NVIDIA DOCA Telemetry Service

Guide

MLNX-15-060509 _v1.2 | January 2022

Table of Contents

Chapter 1. Introduction

DOCA Telemetry Service (DTS) runs inside of its own Kubernetes pod on BlueField collecting data from built-in providers and from external telemetry applications. The following 3 providers are available (disabled by default):

- sysfs
- ethtool
- tc (traffic controll)

Additional telemetry applications, such as the DOCA Telemetry Client Reference App, should run in their own pods on the same BlueField.

DTS stores collected data into binary files under the `/opt/mellanox/doca/services/telemetry/data` directory. Data write is disabled by default due to BlueField storage restrictions.

DTS can export the data via Prometheus Endpoint (pull) or Fluent Bit (push). The Prometheus endpoint is bound to port 9100 and can be enabled using the `dts_config.ini` config file.

DTS allows exporting NetFlow packets when data is collected from the DOCA Telemetry NetFlow API client application. NetFlow exporter is enabled from `dts_config.ini` by setting NetFlow collector IP/address and port.

Chapter 2. Service Deployment

For more information about the deployment of DOCA containers on top of the BlueField DPU, refer to NVIDIA DOCA Container Deployment Guide.

2.1. DOCA Service on NGC

DTS is available on NGC, NVIDIA's container catalog. Service-specific configuration steps and deployment instructions can be found under the service's container page.

2.2. Standalone Deployment – BlueField OS

In addition to being available through NGC, DTS is also available in an offline standalone form as part of the BlueField OS image at `/opt/mellanox/doca/services/telemetry/doca_telemetry_service_${version}_arm64.tar.gz`.

Note: It is highly recommended to deploy the service through NGC rather than using the standalone deployment. The standalone deployment is aimed to only be used in offline environments in which NGC is not accessible.

If offline deployment is required, follow these instructions:

```
# Pre-Requisites (explained in the DOCA Container Deployment Guide)
systemctl start kubelet
systemctl start containerd
mkdir -p /opt/mellanox/doca/services/telemetry/config
mkdir -p /opt/mellanox/doca/services/telemetry/data
mkdir -p /opt/mellanox/doca/services/telemetry/ipc_sockets
# Unpacking the built-in container
cd /opt/mellanox/doca/services/telemetry && \
    gunzip -k doca_telemetry_service_1.2_arm64.tar.gz && \
    ctr --namespace k8s.io image import doca_telemetry_1.2_arm64.tar && \
    rm -rf doca_telemetry_service_1.2_arm64.tar
```

The `.yaml` file for the standalone deployment is also placed under the service's directory, and is named `doca_telemetry_standalone.yaml`:

```
/opt/mellanox/doca/services/telemetry/doca_telemetry_standalone.yaml
```

Once extracted, the deployment of the DOCA service container is similar to other DOCA containers and requires copying the service's `.yaml` file to Kubelet's input directory:

```
cp /opt/mellanox/doca/services/telemetry/doca_telemetry_standalone.yaml /etc/
kubelet.d
```

Note: Before the DTS can be deployed, some preliminary configuration steps must be completed. A detailed overview of the service's configuration steps can be found in the next sections.

Note: When using IPC transport to collect data from a DOCA-Telemetry-API-based application (or telemetry client), the application should run from the same pod. This means the `.yaml` has to support a configuration where two containers are running in a single pod and `hostIPC=false`.

Chapter 3. Configuration

The configuration of DTS is placed under `/opt/mellanox/doca/services/telemetry/config` by DTS during initialization. The user can interact with the `dts_config.ini` file and `fluent_bit_configs` folder. `dts_config.ini` contains the main configuration for the service and must be used to enable/disable providers, exporters, data writing. More details are provided in the corresponding sections. For every update in this file, DST must be restarted. Interaction with `fluent_bit_configs` folder is described in section Fluent Bit.

