Table of contents

Introduction

Developing Using BlueField Networking Platform
  Setup
  Development
  Testing
  Publishing

Developing Without BlueField Networking Platform
  Setup
  Development
  Testing
  Publishing

Cross-compilation from Host
  Setup
  DOCA and CUDA Setup
  Development
  Testing
  Publishing
This guide details the recommended steps to set up an NVIDIA DOCA development environment.

**Introduction**

This guide is intended for software developers aiming to modify existing NVIDIA DOCA applications or develop their own DOCA-based software.

Instructions for installing DOCA on the NVIDIA® BlueField® Networking Platform (i.e., DPU or SuperNIC) can be found in the *[NVIDIA DOCA Installation Guide for Linux]*.

This guide focuses on the recommended flow for developing DOCA-based software, and will address the following scenarios:

- BlueField is accessible and can be used during the development and testing process
  - Working within a development container
- BlueField is inaccessible, and the development happens on the host or on a different server
  - Cross-compilation from the host
  - Working within a development container on top of QEMU running on the host

It is recommended to follow the instructions for the first scenario, leveraging BlueField during the development and testing process.

This guide recommends using DOCA's development container during the development process on BlueField Platforms or on the host. Deploying development containers allows multiple developers to work simultaneously on the same device (host or BlueField Platform) in an isolated manner and even across multiple different DOCA SDK versions. This can allow multiple developers to work on the BlueField Platform itself, for example, without needing to have a dedicated BlueField per developer.

Another benefit of this container-based approach is that the development container allows developers to create and test their DOCA-based software in a user-friendly environment that comes pre-shipped with a set of handy development tools. The development container is focused on improving the development experience and is
designed for that purpose, whereas the BlueField software is meant to be an efficient runtime environment for DOCA products.

![Diagram of Docker setup]

### Developing Using BlueField Networking Platform

#### Setup

DOCA’s base image containers include a DOCA development container for the BlueField (doca:devel) which can be found on NGC. It is recommended to deploy this container on top of BlueField when preparing a development setup.

The recommended approach for working using DOCA's development container on top of the BlueField, is by using `docker`, which is already included in the supplied BFB image.

1. Make sure the docker service is started. Run:

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.
2. Pull the container image:

1. Visit the [NGC page](https://ngc.nvidia.com) of the DOCA base image.

2. Under the "Tags" menu, select the desired development tag for BlueField.

3. The container tag for the docker `pull` command is copied to your clipboard once selected. Example docker `pull` command using the selected tag:

   ```bash
   sudo docker pull nvcr.io/nvidia/doca/doca:1.5.1-devel
   ```

3. Once loaded locally, you may find the image's ID using the following command:

   ```bash
   sudo docker images
   ```

   Example output:

   ```
   REPOSITORY        TAG          IMAGE ID       CREATED             SIZE
   nvcr.io/nvidia/doca 1.5.1-devel 931bd576eb49 10 months ago 1.49GB
   ```

4. Run the docker image:

   ```bash
   sudo docker run -v <source-code-folder>:/doca_devel -v /dev/hugepages:/dev/hugepages --privileged --net=host -it <image-name/ID>
   ```

   For example, to map a source folder named `my_sources` into the same container tag from the example above, the command should look like this:
After running the command, you get a shell inside the container where you can build your project using the regular build commands:

- From the container's perspective, the mounted folder will be named `/doca_devel`

**Note**

Make sure to map a folder with write privileges to everyone. Otherwise, the docker would not be able to write the output files to it.

- `--net=host` ensures the container has network access, including visibility to SFs and VFs as allocated on BlueField

- `-v /dev/hugepages:/dev/hugepages` ensures that allocated huge pages are accessible to the container

**Development**

It is recommended to do the development within the `doca:devel` container. That said, some developers prefer different integrated development environments (IDEs) or development tools, and sometimes prefer working using a graphical IDE until it is time to compile the code. As such, the recommendation is to mount a network share to BlueField (refer to NVIDIA DOCA DPU CLI for more information) and to the container.

**Note**

Having the same code folder accessible from the IDE and the container helps prevent edge cases where the compilation fails due
Testing

The container is marked as "privileged", hence it can directly access the hardware capabilities of the BlueField Platform. This means that once the tested program compiles successfully, it can be directly tested from within the container without the need to copy it to BlueField and running it there.

Publishing

Once the program passes the testing phase, it should be prepared for deployment. While some proof-of-concept (POC) programs are just copied "as-is" in their binary form, most deployments will probably be in the form of a package (.deb/.rpm) or a container.

