ska-sdp-qa-metric-generator Documentation

Release 0.18.0

SKAO, Naledi

Jan 29, 2024

CONTENTS:

This section will explain basic usage of the Metric Generator. This will explain how to do basic local development, as well as the configs required to run the generator as part of the receive workflow process.

This project is meant to be used along with the Signal Displays. Further reading can be found for the Signal system here.

The code can be found on GitLab.

CHAPTER

ONE

DEVELOPMENT

For local development there is a poetry environment that will be able to install most of the components, however some of the components assume you also have specific libraries installed on the host machine (you can find these in the `Dockerfile`). To work around this it is easier to use the Docker image instead for testing.

There are 2 scripts that are currently working, however we are changing from one to the other. The current script is `plasma_to_qa.py`, however we are changing over to `plasma_to_metrics.py`. So if you are using the one of the scripts and nothing is coming through, try the other.

The current processor is built based on the plasma processors available here.

1.1 Running the script

To run the script run the following command:

Note: You need to update the Kafka host and the MS file location.

```
BROKER_INSTANCE=broker plasma-processor \
  ska_sdp_qa_metric_generator.plasma_to_metrics.SignalDisplayMetrics \
  --input <ms file>
```

There are also helper arguments that can be used:

Argument	Description
`--use-random-ids`	This will generate random IDs instead of trying to read them from the enviroment.
`--ignore-config-db`	This instructs not to try to read the config DB.
`--disable-kafka`	This will run the process, but not output to Kafka.
`--metrics <all, stats,spectrum, spectograms,phase, amplitude,lag>`	This specifies a comma seperated list of metrics to generate, using `all` will generate everything.

For a list of other commands and options available use `--help` when running the script.

CHAPTER

TWO

USAGE IN PIPELINES

The Metric Generator is meant to be run with the processing pipelines as part of a visibility receive process.

The current generator config is as follows:

Note: The kafka host needs to be updated as required

Note: This config is meant to be appended to `processing_blocks[].parameters.processors`

```
{"signal-metrics": {
    "name": "qa-metrics-generator-plasma-receiver",
    "image": "artefact.skao.int/ska-sdp-qa-metric-generator",
    "version": "0.18.0",
    "command": [
        "plasma-processor",
        "ska_sdp_qa_metric_generator.plasma_to_metrics.SignalDisplayMetrics",
        "--plasma_socket",
        "/plasma/socket",
        "--readiness-file",
        "/tmp/processor_ready",
        "--use-sdp-metadata",
        "False",
        "--metrics",
        "all"
    ],
    "readinessProbe": {
        "initialDelaySeconds": 5,
        "periodSeconds": 5,
        "exec": {"command": ["cat", "/tmp/processor_ready"]}
    },
    "env": [
        {"name": "BROKER_INSTANCE", "value": "ska-sdp-kafka.test"}
    ]
}}
```

However, it is better to rather use the new style of config. You can also use this if you just want to override some of the values.

```
{"signal-display-metrics-all": {}}
```

The names that can be used is a follows:

- `signal-display-metrics-all` - this will run all the metrics
- `signal-display-metrics-amplitude` - this will only generate the amplitude graphs
- `signal-display-metrics-basic` - this will generate the base stats, and fast stats
- `signal-display-metrics-phase` - this will only generate the phase graphs

If you want to override the version you can do the following (any of the values can be overriden):

```
{"signal-display-metrics-all": {"version": "0.18.0"}}
```

The generator also outputs data on another Kafka topic used for monitoring. And to use that topic with the Queue Connector use the following config.

Note: The kafka host needs to be updated as required. And the topic name contains the subarray ID

Note: This config is meant to be appended to `processing_blocks[].parameters.queue_connector_configuration.exchanges`

```
{
    "dtype": "object",
    "shape": [],
    "source": {
        "type": "KafkaConsumerSource",
        "servers": "ska-sdp-kafka.test",
        "topic": "metrics-receive_state-01",
        "encoding": "json",
    },
    "sink": {
        "type": "TangoObjectScatterAttributeSink",
        "attributes": [
            {
                "attribute_name": "receiver_state",
                "filter": "type=='visibility_receive'",
                "path": "state",
                "dtype": "str",
                "default_value": "unknown",
            },
            {
                "attribute_name": "last_update",
                "filter": "type=='visibility_receive'",
                "path": "time",
                "dtype": "float",
                "default_value": 0.0,
            },
            {
                "attribute_name": "processing_block_id",
                "filter": "type=='visibility_receive'",
                "path": "processing_block_id",
                "dtype": "str",
```

(continues on next page)

(continued from previous page)

```
                "default_value": "",
            },
            {
                "attribute_name": "execution_block_id",
                "filter": "type=='visibility_receive'",
                "path": "execution_block_id",
                "dtype": "str",
                "default_value": "",
            },
            {
                "attribute_name": "subarray_id",
                "filter": "type=='visibility_receive'",
                "path": "subarray_id",
                "dtype": "str",
                "default_value": "-1",
            },
            {
                "attribute_name": "scan_id",
                "filter": "type=='visibility_receive'",
                "path": "scan_id",
                "dtype": "int",
                "default_value": 0,
            },
            {
                "attribute_name": "payloads_received",
                "filter": "type=='visibility_receive'",
                "path": "payloads_received",
                "dtype": "int",
                "default_value": 0,
            },
            {
                "attribute_name": "time_slices_received",
                "filter": "type=='visibility_receive'",
                "path": "time_slices",
                "dtype": "int",
                "default_value": 0,
            },
            {
                "attribute_name": "time_since_last_payload",
                "filter": "type=='visibility_receive'",
                "path": "time_since_last_payload",
                "dtype": "float",
                "default_value": 0.0,
            },
        ],
    },
}
```

CHAPTER

THREE

USAGE IN LOCAL TESTING

For local testing it is simplest within the docker container. However, it is possible to run on your host machine, refer to `Dockerfile` for the required libraries to be installed on your host machine.

To be able to use the helper commands it is assumed that you have the Signal Metric API locally and already running.

You then need to build the Docker environment using:

```
make build
```

Then to run system copy a MS directory into `data/`. And run the following:

```
make run MS=data/<name of MS file>
```

The processor will then send the data through to Kafka that is assumed to be running.