NVIDIA DOCA gRPC Infrastructure
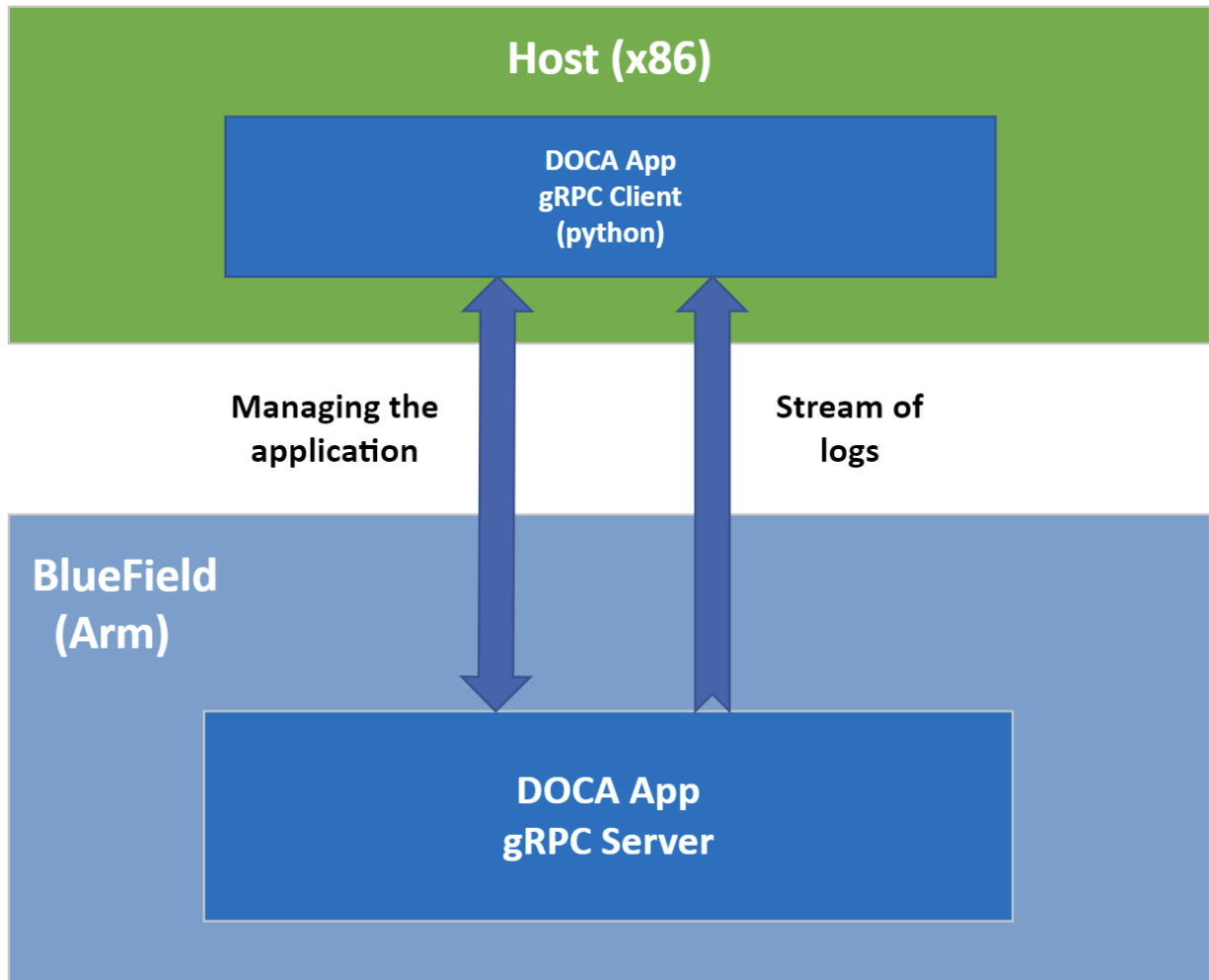
User Guide

Table of Contents

Chapter 1.   Introduction

The recommended setup for deploying DOCA applications and services is deployment on the DPU itself. However, in some cases it is useful to be able to manage and configure the applications running on top of it directly from the host (x86).

