USER GUIDE

Intel Corporation
Software Defined Datacenter Solutions Group

Deploying Kubernetes and Container Bare Metal Platform for NFV Use Cases with Intel® Xeon® Scalable Processors

Table of Contents

1.0 Introduction

Kubernetes, the leading container orchestration engine (COE), provides an open source system for automating deployment, scaling, and management of containerized applications. To enhance Kubernetes for network functions virtualization (NFV) and networking uses, Intel® and its partners are developing the following:

- **Multus**: a plugin that enables multiple network interfaces for pods within a Kubernetes container
- **Enhanced Platform Awareness**: a suite of capabilities and methodologies that exposes Intel Architecture platform features for increased and deterministic application and network performance
- **Dataplane Acceleration**: Includes vhost-user and SR-IOV to improve data throughput.

Multus expands the networking capability of Kubernetes, which is currently limited to a single network interface per pod. Support for multiple interfaces is a key requirement for many virtual network functions (VNFs). Traditionally, these VNFs employ multiple network interfaces to provide for separation of control, management and data/user network planes. They are also used to support different protocols or software stacks and different tuning and configuration requirements.

Enhanced Platform Awareness (EPA) for Kubernetes represents a methodology and a related suite of changes across multiple layers of the orchestration stack targeting intelligent platform capability, configuration & capacity consumption. Specifically, EPA underpins the three-fold objective of the discovery, scheduling and isolation of server hardware capabilities. Intel and partners have worked together to progress this strategy further through the following technologies:

- Node Feature Discovery (NFD) enables Intel Xeon® Processor-based server hardware capability discovery in Kubernetes
- CPU Manager for Kubernetes (CMK) provides a mechanism for CPU core pinning and isolation of containerized workloads
- Huge page support, added to Kubernetes 1.8, enables the discovery, scheduling and allocation of huge pages as a native first-class resource

To help developers adopt and utilize these advanced networking technologies, Intel integrated EPA, Multus and data plane acceleration features into a container bare-metal reference architecture for Intel Xeon Scalable processors. This reference architecture represents a baseline configuration of components that are combined to achieve optimal system performance for applications running in a container-based environment.

The intended audience of this user guide includes system architects, developers and engineers that are involved in developing, testing or deploying NFV workloads in a Kubernetes environment and need to put together a software solution on a bare metal server. The content includes a high-level architectural overview and procedures for setup, configuration and provisioning.

This document is part of the Container Experience Kit for Kubernetes Networking. Container Experience Kits are collections of user guides, application notes, feature briefs and other collateral that provide a library of best-practice documents for engineers who are developing container-based applications. Other documents in this experience kit can be found online at: https://networkbuilders.intel.com/network-technologies/container-experience-kits.

2.0 Hardware components

This section lists the hardware components and systems that were utilized in this reference architecture. More details on the software involved are in Section 3.0. Intel Xeon Scalable processors feature a scalable, open architecture designed for the convergence of key workloads such as applications and services, control plane processing, high-performance packet processing and signal processing.

2.1 Intel Xeon Scalable Processor-based platform

Table 2-1 Intel Xeon Scalable Family platform components used in integration tests.

Item	Description	Notes
Platform	Intel Server Board S2600WFQ	Intel Xeon processor-based dual-processor server board 2 x 10 GbE integrated LAN ports
Processors	2x Intel Xeon Platinum 8176 Processor	28 cores, 56 threads, 2.1 GHz, 165 W, 38.5 MB L3 total cache per processor, 3 UPI Links, DDR4-2666, 6 memory channels
	2x Intel Xeon Platinum 8160 Processor	24 cores, 48 threads, 2.1 GHz, 150 W, 33 MB L3 total cache per processor, 73 UPI Links, DDR4-2666, 6 memory channels
	2x Intel Xeon Gold 6152 Processor	22 cores, 44 threads, 2.1 GHz, 140 W, 30.25 MB L3 total cache per processor, 3 UPI Links, DDR4-2666, 6 memory channels
	2x Intel Xeon Gold 6138T Processor	20 cores, 40 threads, 2.0 GHz, 125 W, 27.5 MB L3 cache per processor, 3 UPI Links, DDR4-2666, 6 memory channels
	2x Intel Xeon Gold 6130T Processor	16 cores, 32 threads, 2.1 GHz, 125 W, 22 MB L3 cache per processor, 3 UPI Links, DDR4-2666, 6 memory channels
Memory	192 GB total; Micron* MTA18ASF2G72PDZ	12x DDR4 2400 MHz, 16 GB
	192 GB total; SK Hynix* HMA42GR7AFR4N	12x DDR4 2400 MHz, 16 GB
Networking	Intel Ethernet Network Adapter XXV710–DA2	2 x 1/10/25 GbE ports Firmware version 5.50 Tested with Intel SFP28 Twinaxial Cables for Intel Ethernet Network Adapter XXV710
	Intel Ethernet Converged Network Adapter X710-DA4	Intel Ethernet Master XL710-AM1 4 x 10 GbE ports Firmware version 4.53 Tested with Intel FTLX8571D3BCV-IT and AFBR 703sDZ IN2 transceivers
	Intel Ethernet Converged Network Adapter XL710-QDA2	Intel Ethernet Master XL710-AM2 2 x 40 GbE ports Firmware version 4.53 Tested with Intel E40QSFPSR transceiver
	Intel Ethernet Converged Network Adapter X540-T2	Intel Ethernet Master X540-BT2 2 x 10 GbaseT ports
	Intel Ethernet Converged Network Adapter X520-SR2	Intel 82599ES 10 Gigabit Ethernet Master 2 x 10 GbE ports Tested with Intel FTLX8571D3BCV-IT transceiver
Local Storage	Intel SSD DC S3500 Series	SSDSC2BB120G4 120 GB SSD 2.5in SATA 6GB/s
	Intel SSD DC P3700 Series	SSDPE2MD800G4 800 GB SSD 2.5in NVMe/PCIe
BIOS	Intel Corporation SE5C620.86 B.0X.01.0007.060920171037 Release Date: 06/09/2017	Intel Hyper-Threading Technology (Intel HT Technology) enabled Intel Virtualization Technology (Intel VT-x) enabled Intel Virtualization Technology for Directed I/O (Intel VT-d) enabled

