: str / *Default*: None /

Returns:

- `success`
  / *Type*: bool /
  Indicates if the computation of the keyword was successful or not
- `result`
  / *Type*: str /
  The result of the computation of the keyword

3.1.15 Keyword: observe_summary

This keyword observes a summary. The summary has to be added with '`add_summary`' before.

Arguments:

- `name`
  The name of the summary
  / *Condition*: required / *Type*: str /
- `value`
  The value assigned to the summary.
  / *Condition*: required / *Type*: int or float /
- `labels`
  A semicolon separated list of labels assigned to the summary. The order of labels must fit to the order of label names like defined in `add_summary`.
  / *Condition*: optional / *Type*: str / *Default*: None /

Returns:

- `success`
  / *Type*: bool /
  Indicates if the computation of the keyword was successful or not
- `result`
  / *Type*: str /
  The result of the computation of the keyword

3.1.16 Keyword: add_histogram

This keyword adds a new histogram. The values of existing histograms can be set with `observe_histogram`‘.

Arguments:

- `name`
  The name of the new histogram
  / *Condition*: required / *Type*: str /
- `description`
  The description of the new histogram
  / *Condition*: required / *Type*: str /

- `labels`
  A semicolon separated list of label names assigned to the new histogram
  / *Condition*: optional / *Type*: str / *Default*: None /

Returns:

- `success`
  / *Type*: bool /
  Indicates if the computation of the keyword was successful or not
- `result`
  / *Type*: str /
  The result of the computation of the keyword

3.1.17 Keyword: observe_histogram

This keyword observes a histogram. The histogram has to be added with '`add_histogram`' before.

Arguments:

- `name`
  The name of the histogram
  / *Condition*: required / *Type*: str /
- `value`
  The value assigned to the histogram.
  / *Condition*: required / *Type*: int or float /
- `labels`
  A semicolon separated list of labels assigned to the histogram. The order of labels must fit to the order of label names like defined in `add_histogram`.
  / *Condition*: optional / *Type*: str / *Default*: None /

Returns:

- `success`
  / *Type*: bool /
  Indicates if the computation of the keyword was successful or not
- `result`
  / *Type*: str /
  The result of the computation of the keyword

Chapter 4

Appendix

About this package:

Table 4.1: Package setup

Setup parameter	Value
Name	PrometheusInterface
Version	0.8.0
Date	21.10.2024
Description	Additional Robot Framework keywords
Package URL	robotframework-prometheus
Author	Holger Queckenstedt
Email	Holger.Queckenstedt@de.bosch.com
Language	Programming Language :: Python :: 3
License	License :: OSI Approved :: Apache Software License
OS	Operating System :: OS Independent
Python required	>=3.0
Development status	Development Status :: 4 - Beta
Intended audience	Intended Audience :: Developers
Topic	Topic :: Software Development

Chapter 5

History

0.1.0	05/2024
Initial version	
0.2.0	05/2024
Added metric type 'Gauge'; code maintenance	
0.3.0	05/2024
Added keywords 'inc_gauge' and 'dec_gauge'	
0.4.0	05/2024
Added interface description	
0.5.0	05/2024
Adapted handling of library version and date	
0.6.0	06/2024
Added metric type 'Info'	
0.6.1	06/2024
- Added prometheus-client as dependency package *- Maintained package's repository and workflow files*	
0.7.0	10/2024
Added metric types 'Summary' and 'Histogram'	
0.8.0	10/2024
"blank within key name" fix	

PrometheusInterface.pdf

Created at 21.10.2024 - 16:05:03

by GenPackageDoc v. 0.41.1