128T Monitoring Agent Release Notes Version 1.0.0

Abstract

128T Monitoring Agent Release Notes

14 February 2020

Contents

128T Monitoring Agent

Monitoring agent is an entity for collecting data from a node running 128T software. It is capable of collecting the data from several sources such as metrics, events etc. The current mechanism of monitoring a 128T router involves doing REST or GraphQL queries from the conductor. At scale, this can be really inefficient and can be problematic in terms of the performance of the conductor. In addition, it's important to interact with other 3rd party monitoring platforms as means for organizations to collect, analyze and report using various KPIs available from 128T software and other application in the network.

The monitoring agent at its core is designed to be able to push data to external platforms. It currently leverages the telegraf collection stack on every 128T router. However, is designed with other tools and scale in mind. A monitoring agent is composed of the following:

- monitoring-agent-cli: Used for configuring and interacting with the underlying application
- collectors A set of inputs designed to collect data from several sources such as metrics, events etc.
- targets A set of outputs for collecting and pushing the data to various data sinks such as external monitoring platforms, files on disk etc.

Installation

The 128T Monitoring Agent is installed using the dnf utility:

```
dnf install 128T-monitoring-agent
```

example:

```
# dnf install 128T-monitoring-agent
128 Technology 7 - x86_64
13 MB/s |  30 MB     00:02
Dependencies resolved.
============================================================================================
===============================
 Package                              Arch                 Version
Repository                        Size
============================================================================================
===============================
Installing:
 128T-monitoring-agent                x86_64               1.0.0-1
128tech-release                  6.7 M
Installing dependencies:
 telegraf-128tech                     x86_64               1.13.1-2
128tech-release                   15 M

Transaction Summary
============================================================================================
===============================
Install  2 Packages

Total download size: 22 M
Installed size: 97 M
Is this ok [y/N]: y
Downloading Packages:
```

```
(1/2): 128T-monitoring-agent-1.0.0-1.x86_64.rpm
6.0 MB/s | 6.7 MB     00:01
(2/2): telegraf-128tech-1.13.1-2.x86_64.rpm
10 MB/s |  15 MB     00:01
---------------------------------------------------------------------------------------------
-------------------------------
Total
13 MB/s |  22 MB     00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                          1/1
  Running scriptlet: telegraf-128tech-1.13.1-2.x86_64                         1/2
  Installing       : telegraf-128tech-1.13.1-2.x86_64                         1/2
  Running scriptlet: telegraf-128tech-1.13.1-2.x86_64
1/2
Created symlink from /etc/systemd/system/multi-user.target.wants/telegraf.service to
/usr/lib/systemd/system/telegraf.service.
  Installing       : 128T-monitoring-agent-1.0.0-1.x86_64                     2/2
  Running scriptlet: 128T-monitoring-agent-1.0.0-1.x86_64                     2/2
  Running scriptlet: telegraf-128tech-1.13.1-2.x86_64                         2/2
  Verifying        : 128T-monitoring-agent-1.0.0-1.x86_64                     1/2
  Verifying        : telegraf-128tech-1.13.1-2.x86_64                         2/2

Installed:
  128T-monitoring-agent.x86_64 1.0.0-1                  telegraf-128tech.x86_64 1.13.1-2

Complete!
```

Configuration

The monitoring agent has its own set of configurations and looks for inputs from specific directories on disk. By default, the configuration for the agent should be present in `/etc/128t-monitoring/config.yaml` and uses YAML format which looks something like this:

```
enabled: true
tags:
  key: router
  value: ${ROUTER}
sample-interval: 1
push-interval: 1
inputs:
- name: events
- name: t128_metrics
  include-outputs: [message_queue]
- name: t128_device_state
- name: t128_peer_path
- name: lte_metric
  exclude-outputs: [file]
outputs:
- name: file
- name: message_queue
```

The `enabled` field is meant as global toggle for applying the monitoring agent functionality.

The `tags` are used to add meta information to the collected metrics, state etc in order to make it easier to identify the origin, ease of identification, filtering etc. By default, the agent will include the `${HOSTNAME}`, `${ROUTER}` and `${NODE}` tags to every collected input. The corresponding values are derived from the running system. The same config can ideally be used for each node in the authority and the appropriate values will be determined at runtime.