3.0 Software versions

Table 3-1 outlines the software versions used in this reference architecture.

Table 3-1 Software versions

Software Function	Software Component
Host OS	Ubuntu* Server 16.04.3 Kernel version: 4.10.0
Deployment software	Ansible* 2.3.1 Kubeadm* 1.6.6, 1.8.4
Docker*	Docker* 1.13.1
Container orchestration engine	Kubernetes 1.6.6 & 1.8.4 Kubectl* 1.6.6 & 1.8.4 Kubelet 1.6.6 & 1.8.4 Kubernetes-cni 0.5.1
Kubernetes CNI Plugins	SRIOV-CNI v0.2-alpha MULTUS-CNI v1.0 Flannel 0.7.1
CPU Manager for Kubernetes	CMK v1.1.0 & v1.2.0
Node Feature Discovery	NFD v0.1.0
Data Plane	DPDK 17.05.0
vhost-user-net-plugin	VUNP v1.0

3.1 Obtaining the software

All of the open source software ingredients used can be downloaded from the open source repositories shown in Table 3-2. A specific set of commit IDs were used as part of the integration of this reference architecture. The commit IDs shown in Table 3-3 and Table 3-4 were used at the time of testing this reference architecture.

Table 3-2 Sources for the software ingredients

Component	Location	Comments
Ubuntu 16.04.3	http://releases.ubuntu.com/16.04/	Kernel .4.10.0.30-generic
Bare Metal Container Environment Setup scripts	https://github.com/intel/container-experience-kits	Includes Ansible* scripts to deploy Kubernetes
Ansible 2.3.1	https://github.com/ansible/ansible/releases	Install Ansible 2.3.1 on jumphost. Pip is preferred method for installation pip install ansible==2.3.1
Docker 1.13.1	https://docs.docker.com/engine/installation/	Ansible scripts will install and configure Docker.
Intel Ethernet Drivers	https://sourceforge.net/projects/e1000/files/ixgbe%20stable/5.2.1/ixgbe-5.2.1.tar.gz/download https://sourceforge.net/projects/e1000/files/ixgbevf%20stable/4.2.1/ixgbevf-4.2.1.tar.gz/download https://sourceforge.net/projects/e1000/files/i40e%20stable/2.0.30/i40e-2.0.30.tar.gz/download https://sourceforge.net/projects/e1000/files/i40evf%20stable/2.0.30/i40evf-2.0.30.tar.gz/download	Ansible scripts will install and configure the following network drivers.
Python 2.7	Ubuntu inbox repo can be used	Needed on all nodes apt install python-minimal
Sshpass, Python-pip, Git	Ubuntu inbox repo can be used	Only needed on Jumphost apt install sshpass python-pip git
Ansible	https://pypi.python.org/pypi/ansible	PIP is recommended to install ansible pip install ansible==2.3.1

Table 3-3 Commit IDs for the major software components – Kubernetes 1.6.6

Software Component	Location	Commit IDs
DPDK v17.05	http://dpdk.org/git/dpdk	b2b58b51df4943bcab770b1eb66912eaa41e1306
SR-IOV CNI plugin v0.2-alpha	https://github.com/Intel-Corp/sriov-cni	a2b6a7e03d8da456f3848a96c6832e6aefc968a6
Multus CNI plugin v1.0.0	https://github.com/Intel-Corp/multus-cni	1aec06ca558883f80cde72ccc5f8f5e1b1725335
NFD v0.1.0	https://github.com/Intel-Corp/node-feature-discovery	41da99a4359939950b7a05e8e73eeb44ebb63dd7
CMK v1.1.0	https://github.com/Intel-Corp/CPU-Manager-for-Kubernetes	c7d59be0fa0f446434cc0288c09bbd3396d798d9