For this purpose, DOCA now includes built-in gRPC support, thereby exposing to the host the application's logical interface in the form of a gRPC-equivalent API.

This guide elaborates on the different components that enable this support, including instructions for developers who wish to modify the gRPC support of the example applications.

Chapter 2. Prerequisites

▶ Refer to the DOCA Installation Guide for details on how to install BlueField related software
▶ BlueField OS version required is 3.8.0 and higher (Ubuntu 20.04)

Chapter 3. gRPC Overview

gRPC#is Google's open-source remote procedure call (RPC) library and is the most widely used RPC solution.

gRPC#consists of two layers:

► Protobuf - Google library for semantically defining message formats

► gRPC "services" - definitions of the exposed RPC functionality

gRPC's support for different language bindings, combined with the unified protocol implementation, allows the client and server to run on different machines using different programming languages.

Chapter 4. OVS and Connectivity

It is important to differentiate between and separate the application's data path and the gRPC management interface. The following figure contains an overview of a sample setup for a bump-on-the wire application:
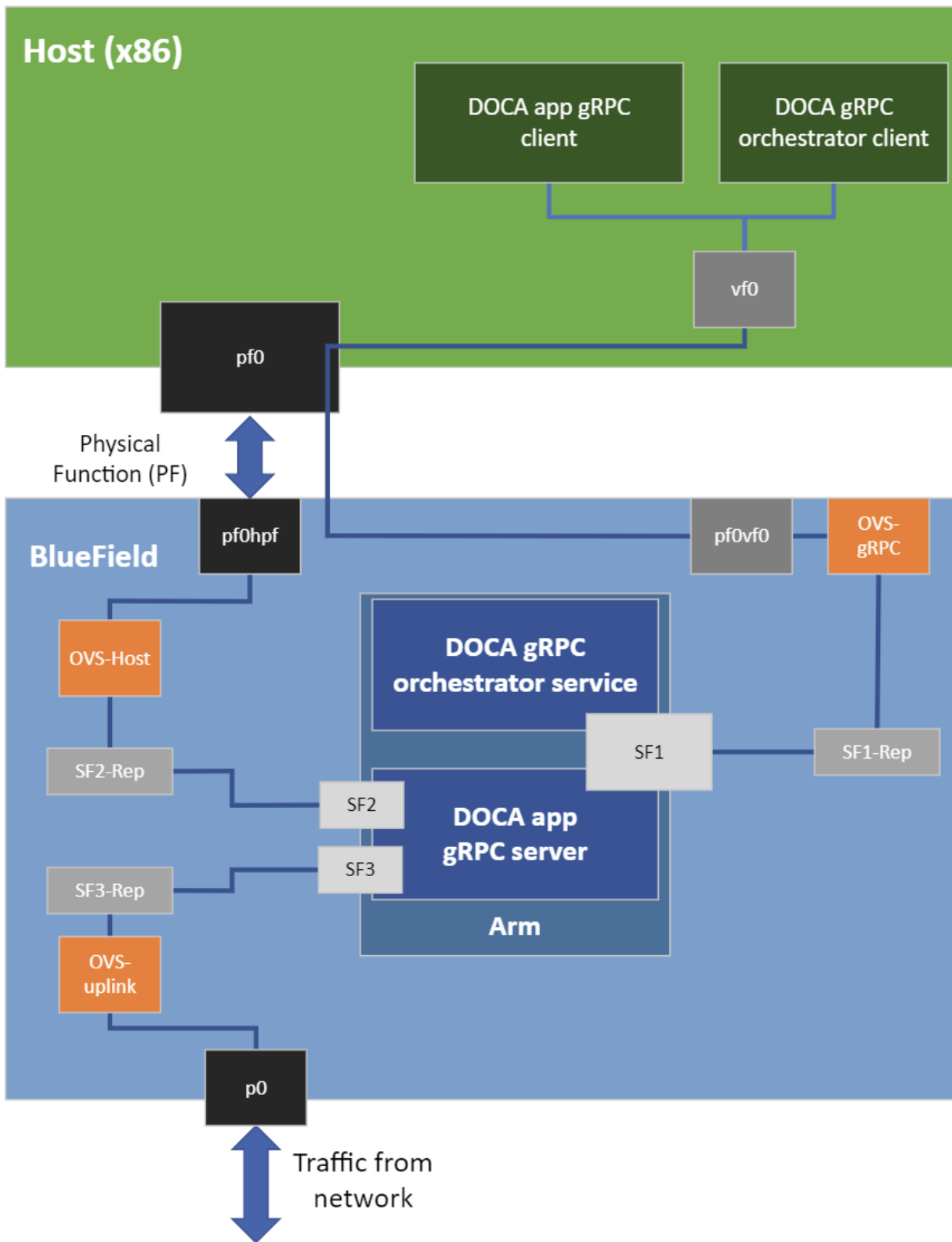
As can be seen above, the application is a bump-on-the-wire, and the gRPC-related traffic flows through a separate OVS bridge connected to the host using a virtual function (VF). More

