CNF Reference Architecture

Release 22.06

Arm Limited.

Dec 14, 2022

CONTENTS

CHAPTER

ONE

CONTENTS

1.1 Overview

1.1.1 Introduction

This software provides a management/orchestration template for containerized networking applications on various Arm platforms, by leveraging cloud-native and Kubernetes cluster technologies. The software is implemented and validated on AArch64.

1.1.2 Solution Architecture

The solution is prepared to setup a single or multi-node Kubernetes (K8s) cluster. The cluster is created by executing an Ansible playbook on a separate management node. In the case of the single-node cluster, the worker nodes and controller node are running on the same physical machine. Each node in the K8s cluster utilizes the `containerd` container runtime. The worker nodes have hugepages and isolated CPUs dedicated for the networking application. These CPUs and hugepages are managed and assigned to the K8s pods by the K8s control plane. The controller node also runs a private Docker registry, which stores custom-built container images.

1.1.3 Use Cases

The solution's goal is to establish a K8s cluster template to run networking applications. Sample application(s) are included as a reference.

The implementation mechanism of each use case below will be described in detail in later chapter of this documentation.

- **K8s Cluster Template**

  Setup of K8s cluster template to run cloud-native networking applications.

- **DPDK L3FWD**

  Deployment of cloud-native Data Plane Development Kit (DPDK) based L3 routing application.

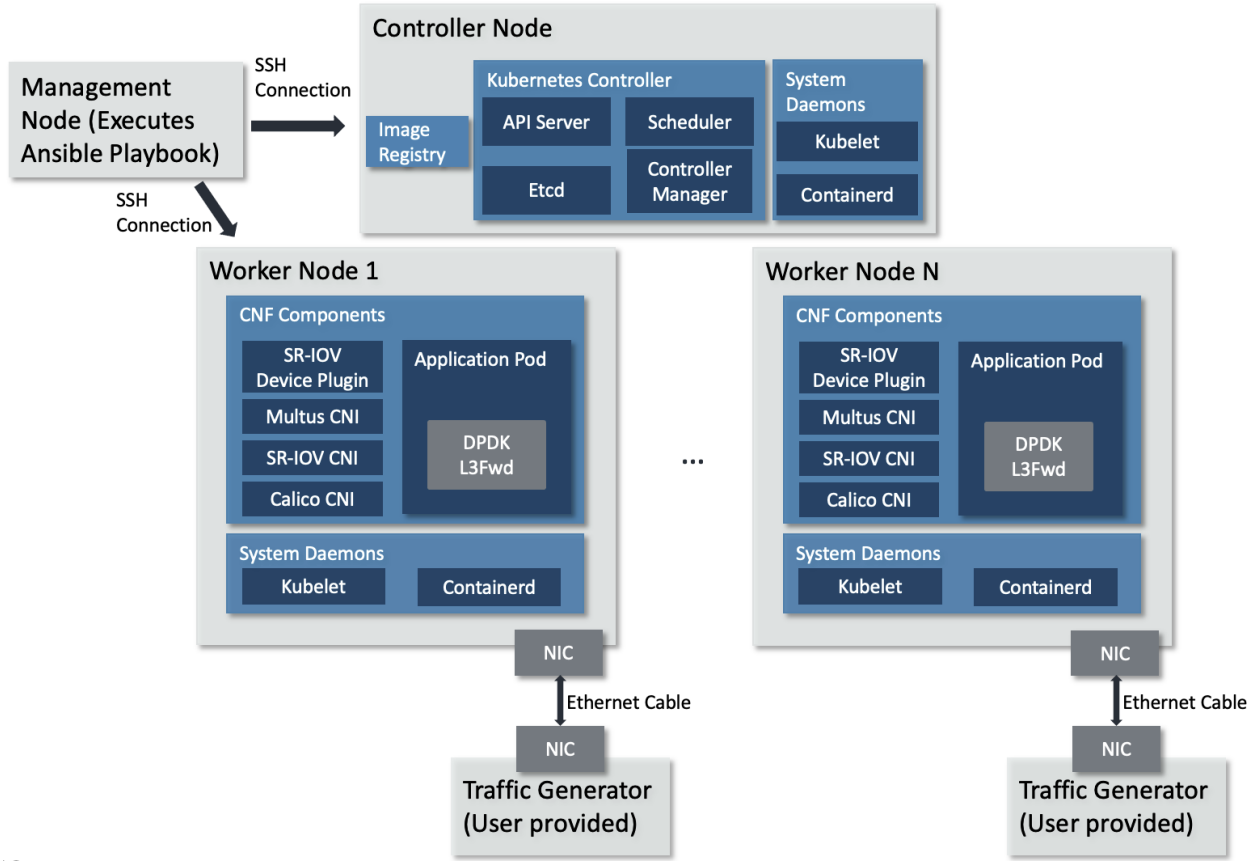


Fig. 1: CNF Reference Architecture solution architecture

1.1.4 Features

- Automated Kubernetes cluster setup
- Automated private Docker registry setup
- Automated pinning of networking workload to isolated CPU cores

Implementation Considerations

There are several technical considerations in the implementation process:

- **Template for creating a Kubernetes cluster for any networking application**

  End users can take this software to instantiate any cloud native network functions, e.g., Snort3 & VPP based security router per their actual use case.

- **Provide network scalability and high availability**

  The system or application should be easily scalable to handle bigger amounts of work, or to be easily expanded, in response to increased demand for network processing capability.

- **Enable centralized and distributed deployment**

  The control plane controllers and data plane workers could run on the same node or different nodes.

- **Support applications from edge to cloud – K3s, Micro-K8s, K8s**

  Cloud computing has led many organizations to centralize their services within large data centers. However, new end-user experiences like the Internet of Things (IoT) require service provisioning closer to the outer "edges" of a network, where the physical devices exist. In the future, this solution will target more container orchestrators than just K8s.

Tested Platforms

The below machines running Ubuntu 20.04 have been validated as Kubernetes cluster nodes:

- Ampere Altra (Neoverse-N1)

The below operating systems have been validated as management nodes:

- Ubuntu 20.04

1.1.5 Repository Structure

The code repository are stored in Arm GitLab arm-reference-solutions group, with below link addresses.

- https://gitlab.arm.com/arm-reference-solutions/cnf-reference-arch

Under the `cnf-reference-arch` repository are the following directories/files:

```
├── CHANGELOG.rst          - Change notes for each releases
├── create-cluster.yaml    - Ansible playbook to setup K8s cluster
├── doc/                   - Sphinx based documents
├── examples/              - Example networking applications to deploy
├── inventory.ini          - Inventory file listing the K8s cluster nodes
├── LICENSE.rst            - License file of this project
├── MAINTAINERS            - Maintainer recording of specific module/use-cases
```

(continues on next page)

(continued from previous page)

```
├── Makefile                - Top level makefile
├── README.md               - Project introduction file
├── roles/                  - Ansible roles consumed by create-cluster.yaml
├── tools/                  - Tools used to manage the project
```

1.1.6 Repository License

The software is provided under an Apache 2.0 license (more details in Apache 2.0).

Contributions to the project should follow the same license.

1.1.7 Contributions and Bug Reports

This project has not put in place a process for contributions currently.

For bug reports, please submit an Issue via GitLab.

1.1.8 Feedback and Support

To request support please contact Arm at support@arm.com. Arm licensees may also contact Arm via their partner managers.

1.1.9 Maintainer(s)

- Honnappa Nagarahalli <Honnappa.Nagarahalli@arm.com>
- Lijian Zhang <Lijian.Zhang@arm.com>
- Nathan Brown <Nathan.Brown@arm.com>
- Tianyu Li <Tianyu.Li@arm.com>

1.2 Quickstart Guide

1.2.1 Introduction

Welcome to the CNF Reference Architecture quickstart guide. This guide provides the quick guidance to run a sample containerized networking application in a Kubernetes cluster, which is comprised of AArch64 machines.

This reference solution is targeted for a networking software development or performance analysis engineer with in-depth Kubernetes and networking knowledge, but does not know AArch64 architecture necessarily.

Mastering knowledge on certain open source projects, e.g., Ansible, Kubernetes, DPDK, will help gain deeper understanding of this guide and the reference solution more easily.

By following the steps in this quickstart guide to the end, you will set up a single-node Kubernetes cluster. Kubernetes controller and Application Pods are deployed on a single AArch64 machine. The DPDK L3 forward sample application is deployed in the Application Pod. The application receives and forwards the packets based on the destination IP address using a single port. The single-node Kubernetes cluster topology is shown as below.

The topology diagram above illustrates the major components of the deployment and their relationship.