Table 3-4 Commit IDs for the major software components – Kubernetes 1.8.4

Software Component	Location	Commit IDs
DPDK v17.05	http://dpdk.org/git/dpdk	b2b58b51df4943bcab770b1eb66912eaa41e1306
SR-IOV CNI plugin v0.2-alpha	https://github.com/Intel-Corp/sriov-cni	a2b6a7e03d8da456f3848a96c6832e6aefc968a6
Multus CNI plugin v2.0.0	https://github.com/Intel-Corp/multus-cni	b2d67c8909b940ebf8eae5e87a4ac8897e3ef9d
NFD v0.1.0	https://github.com/Intel-Corp/node-feature-discovery	41da99a4359939950b7a05e8e73eeb44ebb63dd7
CMK v1.2.1	https://github.com/Intel-Corp/CPU-Manager-for-Kubernetes	945b10c5039a4cabdf91a90957afe1e5bc65a4b2
VUNP v1.0	https://github.com/intel/vhost-user-net-plugin	9c2dc4306398aace5c0aa8037e332b24bd00df68

4.0 Kubernetes cluster deployment software overview

This section describes how the software listed in Table 3-1 was used to create the Kubernetes cluster that is the basis of this reference design. Ansible scripts were developed to utilize these components to deploy the cluster with enhanced functionality. The location for these software components is found in Table 3-1, Table 3-2 and Table 3-3 in the previous section.

4.0.1 Ansible

Ansible was utilized to deploy Kubernetes and its latest enhanced features. To allow for a predictable deployment, ensure all hosts have the minimal package prerequisites installed and hardware configured accordingly, Table 5-1 & Table 5-2. The Ansible scripts have been tested on and are supported by the Operating Systems listed in Table 3-1 in section 3.0, and have been tested with the software ingredients in Table 3-1, Table 3-2, and Table 3-3. Installation scripts can be obtained from the download link in the Table 3-2.

More information on Ansible can be found on the Ansible website: https://github.com/ansible/ansible

4.0.2 Kubernetes

Kubernetes is an open source platform that automates, deploys, maintains and scales Linux container operations. Kubernetes provides orchestration and management capabilities across multiple containers and multiple server hosts. Flannel with Virtual Extensible LAN (VXLAN) is used to create a layer 3 IPv4 network that provides networking between multiples nodes in a cluster. More information on Kubernetes can be found at Kubernetes site https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/. More information on flannel can be found at the flannel site https://github.com/coreos/flannel.

4.0.3 Kubeadm – creating a Kubernetes cluster

Kubeadm is an experimental multi-node Kubernetes installer. Currently, Kubeadm is limited to a single master, but supports multiple minions. Kubeadm helps simplify the installation of a secure Kubernetes cluster on a server and the deployment of a pod network for application component communication. Ansible utilizes Kubeadm to deploy a Kubernetes cluster. All kubectl commands are run on the Kubernetes Master. More information on Kubeadm can be found at the Kubernetes site https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/.

4.0.4 Container networking interface (CNI)

Container Networking Interface (CNI) consists of specifications and libraries for writing plugins to configure network interfaces in Linux containers. For this reference architecture, Kubernetes will utilize two key CNI plugins, SR-IOV-CNI section 4.1.3 and MULTUS-CNI section 4.1.1, to enhance the pod's network capabilities in addition to the default flannel network fabric.

Kubernetes also requires the standard CNI lo plugin and any plugins referenced by configuration `-cni-confi-dir (/etc/cni/net.d)` must be present in `-cni-bin-dir (/opt/cni/bin)`. More information on CNIs can be found at the container networking GitHub: https://github.com/containernetworking/cni.

4.0.5 Data Plane Development Kit (DPDK)

DPDK is a set of open source libraries and drivers for fast packet processing. It was first designed to run on Intel Architecture processors, but it is now an open source standard that supports other processors. DPDK is an open source BSD-licensed project. DPDK's most recent patches and enhancements, provided by the community, are available in the master branch (http://dpdk.org/browse/dpdk/log/).

More information on DPDK can be found on the DPDK website: http://dpdk.org.

4.1 Platform enhancement software overview

This section provides a quick overview of the features that enhance Kubernetes networking functionality. For software versions, reference Table 3-1, Table 3-2, and Table 3-3.

