



# NVIDIA BlueField DPU Container Deployment Guide

User Guide

# Table of Contents

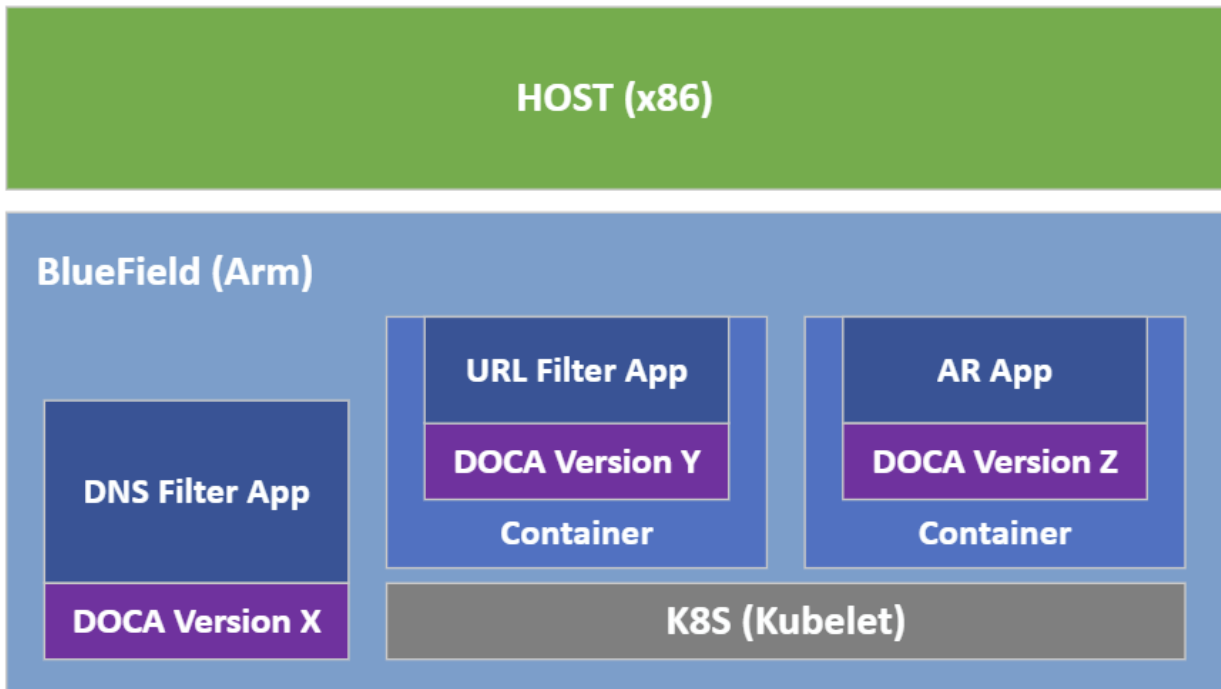
Chapter 1. Introduction.....	1
Chapter 2. Prerequisites.....	2
Chapter 3. Container Configuration.....	3
3.1. Configure NGC Credentials on BlueField.....	4
3.2. Pull Container YAML Configurations.....	5
3.3. Activate Container-related Services.....	5
3.4. Container-specific Instructions.....	5
3.5. Spawn Container.....	6
3.6. Stop Container.....	6
3.7. Useful Container Commands.....	6
Chapter 4. Application Execution Within Container.....	7
4.1. Using Entrypoint Script.....	7
4.2. Manual Execution From Within Container.....	7
Chapter 5. Troubleshooting Common Errors.....	8
5.1. Yaml Syntax.....	8
5.2. Resources.....	8
5.3. Shared Folders and Files.....	8
Chapter 6. DOCA Development Container.....	9

---

# Chapter 1. Introduction

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

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



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 [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)