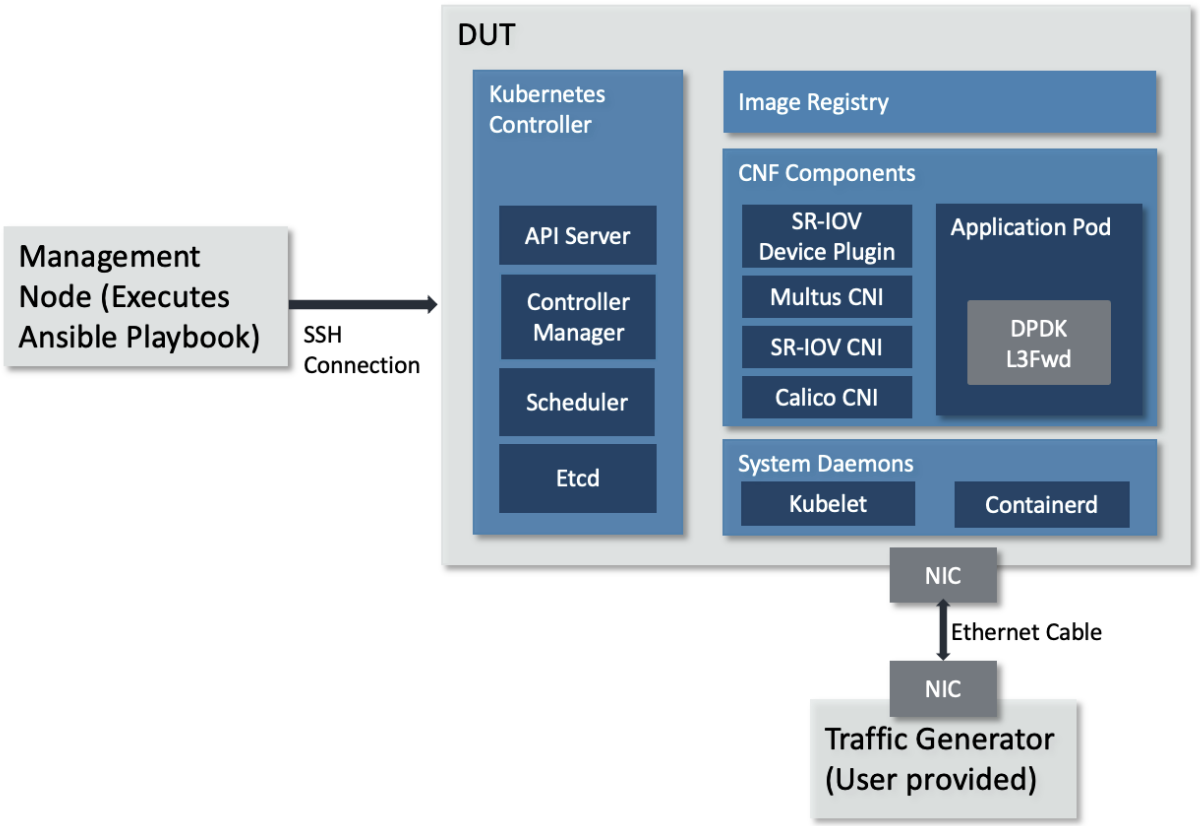


Fig. 2: Single-node Kubernetes cluster topology

- DUT (Device Under Test), is the only AArch64 machine in single-node Kubernetes cluster. Kubernetes controller and Application Pods run on this machine.
- DPDK L3 forwarding application, implements L3 networking function in software and forwards packets per their destination IP address.
- TG (Traffic Generator), generates and sends packets to the AArch64 machine's NIC card via the Ethernet cable. It can be hardware TG, e.g., IXIA chassis, or software TG running on regular server, e.g., TRex, DPDK Pktgen, Scapy.
- Management Node, can be any bare-metal machine, VM, or container. It is used to download the project source code, login to the DUT to create the Kubernetes cluster and deploy the application.

1.2.2 Hardware Setup

This guide requires the following setup:

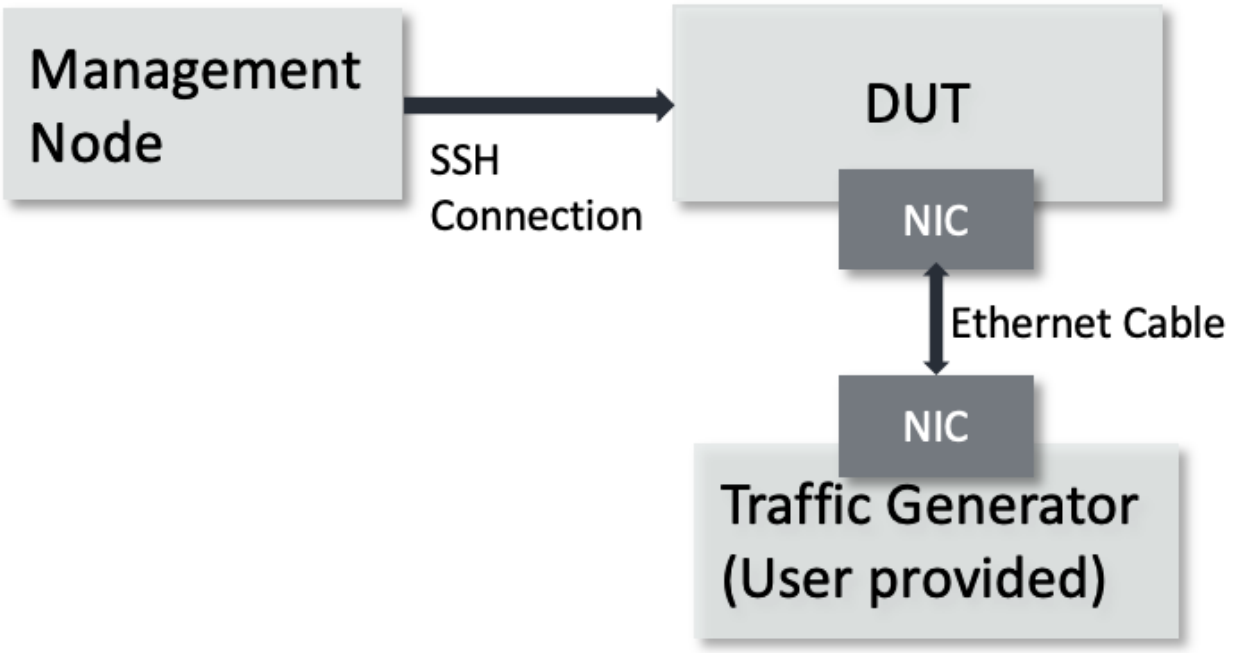


Fig. 3: Required hardware setup

1. DUT is an AArch64 architecture machine and the only node in Kubernetes cluster. NIC card is plugged in its PCIe slot to connect the traffic generator via Ethernet cable.

   Hardware Minimum Requirements

   The DUT has the following hardware requirements:

   - AArch64 v8 CPU
   - Minimum 1GHz and 4 CPU cores
   - DPDK compatible NIC
   - Connection to the internet to download and install packages
   - Minimum 8G of RAM

- Support 1G Hugepages

Software Minimum Requirements

The following items are expected of the DUT's software environment.

- DUT is running Ubuntu 20.04 (Focal)
- Admin (root) privileges are required to setup the DUT
- The Fully Qualified Domain Name (FQDN) of the DUT can be checked with `python3 -c 'import socket; print(socket.getfqdn())'` command. See *FAQ* if the proper FQDN is not shown.
- PCIe address of the NIC port attached to the traffic generator is confirmed with `sudo lshw -C network -businfo`
- CPU cores are isolated via `isolcpus, nohz_full, rcu_nocbs, cpuidle.off, cpufreq.off` Linux command line parameters. See *FAQ* for more details.

2. Management node can be any bare-metal, VM, or container. The management node is used to download the repository, access the DUT via `ssh` and configure Kubernetes cluster by executing an Ansible playbook. The Ansible playbook is executed locally on management node and it configures the DUT via `ssh`.

   Software Minimum Requirements

   - Can execute Ansible
   - Can `ssh` into the DUT using SSH keys. See *FAQ* for more details.
   - Admin (root) or `sudo` privileges are required

3. TG can be any traffic generator capable of generating IP packets.

Tested Platforms

The steps described in this quickstart guide have been validated on the following platforms.

DUT

- Ampere Altra (Neoverse-N1)
  - Ubuntu 20.04.3 LTS (Focal Fossa)
  - Kernel 5.17.0-051700-generic

NIC

- Mellanox ConnectX-5
  - OFED driver: MLNX_OFED_LINUX-5.4-3.1.0.0
  - Firmware version: 16.30.1004 (MT_0000000013).
- Intel X710
  - Firmware version: 6.01

Note: To use Mellanox NIC, install OFED driver, update and configure NIC firmware by following the guidance in *FAQ*.

Management Node

- Ubuntu 20.04 system
  - Python 3.8
  - Ansible 6.5.0

1.2.3 Prerequisite

Management Node

Management node needs to install dependencies, e.g., `git, curl, python3.8, pip, Ansible, repo`. Follow below guidelines on Ubuntu 20.04.

1. Make sure `sudo` is available and install `git, curl, python3.8, python3-pip, python-is-python3` by executing

```
$ sudo apt-get update
$ sudo apt-get install git curl python3.8 -y
$ sudo apt-get install python3-pip python-is-python3 -y
```

2. Install `ansible` by executing

```
$ sudo python3 -m pip install ansible==6.5.0
```

Note: Install the `ansible` and not the `ansible-core` package, as this solution makes use of community packages not included in the `ansible-core` python package.

3. Configure git with your name and email address

```
$ git config --global user.email "you@example.com"
$ git config --global user.name "Your Name"
```

4. Follow the instructions provided in git-repo to install the `repo` tool manually
5. Follow the *FAQ* to setup SSH keys on the management node

DUT

Complete below steps by following the suggestions provided.

6. Follow the *FAQ* to setup DUT with isolated CPUs, and 1G hugepages.
7. Update NIC firmware and drivers by following the guidance in the *FAQ*.

1.2.4 Download Source Code

Unless mentioned specifically, all operations in this section are executed on management node.

