Red Hat OpenStack Services on OpenShift 18.0

Deploying a network functions virtualization environment

Planning, installing, and configuring network functions virtualization (NFV) in Red Hat OpenStack Services on OpenShift

Last Updated: 2024-09-12

Red Hat OpenStack Services on OpenShift 18.0 Deploying a network functions virtualization environment

Planning, installing, and configuring network functions virtualization (NFV) in Red Hat OpenStack Services on OpenShift

OpenStack Team
rhos-docs@redhat.com

Abstract

Plan, install, and configure single root input/output virtualization (SR-IOV) and Open vSwitch Data Plane Development Kit (OVS-DPDK) for network functions virtualization infrastructure (NFVi) in a Red Hat OpenStack Services on OpenShift environment.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Providing documentation feedback in Jira

Use the Create Issue form to provide feedback on the documentation for Red Hat OpenStack Services on OpenShift (RHOSO) or earlier releases of Red Hat OpenStack Platform (RHOSP). When you create an issue for RHOSO or RHOSP documents, the issue is recorded in the RHOSO Jira project, where you can track the progress of your feedback.

To complete the Create Issue form, ensure that you are logged in to Jira. If you do not have a Red Hat Jira account, you can create an account at https://issues.redhat.com.

1. Click the following link to open a **Create Issue** page: Create Issue
2. Complete the **Summary** and **Description** fields. In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue. Do not modify any other fields in the form.
3. Click **Create**.

CHAPTER 1. UNDERSTANDING RED HAT NETWORK FUNCTIONS VIRTUALIZATION (NFV)

Network functions virtualization (NFV) is a software-based solution that helps communication service providers (CSPs) to move beyond the traditional, proprietary hardware to achieve greater efficiency and agility and to reduce operational costs.

Using NFV in a Red Hat OpenStack Services on OpenShift (RHOSO) environment allows for IT and network convergence by providing a virtualized infrastructure that uses the standard virtualization technologies to virtualize network functions (VNFs) that run on hardware devices such as switches, routers, and storage.

1.1. ADVANTAGES OF NFV

The main advantages of implementing network functions virtualization (NFV) in a Red Hat OpenStack Services on OpenShift (RHOSO) environment are:

- Accelerates the time-to-market by enabling you to quickly deploy and scale new networking services to address changing demands.
- Supports innovation by enabling service developers to self-manage their resources and prototype using the same platform that will be used in production.
- Addresses customer demands in hours or minutes instead of weeks or days, without sacrificing security or performance.
- Reduces capital expenditure because it uses commodity-off-the-shelf hardware instead of expensive tailor-made equipment.
- Uses streamlined operations and automation that optimize day-to-day tasks to improve employee productivity and reduce operational costs.

1.2. SUPPORTED CONFIGURATIONS FOR NFV DEPLOYMENTS

Red Hat supports network functions virtualization (NFV) on Red Hat OpenStack Services on OpenShift (RHOSO) environments using Data Plane Development Kit (DPDK) and Single Root I/O Virtualization (SR-IOV).

Other configurations include:

- Open vSwitch (OVS) with LACP
- Hyper-converged infrastructure (HCI)

IMPORTANT

Red Hat does not support the use of OVS-DPDK for non-NFV workloads. If you need OVS-DPDK functionality for non-NFV workloads, contact your Technical Account Manager (TAM) or open a customer service request case to discuss a Support Exception and other options. To open a customer service request case, go to Create a case and choose **Account > Customer Service Request**

Additional resources

- *Deploying a hyper-converged infrastructure environment*

1.3. NFV DATA PLANE CONNECTIVITY

With the introduction of network functions virtualization (NFV), more networking vendors are starting to implement their traditional devices as VNFs. While the majority of networking vendors are considering virtual machines, some are also investigating a container-based approach as a design choice. A Red Hat OpenStack Services on OpenShift (RHOSO) environment should be rich and flexible because of two primary reasons:

- **Application readiness** - Network vendors are currently in the process of transforming their devices into VNFs. Different VNFs in the market have different maturity levels; common barriers to this readiness include enabling RESTful interfaces in their APIs, evolving their data models to become stateless, and providing automated management operations. OpenStack should provide a common platform for all.

- **Broad use-cases** - NFV includes a broad range of applications that serve different use-cases. For example, Virtual Customer Premise Equipment (vCPE) aims at providing a number of network functions such as routing, firewall, virtual private network (VPN), and network address translation (NAT) at customer premises. Virtual Evolved Packet Core (vEPC), is a cloud architecture that provides a cost-effective platform for the core components of Long-Term Evolution (LTE) network, allowing dynamic provisioning of gateways and mobile endpoints to sustain the increased volumes of data traffic from smartphones and other devices.
  These use cases are implemented using different network applications and protocols, and require different connectivity, isolation, and performance characteristics from the infrastructure. It is also common to separate between control plane interfaces and protocols and the actual forwarding plane. OpenStack must be flexible enough to offer different datapath connectivity options.

In principle, there are two common approaches for providing data plane connectivity to virtual machines:

- **Direct hardware access** bypasses the linux kernel and provides secure direct memory access (DMA) to the physical NIC using technologies such as PCI Passthrough or single root I/O virtualization (SR-IOV) for both Virtual Function (VF) and Physical Function (PF) pass-through.

- **Using a virtual switch (vswitch)**, implemented as a software service of the hypervisor. Virtual machines are connected to the vSwitch using virtual interfaces (vNICs), and the vSwitch is capable of forwarding traffic between virtual machines, as well as between virtual machines and the physical network.

Some of the fast data path options are as follows:

- **Single Root I/O Virtualization (SR-IOV)** is a standard that makes a single PCI hardware device appear as multiple virtual PCI devices. It works by introducing Physical Functions (PFs), which are the fully featured PCIe functions that represent the physical hardware ports, and Virtual Functions (VFs), which are lightweight functions that are assigned to the virtual machines. To the VM, the VF resembles a regular NIC that communicates directly with the hardware. NICs support multiple VFs.

- **Open vSwitch (OVS)** is an open source software switch that is designed to be used as a virtual switch within a virtualized server environment. OVS supports the capabilities of a regular L2-L3 switch and also offers support to the SDN protocols such as OpenFlow to create user-defined overlay networks (for example, VXLAN). OVS uses Linux kernel networking to switch packets between virtual machines and across hosts using physical NIC. OVS now supports connection tracking (Conntrack) with built-in firewall capability to avoid the overhead of Linux bridges that use iptables/ebtables. Open vSwitch for Red Hat OpenStack Platform environments offers default OpenStack Networking (neutron) integration with OVS.

- **Data Plane Development Kit (DPDK)** consists of a set of libraries and poll mode drivers (PMD) for fast packet processing. It is designed to run mostly in the user-space, enabling applications to perform their own packet processing directly from or to the NIC. DPDK reduces latency and allows more packets to be processed. DPDK Poll Mode Drivers (PMDs) run in busy loop, constantly scanning the NIC ports on host and vNIC ports in guest for arrival of packets.

- **DPDK accelerated Open vSwitch (OVS-DPDK)** is Open vSwitch bundled with DPDK for a high performance user-space solution with Linux kernel bypass and direct memory access (DMA) to physical NICs. The idea is to replace the standard OVS kernel data path with a DPDK-based data path, creating a user-space vSwitch on the host that uses DPDK internally for its packet forwarding. The advantage of this architecture is that it is mostly transparent to users. The interfaces it exposes, such as OpenFlow, OVSDB, the command line, remain mostly the same.

1.4. ETSI NFV ARCHITECTURE

The European Telecommunications Standards Institute (ETSI) is an independent standardization group that develops standards for information and communications technologies (ICT) in Europe.

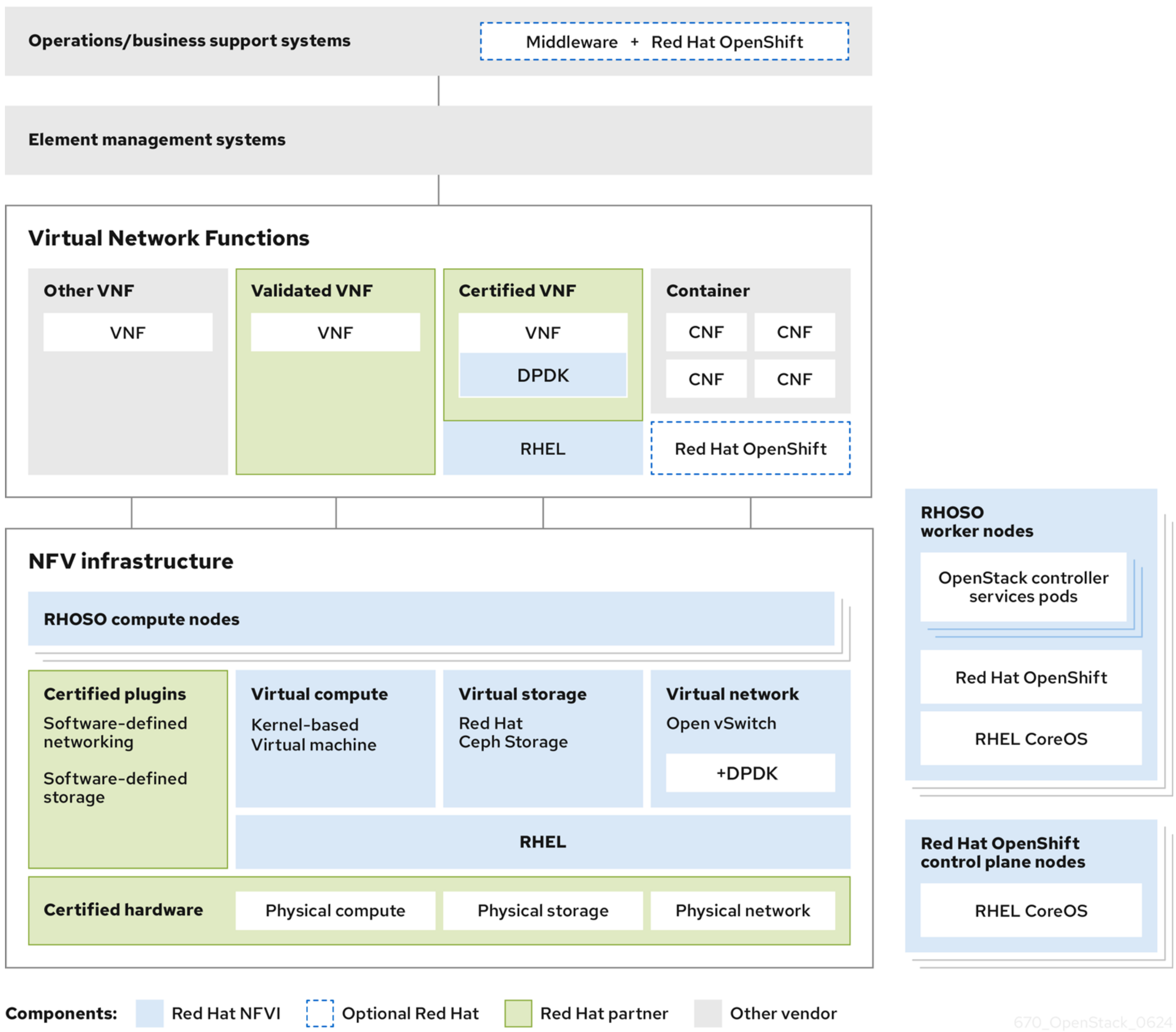
Network functions virtualization (NFV) focuses on addressing problems involved in using proprietary hardware devices. With NFV, the necessity to install network-specific equipment is reduced, depending upon the use case requirements and economic benefits. The ETSI Industry Specification Group for Network Functions Virtualization (ETSI ISG NFV) sets the requirements, reference architecture, and the infrastructure specifications necessary to ensure virtualized functions are supported.

Red Hat is offering an open-source based cloud-optimized solution to help the Communication Service Providers (CSP) to achieve IT and network convergence. Red Hat adds NFV features such as single root I/O virtualization (SR-IOV) and Open vSwitch with Data Plane Development Kit (OVS-DPDK) to Red Hat OpenStack Services on OpenShift (RHOSO) environments.

1.5. NFV ETSI ARCHITECTURE AND COMPONENTS

In general, a network functions virtualization (NFV) on Red Hat OpenStack Services on OpenShift (RHOSO) environments has the following components:

Figure 1.1. NFV ETSI architecture and components



- **Virtualized Network Functions (VNFs)** - the software implementation of routers, firewalls, load balancers, broadband gateways, mobile packet processors, servicing nodes, signalling, location services, and other network functions.

- **NFV Infrastructure (NFVi)** - the physical resources (compute, storage, network) and the virtualization layer that make up the infrastructure. The network includes the datapath for forwarding packets between virtual machines and across hosts. This allows you to install VNFs without being concerned about the details of the underlying hardware. NFVi forms the foundation of the NFV stack. NFVi supports multi-tenancy and is managed by the Virtual Infrastructure Manager (VIM). Enhanced Platform Awareness (EPA) improves the virtual machine packet forwarding performance (throughput, latency, jitter) by exposing low-level CPU and NIC acceleration components to the VNF.

- **NFV Management and Orchestration (MANO)** - the management and orchestration layer focuses on all the service management tasks required throughout the life cycle of the VNF. The main goals of MANO is to allow service definition, automation, error-correlation, monitoring, and life-cycle management of the network functions offered by the operator to its customers, decoupled from the physical infrastructure. This decoupling requires additional layers of management, provided by the Virtual Network Function Manager (VNFM). VNFM manages the life cycle of the virtual machines and VNFs by either interacting directly with them or through the Element Management System (EMS) provided by the VNF vendor. The other important component defined by MANO is the Orchestrator, also known as NFVO. NFVO interfaces with various databases and systems including Operations/Business Support Systems (OSS/BSS) on the top and the VNFM on the bottom. If the NFVO wants to create a new service for a customer, it asks the VNFM to trigger the instantiation of a VNF, which may result in multiple virtual machines.

- **Operations and Business Support Systems (OSS/BSS)**- provides the essential business function applications, for example, operations support and billing. The OSS/BSS needs to be adapted to NFV, integrating with both legacy systems and the new MANO components. The BSS systems set policies based on service subscriptions and manage reporting and billing.
- **Systems Administration, Automation and Life-Cycle Management**- manages system administration, automation of the infrastructure components and life cycle of the NFVi platform.

1.6. RED HAT NFV COMPONENTS

Red Hat’s solution for network functions virtualization (NFV) includes a range of products that can act as the different components of the NFV framework in the ETSI model. The following products from the Red Hat portfolio integrate into an NFV solution:

- Red Hat OpenStack Services on OpenShift (RHOSO) - Supports IT and NFV workloads. The Enhanced Platform Awareness (EPA) features deliver deterministic performance improvements through CPU pinning, huge pages, Non-Uniform Memory Access (NUMA) affinity, and network adaptors (NICs) that support SR-IOV and OVS-DPDK.
- Red Hat Enterprise Linux and Red Hat Enterprise Linux Atomic Host - Create virtual machines and containers as VNFs.
- Red Hat Ceph Storage - Provides the unified elastic and high-performance storage layer for all the needs of the service provider workloads.
- Red Hat JBoss Middleware and OpenShift Enterprise by Red Hat - Optionally provide the ability to modernize the OSS/BSS components.
- Red Hat CloudForms - Provides a VNF manager and presents data from multiple sources, such as the VIM and the NFVi in a unified display.
- Red Hat Satellite and Ansible by Red Hat - Optionally provide enhanced systems administration, automation and life-cycle management.

CHAPTER 2. NFV PERFORMANCE CONSIDERATIONS

For a network functions virtualization (NFV) solution to be useful, its virtualized functions must meet or exceed the performance of physical implementations. Red Hat's virtualization technologies are based on the high-performance Kernel-based Virtual Machine (KVM) hypervisor, common in OpenStack and cloud deployments.

In Red Hat OpenStack Services on OpenShift (RHOSO), you configure the Compute nodes to enforce resource partitioning and fine tuning to achieve line rate performance for the guest virtual network functions (VNFs). The key performance factors in the NFV use case are throughput, latency, and jitter.

You can enable high-performance packet switching between physical NICs and virtual machines using data plane development kit (DPDK) accelerated virtual machines. Open vSwitch (OVS) embeds support for Data Plane Development Kit (DPDK) and includes support for vhost-user multiqueue, allowing scalable performance. OVS-DPDK provides line-rate performance for guest VNFs.

Single root I/O virtualization (SR-IOV) networking provides enhanced performance, including improved throughput for specific networks and virtual machines.

Other important features for performance tuning include huge pages, NUMA alignment, host isolation, and CPU pinning. VNF flavors require huge pages and emulator thread isolation for better performance. Host isolation and CPU pinning improve NFV performance and prevent spurious packet loss.

2.1. CPUS AND NUMA NODES

Previously, all memory on x86 systems was equally accessible to all CPUs in the system. This resulted in memory access times that were the same regardless of which CPU in the system was performing the operation and was referred to as Uniform Memory Access (UMA).

In Non-Uniform Memory Access (NUMA), system memory is divided into zones called nodes, which are allocated to particular CPUs or sockets. Access to memory that is local to a CPU is faster than memory connected to remote CPUs on that system. Normally, each socket on a NUMA system has a local memory node whose contents can be accessed faster than the memory in the node local to another CPU or the memory on a bus shared by all CPUs.

Similarly, physical NICs are placed in PCI slots on the Compute node hardware. These slots connect to specific CPU sockets that are associated to a particular NUMA node. For optimum performance, connect your datapath NICs to the same NUMA nodes in your CPU configuration (SR-IOV or OVS-DPDK).

The performance impact of NUMA misses are significant, generally starting at a 10% performance hit or higher. Each CPU socket can have multiple CPU cores which are treated as individual CPUs for virtualization purposes.

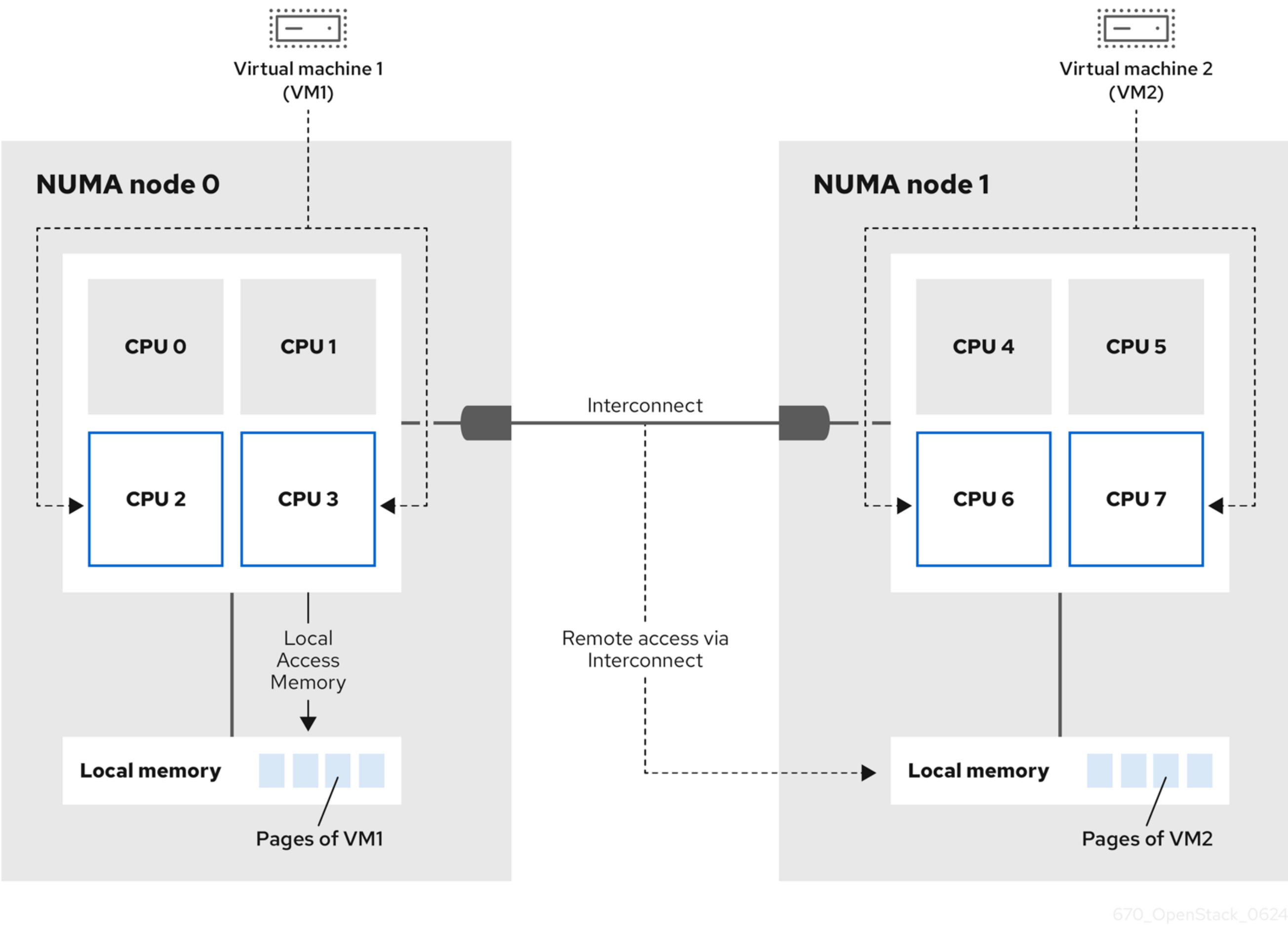
TIP

For more information about NUMA, see What is NUMA and how does it work on Linux?

2.1.1. NUMA node example

The following diagram provides an example of a two-node NUMA system and the way the CPU cores and memory pages are made available:

Figure 2.1. Example: two-node NUMA system



NOTE

Remote memory available via Interconnect is accessed **only** if VM1 from NUMA node 0 has a CPU core in NUMA node 1. In this case, the memory of NUMA node 1 acts as local for the third CPU core of VM1 (for example, if VM1 is allocated with CPU 4 in the diagram above), but at the same time, it acts as remote memory for the other CPU cores of the same VM.

2.1.2. NUMA aware instances

You can configure an OpenStack environment to use NUMA topology awareness on systems with a NUMA architecture. When running a guest operating system in a virtual machine (VM) there are two NUMA topologies involved:

- NUMA topology of the physical hardware of the host
- NUMA topology of the virtual hardware exposed to the guest operating system

You can optimize the performance of guest operating systems by aligning the virtual hardware with the physical hardware NUMA topology.

2.2. CPU PINNING

CPU pinning is the ability to run a specific virtual machine's virtual CPU on a specific physical CPU, in a given host. vCPU pinning provides similar advantages to task pinning on bare-metal systems. Since virtual machines run as user space tasks on the host operating system, pinning increases cache efficiency.

Additional resources

- XREF TO "Configuring CPU pinning on Compute nodes" in the *Configuring the Compute service for instance creation* guide.

2.3. HUGE PAGES

Physical memory is segmented into contiguous regions called pages. For efficiency, the system retrieves memory by accessing entire pages instead of individual bytes of memory. To perform this translation, the system looks in the Translation Lookaside Buffers (TLB) that contain the physical to virtual address mappings for the most recently or frequently used pages. When the system cannot find a mapping in the TLB, the processor must iterate through all of the page tables to determine the address mappings. Optimize the TLB to minimize the performance penalty that occurs during these TLB misses.

The typical page size in an x86 system is 4KB, with other larger page sizes available. Larger page sizes mean that there are fewer pages overall, and therefore increases the amount of system memory that can have its virtual to physical address translation stored in the TLB. Consequently, this reduces TLB misses, which increases performance. With larger page sizes, there is an increased potential for memory to be under-utilized as processes must allocate in pages, but not all of the memory is likely required. As a result, choosing a page size is a compromise between providing faster access times with larger pages, and ensuring maximum memory utilization with smaller pages.

CHAPTER 3. REQUIREMENTS FOR NFV

This section describes the requirements for network functions virtualization (NFV) in a Red Hat OpenStack Services on OpenShift (RHOSO) environment.

Red Hat certifies hardware for use with RHOSO. For more information, see Certified hardware.

3.1. TESTED NICS FOR NFV

For a list of tested NICs for NFV, see the Red Hat Knowledgebase solution Network Adapter Fast Datapath Feature Support Matrix.

Use the default driver for the supported NIC, unless you are configuring NVIDIA (Mellanox) network interfaces. For NVIDIA network interfaces, you must specify the kernel driver during configuration.

Example

In this example, an OVS-DPDK port is being configured. Because the NIC being used is an NVIDIA ConnectX-5, the driver must be specified:

```
members
 - type: ovs_dpdk_port
   name: dpdk0
   driver: mlx5_core
   members:
   - type: interface
     name: enp3s0f0
```

3.2. DISCOVERING YOUR NUMA NODE TOPOLOGY

For network functions virtualization (NFV) on Red Hat OpenStack Services on OpenShift (RHOSO) environments, you must understand the NUMA topology of your Compute node to partition the CPU and memory resources for optimum performance. To determine the NUMA information, perform one of the following tasks:

- Enable hardware introspection to retrieve this information from bare-metal nodes.
- Log on to each bare-metal node to manually collect the information.

Additional resources

- Bare metal configuration in the Red Hat OpenShift Container Platform (RHOCP) *Postinstallation configuration* guide.

3.3. NFV BIOS SETTINGS

The following table describes the required BIOS settings for network functions virtualization (NFV) on Red Hat OpenStack Services on OpenShift (RHOSO) environments:

> **NOTE**
>
> You must enable SR-IOV global and NIC settings in the BIOS, or your RHOSO deployment with SR-IOV Compute nodes will fail.

Table 3.1. BIOS Settings

Parameter	Setting
C3 Power State	Disabled.
C6 Power State	Disabled.
MLC Streamer	Enabled.
MLC Spatial Prefetcher	Enabled.
DCU Data Prefetcher	Enabled.
DCA	Enabled.
CPU Power and Performance	Performance.
Memory RAS and Performance Config → NUMA Optimized	Enabled.
Turbo Boost	Disabled in NFV deployments that require deterministic performance. Enabled in all other scenarios.
VT-d	Enabled for Intel cards if VFIO functionality is needed.
NUMA memory interleave	Disabled.

On processors that use the **intel_idle** driver, Red Hat Enterprise Linux can ignore BIOS settings and re-enable the processor C-state.

You can disable **intel_idle** and instead use the **acpi_idle** driver by specifying the key-value pair **intel_idle.max_cstate=0** on the kernel boot command line.

Confirm that the processor is using the **acpi_idle** driver by checking the contents of **current_driver**:

```
$ cat /sys/devices/system/cpu/cpuidle/current_driver
```

Sample output

```
acpi_idle
```

> **NOTE**
>
> You will experience some latency after changing drivers, because it takes time for the Tuned daemon to start. However, after Tuned loads, the processor does not use the deeper C-state.

3.4. SUPPORTED DRIVERS FOR NFV

For a complete list of supported drivers for network functions virtualization (NFV) on Red Hat OpenStack Services on OpenShift (RHOSO) environments, see Component, Plug-In, and Driver Support in Red Hat OpenStack Platform .

For a list of NICs tested for NFV on RHOSO environments, see Tested NICs for NFV .

CHAPTER 4. PLANNING AN SR-IOV DEPLOYMENT

To optimize single root I/O virtualization (SR-IOV) deployments for NFV in Red Hat OpenStack Services on OpenShift (RHOSO) environments, it is important to understand how SR-IOV uses the Compute node hardware (CPU, NUMA nodes, memory, NICs). This understanding will help you to determine the values required for the parameters used in your SR-IOV configuration.

To evaluate your hardware impact on the SR-IOV parameters, see Discovering your NUMA node topology.

4.1. NIC PARTITIONING FOR AN SR-IOV DEPLOYMENT

You can reduce the number of NICs that you need for each host by configuring single root I/O virtualization (SR-IOV) virtual functions (VFs) for Red Hat OpenStack Services on OpenShift (RHOSO) management networks and provider networks. When you partition a single, high-speed NIC into multiple VFs, you can use the NIC for both control and data plane traffic. This feature has been validated on Intel Fortville NICs, and Mellanox CX-5 NICs.

To partition your NICs, you must adhere to the following requirements:

- The NICs, their applications, the VF guest, and OVS reside on the same NUMA Compute node. Doing so helps to prevent performance degradation from cross-NUMA operations.
- Ensure that the NIC firmware updated.
  Yum or **dnf** updates might not complete the firmware update. For more information, see your vendor documentation.