information about VFs and how to configure them can be found in the NVIDIA DOCA Virtual Functions User Guide.

The architecture above allows us to associate a network address to SF1, effectively making the application's gRPC server part of an IP network with the host. This allows for an easy client-server setup, masking away the hardware details from the logical gRPC interface.

Chapter 5. gRPC-Enabled DOCA Applications

The process of adding gRPC-support for an existing DOCA application focuses on a single goal: Extracting the logical (management) interface exposed by the application and converting it to an equivalent `.proto` definition.

```
/opt/mellanox/doca/examples/<app_name>/src/grpc/<app_name>.proto
```

Once extracted, we define a simple wrapper layer that translates the incoming gRPC commands to the C API of the application and translates back the results to the equivalent gRPC messages to be sent back to the Python client running on the host.

Together with simple argument parsing, this binding layer serves as the main file of the application's gRPC server:

```
/opt/mellanox/doca/examples/<app_name>/src/grpc/server/<app_name>_gRPC_server.cc
```

The updated folder structure for a gRPC-enabled application is the following:

```
+ /opt/mellanox/doca
+-+ examples
  |   ...
  +-+ <app_name>                  <== gRPC-Enabled Application
  | +-+ bin
  | | +-- doca_<app_name>         <== "Vanilla" Application, without gRPC support
  | | +-+ grpc                    <== New directory for gRPC
  | |   +-+ client                <== gRPC Client, to be used from the host (x86)
  | |   | +-- doca_<app_name>_gRPC_client.py
  | |   | +-- *.py
  | |   +-+ server                <== gRPC Server, to be used from the DPU (arm)
  | |     +-- doca_<app_name>_grpc  <== gRPC-Enabled Application
  | +-+ src
  |   +-- ...
  |   +-- meson_options.txt        <== Activate/Disable gRPC compilation support
  |   +-+ grpc                     <== New directory for gRPC
  |     +-- <app_name>.proto
  |     +-- meson.build
  |     +-+ client
  |     | +-- doca_<app_name>_gRPC_client.py  <== Application gRPC Client source
  |     | +-- meson.build
  |     +-+ server
  |       +-- *.h/*.c
  |       +-- <app_name>_gRPC_server.cc       <== Application gRPC server source
  |       +-- meson.build
  +-+ common
    +-+ src
      |   ...
      +-+ grpc                     <== New directory for common gRPC definitions
        +-- common.proto
```

Chapter 6. Compilation Instructions

A gRPC-enabled application consists of two newly added folders within the `grpc` directory:

▶ `client`

▶ `server`

Rebuilding the server and the client must be performed in the same environment to ensure they both use the same .proto file. We recommend performing this compilation on the BlueField, where the server is later used.

The server and the client are compiled as part of the default application compilation from the application's src directory.

As recompiling the gRPC-enabled application requires a gRPC development setup, there is a `meson_options.txt` file that controls the gRPC support and that is set to "off" by default.
```
option('grpc_support', type : 'boolean', value : false)
```

Once a gRPC setup is installed, according to the instructions in the next section, the `meson_options.txt` file can be updated to enable gRPC support so as to allow for recompilation of the gRPC-enabled application.
```
option('grpc_support', type : 'boolean', value : true)
```

The compiled gRPC-enabled application is created under the `grpc/server` and `grpc/client` directories, inside the configured build directory (usually `/tmp/build`).

