Abstract

This guide contains important planning information and describes the configuration procedures for single root input/output virtualization (SR-IOV) and dataplane development kit (DPDK) for network functions virtualization infrastructure (NFVI) in your Red Hat OpenStack Platform deployment.
# Table of Contents

**PREFACE** .................................................................................................................. 4

**CHAPTER 1. OVERVIEW** ............................................................................................. 5

**CHAPTER 2. HARDWARE REQUIREMENTS** ................................................................. 6
  2.1. NETWORK ADAPTER SUPPORT ......................................................................... 6
  2.2. DISCOVERING YOUR NUMA NODE TOPOLOGY ........................................... 6
  2.3. REVIEW BIOS SETTINGS .................................................................................. 10

**CHAPTER 3. SOFTWARE REQUIREMENTS** ................................................................. 11
  3.1. REGISTERING AND ENABLING REPOSITORIES ........................................... 11
  3.2. SUPPORTED CONFIGURATIONS FOR NFV DEPLOYMENTS .......................... 12
  3.3. SUPPORTED DRIVERS .................................................................................... 12
  3.4. COMPATIBILITY WITH THIRD PARTY SOFTWARE ....................................... 12

**CHAPTER 4. NETWORK CONSIDERATIONS** ............................................................... 14

**CHAPTER 5. PLANNING AN SR-IOV DEPLOYMENT** .................................................... 15
  5.1. HARDWARE PARTITIONING FOR AN SR-IOV DEPLOYMENT ...................... 15
  5.2. TOPOLOGY OF AN NFV SR-IOV DEPLOYMENT .......................................... 15
    5.2.1. NFV SR-IOV without HCI ...................................................................... 16

**CHAPTER 6. DEPLOYING SR-IOV TECHNOLOGIES** .................................................... 18
  6.1. PREREQUISITES ............................................................................................... 18
  6.2. CONFIGURING SR-IOV .................................................................................... 18
  6.3. CONFIGURING HARDWARE OFFLOAD (TECHNOLOGY PREVIEW) .......... 20
    6.3.1. Open vSwitch hardware offload ............................................................. 20
    6.3.2. Verification .............................................................................................. 22
  6.4. DEPLOYING AN INSTANCE FOR SR-IOV ....................................................... 22
  6.5. CREATING HOST AGGREGATES .................................................................... 23
    6.5.1. Procedure ................................................................................................ 23

**CHAPTER 7. PLANNING YOUR OVS-DPDK DEPLOYMENT** .......................................... 25
  7.1. OVS-DPDK WITH CPU PARTITIONING AND NUMA TOPOLOGY .............. 25
  7.2. OVERVIEW OF WORKFLOWS AND DERIVED PARAMETERS ................. 25
  7.3. DERIVED OVS-DPDK PARAMETERS ............................................................. 26
  7.4. OVERVIEW OF MANUALLY CALCULATED OVS-DPDK PARAMETERS ....... 27
    7.4.1. CPU parameters ....................................................................................... 27
    7.4.2. Memory parameters ............................................................................... 28
    7.4.3. Networking parameters ......................................................................... 30
    7.4.4. Other parameters ................................................................................... 31
    7.4.5. Instance extra specifications ................................................................... 31
  7.5. TWO NUMA NODE EXAMPLE OVS-DPDK DEPLOYMENT ......................... 32
  7.6. TOPOLOGY OF AN NFV OVS-DPDK DEPLOYMENT ....................................... 33

**CHAPTER 8. CONFIGURING AN OVS-DPDK DEPLOYMENT** ....................................... 36
  8.1. DERIVING DPDK PARAMETERS WITH WORKFLOWS .............................. 36
  8.2. OVS-DPDK TOPOLOGY .................................................................................. 38
    Prerequisites ...................................................................................................... 40
  8.3. SETTING THE MTU VALUE FOR OVS-DPDK INTERFACES ....................... 40
  8.4. CONFIGURING A FIREWALL FOR SECURITY GROUPS ................................. 41
  8.5. SETTING MULTIQUEUE FOR OVS-DPDK INTERFACES ............................. 42
  8.6. DEPLOYING THE OVERCLOUD ..................................................................... 42
PREFACE

Red Hat OpenStack Platform provides the foundation to build a private or public cloud on top of Red Hat Enterprise Linux. It offers a massively scalable, fault-tolerant platform for the development of cloud-enabled workloads.

This guide describes the steps to plan and configure single root I/O virtualization (SR-IOV) and Open vSwitch (OVS) with Data Plane Development Kit (OVS-DPDK) using the Red Hat OpenStack Platform director for network functions virtualization (NFV) deployments.
CHAPTER 1. OVERVIEW

Network Functions Virtualization (NFV) is a software solution that virtualizes a network function on general purpose, cloud-based infrastructure. NFV allows the Communication Service Provider to move away from traditional hardware.

For a high-level overview of the NFV concepts, see the Network Functions Virtualization Product Guide.

NOTE

This guide provides examples for CPU assignments, memory allocation, and NIC configurations that may vary from your topology and use case.