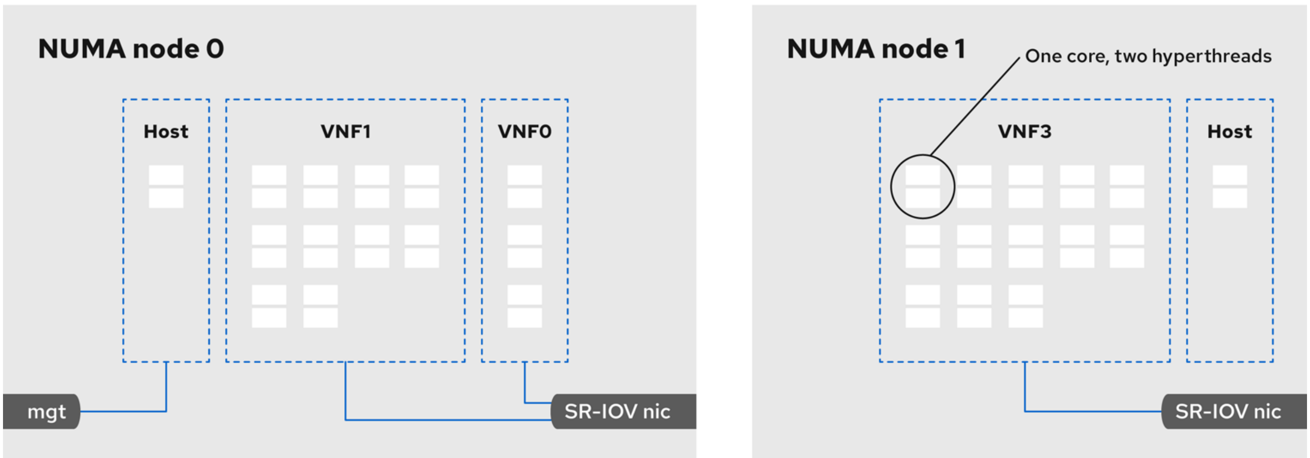
Additional resources

- Example template - partitioned NIC

4.2. HARDWARE PARTITIONING FOR AN SR-IOV DEPLOYMENT

To achieve high performance with SR-IOV, partition the resources between the host and the guest.

Figure 4.1. NUMA node topology

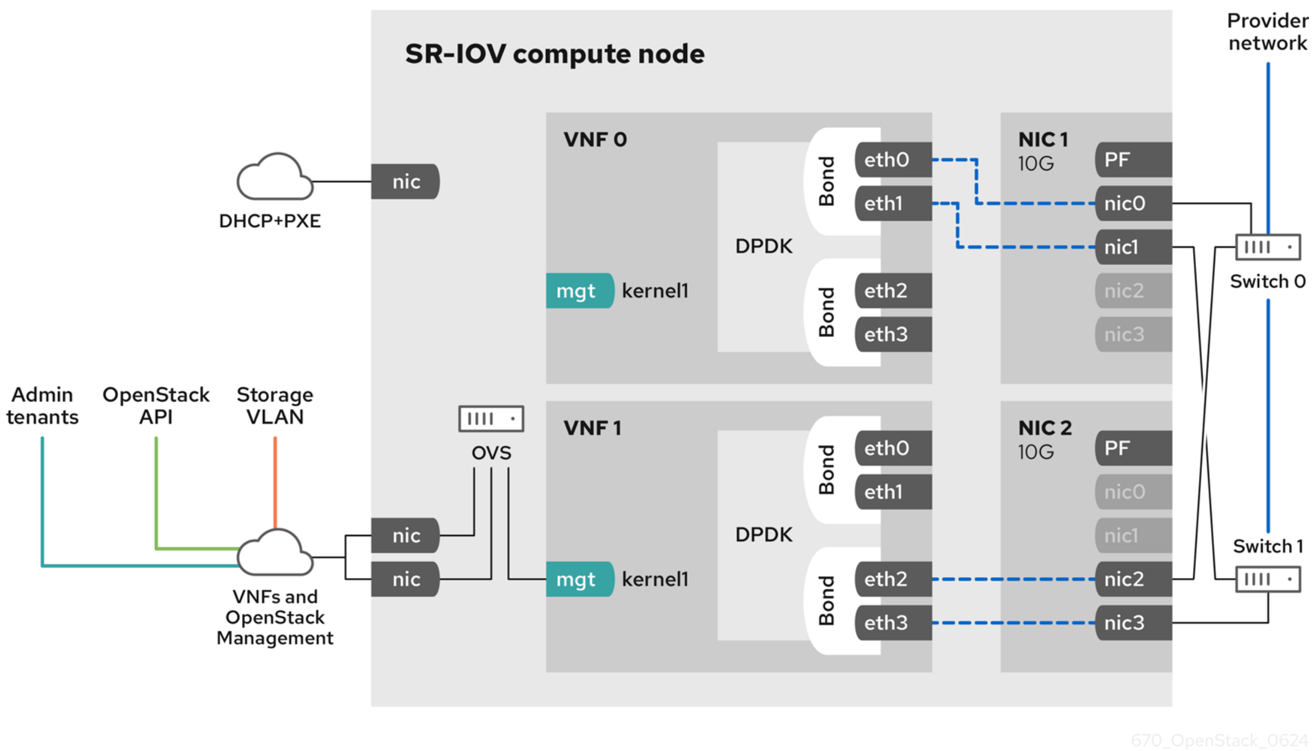


A typical topology includes 14 cores per NUMA node on dual socket Compute nodes. Both hyperthreading (HT) and non-HT cores are supported. Each core has two sibling threads. One core is dedicated to the host on each NUMA node. The virtual network function (VNF) handles the SR-IOV interface bonding. All the interrupt requests (IRQs) are routed on the host cores. The VNF cores are dedicated to the VNFs. They provide isolation from other VNFs and isolation from the host. Each VNF must use resources on a single NUMA node. The SR-IOV NICs used by the VNF must also be associated with that same NUMA node. This topology does not have a virtualization overhead. The host, OpenStack Networking (neutron), and Compute (nova) configuration parameters are exposed in a single file for ease, consistency, and to avoid incoherence that is fatal to proper isolation, causing preemption, and packet loss. The host and virtual machine isolation depend on a **tuned** profile, which defines the boot parameters and any Red Hat OpenStack Platform modifications based on the list of isolated CPUs.

4.3. TOPOLOGY OF AN NFV SR-IOV DEPLOYMENT

The following image has two VNFs each with the management interface represented by **mgt** and the data plane interfaces. The management interface manages the **ssh** access, and so on. The data plane interfaces bond the VNFs to DPDK to ensure high availability, as VNFs bond the data plane interfaces using the DPDK library. The image also has two provider networks for redundancy. The Compute node has two regular NICs bonded together and shared between the VNF management and the Red Hat OpenStack Platform API management.

Figure 4.2. NFV SR-IOV topology

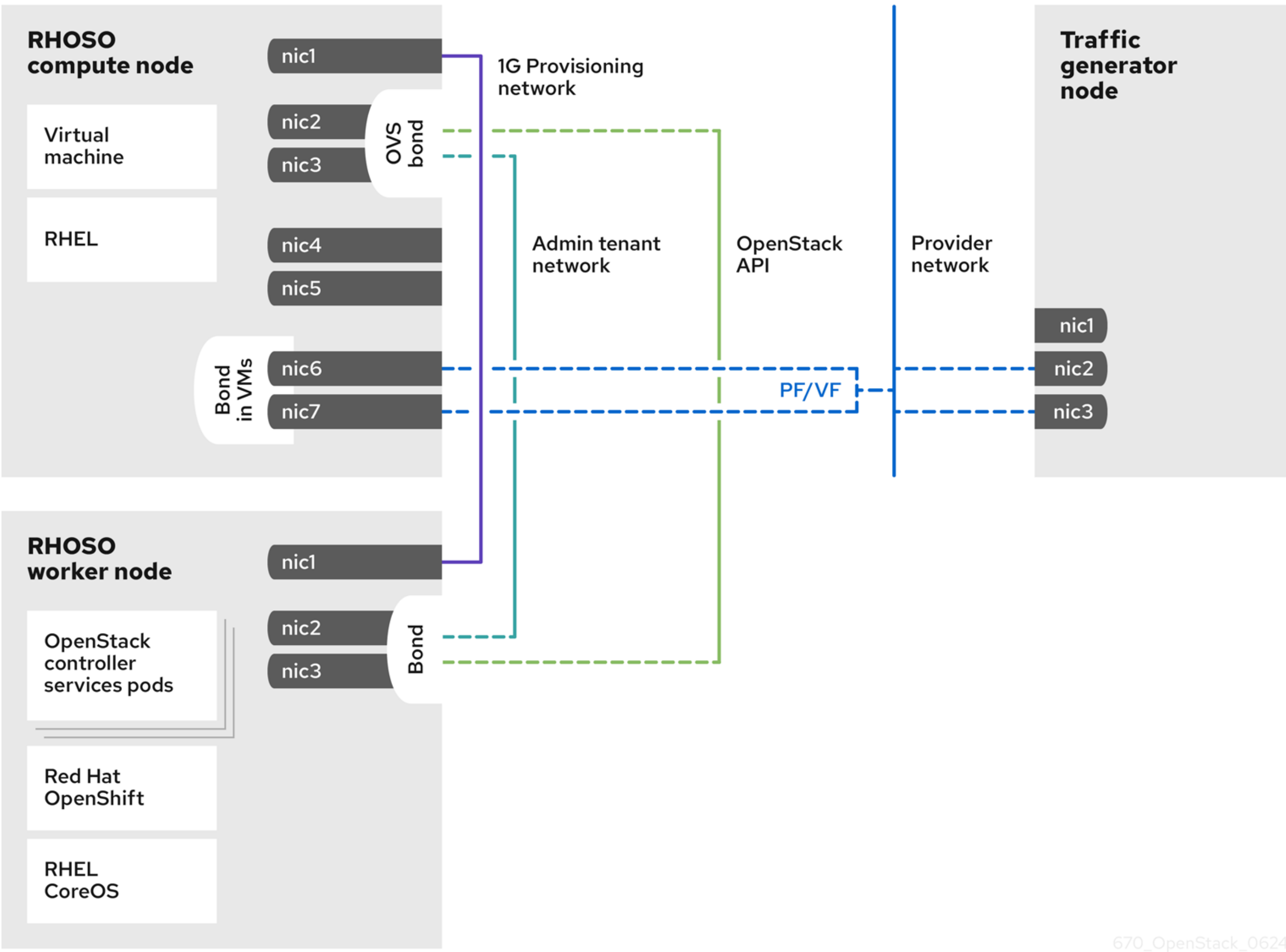


The image shows a VNF that uses DPDK at an application level, and has access to SR-IOV virtual functions (VFs) and physical functions (PFs), for better availability or performance, depending on the fabric configuration. DPDK improves performance, while the VF/PF DPDK bonds provide support for failover, and high availability. The VNF vendor must ensure that the DPDK poll mode driver (PMD) supports the SR-IOV card that is being exposed as a VF/PF. The management network uses OVS, therefore the VNF sees a mgmt network device using the standard virtIO drivers. You can use that device to initially connect to the VNF, and ensure that the DPDK application bonds the two VF/PFs.

4.4. TOPOLOGY FOR NFV SR-IOV WITHOUT HCI

Observe the topology for SR-IOV without hyper-converged infrastructure (HCI) for NFV in the image below. It consists of compute and controller nodes with 1 Gbps NICs, and the RHOSO worker node.

Figure 4.3. NFV SR-IOV topology without HCI



CHAPTER 5. PLANNING AN OVS-DPDK DEPLOYMENT

To optimize your Open vSwitch with Data Plane Development Kit (OVS-DPDK) deployment for NFV in Red Hat OpenStack Services on OpenShift (RHOSO) environments, you should understand how OVS-DPDK uses the Compute node hardware (CPU, NUMA nodes, memory, NICs). This understanding will help you to determine the values required for the parameters used in your OVS-DPDK configuration.

IMPORTANT

When using OVS-DPDK and the OVS native firewall (a stateful firewall based on conntrack), you can track only packets that use ICMPv4, ICMPv6, TCP, and UDP protocols. OVS marks all other types of network traffic as invalid.

IMPORTANT

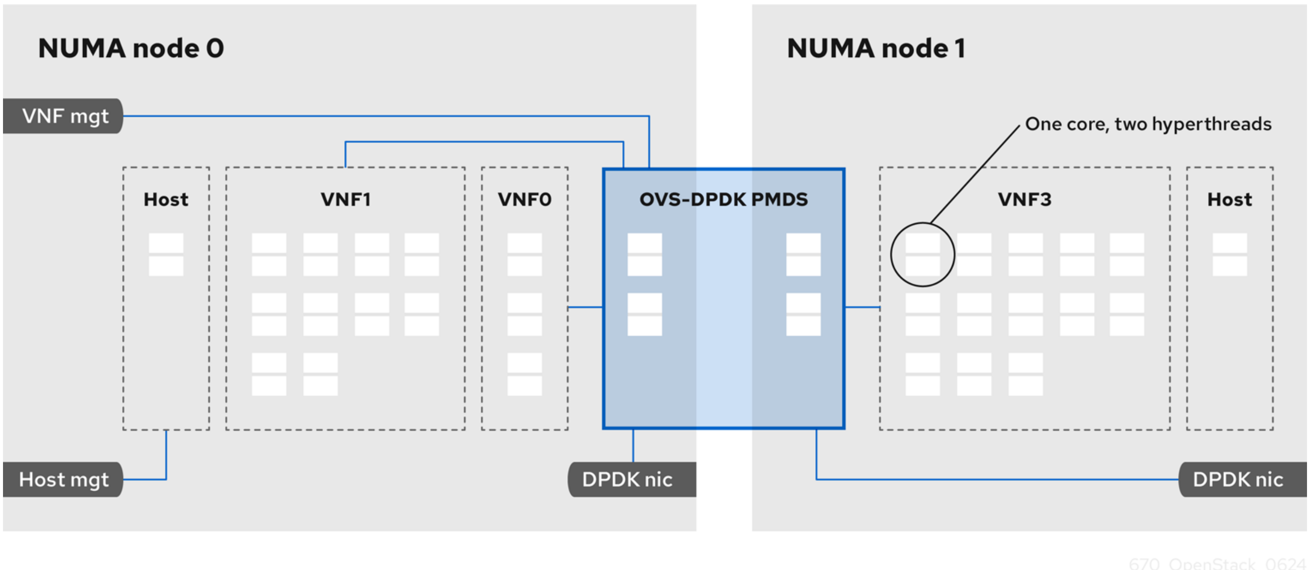
Red Hat does not support the use of OVS-DPDK for non-NFV workloads. If you need OVS-DPDK functionality for non-NFV workloads, contact your Technical Account Manager (TAM) or open a customer service request case to discuss a Support Exception and other options. To open a customer service request case, go to Create a case and choose **Account > Customer Service Request**

5.1. OVS-DPDK WITH CPU PARTITIONING AND NUMA TOPOLOGY

OVS-DPDK partitions the hardware resources for host, guests, and itself. The OVS-DPDK Poll Mode Drivers (PMDs) run DPDK active loops, which require dedicated CPU cores. Therefore you must allocate some CPUs, and huge pages, to OVS-DPDK.

A sample partitioning includes 16 cores per NUMA node on dual-socket Compute nodes. The traffic requires additional NICs because you cannot share NICs between the host and OVS-DPDK.

Figure 5.1. NUMA topology: OVS-DPDK with CPU partitioning



NOTE

You must reserve DPDK PMD threads on both NUMA nodes, even if a NUMA node does not have an associated DPDK NIC.

For optimum OVS-DPDK performance, reserve a block of memory local to the NUMA node. Choose NICs associated with the same NUMA node that you use for memory and CPU pinning. Ensure that both bonded interfaces are from NICs on the same NUMA node.

5.2. OVS-DPDK PARAMETERS

This section describes how OVS-DPDK uses parameters to configure the CPU and memory for optimum performance. Use this information to evaluate the hardware support on your Compute nodes and how to partition the hardware to optimize your OVS-DPDK deployment.

This section describes the data plane parameters used in custom resources (CRs) to configure an OVS-DPDK deployment.

NOTE

Always pair CPU sibling threads, or logical CPUs, together in the physical core when allocating CPU cores.

For details on how to determine the CPU and NUMA nodes on your Compute nodes, see Discovering your NUMA node topology. Use this information to map CPU and other parameters to support the host, guest instance, and OVS-DPDK process needs.

5.2.1. Data plane (EDPM) Ansible variables

The following variables are part of data plane (EDPM) Ansible roles:

edpm_ovs_dpdk

Enables you to add, modify, and delete OVS-DPDK configurations, by using values defined in the OVS-DPDK **edpm** Ansible variables.

edpm_ovs_dpdk_pmd_core_list

Provides the CPU cores that are used for the DPDK poll mode drivers (PMD). It is recommended that you choose CPU cores that are associated with the local NUMA nodes of the DPDK interfaces.

edpm_ovs_dpdk_lcore_list

List of CPU cores to be used for DPDK **lcore** threads.

edpm_tuned_profile

Name of the custom TuneD profile. The default value is **throughput-performance**.

edpm_tuned_isolated_cores

A set of CPU cores isolated from the host processes.

edpm_ovs_dpdk_socket_memory

Specifies the amount of memory in MB to pre-allocate from the hugepage pool, per NUMA node. **dpm_ovs_dpdk_socket_memory** is the **other_config:dpdk-socket-mem** value in OVS. Observe the following recommendations:

- Provide as a comma-separated list.
- For a NUMA node without a DPDK NIC, use the static recommendation of 1024MB (1GB).
- Calculate the edpm_ovs_dpdk_socket_memory value from the MTU value of each NIC on the NUMA node. The following equation approximates the value:

```
MEMORY_REQD_PER_MTU = (ROUNDUP_PER_MTU + 800) * (4096 * 64) Bytes
```

  - **800** is the overhead value.
  - **4096 * 64** is the number of packets in the mempool.
- Add the **MEMORY_REQD_PER_MTU** for each of the MTU values set on the NUMA node and add another **512MB** as buffer. Round the value up to a multiple of **1024**.

edpm_ovs_dpdk_memory_channels

Maps memory channels in the CPU per NUMA node. edpm_ovs_dpdk_memory_channels is the other_config:dpdk-extra="-n <value>" value in OVS. Observe the following recommendations:

- Use **dmidecode -t memory** or your hardware manual to determine the number of memory channels available.
- Use **ls /sys/devices/system/node/node* -d** to determine the number of NUMA nodes.
- Divide the number of memory channels available by the number of NUMA nodes.

edpm_ovs_dpdk_vhost_postcopy_support

Enable or disable OVS-DPDK vhost post-copy support. Setting this to **true** enables post-copy support for all vhost user client ports.

edpm_nova_libvirt_qemu_group

Set **edpm_nova_libvirt_qemu_group** to **hugetlbfs `so that the `ovs-vswitchd** and **qemu** processes can access the shared huge pages and UNIX socket that configures the **virtio-net device**. This value is role-specific and should be applied to any role leveraging OVS-DPDK.

edpm_ovn_bridge_mappings

List of bridge and dpdk ports mappings.

edpm_kernel_args

Provides multiple kernel arguments to **/etc/default/grub** for the compute nodes at boot time.

5.2.2. Configuration map parameters

The following list describes parameters that you can use in **ConfigMap** sections:

cpu_shared_set

List or range of host CPU cores used to determine the host CPUs that instance emulator threads should be offloaded to for instances configured with the share emulator thread policy (**hw::emulator_threads_policy=share**).

cpu_dedicated_set

A comma-separated list or range of physical host CPU numbers to which processes for pinned instance CPUs can be scheduled.

- Exclude all cores from the **edpm_ovs_dpdk_pmd_core_list**.
- Include all remaining cores.
- Pair the sibling threads together.

reserved_host_memory_mb

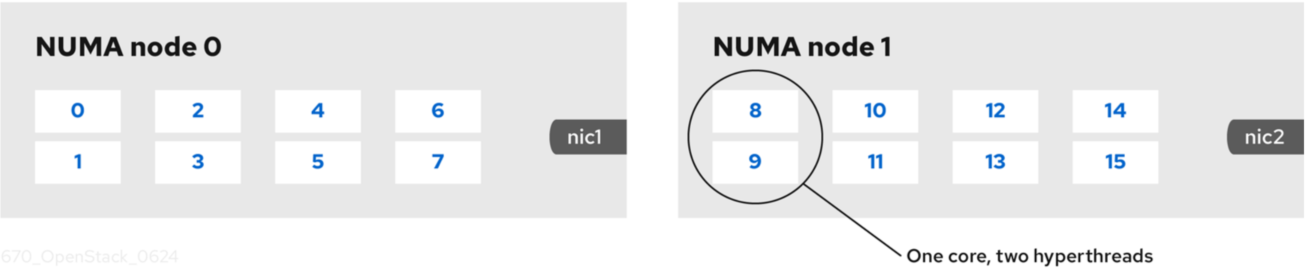
Reserves memory in MB for tasks on the host. Use the static recommended value of **4096MB**.

5.3. TWO NUMA NODE EXAMPLE OVS-DPDK DEPLOYMENT

The Red Hat OpenStack Services on OpenShift (RHOSO) Compute node in the following example includes two NUMA nodes:

- NUMA 0 has logical cores 0-7 (four physical cores). The sibling thread pairs are (0,1), (2,3), (4,5), and (6,7)
- NUMA 1 has cores 8-15. The sibling thread pairs are (8,9), (10,11), (12,13), and (14,15).
- Each NUMA node connects to a physical NIC, namely NIC1 on NUMA 0, and NIC2 on NUMA 1.

Figure 5.2. OVS-DPDK: two NUMA nodes example



NOTE

Reserve the first physical cores or both thread pairs on each NUMA node (0,1 and 8,9) for non-datapath DPDK processes.

This example also assumes a 1500 MTU configuration, so the **OvsDpdkSocketMemory** is the same for all use cases:

```
OvsDpdkSocketMemory: "1024,1024"
```

NIC 1 for DPDK, with one physical core for PMD

In this use case, you allocate one physical core on NUMA 0 for PMD. You must also allocate one physical core on NUMA 1, even though DPDK is not enabled on the NIC for that NUMA node. The remaining cores are allocated for guest instances. The resulting parameter settings are:

```
OvsPmdCoreList: "2,3,10,11"
NovaComputeCpuDedicatedSet: "4,5,6,7,12,13,14,15"
```

NIC 1 for DPDK, with two physical cores for PMD

In this use case, you allocate two physical cores on NUMA 0 for PMD. You must also allocate one physical core on NUMA 1, even though DPDK is not enabled on the NIC for that NUMA node. The remaining cores are allocated for guest instances. The resulting parameter settings are:

```
OvsPmdCoreList: "2,3,4,5,10,11"
NovaComputeCpuDedicatedSet: "6,7,12,13,14,15"
```

NIC 2 for DPDK, with one physical core for PMD

In this use case, you allocate one physical core on NUMA 1 for PMD. You must also allocate one physical core on NUMA 0, even though DPDK is not enabled on the NIC for that NUMA node. The remaining cores are allocated for guest instances. The resulting parameter settings are:

```
OvsPmdCoreList: "2,3,10,11"
NovaComputeCpuDedicatedSet: "4,5,6,7,12,13,14,15"
```

NIC 2 for DPDK, with two physical cores for PMD

In this use case, you allocate two physical cores on NUMA 1 for PMD. You must also allocate one physical core on NUMA 0, even though DPDK is not enabled on the NIC for that NUMA node. The remaining cores are allocated for guest instances. The resulting parameter settings are:

```
OvsPmdCoreList: "2,3,10,11,12,13"
NovaComputeCpuDedicatedSet: "4,5,6,7,14,15"
```

NIC 1 and NIC2 for DPDK, with two physical cores for PMD

In this use case, you allocate two physical cores on each NUMA node for PMD. The remaining cores are allocated for guest instances. The resulting parameter settings are:

```
OvsPmdCoreList: "2,3,4,5,10,11,12,13"
NovaComputeCpuDedicatedSet: "6,7,14,15"
```

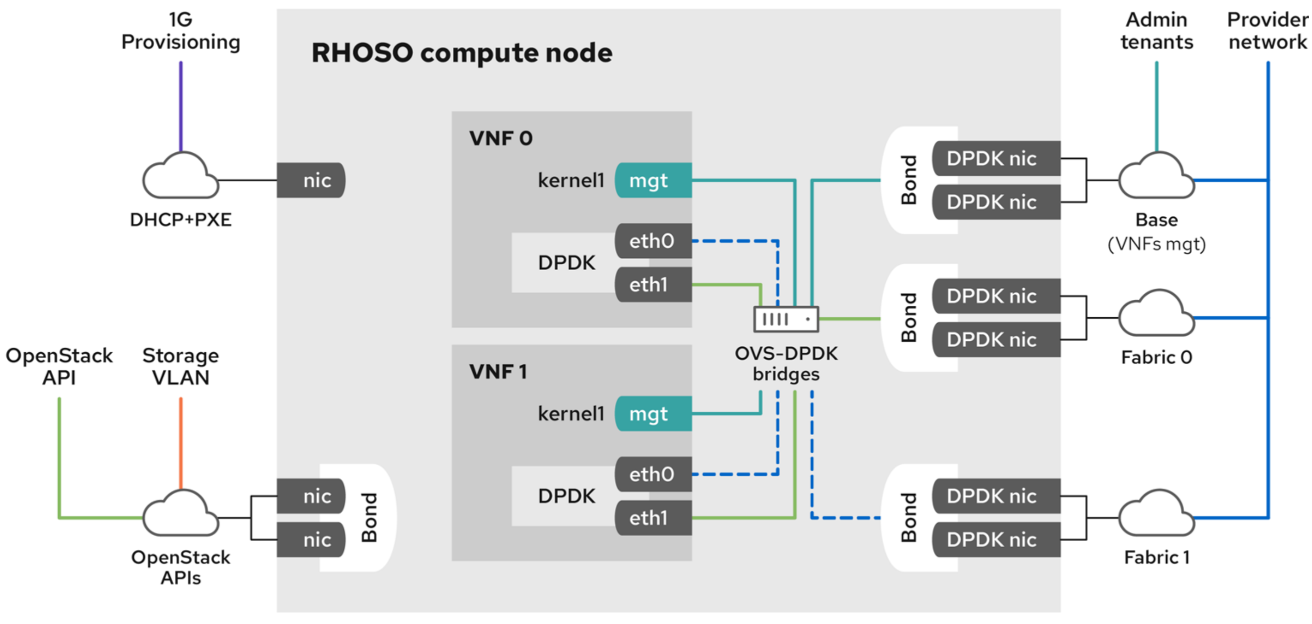
5.4. TOPOLOGY OF AN NFV OVS-DPDK DEPLOYMENT

This example deployment shows an OVS-DPDK configuration and consists of two virtual network functions (VNFs) with two interfaces each:

- The management interface, represented by **mgt**.
- The data plane interface.

In the OVS-DPDK deployment, the VNFs operate with inbuilt DPDK that supports the physical interface. OVS-DPDK enables bonding at the vSwitch level. For improved performance in your OVS-DPDK deployment, it is recommended that you separate kernel and OVS-DPDK NICs. To separate the management (**mgt**) network, connected to the Base provider network for the virtual machine, ensure you have additional NICs. The Compute node consists of two regular NICs for the Red Hat OpenStack Platform API management that can be reused by the Ceph API but cannot be shared with any OpenStack project.