4.1.1 Multus-CNI

Kubernetes natively supports only a single network interface, however it is possible to implement multiple network interfaces using Multus. Multus is a CNI proxy and arbiter of other CNI plugins. It invokes other CNI plugins for network interface creation. When Multus is used, a master plugin (flannel, Calico, weave) is identified to manage the primary network interface (eno2) for the pod and it is returned to Kubernetes. Other CNI minion plugins (SR-IOV, vhost CNI, etc.) can create additional pod interfaces (net0, net1, etc.) during their normal instantiation process.

More information can be found in section 4.0 of the application note titled "Multiple Network Interfaces in Kubernetes Application Note." This document can be found at the Intel container experience kit website at: https://networkbuilders.intel.com/network-technologies/container-experience-kits.

More information and source code can be found at https://github.com/Intel-Corp/multus-cni.

4.1.2 vhost-user CNI plugin

vhost-user plugin is a CNI plugin that runs with the DPDK-enhanced open vSwitch (OVS-DPDK) and VPP along with the Multus CNI plugin in Kubernetes for bare metal container deployment model. It enhances high performance container networking solutions and data plane acceleration for NFV environment.

More information and source code can be found on https://github.com/intel/vhost-user-net-plugin.

4.1.3 SR-IOV-CNI and DPDK-SR-IOV-CNI plugin

SR-IOV introduces the concept of virtual functions (VF) that represent a regular PCIe physical function (PF) to a VNF. In Kubernetes, SR-IOV is implemented by utilizing an SR-IOV CNI plugin that lets the pod attach directly to the VF. Now, multiple containers within a Kubernetes pod can each have access to their own Ethernet VF. Each VF can be treated as a separate physical NIC and configured with separate MAC, VLAN and IP addresses and other parameters. The VF gets mapped to a specific physical Ethernet port. Additional capabilities with SR-IOV allow binding the VF to a DPD- bound driver (i.e vfio_pci) when a pod is instantiated. As a result, performance is vastly improved as packets move directly between the NIC and the pod.

Note: Multus is required to run SR-IOV, as more than one CNI-plugin are required to support SR-IOV implemented in the form of CNI-plugins. In addition, flannel is required in order to check the health of the pod and for layer three networking between multiple nodes in a cluster.

More information can be found in section 5.0 of the application note titled "Multiple Network Interfaces in Kubernetes Application Note." This document can be found at the Intel container experience kit website at: https://networkbuilders.intel.com/network-technologies/container-experience-kits.

More information and source code can be found on: https://github.com/Intel-Corp/sriov-cni.

4.1.4 Native huge page support in Kubernetes

Up until Kubernetes v1.8, there was no mechanism to discover, request or schedule huge pages as a resource and hence it had to be manually managed. Huge page support has now been introduced as an alpha first-class resource, which enables huge page management natively in Kubernetes. The supported huge page sizes are 2MB and 1GB.

More information can be found in section 5.0 of the application note titled "Enhanced Platform Awareness in Kubernetes Application Note." This document can be found at the Intel container experience kit website at: https://networkbuilders.intel.com/network-technologies/container-experience-kits.

More information on Kubernetes manual huge pages can be found at https://kubernetes.io/docs/tasks/manage-hugepages/scheduling-hugepages/.

4.1.5 CPU Manager for Kubernetes (CMK)

CMK is a tool for managing core pinning and isolation in Kubernetes. CMK creates core isolation by applying CPU masks, which represent cores on which the workload can be executed. The core availability state is maintained in a host file system that incorporates a system lock to avoid any conflicts. The core availability state (allocated or free) is maintained in a host file system that incorporates a system lock to avoid any conflicts. This state of the core is structured as a directory hierarchy where pools are represented as directories. Workloads can acquire slots from these directory structures. These slots represent physically allocable cores in the form of a list of their logical core IDs.

These pools can be exclusive, i.e. only one workload can run per slot, or they can be shared. The slot directory keeps track of processes that have acquired the slots through their process IDs. When a workload has completed its task, CMK will enforce the directory system lock and remove that workload's process ID from the relevant slot in order to free the core so that another workload can use it. In the case where the CMK program is unable to clean up the process IDs due to being killed or terminated unexpectedly, a periodic process is run to act as garbage collection. This frees up cores by removing the process IDs of workloads no longer running.

More information can be found in section 4.0 of the application note titled "Enhanced Platform Awareness in Kubernetes Application Note." This document can be found at the Intel container experience kit website at: https://networkbuilders.intel.com/network-technologies/container-experience-kits.

More information and source code can be found on: https://github.com/Intel-Corp/CPU-Manager-for-Kubernetes.

4.1.6 Node Feature Discovery (NFD)

Node Feature Discovery (NFD), a project in the Kubernetes incubator, discovers and advertises hardware capabilities of a platform which are, in turn, used to facilitate intelligent scheduling of a workload. The NFD script launches a job that deploys a single pod on each node (labeled or unlabeled) in the cluster. NFD can be run on labeled nodes to detect additional features that might have been introduced at a later stage (e.g. introduction of an SR-IOV capable NIC to the node). When each pod runs, it connects to the Kubernetes API server to add labels to the node.

