NVIDIA BlueField DPU Container Deployment Guide

User Guide

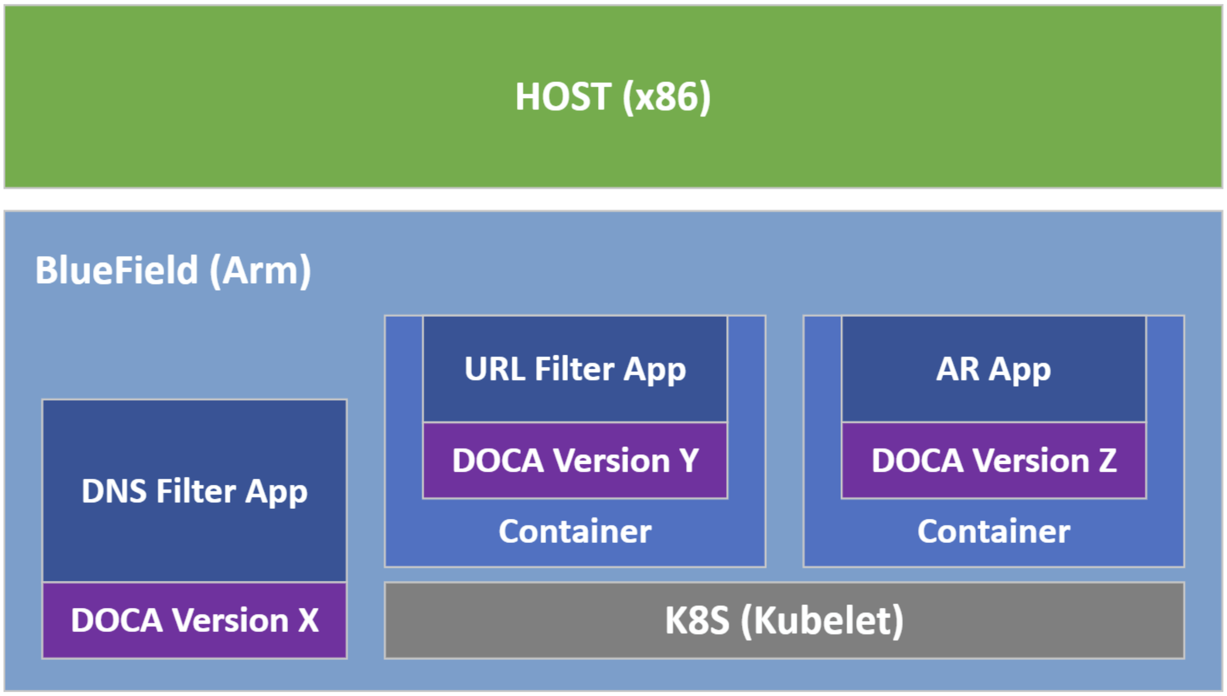
MLNX-15-060500 _v2.2.0 | October 2023

Table of Contents

Chapter 1. Introduction

DOCA containers allow for easy deployment of ready-made DOCA environments to the DPU, whether it is a DOCA service bundled inside a container and ready to be deployed, a DOCA application container to play with, or a development environment already containing the desired DOCA version.

Containerized environments enable the users to decouple DOCA programs from the underlying BlueField software. Each container is pre-built with all needed libraries and configurations to match the specific DOCA version of the program at hand. One only needs to pick the desired version of the application or service and pull the ready-made container of that version from NVIDIA's container catalog.



The different DOCA containers are listed on NGC, NVIDIA's container catalog, and can be found under both the "DOCA" and "DPU" labels.

Chapter 2. Prerequisites

- Refer to the NVIDIA DOCA Installation Guide for Linux for details on how to install BlueField related software
- BlueField image version required is 3.8.0 and higher

Note: Container deployment based on standalone Kubelet, as presented in this guide, is currently in alpha version and is subject to change in future releases.