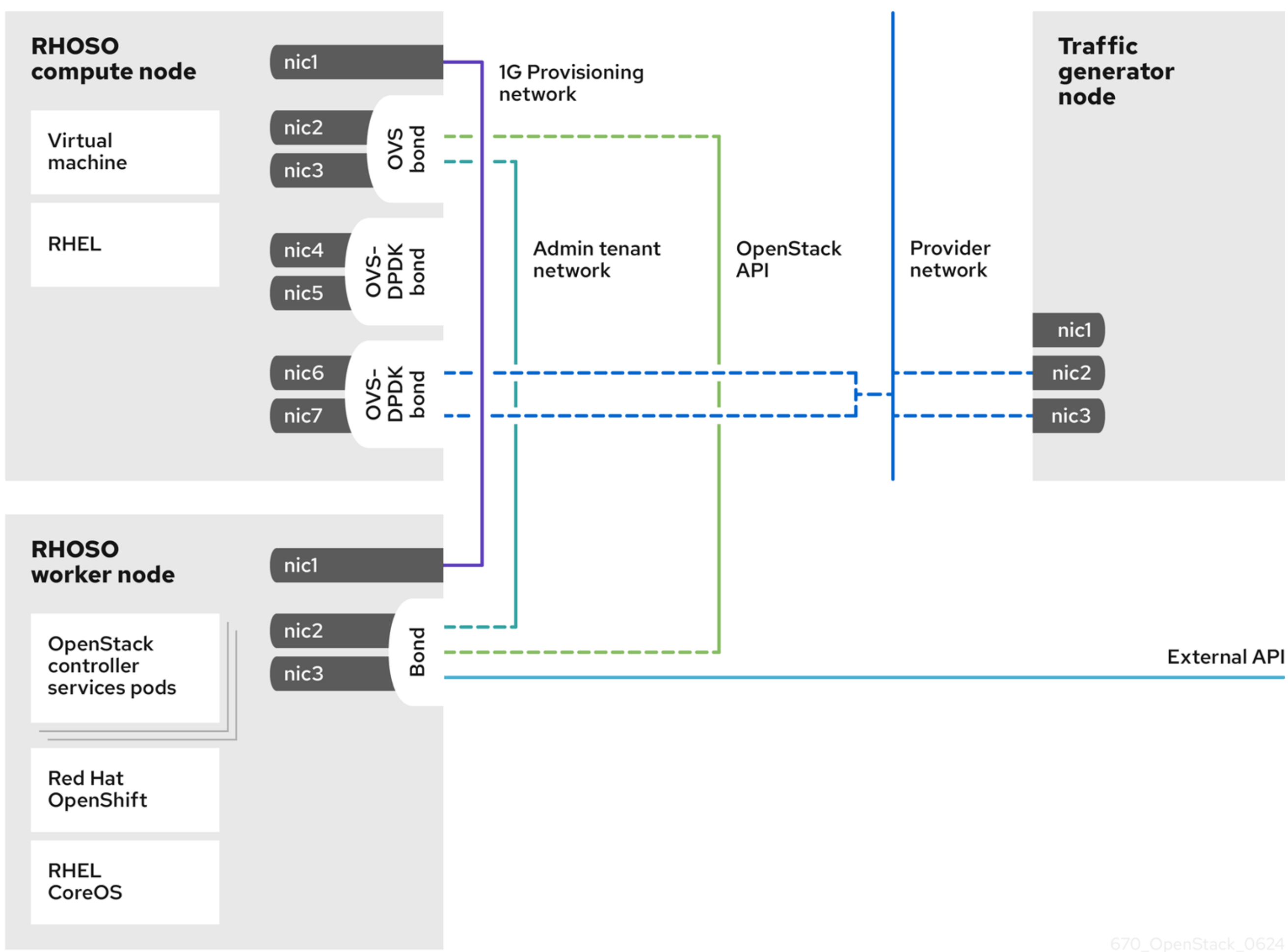
Figure 5.3. Compute node: NFV OVS-DPDK



OVS-DPDK Topology for NFV

The following image shows the topology for OVS-DPDK on an NFV environment.

Figure 5.4. OVS-DPDK Topology for NFV



CHAPTER 6. INSTALLING AND PREPARING THE OPERATORS

You install the Red Hat OpenStack Services on OpenShift (RHOSO) OpenStack Operator (**openstack-operator**) and create the RHOSO control plane on an operational Red Hat OpenShift Container Platform (RHOCP) cluster. You install the OpenStack Operator by using the RHOCP web console. You perform the control plane installation tasks and all data plane creation tasks on a workstation that has access to the RHOCP cluster.

6.1. PREREQUISITES

- An operational RHOCP cluster, version 4.16. For the RHOCP system requirements, see Red Hat OpenShift Container Platform cluster requirements in *Planning your deployment*.
- The **oc** command line tool is installed on your workstation.
- You are logged in to the RHOCP cluster as a user with **cluster-admin** privileges.

6.2. INSTALLING THE OPENSTACK OPERATOR

You use OperatorHub on the Red Hat OpenShift Container Platform (RHOCP) web console to install the OpenStack Operator (**openstack-operator**) on your RHOCP cluster.

Procedure

1. Log in to the RHOCP web console as a user with **cluster-admin** permissions.
2. Select **Operators → OperatorHub**.
3. In the **Filter by keyword** field, type **OpenStack**.
4. Click the **OpenStack Operator** tile with the **Red Hat** source label.
5. Read the information about the Operator and click **Install**.
6. On the **Install Operator** page, select "Operator recommended Namespace: openstack-operators" from the **Installed Namespace** list.
7. Click **Install** to make the Operator available to the **openstack-operators** namespace. The Operators are deployed and ready when the Status of the OpenStack Operator is **Succeeded**.

CHAPTER 7. PREPARING RED HAT OPENSHIFT CONTAINER PLATFORM FOR RED HAT OPENSTACK SERVICES ON OPENSHIFT

You install Red Hat OpenStack Services on OpenShift (RHOSO) on an operational Red Hat OpenShift Container Platform (RHOCP) cluster. To prepare for installing and deploying your RHOSO environment, you must configure the RHOCP worker nodes and the RHOCP networks on your RHOCP cluster.

7.1. CONFIGURING RED HAT OPENSHIFT CONTAINER PLATFORM NODES FOR A RED HAT OPENSTACK PLATFORM DEPLOYMENT

Red Hat OpenStack Services on OpenShift (RHOSO) services run on Red Hat OpenShift Container Platform (RHOCP) worker nodes. By default, the OpenStack Operator deploys RHOSO services on any worker node. You can use node labels in your **OpenStackControlPlane** custom resource (CR) to specify which RHOCP nodes host the RHOSO services. By pinning some services to specific infrastructure nodes rather than running the services on all of your RHOCP worker nodes, you optimize the performance of your deployment. You can create labels for the RHOCP nodes, or you can use the existing labels, and then specify those labels in the **OpenStackControlPlane** CR by using the **nodeSelector** field.

For example, the Block Storage service (cinder) has different requirements for each of its services:

- The **cinder-scheduler** service is a very light service with low memory, disk, network, and CPU usage.
- The **cinder-api** service has high network usage due to resource listing requests.
- The **cinder-volume** service has high disk and network usage because many of its operations are in the data path, such as offline volume migration, and creating a volume from an image.
- The **cinder-backup** service has high memory, network, and CPU requirements.

Therefore, you can pin the **cinder-api**, **cinder-volume**, and **cinder-backup** services to dedicated nodes and let the OpenStack Operator place the **cinder-scheduler** service on a node that has capacity.

Additional resources

- Placing pods on specific nodes using node selectors
- Machine configuration overview
- Node Feature Discovery Operator

7.2. CREATING A STORAGE CLASS

You must create a storage class for your Red Hat OpenShift Container Platform (RHOCP) cluster storage back end, to provide persistent volumes to Red Hat OpenStack Services on OpenShift (RHOSO) pods. Red Hat recommends that you use the Logical Volume Manager (LVM) Storage storage class with RHOSO, although you can use other implementations, such as Container Storage Interface (CSI) or OpenShift Data Foundation (ODF). You specify this storage class as the cluster storage back end for the RHOSO deployment. Red Hat recommends that you use a storage back end based on SSD or NVMe drives for the storage class.

You must wait until the LVM Storage Operator announces that the storage is available before creating the control plane. The LVM Storage Operator announces that the cluster and LVMS storage configuration is complete through the annotation for the volume group to the worker node object. If you deploy pods before all the control plane nodes are ready, then multiple PVCs and pods are scheduled on the same nodes.

To check that the storage is ready, you can query the nodes in your **lvmclusters.lvm.topolvm.io** object. For example, run the following command if you have three worker nodes and your device class for the LVM Storage Operator is named "local-storage":

```
# oc get node -l "topology.topolvm.io/node in ($(oc get nodes -l node-role.kubernetes.io/worker -o
name | cut -d '/' -f 2 | tr '\n' ',' | sed 's/.\{1\}$//'))" -
o=jsonpath='{.items[*].metadata.annotations.capacity\.topolvm\.io/local-storage}' | tr ' ' '\n'
```

The storage is ready when this command returns three non-zero values

For more information about how to configure the LVM Storage storage class, see Persistent storage using Logical Volume Manager Storage in the RHOCP *Storage* guide.

7.3. CREATING THE OPENSTACK NAMESPACE

You must create a namespace within your Red Hat OpenShift Container Platform (RHOCP) environment for the service pods of your Red Hat OpenStack Services on OpenShift (RHOSO) deployment. The service pods of each RHOSO deployment exist in their own namespace within the RHOCP environment.

Prerequisites

- You are logged on to a workstation that has access to the RHOCP cluster, as a user with **cluster-admin** privileges.

Procedure

1. Create the **openstack** project for the deployed RHOSO environment:

   ```
   $ oc new-project openstack
   ```

2. Ensure the **openstack** namespace is labeled to enable privileged pod creation by the OpenStack Operators:

   ```
   $ oc get namespace openstack -ojsonpath='{.metadata.labels}' | jq
   {
     "kubernetes.io/metadata.name": "openstack",
     "pod-security.kubernetes.io/enforce": "privileged",
     "security.openshift.io/scc.podSecurityLabelSync": "false"
   }
   ```

   If the security context constraint (SCC) is not "privileged", use the following commands to change it:

   ```
   $ oc label ns openstack security.openshift.io/scc.podSecurityLabelSync=false --overwrite
   $ oc label ns openstack pod-security.kubernetes.io/enforce=privileged --overwrite
   ```

3. Optional: To remove the need to specify the namespace when executing commands on the **openstack** namespace, set the default **namespace** to **openstack**:

```
$ oc project openstack
```

7.4. PROVIDING SECURE ACCESS TO THE RED HAT OPENSTACK SERVICES ON OPENSHIFT SERVICES

You must create a **Secret** custom resource (CR) to provide secure access to the Red Hat OpenStack Services on OpenShift (RHOSO) service pods.

> **WARNING**
>
> You cannot change a service password once the control plane is deployed. If a service password is changed in **osp-secret** after deploying the control plane, the service is reconfigured to use the new password but the password is not updated in the Identity service (keystone). This results in a service outage.

Procedure

1. Create a **Secret** CR file on your workstation, for example, **openstack_service_secret.yaml**.

2. Add the following initial configuration to **openstack_service_secret.yaml**:

```
apiVersion: v1
data:
  AdminPassword: <base64_password>
  AodhPassword: <base64_password>
  AodhDatabasePassword: <base64_password>
  BarbicanDatabasePassword: <base64_password>
  BarbicanPassword: <base64_password>
  BarbicanSimpleCryptoKEK: <base64_fernet_key>
  CeilometerPassword: <base64_password>
  CinderDatabasePassword: <base64_password>
  CinderPassword: <base64_password>
  DatabasePassword: <base64_password>
  DbRootPassword: <base64_password>
  DesignateDatabasePassword: <base64_password>
  DesignatePassword: <base64_password>
  GlanceDatabasePassword: <base64_password>
  GlancePassword: <base64_password>
  HeatAuthEncryptionKey: <base64_password>
  HeatDatabasePassword: <base64_password>
  HeatPassword: <base64_password>
  IronicDatabasePassword: <base64_password>
  IronicInspectorDatabasePassword: <base64_password>
  IronicInspectorPassword: <base64_password>
  IronicPassword: <base64_password>
  KeystoneDatabasePassword: <base64_password>
  ManilaDatabasePassword: <base64_password>
```

```
  ManilaPassword: <base64_password>
  MetadataSecret: <base64_password>
  NeutronDatabasePassword: <base64_password>
  NeutronPassword: <base64_password>
  NovaAPIDatabasePassword: <base64_password>
  NovaAPIMessageBusPassword: <base64_password>
  NovaCell0DatabasePassword: <base64_password>
  NovaCell0MessageBusPassword: <base64_password>
  NovaCell1DatabasePassword: <base64_password>
  NovaCell1MessageBusPassword: <base64_password>
  NovaPassword: <base64_password>
  OctaviaDatabasePassword: <base64_password>
  OctaviaPassword: <base64_password>
  PlacementDatabasePassword: <base64_password>
  PlacementPassword: <base64_password>
  SwiftPassword: <base64_password>
kind: Secret
metadata:
  name: osp-secret
  namespace: openstack
type: Opaque
```

- Replace **<base64_password>** with a 32-character key that is base64 encoded. You can use the following command to manually generate a base64 encoded password:

  ```
  $ echo -n <password> | base64
  ```

  Alternatively, if you are using a Linux workstation and you are generating the **Secret** CR definition file by using a Bash command such as **cat**, you can replace **<base64_password>** with the following command to auto-generate random passwords for each service:

  ```
  $(tr -dc 'A-Za-z0-9' < /dev/urandom | head -c 32 | base64)
  ```

- Replace the **<base64_fernet_key>** with a fernet key that is base64 encoded. You can use the following command to manually generate the fernet key:

  ```
  python3 -c "from cryptography.fernet import Fernet;
  print(Fernet.generate_key().decode('UTF-8'))" | base64
  ```

NOTE

The **HeatAuthEncryptionKey** password must be a 32-character key for Orchestration service (heat) encryption. If you increase the length of the passwords for all other services, ensure that the **HeatAuthEncryptionKey** password remains at length 32.

3. Create the **Secret** CR in the cluster:

```
$ oc create -f openstack_service_secret.yaml -n openstack
```

4. Verify that the **Secret** CR is created:

```
$ oc describe secret osp-secret -n openstack
```

CHAPTER 8. PREPARING NETWORKS FOR RHOSO WITH NFV

To prepare for configuring and deploying your Red Hat OpenStack Services on OpenShift (RHOSO) on a network functions virtualization (NFV) environment, you must configure the Red Hat OpenShift Container Platform (RHOCP) networks on your RHOCP cluster.

8.1. DEFAULT RED HAT OPENSTACK SERVICES ON OPENSHIFT NETWORKS

The following physical data center networks are typically implemented for a Red Hat OpenStack Services on OpenShift (RHOSO) deployment:

- Control plane network: This network is used by the OpenStack Operator for Ansible SSH access to deploy and connect to the data plane nodes from the Red Hat OpenShift Container Platform (RHOCP) environment. This network is also used by data plane nodes for live migration of instances.
- External network: (Optional) You can configure an external network if one is required for your environment. For example, you might create an external network for any of the following purposes:
  - To provide virtual machine instances with Internet access.
  - To create flat provider networks that are separate from the control plane.
  - To configure VLAN provider networks on a separate bridge from the control plane.
  - To provide access to virtual machine instances with floating IPs on a network other than the control plane network.
- Internal API network: This network is used for internal communication between RHOSO components.
- Storage network: This network is used for block storage, RBD, NFS, FC, and iSCSI.
- Tenant (project) network: This network is used for data communication between virtual machine instances within the cloud deployment.
- Storage Management network: (Optional) This network is used by storage components. For example, Red Hat Ceph Storage uses the Storage Management network in a hyperconverged infrastructure (HCI) environment as the **cluster_network** to replicate data.

> **NOTE**
>
> For more information on Red Hat Ceph Storage network configuration, see Ceph network configuration in the *Red Hat Ceph Storage Configuration Guide* .

The following table details the default networks used in a RHOSO deployment. If required, you can update the networks for your environment.

NOTE

By default, the control plane and external networks do not use VLANs. Networks that do not use VLANs must be placed on separate NICs. You can use a VLAN for the control plane network on new RHOSO deployments. You can also use the Native VLAN on a trunked interface as the non-VLAN network. For example, you can have the control plane and the internal API on one NIC, and the external network with no VLAN on a separate NIC.

Table 8.1. Default RHOSO networks

Network name	VLAN	CIDR	NetConfig **allocationRange**	MetalLB **IPAddressPool** range	net-attach-def **ipam** range	OCP worker **nncp** range
ctlplane	n/a	192.168.122.0/24	192.168.122.100 - 192.168.122.250	192.168.122.80 - 192.168.122.90	192.168.122.30 - 192.168.122.70	192.168.122.10 - 192.168.122.20
external	n/a	10.0.0.0/24	10.0.0.100 - 10.0.0.250	n/a	n/a	
internalapi	20	172.17.0.0/24	172.17.0.100 - 172.17.0.250	172.17.0.80 - 172.17.0.90	172.17.0.30 - 172.17.0.70	172.17.0.10 - 172.17.0.20
storage	21	172.18.0.0/24	172.18.0.100 - 172.18.0.250	n/a	172.18.0.30 - 172.18.0.70	172.18.0.10 - 172.18.0.20
tenant	22	172.19.0.0/24	172.19.0.100 - 172.19.0.250	n/a	172.19.0.30 - 172.19.0.70	172.19.0.10 - 172.19.0.20
storageMgmt	23	172.20.0.0/24	172.20.0.100 - 172.20.0.250	n/a	172.20.0.30 - 172.20.0.70	172.20.0.10 - 172.20.0.20

8.2. NIC CONFIGURATIONS FOR NFV

The Red Hat OpenStack Services on OpenShift (RHOSO) nodes that host the data plane require one of the following NIC configurations:

- Single NIC configuration - One NIC for the provisioning network on the native VLAN and tagged VLANs that use subnets for the different data plane network types.
- Dual NIC configuration - One NIC for the provisioning network and the other NIC for the external network.
- Dual NIC configuration - One NIC for the provisioning network on the native VLAN, and the other NIC for tagged VLANs that use subnets for different data plane network types.

- Multiple NIC configuration - Each NIC uses a subnet for a different data plane network type.

8.3. PREPARING RHOCP FOR RHOSO NETWORKS

The Red Hat OpenStack Services on OpenShift (RHOSO) services run as a Red Hat OpenShift Container Platform (RHOCP) workload. You use the NMState Operator to connect the worker nodes to the required isolated networks. You create a **NetworkAttachmentDefinition** (**net-attach-def**) custom resource (CR) for each isolated network to attach service pods to the isolated networks, where needed. You use the MetalLB Operator to expose internal service endpoints on the isolated networks. By default, the public service endpoints are exposed as RHOCP routes.

You must also create an **L2Advertisement** resource to define how the Virtual IPs (VIPs) are announced, and an **IPAddressPool** resource to configure which IPs can be used as VIPs. In layer 2 mode, one node assumes the responsibility of advertising a service to the local network.

> **NOTE**
>
> The examples in the following procedure use IPv4 addresses. You can use IPv6 addresses instead of IPv4 addresses. Dual stack IPv4/6 is not available. For information about how to configure IPv6 addresses, see the following resources in the RHOCP *Networking* guide:
>
> - Installing the Kubernetes NMState Operator
> - Configuring MetalLB address pools

Procedure

1. Create a **NodeNetworkConfigurationPolicy** (**nncp**) CR file on your workstation, for example, **openstack-nncp.yaml**.
2. Retrieve the names of the worker nodes in the RHOCP cluster:

```
$ oc get nodes -l node-role.kubernetes.io/worker -o jsonpath="{.items[*].metadata.name}"
```

3. Discover the network configuration:

```
$ oc get nns/<worker_node> -o yaml | more
```

   - Replace **<worker_node>** with the name of a worker node retrieved in step 2, for example, **worker-1**. Repeat this step for each worker node.

4. In the **nncp** CR file, configure the interfaces for each isolated network on each worker node in the RHOCP cluster. For information about the default physical data center networks that must be configured with network isolation, see Default Red Hat OpenStack Services on OpenShift networks.
   In the following example, the **nncp** CR configures the **enp6s0** interface for worker node 1, **osp-enp6s0-worker-1**, to use VLAN interfaces with IPv4 addresses for network isolation:

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: osp-enp6s0-worker-1
spec:
```

```
desiredState:
  interfaces:
  - description: internalapi vlan interface
    ipv4:
      address:
      - ip: 172.17.0.10
        prefix-length: 24
      enabled: true
      dhcp: false
    ipv6:
      enabled: false
    name: internalapi
    state: up
    type: vlan
    vlan:
      base-iface: enp6s0
      id: 20
      reorder-headers: true
  - description: storage vlan interface
    ipv4:
      address:
      - ip: 172.18.0.10
        prefix-length: 24
      enabled: true
      dhcp: false
    ipv6:
      enabled: false
    name: storage
    state: up
    type: vlan
    vlan:
      base-iface: enp6s0
      id: 21
      reorder-headers: true
  - description: tenant vlan interface
    ipv4:
      address:
      - ip: 172.19.0.10
        prefix-length: 24
      enabled: true
      dhcp: false
    ipv6:
      enabled: false
    name: tenant
    state: up
    type: vlan
    vlan:
      base-iface: enp6s0
      id: 22
      reorder-headers: true
  - description: Configuring enp6s0
    ipv4:
      address:
      - ip: 192.168.122.10
        prefix-length: 24
      enabled: true
```

```
        dhcp: false
      ipv6:
        enabled: false
      mtu: 1500
      name: enp6s0
      state: up
      type: ethernet
  nodeSelector:
    kubernetes.io/hostname: worker-1
    node-role.kubernetes.io/worker: ""
```

5. Create the **nncp** CR in the cluster:

```
$ oc apply -f openstack-nncp.yaml
```

6. Verify that the **nncp** CR is created:

```
$ oc get nncp -w
NAME                   STATUS        REASON
osp-enp6s0-worker-1   Progressing   ConfigurationProgressing
osp-enp6s0-worker-1   Progressing   ConfigurationProgressing
osp-enp6s0-worker-1   Available     SuccessfullyConfigured
```

7. Create a **NetworkAttachmentDefinition** (**net-attach-def**) CR file on your workstation, for example, **openstack-net-attach-def.yaml**.

8. In the **NetworkAttachmentDefinition** CR file, configure a **NetworkAttachmentDefinition** resource for each isolated network to attach a service deployment pod to the network. The following examples create a **NetworkAttachmentDefinition** resource for the **internalapi**, **storage**, **ctlplane**, and **tenant** networks of type **macvlan**:

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: internalapi
  namespace: openstack (1)
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "internalapi",
      "type": "macvlan",
      "master": "internalapi", (2)
      "ipam": { (3)
        "type": "whereabouts",
        "range": "172.17.0.0/24",
        "range_start": "172.17.0.30", (4)
        "range_end": "172.17.0.70"
      }
    }
---
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
```

```
  name: ctlplane
  namespace: openstack
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "ctlplane",
      "type": "macvlan",
      "master": "enp6s0",
      "ipam": {
        "type": "whereabouts",
        "range": "192.168.122.0/24",
        "range_start": "192.168.122.30",
        "range_end": "192.168.122.70"
      }
    }
---
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: storage
  namespace: openstack
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "storage",
      "type": "macvlan",
      "master": "storage",
      "ipam": {
        "type": "whereabouts",
        "range": "172.18.0.0/24",
        "range_start": "172.18.0.30",
        "range_end": "172.18.0.70"
      }
    }
---
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: tenant
  namespace: openstack
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "tenant",
      "type": "macvlan",
      "master": "tenant",
      "ipam": {
        "type": "whereabouts",
        "range": "172.19.0.0/24",
        "range_start": "172.19.0.30",
        "range_end": "172.19.0.70"
      }
    }
```

1. The namespace where the services are deployed.
2. The node interface name associated with the network, as defined in the **nncp** CR.
3. The **whereabouts** CNI IPAM plugin to assign IPs to the created pods from the range **.30 - .70**.
4. The IP address pool range must not overlap with the MetalLB **IPAddressPool** range and the **NetConfig allocationRange**.

9. Create the **NetworkAttachmentDefinition** CR in the cluster:

```
$ oc apply -f openstack-net-attach-def.yaml
```

10. Verify that the **NetworkAttachmentDefinition** CR is created:

```
$ oc get net-attach-def -n openstack
```

11. Create an **IPAddressPool** CR file on your workstation, for example, **openstack-ipaddresspools.yaml**.

12. In the **IPAddressPool** CR file, configure an **IPAddressPool** resource on the isolated network to specify the IP address ranges over which MetalLB has authority:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: internalapi
  namespace: metallb-system
spec:
  addresses:
    - 172.17.0.80-172.17.0.90 (1)
  autoAssign: true
  avoidBuggyIPs: false
---
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: ctlplane
spec:
  addresses:
    - 192.168.122.80-192.168.122.90
  autoAssign: true
  avoidBuggyIPs: false
---
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: storage
spec:
  addresses:
    - 172.18.0.80-172.18.0.90
```

```
  autoAssign: true
  avoidBuggyIPs: false
---
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: tenant
spec:
  addresses:
    - 172.19.0.80-172.19.0.90
  autoAssign: true
  avoidBuggyIPs: false
```

1. The **IPAddressPool** range must not overlap with the **whereabouts** IPAM range and the NetConfig **allocationRange**.

For information about how to configure the other **IPAddressPool** resource parameters, see Configuring MetalLB address pools in the RHOCP *Networking* guide.

13. Create the **IPAddressPool** CR in the cluster:

```
$ oc apply -f openstack-ipaddresspools.yaml
```

14. Verify that the **IPAddressPool** CR is created:

```
$ oc describe -n metallb-system IPAddressPool
```

15. Create a **L2Advertisement** CR file on your workstation, for example, **openstack-l2advertisement.yaml**.

16. In the **L2Advertisement** CR file, configure **L2Advertisement** CRs to define which node advertises a service to the local network. Create one **L2Advertisement** resource for each network.
In the following example, each **L2Advertisement** CR specifies that the VIPs requested from the network address pools are announced on the interface that is attached to the VLAN:

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: internalapi
  namespace: metallb-system
spec:
  ipAddressPools:
  - internalapi
  interfaces:
  - internalapi (1)
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: ctlplane
  namespace: metallb-system
spec:
```

```
  ipAddressPools:
  - ctlplane
  interfaces:
  - enp6s0
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: storage
  namespace: metallb-system
spec:
  ipAddressPools:
  - storage
  interfaces:
  - storage
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: tenant
  namespace: metallb-system
spec:
  ipAddressPools:
  - tenant
  interfaces:
  - tenant
```

1. The interface where the VIPs requested from the VLAN address pool are announced.

For information about how to configure the other **L2Advertisement** resource parameters, see Configuring MetalLB with a L2 advertisement and label in the RHOCP *Networking* guide.

17. Create the **L2Advertisement** CRs in the cluster:

```
$ oc apply -f openstack-l2advertisement.yaml
```

18. Verify that the **L2Advertisement** CRs are created:

```
$ oc get -n metallb-system L2Advertisement
NAME          IPADDRESSPOOLS    IPADDRESSPOOL SELECTORS   INTERFACES
ctlplane      ["ctlplane"]                                 ["enp6s0"]
internalapi   ["internalapi"]                              ["internalapi"]
storage       ["storage"]                                  ["storage"]
tenant        ["tenant"]                                   ["tenant"]
```

19. If your cluster has OVNKubernetes as the network back end, then you must enable global forwarding so that MetalLB can work on a secondary network interface.

    a. Check the network back end used by your cluster:

    ```
    $ oc get network.operator cluster --output=jsonpath='{.spec.defaultNetwork.type}'
    ```

    b. If the back end is OVNKubernetes, then run the following command to enable global IP forwarding:

    –

```
$ oc patch network.operator cluster -p '{"spec":{"defaultNetwork":
{"ovnKubernetesConfig":{"gatewayConfig":{"ipForwarding": "Global"}}}}}' --type=merge
```

8.4. CREATING THE DATA PLANE NETWORK

To create the data plane network, you define a **NetConfig** custom resource (CR) and specify all the subnets for the data plane networks. You must define at least one control plane network for your data plane. You can also define VLAN networks to create network isolation for composable networks, such as **InternalAPI**, **Storage**, and **External**. Each network definition must include the IP address assignment.