More information can be found in section 3.0 of the application note titled " Enhanced Platform Awareness in Kubernetes Application Note." This document can be found at the Intel container experience kit website at: https://networkbuilders.intel.com/network-technologies/container-experience-kits.

More information and source code can be found on: https://github.com/Intel-Corp/node-feature-discovery

5.0 System prerequisites

This section describes the minimal system prerequisites needed for the jumphost and master/minion nodes.

5.1 Jumphost, master, and minion software prerequisites

Assumptions: It is presumed the master and minion nodes will be freshly imaged, and booted into the correct kernel with internet access and remote root access enabled.

The jumphost requires internet access and root access to the master/minion nodes. Ansible will install additional software dependencies based on the deployment scenario. Minimal package requirements are described in Table 5-1.

Package	Jumpshot	Master/Minion
Python 2.7	yes	yes
Ansible 2.3.1	yes	no
Sshpass	yes	no
PIP	yes	no
Git	yes	no

Table 5-1 package prerequisites

The current scripts support the Operating Systems presented in Section 3.0. Table 3-1 lists the sources to the ISO images of the supported Operating Systems. Users may want to download the preferable image, burn it to a DVD and create an installation disk. During the installation, be sure to select the SSH server.

5.2 Master and minion BIOS prerequisites

Enter the BIOS menu and update the configuration as described in Table 5-2.

Package	Jumpshot	Master/Minion
Intel VT-x	Enabled	Enabled
Intel HT Technology	Enabled	Enabled
Intel VT-d (SR-IOV)	Enabled	Enabled

Table 5-2 BIOS Settings

5.3 Master & minion network interface requirements

Masters require at least a minimum of two network interfaces and minions require a minimum of three network interfaces to deploy a Kubernetes cluster in a lab environment. When using predefined Multus templates, both the master and minion must have the same number of tenant interfaces and naming scheme. Predefined Multus templates use a single configuration file for both master and minions, thus the requirement.

A brief explanation of interfaces is as follows:

- Internet network
  - Jumphost accessible
  - Capable of downloading packages from the internet
  - Can be configured for Dynamic Host Configuration Protocol (DHCP) or with static IP address
- Management network and flannel pod network interface
  - Kubernetes master and minion nodes inter-node communications
  - Flannel pod network will run over this network
  - Configured to use a private static address
- Tenant data network(s)
  - Dedicated networks for traffic
  - SR-IOV enabled
  - VF can be DPDK bound in pod

5.4 Non-predefined network interfaces example

The following is an example of a non-predefined Multus template deployment utilizing four network interfaces configured in deploy.yml.

Master

- **eno1:** For the Internet network – needed to connect to host and used to pull all the necessary packages/patches from repositories on the Internet.
- **enp3s0f3:** Kubernetes management network and flannel (VXLAN) pod network – API server and pod network.

Minion

- **eno1:** For the Internet network – needed to connect to host and used to pull all the necessary packages/patches from repositories on the Internet.
- **enp3s0f3:** Kubernetes management network and flannel (VXLAN) pod network – API server and pod network.
- **ens786f0:** Tenant data network – additional network interface in Pods allowing dedicated traffic .
- **ens786f1:** Tenant data network – additional network interface in Pods allowing dedicated traffic.

5.5 Predefined network interfaces example

Predefined Multus templates are preconfigured for a particular number of interfaces. Both master and minion have the same number of interfaces and names. The following is an example of a predefined Multus template deployment utilizing four network interfaces configured in deploy.yml.

Master

- **eno1:** For the Internet network – needed to connect to host and used to pull all the necessary packages/patches from repositories on the Internet.
- **enp3s0f3:** Kubernetes management network and flannel (VXLAN) pod network – API server and pod network.
- **ens786f0:** Tenant data network – additional network interface in Pods allowing dedicated traffic .
- **ens786f1:** Tenant data network – additional network interface in Pods allowing dedicated traffic.

Minion

- **eno1:** For the Internet network – needed to connect to host and used to pull all the necessary packages/patches from repositories on the Internet.
- **enp3s0f3:** Kubernetes management network and flannel (VXLAN) pod network – API server and pod network.
- **ens786f0:** Tenant data network – additional network interface in Pods allowing dedicated traffic .
- **ens786f1:** Tenant data network – additional network interface in Pods allowing dedicated traffic.

Note: The names of these network interfaces are only examples and will be relative for each installation environment.

Note: Predefined Multus template deployments use a single configuration file for masters and minions, which requires consistent naming across network interfaces. For deployments where network interface names are not consistent, `use _ udev` variable can be set in `deploy.yml` to allow for consistent network interface names.