You can use Red Hat OpenStack Platform director to separate specific overcloud network types, for example, external, tenant, and internal API networks, into isolated networks. You can deploy a network on a single network interface or distributed over a multiple host network interface. Create bonds by assigning multiple interfaces to a single bridge with Open vSwitch. Use template configuration files to configure network isolation. Without template configuration files, all the service networks in your Red Hat OpenStack Platform installation are deployed on the provisioning network. There are multiple types of template configuration files:

network-environment.yaml

This file contains network details, subnets, and IP address ranges for the overcloud nodes. This file also contains settings that override the default parameter values for various scenarios.

compute.yaml and controller.yaml

These files contain the host network interface configuration for the overcloud nodes.

host-config-and-reboot.yaml

This file replaces the deprecated first-boot.yaml file, and contains configuration for host installation.

These Heat template files are located at /usr/share/openstack-tripleo-heat-templates/ on the undercloud node.

The following sections provide more details on how to plan and configure the Heat template files for NFV using the Red Hat OpenStack Platform director.

NOTE

NFV configuration makes use of YAML files. See YAML in a Nutshell for an introduction to the YAML file format.
CHAPTER 2. HARDWARE REQUIREMENTS

This section describes the hardware requirements for NFV.

You can use Red Hat Technologies Ecosystem to check for a list of certified hardware, software, cloud providers, and components. Choose the category and select the product version.

For a complete list of the certified hardware for Red Hat OpenStack Platform, see Red Hat OpenStack Platform certified hardware.

2.1. NETWORK ADAPTER SUPPORT

For a list of tested NICs for NFV, see Network Adapter Support. Customer Portal login required.

If you configure OVS-DPDK on Mellanox ConnectX-4 or ConnectX-5 network interfaces, you must set the corresponding kernel driver in the compute-ovs-dpdk.yaml file:

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

2.2. DISCOVERING YOUR NUMA NODE TOPOLOGY

When you plan your deployment, 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 onto each bare-metal node to manually collect the information.

**NOTE**

You must install and configure the undercloud before you can retrieve NUMA information through hardware introspection. See the Director Installation and Usage Guide for details.

Retrieving Hardware Introspection Details

The Bare Metal service hardware inspection extras (inspection_extras) is enabled by default to retrieve hardware details. You can use these hardware details to configure your overcloud. See Configuring the Director for details on the inspection_extras parameter in the undercloud.conf file.

For example, the numa_topology collector is part of these hardware inspection extras and includes the following information for each NUMA node:

- RAM (in kilobytes)
- Physical CPU cores and their sibling threads
• NICs associated with the NUMA node

Use the `openstack baremetal introspection data save _UUID_ | jq .numa_topology` command to retrieve this information, with the `_UUID_` of the bare-metal node.

The following example shows the retrieved NUMA information for a bare-metal node:

```json
{
  "cpus": [
    {
      "cpu": 1,
      "thread_siblings": [
        1,
        17
      ],
      "numa_node": 0
    },
    {
      "cpu": 2,
      "thread_siblings": [
        10,
        26
      ],
      "numa_node": 1
    },
    {
      "cpu": 0,
      "thread_siblings": [
        0,
        16
      ],
      "numa_node": 0
    },
    {
      "cpu": 5,
      "thread_siblings": [
        13,
        29
      ],
      "numa_node": 1
    },
    {
      "cpu": 7,
      "thread_siblings": [
        15,
        31
      ],
      "numa_node": 1
    },
    {
      "cpu": 7,
      "thread_siblings": [
        7,
        23
      ],
      "numa_node": 0

    }
  ]
}
```
CHAPTER 2. HARDWARE REQUIREMENTS

```json
{
  "cpu": 6,
  "thread_siblings": [14, 30],
  "numa_node": 1
},
{
  "cpu": 3,
  "thread_siblings": [3, 19],
  "numa_node": 0
},
{
  "cpu": 2,
  "thread_siblings": [2, 18],
  "numa_node": 0
}
],
"ram": [
{  
  "size_kb": 66980172,
  "numa_node": 0
},
{  
  "size_kb": 67108864,
  "numa_node": 1
}
],
"nics": [
{  
  "name": "ens3f1",
  "numa_node": 1
},
{  
  "name": "ens3f0",
  "numa_node": 1
},
{  
  "name": "ens2f0",
  "numa_node": 0
},
{  
  "name": "ens2f1",
  "numa_node": 0
},
{  
  "name": "ens1f1",
  "numa_node": 0
},
```
2.3. REVIEW BIOS SETTINGS

The following listing describes the required BIOS settings for NFV:

- **C3 Power State**: Disabled.
- **C6 Power State**: Disabled.
- **MLC Streamer**: Enabled.
- **MLC Spacial Prefetcher**: Enabled.
- **DCU Data Prefetcher**: Enabled.
- **DCA**: Enabled.
- **CPU Power and Performance**: Performance.
- **Memory RAS and Performance Config → NUMA Optimized**: Enabled.
- **Turbo Boost**: Disabled.
- **VT-d**: Enabled for Intel cards if VFIO functionality is needed.
CHAPTER 3. SOFTWARE REQUIREMENTS

This section describes the supported configurations and drivers, and subscription details necessary for NFV.