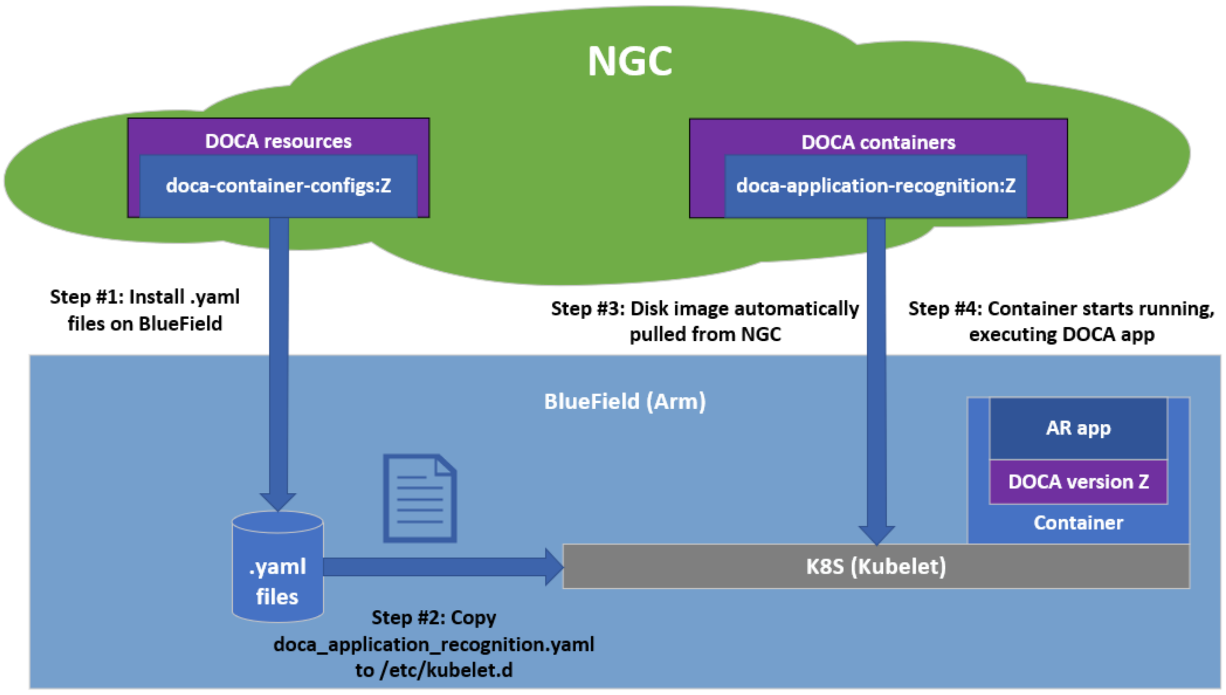
Chapter 3. Container Configuration

Deploying containers on top of the BlueField DPU requires the following setup sequence:

1. Pull the container `.yaml` configuration files.
2. Modify the container's `.yaml` configuration file.
3. Deploy the container. The image is automatically pulled from NGC.

Some of the steps only need to be performed once, while others are required before the deployment of each container.

What follows is an example of the overall setup sequence using the DOCA application recognition (AR) container as an example.


NGC
DOCA resources
doca-container-configs:Z
DOCA containers
doca-application-recognition:Z
Step #1: Install .yaml files on BlueField
Step #3: Disk image automatically pulled from NGC
Step #4: Container starts running, executing DOCA app
BlueField (Arm)
AR app
DOCA version Z
Container
.yaml files
K8S (Kubelet)
Step #2: Copy doca_application_recognition.yaml to /etc/kubelet.d

3.1. Activate Container-related Services on DPU

Note: This step is necessary only for BlueField OS versions prior to 3.9.0.

Note: This step should only be performed once per DPU.

Containers are deployed on the BlueField using Kubernetes (K8S) using the standalone Kubelet service.

To start and enable Kubelet and containerd, run:

```
systemctl start kubelet
systemctl start containerd
systemctl enable kubelet
systemctl enable containerd
```

3.2. Pull Container YAML Configurations

Note: This step pulls the `.yaml` configurations from NGC. If you have already performed this step for other DOCA containers you may skip to the next section.

Pulling the latest resource version can be done using the following command:

```
# Pull the entire resource as a *.zip file
wget --content-disposition https://api.ngc.nvidia.com/v2/resources/nvidia/doca/
doca_container_configs/versions/2.2.0v2/zip -O doca_container_configs_2.2.0v2.zip
# Unzip the resource
unzip -o doca_container_configs_2.2.0v2.zip -d doca_container_configs_2.2.0v2
```

More information about additional versions can be found in the NGC resource page.

The resource contains a `configs` directory, under which can be found a dedicated folder per DOCA version. For example, `1.5.1` will include all currently available `.yaml` configuration files for DOCA 1.5.1 containers.