6.1. Installing gRPC Development Setup

Rebuilding a gRPC-enabled DOCA application on BlueField requires a gRPC development setup. A default such setup is automatically installed as part of all DOCA development packages that embed gRPC support. Due to the lack of packaging support for gRPC, at least for C/C++ environments, gRPC is compiled from the source and can be found at `/opt/mellanox/grpc`.

Upon compilation, Meson will make sure that the requirements exist. If some environment variables were somehow corrupted, they can be restored using the following commands:
```
export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/opt/mellanox/grpc/lib/pkgconfig
export PATH=$PATH:/opt/mellanox/grpc/bin
```

If a different gRPC version is needed, these are Google's instructions for creating a development setup. If a different gRPC setup is used, do not forget to update the above environment variables to point at your installation directory.

Chapter 7. Running gRPC-Enabled Application

7.1. Running Application Server on BlueField

The gRPC-enabled application is equivalent to the "regular" DOCA application as it:

▶ Requires the same configuration steps (huge pages, etc.)

▶ Requires the same application command line arguments

The only difference is that the application also requires one more command line argument:
```
doca_<app_name>_grpc [DPDK flags] -- [DOCA flags] [Program flags] --grpc-address
 <ip-address[:port]>
```

▶ `ip-address` – IP address to be used by the gRPC server

▶ `port` – TCP port to be used by the server instead of the application's default gRPC port (optional)

One could also use the json configuration file as follows:
```
doca_<app_name>_grpc -j/--json <path to grpc configuration json file>
```

For more information, refer to NVIDIA DOCA Arg Parser User Guide.

7.2. Running Application Client on Host

While the gRPC Python environment is already installed on the host as part of the DOCA installation, it has not been added to the default Python path so as to not clutter it. The environment variable definitions needed for using the gRPC Python environment, as needed by the client, are:
```
export PYTHONPATH=${PYTHONPATH}:/opt/mellanox/grpc/python3/lib
```

To run the Python client of the gRPC-enabled application:
```
doca_<app_name>_gRPC_client.py -d/--debug <server address[:server port]>
```

7.3. Installing gRPC on RHEL CentOS 7.6 and Older

On RHEL/CentOS distributions 7.6 and older for x64 host, there is a known issue in the python `grpcio` package which causes the following error:

```
from grpc._cython import cygrpc as _cygrpc
ImportError: /opt/mellanox/grpc/python3/lib/grpc/_cython/cygrpc.cpython-36m-x86_64-
linux-gnu.so: undefined symbol: _ZSt24__throw_out_of_range_fmtPKcz
```

To fix this error, please run the following commands on the host:

```
rm -rf /opt/mellanox/grpc/python3/lib/grpc*
wget https://files.pythonhosted.org/
packages/67/3c/53cc28f04fb9bd3e8bad6fa603aa8b01fa399dd74601ab0991e6385dbfe4/
grpcio-1.39.0-cp36-cp36m-manylinux2010_x86_64.whl -P /tmp && unzip /tmp/
grpcio-1.39.0-cp36-cp36m-manylinux2010_x86_64.whl -d /opt/mellanox/grpc/python3/lib
```