Create a new folder that will be the workspace, henceforth referred to as <nw_cra_workspace> in these instructions:

```
mkdir <nw_cra_workspace>
cd <nw_cra_workspace>
export NW_CRA_RELEASE=refs/tags/NW-CRA-2022.12.30
```

Note: Sometimes new features and additional bug fixes are made available in the git repositories, but are not tagged yet as part of a release. To pick up these latest changes, remove the `-b <release tag>` option from the `repo init` command below. However, please be aware that such untagged changes may not be formally verified and should be considered unstable until they are tagged in an official release.

To clone the repository, run the following commands:

```
repo init \
    -u https://git.gitlab.arm.com/arm-reference-solutions/arm-reference-solutions-
↪manifest.git \
    -b ${NW_CRA_RELEASE} \
    -m cnf-reference-arch.xml
repo sync
```

1.2.5 Create Single Node Cluster

Unless mentioned specifically, all operations henceforth are executed on management node.

Create Ansible Inventory File

The Ansible playbooks in this repository are easiest to use with inventory files to keep track of the cluster nodes. For this solution we need one inventory file.

A template `inventory.ini` is provided at <nw_cra_workspace>/cnf-reference-arch/inventory.ini with the following contents:

```
[controller]
<fqdn> ansible_user=<remote_user>

[worker]
<fqdn> ansible_user=<remote_user> pcie_addr=<pcie_addr_from_lshw> dpdk_driver=<vfio-pci>
```

Replace <fqdn> with the FQDN of the DUT. The same FQDN should be used for both `[controller]` and `[worker]` as this is a single-node setup.

<remote_user> specifies the user name to use to login to the DUT.

Replace <pcie_addr_from_lshw> with the PCIe address of the port on the DUT connected to the traffic generator.

If the DUT uses Mellanox ConnectX-5 NIC to connect the traffic generator, replace <driver-name> with `mlx5_core`. Otherwise, replace it with `vfio-pci`.

As an example, if the user name used to access DUT is user1, the DUT FQDN is `dut.arm.com` and is connected to the traffic generator on PCIe address `0000:06:00.1` with a NIC compatible with the `vfio-pci` driver, then `inventory.ini` would contain:

```
[controller]
dut.arm.com ansible_user=user1

[worker]
dut.arm.com ansible_user=user1 pcie_addr=0000:06:00.1 dpdk_driver=vfio-pci
```

Execute the Playbook

To setup the Kubernetes cluster, run:

```
$ ansible-playbook -i inventory.ini create-cluster.yaml -K
```

It will start by asking for the `sudo` password of the user name on DUT (the prompt may say `BECOME` password instead). If remote user has passwordless sudo on DUT, the `-K` flag can be omitted.

See the *user guide* for the full list of actions this playbook will take.

Validate the Cluster

At this point in time, the setup should look like the *Single-node Kubernetes cluster topology* at the beginning of this document. The DUT should be in a Kubernetes cluster and running a private docker registry.

To verify, `ssh` into the DUT and run `kubectl get nodes`. The output should look like:

```
$ kubectl get nodes
NAME            STATUS   ROLES           AGE   VERSION
dut.arm.com     Ready    control-plane   24m   v1.24.2
```

Also run `kubectl describe node $(hostname) | grep -A 5 ^Allocatable:` to ensure allocatable CPUs and 1G hugepages are correct. The output should look like:

```
$ kubectl describe node $(hostname) | grep -A 5 ^Allocatable:
Allocatable:
arm.com/dpdk:       2
cpu:                3
ephemeral-storage:  189217404206
hugepages-1Gi:      1Gi
```

Finally, verify the local docker registry is running with: `docker ps -f name=registry`. The output should look like:

```
$ docker ps -f name=registry
CONTAINER ID   IMAGE        COMMAND                  CREATED        STATUS          ␣
↪PORTS                              NAMES
53656144b298   registry:2   "/entrypoint.sh /etc..."   46 hours ago   Up 33 minutes   0.
↪0.0.0:443->443/tcp, 5000/tcp   registry
```

1.2.6 Run the Sample Application

Run

Now, it is time to apply the `dpdk-l3fwd.yaml` Ansible playbook. To do so, run the following commands:

```
$ cd <nw_cra_workspace>/cnf-reference-arch/examples/dpdk-l3fwd
$ ansible-playbook -i ../../inventory.ini dpdk-l3fwd.yaml
```

See the *dpdk-l3fwd user guide* for the full list of actions this playbook will take.

Once the playbook finishes, `ssh` into the DUT and deploy the DPDK L3 forwarding application with `dpdk-deployment.yaml` file which is copied and stored in DUT home directory:

```
$ cd $HOME
$ kubectl apply -f dpdk-deployment.yaml
```

Check deployment status with command:

```
$ kubectl get deploy
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
dpdk   1/1     1            1           74s
```

Test

Monitor the application pod status by running `kubectl get pods` on the DUT. It may take some time to start up.

`kubectl get pods` should show something like:

```
$ kubectl get pods
NAME                   READY   STATUS    RESTARTS   AGE
dpdk-9d8474bd8-xl7kk   1/1     Running   0          6s
```

Once the pod is in the "Running" state, view its logs with `kubectl logs <podname>`.

The logs should contain something similar to:

```
$ kubectl logs dpdk-9d8474bd8-xl7kk
...
Initializing port 0 ... Creating queues: nb_rxq=1 nb_txq=1...
Address:98:03:9B:71:24:2E, Destination:02:00:00:00:00:00, Allocated mbuf pool on socket 0
LPM: Adding route 198.18.0.0 / 24 (0) [0001:01:00.0]
LPM: Adding route 2001:200:: / 64 (0) [0001:01:00.0]
txq=2,0,0
```

These logs show port 0 has MAC address `98:03:9B:71:24:2E` with PCIe address `0001:01:00.0` on the DUT. 1 IPv4 route matching the subnet `198.18.0.0/24` is added.

Configure the traffic generator to send packets to the NIC port, using the MAC and IP address displayed in the logs. In this example, use a destination MAC address of `98:03:9B:71:24:2E` and a destination IP of `198.18.0.21`. Then, `dpdk-l3fwd` will forward those packets out on port 0.

Stop

The pods can be stopped by deleting the deployment by running `kubectl delete deploy dpdk` on the DUT. Then, clean up the Kubernetes cluster by executing `sudo kubeadm reset -f` and `sudo rm -rf /etc/cni/net.d` on the DUT.

1.3 User Guide

1.3.1 Introduction

Welcome to the CNF Reference Architecture user guide. This guide provides instructions on how to run a sample containerized networking application in a multi-node Kubernetes cluster comprised of AArch64 machines.

This reference solution is targeted for a networking software development or performance analysis engineer who has in-depth networking knowledge, but does not know AArch64 architecture necessarily.

Mastering knowledge on certain open source projects, e.g., Ansible, Kubernetes, DPDK, will help gain deeper understanding of this guide and reference solution more easily.

This guide is intended to describe complex and practical uses cases requiring complex test setup. By following the steps of this guide to the end, you will setup a multi-node Kubernetes cluster. One machine will serve as the Kubernetes controller and host a private Docker registry to hold custom container images. The worker nodes will run Application Pods, like DPDK L3 forward. The multi-node Kubernetes cluster topology is shown below.