3.3. Container-specific Instructions

Some containers require specific configuration steps for the resources used by the application running inside the container and modifications for the `.yaml` configuration file of the container itself.

Please refer to the container-specific instructions as listed under the container's respective page on NGC.

3.4. Structure of NGC Resource

The DOCA NGC resource downloaded in section Pull Container YAML Configurations contains a `configs` directory under which a dedicated folder per DOCA version is located. For example, `1.5.1` will include all currently available `.yaml` configuration files for DOCA 1.5.1 containers.

In addition, the resource also contains a `scripts` directory under which services may choose to provide additional helper-scripts and configuration files to use with their services.

The folder structure of the `scripts` directory is as follows:

```
+ doca_container_configs_2.0.2v1
+-+ configs
| +--  ...
+-+ scripts
  +-+ doca_firefly                <== Name of DOCA Service
  +-+ doca_hbn                    <== Name of DOCA Service
  | +-+ 1.3.0
  | | +-- ...                     <== Files for the DOCA HBN version "1.3.0"
  | +-+ 1.4.0
  | | +-- ...                     <== Files for the DOCA HBN version "1.4.0"
```

A user wishing to deploy an older version of the DOCA service would still have access to the suitable YAML file (per DOCA release under `configs`) and scripts (under the service-specific version folder which resides under `scripts`).

3.5. Spawn Container

Once the desired `.yaml` file is updated, simply copy the configuration file to Kubelet's input folder. Here is an example using the `doca_application_recognition.yaml`, corresponding to the DOCA AR application.

```
cp doca_application_recognition.yaml /etc/kubelet.d
```

Kubelet automatically pulls the container image from NGC and spawns a pod executing the container. In this example, the DOCA AR application starts executing right away and its printouts would be seen via the container's logs.

3.6. Stop Container

The recommended way to stop a pod and its containers is as follows:

1. Delete the `.yaml` configuration file for Kubelet to stop the pod:
   ```
   rm /etc/kubelet.d/<file name>.yaml
   ```
2. Stop the pod directly (only if it still shows "Ready"):
   ```
   crictl stopp <pod-id>
   ```
3. Once the pod stops, it may also be necessary to stop the container itself:
   ```
   crictl stop <container-id>
   ```

3.7. Useful Container Commands

- View currently active pods and their IDs (it might take up to 20 seconds for the pod to start):

```
crictl pods
```

- View currently active containers and their IDs:

```
crictl ps
```

- View all containers, including containers that recently finished their execution:

```
crictl ps -a
```

- Examine the logs of a given container:

```
crictl logs <container-id>
```

- Attach a shell to a running container:

```
crictl exec -it <container-id> /bin/bash
```

- Examine the Kubelet logs in case something did not work as expected:

```
journalctl -u kubelet
```

For additional information and guides on using `crictl`, refer to Kubernetes own documentation.

Chapter 4. Air-gapped Container Deployment

Container deployment on the BlueField DPU can be done in air-gapped networks and does not require an Internet connection. As explained previously, per DOCA service container, there are 2 required components for successful deployment:

- Container image – hosted on NVIDIA's NGC catalog
- YAML file for the container

From an infrastructure perspective, one additional module is required:

- `k8s.gcr.io/pause` container image

4.1. Pulling Container for Offline Deployment

When preparing an air-gapped environment, users must pull the required container images in advance so they could be imported locally to the target machine:

```
docker pull <container-image:tag>
docker save <container-image:tag> > <name>.tar
```

The following example pulls DOCA Firefly `1.1.0-doca2.0.2`:

```
docker pull nvcr.io/nvidia/doca/doca_firefly:1.1.0-doca2.0.2
docker save nvcr.io/nvidia/doca/doca_firefly:1.1.0-doca2.0.2 > firefly_v1.1.0.tar
```

Note: Some of DOCA's container images support multiple architectures, causing the `docker pull` command to pull the image according to the architecture of the machine on which it is invoked. Users may force the operation to pull an Arm image by passing the `--platform` flag:

```
docker pull --platform=linux/arm64 <container-image:tag>
```

4.2. Importing Container Image

After exporting the image from the container catalog, users must place the created `*.tar` files on the target machine on which to deploy them. The import command is as follows:

```
ctr --namespace k8s.io image import <name>.tar
```

For example, to import the firefly `.tar` file pulled in the previous section:

```
ctr --namespace k8s.io image import firefly_v1.1.0.tar
```

Examining the status of the operation can be done using the image inspection command:

```
crictl images
```

4.3. Built-in Infrastructure Support

The DOCA image comes pre-shipped with the `k8s.gcr.io/pause` image:

```
/opt/mellanox/doca/services/infrastructure/
├── docker_pause_3_2.tar
└── enable_offline_containers.sh
```

This image is imported by default during boot as part of the automatic activation of DOCA Telemetry Service (DTS).