Construction of the binary package can be done as-is inside the current doca:devel container, or as part of a CI pipeline that will leverage the same development container as part of it.

For the construction of a container to ship the developed software, it is recommended to use a multi-staged build that ships the software on top of the runtime-oriented DOCA base images:

- doca:base-rt – slim DOCA runtime environment
- doca:full-rt – full DOCA runtime environment similar to the BlueField image

The runtime DOCA base images, alongside more details about their structure, can be found under the same NGC page that hosts the doca:devel image.

For a multi-staged build, it is recommended to compile the software inside the doca:devel container, and later copy it to one of the runtime container images. All relevant images must be pulled directly from NGC (using docker pull) to the container registry of BlueField.
Developing Without BlueField Networking Platform

If the development process needs to be done without access to a BlueField Platform, the recommendation is to use a QEMU-based deployment of a container on top of a regular x86 server. The development container for the host will be the same `doca:devel` image we mentioned previously.

Setup

1. Make sure Docker is installed on your host. Run:

   ```bash
docker version
   ```

   If it is not installed, visit the official [Install Docker Engine] webpage for installation instructions.

2. Install QEMU on the host.
Note

This step is for x86 hosts only. If you are working on an aarch64 host, move to the next step.

<table>
<thead>
<tr>
<th>Host OS</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu</td>
<td>sudo apt-get install qemu binfmt-support qemu-user-\texttt{static}</td>
</tr>
<tr>
<td></td>
<td>sudo docker run --rm --privileged multiarch/qemu-user-\texttt{static}</td>
</tr>
<tr>
<td></td>
<td>--reset -p yes</td>
</tr>
<tr>
<td></td>
<td>echo &quot;:qemu-aarch64:M::\x7fELF\x02\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02\x01\x00\x00\x02\x01\x07:xff\xff\xff\xff&quot;</td>
</tr>
<tr>
<td>CentOS/RHEL 7.x</td>
<td>sudo yum install epel-release</td>
</tr>
<tr>
<td></td>
<td>sudo yum install qemu-system-arm</td>
</tr>
<tr>
<td>CentOS 8.0/8.2</td>
<td>sudo yum install epel-release</td>
</tr>
<tr>
<td></td>
<td>sudo yum install qemu-kvm</td>
</tr>
<tr>
<td>Fedora</td>
<td>sudo yum install qemu-system-aarch64</td>
</tr>
</tbody>
</table>

3. If you are using CentOS or Fedora on the host, verify if `qemu-aarch64.conf` Run:

```bash
cat /etc/binfmt.d/qemu-aarch64.conf
```

If it is missing, run:

```bash
echo ":qemu-aarch64\:M:\:\x7fELF\x02\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02\x01\x00\x00\x02\x01\x07:xff\xff\xff\xff"
```
4. If you are using CentOS or Fedora on the host, restart system binfmt. Run:

$ sudo systemctl restart systemd-binfmt

5. To load and execute the development container, refer to the "Setup" section discussing the same docker-based deployment on the BlueField side.

Note

The `doca:devel` container supports multiple architectures. Therefore, Docker by default attempts to pull the one matching that of the current machine (i.e., `amd64` for the host and `arm64` for BlueField). Pulling the `arm64` container from the x86 host can be done by adding the flag `--platform=linux/arm64`:

```
sudo docker pull --platform=linux/arm64 nvcr.io/nvidia/doca/doca:1.5.1-devel
```

Development

Much like the development phase using a BlueField DPU, it is recommended to develop within the container running on top of QEMU.

Testing

While the compilation can be performed on top of the container, testing the compiled software must be done on top of a BlueField Platform. This is because the QEMU
environment emulates an aarch64 architecture, but it does not emulate the hardware devices present on the BlueField Platform. Therefore, the tested program will not be able to access the devices needed for its successful execution, thus mandating that the testing is done on top of a physical BlueField.

Note

Make sure that the DOCA version used for compilation is the same as the version installed on BlueField used for testing.

Publishing

The publishing process is identical to the publishing process when using a BlueField DPU.

Cross-compilation from Host

In a typical setup, developers prefer to work on a familiar host since compilation is often significantly faster there. Therefore, developers may work on their host while cross-compiling their project to BlueField's Arm architecture.
Setup

1. Install Docker and QEMU your host. See steps 1-4 under section Setup.

2. Download the doca-cross component as described in the NVIDIA DOCA Installation Guide for Linux and unpack it under the /root directory.