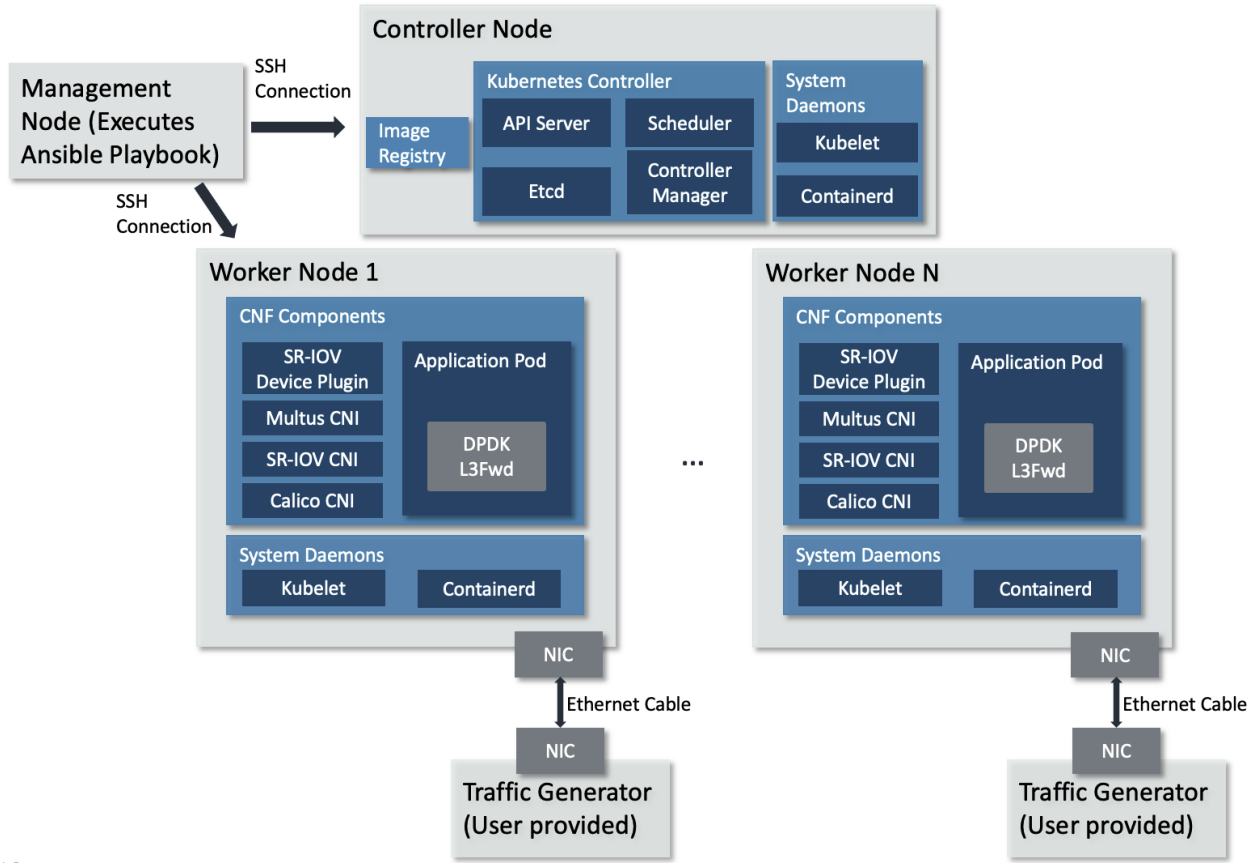


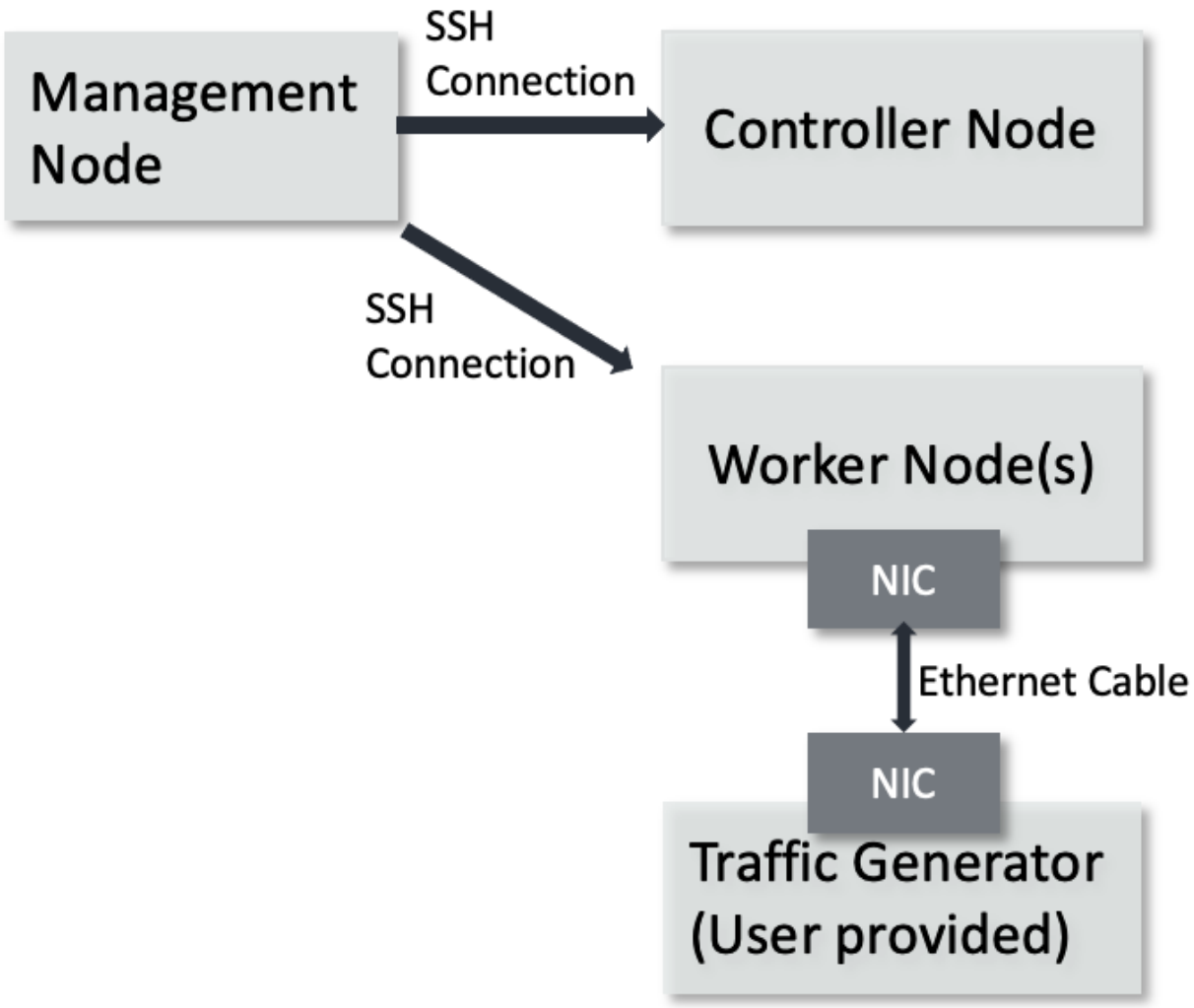
Fig. 4: Multi-node Kubernetes cluster topology

The topology diagram above illustrates the major components of the deployment and their relationship.

- DPDK L3 forwarding application, implements L3 networking function in software and forwards packets per their destination IP address.
- TG (Traffic Generator), generates and sends packets to the worker node's NIC card via the Ethernet cable. It can be hardware TG, e.g., IXIA chassis, or software TG running on regular server, e.g., TRex, DPDK Pktgen, Scapy.
- Management Node, can be any bare-metal machine, VM, or container. It is used to download the project source code, login to the controller and worker nodes to create the Kubernetes cluster and deploy the application.

1.3.2 Hardware Setup

This guide requires the following setup:



1. Controller Node can be any machine that has a network connection to the other machines in the Kubernetes cluster. The solution is tested against an AArch64 machine as the controller node.

   Hardware Minimum Requirements

   The Controller Node has the following hardware requirements:

   - Minimum 1GHz and 2 CPU cores

- Minimum 8GB RAM
- Connection to the internet to download and install packages
- Connection to the worker nodes

Software Minimum Requirements

The following items are expected of the Controller Node's software environment:

- Controller Node is running Ubuntu 20.04 (Focal)
- Admin (root) privileges are required
- The Fully Qualified Domain Name (FQDN) of the Controller Node can be checked with `python3 -c 'import socket; print(socket.getfqdn())'` command. See *FAQ* if the proper FQDN is not shown.

2. Worker Nodes are any number of AArch64 architecture machines. NIC card is plugged into a PCIe slot and is connected to a traffic generator with an Ethernet cable.

   Hardware Minimum Requirements

   The Worker Nodes have the following hardware requirements:

   - AArch64 v8 CPU
   - Minimum 1GHz and 4 CPU cores
   - DPDK compatible NIC
   - Connection to the internet to download and install packages
   - Minimum 8G of RAM
   - Support 1G Hugepages

   Software Minimum Requirements

   - Worker node is running Ubuntu 20.04 (Focal)
   - Admin (root) privileges are required
   - PCIe address of the NIC port(s) attached to the traffic generator is confirmed with `sudo lshw -C network -businfo`
   - CPU cores are isolated and 1GB hugepages reserved via required Linux command line parameters. See *FAQ* for more details.

There can be any number of worker nodes. To use a single-node cluster, refer to the *Quickstart Guide*.

3. Management node can be any bare-metal, VM, or container. The management node is used to download the repository, access the cluster nodes via `ssh` and configure the Kubernetes cluster by executing an Ansible playbook. The Ansible playbook is executed locally on management node and it configures the cluster nodes via `ssh`.

   Software Minimum Requirements

   - Can execute Ansible
   - Can `ssh` into each cluster node using SSH keys. See *FAQ* for more details.
   - Admin (root) or `sudo` privileges are required

4. TG can be any traffic generator capable of generating IP packets.

1.3.3 Tested Platforms

This solution is tested on the following platforms.

Cluster Nodes

- Ampere Altra (Neoverse-N1)
  - Ubuntu 20.04.3 LTS (Focal Fossa)
  - Kernel 5.17.0-051700-generic

NIC

- Mellanox ConnectX-5
  - OFED driver: MLNX_OFED_LINUX-5.4-3.1.0.0
  - Firmware version: 16.30.1004 (MT_0000000013).
- Intel X710
  - Firmware version: 6.01

Note: To use Mellanox NIC, install OFED driver, update and configure NIC firmware by following the guidance in *FAQ*.

Management Node

- Ubuntu 20.04 system
  - Python 3.8
  - Ansible 6.5.0

1.3.4 Prerequisite

Management Node

Management node needs to install dependencies, e.g., `git, curl, python3.8, pip, Ansible, repo`. Follow below guidelines on Ubuntu 20.04.

1. Make sure `sudo` is available and install `git, curl, python3.8, python3-pip, python-is-python3` by executing

```
$ sudo apt-get update
$ sudo apt-get install git curl python3.8 -y
$ sudo apt-get install python3-pip python-is-python3 -y
```

2. Install `ansible` by executing

```
$ sudo python3 -m pip install ansible==6.5.0
```

Note: Install the `ansible` and not the `ansible-core` package, as this solution makes use of community packages not included in the `ansible-core` python package.