The `sample-interval` and `push-interval` indicate the frequency (in seconds) of how often the data is collected and eventually pushed to the collection target. When the `push-interval` is higher than the `sample-interval`, it will produce `N` samples collected within the push duration. As a result, its recommended to configure the `push-interval` as a multiple of `sample-interval`.

The `inputs` represent a single unit of collection. This can be a combination of inputs available from `telegraf` as well as other inputs developed by 128T. The function and configuration of each of the 128T provided inputs can be found in subsequent sections. For the `telegraf` inputs please refer to the influx documentation online. Each `input` can be a combination of one or more collectors and can contain other collector specific information. For each of the inputs, a user can also configure an `include-outputs` which is a list of outputs to send the collected information to. This allows the user to build a matrix of inputs and outputs and provides a granular control over which input should be sent to what output. Similarly, the user can also configure a `exclude-outputs` which will include all defined outputs except the one specified.

The `outputs` represent a data sink where the collected information is to be delivered. By virtue of using `telegraf`, the monitoring agent gets automatic support of the available outputs supported by telegraf. Each `input` can be configured to be delivered to one or more `output`.

Directory Structure

The `monitoring-agent` uses a well-defined directory structure where it derives the inputs from various configuration. The following directories are especially important:

- /var/lib/128t-monitoring/inputs/ Setup the `inputs` directory with config files for the various inputs enabled in the monitoring-agent configuration. The monitoring agent expects to see a file called `<input-name.conf>` in this directory. Users can override the file name by specifying `conf: <filename.conf>` in the input definition within the config above. This file should only contain the telegraf definition for the input(s) that belong and not any other configuration. For example, the configuration for the `t128_metrics` input would look something like this

```
[[inputs.t128_metrics]]
    ## When configured, the metric collector input will pull KPIs from the 128T system
    ## running on the current node. Depending on the KPI, the information can be used for
    ## monitoring various aspects of the running system such as services, interfaces, errors
etc.

    ## By default, if no configuration is present, the set of metrics defined in
    ## /etc/128t-monitoring/collectors/t128_metrics/default_config.toml will be used
    ## for monitoring. Here's a sample configuration on how to define custom metrics.
    ##
    ## [[inputs.t128_metric.metric]]
    ## name = "peer_path"
    ##
    ## [inputs.t128_metric.metric.fields]
    ## Refer to the 128T REST swagger documentation for the list of available metrics
    ##     key_name = "stats/<path_to_metric>"
    ##     latency = "stats/bfd/peer-path/latency"
    ##
    ## [inputs.t128_metric.metric.parameters]
    ##     parameter_name = ["value1", "value2"]
    ##     peer_path = ["path1"]

    timeout = "15s"
```

- /var/lib/128t-monitoring/inputs/ The `outputs` directory will contain the config files for the various data sink configured in the monitoring-agent configuration. For each `output` the conf file should contain the telegraf configuration for that one output only. This allows the monitoring-agent to create a telegraf config per input and include the appropriate outputs. For example:

```
[[outputs.file]]
  ## Files to write to, "stdout" is a specially handled file.
  files = ["stdout", "/tmp/metrics.out"]

  ## Use batch serialization format instead of line based delimiting.  The
  ## batch format allows for the production of non line based output formats and
  ## may more efficiently encode metric groups.
  # use_batch_format = false

  ## The file will be rotated after the time interval specified.  When set
  ## to 0 no time based rotation is performed.
  # rotation_interval = "0d"

  ## The logfile will be rotated when it becomes larger than the specified
  ## size.  When set to 0 no size based rotation is performed.
  # rotation_max_size = "0MB"

  ## Maximum number of rotated archives to keep, any older logs are deleted.
  ## If set to -1, no archives are removed.
  # rotation_max_archives = 5

  ## Data format to output.
  ## Each data format has its own unique set of configuration options, read
  ## more about them here:
  ## https://github.com/influxdata/telegraf/blob/master/docs/DATA_FORMATS_OUTPUT.md
  data_format = "influx"
```