TIP

Use the following commands to view the **NetConfig** CRD definition and specification schema:

```
$ oc describe crd netconfig

$ oc explain netconfig.spec
```

Procedure

1. Create a file named **openstack_netconfig.yaml** on your workstation.
2. Add the following configuration to **openstack_netconfig.yaml** to create the **NetConfig** CR:

```
apiVersion: network.openstack.org/v1beta1
kind: NetConfig
metadata:
  name: openstacknetconfig
  namespace: openstack
```

3. In the **openstack_netconfig.yaml** file, define the topology for each data plane network. To use the default Red Hat OpenStack Services on OpenShift (RHOSO) networks, you must define a specification for each network. For information about the default RHOSO networks, see Default Red Hat OpenStack Services on OpenShift networks. The following example creates isolated networks for the data plane:

```
spec:
  networks:
  - name: CtlPlane 1
    dnsDomain: ctlplane.example.com
    subnets: 2
    - name: subnet1 3
      allocationRanges: 4
      - end: 192.168.122.120
        start: 192.168.122.100
      - end: 192.168.122.200
        start: 192.168.122.150
      cidr: 192.168.122.0/24
      gateway: 192.168.122.1
  - name: InternalApi
    dnsDomain: internalapi.example.com
    subnets:
    - name: subnet1
```

```
      allocationRanges:
      - end: 172.17.0.250
        start: 172.17.0.100
      excludeAddresses: 5
      - 172.17.0.10
      - 172.17.0.12
      cidr: 172.17.0.0/24
      vlan: 20 6
  - name: External
    dnsDomain: external.example.com
    subnets:
    - name: subnet1
      allocationRanges:
      - end: 10.0.0.250
        start: 10.0.0.100
      cidr: 10.0.0.0/24
      gateway: 10.0.0.1
  - name: Storage
    dnsDomain: storage.example.com
    subnets:
    - name: subnet1
      allocationRanges:
      - end: 172.18.0.250
        start: 172.18.0.100
      cidr: 172.18.0.0/24
      vlan: 21
  - name: Tenant
    dnsDomain: tenant.example.com
    subnets:
    - name: subnet1
      allocationRanges:
      - end: 172.19.0.250
        start: 172.19.0.100
      cidr: 172.19.0.0/24
      vlan: 22
```

1. The name of the network, for example, **CtlPlane**.
2. The IPv4 subnet specification.
3. The name of the subnet, for example, **subnet1**.
4. The **NetConfig allocationRange**. The **allocationRange** must not overlap with the MetalLB **IPAddressPool** range and the IP address pool range.
5. Optional: List of IP addresses from the allocation range that must not be used by data plane nodes.
6. The network VLAN. For information about the default RHOSO networks, see Default Red Hat OpenStack Services on OpenShift networks.

4. Save the **openstack_netconfig.yaml** definition file.
5. Create the data plane network:

```
$ oc create -f openstack_netconfig.yaml -n openstack
```

6. To verify that the data plane network is created, view the **openstacknetconfig** resource:

```
$ oc get netconfig/openstacknetconfig -n openstack
```

If you see errors, check the underlying **network-attach-definition** and node network configuration policies:

```
$ oc get network-attachment-definitions -n openstack
$ oc get nncp
```

CHAPTER 9. CREATING THE CONTROL PLANE FOR NFV ENVIRONMENTS

The Red Hat OpenStack Services on OpenShift (RHOSO) control plane contains the RHOSO services that manage the cloud. These control plane services are services that provide APIs and do not run Compute node workloads. The RHOSO control plane services run as a Red Hat OpenShift Container Platform (RHOCP) workload, and you deploy these services using Operators in OpenShift. When you configure these OpenStack control plane services, you use one custom resource (CR) definition called **OpenStackControlPlane**.

> **NOTE**
>
> Creating the control plane also creates an **OpenStackClient** pod that you can access through a remote shell (**rsh**) to run RHOSO CLI commands.
>
> ```
> $ oc rsh -n openstack openstackclient
> ```

9.1. PREREQUISITES

- The RHOCP cluster is prepared for RHOSO network isolation. For more information, see Preparing RHOCP for RHOSO networks.
- The OpenStack Operator (**openstack-operator**) is installed. For more information, see Installing and preparing the Operators.
- The RHOCP cluster is not configured with any network policies that prevent communication between the **openstack-operators** namespace and the control plane namespace (default **openstack**). Use the following command to check the existing network policies on the cluster:

  ```
  $ oc get networkpolicy -n openstack
  ```

- You are logged on to a workstation that has access to the RHOCP cluster, as a user with **cluster-admin** privileges.
- Use the generic **CustomServiceConfig** interface available in each service's specification to override any and all service-specific configuration settings.

9.2. CREATING THE CONTROL PLANE

Define an **OpenStackControlPlane** custom resource (CR) to perform the following tasks:

- Create the control plane.
- Enable the Red Hat OpenStack Services on OpenShift (RHOSO) services.

The following procedure creates an initial control plane with the recommended configurations for each service. The procedure helps you create an operational control plane environment. You can use the environment to test and troubleshoot issues before additional required service customization. Services can be added and customized after the initial deployment.

For more information on how to customize your control plane after deployment, see the Customizing the Red Hat OpenStack Services on OpenShift deployment guide.

For more information, see Example **OpenStackControlPlane** CR.

TIP

Use the following commands to view the **OpenStackControlPlane** CRD definition and specification schema:

```
$ oc describe crd openstackcontrolplane

$ oc explain openstackcontrolplane.spec
```

For NFV environments, when you add the Networking service (neutron) and OVN service configurations, you must supply the following information:

- Physical networks where your gateways are located.
- Path to vhost sockets.
- VLAN ranges.
- Number of NUMA nodes.
- NICs that connect to the gateway networks.

NOTE

If you are using SR-IOV, you must also add the **sriovnicswitch** mechanism driver to the Networking service configuration.

Procedure

1. Create the **openstack** project for the deployed RHOSO environment:

```
$ oc new-project openstack
```

2. Ensure the **openstack** namespace is labeled to enable privileged pod creation by the OpenStack Operators:

```
$ oc get namespace openstack -ojsonpath='{.metadata.labels}' | jq
{
  "kubernetes.io/metadata.name": "openstack",
  "pod-security.kubernetes.io/enforce": "privileged",
  "security.openshift.io/scc.podSecurityLabelSync": "false"
}
```

If the security context constraint (SCC) is not "privileged", use the following commands to change it:

```
$ oc label ns openstack security.openshift.io/scc.podSecurityLabelSync=false --overwrite
$ oc label ns openstack pod-security.kubernetes.io/enforce=privileged --overwrite
```

3. Create a file on your workstation named **openstack_control_plane.yaml** to define the **OpenStackControlPlane** CR:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-control-plane
  namespace: openstack
```

4. Specify the **Secret** CR you created to provide secure access to the RHOSO service pods in Providing secure access to the Red Hat OpenStack Services on OpenShift services :

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-control-plane
spec:
  secret: osp-secret
```

5. Specify the **storageClass** you created for your Red Hat OpenShift Container Platform (RHOCP) cluster storage back end:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-control-plane
spec:
  secret: osp-secret
  storageClass: your-RHOCP-storage-class
```

NOTE

For information about storage classes, see Creating a storage class .

6. Add the following service configurations:

- Block Storage service (cinder):

```
  cinder:
    apiOverride:
      route: {}
    template:
      databaseInstance: openstack
      secret: osp-secret
      cinderAPI:
        replicas: 3
        override:
          service:
            internal:
              metadata:
                annotations:
                  metallb.universe.tf/address-pool: internalapi
                  metallb.universe.tf/allow-shared-ip: internalapi
                  metallb.universe.tf/loadBalancerIPs: 172.17.0.80
              spec:
                type: LoadBalancer
      cinderScheduler:
```

```
    replicas: 1
  cinderBackup:
    networkAttachments:
    - storage
    replicas: 0 # backend needs to be configured to activate the service
  cinderVolumes:
    volume1:
      networkAttachments:
      - storage
      replicas: 0 # backend needs to be configured to activate the service
```

IMPORTANT

This definition for the Block Storage service is only a sample. You might need to modify it for your NFV environment. For more information, see Planning storage and shared file systems in *Planning your deployment*.

NOTE

For the initial control plane deployment, the **cinderBackup** and **cinderVolumes** services are deployed but not activated (replicas: 0). You can configure your control plane post-deployment with a back end for the Block Storage service and the backup service.

- Compute service (nova):

```
nova:
  apiOverride:
    route: {}
  template:
    apiServiceTemplate:
      replicas: 3
      override:
        service:
          internal:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/allow-shared-ip: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.80
            spec:
              type: LoadBalancer
    schedulerServiceTemplate:
      customServiceConfig: |
        [filter_scheduler]
        enabled_filters = AvailabilityZoneFilter, ComputeFilter, ComputeCapabilitiesFilter,
ImagePropertiesFilter, ServerGroupAntiAffinityFilter, ServerGroupAffinityFilter,
PciPassthroughFilter, AggregateInstanceExtraSpecsFilter
        available_filters = nova.scheduler.filters.all_filters
    metadataServiceTemplate:
      replicas: 3
      override:
        service:
          metadata:
            annotations:
```

```
            metallb.universe.tf/address-pool: internalapi
            metallb.universe.tf/allow-shared-ip: internalapi
            metallb.universe.tf/loadBalancerIPs: 172.17.0.80
        spec:
          type: LoadBalancer
    schedulerServiceTemplate:
      replicas: 3
      override:
        service:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
    cellTemplates:
      cell1:
        noVNCProxyServiceTemplate:
          enabled: true
          networkAttachments:
          - ctlplane
    secret: osp-secret
```

> **NOTE**
>
> A full set of Compute services (nova) are deployed by default for each of the default cells, **cell0** and **cell1**: **nova-api**, **nova-metadata**, **nova-scheduler**, and **nova-conductor**. The **novncproxy** service is also enabled for **cell1** by default.

- DNS service for the data plane:

```
dns:
  template:
    options: (1)
    - key: server (2)
      values: (3)
      - 192.168.122.1
    - key: server
      values:
      - 192.168.122.2
    override:
      service:
        metadata:
          annotations:
            metallb.universe.tf/address-pool: ctlplane
            metallb.universe.tf/allow-shared-ip: ctlplane
            metallb.universe.tf/loadBalancerIPs: 192.168.122.80
        spec:
          type: LoadBalancer
    replicas: 2
```

1. Defines the dnsmasq instances required for each DNS server by using key-value pairs. In this example, there are two key-value pairs defined because there are two DNS servers

2. Specifies the dnsmasq parameter to customize for the deployed dnsmasq instance. Set to one of the following valid values:

   - **server**
   - **rev-server**
   - **srv-host**
   - **txt-record**
   - **ptr-record**
   - **rebind-domain-ok**
   - **naptr-record**
   - **cname**
   - **host-record**
   - **caa-record**
   - **dns-rr**
   - **auth-zone**
   - **synth-domain**
   - **no-negcache**
   - **local**

3. Specifies the values for the dnsmasq parameter. You can specify a generic DNS server as the value, for example, **1.1.1.1**, or a DNS server for a specific domain, for example, **/google.com/8.8.8.8**.

- A Galera cluster for use by all RHOSO services (**openstack**), and a Galera cluster for use by the Compute service for **cell1** (**openstack-cell1**):

```
galera:
  templates:
    openstack:
      storageRequest: 5000M
      secret: osp-secret
      replicas: 3
    openstack-cell1:
      storageRequest: 5000M
      secret: osp-secret
      replicas: 3
```

- Identity service (keystone)

```
keystone:
  apiOverride:
    route: {}
  template:
    override:
      service:
        internal:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
    databaseInstance: openstack
    secret: osp-secret
    replicas: 3
```

- Image service (glance):

```
glance:
  apiOverrides:
    default:
      route: {}
  template:
    databaseInstance: openstack
    storage:
      storageRequest: 10G
    secret: osp-secret
    keystoneEndpoint: default
    glanceAPIs:
      default:
        replicas: 0 # backend needs to be configured to activate the service
        override:
          service:
            internal:
              metadata:
                annotations:
                  metallb.universe.tf/address-pool: internalapi
                  metallb.universe.tf/allow-shared-ip: internalapi
                  metallb.universe.tf/loadBalancerIPs: 172.17.0.80
              spec:
                type: LoadBalancer
        networkAttachments:
        - storage
```

NOTE

For the initial control plane deployment, the Image service is deployed but not activated (replicas: 0). You can configure your control plane post-deployment with a back end for the Image service.

- Key Management service (barbican):

```
barbican:
  apiOverride:
    route: {}
  template:
    databaseInstance: openstack
    secret: osp-secret
    barbicanAPI:
      replicas: 3
      override:
        service:
          internal:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/allow-shared-ip: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.80
            spec:
              type: LoadBalancer
    barbicanWorker:
      replicas: 3
    barbicanKeystoneListener:
      replicas: 1
```

- Memcached:

```
memcached:
  templates:
    memcached:
       replicas: 3
```

- Networking service (neutron):

```
neutron:
  apiOverride:
    route: {}
  template:
    replicas: 3
    override:
      service:
        internal:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
    databaseInstance: openstack
    secret: osp-secret
    networkAttachments:
    - internalapi
    customServiceConfig: |
      [DEFAULT]
      global_physnet_mtu = 9000
      [ml2]
```

```
mechanism_drivers = ovn
[ovn]
vhost_sock_dir = <path>
[ml2_type_vlan]
network_vlan_ranges = <network_name1>:<VLAN-ID1>:<VLAN-ID2>,<network_name2>:<VLAN-ID1>:<VLAN-ID2>
```

- If you are using SR-IOV, you must also add the **sriovnicswitch** mechanism driver, for example, **mechanism_drivers = ovn,sriovnicswitch**.
- Replace **<path>** with the absolute path to the **vhost** sockets, for example, **/var/lib/vhost**.
- Replace **<network_name1>** and **<network_name2>** with the names of the physical networks that your gateways are on. (This network is set in the neutron network **provider:*name** field.)
- Replace **<VLAN-ID1>** and`<VLAN-ID2>` with the VLAN IDs you are using.

- Object Storage service (swift):

```
swift:
  enabled: true
  proxyOverride:
    route: {}
  template:
    swiftProxy:
      networkAttachments:
      - storage
      override:
        service:
          internal:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/allow-shared-ip: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.80
            spec:
              type: LoadBalancer
      replicas: 1
    swiftRing:
      ringReplicas: 1
    swiftStorage:
      networkAttachments:
      - storage
      replicas: 1
      storageClass: local-storage
      storageRequest: 10Gi
```

- OVN:

```
ovn:
  template:
    ovnDBCluster:
      ovndbcluster-nb:
        replicas: 3
        dbType: NB
```

```
      storageRequest: 10G
      networkAttachment: internalapi
    ovndbcluster-sb:
      dbType: SB
      storageRequest: 10G
      networkAttachment: internalapi
  ovnNorthd:
    networkAttachment: internalapi
  ovnController:
    networkAttachment: tenant
    nicMappings:
      <network_name>: <nic_name>
```

- Replace **<network_name>** with the name of the physical network your gateway is on. (This network is set in the neutron network **provider:*name** field.)
- Replace **<nic_name>** with the name of the NIC connecting to the gateway network.
- Optional: Add additional **<network_name>:<nic_name>** pairs under **nicMappings** as required.

- Placement service (placement):

```
placement:
  apiOverride:
    route: {}
  template:
    override:
      service:
        internal:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
    databaseInstance: openstack
    replicas: 3
    secret: osp-secret
```

- RabbitMQ:

```
rabbitmq:
  templates:
    rabbitmq:
      replicas: 3
      override:
        service:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.85
          spec:
            type: LoadBalancer
    rabbitmq-cell1:
```

```
        replicas: 3
        override:
          service:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.86
            spec:
              type: LoadBalancer
```

- Telemetry service (ceilometer, prometheus):

```
telemetry:
  enabled: true
  template:
    metricStorage:
      enabled: true
      monitoringStack:
        alertingEnabled: true
        scrapeInterval: 30s
        storage:
          strategy: persistent
          retention: 24h
          persistent:
            pvcStorageRequest: 20G
    autoscaling: (1)
      enabled: false
      aodh:
        passwordSelectors:
        databaseAccount: aodh
        databaseInstance: openstack
        memcachedInstance: memcached
        secret: osp-secret
      heatInstance: heat
    ceilometer:
      enabled: true
      secret: osp-secret
    logging:
      enabled: false
      ipaddr: 172.17.0.80
```

(1) You must have the **autoscaling** field present, even if autoscaling is disabled.

1. Create the control plane:

```
$ oc create -f openstack_control_plane.yaml -n openstack
```

NOTE

Creating the control plane also creates an **OpenStackClient** pod that you can access through a remote shell (**rsh**) to run RHOSO CLI commands.

```
$ oc rsh -n openstack openstackclient
```

2. Wait until RHOCP creates the resources related to the **OpenStackControlPlane** CR. Check the status of the control plane deployment:

```
$ oc get openstackcontrolplane -n openstack
```

Sample output

```
NAME                    STATUS    MESSAGE
openstack-control-plane   Unknown   Setup started
```

The **OpenStackControlPlane** resources are created when the status is "Setup complete".

TIP

Append the **-w** option to the end of the **get** command to track deployment progress.

NOTE

Creating the control plane also creates an **OpenStackClient** pod that you can access through a remote shell (**rsh**) to run RHOSO CLI commands.

```
$ oc rsh -n openstack openstackclient
```

3. Optional: Confirm that the control plane is deployed by reviewing the pods in the **openstack** namespace:

```
$ oc get pods -n openstack
```

The control plane is deployed when all the pods are either completed or running.

Verification

1. Open a remote shell connection to the **OpenStackClient** pod:

```
$ oc rsh -n openstack openstackclient
```

2. Confirm that the internal service endpoints are registered with each service:

```
$ openstack endpoint list -c 'Service Name' -c Interface -c URL --service glance
```

Sample output

```
+--------------+-----------+---------------------------------------------------------------+
| Service Name | Interface | URL                                                           |
+--------------+-----------+---------------------------------------------------------------+
| glance       | internal  | http://glance-internal.openstack.svc:9292                     |
| glance       | public    | http://glance-public-openstack.apps.ostest.test.metalkube.org |
+--------------+-----------+---------------------------------------------------------------+
```

3. Exit the **OpenStackClient** pod:

```
$ exit
```

9.3. EXAMPLE OPENSTACKCONTROLPLANE CR

The following example **OpenStackControlPlane** CR is a complete control plane configuration that includes all the key services that must always be enabled for a successful deployment.

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-control-plane
  namespace: openstack
spec:
  secret: osp-secret
  storageClass: your-RHOCP-storage-class 1
  cinder: 2
    apiOverride:
      route: {}
    template:
      databaseInstance: openstack
      secret: osp-secret
      cinderAPI:
        replicas: 3
        override:
          service:
            internal:
              metadata:
                annotations:
                  metallb.universe.tf/address-pool: internalapi
                  metallb.universe.tf/allow-shared-ip: internalapi
                  metallb.universe.tf/loadBalancerIPs: 172.17.0.80
              spec:
                type: LoadBalancer
      cinderScheduler:
        replicas: 1
      cinderBackup: 3
        networkAttachments:
        - storage
        replicas: 0 # backend needs to be configured to activate the service
      cinderVolumes: 4
        volume1:
          networkAttachments: 5
          - storage
          replicas: 0 # backend needs to be configured to activate the service
  nova: 6
    apiOverride: 7
      route: {}
    template:
      apiServiceTemplate:
        replicas: 3
        override:
          service:
```

```
          internal:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi 8
                metallb.universe.tf/allow-shared-ip: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.80 9
            spec:
              type: LoadBalancer
    metadataServiceTemplate:
      replicas: 3
      override:
        service:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
    schedulerServiceTemplate:
      replicas: 3
      override:
        service:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
    cellTemplates:
      cell0:
        cellDatabaseAccount: nova-cell0
        cellDatabaseInstance: openstack
        cellMessageBusInstance: rabbitmq
        hasAPIAccess: true
      cell1:
        cellDatabaseAccount: nova-cell1
        cellDatabaseInstance: openstack-cell1
        cellMessageBusInstance: rabbitmq-cell1
        noVNCProxyServiceTemplate:
          enabled: true
          networkAttachments:
          - internalapi
          - ctlplane
        hasAPIAccess: true
    secret: osp-secret
dns:
  template:
    options:
    - key: server
      values:
      - 192.168.122.1
    - key: server
      values:
      - 192.168.122.2
```

```
      override:
        service:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: ctlplane
              metallb.universe.tf/allow-shared-ip: ctlplane
              metallb.universe.tf/loadBalancerIPs: 192.168.122.80
          spec:
            type: LoadBalancer
      replicas: 2
  galera:
    templates:
      openstack:
        storageRequest: 5000M
        secret: osp-secret
        replicas: 3
      openstack-cell1:
        storageRequest: 5000M
        secret: osp-secret
        replicas: 3
  keystone:
    apiOverride:
      route: {}
    template:
      override:
        service:
          internal:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/allow-shared-ip: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.80
            spec:
              type: LoadBalancer
      databaseInstance: openstack
      secret: osp-secret
      replicas: 3
  glance:
    apiOverrides:
      default:
        route: {}
    template:
      databaseInstance: openstack
      storage:
        storageRequest: 10G
      secret: osp-secret
      keystoneEndpoint: default
      glanceAPIs:
        default:
          replicas: 0 # backend needs to be configured to activate the service
          override:
            service:
              internal:
                metadata:
                  annotations:
                    metallb.universe.tf/address-pool: internalapi
```

```
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
      networkAttachments:
      - storage
barbican:
  apiOverride:
    route: {}
  template:
    databaseInstance: openstack
    secret: osp-secret
    barbicanAPI:
      replicas: 3
      override:
        service:
          internal:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/allow-shared-ip: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.80
            spec:
              type: LoadBalancer
    barbicanWorker:
      replicas: 3
    barbicanKeystoneListener:
      replicas: 1
memcached:
  templates:
    memcached:
       replicas: 3
neutron:
  apiOverride:
    route: {}
  template:
    replicas: 3
    override:
      service:
        internal:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
    databaseInstance: openstack
    secret: osp-secret
    networkAttachments:
    - internalapi
swift:
  enabled: true
  proxyOverride:
    route: {}
  template:
```

```
    swiftProxy:
      networkAttachments:
      - storage
      override:
        service:
          internal:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/allow-shared-ip: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.80
            spec:
              type: LoadBalancer
      replicas: 1
    swiftRing:
      ringReplicas: 1
    swiftStorage:
      networkAttachments:
      - storage
      replicas: 1
      storageRequest: 10Gi
ovn:
  template:
    ovnDBCluster:
      ovndbcluster-nb:
        replicas: 3
        dbType: NB
        storageRequest: 10G
        networkAttachment: internalapi
      ovndbcluster-sb:
        dbType: SB
        storageRequest: 10G
        networkAttachment: internalapi
    ovnNorthd:
      networkAttachment: internalapi
placement:
  apiOverride:
    route: {}
  template:
    override:
      service:
        internal:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
    databaseInstance: openstack
    replicas: 3
    secret: osp-secret
rabbitmq: 10
  templates:
    rabbitmq:
      replicas: 3
```

```
      override:
        service:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.85 (11)
          spec:
            type: LoadBalancer
    rabbitmq-cell1:
      replicas: 3
      override:
        service:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.86 (12)
          spec:
            type: LoadBalancer
telemetry:
  enabled: true
  template:
    metricStorage:
      enabled: true
      monitoringStack:
        alertingEnabled: true
        scrapeInterval: 30s
        storage:
          strategy: persistent
          retention: 24h
          persistent:
            pvcStorageRequest: 20G
    autoscaling:
      enabled: false
      aodh:
        databaseAccount: aodh
        databaseInstance: openstack
        passwordSelector:
          aodhService: AodhPassword
        rabbitMqClusterName: rabbitmq
        serviceUser: aodh
        secret: osp-secret
      heatInstance: heat
    ceilometer:
      enabled: true
      secret: osp-secret
    logging:
      enabled: false
      ipaddr: 172.17.0.80
```

(1) The storage class that you created for your Red Hat OpenShift Container Platform (RHOCP) cluster storage back end.

(2) Service-specific parameters for the Block Storage service (cinder).

(3) The Block Storage service back end. For more information on configuring storage services, see the Configuring persistent storage guide.

4 The Block Storage service configuration. For more information on configuring storage services, see the Configuring persistent storage guide.

5 The list of networks that each service pod is directly attached to, specified by using the **NetworkAttachmentDefinition** resource names. A NIC is configured for the service for each specified network attachment.

NOTE

If you do not configure the isolated networks that each service pod is attached to, then the default pod network is used. For example, the Block Storage service uses the storage network to connect to a storage back end; the Identity service (keystone) uses an LDAP or Active Directory (AD) network; the **ovnDBCluster** and **ovnNorthd** services use the **internalapi** network; and the **ovnController** service uses the **tenant** network.

6 Service-specific parameters for the Compute service (nova).

7 Service API route definition. You can customize the service route by using route-specific annotations. For more information, see Route-specific annotations in the RHOCP *Networking* guide. Set **route:** to **{}** to apply the default route template.

8 The internal service API endpoint registered as a MetalLB service with the **IPAddressPool internalapi**.

9 The virtual IP (VIP) address for the service. The IP is shared with other services by default.

10 The RabbitMQ instances exposed to an isolated network with distinct IP addresses defined in the **loadBalancerIPs** annotation, as indicated in 11 and 12.

NOTE

Multiple RabbitMQ instances cannot share the same VIP as they use the same port. If you need to expose multiple RabbitMQ instances to the same network, then you must use distinct IP addresses.

11 The distinct IP address for a RabbitMQ instance that is exposed to an isolated network.

12 The distinct IP address for a RabbitMQ instance that is exposed to an isolated network.

9.4. REMOVING A SERVICE FROM THE CONTROL PLANE

You can completely remove a service and the service database from the control plane after deployment by disabling the service. Many services are enabled by default, which means that the OpenStack Operator creates resources such as the service database and Identity service (keystone) users, even if no service pod is created because **replicas** is set to **0**.

> **WARNING**
>
> Remove a service with caution. Removing a service is not the same as stopping service pods. Removing a service is irreversible. Disabling a service removes the service database and any resources that referenced the service are no longer tracked. Red Hat recommends creating a backup of the service database before removing a service.

Procedure

1. Open the **OpenStackControlPlane** CR file on your workstation.
2. Locate the service you want to remove from the control plane and disable it:

```
cinder:
  enabled: false
  apiOverride:
    route: {}
    ...
```

3. Update the control plane:

```
$ oc apply -f openstack_control_plane.yaml -n openstack
```

4. Wait until RHOCP removes the resource related to the disabled service. Run the following command to check the status:

```
$ oc get openstackcontrolplane -n openstack
NAME        STATUS  MESSAGE
openstack-control-plane  Unknown  Setup started
```

   The **OpenStackControlPlane** resource is updated with the disabled service when the status is "Setup complete".

   TIP

   Append the **-w** option to the end of the **get** command to track deployment progress.

5. Optional: Confirm that the pods from the disabled service are no longer listed by reviewing the pods in the **openstack** namespace:

```
$ oc get pods -n openstack
```

6. Check that the service is removed:

```
$ oc get cinder -n openstack
```

   This command returns the following message when the service is successfully removed:

```
No resources found in openstack namespace.
```

7. Check that the API endpoints for the service are removed from the Identity service (keystone):

```
$ oc rsh -n openstack openstackclient
$ openstack endpoint list --service volumev3
```

This command returns the following message when the API endpoints for the service are successfully removed:

```
No service with a type, name or ID of 'volumev3' exists.
```

9.5. ADDITIONAL RESOURCES

- Kubernetes NMState Operator
- The Kubernetes NMState project
- Load balancing with MetalLB
- MetalLB documentation
- MetalLB in layer 2 mode
- Specify network interfaces that LB IP can be announced from
- Multiple networks
- Using the Multus CNI in OpenShift
- macvlan plugin
- whereabouts IPAM CNI plugin - Extended configuration
- About advertising for the IP address pools
- Dynamic provisioning
- Configuring the Block Storage backup service in *Configuring persistent storage*.
- Configuring the Image service (glance) in *Configuring persistent storage*.

CHAPTER 10. CREATING THE DATA PLANE FOR SR-IOV AND DPDK ENVIRONMENTS

The Red Hat OpenStack Services on OpenShift (RHOSO) data plane consists of RHEL 9.4 nodes. Use the **OpenStackDataPlaneNodeSet** custom resource definition (CRD) to create the custom resources (CRs) that define the nodes and the layout of the data plane. After you have defined your **OpenStackDataPlaneNodeSet** CRs, you create an **OpenStackDataPlaneDeployment** CR that deploys each of your **OpenStackDataPlaneNodeSet** CRs.

An **OpenStackDataPlaneNodeSet** CR is a logical grouping of nodes of a similar type. A data plane typically consists of multiple **OpenStackDataPlaneNodeSet** CRs to define groups of nodes with different configurations and roles. You can use pre-provisioned or unprovisioned nodes in an **OpenStackDataPlaneNodeSet** CR:

- Pre-provisioned node: You have used your own tooling to install the operating system on the node before adding it to the data plane.
- Unprovisioned node: The node does not have an operating system installed before you add it to the data plane. The node is provisioned by using the Cluster Baremetal Operator (CBO) as part of the data plane creation and deployment process.

