NVIDIA DOCA Flow Inspector Service

Guide

MLNX-15-060530 _v2.2.0 | October 2023

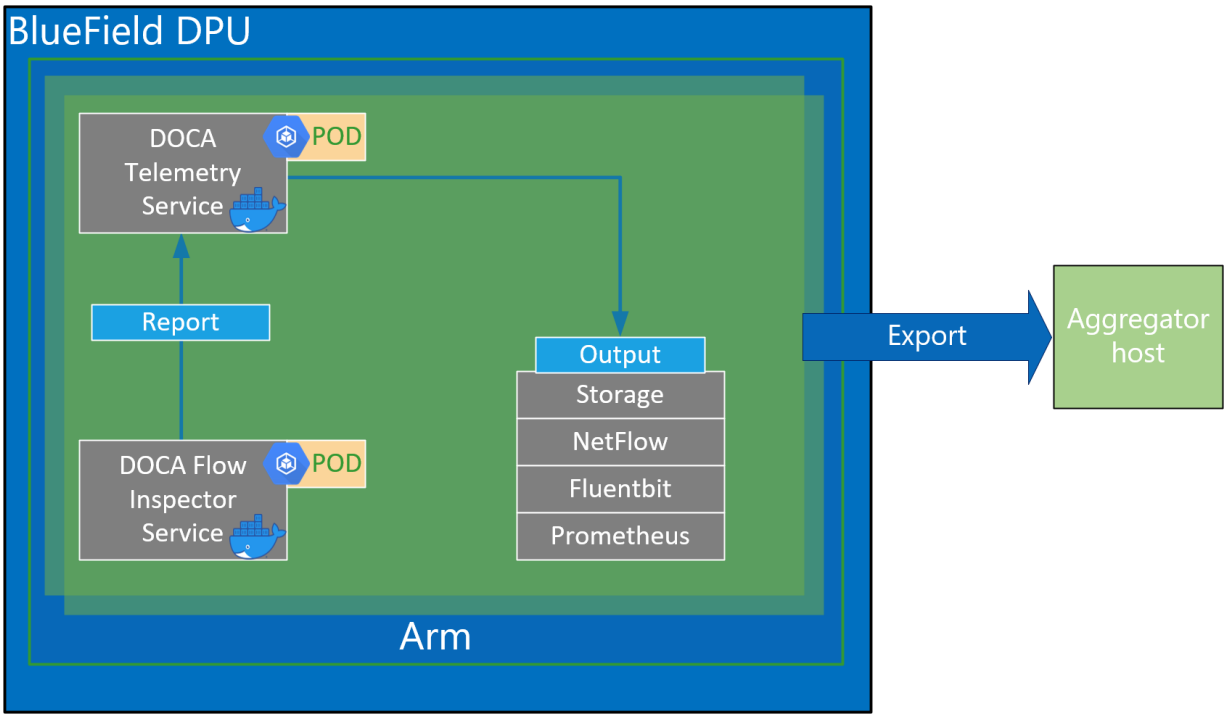
Table of Contents

Chapter 1. Introduction

DOCA Flow Inspector service allows monitoring real-time data and extraction of telemetry components which can be utilized by various services for security, big data, and other purposes.

DOCA Flow Inspector service is linked to DOCA Telemetry Service (DTS). DOCA Flow Inspector receives mirrored packets from the user and parses and forwards the data to the DTS which gathers predefined statistics forwarded by various providers/sources. DOCA Flow Inspector uses the DOCA Telemetry API to communicate with the DTS while the DPDK infrastructure allows acquiring packets at a user-space layer.

DOCA Flow Inspector runs inside of its own Kubernetes pod on BlueField and is intended to receive mirrored packets for analysis. The packets received are parsed and sent, in a predefined struct, to a telemetry collector which manages the rest of the telemetry aspects.