- /var/lib/128t-monitoring/config When the monitoring-agent `config`, `inputs` and `outputs` directories are setup correctly, the monitoring agent will then create fully formed telegraf config per input in this folder. For example, the metrics input and the file output above will result in a configuration file such as:

```
[global_tags]
router = "lte-router"
node = "lte-node"

[agent]
interval = 10
flush_interval = 20

[inputs]
[[inputs.exec]]
timeout = "15s"
```

```
commands = [ "/usr/bin/metricCollector128t --config /etc/128t-
monitoring/collectors/t128_metrics/default_config.toml",]
data_format = "influx"

[outputs]
[[outputs.file]]
files = ["stdout", "/tmp/metrics.out"]
data_format = "influx"
```

Monitoring Agent CLI

The `monitoring-agent-cli` is a utility for validating and executing the configuration for the monitoring-agent. The various components of the CLI as follows:

Validation

The `monitoring-agent validate` command will ensure that the monitoring-agent config along with other inputs and outputs are correctly setup and flag any particular errors to the user. The `validate` command will not make any changes to the running system.

Sample Config

The `monitoring-agent-cli sample` command can be used to view the various collectors that are created as part of the 128T monitoring agent. The `list-available` command will simply show the set of available inputs (and outputs) that are packaged as part of the 128T monitoring-agent. These are in addition to the one's available natively via telegraf.For example:

```
# monitoring-agent-cli sample list-available
inputs:
- events
- t128_metrics
- t128_device_state
- t128_peer_path
- lte_metric
```

The configuration for each of these inputs can be viewed via `monitoring-agent-cli sample view <plugin_name>` command such as:

```
# monitoring-agent-cli sample view t128_metrics
[[inputs.t128_metrics]]
    ## When configured, the metric collector input will pull KPIs from the 128T system
    ## running on the current node. Depending on the KPI, the information can be used for
    ## monitoring various aspects of the running system such as services, interfaces, errors
etc.

    ## By default, if no configuration is present, the set of metrics defined in
    ## /etc/128t-monitoring/collectors/t128_metrics/default_config.toml will be used
    ## for monitoring. Here's a sample configuration on how to define custom metrics.
    ##
    ## [[inputs.t128_metric.metric]]
    ## name = "peer_path"
    ##
    ## [inputs.t128_metric.metric.fields]
    ## Refer to the 128T REST swagger documentation for the list of available metrics
    ##     key_name = "stats/<path_to_metric>"
    ##     latency = "stats/bfd/peer-path/latency"
    ##
    ## [inputs.t128_metric.metric.parameters]
    ##     parameter_name = ["value1", "value2"]
    ##     peer_path = ["path1"]
```

Configuration

When the `monitoring-agent config` command is run, it will first validate and report any errors to the user. Once valid configuration is in place, the configure command does the following at a high level:

- For each of the configured and enabled inputs, generate a telegraf config file in the `/var/lib/128t-monitoring/config` directory
- Launch an instance of the `128T-telegraf` service for each of the configure inputs which allows us to run each input independently.

At this point, each input will be running a telegraf instance and will allow the collection of inputs & outputs to run on the system.

128T Collectors

The 128T monitoring-agent comes pre-packaged with a set of collectors to assist in the monitoring of the 128T platform. Here are the various collectors and how to use them:

Metric collector

The `metricCollector128t` python executable is responsible for collecting the configured metrics from a running system. By default, the metrics specified in `/etc/128t-monitoring/collectors/t128_metrics/default_config.toml` will be used by the collector. This represents a set of pre-configured metrics that 128T recommends that we monitor. The configuration file in a `TOML` definition of metrics and has the following format:

```
[[metric]]
  name = "service"
  [metric.fields]
    packets-received = "stats/aggregate-session/service/packets-received"
    packets-transmitted = "stats/aggregate-session/service/packets-transmitted"
    session-arrival-rate = "stats/aggregate-session/service/session-arrival-rate"
    session-departure-rate = "stats/aggregate-session/service/session-departure-
rate"
    bandwidth-received = "stats/aggregate-session/service/bandwidth-received"
    bandwidth-transmitted = "stats/aggregate-session/service/bandwidth-transmitted"
    tcp-retransmissions = "stats/aggregate-session/service/tcp-retransmissions"
    session-count = "stats/aggregate-session/service/session-count"
  [metric.parameters]
    service = []
```