3.1. Init Scripts

The `InitContainers` section of the `.yaml` file has 2 scripts for config initialization:

- `/usr/bin/telemetry-init.sh` – generates the default configuration files if, and only if, the `/opt/mellanox/doca/services/telemetry/config` folder is empty.
- `/usr/bin/enable-forward-to-morpheus.sh` – configures the destination host and port for Fluent Bit forwarding. The scripts requires that both the host and port are present, and only in this case it would start. The script overwrites the `/opt/mellanox/doca/services/telemetry/config/fluent_bit_configs` folder and configures the `forward.exp` file. It inputs 3 arguments: `host`, `port`, and `data_set`. The `data_set` argument must be set to `all_data` which signifies no data filtering.

> **Note:** Not setting `data_set` argument filters out all data from the exporter.

3.2. Enabling Fluent Bit Forwarding

If enabling Fluent Bit forwarding is desired, add the destination host and port to the command line found in the `initContainers` section of the `.yaml` file:

```
command: ["/bin/bash/", "-c", /usr/bin/telemetry-init.sh && /usr/bin/enable-forward-to-morpheus.sh 127.0.0.1 24224 all_data"]
```

> **Note:** The host and port shown above are just an example and `all_data` must be set as a 3rd argument to allow data streaming. See section Fluent Bit to learn about manual configuration.

3.3. Generating Configuration

The configuration folder `/opt/mellanox/doca/services/telemetry/config` starts empty by default. Once the service starts, the initial scripts run as a part of the initial container and create configuration as described in section Enabling Fluent Bit Forwarding.

3.4. Resetting Configuration

Resetting the configuration can be done by deleting the content found in the configuration folder and restarting the service to generate the default configuration.

3.5. Disabling Providers

Disabling a provider can be done using the `dts_config.ini` configuration file. Uncomment the `disable-provider=$provider-name` line to disable data collection for this provider. For example, uncommenting the following line disables the `ethtool` provider:

```
#disable-provider=ethtool
```

Note: More information about telemetry providers can be found under the Providers section.

3.6. Enabling Data Write

Uncomment the following line in `dts_config.ini`:

```
#output=/data
```

Note: Any changes in `dts_config.ini` file necessitate restarting the pod for the new settings to apply.

Chapter 4. Providers

DTS supports on-board data collection from sysf, ethtool, and tc providers.

4.1. Sysfs Counters List

- `ib_port` counters:

```
mlx5_0:1:VL15_dropped
mlx5_0:1:excessive_buffer_overrun_errors
mlx5_0:1:link_downed
mlx5_0:1:link_error_recovery
mlx5_0:1:local_link_integrity_errors
mlx5_0:1:multicast_rcv_packets
mlx5_0:1:multicast_xmit_packets
mlx5_0:1:port_rcv_constraint_errors
mlx5_0:1:port_rcv_data
mlx5_0:1:port_rcv_errors
mlx5_0:1:port_rcv_packets
mlx5_0:1:port_rcv_remote_physical_errors
mlx5_0:1:port_rcv_switch_relay_errors
mlx5_0:1:port_xmit_constraint_errors
mlx5_0:1:port_xmit_data
mlx5_0:1:port_xmit_discards
mlx5_0:1:port_xmit_packets
mlx5_0:1:port_xmit_wait
mlx5_0:1:symbol_error
mlx5_0:1:unicast_rcv_packets
mlx5_0:1:unicast_xmit_packets
```

- `ib_hw` counters:

```
mlx5_0:1:hw_duplicate_request
mlx5_0:1:hw_implied_nak_seq_err
mlx5_0:1:hw_lifespan
mlx5_0:1:hw_local_ack_timeout_err
mlx5_0:1:hw_out_of_buffer
mlx5_0:1:hw_out_of_sequence
mlx5_0:1:hw_packet_seq_err
mlx5_0:1:hw_req_cqe_error
mlx5_0:1:hw_req_cqe_flush_error
mlx5_0:1:hw_req_remote_access_errors
mlx5_0:1:hw_req_remote_invalid_request
mlx5_0:1:hw_resp_cqe_error
mlx5_0:1:hw_resp_cqe_flush_error
mlx5_0:1:hw_resp_local_length_error
mlx5_0:1:hw_resp_remote_access_errors
mlx5_0:1:hw_rnr_nak_retry_err
mlx5_0:1:hw_rx_atomic_requests
mlx5_0:1:hw_rx_dct_connect
mlx5_0:1:hw_rx_icrc_encapsulated
mlx5_0:1:hw_rx_read_requests
```

```
mlx5_0:1:hw_rx_write_requests
```