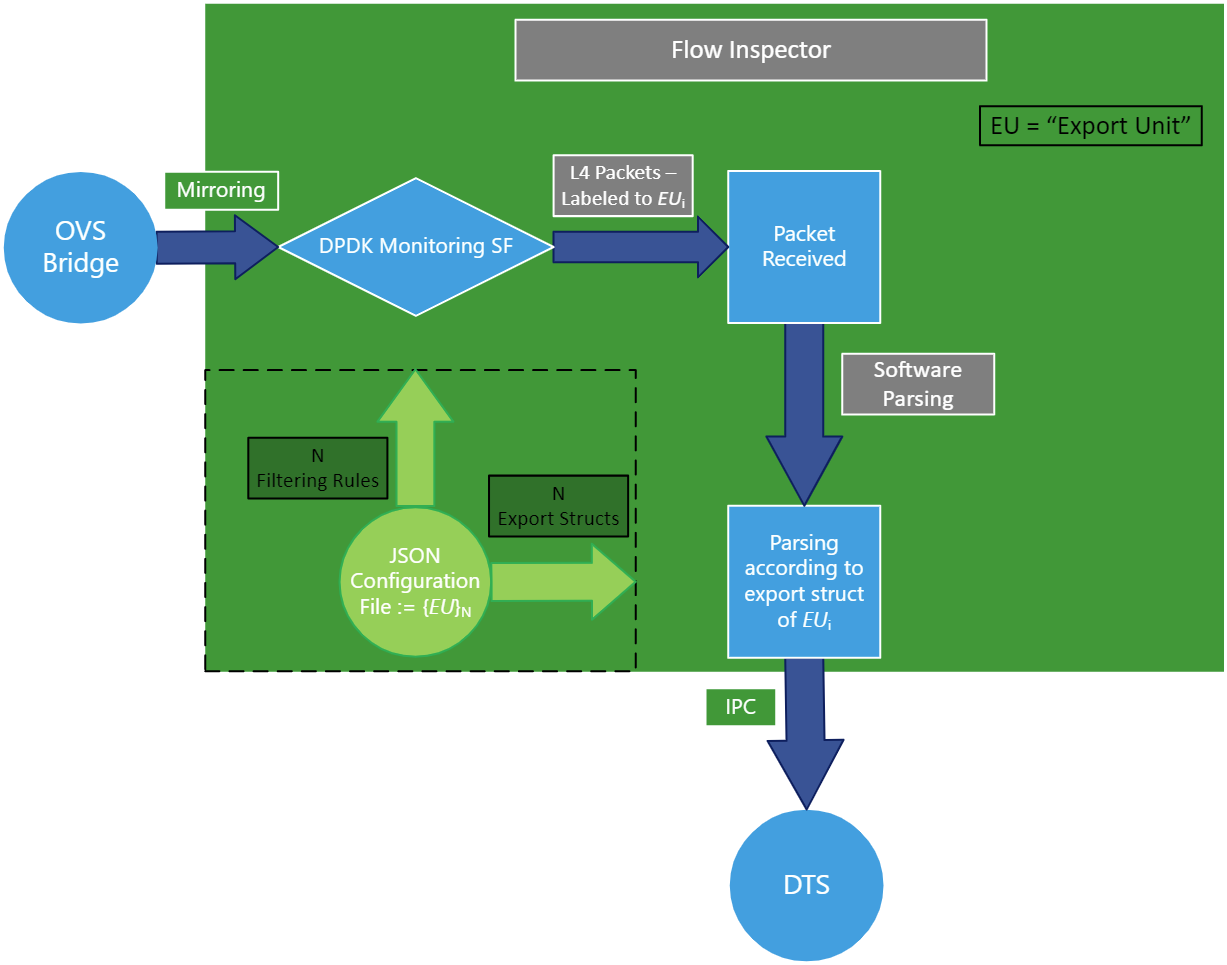
1.1. Service Flow

DOCA Flow Inspector receives a configuration file in a JSON format indicating which of the mirrored packets should be filtered out based on the L4 network header.

The configuration file can include several "export units". Each one is comprised of a "filter" and an "export". Each packet that matches one filter (based on the protocol and ports in the L4 header) is then parsed to the corresponding requested struct defined in the export. That information only is sent for inspection. A packet that does not match any filter is dropped. See JSON format and example in the Configuration section.

In addition, the service watches for changes in the JSON configuration file in runtime and for any change that reconfigures the service.

The DOCA Flow Inspector runs on top of DPDK to acquire L4. The packets are then filtered and HW-marked with their export unit index. The packets are then parsed according to their export unit and export struct, and then forwarded to the telemetry collector using IPC.