Chapter 8. DOCA gRPC Service

A gRPC-enabled application must first be executed on BlueField for it to be managed from the host. This creates a bootstrapping issue that is solved by the DOCA gRPC service, as can be seen in the following figure:
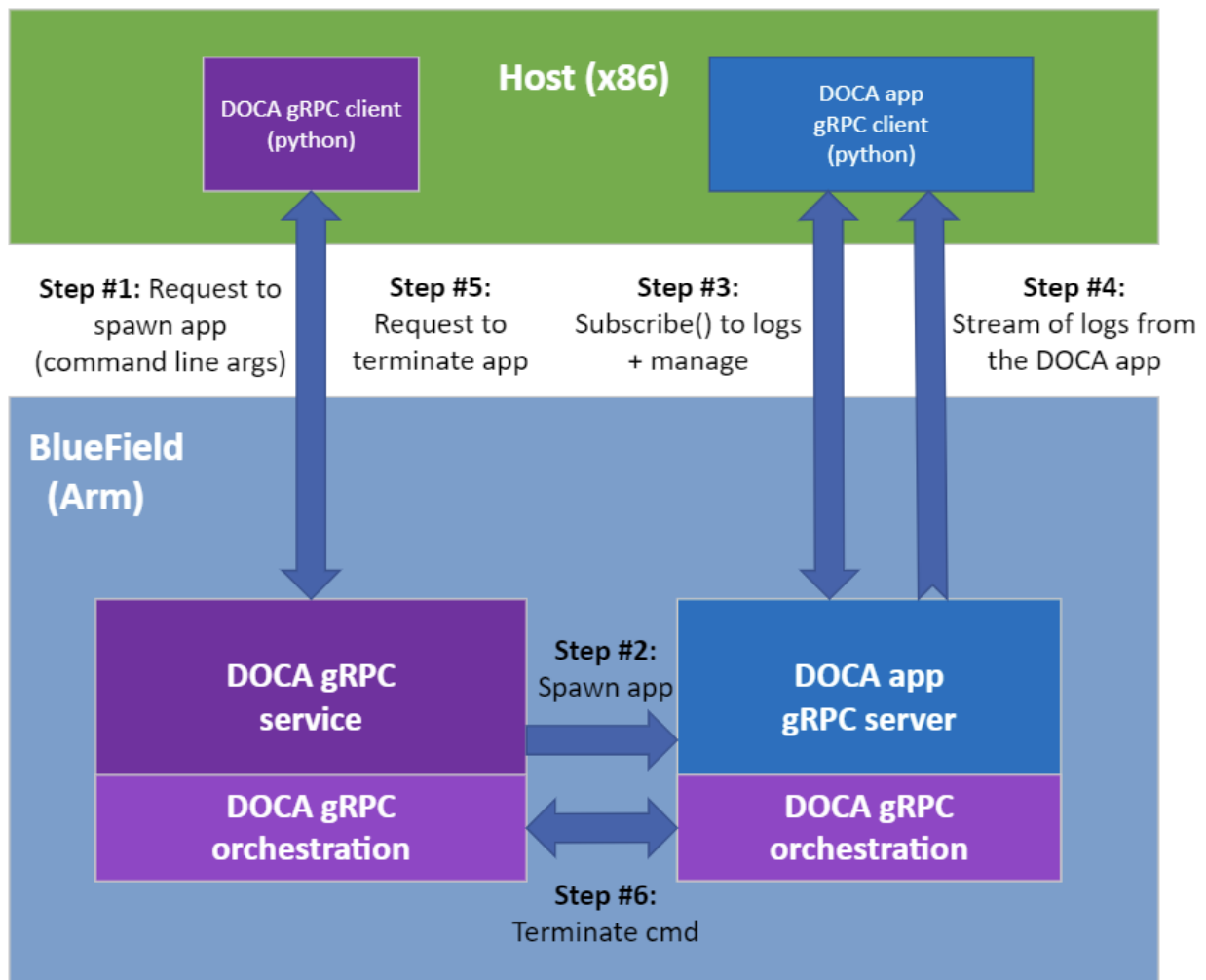
After a one-time configuration step, the DOCA gRPC service runs on BlueField, listening for incoming requests from the host to start/stop a given gRPC-enabled DOCA application/service.

8.1. Enabling and Configuring DOCA gRPC Service

The `doca_grpc` service on the DPU starts as "disabled" by default and has a one-time configuration step for enabling it:

```
# One-time only, enable the DOCA gRPC service
systemctl enable doca_grpc.service
# One-time only, start the service
systemctl start doca_grpc.service
```

The service is controlled via a configuration file stored at `/etc/doca_grpc/doca_grpc.conf`.

This file comes prepopulated with a list of all gRPC-enabled DOCA programs and can be modified to support additional proprietary programs. The file also defines the configurations for every DOCA gRPC server that the service spawns alongside the list of programs it exposes to the host.

Once the configuration file is modified to suit the requested deployment, the service needs to be restarted so it could pull the new configuration:

```
systemctl restart doca_grpc.service
```

8.2. Running DOCA gRPC Client

The DOCA gRPC client is located at `/opt/mellanox/doca/infrastructure/doca_grpc/orchestrator`.