NOTE

You cannot include both pre-provisioned and unprovisioned nodes in the same OpenStackDataPlaneNodeSet CR.

To create and deploy a data plane, you must perform the following tasks:

1. Create a **Secret** CR for each node set for Ansible to use to execute commands on the data plane nodes.
2. Create the **OpenStackDataPlaneNodeSet** CRs that define the nodes and layout of the data plane.
3. Create the **OpenStackDataPlaneDeployment** CR that triggers the Ansible execution that deploys and configures the software for the specified list of **OpenStackDataPlaneNodeSet** CRs.

The following procedures create two simple node sets, one with pre-provisioned nodes, and one with bare-metal nodes that must be provisioned during the node set deployment. The procedures aim to get you up and running quickly with a data plane environment that you can use to troubleshoot issues and test the environment before adding all the customizations you require. You can add additional node sets to a deployed environment, and you can customize your deployed environment by updating the common configuration in the default **ConfigMap** CR for the service, and by creating custom services. For more information on how to customize your data plane after deployment, see the Customizing the Red Hat OpenStack Services on OpenShift deployment guide.

10.1. PREREQUISITES

- A functional control plane, created with the OpenStack Operator. For more information, see Creating the control plane for NFV environments .
- You are logged on to a workstation that has access to the Red Hat OpenShift Container Platform (RHOCP) cluster as a user with **cluster-admin** privileges.

- Use the generic **CustomServiceConfig** interface available in each service's specification to override any and all service-specific configuration settings.

10.2. CREATING THE DATA PLANE SECRETS

The data plane requires several **Secret** custom resources (CRs) to operate. The **Secret** CRs are used by the data plane nodes for the following functionality:

- To enable secure access between nodes:
  - You must generate an SSH key and create an SSH key **Secret** CR for each key to enable Ansible to manage the RHEL nodes on the data plane. Ansible executes commands with this user and key. You can create an SSH key for each node set in your data plane.
  - You must generate an SSH key and create an SSH key **Secret** CR for each key to enable migration of instances between Compute nodes.
- To register the operating system of the nodes that are not registered to the Red Hat Customer Portal.
- To enable repositories for the nodes.
- To provide access to libvirt.

Prerequisites

- Pre-provisioned nodes are configured with an SSH public key in the **$HOME/.ssh/authorized_keys** file for a user with passwordless **sudo** privileges. For information, see Configuring reserved user and group IDs in the RHEL *Configuring basic system settings* guide.

Procedure

1. For unprovisioned nodes, create the SSH key pair for Ansible:

```
$ ssh-keygen -f <key_file_name> -N "" -t rsa -b 4096
```

   - Replace **<key_file_name>** with the name to use for the key pair.

2. Create the **Secret** CR for Ansible and apply it to the cluster:

```
$ oc create secret generic dataplane-ansible-ssh-private-key-secret \
--save-config \
--dry-run=client \
[--from-file=authorized_keys=<key_file_name>.pub \]
--from-file=ssh-privatekey=<key_file_name> \
--from-file=ssh-publickey=<key_file_name>.pub \
-n openstack \
-o yaml | oc apply -f -
```

   - Replace **<key_file_name>** with the name and location of your SSH key pair file.
   - Include the **--from-file=authorized_keys** option for bare-metal nodes that must be provisioned when creating the data plane.

3. Create the SSH key pair for instance migration:

```
$ ssh-keygen -f ./nova-migration-ssh-key -t ecdsa-sha2-nistp521 -N ''
```

4. Create the **Secret** CR for migration and apply it to the cluster:

```
$ oc create secret generic nova-migration-ssh-key \
--save-config \
--from-file=ssh-privatekey=nova-migration-ssh-key \
--from-file=ssh-publickey=nova-migration-ssh-key.pub \
-n openstack \
-o yaml | oc apply -f -
```

5. Create a file on your workstation named **secret_subscription.yaml** that contains the **subscription-manager** credentials for registering the operating system of the nodes that are not registered to the Red Hat Customer Portal:

```
apiVersion: v1
kind: Secret
metadata:
  name: subscription-manager
data:
  username: <base64_encoded_username>
  password: <base64_encoded_password>
```

6. Create the **Secret** CR:

```
$ oc create -f secret_subscription.yaml
```

7. Create a file on your workstation named **secret_registry.yaml** that contains the Red Hat registry credentials:

```
apiVersion: v1
kind: Secret
metadata:
  name: redhat-registry
data:
  username: <registry_username>
  password: <registry_password>
```

8. Create the **Secret** CR:

```
$ oc create -f secret_registry.yaml
```

9. Create a file on your workstation named **secret_libvirt.yaml** to define the libvirt secret:

```
apiVersion: v1
data:
 LibvirtPassword: <base64_password>
kind: Secret
metadata:
```

```
  name: libvirt-secret
  namespace: openstack
type: Opaque
```

- Replace **<base64_password>** with a base64 encoded string with maximum length 63 characters. Use the following command to generate a base64 encoded password:

  ```
  $ echo -n <password> | base64
  ```

10. Create the **Secret** CR:

    ```
    $ oc apply -f secret_libvirt.yaml -n openstack
    ```

11. Verify that the **Secret** CRs are created:

    ```
    $ oc describe secret dataplane-ansible-ssh-private-key-secret
    $ oc describe secret nova-migration-ssh-key
    $ oc describe secret subscription-manager
    $ oc describe secret redhat-registry
    $ oc describe secret libvirt-secret
    ```

10.3. CREATING A CUSTOM SR-IOV COMPUTE SERVICE

You must create a custom SR-IOV Compute service for NFV in a Red Hat OpenStack Services on OpenShift (RHOSO) environment. This service is an Ansible service that is executed on the data plane. This custom service performs the following tasks on the SR-IOV Compute nodes:

- Applies CPU pinning parameters.
- Performs PCI passthrough.

To create the SR-IOV custom service, you must perform these actions:

- Create a **ConfigMap** for CPU pinning that maps a CPU pinning configuration to a specified set of SR-IOV Compute nodes.
- Create a **ConfigMap** for PCI passthrough that maps a PCI passthrough configuration to a specified set of SR-IOV Compute nodes.
- Create the actual SR-IOV custom service that will implement the **configMaps** on your data plane.

Prerequisites

- You have the **oc** command line tool installed on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with **cluster-admin** privileges.

Procedure

1. Create a **ConfigMap** CR that defines configurations for CPU pinning and PCI passthrough, and save it to a YAML file on your workstation, for example, **pinning-passthrough.yaml**.
   Change the values (in boldface) as appropriate for your environment:

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: cpu-pinning-nova
data:
  25-cpu-pinning-nova.conf: |
    [DEFAULT]
    reserved_host_memory_mb = 4096
    [compute]
    cpu_shared_set = 0-3,24-27
    cpu_dedicated_set = 8-23,32-47
    [neutron]
    physnets = <network_name1>, <network_name2>
    [neutron_physnet_<network_name1>]
    numa_nodes = <number>
    [neutron_physnet_<network_name2>]
    numa_nodes = <number>
    [neutron_tunnel]
    numa_nodes = <number>
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: sriov-nova
data:
  26-sriov-nova.conf: |
    [libvirt]
    cpu_power_management=false
    [pci]
    passthrough_whitelist = {"address": "0000:05:00.2", "physical_network":"sriov-1",
"trusted":"true"}
    passthrough_whitelist = {"address": "0000:05:00.3", "physical_network":"sriov-2",
"trusted":"true"}
---
```

- **cpu_shared_set**: enter a comma-separated list or range of physical host CPU numbers used to provide vCPU inventory, determine the host CPUs that unpinned instances can be scheduled to, and determine the host CPUs that instance emulator threads should be offloaded to for instances configured with the share emulator thread policy.
- **cpu_dedicated_set**: enter a comma-separated list or range of physical host CPU numbers to which processes for pinned instance CPUs can be scheduled. For example, **4-12,^8,15** reserves cores from 4-12 and 15, excluding 8.
- **<network_name_n_>**: replace **<network_name1>** and **<network_name2>** with the names of the physical networks that your gateways are on. (This network is set in the neutron network **provider:*name** field.)
- **<number>**: replace **<number>** with the number of NUMA nodes you are using.
- **passthrough_whitelist**: specify valid NIC addresses and names for **"address"** and **"physical_network"**.

2. Create the **ConfigMap** object, using the **ConfigMap** CR file:

Example

Example

```
$ oc create -f sriov-pinning-passthru.yaml -n openstack
```

3. Create an **OpenStackDataPlaneService** CR that defines the SR-IOV custom service, and save it to a YAML file on your workstation, for example **nova-custom-sriov.yaml**:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: nova-custom-sriov
```

4. Add the **ConfigMap** CRs to the custom service, and specify the **Secret** CR for the cell that the node set that runs this service connects to:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: nova-custom-sriov
spec:
  label: dataplane-deployment-nova-custom-sriov
  configMaps:
    - cpu-pinning-nova
    - sriov-nova
  secrets:
    - nova-cell1-compute-config
    - nova-migration-ssh-key
```

5. Specify the Ansible commands to create the custom service, by referencing an Ansible playbook or by including the Ansible play in the **playbookContents** field:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: nova-custom-sriov
spec:
  label: dataplane-deployment-nova-custom-sriov
  playbook: osp.edpm.nova
  configMaps:
    - cpu-pinning-nova
    - sriov-nova
  secrets:
    - nova-cell1-compute-config
    - nova-migration-ssh-key
```

- **playbook**: identifies the default playbook available for your service.
  In this case, it is the Compute service (nova). To see the listing of default playbooks, see https://openstack-k8s-operators.github.io/edpm-ansible/playbooks.html.

6. Create the **custom-nova-sriov** service:

```
$ oc apply -f nova-custom-sriov.yaml -n openstack
```

7. Verify that the custom service is created:

```
$ oc get openstackdataplaneservice nova-custom-sriov -o yaml -n openstack
```

10.4. CREATING A CUSTOM OVS-DPDK COMPUTE SERVICE

You must create a custom OVS-DPDK Compute service for NFV in a Red Hat OpenStack Services on OpenShift (RHOSO) environment. This service is an Ansible service that is executed on the data plane. This custom service applies CPU pinning parameters on the OVS-DPDK Compute nodes.

To create the SR-IOV custom service, you must perform these actions:

- Create a **ConfigMap** for CPU pinning that maps a CPU pinning configuration to a specified set of OVS-DPDK Compute nodes.
- Create the actual OVS-DPDK custom service that will implement the **ConfigMap** on your data plane.

Prerequisites

- You have the **oc** command line tool installed on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with **cluster-admin** privileges.

Procedure

1. Create a **ConfigMap** CR that defines a configuration for CPU pinning, and save it to a YAML file on your workstation, for example, **dpdk-pinning.yaml**.
   Change the values (in boldface) as appropriate for your environment:

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: cpu-pinning-nova
data:
  25-cpu-pinning-nova.conf: |
    [DEFAULT]
    reserved_host_memory_mb = 4096
    [compute]
    cpu_shared_set = 0-3,24-27
    cpu_dedicated_set = 8-23,32-47
    [neutron]
    physnets = <network_name1>, <network_name2>
    [neutron_physnet_<network_name1>]
    numa_nodes = <number>
    [neutron_physnet_<network_name2>]
    numa_nodes = <number>
    [neutron_tunnel]
    numa_nodes = <number>
---
```

   - **cpu_shared_set**: enter a comma-separated list or range of physical host CPU numbers used to provide vCPU inventory, determine the host CPUs that unpinned instances can be scheduled to, and determine the host CPUs that instance emulator threads should be offloaded to for instances configured with the share emulator thread policy.

- **cpu_dedicated_set**: enter a comma-separated list or range of physical host CPU numbers to which processes for pinned instance CPUs can be scheduled. For example, **4-12,^8,15** reserves cores from 4-12 and 15, excluding 8.
- **<network_name_n_>**: replace **<network_name1>** and **<network_name2>** with the names of the physical networks that your gateways are on. (This network is set in the neutron network **provider:*name** field.)
- **<number>**: replace **<number>** with the number of NUMA nodes you are using.

2. Create the **ConfigMap** object, using the **ConfigMap** CR file:

   ### Example

   ```
   $ oc create -f dpdk-pinning.yaml -n openstack
   ```

3. Create an **OpenStackDataPlaneService** CR that defines the OVS-DPDK custom service, and save it to a YAML file on your workstation, for example **nova-custom-ovsdpdk.yaml**:

   ```
   apiVersion: dataplane.openstack.org/v1beta1
   kind: OpenStackDataPlaneService
   metadata:
     name: nova-custom-ovsdpdk
   ```

4. Add the **ConfigMap** CR to the custom service, and specify the **Secret** CR for the cell that the node set that runs this service connects to:

   ```
   apiVersion: dataplane.openstack.org/v1beta1
   kind: OpenStackDataPlaneService
   metadata:
     name: nova-custom-ovsdpdk
   spec:
     label: dataplane-deployment-nova-custom-ovsdpdk
     configMaps:
       - cpu-pinning-nova
     secrets:
       - nova-cell1-compute-config
       - nova-migration-ssh-key
   ```

5. Specify the Ansible commands to create the custom service, by referencing an Ansible playbook or by including the Ansible play in the **playbookContents** field:

   ```
   apiVersion: dataplane.openstack.org/v1beta1
   kind: OpenStackDataPlaneService
   metadata:
     name: nova-custom-ovsdpdk
     playbook: osp.edpm.nova
   spec:
     label: dataplane-deployment-nova-custom-ovsdpdk
     configMaps:
       - cpu-pinning-nova
     secrets:
       - nova-cell1-compute-config
       - nova-migration-ssh-key
   ```

- **playbook**: identifies the default playbook available for your service.
  In this case, it is the Compute service (nova). To see the listing of default playbooks, see https://openstack-k8s-operators.github.io/edpm-ansible/playbooks.html.

6. Create the **nova-custom-ovsdpdk** service:

```
$ oc apply -f nova-custom-ovsdpdk.yaml -n openstack
```

7. Verify that the custom service is created:

```
$ oc get openstackdataplaneservice nova-custom-ovsdpdk -o yaml -n openstack
```

10.5. CREATING A SET OF DATA PLANE NODES WITH PRE-PROVISIONED NODES

Define an **OpenStackDataPlaneNodeSet** custom resource (CR) for each logical grouping of pre-provisioned nodes in your data plane, for example, nodes grouped by hardware, location, or networking. You can define as many node sets as necessary for your deployment. Each node can be included in only one **OpenStackDataPlaneNodeSet** CR. Each node set can be connected to only one Compute cell. By default, node sets are connected to **cell1**. If you customize your control plane to include additional Compute cells, you must specify the cell to which the node set is connected. For more information on adding Compute cells, see Connecting an **OpenStackDataPlaneNodeSet** CR to a Compute cell in the *Customizing the Red Hat OpenStack Services on OpenShift deployment* guide.

You use the **nodeTemplate** field to configure the properties that all nodes in an **OpenStackDataPlaneNodeSet** CR share, and the **nodeTemplate.nodes** field for node-specific properties. Node-specific configurations override the inherited values from the **nodeTemplate**.

Procedure

1. Create a file on your workstation named **openstack_preprovisioned_node_set.yaml** to define the **OpenStackDataPlaneNodeSet** CR:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-data-plane (1)
  namespace: openstack
spec:
  env:
    - name: ANSIBLE_FORCE_COLOR
      value: "True"
```

(1) The **OpenStackDataPlaneNodeSet** CR name must be unique, must consist of lower case alphanumeric characters, **-** (hyphen) or **.** (period), must start and end with an alphanumeric character, and must have a maximum length of 20 characters. Update the name in this example to a name that reflects the nodes in the set.

2. Specify that the nodes in this set are pre-provisioned:

```
  preProvisioned: true
```

3. Add the SSH key secret that you created to enable Ansible to connect to the data plane nodes:

```
nodeTemplate:
  ansibleSSHPrivateKeySecret: <secret-key>
```

- Replace **<secret-key>** with the name of the SSH key **Secret** CR you created for this node set in Creating the data plane secrets, for example, **dataplane-ansible-ssh-private-key-secret**.

4. Create a Persistent Volume Claim (PVC) on your Red Hat OpenShift Container Platform (RHOCP) cluster to store logs. For information on how to create a PVC, see Understanding persistent storage in the RHOCP *Storage* guide.

5. Enable persistent logging for the data plane nodes:

```
nodeTemplate:
  ansibleSSHPrivateKeySecret: <secret-key>
  extraMounts:
    - extraVolType: Logs
      volumes:
      - name: ansible-logs
        persistentVolumeClaim:
          claimName: <pvc_name>
      mounts:
      - name: ansible-logs
        mountPath: "/runner/artifacts"
```

- Replace **<pvc_name>** with the name of the Persistent Volume Claim (PVC) storage on your RHOCP cluster.

6. Add the common configuration for the set of nodes in this group under the **nodeTemplate** section. Each node in this **OpenStackDataPlaneNodeSet** inherits this configuration. For information about the properties you can use to configure common node attributes, see **OpenStackDataPlaneNodeSet** CR **spec** properties.

7. Register the operating system of the nodes that are not registered to the Red Hat Customer Portal, and enable repositories for your nodes. The following steps demonstrate how to register your nodes to CDN. For details on how to register your nodes with Red Hat Satellite 6.13, see Managing Hosts.

   a. Create a **Secret** CR that contains the **subscription-manager** credentials:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: subscription-manager
   data:
     username: <base64_encoded_username>
     password: <base64_encoded_password>
   ```

   b. Create a **Secret** CR that contains the Red Hat registry credentials:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
   ```

```
  name: redhat-registry
data:
  username: <registry_username>
  password: <registry_password>
```

c. Specify the **Secret** CRs to use to source the usernames and passwords:

```
  nodeTemplate:
    ansible:
      ...
      ansibleVarsFrom:
        - prefix: subscription_manager_
          secretRef:
            name: subscription-manager
        - prefix: registry_
          secretRef:
            name: redhat-registry
      ansibleVars:
        edpm_bootstrap_command: |
          subscription-manager register --username {{ subscription_manager_username }} --
password {{ subscription_manager_password }}
          subscription-manager release --set=9.4
          subscription-manager repos --disable=*
          subscription-manager repos --enable=rhel-9-for-x86_64-baseos-eus-rpms --
enable=rhel-9-for-x86_64-appstream-eus-rpms --enable=rhel-9-for-x86_64-
highavailability-eus-rpms --enable=fast-datapath-for-rhel-9-x86_64-rpms --enable=rhoso-
18-beta-for-rhel-9-x86_64-rpms --enable=rhceph-7-tools-for-rhel-9-x86_64-rpms
          podman login -u {{ registry_username }} -p {{ registry_password }} registry.redhat.io
```

For a complete list of the Red Hat Customer Portal registration commands, see https://access.redhat.com/solutions/253273. For information about how to log into **registry.redhat.io**, see https://access.redhat.com/RegistryAuthentication#creating-registry-service-accounts-6.

8. Define each node in this node set:

```
  nodes:
    edpm-compute-0: (1)
      hostName: edpm-compute-0
      networks: (2)
        - name: ctlplane
          subnetName: subnet1
          defaultRoute: true
          fixedIP: 192.168.122.100 (3)
        - name: internalapi
          subnetName: subnet1
          fixedIP: 172.17.0.100
        - name: storage
          subnetName: subnet1
          fixedIP: 172.18.0.100
        - name: tenant
          subnetName: subnet1
          fixedIP: 172.19.0.100
      ansible:
        ansibleHost: 192.168.122.100
```

```
      ansibleUser: cloud-admin
      ansibleVars: (4)
        fqdn_internal_api: edpm-compute-0.example.com
  edpm-compute-1:
    hostName: edpm-compute-1
    networks:
      - name: ctlplane
        subnetName: subnet1
        defaultRoute: true
        fixedIP: 192.168.122.101
      - name: internalapi
        subnetName: subnet1
        fixedIP: 172.17.0.101
      - name: storage
        subnetName: subnet1
        fixedIP: 172.18.0.101
      - name: tenant
        subnetName: subnet1
        fixedIP: 172.19.0.101
    ansible:
      ansibleHost: 192.168.122.101
      ansibleUser: cloud-admin
      ansibleVars:
        fqdn_internal_api: edpm-compute-1.example.com
```

1. The node definition reference, for example, **edpm-compute-0**. Each node in the node set must have a node definition.
2. Defines the IPAM and the DNS records for the node.
3. Defines the predictable IP addresses for each network.
4. Node-specific Ansible variables that customize the node.

> **NOTE**
>
> - Nodes defined within the **nodes** section can configure the same Ansible variables that are configured in the **nodeTemplate** section. Where an Ansible variable is configured for both a specific node and within the **nodeTemplate** section, the node-specific values override those from the **nodeTemplate** section.
> - You do not need to replicate all the **nodeTemplate** Ansible variables for a node to override the default and set some node-specific values. You only need to configure the Ansible variables you want to override for the node.
> - Many **ansibleVars** include **edpm** in the name, which stands for "External Data Plane Management".

For more information, see:

- **OpenStackDataPlaneNodeSet** CR properties
- Network interface configuration options

- Example custom network interfaces for NFV

9. Save the **openstack_preprovisioned_node_set.yaml** definition file.

10. Create the data plane resources:

```
$ oc create -f openstack_preprovisioned_node_set.yaml -n openstack
```

11. Verify that the data plane resources have been created:

```
$ oc get openstackdataplanenodeset -n openstack
NAME            STATUS MESSAGE
openstack-data-plane  False  Deployment not started
```

For information about the meaning of the returned status, see Data plane conditions and states .

12. Verify that the **Secret** resource was created for the node set:

```
$ oc get secret | grep openstack-data-plane
dataplanenodeset-openstack-data-plane Opaque 1 3m50s
```

13. Verify the services were created:

```
$ oc get openstackdataplaneservice -n openstack
NAME                AGE
configure-network   6d7h
configure-os        6d6h
install-os          6d6h
run-os              6d6h
validate-network    6d6h
ovn                 6d6h
libvirt             6d6h
nova                6d6h
telemetry           6d6h
```

10.5.1. Example OpenStackDataPlaneNodeSet CR for pre-provisioned nodes

The following example **OpenStackDataPlaneNodeSet** CR creates a node set from pre-provisioned Compute nodes with some node-specific configuration. Update the name of the **OpenStackDataPlaneNodeSet** CR in this example to a name that reflects the nodes in the set. The **OpenStackDataPlaneNodeSet** CR name must be unique, must consist of lower case alphanumeric characters, **-** (hyphen) or **.** (period), must start and end with an alphanumeric character, and must have a maximum length of 20 characters. Update the name in this example to a name that reflects the nodes in the set.

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-data-plane
  namespace: openstack
spec:
  env: (1)
   - name: ANSIBLE_FORCE_COLOR
     value: "True"
```

```
services:
- bootstrap
- configure-network
- validate-network
- install-os
- configure-os
- ssh-known-hosts
- run-os
- reboot-os
- install-certs
- ovn
- neutron-metadata
- libvirt
- nova
- telemetry
networkAttachments:
  - ctlplane
preProvisioned: true 2
nodeTemplate: 3
  ansibleSSHPrivateKeySecret: dataplane-ansible-ssh-private-key-secret 4
  extraMounts:
    - extraVolType: Logs
      volumes:
      - name: ansible-logs
        persistentVolumeClaim:
          claimName: <pvc_name>
      mounts:
      - name: ansible-logs
        mountPath: "/runner/artifacts"
  managementNetwork: ctlplane
  ansible:
    ansibleUser: cloud-admin 5
    ansiblePort: 22
    ansibleVarsFrom:
      - prefix: subscription_manager_
        secretRef:
          name: subscription-manager
      - prefix: registry_
        secretRef:
          name: redhat-registry
    ansibleVars: 6
      edpm_bootstrap_command: |
        subscription-manager register --username {{ subscription_manager_username }} --password {{
subscription_manager_password }}
        subscription-manager release --set=9.4
        subscription-manager repos --disable=*
        subscription-manager repos --enable=rhel-9-for-x86_64-baseos-eus-rpms --enable=rhel-9-for-
x86_64-appstream-eus-rpms --enable=rhel-9-for-x86_64-highavailability-eus-rpms --enable=fast-
datapath-for-rhel-9-x86_64-rpms --enable=rhoso-18.0-for-rhel-9-x86_64-rpms --enable=rhceph-7-
tools-for-rhel-9-x86_64-rpms
        podman login -u {{ registry_username }} -p {{ registry_password }} registry.redhat.io
      edpm_bootstrap_release_version_package: []
      edpm_network_config_os_net_config_mappings:
        edpm-compute-1:
          nic1: 52:54:04:60:55:22 7
```

```
      neutron_physical_bridge_name: br-ex
      neutron_public_interface_name: eth0
      edpm_network_config_template: |
        ---
        {% set mtu_list = [ctlplane_mtu] %}
        {% for network in nodeset_networks %}
        {{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
        {%- endfor %}
        {% set min_viable_mtu = mtu_list | max %}
        network_config:
        - type: ovs_bridge
          name: {{ neutron_physical_bridge_name }}
          mtu: {{ min_viable_mtu }}
          use_dhcp: false
          dns_servers: {{ ctlplane_dns_nameservers }}
          domain: {{ dns_search_domains }}
          addresses:
          - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_cidr }}
          routes: {{ ctlplane_host_routes }}
          members:
          - type: interface
            name: nic1
            mtu: {{ min_viable_mtu }}
            # force the MAC address of the bridge to this interface
            primary: true
        {% for network in nodeset_networks %}
          - type: vlan
            mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
            vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
            addresses:
            - ip_netmask:
                {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
            routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
        {% endfor %}
  nodes:
    edpm-compute-0: (8)
      hostName: edpm-compute-0
      ansible:
        ansibleHost: 192.168.122.100
        ansibleUser: cloud-admin
        ansibleVars:
          fqdn_internal_api: edpm-compute-0.example.com
      networks:
      - name: ctlplane
        subnetName: subnet1
        defaultRoute: true
        fixedIP: 192.168.122.100
      - name: internalapi
        subnetName: subnet1
        fixedIP: 172.17.0.100
      - name: storage
        subnetName: subnet1
        fixedIP: 172.18.0.100
      - name: tenant
        subnetName: subnet1
```

```
      fixedIP: 172.19.0.100
  edpm-compute-1:
    hostName: edpm-compute-1
    ansible:
      ansibleHost: 192.168.122.101
      ansibleUser: cloud-admin
      ansibleVars:
        fqdn_internal_api: edpm-compute-1.example.com
    networks:
    - name: ctlplane
      subnetName: subnet1
      defaultRoute: true
      fixedIP: 192.168.122.101
    - name: internalapi
      subnetName: subnet1
      fixedIP: 172.17.0.101
    - name: storage
      subnetName: subnet1
      fixedIP: 172.18.0.101
    - name: tenant
      subnetName: subnet1
      fixedIP: 172.19.0.101
```

1. Optional: A list of environment variables to pass to the pod.
2. Specify that the nodes in this set are pre-provisioned.
3. The common configuration to apply to all nodes in this set of nodes.
4. The name of the secret that you created in Creating the data plane secrets.
5. The user associated with the secret you created in Creating the data plane secrets.
6. The Ansible variables that customize the set of nodes. For a list of Ansible variables that you can use, see https://openstack-k8s-operators.github.io/edpm-ansible/.
7. The MAC address assigned to the NIC to use for network configuration on the Compute node.
8. The node definition reference, for example, **edpm-compute-0**. Each node in the node set must have a node definition.

10.6. CREATING A SET OF DATA PLANE NODES WITH UNPROVISIONED NODES

Define an **OpenStackDataPlaneNodeSet** custom resource (CR) for each logical grouping of unprovisioned nodes in your data plane, for example, nodes grouped by hardware, location, or networking. You can define as many node sets as necessary for your deployment. Each node can be included in only one **OpenStackDataPlaneNodeSet** CR. Each node set can be connected to only one Compute cell. By default, node sets are connected to **cell1**. If you customize your control plane to include additional Compute cells, you must specify the cell to which the node set is connected. For more information on adding Compute cells, see Connecting an **OpenStackDataPlaneNodeSet** CR to a Compute cell in the *Customizing the Red Hat OpenStack Services on OpenShift deployment* guide.

You use the **nodeTemplate** field to configure the properties that all nodes in an **OpenStackDataPlaneNodeSet** CR share, and the **nodeTemplate.nodes** field for node-specific properties. Node-specific configurations override the inherited values from the **nodeTemplate**.

For more information about provisioning bare-metal nodes, see Provisioning bare-metal data plane nodes.

Prerequisites

- Cluster Baremetal Operator (CBO) is installed and configured for provisioning. For more information, see Provisioning bare-metal data plane nodes .
- A **BareMetalHost** CR is registered and inspected for each bare-metal data plane node. Each bare-metal node must be in the **Available** state after inspection. For more information about configuring bare-metal nodes, see Bare metal configuration in the Red Hat OpenShift Container Platform (RHOCP) *Postinstallation configuration* guide.

Procedure

1. Create a file on your workstation named **openstack_unprovisioned_node_set.yaml** to define the **OpenStackDataPlaneNodeSet** CR:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-data-plane (1)
  namespace: openstack
spec:
  tlsEnabled: true
  env:
    - name: ANSIBLE_FORCE_COLOR
      value: "True"
```

(1) The **OpenStackDataPlaneNodeSet** CR name must be unique, must consist of lower case alphanumeric characters, **-** (hyphen) or **.** (period), must start and end with an alphanumeric character, and must have a maximum length of 20 characters. Update the name in this example to a name that reflects the nodes in the set.