3. Configure git with your name and email address

```
$ git config --global user.email "you@example.com"
$ git config --global user.name "Your Name"
```

4. Follow the instructions provided in git-repo to install the `repo` tool manually
5. Follow the *FAQ* to setup SSH keys on the management node

DUT

Complete below steps by following the suggestions provided.

6. Follow the *FAQ* to setup DUT with isolated CPUs and 1G hugepages.
7. Update NIC firmware and drivers by following the guidance in the *FAQ*.

1.3.5 Download Source Code

Unless mentioned specifically, all operations in this section are executed on management node.

Create a new folder that will be the workspace, henceforth referred to as `<nw_cra_workspace>` in these instructions:

```
mkdir <nw_cra_workspace>
cd <nw_cra_workspace>
export NW_CRA_RELEASE=refs/tags/NW-CRA-2022.12.30
```

Note: Sometimes new features and additional bug fixes are made available in the git repositories, but are not tagged yet as part of a release. To pick up these latest changes, remove the `-b <release tag>` option from the `repo init` command below. However, please be aware that such untagged changes may not be formally verified and should be considered unstable until they are tagged in an official release.

To clone the repository, run the following commands:

```
repo init \
    -u https://git.gitlab.arm.com/arm-reference-solutions/arm-reference-solutions-
↪manifest.git \
    -b ${NW_CRA_RELEASE} \
    -m cnf-reference-arch.xml
repo sync
```

1.3.6 Create Kubernetes Cluster

Unless mentioned specifically, all operations henceforth are executed on management node.

Create Ansible Inventory File

The Ansible playbooks in this repository are easiest to use with inventory files to keep track of the cluster nodes. For this solution we need one inventory file.

A template `inventory.ini` is provided at `<nw_cra_workspace>/cnf-reference-arch/inventory.ini` with the following contents:

```
[controller]
<fqdn> ansible_user=<remote_user>

[worker]
<fqdn> ansible_user=<remote_user> pcie_addr=<pcie_addr_from_lshw> dpdk_driver=<vfio-pci>
```

Under the `[controller]` heading, replace `<fqdn>` with the FQDN of the Controller Node. Under the `[worker]` heading, replace `<fqdn>` with the FQDN of a worker node, or an SSH destination for a worker node.

`<remote_user>` specifies the user name to use to login to that node.

Replace `<pcie_addr_from_lshw>` with the PCIe address of the port on the worker node connected to the traffic generator.

If the worker node uses Mellanox ConnectX-5 NIC to connect the traffic generator, replace `<driver-name>` with `mlx5_core`. Otherwise, replace it with `vfio-pci`.

If multiple worker nodes are to be used, each one should be a separate line under the `worker` tag, with `ansible_user`, `pcie_addr` and `dpdk_driver` filled in per worker node.

As an example, if the user name used to access the cluster nodes is `user1`, the controller's FQDN is `dut.arm.com`, the sole worker is reachable at `worker-1` and is connected to the traffic generator on PCIe address `0000:06:00.1` with a NIC compatible with the `vfio-pci` driver, then `inventory.ini` would contain:

```
[controller]
dut.arm.com ansible_user=user1

[worker]
worker-1 ansible_user=user1 pcie_addr=0000:06:00.1 dpdk_driver=vfio-pci
```

Note: All PCIe addresses for a single node must work with the same DPDK driver. This solution does not support per-address DPDK drivers without modification.

If `worker-1` also had PCIe address `0000:06:00.0` connected to a traffic generator, then `inventory.ini` would contain:

```
[controller]
dut.arm.com ansible_user=user1

[worker]
worker-1 ansible_user=user1 pcie_addr="['0000:06:00.1', '0000:06:00.0']" dpdk_
↪driver=vfio-pci
```

If the same setup also included a `worker-2` which is connected to a traffic generator on PCIe address `0000:09:00.0` with a Mellanox NIC, then `inventory.ini` would contain:

```
[controller]
dut.arm.com ansible_user=user1

[worker]
worker-1 ansible_user=user1 pcie_addr="['0000:06:00.1', '0000:06:00.0']" dpdk_
↪driver=vfio-pci
worker-2 ansible_user=user1 pcie_addr=0000:09:00.0 dpdk_driver=mlx5_core
```

Setup Kubernetes Cluster

Next, setup the Kubernetes cluster by executing the `create-cluster.yaml` playbook. The playbook takes multiple override parameters that slightly modify its behavior.

To execute the playbook without any override parameters, run `ansible-playbook -i inventory.ini -K create-cluster.yaml`.

The playbook will operate in a few stages.

Stage 1: Install necessary packages and configuration

1. Install packages to use apt over HTTPS
2. Install python3 and pip
3. Install required python packages via pip
4. Add the Docker apt repository and install Docker CE
5. Add remote user to the docker group
6. Disable swap
7. Add Kubernetes apt repository and install Kubernetes packages
8. Clean up any prior K8s clusters
9. Configure `containerd` to use systemd cgroups

Stage 2: Create and bind VFs

The playbook will create 2 VFs per PF and note the VF vendor/device ID for each worker node. It will also bind the VFs to the designated Linux driver for DPDK.

Stage 3: Create and trust a self-signed certificate

The playbook will create a self-signed certificate on the controller node, and have each node trust it. This is used by the docker registry to communicate over HTTPS.

Stage 4: Setup Kubernetes controller node

The playbook will perform the following steps on the controller node:

1. Start the Kubernetes control plane using `kubeadm`
2. Allow the controller node user to use `kubectl` to interact with the cluster
3. Install Calico CNI
4. Copy the command to join worker nodes to the cluster to the management node
5. Start a private docker registry using the self-signed certificate
6. Generate and apply a configuration for the SR-IOV Device Manager
7. Install Multus CNI
8. Apply a Multus configuration

Stage 5: Setup the Kubernetes worker node(s)

The playbook will perform the following steps on the worker nodes:

1. Get a list of non-isolated CPUs
2. Join the Kubernetes cluster
3. Configure the kubelet to use the static CPU policy & dedicate isolated CPUs to Pods
4. Build an SR-IOV CNI image for Arm & push to the controller's private registry (performed by only one worker node)
5. Install SR-IOV CNI
6. Install the SR-IOV Device Plugin

Override Options

This solution allows for modifying its behavior either by setting variables. To set certain variables at run-time, follow these docs.

Force VF creation

The default behavior of VF creation for a certain PCIe address would just try to create a certain number (2 by default) of VFs under it, but it may fail and show error like this:

```
echo: write error: Device or resource busy
```

which is due to existing VFs which have been created before. To override this error condition, set the `force_vf_creation` to `true`, which would clear prior VFs before creating new VFs. Only set this option if the existing VFs are not used now. The default value of `force_vf_creation` is `false`.

Skip VF creation

To skip VF creation, set the `pcie_addr_is_vf` variable to `true`. If VF creation is skipped, the PCIe addresses in the `inventory.ini` will be used directly by application pods. This differs from the default behavior where the supplied PCIe addresses are used to create VFs which are then dedicated to application pods.

Modify Pod CIDR

Each K8s Pod is assigned its own IP address. It is important the IP block for pods has no overlap with other IPs on the network. To change the Pod CIDR, set `pod_cidr` to an unoccupied CID.

Supply additional arguments to `kubeadm init`

Any additional arguments needed to be supplied to `kubeadm init` can be done so by setting `kubeadm_init_extra_args` to a string.

Use VFIO without IOMMU

When deploying to a platform without an IOMMU (like a virtual machine), the `vfio-pci` kernel module needs a parameter set. By setting `no_iommu` to 1, the playbook will take care of loading the kernel module properly.

Change number of VFs per PF

Set `num_vfs` to the number of VFs to create for each PF.

Self-signed certificate directory

Set `cert_dir` to place the self-signed certificates in the specified directory. By default, they will be placed in `~/certs` on the controller node.

Timeout for Nodes to be Ready

Set `node_wait_timeout` to configure how long to wait for all K8s nodes to reach the Ready state. If any node is not ready by the end of the timeout, the playbook will exit with error. The wait occurs after joining worker nodes to the K8s cluster (if not a single-node cluster), but before building/installing the SR-IOV CNI. The default is 600s, or 10 minutes.

Example

For example, the following command sets all possible overrides:

```
ansible-playbook -i inventory.ini -K create-cluster.yaml -e @vars.yaml
```

The `-e` parameter loads variables from the `vars.yaml` file. In this example, it contains:

```
pcie_addr_is_vf: true
pod_cidr: 192.168.54.0/24
kubeadm_init_extra_args: "--apiserver-advertise-address=\"192.168.0.24\" --apiserver-
↪cert-extra-sans=\"192.168.0.24\""
no_iommu: 1
num_vfs: 5
cert_dir: ~/my-cert-dir
node_wait_timeout: "300s"
```

If the user is sure that VFs can be created on the desired PF PCIe address, a tag of `force_vf_creation` can be added and set to true when `pcie_addr_is_vf` is false:

```
force_vf_creation: true
```

1.3.7 Porting/Integrating to another Arm platform

Although the solution is tested on the platforms listed in the *Tested Platforms* section, the solution should work on other Arm platforms. However, such platforms should support Arm v8 architecture at least and be supported by the underlying components.