Configuration phase:

1. A JSON file is used as input to configure the export units (i.e., filters and corresponding export structs).
2. The filters are translated to HW rules on the SF (scalable function port) using the DOCA Flow library.
3. The connection to the telemetry collector is initialized and all export structures are registered to DTS.

Inspection phase:

1. Traffic is mirrored to the relevant SF.
2. Ingress traffic is received through the configured SF.
3. Non-L4 traffic and packets that do not match any filter are dropped using hardware rules.
4. Packets matching a filter are marked with the export unit index they match and are passed to the software layer in the Arm cores.
5. Packets are parsed to the desired struct by the index of export unit.
6. The telemetry information is forwarded to the telemetry agent using IPC.
7. Mirrored packets are freed.
8. If the JSON file is changed, run the configuration phase with the updated file.

Chapter 2. Requirements

DOCA Flow Inspector service must be used with DTS of the same DOCA version.

Before deploying the flow inspector container, ensure that the following prerequisites are satisfied:

1. Create the needed files and directories.

   Folders should be created automatically. Make sure the `.json` file resides inside the folder:

   ```
   $ touch /opt/mellanox/doca/services/flow_inspector/bin/flow_inspector_cfg.json
   ```

   Validate that DTS's configuration folders exist. They should be created automatically when DTS is deployed.

   ```
   $ 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
   ```

2. Allocate hugepages.

   Allocated huge pages as needed by DPDK. This requires root privileges.

   ```
   $ sudo echo 2048 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
   ```

   Or alternatively:

   ```
   $ sudo echo '2048' | sudo tee -a /sys/kernel/mm/hugepages/hugepages-2048kB/
   nr_hugepages
   $ sudo mkdir /mnt/huge
   $ sudo mount -t hugetlbfs nodev /mnt/huge
   ```

   Deploy a scalable function according to Scalable Function Setup Guide and mirror packets accordingly using the Open vSwitch command.

   For example:

   a). Mirror packets from `p0` to `sf0`:

   ```
   $ ovs-vsctl add-br ovsbr1
   $ ovs-vsctl add-port ovsbr1 p0
   $ ovs-vsctl add-port ovsbr1 en3f0pf0sf0
   $ ovs-vsctl -- --id=@p1 get port en3f0pf0sf0 \
               -- --id=@p2 get port p0 \
               -- --id=@m create mirror name=m0 select-dst-port=@p2 select-src-
   port=@p2 output-port=@p1 \
               -- set bridge ovsbr1 mirrors=@m
   ```

   b). Mirror packets from `pf0hpf` to `sf0`:

   ```
   $ ovs-vsctl add-br ovsbr1
   $ ovs-vsctl add-port ovsbr1 pf0hpf
   $ ovs-vsctl add-port ovsbr1 en3f0pf0sf0
   $ ovs-vsctl -- --id=@p1 get port en3f0pf0sf0 \
               -- --id=@p2 get port pf0hpf \
   ```

```
            -- --id=@m create mirror name=m0 select-dst-port=@p2 select-src-
port=@p2 output-port=@p1 \
            -- set bridge ovsbr1 mirrors=@m
```

The output of last command should be in the following format:

```
exp: 0d248ca8-66af-427c-b600-af1e286056e1
```

Note: The designated SF must be created as a trusted function. Additional details can be found in the Scalable Function Setup Guide.

Chapter 3. Service Deployment

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

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.

Chapter 4. Configuration

4.1. JSON Input

The flow inspector configuration file should be placed under `/opt/mellanox/doca/services/flow_inspector/bin/<json_file_name>.json` and be built in the following format:

```
[
    {
    # Export Unit:
        "filter":
        {
            "protocols": [<L4 protocols separated by comma>], # What L4 protocols
 are allowed
            "ports":                                          # What L4 port ranges
 are allowed [src,dst]
                [
                    [<source port>, <destination port>],
                    [<source ports range>, <destination ports range>],
                    <... more pairs of source, dest ports>
                ]
        },
        "export":
        {
            "fields": [<fields to be part of export struct, separated by comma>] #
 the Telemetry event will contain these fields.
        }
    },

    <... More Export Units>
]
```

Allowed protocols:

- `TCP`
- `UDP`

Port range:

- It is possible to insert a range of ports for both source and destination
- Range should include borders `[start_port-end_port]`

Allowed ports:

- All ports in range `0-65535` as a string

- Or * to indicate any port

Allowed fields in export struct:

timestamp
Timestamp indicating when it was received by the service.

host_ip
The IP of the host running the service

src_mac
Source MAC address

dst_mac
Destination MAC address

src_ip
Source IP

dst_ip
Destination IP

protocol
L4 protocol

src_port
Source port

dst_port
Destination port

flags
Additional flags (relevant to TCP only)

data_len
Data payload length

data_short
Short version of data (payload sliced to first 64 bytes)

data_medium
Medium version of data (payload sliced to first 1500 bytes)

data_long
Long version of data (payload sliced to first 9*1024 bytes)

JSON example:

```
[
    /* Export Unit 0 */
    {          "filter":
        {              "protocols": ["udp"],             # What L4 protocols are
 allowed
               "ports":                                  # What L4 port ranges are
 allowed
                 [
                     ["*","433-460"],                    # In this case, "*" stands
 for all source ports and dst port in range of [433-460]
                     ["20480","28341"]                   # src port 20480, dst port
 28341
                 ]
        },          "export":
        {             "fields": ["timestamp", "src_mac", "dst_mac", "protocol",
 "data_len", "data_long"]
        }
    },

    /* Export Unit 1 */
    {          "filter":
        {              "protocols": ["tcp"],
               "ports":
```

```
                [
                    ["5-10","422"]
                ]
        },          "export":
        {              "fields": ["timestamp", "host_ip", "src_ip", "dst_ip",
 "data_len", "data_short"]
        }
    }
]
```

Note: If a packet contains L4 ports which are not specified in the file, it is filtered out.

Note: The JSON file can be changed during runtime. The new configuration is applied in less than 60 seconds. This time period can be changed using the `-t` flag (see section Yaml File).

4.2. Yaml File

The `.yaml` file downloaded from NGC can be easily edited according to your needs.

```
env:
  # Set according to the local setup
  - name: SF_NUM_1
    value: "2"
  # Additional EAL flags, if needed
  - name: EAL_FLAGS
    value: ""
  # Service-Specific command line arguments
  - name: SERVICE_ARGS
    value: "--policy /flow_inspector/flow_inspector_cfg.json -l 60"
```

- The `SF_NUM_1` value can be changed according to the SF used in the OVS configuration and can be found using the command in Scalable Function Setup Guide.
- The `EAL_FLAGS` value must be changed according to the DPDK flags required when running the container.
- The `SERVICE_ARGS` are the runtime arguments received by the service:
  - `-l, --log-level <value>` – sets the log level (CRITICAL=20, ERROR=30, WARNING=40, INFO=50, DEBUG=60)
  - `-p, --policy <json_path>` – sets the JSON path inside the container
  - `-t, --time <seconds>` – time period to check for changes in JSON config file (default is 60 seconds)

4.3. Verifying Output

Enabling write to data in the DTS allows debugging the validity of DOCA Flow Inspector.

Uncomment the following line in `/opt/mellanox/doca/services/telemetry/config/dts_config.ini`:

```
#output=/data
```

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

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:

Note: Requires installing the `tree` runtime utility (`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 appear when:

- The service starts
- When the binary file's max age/size restriction is reached
- When JSON file is changed and new schemas of telemetry are created
- An hour passes

If no schema or no data folders are present, refer to the Troubleshooting section in the NVIDIA DOCA Telemetry Service Guide.

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
```

The data written locally should be shown in the following format assuming a packet matching Export Unit 1 from the example has arrived:

```
{
    "timestamp": 1656427771076130,
    "host_ip": "10.237.69.238",
    "src_ip": "11.7.62.4",
    "dst_ip": "11.7.62.5",
    "data_len": 1152,
    "data_short": "Hello World"
}
```

Chapter 5. Troubleshooting

Refer to the Troubleshootings section in NVIDIA DOCA Container Deployment Guide and NVIDIA DOCA Telemetry Service Guide.

Additional notes:

- Make sure hugepages are allocated (step 2 under Requirements)
- Validate the JSON file and port config
- When running both containers, you must first run DOCA Telemetry Service, wait a few seconds, and then run DOCA Flow Inspector.

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