2. Define the **baremetalSetTemplate** field to describe the configuration of the bare-metal nodes:

```
  preProvisioned: false
  baremetalSetTemplate:
    deploymentSSHSecret: dataplane-ansible-ssh-private-key-secret
    bmhNamespace: <bmh_namespace>
    cloudUserName: <ansible_ssh_user>
    bmhLabelSelector:
      app: <bmh_label>
    ctlplaneInterface: <interface>
    dnsSearchDomains:
    - osptest.openstack.org
```

- Replace **<bmh_namespace>** with the namespace defined in the corresponding **BareMetalHost** CR for the node.
- Replace **<ansible_ssh_user>** with the username of the Ansible SSH user.

- Replace **<bmh_label>** with the label defined in the corresponding **BareMetalHost** CR for the node.
- Replace **<interface>** with the control plane interface the node connects to, for example, **enp6s0**.

3. The BMO manages **BareMetalHost** CRs in the **openshift-machine-api** namespace by default. You must update the **Provisioning** CR to watch all namespaces:

```
$ oc patch provisioning provisioning-configuration --type merge -p '{"spec":
{"watchAllNamespaces": true }}'
```

4. Add the SSH key secret that you created to enable Ansible to connect to the data plane nodes:

```
nodeTemplate:
  ansibleSSHPrivateKeySecret: <secret-key>
```

- Replace **<secret-key>** with the name of the SSH key **Secret** CR you created in Creating the data plane secrets, for example, **dataplane-ansible-ssh-private-key-secret**.

5. Create a Persistent Volume Claim (PVC) on your RHOCP cluster to store logs. For information about how to create a PVC, see Understanding persistent storage in the RHOCP *Storage* guide.

6. Enable persistent logging for the data plane nodes:

```
nodeTemplate:
  ansibleSSHPrivateKeySecret: <secret-key>
  extraMounts:
    - extraVolType: Logs
      volumes:
      - name: ansible-logs
        persistentVolumeClaim:
          claimName: <pvc_name>
      mounts:
      - name: ansible-logs
        mountPath: "/runner/artifacts"
```

- Replace **<pvc_name>** with the name of the Persistent Volume Claim (PVC) storage on your RHOCP cluster.

7. Add the common configuration for the set of nodes in this group under the **nodeTemplate** section. Each node in this **OpenStackDataPlaneNodeSet** inherits this configuration.
For more information, see:
    - **OpenStackDataPlaneNodeSet** CR properties
    - Network interface configuration options
    - Example custom network interfaces for NFV

8. Define each node in this node set:

```
nodes:
  edpm-compute-0: (1)
    hostName: edpm-compute-0
```

```
    networks: (2)
      - name: ctlplane
        subnetName: subnet1
        defaultRoute: true
        fixedIP: 192.168.122.100 (3)
      - name: internalapi
        subnetName: subnet1
      - name: storage
        subnetName: subnet1
      - name: tenant
        subnetName: subnet1
    ansible:
      ansibleHost: 192.168.122.100
      ansibleUser: cloud-admin
      ansibleVars: (4)
        fqdn_internal_api: edpm-compute-0.example.com
  edpm-compute-1:
    hostName: edpm-compute-1
    networks:
      - name: ctlplane
        subnetName: subnet1
        defaultRoute: true
        fixedIP: 192.168.122.101
      - name: internalapi
        subnetName: subnet1
      - name: storage
        subnetName: subnet1
      - name: tenant
        subnetName: subnet1
    ansible:
      ansibleHost: 192.168.122.101
      ansibleUser: cloud-admin
      ansibleVars:
        fqdn_internal_api: edpm-compute-1.example.com
```

1. The node definition reference, for example, **edpm-compute-0**. Each node in the node set must have a node definition.
2. Defines the IPAM and the DNS records for the node.
3. Defines the predictable IP addresses for each network.
4. Node-specific Ansible variables that customize the node.

NOTE

- Nodes defined within the **nodes** section can configure the same Ansible variables that are configured in the **nodeTemplate** section. Where an Ansible variable is configured for both a specific node and within the **nodeTemplate** section, the node-specific values override those from the **nodeTemplate** section.
- You do not need to replicate all the **nodeTemplate** Ansible variables for a node to override the default and set some node-specific values. You only need to configure the Ansible variables you want to override for the node.
- Many **ansibleVars** include **edpm** in the name, which stands for "External Data Plane Management".

For information about the properties you can use to configure node attributes, see **OpenStackDataPlaneNodeSet** CR properties.

9. Save the **openstack_unprovisioned_node_set.yaml** definition file.
10. Create the data plane resources:

```
$ oc create -f openstack_unprovisioned_node_set.yaml -n openstack
```

11. Verify that the data plane resources have been created:

```
$ oc get openstackdataplanenodeset -n openstack
NAME            STATUS MESSAGE
openstack-data-plane  False  Deployment not started
```

For information on the meaning of the returned status, see Data plane conditions and states.

12. Verify that the **Secret** resource was created for the node set:

```
$ oc get secret -n openstack | grep openstack-data-plane
dataplanenodeset-openstack-data-plane Opaque 1 3m50s
```

13. Verify the services were created:

```
$ oc get openstackdataplaneservice -n openstack
NAME              AGE
configure-network   6d7h
configure-os        6d6h
install-os         6d6h
run-os             6d6h
validate-network    6d6h
ovn               6d6h
libvirt            6d6h
nova              6d6h
telemetry          6d6h
```

10.6.1. Example OpenStackDataPlaneNodeSet CR for unprovisioned nodes

The following example **OpenStackDataPlaneNodeSet** CR creates a node set from unprovisioned

Compute nodes with some node-specific configuration. The unprovisioned Compute nodes are provisioned when the node set is created. Update the name of the **OpenStackDataPlaneNodeSet** CR in this example to a name that reflects the nodes in the set. The **OpenStackDataPlaneNodeSet** CR name must be unique, must consist of lower case alphanumeric characters, **-** (hyphen) or **.** (period), must start and end with an alphanumeric character, and must have a maximum length of 20 characters. Update the name in this example to a name that reflects the nodes in the set.

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-data-plane
  namespace: openstack
spec:
  env: 1
    - name: ANSIBLE_FORCE_COLOR
      value: "True"
  services:
  - bootstrap
  - configure-network
  - validate-network
  - install-os
  - configure-os
  - ssh-known-hosts
  - run-os
  - reboot-os
  - install-certs
  - ovn
  - neutron-metadata
  - libvirt
  - nova
  - telemetry
  networkAttachments:
    - ctlplane
  preProvisioned: false 2
  baremetalSetTemplate: 3
    deploymentSSHSecret: dataplane-ansible-ssh-private-key-secret
    bmhNamespace: openshift-machine-api 4
    cloudUserName: <ansible_ssh_user>
    bmhLabelSelector:
      app: openstack 5
    ctlplaneInterface: enp1s0
    dnsSearchDomains:
      - osptest.openstack.org
  nodeTemplate: 6
    ansibleSSHPrivateKeySecret: dataplane-ansible-ssh-private-key-secret 7
    extraMounts:
      - extraVolType: Logs
        volumes:
        - name: ansible-logs
          persistentVolumeClaim:
            claimName: <pvc_name>
        mounts:
        - name: ansible-logs
          mountPath: "/runner/artifacts"
    managementNetwork: ctlplane
```

```
ansible:
  ansibleUser: cloud-admin # (8)
  ansiblePort: 22
  ansibleVarsFrom:
    - prefix: subscription_manager_
      secretRef:
        name: subscription-manager
    - prefix: registry_
      secretRef:
        name: redhat-registry
  ansibleVars: # (9)
    edpm_bootstrap_command: |
      subscription-manager register --username {{ subscription_manager_username }} --password {{
subscription_manager_password }}
      subscription-manager release --set=9.4
      subscription-manager repos --disable=*
      subscription-manager repos --enable=rhel-9-for-x86_64-baseos-eus-rpms --enable=rhel-9-for-
x86_64-appstream-eus-rpms --enable=rhel-9-for-x86_64-highavailability-eus-rpms --enable=fast-
datapath-for-rhel-9-x86_64-rpms --enable=rhoso-18.0-for-rhel-9-x86_64-rpms --enable=rhceph-7-
tools-for-rhel-9-x86_64-rpms
      podman login -u {{ registry_username }} -p {{ registry_password }} registry.redhat.io
    edpm_bootstrap_release_version_package: []
    edpm_network_config_os_net_config_mappings:
      edpm-compute-0:
        nic1: 52:54:04:60:55:22 # (10)
      edpm-compute-1:
        nic1: 52:54:04:60:55:22
    neutron_physical_bridge_name: br-ex
    neutron_public_interface_name: eth0
    edpm_network_config_template: |
      ---
      {% set mtu_list = [ctlplane_mtu] %}
      {% for network in nodeset_networks %}
      {{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
      {%- endfor %}
      {% set min_viable_mtu = mtu_list | max %}
      network_config:
      - type: ovs_bridge
        name: {{ neutron_physical_bridge_name }}
        mtu: {{ min_viable_mtu }}
        use_dhcp: false
        dns_servers: {{ ctlplane_dns_nameservers }}
        domain: {{ dns_search_domains }}
        addresses:
        - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_cidr }}
        routes: {{ ctlplane_host_routes }}
        members:
        - type: interface
          name: nic1
          mtu: {{ min_viable_mtu }}
          # force the MAC address of the bridge to this interface
          primary: true
      {% for network in nodeset_networks %}
        - type: vlan
          mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
          vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
```

```
            addresses:
            - ip_netmask:
              {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
            routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
          {% endfor %}
  nodes:
    edpm-compute-0: 11
      hostName: edpm-compute-0
      ansible:
        ansibleHost: 192.168.122.100
        ansibleUser: cloud-admin
        ansibleVars:
          fqdn_internal_api: edpm-compute-0.example.com
      networks: 12
      - name: ctlplane
        subnetName: subnet1
        defaultRoute: true
        fixedIP: 192.168.122.100 13
      - name: internalapi
        subnetName: subnet1
      - name: storage
        subnetName: subnet1
      - name: tenant
        subnetName: subnet1
    edpm-compute-1:
      hostName: edpm-compute-1
      ansible: 14
        ansibleHost: 192.168.122.101
        ansibleUser: cloud-admin
        ansibleVars:
          fqdn_internal_api: edpm-compute-1.example.com
      networks:
      - name: ctlplane
        subnetName: subnet1
        defaultRoute: true
        fixedIP: 192.168.122.101
      - name: internalapi
        subnetName: subnet1
      - name: storage
        subnetName: subnet1
      - name: tenant
        subnetName: subnet1
```

1. Optional: A list of environment variables to pass to the pod.
2. Specify that the nodes in this set are unprovisioned and must be provisioned when creating the resource.
3. Configure the bare-metal template for bare-metal nodes that must be provisioned when creating the resource.
4. The namespace defined in the corresponding **BareMetalHost** CR for the node.
5. The label defined in the corresponding **BareMetalHost** CR for the node.

6 The common configuration to apply to all nodes in this set of nodes.

7 The name of the secret that you created in Creating the data plane secrets .

8 The user associated with the secret you created in Creating the data plane secrets .

9 The Ansible variables that customize the set of nodes. For a list of Ansible variables that you can use, see https://openstack-k8s-operators.github.io/edpm-ansible/.

10 The MAC address assigned to the NIC to use for network configuration on the Compute node.

11 The node definition reference, for example, **edpm-compute-0**. Each node in the node set must have a node definition.

12 Defines the IPAM and the DNS records for the node.

13 14 Defines the predictable IP addresses for each network.

10.6.2. Provisioning bare-metal data plane nodes

Provisioning bare-metal nodes on the data plane is supported with the Red Hat OpenShift Container Platform (RHOCP) Cluster Baremetal Operator (CBO). The CBO deploys the components required to provision bare-metal nodes within the RHOCP cluster, including the Bare Metal Operator (BMO) and Ironic containers.

The BMO manages the available hosts on clusters and performs the following operations:

- Inspects node hardware details and reports them to the corresponding **BareMetalHost** CR. This includes information about CPUs, RAM, disks, and NICs.
- Provisions nodes with a specific image.
- Cleans node disk contents before and after provisioning.

The availability of the CBO depends on which of the following installation methods was used for the RHOCP cluster:

Assisted Installer

You can enable CBO on clusters installed with the Assisted Installer, and you can manually add the provisioning network to the Assisted Installer cluster after installation.

Installer-provisioned infrastructure

CBO is enabled by default on RHOCP clusters that are installed with the bare-metal installer-provisioned infrastructure. You can configure installer-provisioned clusters with a provisioning network to enable both virtual media and network boot installations. Alternatively, you can configure an installer-provisioned cluster without a provisioning network so that only virtual media provisioning is available. For more information about installer-provisioned clusters on bare metal, see Deploying installer-provisioned clusters on bare metal.

User-provisioned infrastructure

You can activate CBO on RHOCP clusters installed with user-provisioned infrastructure by creating a Provisioning CR. You cannot add a provisioning network to a user-provisioned cluster. For more information about how to create a Provisioning CR, see Scaling a user-provisioned cluster with the Bare Metal Operator.

10.7. OPENSTACKDATAPLANENODESET CR SPEC PROPERTIES

The following sections detail the **OpenStackDataPlaneNodeSet** CR **spec** properties you can configure.

10.7.1. nodeTemplate

Defines the common attributes for the nodes in this **OpenStackDataPlaneNodeSet**. You can override these common attributes in the definition for each individual node.

Table 10.1. **nodeTemplate** properties

Field	Description
ansibleSSHPrivateKeySecret	Name of the private SSH key secret that contains the private SSH key for connecting to nodes. Secret name format: Secret.data.ssh-privatekey For more information, see Creating an SSH authentication secret. Default: **dataplane-ansible-ssh-private-key-secret**
managementNetwork	Name of the network to use for management (SSH/Ansible). Default: **ctlplane**
networks	Network definitions for the **OpenStackDataPlaneNodeSet**.
ansible	Ansible configuration options. For more information, see **ansible** properties.
extraMounts	The files to mount into an Ansible Execution Pod.
userData	UserData configuration for the **OpenStackDataPlaneNodeSet**.
networkData	NetworkData configuration for the **OpenStackDataPlaneNodeSet**.

10.7.2. nodes

Defines the node names and node-specific attributes for the nodes in this **OpenStackDataPlaneNodeSet**. Overrides the common attributes defined in the **nodeTemplate**.

Table 10.2. **nodes** properties

Field	Description
ansible	Ansible configuration options. For more information, see **ansible** properties.

Field	Description
extraMounts	The files to mount into an Ansible Execution Pod.
hostName	The node name.
managementNetwork	Name of the network to use for management (SSH/Ansible).
networkData	NetworkData configuration for the node.
networks	Instance networks.
preprovisioningNetworkDataName	NetworkData secret name in the local namespace for pre-provisioning.
userData	Node-specific user data.

10.7.3. ansible

Defines the group of Ansible configuration options.

Table 10.3. ansible properties

Field	Description
ansibleUser	The user associated with the secret you created in Creating the data plane secrets. Default: **rhel-user**
ansibleHost	SSH host for the Ansible connection.
ansiblePort	SSH port for the Ansible connection.
ansibleVars	The Ansible variables that customize the set of nodes. You can use this property to configure any custom Ansible variable, including the Ansible variables available for each **edpm-ansible** role. For a complete list of Ansible variables by role, see the **edpm-ansible** documentation. NOTE The **ansibleVars** parameters that you can configure for an **OpenStackDataPlaneNodeSet** CR are determined by the services defined for the **OpenStackDataPlaneNodeSet**. The **OpenStackDataPlaneService** CRs call the Ansible playbooks from the **edpm-ansible** playbook collection, which include the roles that are executed as part of the data plane service.

Field	Description
ansibleVarsFrom	A list of sources to populate Ansible variables from. Values defined by an **AnsibleVars** with a duplicate key take precedence. For more information, see **ansibleVarsFrom** properties.

10.7.4. ansibleVarsFrom

Defines the list of sources to populate Ansible variables from.

Table 10.4. ansibleVarsFrom properties

Field	Description
prefix	An optional identifier to prepend to each key in the **ConfigMap**. Must be a C_IDENTIFIER.
configMapRef	The **ConfigMap** CR to select the **ansibleVars** from.
secretRef	The **Secret** CR to select the **ansibleVars** from.

10.8. NETWORK INTERFACE CONFIGURATION OPTIONS

Use the following tables to understand the available options for configuring network interfaces for Red Hat OpenStack Services on OpenShift (RHOSO) environments.

- interface
- vlan
- ovs_bridge
- Network interface bonding
- ovs_bond
- LACP with OVS bonding modes
- linux_bond
- routes

10.8.1. interface

Defines a single network interface. The network interface **name** uses either the actual interface name (**eth0**, **eth1**, **enp0s25**) or a set of numbered interfaces (**nic1**, **nic2**, **nic3**). The network interfaces of hosts within a role do not have to be exactly the same when you use numbered interfaces such as **nic1** and **nic2**, instead of named interfaces such as **eth0** and **eno2**. For example, one host might have interfaces **em1** and **em2**, while another has **eno1** and **eno2**, but you can refer to the NICs of both hosts as **nic1** and **nic2**.

The order of numbered interfaces corresponds to the order of named network interface types:

- **ethX** interfaces, such as **eth0**, **eth1**, and so on.
  Names appear in this format when consistent device naming is turned off in **udev**.
- **enoX** and **emX** interfaces, such as **eno0**, **eno1**, **em0**, **em1**, and so on.
  These are usually on-board interfaces.
- **enX** and any other interfaces, sorted alpha numerically, such as **enp3s0**, **enp3s1**, **ens3**, and so on.
  These are usually add-on interfaces.

The numbered NIC scheme includes only live interfaces, for example, if the interfaces have a cable attached to the switch. If you have some hosts with four interfaces and some with six interfaces, use **nic1** to **nic4** and attach only four cables on each host.

Table 10.5. interface options

Option	Default	Description
name		Name of the interface. The network interface **name** uses either the actual interface name (**eth0**, **eth1**, **enp0s25**) or a set of numbered interfaces (**nic1**, **nic2**, **nic3**).
use_dhcp	False	Use DHCP to get an IP address.
use_dhcpv6	False	Use DHCP to get a v6 IP address.
addresses		A list of IP addresses assigned to the interface.
routes		A list of routes assigned to the interface. For more information, see Section 10.8.7, “routes”.
mtu	1500	The maximum transmission unit (MTU) of the connection.
primary	False	Defines the interface as the primary interface. Required only when the **interface** is a member of a bond.
persist_mapping	False	Write the device alias configuration instead of the system names.

Option	Default	Description
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the interface.
ethtool_opts		Set this option to **"rx-flow-hash udp4 sdfn"** to improve throughput when you use VXLAN on certain NICs.

Example

```
...
        edpm_network_config_template: |
          ---
          {% set mtu_list = [ctlplane_mtu] %}
          {% for network in nodeset_networks %}
          {{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
          {%- endfor %}
          {% set min_viable_mtu = mtu_list | max %}
          network_config:
          - type: interface
            name: nic2
            ...
```

10.8.2. vlan

Defines a VLAN. Use the VLAN ID and subnet passed from the **parameters** section.

Table 10.6. **vlan** options

Option	Default	Description
vlan_id		The VLAN ID.
device		The parent device to attach the VLAN. Use this parameter when the VLAN is not a member of an OVS bridge. For example, use this parameter to attach the VLAN to a bonded interface device.
use_dhcp	False	Use DHCP to get an IP address.
use_dhcpv6	False	Use DHCP to get a v6 IP address.

Option	Default	Description
addresses		A list of IP addresses assigned to the VLAN.
routes		A list of routes assigned to the VLAN. For more information, see Section 10.8.7, “routes”.
mtu	1500	The maximum transmission unit (MTU) of the connection.
primary	False	Defines the VLAN as the primary interface.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the VLAN.

Example

```
...
        edpm_network_config_template: |
          ---
          {% set mtu_list = [ctlplane_mtu] %}
          {% for network in nodeset_networks %}
          {{ mtu_list.append(lookup(vars, networks_lower[network] ~ _mtu)) }}
          {%- endfor %}
          {% set min_viable_mtu = mtu_list | max %}
          network_config:
          ...
            members:
            - type: vlan
              device: nic{{ loop.index + 1 }}
              mtu: {{ lookup(vars, networks_lower[network] ~ _mtu) }}
              vlan_id: {{ lookup(vars, networks_lower[network] ~ _vlan_id) }}
              addresses:
              - ip_netmask:
                  {{ lookup(vars, networks_lower[network] ~ _ip) }}/{{ lookup(vars, networks_lower[network]
~ _cidr) }}
              routes: {{ lookup(vars, networks_lower[network] ~ _host_routes) }}
...
```

10.8.3. ovs_bridge

Defines a bridge in Open vSwitch (OVS), which connects multiple **interface**, **ovs_bond**, and **vlan** objects together.

The network interface type, **ovs_bridge**, takes a parameter **name**.

IMPORTANT

The **ovs_bridge** interface is not recommended for control plane network traffic. The OVS bridge connects to the Networking service (neutron) server to obtain configuration data. If the OpenStack control traffic, typically the Control Plane and Internal API networks, is placed on an OVS bridge, then connectivity to the neutron server is lost whenever you upgrade OVS, or the OVS bridge is restarted by the admin user or process. This causes some downtime. If downtime is not acceptable in these circumstances, then you must place the Control group networks on a separate interface or bond rather than on an OVS bridge:

- You can achieve a minimal setting when you put the Internal API network on a VLAN on the provisioning interface and the OVS bridge on a second interface.
- To implement bonding, you need at least two bonds (four network interfaces). Place the control group on a Linux bond. If the switch does not support LACP fallback to a single interface for PXE boot, then this solution requires at least five NICs.

NOTE

If you have multiple bridges, you must use distinct bridge names other than accepting the default name of **bridge_name**. If you do not use distinct names, then during the converge phase, two network bonds are placed on the same bridge.

Table 10.7. ovs_bridge options

Option	Default	Description
name		Name of the bridge.
use_dhcp	False	Use DHCP to get an IP address.
use_dhcpv6	False	Use DHCP to get a v6 IP address.
addresses		A list of IP addresses assigned to the bridge.
routes		A list of routes assigned to the bridge. For more information, see Section 10.8.7, “routes”.
mtu	1500	The maximum transmission unit (MTU) of the connection.

Option	Default	Description
members		A sequence of interface, VLAN, and bond objects that you want to use in the bridge.
ovs_options		A set of options to pass to OVS when creating the bridge.
ovs_extra		A set of options to to set as the OVS_EXTRA parameter in the network configuration file of the bridge.
defroute	True	Use a default route provided by the DHCP service. Only applies when you enable **use_dhcp** or **use_dhcpv6**.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the bridge.

Example

```
...
    edpm_network_config_template: |
     ---
     {% set mtu_list = [ctlplane_mtu] %}
     {% for network in nodeset_networks %}
     {{ mtu_list.append(lookup(vars, networks_lower[network] ~ _mtu)) }}
     {%- endfor %}
     {% set min_viable_mtu = mtu_list | max %}
     network_config:
     - type: ovs_bridge
       name: br-bond
       dns_servers: {{ ctlplane_dns_nameservers }}
       domain: {{ dns_search_domains }}
       members:
       - type: ovs_bond
         name: bond1
         mtu: {{ min_viable_mtu }}
         ovs_options: {{ bound_interface_ovs_options }}
         members:
         - type: interface
           name: nic2
```

```
    mtu: {{ min_viable_mtu }}
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu }}
    ...
```

10.8.4. Network interface bonding

You can bundle multiple physical NICs together to form a single logical channel known as a bond. You can configure bonds to provide redundancy for high availability systems or increased throughput.

Red Hat OpenStack Platform supports Open vSwitch (OVS) kernel bonds, OVS-DPDK bonds, and Linux kernel bonds.

Table 10.8. Supported interface bonding types

Bond type	Type value	Allowed bridge types	Allowed members
OVS kernel bonds	**ovs_bond**	**ovs_bridge**	**interface**
OVS-DPDK bonds	**ovs_dpdk_bond**	**ovs_user_bridge**	**ovs_dpdk_port**
Linux kernel bonds	**linux_bond**	**ovs_bridge**	**interface**

IMPORTANT

Do not combine **ovs_bridge** and **ovs_user_bridge** on the same node.

10.8.4.1. ovs_bond

Defines a bond in Open vSwitch (OVS) to join two or more **interfaces** together. This helps with redundancy and increases bandwidth.

Table 10.9. ovs_bond options

Option	Default	Description
name		Name of the bond.
use_dhcp	False	Use DHCP to get an IP address.
use_dhcpv6	False	Use DHCP to get a v6 IP address.
addresses		A list of IP addresses assigned to the bond.
routes		A list of routes assigned to the bond. For more information, see Section 10.8.7, “routes”.

Option	Default	Description
mtu	1500	The maximum transmission unit (MTU) of the connection.
primary	False	Defines the interface as the primary interface.
members		A sequence of interface objects that you want to use in the bond.
ovs_options		A set of options to pass to OVS when creating the bond. For more information, see Table 10.10, "**ovs_options** parameters for OVS bonds".
ovs_extra		A set of options to set as the OVS_EXTRA parameter in the network configuration file of the bond.
defroute	True	Use a default route provided by the DHCP service. Only applies when you enable **use_dhcp** or **use_dhcpv6**.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the bond.

Table 10.10. **ovs_options** parameters for OVS bonds

ovs_option	Description

ovs_option	Description
bond_mode=balance-slb	Source load balancing (slb) balances flows based on source MAC address and output VLAN, with periodic rebalancing as traffic patterns change. When you configure a bond with the **balance-slb** bonding option, there is no configuration required on the remote switch. The Networking service (neutron) assigns each source MAC and VLAN pair to a link and transmits all packets from that MAC and VLAN through that link. A simple hashing algorithm based on source MAC address and VLAN number is used, with periodic rebalancing as traffic patterns change. The **balance-slb** mode is similar to mode 2 bonds used by the Linux bonding driver. You can use this mode to provide load balancing even when the switch is not configured to use LACP.
bond_mode=active-backup	When you configure a bond using **active-backup** bond mode, the Networking service keeps one NIC in standby. The standby NIC resumes network operations when the active connection fails. Only one MAC address is presented to the physical switch. This mode does not require switch configuration, and works when the links are connected to separate switches. This mode does not provide load balancing.
lacp=[active | passive | off]	Controls the Link Aggregation Control Protocol (LACP) behavior. Only certain switches support LACP. If your switch does not support LACP, use **bond_mode=balance-slb** or **bond_mode=active-backup**.
other-config:lacp-fallback-ab=true	Set active-backup as the bond mode if LACP fails.
other_config:lacp-time=[fast | slow]	Set the LACP heartbeat to one second (fast) or 30 seconds (slow). The default is slow.
other_config:bond-detect-mode=[miimon | carrier]	Set the link detection to use miimon heartbeats (miimon) or monitor carrier (carrier). The default is carrier.
other_config:bond-miimon-interval=100	If using miimon, set the heartbeat interval (milliseconds).
bond_updelay=1000	Set the interval (milliseconds) that a link must be up to be activated to prevent flapping.

ovs_option	Description
other_config:bond-rebalance-interval=10000	Set the interval (milliseconds) that flows are rebalancing between bond members. Set this value to zero to disable flow rebalancing between bond members.

Example - OVS bond