1.3.8 Sample Applications

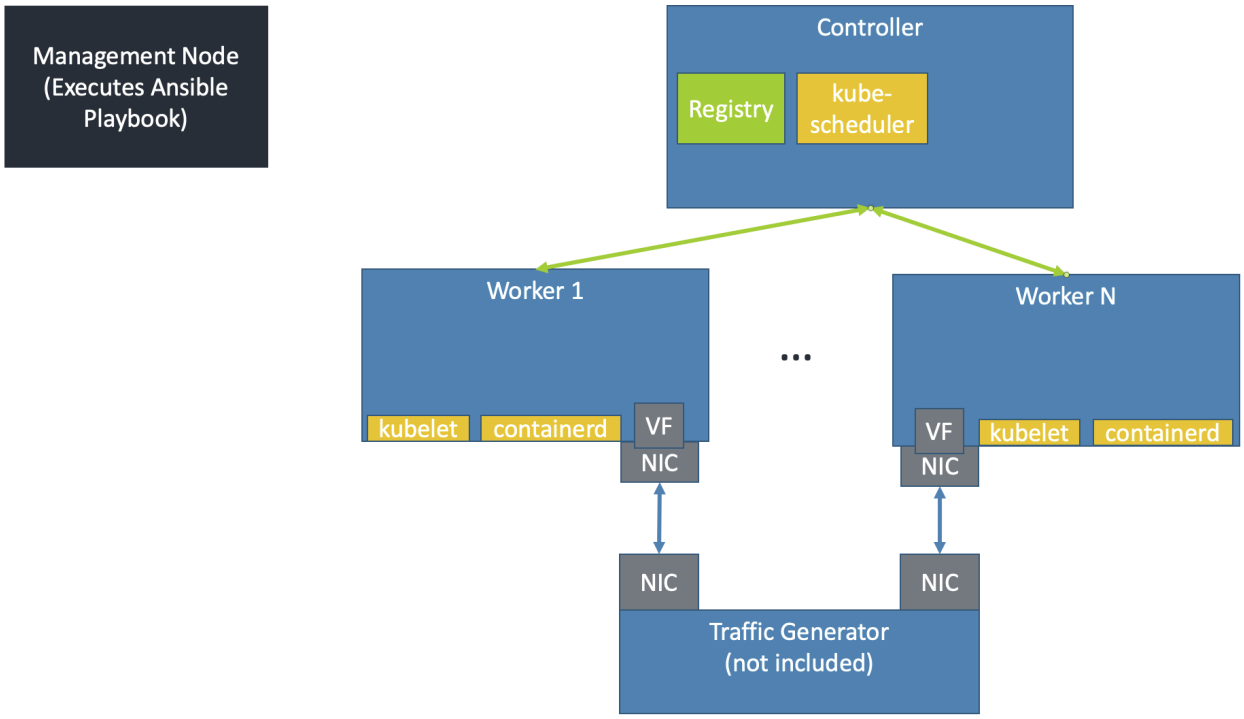
DPDK L3FWD

Introduction

The `dpdk-l3fwd` sample application demonstrates the use of the hash, LPM and FIB based lookup methods provided in DPDK to implement packet forwarding using poll mode or event mode PMDs for packet I/O. The instructions provided in this guide do not cover all the features of this sample application. Users can refer to dpdk-l3fwd user guide to learn and experiment additional features.

Test Setup

This guide assumes the following setup:



The Kubernetes cluster must have been set up by the `create-cluster.yaml` playbook. As shown, every worker node should be connected to the traffic generator & have PFs/VFs available to bind to application pod(s).

Get NIC PCIe Addresses

The PCIe address of the NIC connected to the traffic generator must be known for every worker. Additionally, the DPDK Linux driver for that NIC must be known. While this solution does support using multiple PCIe addresses on a single worker node, every PCIe address for a node must work with the same DPDK Linux driver. Different worker nodes may use different DPDK Linux drivers.

Ensure the PCIe addresses and DPDK Linux drivers for each worker are listed in the inventory file, as described in *Create Ansible Inventory File*.

Execute the DPDK-L3fwd Ansible Playbook

Execute the Ansible playbook in the `<nw_cra_workspace>/cnf-reference-arch/examples/dpdk-l3fwd/` directory by running:

```
cd <nw_cra_workspace>/cnf-reference-arch/examples/dpdk-l3fwd/
ansible-playbook -i ../../inventory.ini dpdk-l3fwd.yaml
```

This playbook will execute the following steps:

1. Expands the DPDK deployment template to the controller node. By default, it will be placed at `~/dpdk-deployment.yaml`.

2. Generates a ConfigMap onto the controller node and adds it to the cluster. By default, it will be placed at ~/configmap.yaml.
3. Copies files needed to build the dpdk docker image to temporary directory on the worker node
4. One worker node builds and pushes the image to the private docker registry on the controller node

The directory which hosts the dpdk-deployment.yaml and configmap.yaml files is controlled by the output_dir parameter. To put those files in a different directory, add -e output_dir=path to the ansible-playbook command above. For example, to place the files in ~/different_output_dir, the full command would look like:

```
ansible-playbook -i ../../inventory.ini dpdk-l3fwd.yaml -e output_dir=~/different_output_
↪dir
```

Once the playbook successfully executes, ssh into the controller node and run:

```
cd <output_dir>
kubectl apply -f dpdk-deployment.yaml
```

This will create pods running the dpdk-l3fwd application. While the number of pods created equals the number of worker nodes, it is up to the Kubernetes scheduler to decide on which nodes the pods will run.

Test

Monitor the application by running kubectl get pods on the controller node. It may take some time for the pods to start up. Once the pods are in the Running state, their logs can be viewed with kubectl logs <pod name>. The pod name can be obtained from the kubectl get pods command. To get more information about the pods (such as which node it is running on), use kubectl get pods -o wide or kubectl describe pod <pod name>.

The logs of each pod should contain something similar to:

```
Initializing port 0 ... Creating queues: nb_rxq=1 nb_txq=1...
Address:98:03:9B:71:24:2E, Destination:02:00:00:00:00:00, Allocated mbuf pool on socket 0
LPM: Adding route 198.18.0.0 / 24 (0) [0001:01:00.0]
LPM: Adding route 2001:200:: / 64 (0) [0001:01:00.0]
txq=2,0,0
```

These logs show port 0 has MAC address 98:03:9B:71:24:2E with PCIe address 0001:01:00.0 on the worker node. 1 IPv4 route matching the subnet 198.18.0.0/24 is added.

Configure the traffic generator to send packets to the NIC port, using the MAC and IP address displayed in the logs. In this example, use a destination MAC address of 98:03:9B:71:24:2E and a destination IP of 198.18.0.21. Then, dpdk-l3fwd will forward those packets out on port 0.

Stop

To stop the application pods, delete the deployment with kubectl delete deploy dpdk. To clean up the K8s cluster, run sudo kubeadm reset -f and sudo rm -rf /etc/cni/net.d on controller and worker nodes.

Suggested Experiments

The example provided above covers a very simple use case of the DPDK L3fwd application. Users are encouraged to experiment with various options provided by the application.

Some experiments will involve changing the DPDK source code and/or the command line arguments to `dpdk-l3fwd`. Changes to these will require updating the deployed container image.

There are multiple ways to update the source code/executable in the sample application.

This guide recommends developing a patch locally & generate a patch file with `git format-patch`. The DPDK contributor guidelines have some examples on creating such a patch file. Then, place the patch file in the `<nw_cra_workspace>/cnf-reference-arch/examples/dpdk-l3fwd` directory, and modify the `dpdk-l3fwd.yaml` playbook to copy the patch file to the workers. See the tasks copying the `Dockerfile` and `dpdk-launch.sh` files for reference. Finally, modify the `Dockerfile` to apply your patch (e.g. using `git apply`) before the `ninja` step.

Once the changes are in place, it is important they are used to generate an updated container image. To ensure the changes are always included, add `force_source: yes` to the `docker_image` task in `dpdk-l3fwd.yaml`. Once this is in place, an updated container image can be built at any time by re-running the `dpdk-l3fwd.yaml` playbook.

The users are also encouraged to try the following options to understand the performance and scalability possible with Arm platforms.

- Number of RX/TX ring descriptors: This can affect the performance in multiple ways. For example, if the worker node is capable of storing the incoming packets in system cache, the incoming packets can trash the system cache, reducing the overall performance. To understand how these affect the performance, experiment by changing the number of descriptors. Change `RTE_TEST_RX_DESC_DEFAULT` and `RTE_TEST_TX_DESC_DEFAULT` in file `l3fwd.h` and update the container image.
- `--config`: This parameter assigns the NIC RX queues to CPU cores. It is possible that a single queue might not be able to saturate a single CPU core. One can experiment by assigning multiple queues to a single core. For example, the option `--config='(0,0,1),(0,1,1)'` assigns the queues 0 and 1 of port 0 to lcore 1. Ensure that Receive Side Scaling (RSS) distributes the packets equally to all the enabled queues by sending multiple flows of traffic. Modify this parameter in `dpdk-launch.sh` and update the container image.
- CPU Scalability: Add more VFs to the deployment by increasing the `arm.com/dpdk` and CPU requests/limits. Ensure the limits/requests for CPU and memory remain equal, otherwise the Pod will no longer be given dedicated CPUs. Additionally, update the `k8s.v1.cni.cncf.io/networks` annotation to repeat `sriov-dpdk` as many times as `arm.com/dpdk` is listed in the limits/requests sections. For example, to request 3 VFs for a single pod, then set `k8s.v1.cni.cncf.io/networks: sriov-dpdk, sriov-dpdk, sriov-dpdk` and `arm.com/dpdk: 3` for both limits and requests. Ensure that Receive Side Scaling (RSS) distributes the packets equally to all the enabled queues by sending multiple flows of traffic.
- Route Scalability: Add additional routes and multiple flows of traffic that exercise these routes. Additional routes can be added such that the accessed data size is more than the available L1, L2 or system cache size.