- `ib_mr_cache` counters:

> **Note:** `n` ranges from 0 to 24.

```
mlx5_0:mr_cache:size_{n}:cur
mlx5_0:mr_cache:size_{n}:limit
mlx5_0:mr_cache:size_{n}:miss
mlx5_0:mr_cache:size_{n}:size
```

4.2. Ethtool Counters

Ethtool counters is the generated list of counters which corresponds to Ethtool utility. Counters are generated on a per-device basis.

4.3. Traffic Control Info

The following TC objects are supported and reported regarding the ingress filters:

- Filters
  - flower
- Actions
  - mirred
  - tunnel_key

The info is provided as one of the following events:

- Basic filter event
- flower/ipv4 filter event
- flower/ipv6 filter event
- Basic action event
- mirred action event
- tunnel_key/ipv4 action event
- tunnel_key/ipv6 action event

General notes:

- Actions always belong to a filter, so action events share the filter event's ID via the `event_id` data member
- Basic filter event only contains textual *kind* (so users can see which real life objects' support they are lacking)
- Basic action event only contains textual *kind* and some basic common statistics if available

Chapter 5. Data Outputs

DTS can send the collected data to the following outputs:

- Data writer (saves binary data to disk)
- Fluent Bit (push-model streaming)
- Prometheus endpoint (keeps the most recent data to be pulled).

5.1. Data Writer

The data writer is disabled by default to save space on BlueField. Steps for activating data write during debug can be found under section Enabling Data Write.

The schema folder contains JSON-formatted metadata files which allow reading the binary files containing the actual data. The binary files are written according to the naming convention shown in the following example (`apt install tree`):

```
tree /opt/mellanox/doca/services/telemetry/data/
/opt/mellanox/doca/services/telemetry/data/
├── {year}
│   └── {mmdd}
│       └── {hash}
│           ├── {source_id}
│           │   └── {source_tag}{timestamp}.bin
│           └── {another_source_id}
│               └── {another_source_tag}{timestamp}.bin
└── schema
    └── schema_{MD5_digest}.json
```

New binary files appears when the service starts or when binary file age/size restriction is reached. If no schema or no data folders are present, refer to the Troubleshooting section.

Note: `source_id` is usually set to the machine hostname. `source_tag` is a line describing the collected counters, and it is often set as the provider's name or name of user-counters.

Reading the binary data can be done from within the DTS container using the following command:

```
crictl exec -it <Container ID> /opt/mellanox/collectx/bin/clx_read -s /data/schema /
data/path/to/datafile.bin
```

Note: The path to the data file must be an absolute path.

Example output:

```
{
    "timestamp": 1634815738799728,
    "event_number": 0,
    "iter_num": 0,
    "string_number": 0,
    "example_string": "example_str_1"
}
{
    "timestamp": 1634815738799768,
    "event_number": 1,
    "iter_num": 0,
    "string_number": 1,
    "example_string": "example_str_2"
}
…
```

5.2. Prometheus

The Prometheus endpoint keeps the most recent data to be pulled by the Prometheus server and is enabled by default.

To check that data is available, run the following command on BlueField:

```
curl -s http://0.0.0.0:9100/metrics
```

The command dumps every counter in the following format:

```
counter_name {list of meta fields} counter_value timestamp
```

Note: The default port for Prometheus can be changed in `dts_config.ini`.

5.3. Fluent Bit

Fluent Bit allows streaming to multiple destinations. Destinations are configured in `.exp` files that are documented in-place and can be found under:

```
/opt/mellanox/doca/services/telemetry/config/fluent_bit_configs
```

Fluent Bit allows exporting data via "Forward" protocol which connects to the Fluent Bit/FluentD instance on customer side.