   Inside this directory one can find:

   - arm64_armv8_linux_gcc – cross file containing specific information about the cross compiler and the host machine
   - DOCA_cross.sh – script which handles all the required dependencies and pre-installations steps
   - A .txt file used by the script

3. To load the development container, refer to section "Docker Deployment" of the NVIDIA BlueField Container Deployment Guide.

Note

It is important to ensure that the same DOCA version is used in the development container and the DOCA metapackages
4. Start running the container using the container's tag while mapping the `doca-cross` directory to the container's `/doca_devel` directory:

```
sudo docker run -v /root/doca-cross:/doca_devel --privileged -it <imagename/ID>
```

Now the shell will be redirected to be within the container.

5. Run the preparation script to copy all the Arm dependencies required for DOCA's cross compilation. The script will be in the mapped directory named `doca_devel`.

```
(container) /# cd doca_devel/
(contaier) /doca_devel# ./DOCA_cross.sh
```

6. Exit the container and run the same script from the host side:

```
(host) /root/doca-cross# ./DOCA_cross.sh
```

The `/root/doca-cross` directory is now fully configured and prepared for cross-compilation against DOCA.

7. Update the environment variables to point at the Linaro cross-compiler:

```
export PATH=${PATH}:/opt/gcc-linaro/<linaro_version_dir>/aarch64-linux-gnu/bin:/opt/gcc-linaro/<linaro_version_dir>/bin
```

Everything is set up and the cross-compilation can now be used.

**Note**
DOCA and CUDA Setup

1. To cross-compile DOCA and CUDA applications, you must install CUDA Toolkit 12.1:
   1. The first toolkit installation is for x86 architecture. Select x86_64.
   2. The second toolkit installation is for Arm. Select arm64-sbsa and then cross.
   3. Select your host operating system, architecture, OS distribution, and version and select the installation type. It is recommended to use the deb (local) type.

2. Execute the following exports:

   ```
   export CPATH=/usr/local/cuda/targets/sbsa-linux/include:$CPATH
   export LD_LIBRARY_PATH=/usr/local/cuda/targets/sbsa-linux/lib:$LD_LIBRARY_PATH
   export PATH=/usr/local/cuda/bin:/usr/local/cuda-11.6/bin:$PATH
   ```

3. Verify the meson version is at least 0.61.2 as provided with DOCA's installation.

   Everything is set up and the cross-compilation can now be used.
Development

It is recommended to develop normally while remembering to compile using the cross-compilation configuration file `arm64_armv8_linux_gcc` which can be found under the `doca-cross` directory.

The following is an example procedure for cross-compiling DOCA applications from the host and to the Arm architecture:

1. Enable the meson cross-compilation option in
   `/opt/mellanox/doca/applications/meson_options.txt` by setting `enable_cross_compilation_to_dpu` to `true`.

2. Cross-compile the DOCA applications:

   ```bash
   /opt/mellanox/doca/applications # meson cross-build --cross-file /root/doca-cross/arm64_armv8_linux_gcc
   /opt/mellanox/doca/applications # ninja -C cross-build
   ```

   **Info**

   The cross-compiled binaries are created under the `cross-build` directory.

3. Cross-compile the DOCA and CUDA application:

   1. Set flag for GPU-enabled cross-compilation, `enable_gpu_support`, in
      `/opt/mellanox/doca/applications/meson_options.txt` to `true`.

   2. Run the compilation command as follows:

      ```bash
      /opt/mellanox/doca/applications # meson cross-build --cross-file /root/doca-cross/arm64_armv8_linux_gcc -Dcuda_ccbindir=aarch64-linux-gnu-g++
      ```
This definition, already provided as part of the supplied cross file, guarantees that meson does not accidently use the build system's environment variable during the cross build.

**Testing**

While the compilation can be performed on top of the host, testing the compiled software must be done on top of a BlueField Platform. This is because the tested program is not able to access the devices needed for its successful execution, which mandates that the testing is performed on top of a physical BlueField.

**Info**

The cross-compiled binaries are created under the cross-build directory.

**Note**

Due to the system's use of the PKG_CONFIG_PATH environment variable, it is crucial that the cross file include the following:

```plaintext
[built-in options]
pkg_config_path = ""
```

This definition, already provided as part of the supplied cross file, guarantees that meson does not accidently use the build system's environment variable during the cross build.
Make sure that the DOCA version used for compilation is the same as the version installed on the BlueField Platform used for testing.

Publishing

The publishing process is identical to the publishing process when using a BlueField DPU.