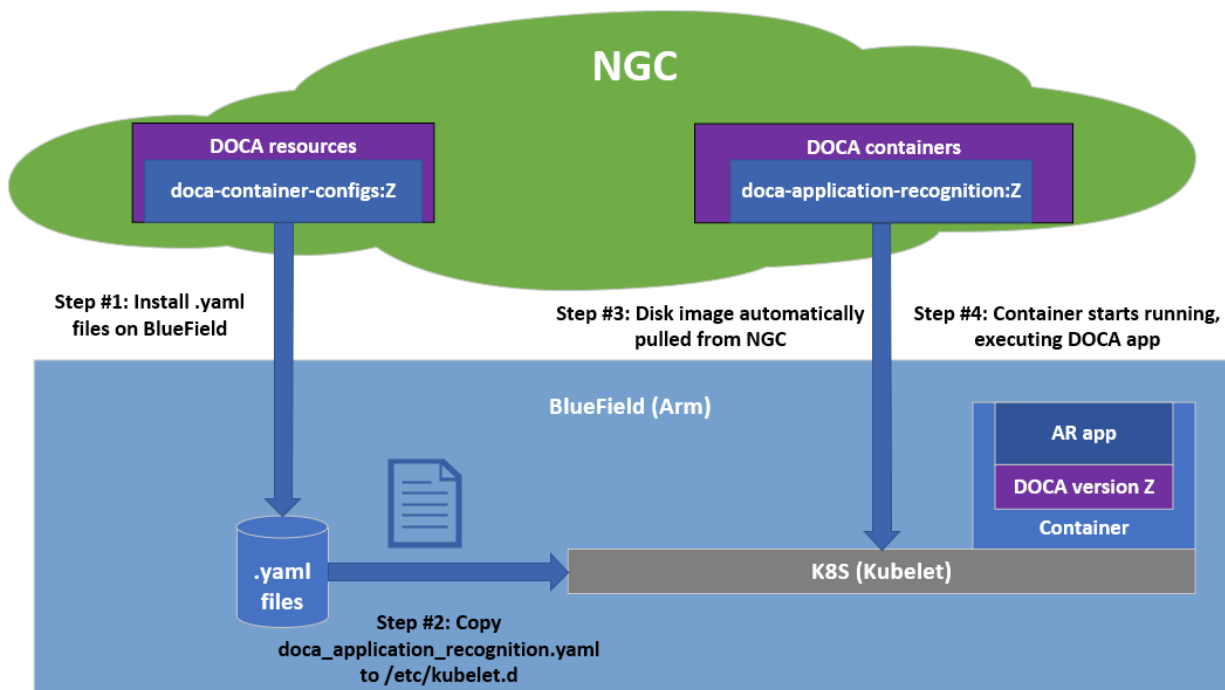
# Chapter 3. Container Configuration

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

1. NGC Sign-in.
2. Pull the container .yaml configuration files.
3. Modify the container's .yaml configuration file.
4. Deploy container. 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.



## 3.1. Configure NGC Credentials on BlueField

Containers are deployed on the BlueField using Kubernetes (K8S) using the standalone Kubelet service. While Kubelet can automatically pull the container image directly from NGC, you must first supply it with login credentials.

This step needs to only be performed once per DPU to allow K8S to access NGC.

1. Create an NGC user according to the [NGC Guide](#).



**Note:** The privileges of the NGC user required for pulling the container images from NGC to the DPU, are the **minimal** privileges of an NGC user. When used in production environments, it is **highly recommended** to create a dedicated NGC user to **only** be used by K8S for deploying the containers on the BlueField.

This reduces the risk for possible exposure of the API key as it only grants pull privileges from public NGC content and cannot be used for read/write access to any sensitive/privileged content.

2. Generate an API key for your user according to the [NGC Guide](#).
3. Log into NGC via docker.

```
# Start docker's service
systemctl start docker
# Login to NGC via docker (using the user's API Key)
docker login nvcr.io
username: $oauthtoken # Yes, "$oauthtoken" is the username that should be used
password: <Your-API-Key>
```

The expected output should roughly include the following:

```
"
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
...
Login Succeeded
"
```

4. Extract the auth token from docker's file. In this example it is located at `/root/.docker/config.json` as shown above.

```
# The config file should look roughly like this:
"
{
  "auths": {
    "nvcr.io": {
      "auth": "<long hexa-decimal string. This is the token we need.>"
    }
  }
}
"
```

5. Update containerd's configuration file at `/etc/containerd/config.toml`:
  - a). Remove the comments from the 3 configuration lines.
  - b). Insert your auth token as described in the line itself.

```
# Add the nvcr.io auth token (taken from docker's config.json file above)
"
```

```

...
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."nvcr.io"]
    endpoint = ["https://nvcr.io"]
  [plugins."io.containerd.grpc.v1.cri".registry.configs]
    [plugins."io.containerd.grpc.v1.cri".registry.configs."nvcr.io".auth]
      auth = "<auth token as copied from docker's config.json file>"
...
"

```



**Note:** The file is **extremely** sensitive to spaces and indentation. Please make sure to use only spaces (' '), and to use two spaces per indentation level.

- Restart the containerd service to apply your changes:
 

```
systemctl restart containerd
```
- Log out from docker to erase the API key from docker's config.json:
 

```
docker logout nvcr.io
```

## 3.2. Pull Container YAML Configurations

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.

- Install unzip:
 

```
apt install unzip
```
- Install NGC CLI according to the [NGC Guide](#).
- Configure the NGC CLI with your API key using `ngc config set`.
- Download the set of `.yaml` configuration files, stored as a resource on NGC, using the CLI command shown on the [resource page](#).

The resource contains a `configs` directory, under which can be found a dedicated folder per DOCA version. For example, 1.2 will include all currently available `.yaml` configuration files for DOCA 1.2 containers.

## 3.3. Activate Container-related Services

Start the following services required for the K8S deployment of your container:

```

# Going forward, enable both services
systemctl enable kubelet
systemctl enable containerd
# For now, start them both
systemctl start kubelet
systemctl restart containerd

```

## 3.4. 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.

## 3.5. 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 so that Kubelet will 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 didn't work as expected:  

```
journalctl -u kubelet
```

For additional information and guides on using `crictl`, refer to [Kubernetes own documentation](#).



---

# Chapter 4. Application Execution Within Container

## 4.1. Using Entrypoint Script

When possible, DOCA application 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 application. Using an application's `.yaml` file, we can control the command line arguments that the script passes to the application.

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

## 4.2. 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 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 5. 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.

## 5.1. 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.
- ▶ Naming conventions – Both the pod name and container name have a strict alphabet (RFC 1123). This means that you must only use "-" and not "\_", as the latter is an illegal character and cannot be used in the pod/container name. However, for the container's image name we will use "\_" instead of "-". This helps differentiate between the two.

## 5.2. Resources

The container only spawns once all needed 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 applications.

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.

## 5.3. 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 application searches for a specific file within said folder, this file must exist as well. Otherwise, the application aborts and stops the container.

---

# Chapter 6. DOCA Development Container

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

- ▶ To serve as a BlueField OS-like development environment
- ▶ Used for a multi-staged build of containers

The DOCA development container, `doca:devel`, is one of several flavors of the DOCA base image.

The `.yaml` file associated with this container is `doca_devel.yaml`, and the complete list of instructions for deploying the container can be found on the container's [NGC page](#).

## 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 NON-INFRINGEMENT, 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

© 2022 NVIDIA Corporation & affiliates. All rights reserved.