Export can be enabled manually:

1. Uncomment the line with `fluent_bit_configs=` in `dts_config.ini`.
2. Set `enable=1` in required `.exp` files for the desired plugins.
3. Additional configurations can be set according to instructions in the `.exp` file if needed.
4. Restart the DTS.
5. Set up receiving instance of Fluent Bit/FluentD if needed.
6. See the data on the receiving side.

Export file destinations are set by configuring `.exp` files or creating new ones. It is recommended to start by going over documented example files. Documented examples exist for the following plugins:

- forward
- file
- influxdb
- stdout

Note: All `.exp` files are disabled by default if not configured by `initContainer` entry point through `.yaml` file.

Note: To forward the data to several destinations, create several `forward_{num}.exp` files. Each of these files must have their own destination host and port.

5.3.1. Export File Configuration Details

Each export destination has the following fields:

- `name` – configuration name
- `plugin_name` – Fluent Bit plugin name
- `enable` – 1 or 0 values to enable/disable this destination
- `host` – the host for Fluent Bit plugin
- `port` – port for Fluent Bit plugin
- `msgpack_data_layout` – the msgpacked data format. Default is `flb_std`. The other option is `custom`. See section Msgpack Data Layout for details.
- `plugin_key=val` – key-value pairs of Fluent Bit plugin parameter (optional)
- `counterset/fieldset` – file paths (optional). See details in section Cset/Fset Filtering.
- `source_tag=source_tag1,source_tag2` – comma separated list of data page source tags for filtering. The rest tags will be filtered out during export.

Note: Use `#` to comment a configuration line.

5.3.2. Msgpack Data Layout

Data layout can be configured using `.exp` files by setting `msgpack_data_layout=layout`. There are two available layouts: Standard and Custom.

The standard `flb_std` data layout is an array of 2 fields:

- timestamp double value
- a plain dictionary (key-value pairs)

The standard layout is appropriate for all Fluent Bit plugins. For example:

```
[timestamp_val, {"timestamp"->ts_val, type=>"counters/
events", "source"=>"source_val", "key_1"=>val_1, "key_2"=>val_2,...}]
```

The custom data layout is a dictionary of meta-fields and counter fields. Values are placed into a separate plain dictionary. Custom data format can be dumped with `stdout_raw` output plugin of Fluent-Bit installed, or can be forwarded with `forward` output plugin.

Counters example:

```
{"timestamp"=>timestamp_val, "type"=>"counters", "source"=>"source_val", "values"=>
 {"key_1"=>val_1, "key_2"=>val_2,...}}
```

Events example

```
{"timestamp"=>timestamp_val, "type"=>"events", "type_name"=>"type_name_val", "source"=>"
 source_val", "values"=>{"key_1"=>val_1, "key_2"=>val_2,...}}
```

5.3.3. Cset/Fset Filtering

Each export file can optionally use one cset and one fset file to filter UFM telemetry counters and events data.

- Cset file contains tokens per line to filter data with `"type"="counters"`.
- Fset contains several blocks started with the header line `[event_type_name]` and tokens under that header. An Fset file is used to filter data with `"type"="events"`.

If several tokens must be matched simultaneously, use `<tok1>+<tok2>+<tok3>`. Exclusive tokens are available as well. For example, the line `<tok1>+<tok2>-<tok3>-<tok4>` filters names that match both tok1 and tok2 and do not match tok3 or tok4.

Note: For more details see documentation in the files `ufm_enterprise.cset` and `ufm_enterprise.fset` inside the UFM Telemetry docker folder `/config/fluent_bit_configs`.