Note: Importing the image independently of DTS can be done using the `enable_offline_container.sh` script located under the same directory as the image's `*.tar` file.

In versions prior to DOCA 4.2.0, this image can be pulled and imported as follows:

- Exporting the image:
  ```
  docker pull k8s.gcr.io/pause:3.2
  docker save k8s.gcr.io/pause:3.2 > docker_pause_3_2.tar
  ```
- Importing the image:
  ```
  ctr --namespace k8s.io image import docker_pause_3_2.tar
  crictl images
  IMAGE                              TAG                 IMAGE ID            SIZE
  k8s.gcr.io/pause                   3.2                 2a060e2e7101d       487kB
  ```

Chapter 5. Configuring Container Execution

5.1. Using Entrypoint Script

When possible, DOCA containers are shipped with an init script, `entrypoint.sh`. This script is the first thing to spawn once a container boots and is responsible for executing the DOCA program. Using a container's `.yaml` file, we can control the command line arguments that the script passes to it.

The exact command-line arguments are described per application on the application's respective reference guide and per DOCA service in the respective DOCA service documentation page. The matching `.yaml` fields are described per application on the application's page on NGC.

5.2. Manual Execution from Within Container

Although most containers define the `entrypoint.sh` script as the container's ENTRYPOINT, this option is only valid for interaction-less sessions. As some DOCA applications expect an interactive shell session, the `.yaml` file supports an additional execution option.

Uncommenting (i.e., removing `#` from) the following 2 lines in the `.yaml` file causes the container to boot without spawning the application.

```
# command: ["sleep"]
# args: ["infinity"]
```

In this execution mode, you can attach a shell to the spawned container:

```
crictl exec -it <container-id> /bin/bash
```

Once attached, you get a full shell session, and you can execute the application as if it were running directly on the DPU, using the exact same command-line arguments.

When dealing with an application that spawns an interactive shell session, this option allows you to interact with the application directly through the shell.

Chapter 6. Troubleshooting Common Errors

Whenever there is some error with spawning a given container, it is recommended to first go over the list of common errors provided in this section. These errors account for the vast majority of deployment errors, and it is usually easier to verify them first before trying to parse the Kubelet journal log.

6.1. Yaml Syntax

The syntax of the `.yaml` file is extremely sensitive, and minor changes could break it and cause it to stop working. Things you should pay attention to are:

- Indentation – the file uses spaces (' ') for indentations (2 per indent). Using any other number of spaces causes an undefined behavior.

6.2. System Resources

The container only spawns once all the required system resources are allocated on the DPU and can be reserved for the container. The most notable resource in this case is the huge pages required for most DOCA programs.

Make sure that the huge pages are allocated as required per container. Both the amount and size of the pages are important and must match precisely.

6.3. Shared Folders and Files

If the `.yaml` file defines a shared folder between the container and the DPU, the folder must exist prior to spawning the container. If the program searches for a specific file within said folder, this file must exist as well. Otherwise, the program aborts and stops the container.

Chapter 7. DOCA Development Containers

The set of DOCA-based containers hosted on NGC also includes development containers that can be used as part of two development workflows:

- To serve as a BlueField software-like development environment
- Used for a multi-staged build of DOCA-based containers

The DOCA development containers, doca:devel and doca:devel-cuda, are a subset of several flavors of the DOCA base image.

More information about these DOCA base images can be found on the containers' NGC page.

7.1. Kubernetes Deployment

Just like other DOCA containers, these development containers can be deployed on top of the DPU using their respective `.yaml` files:

- doca:devel – `doca_devel.yaml`
- doca:devel-cuda - `doca_devel_cuda.yaml`

There are no prerequisites for deploying the containers.

7.2. Docker Deployment

There are scenarios in which development containers are used as development environments, whether on top of the DPU or even on top of a QEMU-emulated environment on the host. More information on the recommended development setup for DOCA-based development can be found in NVIDIA DOCA Developer Guide.