6.0 Quickstart deployment of Kubernetes with Ansible

Under the examples directory, there are several deployment scenario examples provided. Utilizing these examples, create a file deploy.yml to deploy the feature set needed. Some parameters are global defaults, but are referenced for example. For a detailed explanation of Ansible playbooks and configuration files, go to Section 7.0

6.1 Quickstart deployment

Deploying Kubernetes from the jumphost:

1. Configure inventory.ini
2. Configure deploy.yml with set of features accordingly
3. Run the following Ansible command.

```
# ansible-playbook -i inventory.ini -e @deploy.yml multinode.yml
```

Note: Kubernetes admin credentials are configured on master only via environment, located under `KUBECONFIG=/root/admin.conf`

Note: Software components can be found under `/opt/`

Note: Deployment templates can be found under `{{master}}:/tmp/k8s _ config/`

Note: Setting log variables from Readme.md, ansible deployment logs can be found in `<jumphost>:/logs/<timestamp…>` files.

Note: Socket Secure (SOCKS) proxies will clone repositories using git protocol. If there are issues with SOCKS, change the git settings to use the HTTPS protocol instead using the following command.

```
# git config --system url."https:// ".insteadOf git://
```

7.0 Ansible playbooks and configuration files

The scripts used for this reference architecture project provide a simplified procedure for installing and configuring Kubernetes using Ansible. Based on the preferred type of deployment, the user will need to update a deployment file, deploy.yml. The scripts will take care of preparing the system and will deploy the software ingredients, including:

- Configuring, provisioning, and updating the required OS kernel, services, and network settings
- Installing Kubeadm, Kubelet, Kubectl, and Golang
- Deploying Kubernetes cluster with Kubeadm

The installation scripts on the Container Experience Kit GitHub (https://github.com/intel/container-experience-kits) contain the following files to deploy a Kubernetes cluster:

README.md

This file provides instructions on how to update the
deploy.yml configuration file and run Ansible. Reading this file before installation is highly recommended.

It is important to complete all the installation steps described in the README.md file, prior to deploying Kubernetes.

inventory.ini

This is a node configuration file that defines the nodes and permissions.

- Master and minion node info
- Master and minion group_vars (root username and password)

commit_ids.yml

In order to have a predictable installation and working setup of the required open source software, specific commit IDs are used to ensure validated components are installed. This file contains a list of frozen commit IDs including, but not limited to, Kubernetes or DPDK.

all.yml

This global file contains default parameters that can be specified for deployments located in `` `group _ vars/all/all.yml` `` The intention is to provide an Ansible extra_args file, deploy.yml, which then overrides default parameters per deployment scenario.

deploy.yml*

This file is an Ansible extra_vars deployment scenario file. The template for this file is found in `'examples/deploy _ {scenario}.yml'`, while the examples folder has many examples provided for various deployment topologies. The user would copy the `'examples/deploy _ {scenario}.yml'` file in to the Ansible working directory, take the relevant sections from the suitable file in examples folder and rename it to 'deploy.yml' to create a single deployment file. Following are some of the options that can be configured. A full list of parameters can be found in `` `group _ vars/all/all.yml` ``.

- Kubernetes version
  - 1.6.6 & 1.8.4
- Kubernetes pod network
  - Flannel (default) – Multus CNI is automatically enabled in flannel pod network daemonset.
- Predefined Multus map configuration templates
  - Disabled by default, when enabling ensure master and minion have the same number of tenant interfaces and naming scheme.
  - Allows for simple deployment of 2, 4, 8 tenant data networks.
  - Kubernetes 1.8.4 allows for predefined and custom resources.
- SR-IOV enablement (enabled by default)
  - Number of virtual functions (default is 7).
  - DPDK enablement
  - Configurable huge page size, number, and amount of huge pages to reserve.
- K8s native huge page enablement
  - Kubernetes 1.8.4 has support for native huge pages.
- CMK enablement
  - CMK image.
  - Number of data plane and control plane cores. CPU isolation is based on number of data and control pane cores, starting from CPU core 2.
- Proxy (if applicable)
- Udev (if applicable)
- Node information
  - Hostname (needs to match alias in the `inventory.ini` file).
  - Network interfaces to use for each of the Kubernetes networks.

multinode.yml

This is the main Ansible playbook that orchestrates the Kubernetes deployment. The roles defined here include:

1. k8s_prepare_host.yml
2. k8s_deploy.yml
3. k8s_post_deploy.yml

- **k8s_prepare_host.yml**
  - Configures the network interfaces and services
  - Enables or disables system services
  - Install requires OS packages requirements
  - Installs Docker
  - Compiles and installs Linux base drivers for Intel Ethernet network connections

- **K8s_deploy.yml**
  - Installs Kubeadm, Kubectl, Kubelet, Kubernetes-cni
  - Builds SR-IOV, Multus, and DPDK (if applicable)
  - Initializes/configures Kubernetes master and minion nodes
- **k8s_post_deploy.yml**
  - Enables pod scheduling and privilege pods on Kubernetes master
  - Deploys Kubernetes dashboard
  - Deploys CMK (if applicable)
  - Checks pods all-namespaces