The following are details from `/config/fluent_bit_configs/ufm_enterprise.cset`:

```
# put tokens on separate lines
# Tokens are the actual name 'fragments' to be matched
# port$ # match names ending with token "port"
# ^port # match names starting with token "port"
# ^port$ # include name that is exact token "port
# port+xmit # match names that contain both tokens "port" and "xmit"
# port-support # match names that contain the token "port" and do not match the "-"
 token "support"
#
# Tip: To disable counter export put a single token line that fits nothing
# List of available counters:
#
# node_guid
# port_guid
# port_num
# lid
# link_down_counter
# link_error_recovery_counter
# symbol_error_counter
# port_rcv_remote_physical_errors
# port_rcv_errors
# port_xmit_discard
# port_rcv_switch_relay_errors
# excessive_buffer_errors
```

The following are details from `/config/fluent_bit_configs/ufm_enterprise.fset`:

```
# Put your events here
# Usage:
#
# [type_name_1]
```

```
# tokens
# [type_name_2]
# tokens
# [type_name_3]
# tokens
# ...
# Tokens are the actual name 'fragments' to be matched
# port$ # match names ending with token "port"
# ^port # match names starting with token "port"
# ^port$ # include name that is exact token "port
# port+xmit # match names that contain both tokens "port" and "xmit"
# port-support # match names that contain the token "port" and do not match the "-"
 token "support"

# The next example will export the whole "switch_fan" events and events "CableInfo"
 filtered with token "port" :
# [switch_fan]
#
# [CableInfo]
# port

# To know which event type names are available use one of these options:
# 1. Check export and find field "type_name"=>"switch_temperature"
# OR
# 2. Open log file "/tmp/ibd/ibdiagnet2_port_counters.log" and find event types are
 printed to log:
# ...
# [info] type [CableInfo] is type of interest
# [info] type [switch_temperature] is type of interest
# [info] type [switch_fan] is type of interest
# [info] type [switch_general] is type of interest
# ...
# Corner cases:
# 1. Empty fset file will export all events.
# 2. Tokens written above/without [event_type] will be ignored.
# 3. If cannot open fset file, warning will be printed, all event types will be
 exported.
```

5.4. NetFlow Exporter

NetFlow exporter must be used when data is collected as NetFlow packets from the telemetry client applications. In this case, DOCA Telemetry NetFlow API sends NetFlow data packages to DTS via IPC. DTS uses NetFlow exporter to send data to the NetFlow collector (3rd party service).

To enable NetFlow exporter, set `netflow-collector-ip` and `netflow-collector-port` in `dts_config.ini`. `netflow-collector-ip` could be set either to IP or an address.

For additional information, refer to the `dts_config.ini` file.

Chapter 6. Troubleshooting

On top of the troubleshooting section found in the NVIDIA DOCA Container Deployment Guide, here are additional troubleshooting tips for DTS:

- If no pod is created, make sure the folders are created as follows:

```
$ sudo mkdir -p /opt/mellanox/doca/services/telemetry/config
$ sudo mkdir -p /opt/mellanox/doca/services/telemetry/ipc_sockets
$ sudo mkdir -p /opt/mellanox/doca/services/telemetry/data
```

- If the pod's STATE fails to be marked as "Ready", refer to the log `/var/log/syslog`.
- Check if the service is configured to write data to the disk, as this may cause the system to run out of disk space.
- If `/opt/mellanox/doca/services/telemetry/data folder` contains no schema or data folder, refer to the `clx.log` file:

```
crictl exec -it <Container ID> cat /var/log/clx.log
```

  If the error `Failed to allocate data page od size 16384...` appears in the log, it signifies that the buffer size is not big enough to fit the data.

```
[2021-07-22 12:42:26.675] [error][data_page] Failed to allocate data page of size
 16384 which is less then header size 720 + block size 30112
[2021-07-22 12:42:26.675] [error] Data page allocation failed
```

  Increase the buffer size by modifying the buffer size line in the file:

```
# vi /opt/mellanox/doca/services/telemetry/config/dts_config.ini
```

  Refresh the `.yaml` file and check the data using the tree command as shown earlier.

- If a PIC bus error occurs, configure the following files inside the container:

```
crictl exec -it <Container ID> /bin/bash
# Add to /config/clx.env the following line:
"
export UCX_TLS=tcp
"
```

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: "NVIDIA") make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world¬wide basis. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com