```
...
        edpm_network_config_template: |
          ---
          {% set mtu_list = [ctlplane_mtu] %}
          {% for network in nodeset_networks %}
          {{ mtu_list.append(lookup(vars, networks_lower[network] ~ _mtu)) }}
          {%- endfor %}
          {% set min_viable_mtu = mtu_list | max %}
          network_config:
          ...
            members:
              - type: ovs_bond
                name: bond1
                mtu: {{ min_viable_mtu }}
                ovs_options: {{ bond_interface_ovs_options }}
                members:
                - type: interface
                  name: nic2
                  mtu: {{ min_viable_mtu }}
                  primary: true
                - type: interface
                  name: nic3
                  mtu: {{ min_viable_mtu }}
```

Example - OVS DPDK bond

In this example, a bond is created as part of an OVS user space bridge:

```
        edpm_network_config_template: |
          ---
          {% set mtu_list = [ctlplane_mtu] %}
          {% for network in nodeset_networks %}
          {{ mtu_list.append(lookup(vars, networks_lower[network] ~ _mtu)) }}
          {%- endfor %}
          {% set min_viable_mtu = mtu_list | max %}
          network_config:
          ...
            members:
            - type: ovs_user_bridge
              name: br-dpdk0
              members:
              - type: ovs_dpdk_bond
                name: dpdkbond0
                rx_queue: {{ num_dpdk_interface_rx_queues }}
                members:
```

```
        - type: ovs_dpdk_port
          name: dpdk0
          members:
          - type: interface
            name: nic4
        - type: ovs_dpdk_port
          name: dpdk1
          members:
          - type: interface
            name: nic5
```

10.8.5. LACP with OVS bonding modes

You can use Open vSwitch (OVS) bonds with the optional Link Aggregation Control Protocol (LACP). LACP is a negotiation protocol that creates a dynamic bond for load balancing and fault tolerance.

Use the following table to understand support compatibility for OVS kernel and OVS-DPDK bonded interfaces in conjunction with LACP options.

IMPORTANT

On control and storage networks, Red Hat recommends that you use Linux bonds with VLAN and LACP, because OVS bonds carry the potential for control plane disruption that can occur when OVS or the neutron agent is restarted for updates, hot fixes, and other events. The Linux bond-LACP-VLAN configuration provides NIC management without the OVS disruption potential.

Table 10.11. LACP options for OVS kernel and OVS-DPDK bond modes

Objective	OVS bond mode	Compatible LACP options	Notes
High availability (active-passive)	**active-backup**	**active**, **passive**, or **off**	
Increased throughput (active-active)	**balance-slb**	**active**, **passive**, or **off**	• Performance is affected by extra parsing per packet. • There is a potential for vhost-user lock contention.

	balance-tcp	**active** or **passive**	• As with balance-slb, performance is affected by extra parsing per packet and there is a potential for vhost-user lock contention. • LACP must be configured and enabled. • Set **lb-output-action=true**. For example: `ovs-vsctl set port <bond port> other_config:lb-output-action=true`

10.8.6. linux_bond

Defines a Linux bond that joins two or more **interfaces** together. This helps with redundancy and increases bandwidth. Ensure that you include the kernel-based bonding options in the **bonding_options** parameter.

Table 10.12. linux_bond options

Option	Default	Description
name		Name of the bond.
use_dhcp	False	Use DHCP to get an IP address.
use_dhcpv6	False	Use DHCP to get a v6 IP address.
addresses		A list of IP addresses assigned to the bond.
routes		A list of routes assigned to the bond. See Section 10.8.7, “routes”.
mtu	1500	The maximum transmission unit (MTU) of the connection.

Option	Default	Description
members		A sequence of interface objects that you want to use in the bond.
bonding_options		A set of options when creating the bond. See **bonding_options** parameters for Linux bonds.
defroute	True	Use a default route provided by the DHCP service. Only applies when you enable **use_dhcp** or **use_dhcpv6**.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the bond.

bonding_options parameters for Linux bonds

The **bonding_options** parameter sets the specific bonding options for the Linux bond. See the Linux bonding examples that follow this table:

bonding_options	Description
mode	Sets the bonding mode, which in the example is **802.3ad** or LACP mode. For more information about Linux bonding modes, see Configuring a network bond in Red Hat Enterprise Linux 9,*Configuring and managing networking*.
lacp_rate	Defines whether LACP packets are sent every 1 second, or every 30 seconds.
updelay	Defines the minimum amount of time that an interface must be active before it is used for traffic. This minimum configuration helps to mitigate port flapping outages.
miimon	The interval in milliseconds that is used for monitoring the port state using the MIIMON functionality of the driver.

Example - Linux bond

```
...
        edpm_network_config_template: |
          ---
          {% set mtu_list = [ctlplane_mtu] %}
          {% for network in nodeset_networks %}
          {{ mtu_list.append(lookup(vars, networks_lower[network] ~ _mtu)) }}
          {%- endfor %}
          {% set min_viable_mtu = mtu_list | max %}
          network_config:
          - type: linux_bond
            name: bond1
            mtu: {{ min_viable_mtu }}
            bonding_options: "mode=802.3ad lacp_rate=fast updelay=1000 miimon=100
xmit_hash_policy=layer3+4"
            members:
              type: interface
              name: ens1f0
              mtu: {{ min_viable_mtu }}
              primary: true
            type: interface
              name: ens1f1
              mtu: {{ min_viable_mtu }}
              ...
```

Example - Linux bond: bonding two interfaces

```
...
        edpm_network_config_template: |
          ---
          {% set mtu_list = [ctlplane_mtu] %}
          {% for network in nodeset_networks %}
          {{ mtu_list.append(lookup(vars, networks_lower[network] ~ _mtu)) }}
          {%- endfor %}
          {% set min_viable_mtu = mtu_list | max %}
          network_config:
          - type: linux_bond
            name: bond1
            members:
            - type: interface
              name: nic2
            - type: interface
              name: nic3
            bonding_options: "mode=802.3ad lacp_rate=[fast|slow] updelay=1000 miimon=100"
            ...
```

Example - Linux bond set to **active-backup** mode with one VLAN

```
....
        edpm_network_config_template: |
          ---
          {% set mtu_list = [ctlplane_mtu] %}
          {% for network in nodeset_networks %}
```

```
        {{ mtu_list.append(lookup(vars, networks_lower[network] ~ _mtu)) }}
        {%- endfor %}
        {% set min_viable_mtu = mtu_list | max %}
        network_config:
        - type: linux_bond
          name: bond_api
          bonding_options: "mode=active-backup"
          use_dhcp: false
          dns_servers:
            get_param: DnsServers
          members:
          - type: interface
            name: nic3
            primary: true
          - type: interface
            name: nic4

          - type: vlan
            vlan_id:
              get_param: InternalApiNetworkVlanID
            device: bond_api
            addresses:
            - ip_netmask:
                get_param: InternalApiIpSubnet
```

Example - Linux bond on OVS bridge

In this example, the bond is set to **802.3ad** with LACP mode and one VLAN:

```
...
       edpm_network_config_template: |
        ---
        {% set mtu_list = [ctlplane_mtu] %}
        {% for network in nodeset_networks %}
        {{ mtu_list.append(lookup(vars, networks_lower[network] ~ _mtu)) }}
        {%- endfor %}
        {% set min_viable_mtu = mtu_list | max %}
        network_config:
        -  type: ovs_bridge
            name: br-tenant
            use_dhcp: false
            mtu: 9000
            members:
              - type: linux_bond
                name: bond_tenant
                bonding_options: "mode=802.3ad updelay=1000 miimon=100"
                use_dhcp: false
                dns_servers:
                  get_param: DnsServers
                members:
                - type: interface
                  name: p1p1
                  primary: true
                - type: interface
                  name: p1p2
              - type: vlan
```

```
          device: bond_tenant
          vlan_id: {get_param: TenantNetworkVlanID}
          addresses:
            - ip_netmask: {get_param: TenantIpSubnet}
            ...
```

10.8.7. routes

Defines a list of routes to apply to a network interface, VLAN, bridge, or bond.

Table 10.13. routes options

Option	Default	Description
ip_netmask	None	IP and netmask of the destination network.
default	False	Sets this route to a default route. Equivalent to setting **ip_netmask: 0.0.0.0/0**.
next_hop	None	The IP address of the router used to reach the destination network.

Example - routes

```
...
    edpm_network_config_template: |
      ---
      {% set mtu_list = [ctlplane_mtu] %}
      {% for network in nodeset_networks %}
      {{ mtu_list.append(lookup(vars, networks_lower[network] ~ _mtu)) }}
      {%- endfor %}
      {% set min_viable_mtu = mtu_list | max %}
      network_config:
      - type: ovs_bridge
        name: br-tenant
        ...
        routes: {{ [ctlplane_host_routes] | flatten | unique }}
        ...
```

Additional resources

- Section 10.9, “Example custom network interfaces for NFV”

10.9. EXAMPLE CUSTOM NETWORK INTERFACES FOR NFV

The following examples illustrates how you can use a template to customize network interfaces for NFV in Red Hat OpenStack Services on OpenShift (RHOSO) environments.

10.9.1. Example template - non-partitioned NIC

This template example configures the RHOSO networks on a NIC that is not partitioned.

```
apiVersion: v1
data:
 25-igmp.conf: |
   [ovs]
   igmp_snooping_enable = True
kind: ConfigMap
metadata:
 name: neutron-igmp
 namespace: openstack
---
apiVersion: v1
data:
 25-cpu-pinning-nova.conf: |
   [DEFAULT]
   reserved_host_memory_mb = 4096
   [compute]
   cpu_shared_set = "0,20,1,21"
   cpu_dedicated_set = "8-19,28-39"
   [neutron]
   physnets = dpdkdata1
   [neutron_physnet_dpdkdata1]
   numa_nodes = 1
   [libvirt]
   cpu_power_management=false
kind: ConfigMap
metadata:
 name: ovs-dpdk-sriov-cpu-pinning-nova
 namespace: openstack
---
apiVersion: v1
data:
 03-sriov-nova.conf: |
   [pci]
   device_spec = {"address": "0000:05:00.2", "physical_network":"sriov-1", "trusted":"true"}
   device_spec = {"address": "0000:05:00.3", "physical_network":"sriov-2", "trusted":"true"}
   device_spec = {"address": "0000:07:00.0", "physical_network":"sriov-3", "trusted":"true"}
   device_spec = {"address": "0000:07:00.1", "physical_network":"sriov-4", "trusted":"true"}
kind: ConfigMap
metadata:
 name: sriov-nova
 namespace: openstack
---
apiVersion: v1
data:
 NodeRootPassword: cmVkaGF0Cg==
kind: Secret
metadata:
 name: baremetalset-password-secret
 namespace: openstack
type: Opaque
---
apiVersion: v1
data:
 authorized_keys:
```

```
ZWNkc2Etc2hhMi1uaXN0cDUyMSBBQUFBRTJWalpITmhMWE5vWVRJdGJtbHpkSEExTWpFQUFBQ
UlibWx6ZEhBMU1qRUFBQUNGQkFBVFdweE5LNlNYTEo0dnh2Y0F4N0t4c3FLenI0a3pEalRpT0dQa
3pyZWZnTjdVcmo2RUZPUXlBRWk5cXNnYkRVYXp0MktpdzJqc3djbG5TYW1zUDE0V2x3RkN2a1NuU
1o4cTZwWGJTbGpNa3Z1R3FiVXZoSTVxTVlMTDNlRWpyU21nNDlWcTBWZkdFQmxlWUx6TGFncV
BlN1FKR0NCMGlWTVk5b3N0TFdPM1NKbXVuZz09IGNpZm13X3JlcHJvZHVjZXJfa2V5Cg==
 ssh-privatekey:
LS0tLS1CRUdJTiBPUEVOU1NIIFBSSVZBVEUgS0VZLS0tLS0KYjNCbGJuTnphQzFyWlhrdGRqRUFB
QUFBQkc1dmJtVUFBQUFFYm05dVpRQUFBQUFBQUFBQkFBQUFyQUFBQUJObFkyUnpZUwoxe
mFHRXlMVzVwYzNSd05USXhBQUFBQ0c1cGMzUndOVEl4QUFBQWhRUUFFMXFjVFN1a2x5eWV
MOGIzQU1leXNiS2lzNitKCk13NDA0amhqNU02M240RGUxSzQraEJUa01nQkl2YXJJR3cxR3M3ZGlvc
05vN01ISlowbXByRDlIRnBjQlFyNUVwMG1mS3UKcVYyMHBZekpMN2hxbTFMNFNPYWpHQ3k5M2h
JNjBwb09QVmF0Rlh4aEFFaUJDOHkyb0tqM3UwQ1JnZ2RJbFRHUGFMTFMxagp0MGlacnA0QUFBBR
Vl0cGNtdHJhWEpyWUFBQUFUWldOa2MyRXRjMmhoTWkxdWFYTjBjBjRFV5TVFBQUFBaHVhWE4wY
0RVeU1RCkFBQUlVRUFCTmFuRTByeEpjc25pL0c5d0RIc3JHeW9yT3ZpE1PTk9JNFkrVE90NStBM
3RTdVBvUVU1REIBU0wycXlCc04KUnJPM1lxTERhT3pCeVdkSnFhdy9YaGFYQVVLK1JLZEpueXJxb
GR0S1dNeVMrNGFwdFMrRWptb3hnc3ZkNFNPdEthRGoxVwpyUlY4WVFHVWRndk10cUNvOTd0Q
WtZSUhTSlV4ajJpeTB0WTdkSW1hNmVBQUFBUWdHTWZobWFSblZFcnhjZ2Z6aVRpdzFnClBjYXBB
V21TMHh5dDNyclhoSnExd0pRMys3ZFp0Y3l0alg5VVVuNnh0NlE1M0JTT1ZvaWR2L2pZK2krYytNVVh
UZ0FBQUlKUmphV1p0ZDE5eVpYQnliMlIxWTJWeVgydGxlUUVDQXdRRkJnPT0KLS0tLS1FTkQgT1B
FTlNTSCBQUklWQVRFIEtFWS0tLS0tCg==
 ssh-publickey:
ZWNkc2Etc2hhMi1uaXN0cDUyMSBBQUFBRTJWalpITmhMWE5vWVRJdGJtbHpkSEExTWpFQUFBQ
UlibWx6ZEhBMU1qRUFBQUNGQkFBVFdweE5LNlNYTEo0dnh2Y0F4N0t4c3FLenI0a3pEalRpT0dQa
3pyZWZnTjdVcmo2RUZPUXlBRWk5cXNnYkRVYXp0MktpdzJqc3djbG5TYW1zUDE0V2x3RkN2a1NuU
1o4cTZwWGJTbGpNa3Z1R3FiVXZoSTVxTVlMTDNlRWpyU21nNDlWcTBWZkdFQmxlWUx6TGFncV
BlN1FKR0NCMGlWTVk5b3N0TFdPM1NKbXVuZz09IGNpZm13X3JlcHJvZHVjZXJfa2V5Cg==
kind: Secret
metadata:
 name: dataplane-ansible-ssh-private-key-secret
 namespace: openstack
type: Opaque
---
apiVersion: v1
data:
 LibvirtPassword: MTIzNDU2Nzg=
kind: Secret
metadata:
 name: libvirt-secret
 namespace: openstack
type: Opaque
---
apiVersion: v1
data:
 ssh-privatekey:
LS0tLS1CRUdJTiBPUEVOU1NIIFBSSVZBVEUgS0VZLS0tLS0KYjNCbGJuTnphQzFyWlhrdGRqRUFB
QUFBQkc1dmJtVUFBQUFFYm05dVpRQUFBQUFBQUFBQkFBQUFyQUFBQUJObFkyUnpZUwoxe
mFHRXlMVzVwYzNSd05USXhBQUFBQ0c1cGMzUndOVEl4QUFBQWhRUUFFwWTlSRzV5a2pLR3p2
c295dWlDZm1zakEwZkFYCmkvS0hQT3R3Zm9NZjRQZXpRSFFNOHFJZ0pGc0svaVlwNVJJWmNNVQl
cwVVBCNnBpazQ1L3k0QVF4bmVBQWRrN0JQbTc0dG8KSkxoVjY2U3pzV2pHR1NFdzVXVFBwVUV
paXdQMlNiL14dXloNWlLbUJyTE5SRWpYTEJvbjJJZWRBbEJMaC9FaGpkFZjUwo5ZzczQ0tvQUFBBR
VFoeS9PODRjdnp2TUFBQUFUWldOa2MyRXRjMmhoTWkxdWFYTjBjRFV5TVFBQUFBaHVhWE4wY
0RVeU1RCkFBQUlVRUFLV1BVUnVjEl5aHM3N0tNcm9nbjVySXdOSHdGNHZ5aHp6cmNINkNkRlK0Qz
czBCMERQS2lJQ1JiQ3Y0bUtlVVIKMlhGQVZ0RkR3ZXFZcE9PZjh1QUVNWjNnQUhaT3dUNXUrTGF
DUzRWZXVrczdGb3hoa2hNT1ZrejZWQklvc0Q5Sm9pSG5RIF
TNGZ4SVkzYlZYRXZZTzl3aXFBQUFBUWdEQ0lEdHFaZ0JNam8rbG1rRnhzV3NvCkxKOGxBSWF0a
0ZTdDkxcGJHWWIrVFRnS0NSOGhqbXdjalNoRzFlNlRaWZNTkc5TklzVlRYYjNjTkYvaThJTHV1UUF
```

```
BQUEKNXViM1poSUcxcFozSmhkR2x2YmdFQ0F3UT0KLS0tLS1FTkQgT1BFTlNTSCBQUklWQVRFI
EtFWS0tLS0tCg==
 ssh-publickey:
ZWNkc2Etc2hhMi1uaXN0cDUyMSBBQUFBRTJWalpITmhMWE5vWVRJdGJtbHpkSEExTWpFQUFBQ
UlibWx6ZEhBMU1qRUFBQUNGQkFDbGoxRWJuS1NNb2JPK3lqSzZJSitheU1EUjhCZUw4b2M4NjNC
K2d4L2c5N05BZEF6eW9pQWtXd3IrSmlubEVkbHhRRmJSUThIcW1LVGpuL0xnQkRHZDRBQjJUc0Ur
YnZpMmdrdUZYcnBMT3hhTVlaSVREbFpNK2xRU0tMQS9aSnY5akc3S0htSXFZR3NzMUVTTmNzR2l
mWWg1MENVRXVlOFNHTjIxVnhMMkR2Y0lxZz09IG5vdmEgbWlncmF0aW9uCg==
kind: Secret
metadata:
 name: nova-migration-ssh-key
 namespace: openstack
type: kubernetes.io/ssh-auth
---
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
 name: openstack-edpm
 namespace: openstack
spec:
 baremetalSetTemplate:
  bmhLabelSelector:
   app: openstack
  cloudUserName: cloud-admin
  ctlplaneInterface: enp130s0f0
  passwordSecret:
   name: baremetalset-password-secret
   namespace: openstack
  provisioningInterface: enp5s0
 env:
 - name: ANSIBLE_FORCE_COLOR
  value: "True"
 networkAttachments:
 - ctlplane
 nodeTemplate:
  ansible:
   ansiblePort: 22
   ansibleUser: cloud-admin
   ansibleVars:
    dns_search_domains: []
    edpm_bootstrap_command: |-
     # root CA
     cd /etc/pki/ca-trust/source/anchors/
     curl -LOk https://certs.corp.redhat.com/RH-IT-Root-CA.crt
     curl -LOk https://certs.corp.redhat.com/certs/2022-IT-Root-CA.pem
     update-ca-trust


     # install rhos-release repos
     dnf --nogpgcheck --repofrompath=rhos-release,http://download.devel.redhat.com/rcm-
guest/puddles/OpenStack/rhos-release/ --repo=rhos-release install -y rhos-release
     rhos-release ceph-7.1-rhel-9 -r 9.4


     # Issue #2 - edpm_bootstrap fails if we don't update container-selinux
     dnf update -y
```

```
        rpm -ivh --nosignature http://download.devel.redhat.com/rcm-guest/puddles/OpenStack/rhos-release/rhos-release-latest.noarch.rpm
        rhos-release ceph-7.1-rhel-9 -r 9.4
        curl -o /etc/yum.repos.d/delorean.repo https://osp-trunk.hosted.upshift.rdu2.redhat.com/rhel9-osp18/current-podified/delorean.repo
        echo "[osptrunk-candidate-deps]" >> "/etc/yum.repos.d/osptrunk-candidate-deps.repo"
        echo "name=osptrunk-candidate-deps" >> "/etc/yum.repos.d/osptrunk-candidate-deps.repo"
        echo "baseurl=http://download.eng.bos.redhat.com/brewroot/repos/rhos-18.0-rhel-9-trunk-candidate/latest/x86_64/" >> "/etc/yum.repos.d/osptrunk-candidate-deps.repo"
        echo "gpgcheck=0" >> /etc/yum.repos.d/osptrunk-candidate-deps.repo
        echo "enabled=1" >> /etc/yum.repos.d/osptrunk-candidate-deps.repo
        echo "priority=1" >> /etc/yum.repos.d/osptrunk-candidate-deps.repo
         # sets up rhoso release repo
        echo "[rhoso-18.0-rhel-9-nightly-compose]" >> /etc/yum.repos.d/rhosotrunk-compose-deps.repo
        echo "name=rhoso-18.0-rhel-9-nightly-compose" >> /etc/yum.repos.d/rhosotrunk-compose-deps.repo
        echo "baseurl=http://download.hosts.prod.upshift.rdu2.redhat.com/rhel-9/nightly/RHOSO/RHOSO-18.0-trunk/latest-RHOSO_TRUNK-18-RHEL-9/compose/OpenStack/x86_64/os/" >> /etc/yum.repos.d/rhosotrunk-compose-deps.repo
        echo "gpgcheck=0" >> /etc/yum.repos.d/rhosotrunk-compose-deps.repo
        echo "enabled=1" >> /etc/yum.repos.d/rhosotrunk-compose-deps.repo
        echo "priority=1" >> /etc/yum.repos.d/rhosotrunk-compose-deps.repo
        echo "includepkgs=rhoso-release-18*" >> /etc/yum.repos.d/rhosotrunk-compose-deps.repo
      edpm_fips_mode: check
      edpm_kernel_args: default_hugepagesz=1GB hugepagesz=1G hugepages=64 iommu=pt
        intel_iommu=on tsx=off isolcpus=2-19,22-39
      edpm_network_config_hide_sensitive_logs: false
      edpm_network_config_os_net_config_mappings:
        edpm-compute-0: (1)
          dmiString: system-product-name
          id: PowerEdge R730
          nic1: eno1
          nic2: eno2
          nic3: enp130s0f0
          nic4: enp130s0f1
          nic5: enp130s0f2
          nic6: enp130s0f3
          nic7: enp5s0f0
          nic8: enp5s0f1
          nic9: enp5s0f2
          nic10: enp5s0f3
          nic11: enp7s0f0np0
          nic12: enp7s0f1np1
        edpm-compute-1: (2)
          dmiString: system-product-name
          id: PowerEdge R730
          nic1: eno1
          nic2: eno2
          nic3: enp130s0f0
          nic4: enp130s0f1
          nic5: enp130s0f2
          nic6: enp130s0f3
          nic7: enp5s0f0
          nic8: enp5s0f1
```

```
        nic9: enp5s0f2
        nic10: enp5s0f3
        nic11: enp7s0f0np0
        nic12: enp7s0f1np1
    edpm_network_config_template: |
      ---
      {% set mtu_list = [ctlplane_mtu] %}
      {% for network in nodeset_networks %}
      {{ mtu_list.append(lookup(vars, networks_lower[network] ~ _mtu)) }}
      {%- endfor %}
      {% set min_viable_mtu = mtu_list | max %}
      network_config:
      - type: interface
        name: nic1
        use_dhcp: false
      - type: interface
        name: nic2
        use_dhcp: false
      - type: linux_bond      (3)
        name: bond_api
        use_dhcp: false
        bonding_options: "mode=active-backup"
        dns_servers: {{ ctlplane_dns_nameservers }}
        members:
          - type: interface
            name: nic3
            primary: true
        addresses:
        - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_cidr }}
        routes:
        - default: true
          next_hop: {{ ctlplane_gateway_ip }}
      - type: vlan       (4)
        vlan_id: {{ lookup(vars, networks_lower[internalapi] ~ _vlan_id) }}
        device: bond_api
        addresses:
        - ip_netmask: {{ lookup(vars, networks_lower[internalapi] ~ _ip) }}/{{ lookup(vars,
networks_lower[internalapi] ~ _cidr) }}
      - type: vlan       (5)
        vlan_id: {{ lookup(vars, networks_lower[storage] ~ _vlan_id) }}
        device: bond_api
        addresses:
        - ip_netmask: {{ lookup(vars, networks_lower[storage] ~ _ip) }}/{{ lookup(vars,
networks_lower[storage] ~ _cidr) }}
      - type: ovs_user_bridge  (6)
        name: br-link0
        use_dhcp: false
        ovs_extra: "set port br-link0 tag={{ lookup(vars, networks_lower[tenant] ~ _vlan_id) }}"
        addresses:
        - ip_netmask: {{ lookup(vars, networks_lower[tenant] ~ _ip) }}/{{ lookup(vars,
networks_lower[tenant] ~ _cidr) }}
        mtu: {{ lookup(vars, networks_lower[tenant] ~ _mtu) }}
        members:
        - type: ovs_dpdk_bond
          name: dpdkbond0
          mtu: 9000
```

```
      rx_queue: 2
      ovs_extra: "set port dpdkbond0 bond_mode=balance-slb"
      members:
      - type: ovs_dpdk_port
        name: dpdk0
        members:
        - type: interface
          name: nic7
      - type: ovs_dpdk_port
        name: dpdk1
        members:
        - type: interface
          name: nic8
  - type: ovs_user_bridge
    name: br-dpdk0
    mtu: 9000
    use_dhcp: false
    members:
    - type: ovs_dpdk_bond
      name: dpdkbond1
      mtu: 9000
      rx_queue: 3
      ovs_options: "bond_mode=balance-tcp lacp=active other_config:lacp-time=fast other-
config:lacp-fallback-ab=true other_config:lb-output-action=true"
      members:
        - type: ovs_dpdk_port
          name: dpdk2
          members:
            - type: interface
              name: nic5
        - type: ovs_dpdk_port
          name: dpdk3
          members:
            - type: interface
              name: nic6
  - type: ovs_user_bridge
    name: br-dpdk1
    mtu: 9000
    use_dhcp: false
    members:
      - type: ovs_dpdk_port
        name: dpdk4
        mtu: 9000
        rx_queue: 3
        members:
          - type: interface
            name: nic4
  - type: sriov_pf              (7)
    name: nic9
    numvfs: 10                  (8)
    mtu: 9000
    use_dhcp: false
    promisc: true
  - type: sriov_pf
    name: nic10
    numvfs: 10
```

```
          mtu: 9000
          use_dhcp: false
          promisc: true
        - type: sriov_pf   (9)
          name: nic11
          numvfs: 5   (10)
          mtu: 9000
          use_dhcp: false
          promisc: true
        - type: sriov_pf   (11)
          name: nic12
          numvfs: 5   (12)
          mtu: 9000
          use_dhcp: false
          promisc: true
      edpm_neutron_sriov_agent_SRIOV_NIC_physical_device_mappings: sriov-1:enp5s0f2,sriov-
2:enp5s0f3,sriov-3:enp7s0f0np0,sriov-4:enp7s0f1np1
      edpm_nodes_validation_validate_controllers_icmp: false
      edpm_nodes_validation_validate_gateway_icmp: false
      edpm_nova_libvirt_qemu_group: hugetlbfs
      edpm_ovn_bridge_mappings:
      - dpdkmgmt:br-link0
      - dpdkdata0:br-dpdk0
      - dpdkdata1:br-dpdk1
      edpm_ovs_dpdk_lcore_list: 0,20,1,21
      edpm_ovs_dpdk_memory_channels: "4"
      edpm_ovs_dpdk_pmd_auto_lb: "true"
      edpm_ovs_dpdk_pmd_core_list: 2,3,4,5,6,7,22,23,24,25,26,27
      edpm_ovs_dpdk_pmd_improvement_threshold: "25"
      edpm_ovs_dpdk_pmd_load_threshold: "70"
      edpm_ovs_dpdk_pmd_rebal_interval: "2"
      edpm_ovs_dpdk_socket_memory: 4096,4096
      edpm_ovs_dpdk_vhost_postcopy_support: "true"
      edpm_selinux_mode: enforcing
      edpm_sshd_allowed_ranges:
      - 192.168.122.0/24
      edpm_sshd_configure_firewall: true
      edpm_tuned_isolated_cores: 2-19,22-39
      edpm_tuned_profile: cpu-partitioning-powersave
      enable_debug: false
      gather_facts: false
      neutron_physical_bridge_name: br-access
      neutron_public_interface_name: nic1
      service_net_map:
        nova_api_network: internalapi
        nova_libvirt_network: internalapi
      timesync_ntp_servers:
      - hostname: clock.redhat.com
  ansibleSSHPrivateKeySecret: dataplane-ansible-ssh-private-key-secret
  managementNetwork: ctlplane
  networks:
  - defaultRoute: true
    name: ctlplane
    subnetName: subnet1
  - name: internalapi
    subnetName: subnet1
```