8.0 Post-deployment verification and troubleshooting

This section shows an example of how to check the post-deployment node status of master & minion:

```
# kubectl get nodes -o wide
NAME          STATUS    AGE       VERSION   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION
ab11-07-wcp   Ready     12m       v1.6.6    <none>        Ubuntu 16.04.3 LTS   4.10.0-30-generic
ab11-08-wcp   Ready     12m       v1.6.6    <none>        Ubuntu 16.04.3 LTS   4.10.0-30-generic
```

Example pod status of master & minion:

Note: Pod states should be in Running status (kube-dns container in ContainerCreating state, troubleshooting below)

```
# kubectl get pods --all-namespaces -o wide
NAMESPACE     NAME                                    READY     STATUS
RESTARTS   AGE       IP   NODE
kube-system   etcd-ab11-07-wcp                        1/1       Running
0          3m        192.168.197.85   ab11-07-wcp
kube-system   kube-apiserver-ab11-07-wcp              1/1       Running
0          3m        192.168.197.85   ab11-07-wcp
kube-system   kube-controller-manager-ab11-07-wcp     1/1       Running
0          4m        192.168.197.85   ab11-07-wcp
kube-system   kube-dns-692378583-dhvqs                0/3       ContainerCreating
0          4m        <none>           ab11-07-wcp
kube-system   kube-multus-ds-3gv4v                    2/2       Running
1          4m        192.168.197.85   ab11-07-wcp
kube-system   kube-multus-ds-76h1c                    2/2       Running
0          3m        192.168.197.86   ab11-08-wcp
kube-system   kube-proxy-bkn4l                        1/1       Running
0          3m        192.168.197.86   ab11-08-wcp
kube-system   kube-proxy-ffwql                        1/1       Running
0          4m        192.168.197.85   ab11-07-wcp
kube-system   kube-scheduler-ab11-07-wcp              1/1       Running
0          3m        192.168.197.85   ab11-07-wcp
kube-system   kubernetes-dashboard-1039728896-ttpgt   1/1       Running
0          4m        192.168.197.85   ab11-07-wcp
```

On the next page is an example for CMK deployments, verifying CPU initialization. In the example, CMK is enabled with HT, therefore both physical and associated logical processors are isolated. More information can be found in section 4.0 of the application note titled "Enhanced Platform Awareness in Kubernetes Application Note." This document can be found at the Intel container experience kit website at: https://networkbuilders.intel.com/network-technologies/container-experience-kits.

From deploy.yml

-num_dp_cores: 4

-num_cp_cores: 1

-All of the non-isolated cpu's will be added into infra pool

```
# kubectl logs cmk-init-install-discover-pod-ab11-07-wcp  init
INFO:root:Writing config to /etc/cmk.
INFO:root:Requested data plane cores = 4.
INFO:root:Requested control plane cores = 1.
INFO:root:Isolated logical cores: 2,3,4,5,6,38,39,40,41,42
INFO:root:Isolated physical cores: 2,3,4,5,6
INFO:root:Adding dataplane pool.
INFO:root:Adding cpu list 2,38 from socket 0 to dataplane pool.
INFO:root:Adding cpu list 3,39 from socket 0 to dataplane pool.
INFO:root:Adding cpu list 4,40 from socket 0 to dataplane pool.
INFO:root:Adding cpu list 5,41 from socket 0 to dataplane pool.
INFO:root:Adding controlplane pool.
INFO:root:Adding cpu list 6,42 to controlplane pool.
INFO:root:Adding infra pool.
INFO:root:Adding cpu list 0,36,1,37,7,43,8,44,9,45,10,46,11,47,12,48,13,49,14,50,15,51,16,52,17,53 to
infra pool.
INFO:root:Adding cpu list 18,54,19,55,20,56,21,57,22,58,23,59,24,60,25,61,26,62,27,63,28,64,29,65,30,66,3
1,67,32,68,33,69,34,70,35,71 to infra pool.
```

Note: Cores: 0,36,1,37 are not used as kernel can still schedule such cores

Determining physical and logical cores of HT siblings (if applicable)

```
cat /sys/devices/system/node/node0/cpulist
0-17,36-53
cat /sys/devices/system/node/node1/cpulist
18-35,54-71
```

or with lscpu –p

```
lscpu -p
# The following is the parsable format, which can be fed to other
# programs. Each different item in every column has an unique ID
# starting from zero.
# CPU,Core,Socket,Node,,L1d,L1i,L2,L3
0,0,0,0,,0,0,0
1,1,0,0,,1,1,0
2,2,0,0,,2,2,0
3,3,0,0,,3,3,0
4,4,0,0,,4,4,0
…
36,0,0,0,,0,0,0
37,1,0,0,,1,1,0
38,2,0,0,,2,2,0
39,3,0,0,,3,3,0
40,4,0,0,,4,4,0
…
```

Example of NFD labels displayed per node. Checking pod state and SR-IOV, RDTMON labels.