When used as development environments, it is recommended to deploy the containers directly using Docker as it provides a more developer-friendly user experience.

Note: The following steps are intended for docker-based deployment of the doca:devel images. But they could also be used for all other DOCA Base Images whether on the DPU or on the host.

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

   ```
   docker version
   ```

   If it is not installed, visit the official Install Docker Engine webpage for installation instructions.
2. Make sure the docker service is started. Run:

   ```
   sudo systemctl daemon-reload
   sudo systemctl start docker
   ```
3. Pull the container image:
   - DOCA's development containers can be pulled directly from NGC using a simple docker pull command that can be copied directly from NGC:

     a). Visit the DOCA Base Image NGC page.

     b). Under the "Tags" menu, select the desired development tag.

     c). The container tag for the docker pull command is copied to your clipboard.

     Example command:

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

     Note: The container supports multiple architectures. Therefore, docker by default attempts to pull the one matching that of the current machine: amd64 for the host and arm64 for the DPU. 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
     ```
   - Some of DOCA's development containers can also be installed directly on the host/DPU through the SDK Manager:

     a). The SDK Manager drops a `.tar` file on the selected environment (host/DPU).

     b). Go to where the tar file is saved and run the following command:

     ```
     sudo docker load -i <filename>
     ```

     Command example:

     ```
     sudo docker load -i doca_devel_ubuntu_22.04-inbox-5.5.tar
     ```
4. If working with QEMU on an x86-based host, follow the instructions as listed in the "Setup" section of the NVIDIA DOCA Developer Guide.
5. Once loaded locally, you may find the image's ID using the following command:

   ```
   sudo docker images
   ```
6. Run the docker image:

   ```
   sudo docker run -v <source-code-folder>:/doca_devel --privileged -it -e
    container=docker <image-name/ID>
   ```

For example, if the source code folder is `/<...>/buildEnv`, and loaded image has the ID `185c50ecb31d`, the command would look like this:

```
sudo docker run -v /<...>/buildEnv:/doca_devel --privileged -it -e
 container=docker 185c50ecb31d
```

After running the command, you get a shell inside the container where you can build your project using the regular build commands.

Note: Make sure to map a folder that everyone has Write privileges to. Otherwise, the docker would not be able to write the output files to it.

7. To enable docker access to the huge pages and the SFs/VFs allocated on the DPU, use the following extended command:

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

For example, if the source code folder is `/<...>/buildEnv` and the loaded image has the ID `185c50ecb31d`, the command would look like this:

```
sudo docker run -v /<...>/buildEnv:/doca_devel -v /dev/hugepages:/dev/hugepages
 --privileged --net=host -it -e container=docker 185c50ecb31d
```

After running the command, you get a shell inside the container where you can build your project using the regular build commands.

Chapter 8. DOCA Services for Host

A subset of the DOCA services are available for host-based deployment as well. This is indicated in those services' deployment and can also be identified by having container tags on NGC with the "-host" suffix.

In contrast to the managed DPU environment, the deployment of DOCA services on the host is based on docker. This deployment can be extended further based on the user's own container runtime solution.

8.1. Docker Deployment

DOCA services for the host are deployed directly using Docker.

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

   ```
   docker version
   ```

   If it is not installed, visit the official Install Docker Engine webpage for installation instructions.
2. Make sure the Docker service is started. Run:

   ```
   sudo systemctl daemon-reload
   sudo systemctl start docker
   ```
3. Pull the container image directly from NGC (can also be done using the `docker run` command):

   a). Visit the NGC page of the desired container.

   b). Under the "Tags" menu, select the desired tag and click the paste icon so it is copied to your clipboard.

   c). The docker pull command is as follows:

   ```
   sudo docker pull <NGC container tag here>
   ```

   For example:

   ```
   sudo docker pull nvcr.io/nvidia/doca/doca_firefly:1.1.0-doca2.0.2-host
   ```

   Note: For DOCA services with deployments on both DPU and host, please ensure you select the tag ending with "-host".
4. Deploy the DOCA service using Docker:

a). The deployment is performed using the following command:

```
sudo docker run --privileged --net=host -v <host directory>:<container
 directory> -e <env variables> -it <container tag> /entrypoint.sh
```

For more information, please refer to Docker's official documentation.

b). The specific deployment command for each DOCA service is listed in their respective deployment guide.

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