> 📝 **Note:** Being a Python client, the same Python environment variable mentioned earlier for using the application's gRPC client is needed for this client as well.

The usage instructions for the DOCA gRPC client are:

```
Usage: doca_grpc_client.py [OPTIONS] SERVER_ADDRESS COMMAND [ARGS]...
  DOCA gRPC Client CLI tool
Options:
  -d, --debug
  --help       Show this message and exit.

Commands:
  create   Create PROGRAM_NAME [PROGRAM_ARGS]...
  destroy  Destroy PROGRAM_UID Terminate the execution of the program...
  list     List the names of gRPC-supported program.
```

For example:

```
/opt/mellanox/doca/infrastructure/doca_grpc/orchestrator/doca_grpc_client.py
 192.168.103.2:1234 list
```

The supported commands are:

▶ list – prints a list of names of all DOCA gRPC-enabled programs currently supported

▶ create – spawns a gRPC-enabled program on the DPU based on its name and arguments

▶ destroy – terminates the execution of a gRPC-enabled program based on the program's UID as returned from the "create" command

The SERVER_ADDRESS argument is of the form `<server address[:server port]>` allowing the client to use a TCP port other than the default one if the server uses a proprietary port.

The command-specific options are shown when passing the `--help` flag to the respective command:

```
/opt/mellanox/doca/infrastructure/doca_grpc/orchestrator/doca_grpc_client.py
 192.168.103.2 destroy --help
```

> 📝 **Note:** The `create` command expects the same list of arguments as described under a particular application's page in the Reference Applications section of the DOCA SDK Documentation. Taking the NVIDIA DOCA URL Filter Reference App as an example, the command will be:
>
> ```
> /opt/mellanox/doca/infrastructure/doca_grpc/orchestrator/doca_grpc_client.py
>  192.168.103.2 create doca_url_filter -a 0000:03:00.0,class=regex -a
>  auxiliary:mlx5_core.sf.4,sft_en=1 -a auxiliary:mlx5_core.sf.5,sft_en=1 -c3 --
>  -p
> ```

The gRPC-specific command line arguments (listed under Running gRPC-Enabled Application) are only needed when directly invoking the gRPC-enabled application. When invoked through the DOCA gRPC client, the arguments must match those of the "regular" DOCA application.

> 📝 **Note:** The orchestrator client supports the option to spawn a gRPC-enabled program using a non-default port. This option is mandatory for programs that do not support a default gRPC port.
>
> For example, the following command line will spawn the same URL Filter application shown earlier, but this time using the network port 1234:
>
> ```
> /opt/mellanox/doca/infrastructure/doca_grpc/orchestrator/doca_grpc_client.py
>  192.168.103.2 create -p 1234 doca_url_filter -a 0000:03:00.0,class=regex -a
>  auxiliary:mlx5_core.sf.4,sft_en=1 -a auxiliary:mlx5_core.sf.5,sft_en=1 -c3 --
>  -p
> ```

8.3. Running gRPC-Enabled DOCA Libs

All servers for gRPC-enabled DOCA libraries accept the following arguments:

```
Usage: doca_<lib_name>_grpc [DPDK Flags] -- [DOCA Flags]
DOCA Flags:
  -h, --help                             Print a help synopsis
  -l, --log-level                        Set the log level for the program
CRITICAL=0, DEBUG=4>
  --grpc-address ip_address[:port]       Set the IP address for the grpc server
```

Therefore, they should be invoked using the orchestrator client as follows:

```
/opt/mellanox/doca/infrastructure/doca_grpc/orchestrator/doca_grpc_client.py
 192.168.103.2 create doca_dpi_grpc -a 0000:03:00.0,class=regex -a
 auxiliary:mlx5_core.sf.4,sft_en=1 -- -l 3
```

8.4.   Running DOCA Application gRPC Client

Once the application's gRPC server is spawned on BlueField, you can connect to it directly from the host using the respective gRPC client.

If the application's gRPC server is configured to use a TCP port that is not the default port of the application, please remember to also configure the gRPC client to use the same non-default port.

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