```
  - name: storage
    subnetName: subnet1
  - name: tenant
    subnetName: subnet1
 nodes:
  edpm-compute-0:
    hostName: compute-0
  edpm-compute-1:
    hostName: compute-1
 preProvisioned: false
 services:
 - bootstrap
 - download-cache
 - reboot-os
 - configure-ovs-dpdk
 - configure-network
 - validate-network
 - install-os
 - configure-os
 - ssh-known-hosts
 - run-os
 - install-certs
 - ovn
 - neutron-ovn-igmp
 - neutron-metadata
 - neutron-sriov
 - libvirt
 - nova-custom-ovsdpdksriov
 - telemetry
---
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
 name: neutron-ovn-igmp
 namespace: openstack
spec:
 caCerts: combined-ca-bundle
 dataSources:
 - configMapRef:
    name: neutron-igmp
 - secretRef:
    name: neutron-ovn-agent-neutron-config
 edpmServiceType: neutron-ovn
 label: neutron-ovn-igmp
 playbook: osp.edpm.neutron_ovn
 tlsCerts:
  default:
    contents:
    - dnsnames
    - ips
    issuer: osp-rootca-issuer-ovn
    keyUsages:
    - digital signature
    - key encipherment
    - client auth
    networks:
```

```
      - ctlplane
---
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
 name: nova-custom-ovsdpdksriov
 namespace: openstack
spec:
 caCerts: combined-ca-bundle
 dataSources:
 - configMapRef:
     name: ovs-dpdk-sriov-cpu-pinning-nova
 - configMapRef:
     name: sriov-nova
 - secretRef:
     name: nova-cell1-compute-config
 - secretRef:
     name: nova-migration-ssh-key
 edpmServiceType: nova
 label: nova-custom-ovsdpdksriov
 playbook: osp.edpm.nova
 tlsCerts:
   default:
     contents:
     - dnsnames
     - ips
     issuer: osp-rootca-issuer-internal
     networks:
     - ctlplane
```

1 2 **edpm-compute-n**: defines the **edpm_network_config_os_net_config_mappings** variable to map the actual NICs. You identify each NIC by specifying the MAC address or the device name on each compute node to the NIC ID that the RHOSO **os-net-config** tool uses which is typically, `nic`*n*.

3 **linux_bond**: creates a control-plane Linux bond for an isolated network. In this example, a Linux bond is created with mode active-backup on **nic3** and **nic4**.

4 5 **type: vlan**: assign VLANs to Linux bonds. In this example, the VLAN ID of the **internalapi** and **storage** networks is assigned to **bond-api**.

6 **ovs_user_bridge**: set a bridge with OVS-DPDK ports. In this example, an OVS user bridge is created with a DPDK bond that has two DPDK ports that corresponds to **nic7** and **nic8** for the tenant network. A GENEVE tunnel is used.

7 9 11 **sriov_pf**: create SR-IOV VFs. In this example, an interface type of **sriov_pf** is configured as a physical function that the host can use.

8 10 12 **numvfs**: only set the number of VFs that are required.

10.9.2. Example template - partitioned NIC

This template example configures the RHOSO networks on a NIC that is partitioned. This example only shows the portion of the custom resource (CR) definition where the NIC is partitioned.

```
edpm_network_config_os_net_config_mappings:
  dellr750:
    dmiString: system-product-name
    id: PowerEdge R750
    nic1: eno8303
    nic2: ens1f0
    nic3: ens1f1
    nic4: ens1f2
    nic5: ens1f3
    nic6: ens2f0np0
    nic7: ens2f1np1
edpm_network_config_template: |
  ---
  {% set mtu_list = [ctlplane_mtu] %}
  {% for network in nodeset_networks %}
  {{ mtu_list.append(lookup(vars, networks_lower[network] ~ _mtu)) }}
  {%- endfor %}
  {% set min_viable_mtu = mtu_list | max %}
  network_config:
  - type: interface
    name: nic1
    use_dhcp: false
  - type: interface
    name: nic2
    use_dhcp: false
    addresses:
    - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_cidr }}
    routes:
    - default: true
      next_hop: {{ ctlplane_gateway_ip }}
  - type: sriov_pf
    name: nic3
    mtu: 9000
    numvfs: 5
    use_dhcp: false
    defroute: false
    nm_controlled: true
    hotplug: true
  - type: sriov_pf
    name: nic4
    mtu: 9000
    numvfs: 5
    use_dhcp: false
    defroute: false
    nm_controlled: true
    hotplug: true
  - type: linux_bond
    name: bond_api
    use_dhcp: false
    bonding_options: "mode=active-backup"
    dns_servers: {{ ctlplane_dns_nameservers }}
    members:
    - type: sriov_vf
      device: nic3
      vfid: 0
      vlan_id: {{ lookup(vars, networks_lower[internalapi] ~ _vlan_id) }}
```

```
        - type: sriov_vf
          device: nic4
          vfid: 0
          vlan_id: {{ lookup(vars, networks_lower[internalapi] ~ _vlan_id) }}
        addresses:
        - ip_netmask: {{ lookup(vars, networks_lower[internalapi] ~ _ip) }}/{{ lookup(vars,
networks_lower[internalapi] ~ _cidr) }}
      - type: linux_bond
        name: storage_bond
        use_dhcp: false
        bonding_options: "mode=active-backup"
        dns_servers: {{ ctlplane_dns_nameservers }}
        members:
        - type: sriov_vf
          device: nic3
          vfid: 1
          vlan_id: {{ lookup(vars, networks_lower[storage] ~ _vlan_id) }}
        - type: sriov_vf
          device: nic4
          vfid: 1
          vlan_id: {{ lookup(vars, networks_lower[storage] ~ _vlan_id) }}
        addresses:
        - ip_netmask: {{ lookup(vars, networks_lower[storage] ~ _ip) }}/{{ lookup(vars,
networks_lower[storage] ~ _cidr) }}
      - type: linux_bond
        name: mgmtst_bond
        use_dhcp: false
        bonding_options: "mode=active-backup"
        dns_servers: {{ ctlplane_dns_nameservers }}
        members:
        - type: sriov_vf
          device: nic3
          vfid: 2
          vlan_id: {{ lookup(vars, networks_lower[storagemgmt] ~ _vlan_id) }}
        - type: sriov_vf
          device: nic4
          vfid: 2
          vlan_id: {{ lookup(vars, networks_lower[storagemgmt] ~ _vlan_id) }}
        addresses:
        - ip_netmask: {{ lookup(vars, networks_lower[storagemgmt] ~ _ip) }}/{{ lookup(vars,
networks_lower[storagemgmt] ~ _cidr) }}
      - type: ovs_user_bridge
        name: br-link0
        use_dhcp: false
        mtu: 9000
        ovs_extra: "set port br-link0 tag={{ lookup(vars, networks_lower[tenant] ~ _vlan_id) }}"
        addresses:
        - ip_netmask: {{ lookup(vars, networks_lower[tenant] ~ _ip) }}/{{ lookup(vars,
networks_lower[tenant] ~ _cidr) }}
        members:
        - type: ovs_dpdk_bond
          name: dpdkbond0
          mtu: 9000
          rx_queue: 1
          members:
          - type: ovs_dpdk_port
```

```
          name: dpdk0
          members:
          - type: sriov_vf
            device: nic3
            vfid: 3
        - type: ovs_dpdk_port
          name: dpdk1
          members:
          - type: sriov_vf
            device: nic4
            vfid: 3
    - type: ovs_user_bridge
      name: br-dpdk0
      use_dhcp: false
      mtu: 9000
      rx_queue: 1
      members:
        - type: ovs_dpdk_port
          name: dpdk2
          members:
            - type: interface
              name: nic5
    - type: sriov_pf
      name: nic6
      mtu: 9000
      numvfs: 5
      use_dhcp: false
      defroute: false
    - type: sriov_pf
      name: nic7
      mtu: 9000
      numvfs: 5
      use_dhcp: false
      defroute: false
```

Additional resources

- Section 10.8, "Network interface configuration options"

10.10. DEPLOYING THE DATA PLANE

You use the **OpenStackDataPlaneDeployment** CRD to configure the services on the data plane nodes and deploy the data plane. You control the execution of Ansible on the data plane by creating **OpenStackDataPlaneDeployment** custom resources (CRs). Each **OpenStackDataPlaneDeployment** CR models a single Ansible execution.

When the **OpenStackDataPlaneDeployment** successfully completes execution, it does not automatically execute the Ansible again, even if the **OpenStackDataPlaneDeployment** or related **OpenStackDataPlaneNodeSet** resources are changed. To start another Ansible execution, you must create another **OpenStackDataPlaneDeployment** CR.

Create an **OpenStackDataPlaneDeployment** (CR) that deploys each of your **OpenStackDataPlaneNodeSet** CRs.

Procedure

1. Create a file on your workstation named **openstack_data_plane_deploy.yaml** to define the **OpenStackDataPlaneDeployment** CR:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: openstack-data-plane 1
```

1 The **OpenStackDataPlaneDeployment** CR name must be unique, must consist of lower case alphanumeric characters, **-** (hyphen) or **.** (period), must start and end with an alphanumeric character, and must have a maximum length of 20 characters. Update the name in this example to a name that reflects the node sets in the deployment.

2. In the list of services, replace **nova** with **nova-custom-sriov**, **nova-custom-ovsdpdk**, or both:

```
spec:
  services:
    - bootstrap
    - download-cache
    - reboot-os
    - configure-ovs-dpdk
    - configure-network
    - validate-network
    - install-os
    - configure-os
    - ssh-known-hosts
    - run-os
    - install-certs
    - ovn
    - neutron-ovn-igmp
    - neutron-metadata
    - neutron-sriov
    - libvirt
    - nova-custom-sriov
    - nova-custom-ovsdpdk
    - telemetry
  nodeSets:
  ...
```

3. Add all the **OpenStackDataPlaneNodeSet** CRs that you want to deploy.

```
spec:
  nodeSets:
    - openstack-data-plane
    - <nodeSet_name>
    - ...
    - <nodeSet_name>
  services:
  ...
```

- Replace **<nodeSet_name>** with the names of the **OpenStackDataPlaneNodeSet** CRs that you want to include in your data plane deployment.

4. Save the **openstack_data_plane_deploy.yaml** deployment file.

5. Deploy the data plane:

```
$ oc create -f openstack_data_plane_deploy.yaml -n openstack
```

You can view the Ansible logs while the deployment executes:

```
$ oc get pod -l app=openstackansibleee -w
$ oc logs -l app=openstackansibleee -f --max-log-requests 10
```

6. Confirm that the data plane is deployed:

```
$ oc get openstackdataplanedeployment -n openstack
```

Sample output

```
NAME                  STATUS   MESSAGE
openstack-data-plane   True     Setup Complete
```

7. Repeat the **oc get** command until you see the **NodeSet Ready** message:

```
$ oc get openstackdataplanenodeset -n openstack
```

Sample output

```
NAME                  STATUS   MESSAGE
openstack-data-plane   True     NodeSet Ready
```

For information about the meaning of the returned status, see Data plane conditions and states.

If the status indicates that the data plane has not been deployed, then troubleshoot the deployment. For information, see Troubleshooting the data plane creation and deployment.

8. Map the Compute nodes to the Compute cell that they are connected to:

```
$ oc rsh nova-cell0-conductor-0 nova-manage cell_v2 discover_hosts --verbose
```

If you did not create additional cells, this command maps the Compute nodes to **cell1**.

Verification

- Access the remote shell for the **openstackclient** pod and confirm that the deployed Compute nodes are visible on the control plane:

```
$ oc rsh -n openstack openstackclient
$ openstack hypervisor list
```

10.11. DATA PLANE CONDITIONS AND STATES

Each data plane resource has a series of conditions within their **status** subresource that indicates the overall state of the resource, including its deployment progress.

For an **OpenStackDataPlaneNodeSet**, until an **OpenStackDataPlaneDeployment** has been started and finished successfully, the **Ready** condition is **False**. When the deployment succeeds, the **Ready** condition is set to **True**. A subsequent deployment sets the **Ready** condition to **False** until the deployment succeeds, when the **Ready** condition is set to **True**.

Table 10.14. **OpenStackDataPlaneNodeSet** CR conditions

Condition	Description
Ready	• "True": The **OpenStackDataPlaneNodeSet** CR is successfully deployed. • "False": The deployment is not yet requested or has failed, or there are other failed conditions.
SetupReady	"True": All setup tasks for a resource are complete. Setup tasks include verifying the SSH key secret, verifying other fields on the resource, and creating the Ansible inventory for each resource. Each service-specific condition is set to "True" when that service completes deployment. You can check the service conditions to see which services have completed their deployment, or which services failed.
DeploymentReady	"True": The NodeSet has been successfully deployed.
InputReady	"True": The required inputs are available and ready.
NodeSetDNSDataReady	"True": DNSData resources are ready.
NodeSetIPReservationReady	"True": The IPSet resources are ready.
NodeSetBaremetalProvisionReady	"True": Bare-metal nodes are provisioned and ready.

Table 10.15. **OpenStackDataPlaneNodeSet** status fields

Status field	Description
Deployed	• "True": The **OpenStackDataPlaneNodeSet** CR is successfully deployed. • "False": The deployment is not yet requested or has failed, or there are other failed conditions.
DNSClusterAddresses	
CtlplaneSearchDomain	

Table 10.16. **OpenStackDataPlaneDeployment** CR conditions

Condition	Description
Ready	• "True": The data plane is successfully deployed. • "False": The data plane deployment failed, or there are other failed conditions.
DeploymentReady	"True": The data plane is successfully deployed.
InputReady	"True": The required inputs are available and ready.
<NodeSet> Deployment Ready	"True": The deployment has succeeded for the named **NodeSet**, indicating all services for the**NodeSet** have succeeded.
<NodeSet> <Service> Deployment Ready	"True": The deployment has succeeded for the named **NodeSet** and **Service**. Each **<NodeSet> <Service> Deployment Ready** specific condition is set to "True" as that service completes successfully for the named **NodeSet**. Once all services are complete for a **NodeSet**, the **<NodeSet> Deployment Ready** condition is set to "True". The service conditions indicate which services have completed their deployment, or which services failed and for which **NodeSets**.

Table 10.17. **OpenStackDataPlaneDeployment** status fields

Status field	Description
Deployed	• "True": The data plane is successfully deployed. All Services for all NodeSets have succeeded. • "False": The deployment is not yet requested or has failed, or there are other failed conditions.

Table 10.18. **OpenStackDataPlaneService** CR conditions

Condition	Description
Ready	"True": The service has been created and is ready for use. "False": The service has failed to be created.

10.12. TROUBLESHOOTING DATA PLANE CREATION AND DEPLOYMENT

To troubleshoot a deployment when services are not deploying or operating correctly, you can check the job condition message for the service, and you can check the logs for a node set.

10.12.1. Checking the job condition message for a service

Each data plane deployment in the environment has associated services. Each of these services have a job condition message that matches the current status of the AnsibleEE job executing for that service. This information can be used to troubleshoot deployments when services are not deploying or operating correctly.

Procedure

1. Determine the name and status of all deployments:

```
$ oc get openstackdataplanedeployment
```

   The following example output shows two deployments currently in progress:

```
$ oc get openstackdataplanedeployment

NAME                NODESETS                    STATUS   MESSAGE
data-plane-deploy   ["openstack-data-plane-1"]  False    Deployment in progress
data-plane-deploy   ["openstack-data-plane-2"]  False    Deployment in progress
```

2. Determine the name and status of all services and their job condition:

```
$ oc get openstackansibleee
```

   The following example output shows all services and their job condition for all current deployments:

```
$ oc get openstackansibleee

NAME                             NETWORKATTACHMENTS   STATUS   MESSAGE
bootstrap-openstack-edpm         ["ctlplane"]         True     Job complete
download-cache-openstack-edpm    ["ctlplane"]         False    Job is running
repo-setup-openstack-edpm        ["ctlplane"]         True     Job complete
validate-network-another-osdpd   ["ctlplane"]         False    Job is running
```

   For information on the job condition messages, see Job condition messages.

3. Filter for the name and service for a specific deployment:

```
$ oc get openstackansibleee -l \
openstackdataplanedeployment=<deployment_name>
```

   - Replace **<deployment_name>** with the name of the deployment to use to filter the services list.
     The following example filters the list to only show services and their job condition for the **data-plane-deploy** deployment:

```
$ oc get openstackansibleee -l \
openstackdataplanedeployment=data-plane-deploy

NAME                             NETWORKATTACHMENTS   STATUS   MESSAGE
```

```
bootstrap-openstack-edpm        ["ctlplane"]        True     Job complete
download-cache-openstack-edpm   ["ctlplane"]        False    Job is running
repo-setup-openstack-edpm       ["ctlplane"]        True     Job complete
```

10.12.1.1. Job condition messages

AnsibleEE jobs have an associated condition message that indicates the current state of the service job. This condition message is displayed in the **MESSAGE** field of the **oc get openstackansibleee** command output. Jobs return one of the following conditions when queried:

- **Job not started**: The job has not started.
- **Job not found**: The job could not be found.
- **Job is running**: The job is currently running.
- **Job complete**: The job execution is complete.
- **Job error occured <error_message>**: The job stopped executing unexpectedly. The **<error_message>** is replaced with a specific error message.

To further investigate a service that is displaying a particular job condition message, view its logs by using the command **oc logs job/<service>**. For example, to view the logs for the **repo-setup-openstack-edpm** service, use the command **oc logs job/repo-setup-openstack-edpm**.

10.12.2. Checking the logs for a node set

You can access the logs for a node set to check for deployment issues.

Procedure

1. Retrieve pods with the **OpenStackAnsibleEE** label:

```
$ oc get pods -l app=openstackansibleee
configure-network-edpm-compute-j6r4l   0/1     Completed          0         3m36s
validate-network-edpm-compute-6g7n9    0/1     Pending            0         0s
validate-network-edpm-compute-6g7n9    0/1     ContainerCreating  0         11s
validate-network-edpm-compute-6g7n9    1/1     Running            0         13s
```

2. SSH into the pod you want to check:

   a. Pod that is running:

   ```
   $ oc rsh validate-network-edpm-compute-6g7n9
   ```

   b. Pod that is not running:

   ```
   $ oc debug configure-network-edpm-compute-j6r4l
   ```

3. List the directories in the **/runner/artifacts** mount:

```
$ ls /runner/artifacts
configure-network-edpm-compute
validate-network-edpm-compute
```

4. View the **stdout** for the required artifact:

```
$ cat /runner/artifacts/configure-network-edpm-compute/stdout
```

CHAPTER 11. ACCESSING THE RHOSO CLOUD

You can access your Red Hat OpenStack Services on OpenShift (RHOSO) cloud to perform actions on your data plane by either accessing the OpenStackClient pod through a remote shell from your workstation, or by using a web browser to access the Dashboard service (horizon) interface.

11.1. ACCESSING THE OPENSTACKCLIENT POD

You can execute Red Hat OpenStack Services on OpenShift (RHOSO) commands on the deployed data plane by using the **OpenStackClient** pod through a remote shell from your workstation. The OpenStack Operator created the **OpenStackClient** pod as a part of the **OpenStackControlPlane** resource. The **OpenStackClient** pod contains the client tools and authentication details that you require to perform actions on your data plane.

Procedure

1. Access the remote shell for **openstackclient**:

```
$ oc rsh -n openstack openstackclient
```

2. Change to the **cloud-admin** home directory:

```
$ cd /home/cloud-admin
```

3. Run your **openstack** commands. For example, you can create a **default** network with the following command:

```
$ openstack network create default
```

Additional resources

- *Creating and managing instances*
- *Configuring networking services*

11.2. ACCESSING THE DASHBOARD SERVICE (HORIZON) INTERFACE

You can access the Dashboard service (horizon) interface by using a web browser to access the virtual IP address that is reserved by the control plane.

Procedure

1. To log in as the admin user, obtain the admin password from the **AdminPassword** parameter in the **osp-secret** secret:

```
$ oc get secret osp-secret -o jsonpath='{.data.AdminPassword}' | base64 -d
```

2. Retrieve the Dashboard service endpoint URL:

```
$ oc get horizons horizon -o jsonpath='{.status.endpoint}'
```

3. Open a web browser.

4. Enter the Dashboard endpoint URL.

5. Log in to the dashboard with your username and password.

CHAPTER 12. TUNING NFV IN A RED HAT OPENSTACK SERVICES ON OPENSHIFT ENVIRONMENT

12.1. MANAGING PORT SECURITY IN NFV ENVIRONMENTS

Port security is an anti-spoofing measure that blocks any egress traffic that does not match the source IP and source MAC address of the originating network port. You cannot view or modify this behavior using security group rules.

By default, the **port_security_enabled** parameter is set to **enabled** on newly created Networking service (neutron) networks in Red Hat OpenStack Services on OpenShift (RHOSO) environments. Newly created ports copy the value of the **port_security_enabled** parameter from the network they are created on.

For some NFV use cases, such as building a firewall or router, you must disable port security.

Prerequisites

- You have the **oc** command line tool installed on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with **cluster-admin** privileges.

Procedure

1. Access the remote shell for the OpenStackClient pod from your workstation:

```
$ oc rsh -n openstack openstackclient
```

2. To disable port security on a single port, run the following command:

```
$ openstack port set --disable-port-security <port-id>
```

3. To prevent port security from being enabled on any newly created port on a network, run the following command:

```
$ openstack network set --disable-port-security <network-id>
```

4. Exit the **openstackclient** pod:

```
$ exit
```

12.2. CREATING AND USING VF PORTS

By running various OpenStack CLI client commands, you can create and use virtual function (VF) ports.

Prerequisites

- You have the **oc** command line tool installed on your workstation.
- You are logged on to a workstation that has access to the RHOSO control plane as a user with **cluster-admin** privileges.

Procedure

1. Access the remote shell for the OpenStackClient pod from your workstation:

```
$ oc rsh -n openstack openstackclient
```

2. Create a network of type **vlan**.

```
$ openstack network create trusted_vf_network  --provider-network-type vlan \
 --provider-segment 111 --provider-physical-network sriov2 \
 --external --disable-port-security
```

3. Create a subnet.

```
$ openstack subnet create --network trusted_vf_network \
  --ip-version 4 --subnet-range 192.168.111.0/24 --no-dhcp \
 subnet-trusted_vf_network
```

4. Create a port.
Set the **vnic-type** option to **direct**, and the **binding-profile** option to **true**.

```
$ openstack port create --network sriov111 \
--vnic-type direct --binding-profile trusted=true \
sriov111_port_trusted
```

5. Create an instance, and bind it to the previously-created trusted port.

```
$ openstack server create --image rhel --flavor dpdk  --network internal --port
trusted_vf_network_port_trusted --config-drive True --wait rhel-dpdk-sriov_trusted
```

6. Exit the **openstackclient** pod:

```
$ exit
```

Verification

Confirm the trusted VF configuration on the hypervisor by performing the following steps:

1. On the compute node that you created the instance, enter the following command:

```
$ ip link
```

Sample output

```
7: p5p2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether b4:96:91:1c:40:fa brd ff:ff:ff:ff:ff:ff
    vf 6 MAC fa:16:3e:b8:91:c2, vlan 111, spoof checking off, link-state auto, trust on,
query_rss off
    vf 7 MAC fa:16:3e:84:cf:c8, vlan 111, spoof checking off, link-state auto, trust off, query_rss
off
```

2. Verify that the trust status of the VF is **trust on**. The example output contains details of an environment that contains two ports. Note that **vf 6** contains the text **trust on**.

3. You can disable spoof checking if you set **port_security_enabled: false** in the Networking service (neutron) network, or if you include the argument **--disable-port-security** when you run the **openstack port create** command.

12.3. KNOWN LIMITATIONS FOR NUMA-AWARE VSWITCHES

> **IMPORTANT**
>
> This feature is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

This section lists the constraints for implementing a NUMA-aware vSwitch in a Red Hat OpenStack Services on OpenShift (RHOSO) network functions virtualization infrastructure (NFVi).

- You cannot start a VM that has two NICs connected to physnets on different NUMA nodes, if you did not specify a two-node guest NUMA topology.
- You cannot start a VM that has one NIC connected to a physnet and another NIC connected to a tunneled network on different NUMA nodes, if you did not specify a two-node guest NUMA topology.
- You cannot start a VM that has one vhost port and one VF on different NUMA nodes, if you did not specify a two-node guest NUMA topology.
- NUMA-aware vSwitch parameters are specific to overcloud roles. For example, Compute node 1 and Compute node 2 can have different NUMA topologies.
- If the interfaces of a VM have NUMA affinity, ensure that the affinity is for a single NUMA node only. You can locate any interface without NUMA affinity on any NUMA node.
- Configure NUMA affinity for data plane networks, not management networks.
- NUMA affinity for tunneled networks is a global setting that applies to all VMs.

12.4. QUALITY OF SERVICE (QOS) IN NFVI ENVIRONMENTS

You can offer varying service levels for VM instances by using quality of service (QoS) policies to apply rate limits to egress and ingress traffic on Red Hat OpenStack Services on OpenShift (RHOSO) networks in a network functions virtualization infrastructure (NFVi).

In NFVi environments, QoS support is limited to the following rule types:

- **minimum bandwidth** on SR-IOV, if supported by vendor.
- **bandwidth limit** on SR-IOV and OVS-DPDK egress interfaces.

Additional resources

- Using Quality of Service (QoS) policies to manage data traffic in *Configuring networking services*

12.5. CREATING AN HCI DATA PLANE THAT USES DPDK

You can deploy your NFV infrastructure with hyperconverged nodes, by co-locating and configuring Compute and Ceph Storage services for optimized resource usage.

For more information about hyperconverged infrastructure (HCI), see *Deploying a hyperconverged infrastructure environment*.

12.5.1. Example NUMA node configuration

For increased performance, place the tenant network and Ceph object service daemon (OSD)s in one NUMA node, such as NUMA-0, and the VNF and any non-NFV VMs in another NUMA node, such as NUMA-1.

Table 12.1. CPU allocation

NUMA-0	NUMA-1
Number of Ceph OSDs * 4 HT	Guest vCPU for the VNF and non-NFV VMs
DPDK lcore - 2 HT	DPDK lcore - 2 HT
DPDK PMD - 2 HT	DPDK PMD - 2 HT

Table 12.2. Example of CPU allocation

	NUMA-0	NUMA-1
Ceph OSD	32,34,36,38,40,42,76,78,80,82,84,86	
DPDK-lcore	0,44	1,45
DPDK-pmd	2,46	3,47
nova		5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,49,51,53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87

12.5.2. Recommended configuration for HCI-DPDK deployments

Table 12.3. Tunable parameters for HCI deployments

Block Device Type	OSDs, Memory, vCPUs per device
NVMe	Memory : 5GB per OSD OSDs per device: 4 vCPUs per device: 3

Block Device Type	OSDs, Memory, vCPUs per device
SSD	Memory : 5GB per OSD OSDs per device: 1 vCPUs per device: 4
HDD	Memory : 5GB per OSD OSDs per device: 1 vCPUs per device: 1

Use the same NUMA node for the following functions:

- Disk controller
- Storage networks
- Storage CPU and memory

Allocate another NUMA node for the following functions of the DPDK provider network:

- NIC
- PMD CPUs
- Socket memory