More information can be found in section 3.0 of the application note titled "Enhanced Platform Awareness in Kubernetes Application Note." This document can be found at the Intel container experience kit website at: https://networkbuilders.intel.com/network-technologies/container-experience-kits.

```
kubectl get pods --show-all
NAME                            READY     STATUS      RESTARTS   AGE
node-feature-discovery-3w4vc    0/1       Completed   0          6h
node-feature-discovery-lr3tr    0/1       Completed   0          6h
```

```
kubectl get nodes -o json | jq .items[].metadata.labels
{
...
  "node.alpha.kubernetes-incubator.io/nfd-network-sriov": "true",
  "node.alpha.kubernetes-incubator.io/nfd-network-sriov-configured": "true",
  "node.alpha.kubernetes-incubator.io/nfd-rdt-RDTMON": "true",
...
}
{
...
  "node.alpha.kubernetes-incubator.io/nfd-network-sriov": "true",
  "node.alpha.kubernetes-incubator.io/nfd-network-sriov-configured": "true",
  "node.alpha.kubernetes-incubator.io/nfd-rdt-RDTMON": "true",
...
}
```

Example of Kubernetes native huge page support (Kubernetes 1.8.4+ only).

1GB huge pages will show as hugepages-1Gi

2MB huge pages will show as hugepages-2Mi

For more information, reference Enhanced Platform Awareness in Kubernetes, section 5.

```
kubectl get nodes -o json | jq .items[].status.allocatable
{
…
  "hugepages-1Gi": "16Gi",
…
}
{
…
  "hugepages-1Gi": "16Gi",
…
}
```

Troubleshooting

During deployment, kube-dns-xxxxx may fail to fully initialize for various reasons (i.e.: failed to download the DNS image) and stays in a CreatingContainer state (shown in example get pods in the previous instructions). Simply delete the pod kube-dns-xxxxx as Kubernetes will auto-recreate pods that are part of namespace: kube-system.

```
# kubectl delete -n kube-system pods kube-dns-692378583-dhvqs

# kubectl get pods --all-namespaces -o wide
NAMESPACE     NAME                          READY     STATUS    RESTARTS   AGE
kube-system   kube-dns-692378583-qr3jb      3/3       Running   0          30s
```

9.0 Conclusion

Container environments have significant promise for the next-generation data center and communications service provider computing environments. Intel has been working with partners and with open source communities to add new techniques and address key barriers to adoption for commercial containers by uncovering the power of Intel Architecture-based servers to improve configuration, manageability, performance, service-assurance, and resilience of container deployments.

This document is to provide developers with the hands-on instructions in how to utilize the Ansible scripts provided by Intel to deploy these technologies and gain the knowledge and experience to integrate them into solutions under development.

For more information on what Intel is doing with containers, go to https://networkbuilders.intel.com/network-technologies/intel-container-experience-kits.

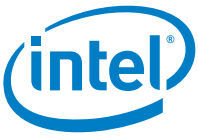
For more information about the open-source deployment tools and software ingredients used in this user guide, visit:

Open Source Deployment Tools	URL
Ubuntu	https://www.ubuntu.com/
Ansible	https://www.ansible.com/
Docker	https://www.docker.com/what-docker
Kubernetes	https://kubernetes.io/
Kubeadm	https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/
MULTUS CNI plugin	https://github.com/Intel-Corp/multus-cni
SR-IOV CNI plugin	https://github.com/Intel-Corp/sriov-cni
CMK	https://github.com/Intel-Corp/CPU-Manager-for-Kubernetes

Appendix A: Acronyms and abbreviations

Table A-1 Acronyms and abbreviations

Abbreviation	Description
BIOS	Basic Input/Output System
CMK	CPU Manager for Kubernetes
CNI	Container Networking Interface
DHCP	Dynamic Host Configuration Protocol
DPDK	Data Plane Development Kit
IA	Intel Architecture
Intel HT Technology	Intel Hyper-Threading Technology
Intel VT-d	Intel VT for Directed I/O
Intel VT-x	Intel Virtualization Technology
K8s	Kubernetes
NFD	Node Feature Discovery
NFV	Network Functions Virtualization
OS	Operating System
RA	Reference Architecture
SA	Service Assurance
SDN	Software-Defined Networking
SHVS	Standard High-Volume Servers
SOCKS	Socket Secure
SR-IOV	Single Root Input/Output Virtualization
VLAN	Virtual LAN
VNF	Virtual Network Function
VXLAN	Virtual Extensible LAN



Legal Information

 Printed in USA Please Recycle 01/04/HM/DJA/PDF001 Jan 2018 pm SKU 336985-001 US