The `name` becomes the name of the measurement in the context of influxdb format. The `metric.fields` represent the various metrics to be collected. The `packets-received` in the above example will be field-name for the `stats/aggregate-session/service/packets-received` KPI which is the path of that KPI from the 128T REST API documentation. Finally, the `metric.parameters` can be used to configure key parameters such as `service` to be used for filtering the set of collected stats. In the above example, the metrics would be collected for all services but the `service` parameter can be used to specify a subset of services to monitor instead

Event Collector

The event collector can be used for collecting and pushing events for various categories such as admin, alarm, system, traffic and provisioning as they occur on the system. The type of the event is available via a `tag` and can be used for filtering only specific events as desired. For example, the following configuration can be used for pushing just the `alarm` and `admin` event

```
[[inputs.execd]]
  ## Create a stream of 128T events for alarm, audit etc. This information is useful for
  ## monitoring the health of the system.
  command = "/usr/bin/eventCollector128t"
  signal = "none"
  data_format = "influx"

  ## input event filtering based on type (admin, alarm, system, traffic, provisioning)
  ## NOTE: For information on filtering severity refer to the output configuration example
  [inputs.execd.tagpass]
  type = ["alarm", "admin"]
```

Device interface state collector

The `deviceInterfaceStateCollector128t` collector can be used for monitoring the admin, oper and redundancy status of various device-interfaces configured on the node. The name is available as `device-interface` tag and telegraf `tagpass` can be used to filter specific interfaces as needed. For example:

```
[[inputs.exec]]
  ## Collect information about the 128T device-interface admin, operational and
  ## redundancy status. This information is useful for monitoring the system health.
  commands = ["/usr/bin/deviceInterfaceStateCollector128t"]

  ## Timeout for the device-interface state collector to finish
  timeout = "5s"

  ## Data format to consume.
  ## Each data format has its own unique set of configuration options, read
  ## more about them here:
  ## https://github.com/influxdata/telegraf/blob/master/docs/DATA_FORMATS_INPUT.md
  data_format = "influx"

  ## To filter on select device interfaces, you can use the `tagpass` and `tagdrop` concepts
  ## from telegraf. For example:
  ## [[inputs.exec.tagpass]]
  ##     device-interface = ["wan1"]
```

Peer Path State collector

The `peerPathStateCollector128t` collector can be used for monitoring the up/down status of all the peer paths on the node. The various part of a peer-path such as `adjacentAddress` and `networkInterface` are available as tags which can be filtered. For example:

```
[[inputs.exec]]
  ## Collect information about the 128T adjacency peer-path status. This information
  ## is useful to monitoring the secure WAN connectivity to the peers
  commands = ["/usr/bin/peerPathStateCollector128t"]

  ## Timeout for the peer-path state collector to finish
  timeout = "5s"

  ## Data format to consume.
  ## Each data format has its own unique set of configuration options, read
  ## more about them here:
  ## https://github.com/influxdata/telegraf/blob/master/docs/DATA_FORMATS_INPUT.md
  data_format = "influx"

  ## To filter on select peer-paths, you can use the `tagpass` and `tagdrop` concepts
  ## from telegraf. For example:
  ## [[inputs.exec.tagpass]]
  ##     adjacentAddress = ["10.10.10.10"]
  ##     networkInterface = ["wan1"]
```

LTE Collector

The `lteMetricCollector128t` collector when run will scan the current node configuration for any 128T supported and configured LTE devices. This collector can be used for pushing the `signal-strength` and `carrier` information to the monitoring stack. For example:

```
[[inputs.exec]]
  ## Collect the signal-strength and carrier information from configured LTE card(s) on
  ## the system. This information is useful for monitoring any fluctuations in carrier
  ## signal causing loss of connectivity.
  commands = ["/usr/bin/lteMetricCollector128t"]

  ## Timeout for the LTE metric collector to finish
  timeout = "10s"

  ## Data format to consume.
  ## Each data format has its own unique set of configuration options, read
  ## more about them here:
  ## https://github.com/influxdata/telegraf/blob/master/docs/DATA_FORMATS_INPUT.md
  data_format = "influx"
```