  To change forwarding rules, edit the global constants in:

  - `main.c`: edit the `ipv4_l3fwd_route_array` or `ipv6_l3fwd_route_array` to adjust default routes for FIB or LPM lookup.

It is also possible to compile additional sample applications and run them following DPDK's Sample Applications User Guide. To compile them, update the `ninja` step in the `Dockerfile`. To run them, modify `dpdk-launch.sh` accordingly.

1.4 Solution Design

1.4.1 Introduction

Welcome to the CNF reference architecture solution design documentation. To best understand this document, an understanding of Kubernetes and network virtualization concepts (i.e. CNI, SR-IOV, VF) is recommended.

This document details design aspects that tailor the cloud-native environment for networking workloads. This is mainly two items:

- Dedicating isolated CPUs to networking workload Pods
- Enabling the Kubernetes control plane to manage NIC virtual functions (VF)

In the cloud-native paradigm, Pods declare the resources they need, and Kubernetes provides them. This means the Kubernetes cluster has to be configured to understand and provision the types of resources networking workload Pods will need. Additionally, Pods need to know how to request the resources they need (i.e., dedicated CPUs and VFs).

Additionally, this document discusses how the local docker registry is created and used.

1.4.2 Cluster Configuration

Dedicated Isolated CPUs

Networking workloads expect to run on dedicated CPUs that are isolated from the Linux kernel scheduler. Isolation is provided by the `isolcpus` kernel command line parameter.

First, Kubernetes has to be configured to provide Pods dedicated CPUs. For this, the `kubelet` on each worker has to be configured to use the static CPU policy. This policy manages a pool of CPUs that pods can share. When a Pod is granted a dedicated CPU, that CPU is removed from this pool and placed in that Pod's `cpuset`. Therefore, there is only one Pod on that CPU.

Once Pods are able to utilize dedicated CPUs, Kubernetes has to be configured to only use isolated CPUs for Pods. This is achieved by setting the `reservedSystemCPUs` field of the kubelet configuration file. This field defines what CPUs are reserved for host level system threads and Kubernetes related threads. So, if this is set to all non-isolated CPUs, then all host level system threads and Kubernetes related threads run on non-isolated CPUs.

Combining these enables Pods to obtain exclusive, dedicated CPUs isolated from the rest of the system.

Enabling Kubernetes to Manage Virtual Functions

Kubernetes enables management of devices via Device Plugins. This enables the Kubernetes control plane to understand what devices are available on which nodes, and assign them to specific Pods.

This solution utilizes the SR-IOV Network Device Plugin to make Virtual Functions (VFs) allocatable to Pods. The SR-IOV Device Plugin runs on every worker node to get information on all available VFs. Based upon the configured selectors, VFs are added under resource names. This solution groups all VFs intended for its use under the `arm.com/dpdk` name.

Once the VFs are grouped together on each node, the SR-IOV Device Plugin makes this known to the Kubernetes control plane through each node's `kubelet`. Once this is done, the Kubernetes control plane understands which VFs are available on which nodes, and can dedicate specific VFs to specific Pods.

Enabling Virtual Functions to be Added to Pods

The SR-IOV Device Plugin by itself is not sufficient to provide Pods access to VFs. The SR-IOV CNI must be used to add VFs as a separate network interface to Pods.

SR-IOV CNI is unable to provide a default Pod network, and thus relies upon a meta CNI such as Multus CNI. Multus CNI enables creation of "multi-homed" pods, or pods with multiple network interfaces. This enables standard CNIs (e.g. Calico, Cilium, Flannel) to provide the Kubernetes-aware networking (Pod to Pod, Pod to service, etc.) while SR-IOV CNI focuses on "just" adding the VF as another network interface to the Pod.

To summarize, the SR-IOV Device Plugin enables Kubernetes to understand and allocate VFs to Pods. The SR-IOV CNI is needed to add the VF as a secondary network interface to a Pod. To enable a multi-homed Pod to use SR-IOV CNI, a meta CNI like Multus is needed.

Multus CNI is configured using a NetworkAttachmentDefinitions. The name of the `NetworkAttachmentDefinition` is used by Pods to request additional network interfaces. A single `NetworkAttachmentDefinition` invokes a single CNI to provide an additional interface. A Pod can ask Multus to invoke any number of CNIs any number of times for any number of interfaces.

This solution needs Multus CNI to add additional interfaces for VFs using SR-IOV CNI. To accomplish this, the k8s.v1.cni.cncf.io/resourceName is added to the `NetworkAttachmentDefinition` metadata. This is needed so Multus will provide SR-IOV CNI with the necessary device information. This solution names the `NetworkAttachmentDefinition` as `sriov-dpdk`.

1.4.3 How Pods Can Utilize These Resources

Dedicated and Exclusive CPUs

Pods must have the `Guaranteed` Quality of Service to be allocated exclusive, dedicated CPUs. This means:

- Every container in the Pod must have a memory limit and memory request
- For every container in the pod, the limit must equal the request
- The same is true for CPU requests

Virtual Function Allocation and Use

Pods first have to declare they need a VF resource. This is done in the same `requests` and `limits` fields as the CPU and memory resources. Since this solution puts every VF into the `arm.com/dpdk` resource name, requesting one VF is done by putting `arm.com/dpdk: 1` into both `limits` and `requests`. For example, the following snippet is used in the DPDK sample application deployment:

```
resources:
  # Limits and requests must be equal for cpu and memory for the container to be pinned␣
↪to CPUs.
        limits:
                hugepages-2Mi: 1Gi
                cpu: 1
                memory: 2Gi
                arm.com/dpdk: 1
        requests:
                cpu: 1
                memory: 2Gi
                arm.com/dpdk: 1
```

Inside the Pod, an environmental variable is set to inform the Pod which resource it has been allocated. In the case of this solution, the `arm.com/dpdk` resource name means the environmental variable is called `PCIDEVICE_ARM_COM_DPDK`. See `examples/dpdk-l3fwd/dpdk-launch.sh` for examples using this variable.

In addition to requesting the VF, the Pod must have Multus add the VF via SR-IOV CNI. This solution has configured the SR-IOV CNI to be invoked with the Multus network name `sriov-dpdk`. So all the pod must do is add the `k8s.v1.cni.cncf.io/networks: sriov-dpdk` annotation to its metadata.

1.4.4 Local Docker Registry

The controller node sets up a Docker registry for this solution to use. This registry holds the container images for the AArch64 build of the SR-IOV CNI and the sample application.

To setup the registry and have it be used by the other nodes in the cluster, the following steps are followed:

1. Create a self-signed certificate for the FQDN of the controller node
2. Trust the certificate for every node in the cluster
3. Launch the registry using the self-signed certificate
4. Nodes interact with the registry using the controller's FQDN

If additional worker nodes are added to the cluster at a later time, they will need to trust the self-signed certificate. Otherwise, the additional nodes cannot pull the SR-IOV CNI Docker image to install it.

1.5 License

1.5.1 Apache-2.0

The software is provided under the Apache-2.0 license (below).

```
                              Apache License
                        Version 2.0, January 2004
                     http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.
```

(continues on next page)

(continued from previous page)

```
   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
```

(continues on next page)

(continued from previous page)

```
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
```

(continues on next page)

(continued from previous page)

```
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.
```

(continues on next page)

(continued from previous page)

```
   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright 2022 Arm, Limited.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

1.5.2 SPDX Identifiers

In the software, individual files contain the SPDX identifiers instead of the full license text. Normally each file uses first line of the file to be SPDX tag. In case of *#!* scripts, SPDX tag can be placed in 2nd line of the file.

For example, to label a file as subject to the Apache-2.0 license, the following text would be used:

```
// SPDX-License-Identifier: Apache-2.0
```

or

```
#!/usr/bin/env bash
# SPDX-License-Identifier: Apache-2.0
```

This enables machine processing of license information based on the SPDX License Identifiers that are here available: http://spdx.org/licenses/

1.5.3 License for patches

The software also bundles patch files, which are applied to the sources of the various packages. Those patches are not covered by the license of this software. Instead, they are covered by the license of the software to which the patches are applied.

When said software is available under multiple licenses, the patches in this software are only provided under the publicly accessible licenses.

1.5.4 Exceptions

Some files in this software contain a different license statement. Those files are licensed under the license contained in the file itself.

The exceptional files and their licenses are recorded as below.

License	file
tools/check-git-log.sh	BSD-3-Clause
tools/git-log-fixes.sh	BSD-3-Clause

1.6 Changelog & Release Notes

1.6.1 CNF Reference Architecture v22.12

New Features

- Automated Kubernetes cluster setup
- Automated private Docker registry setup
- Automated pinning of networking workload to isolated CPU cores

Changed

As the initial version, the features in this release are summarized as below:

Tested platforms:

- DUT
  - Ampere Altra (Neoverse-N1)
    - Ubuntu 20.04.3 LTS (Focal Fossa)
    - Kernel 5.17.0-051700-generic
  - Container Image
    - Latest Ubuntu: 22.04 (as of this release)
- NIC
  - Mellanox ConnectX-5
    - OFED driver: MLNX_OFED_LINUX-5.4-3.1.0.0
    - Firmware Version 16.30.1004 (MT_0000000013)
  - Intel X710
    - Firmware version: 6.01
- Management Node
  - Ubuntu 20.04 system
    - Python 3.8
    - Ansible 6.5.0

Main software components versions:

- Ansible 6.5.0
- DPDK v21.11

Documentation:

- Initial documentation with README, overview, quickstart guide, user guide, solution design and FAQ.

Tools:

- Added scripts for:
  - Sphinx based documentation generation
  - Document spelling check
  - Commit log format check
  - License-Identifier check

Limitations

- Users of this software stack must consider safety and security implications according to their own usage goals.
- Does not provide native traffic generator.
- Application pods using Mellanox VFs require the privileged security context.
- DPDK L3 forward sample application deployed in K8s cluster only supports single Ethernet port case.

Resolved Issues

None.

Known Issues

None.

1.7 FAQ

1. Q: Is there any existing open source project with similar goals or with which this project may conflict or overlap? If yes, please identify the project and explain why we should create the ARM open source project anyway?

   A: Yes, there are some open source projects serve similar purpose, e.g., OPNFV. However, compared with the software we would like to create, OPNFV has a much larger scope for us to cover. Moreover, we would like to simplify the reference solutions, focus on combining some currently being contributed networking projects, e.g., DPDK, VPP, ODP, and provide reference solutions aligned with existing marketing requirements.

   Inside the project, we will apply some optimizations applicable on Arm platform only, to achieve better performance, and those architecture specific optimizations will not be likely accepted by upstream community.

2. Q: Could you please provide the project description of functionality or purpose?

   A: The network functions this software provided serves multiple purposes of,

   1. Showcase the integration of various components and act as poof of concept to all stakeholders.

2. Allow for performance analysis/optimization with a solution that is close to customers' production deployment.
3. Provide customers with a out-of-the-box reference design for rapid design modeling.

3. Q: How to install Mellanox ConnectX-5 OFED driver and update NIC firmware?

   A: To use Mellanox NIC, firstly install the OFED driver MLNX_OFED, and then update NIC Firmware.

   The key steps are:

   - Download the OFED driver
   - Install OFED driver:

   ```
   $ sudo mount -o ro,loop MLNX_OFED_LINUX-5.4-3.1.0.0-ubuntu20.04-aarch64.iso /mnt
   $ sudo /mnt/mlnxofedinstall --upstream-libs --dpdk
   ```

   - Update firmware after OFED installation:

   ```
   $ wget https://www.mellanox.com/downloads/MFT/mft-4.20.0-34-arm64-deb.tgz
   $ tar xvf mft-4.20.0-34-arm64-deb.tgz
   $ cd mft-4.20.0-34-arm64-deb/
   $ sudo ./install.sh
   $ sudo mst start
   $ sudo mlxfwmanager --online -u -d <device PCIe address>
   ```

4. Q: How to enable SR-IOV VFs for Mellanox ConnectX-5?

   A: First, ensure the OFED driver and firmware tools are installed by following the steps above. Next, use this guide to enable SR-IOV on your platform.

   The key steps are:

   - Enable SR-IOV in BIOS settings
   - Enable SR-IOV on NIC firmware and set maximum number of VFs to a non-zero number:

   ```
   $ sudo mst status # lists NIC under "MST Devices"
   $ sudo mlxconfig -d /dev/mst/mt4121_pciconf0 set SRIOV_EN=True NUM_OF_VFS=4 #␣
   ↪replace /dev/mst/... with your MST Device
   ```

   - Reboot the machine

5. Q: How to update firmware for Intel 700 Series NICs?

   A: To update the NIC firmware, first install the NIC on an x86 host. Then, install the NVM Update Utility on the x86 host and follow the included readme.

   The key steps are:

   - Insert the NIC into an x86 host
   - Download the NVM Update Utility
   - Unzip the package & extract the Linux version:

   ```
   $ unzip 700Series_NVMUpdatePackage_v8_70.zip
   $ tar xvf 700Series_NVMUpdatePackage_v8_70_Linux.tar.gz
   ```

   - Run the NVM Update Utility:

```
$ cd 700Series/Linux_x64/
$ sudo ./nvmupdate64e
```

- Press A to update all NICs, or specify a list of cards to update
- Reboot the machine

6. Q: How to setup SSH keys?

   A: SSH keys can be setup on Ubuntu 20.04 by following this guide.

   The key steps are:

   - Check whether SSH key already exists in `~/.ssh/id_rsa.pub`. If not, create new authentication key pairs for SSH:

   ```
   $ ssh-keygen
   Generating public/private rsa key pair.
   Enter file in which to save the key (/home/<remote_user>/.ssh/id_rsa):
   Created directory '/home/<remote_user>/.ssh'.
   Enter passphrase (empty for no passphrase):
   Enter same passphrase again:
   Your identification has been saved in /home/<remote_user>/.ssh/id_rsa
   Your public key has been saved in /home/<remote_user>/.ssh/id_rsa.pub
   ```

   - Copy SSH public key to Arm machine. `remote_user` is the user name on Arm machine and `dut.arm.com` is the FQDN of Arm machine:

   ```
   $ ssh-copy-id remote_user@dut.arm.com
   ```

   - Make sure `remote_user` can login Arm machine from management node without entering password:

   ```
   $ ssh remote_user@dut.arm.com
   ```

7. Q: Can Ansible use SSH keys protected with a passphrase?

   A: Yes, by using the `ssh-agent` utility. Follow these steps to set it up.

8. Q: How to setup DUT with isolated CPUs and 1G hugepages?

   The key steps are:

   - Edit variable `GRUB_CMDLINE_LINUX_DEFAULT` in file `/etc/default/grub` to reserve hugepages using parameters `default_hugepagesz=1G hugepagesz=1G hugepages=4` and reserve isolated CPU cores using parameters `isolcpus=1-3 nohz_full=1-3 rcu_nocbs=1-3 cpuidle.off=1 cpufreq.off=1`

   ```
   $ grep GRUB_CMDLINE_LINUX_DEFAULT /etc/default/grub | grep -v "#"
   GRUB_CMDLINE_LINUX_DEFAULT="default_hugepagesz=1G hugepagesz=1G hugepages=4␣
   ↪iommu.passthrough=1 isolcpus=1-3 nohz_full=1-3 rcu_nocbs=1-3 cpuidle.off=1␣
   ↪cpufreq.off=1"
   ```

   - Enable above kernel parameters by executing `sudo update-grub` and rebooting the system:

   ```
   $ sudo update-grub
   $ sudo reboot now
   ```

   - Make sure kernel parameters take effect after reboot

```
$ cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-5.19.10-051910-generic-64k root=/dev/mapper/ubuntu--vg-
↪ubuntu--lv ro default_hugepagesz=1G hugepagesz=1G hugepages=4 iommu.
↪passthrough=1 isolcpus=1-3 nohz_full=1-3 rcu_nocbs=1-3 cpuidle.off=1 cpufreq.
↪off=1
$ grep Huge /proc/meminfo
HugePages_Total:       4
HugePages_Free:        4
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:    1048576 kB
$ grep Cpus_allowed_list /proc/self/status
Cpus_allowed_list:     0,4-191
```

9. Q: Why isolate CPUs with these parameters?

   A: The combination of the kernel parameters mentioned above ensure that the CPUs run only the desired application. They never process kernel RCU callbacks (`rcu_nocbs`), don't generate scheduling ticks (`nohz_full`) when there is only one process running on them, and are isolated from running any processes other than the ones pinned to them (`isolcpus`). Additionally, the `cpufreq` and `cpuidle` subsystems are disabled.

10. Q: How do I make `python3 -c 'import socket; print(socket.getfqdn())'` show the FQDN?

    A: If the above python command fails to print out the correct FQDN, then your system may be affected by this python bug. The solution is to modify `/etc/hosts` to hardcode the correct FQDN. To do so, follow the steps in the Debian manual to modify `/etc/hosts`.

11. Q: How to build HTML rendered document?

    A: Install following dependencies and run `make doc`:

```
sudo apt-get update
sudo apt-get install python3-venv libenchant-dev -y
cd <nw_cra_workspace>/cnf-reference-arch
make doc
```

12. Q: How to resolve `create VF` task failure `echo 2 > /sys/bus/pci/devices/<pcie_addr>/sriov_numvfs ... echo: I/O error`?

    A: CRA is most easily deployed with exclusive control over a PF. Ensure no other applications like DPDK or QEMU are using the PF. Additionally, ensure the PF is bound to the default kernel driver such as `i40e` or `mlx5_core`.

    1. Check NIC port is used by other applications like QEMU, DPDK, stop them if possible.
    2. Examine NIC is bind with `vfio-pci` driver, this can be done with `driverctl -v list-devices | grep vfio`.
    3. Bind with NIC's default driver, take `i40e` as example, bind with `driverctl set-override <pcie_addr> i40e`.
    4. Re-deploy CRA solution.

    CRA can also be deployed onto specific VFs on a machine. This can be used to share underlying PFs with other applications like DPDK or QEMU. To leverage this deployment model, specify the VF PCIe addresses in the `pcie_addr` for the worker node, and set the `pcie_addr_is_vf` flag. See the user guide for more information.