3.1. REGISTERING AND ENABLING REPOSITORIES

To install Red Hat OpenStack Platform, you must register Red Hat OpenStack Platform director using the Red Hat Subscription Manager, and subscribe to the required channels. See Registering your system for details.

Procedure

1. Register your system with the Content Delivery Network, entering your Customer Portal user name and password when prompted.

   [stack@director ~]$ sudo subscription-manager register

2. Find the entitlement pool ID for Red Hat OpenStack Platform director. For example:

   [stack@director ~]$ sudo subscription-manager list --available --all --matches="Red Hat OpenStack"

   Subscription Name:   Name of SKU
   Provides:            Red Hat Single Sign-On
   Provides:            Red Hat Enterprise Linux Workstation
   Provides:            Red Hat CloudForms
   Provides:            Red Hat OpenStack
   Provides:            Red Hat Software Collections (for RHEL Workstation)
   Provides:            Red Hat Virtualization
   SKU:                 SKU-Number
   Contract:            Contract-Number
   Pool ID:             {Pool-ID}-123456
   Provides Management: Yes
   Available:           1
   Suggested:           1
   Service Level:       Support-level
   Service Type:        Service-Type
   Subscription Type:   Sub-type
   Ends:                End-date
   System Type:         Physical

3. Locate the Pool ID value and attach the Red Hat OpenStack Platform 14 entitlement.

   [stack@director ~]$ sudo subscription-manager attach --pool={Pool-ID}-123456

4. Disable the default repositories.

   subscription-manager repos --disable=* 

5. Enable required repositories for Red Hat OpenStack Platform with network functions virtualization (NFV).

   sudo subscription-manager repos \

   subscription-manager repos

   subscription-manager repos
6. Perform an update on your system to make sure you have the latest base system packages.

```
[stack@director ~] $ sudo yum update -y
[stack@director ~] $ sudo reboot
```

**NOTE**
To register your overcloud nodes, see [Ansible Based Registration](#).

### 3.2. SUPPORTED CONFIGURATIONS FOR NFV DEPLOYMENTS

Red Hat OpenStack Platform supports the following network functions virtualization (NFV) deployments using director:

- Single root I/O virtualization (SR-IOV)
- Open vSwitch with Data Plane Development Kit (OVS-DPDK)

Additionally, you can deploy Red Hat OpenStack Platform with any of the following features:

- Composable roles
- Hyper-converged infrastructure (limited support)
- Real-time KVM
- OVS hardware offload (Technology preview)

**NOTE**
Red Hat’s embedded OpenDaylight SDN solution is being deprecated in OpenStack Platform (OSP) 14. Red Hat will continue to provide support and bug fixes for OpenDaylight, with all support ending with the OSP 13 lifecycle (June 27, 2021).

### 3.3. SUPPORTED DRIVERS

For a complete list of supported drivers, see [Component, Plug-In, and Driver Support in Red Hat OpenStack Platform](#).

For a list of NICs tested for NFV deployments with Red Hat OpenStack, see [Tested NICs](#).

### 3.4. COMPATIBILITY WITH THIRD PARTY SOFTWARE

For a complete list of products and services tested, supported, and certified to perform with Red Hat technologies (Red Hat OpenStack Platform), see [Third Party Software compatible with Red Hat OpenStack Platform](#). You can filter the list by product version and software category.
For a complete list of products and services tested, supported, and certified to perform with Red Hat technologies (Red Hat Enterprise Linux), see Third Party Software compatible with Red Hat Enterprise Linux. You can filter the list by product version and software category.
CHAPTER 4. NETWORK CONSIDERATIONS

The undercloud host requires at least the following networks:

- **Provisioning network** - Provides DHCP and PXE boot functions to help discover bare-metal systems for use in the overcloud.

- **External network** - A separate network for remote connectivity to all nodes. The interface connecting to this network requires a routable IP address, either defined statically or dynamically through an external DHCP service.

The minimal overcloud network configuration includes:

- **Single NIC configuration** - One NIC for the provisioning network on the native VLAN and tagged VLANs that use subnets for the different overcloud 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 the different overcloud network types.

- **Multiple NIC configuration** - Each NIC uses a subnet for a different overcloud network type.

For more information on the networking requirements, see [Networking requirements](#).
CHAPTER 5. PLANNING AN SR-IOV DEPLOYMENT

Optimize single root I/O virtualization (SR-IOV) deployments for NFV by setting individual parameters based on your Compute node hardware.

See Discovering your NUMA node topology to evaluate your hardware impact on the SR-IOV parameters.

5.1. HARDWARE PARTITIONING FOR AN SR-IOV DEPLOYMENT

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

A typical topology includes 14 cores per NUMA node on dual socket Compute nodes. Both hyper-threading (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 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 takes care of the boot parameters and any OpenStack modifications based on the list of CPUs to isolate.

5.2. TOPOLOGY OF AN NFV SR-IOV DEPLOYMENT

The following image has two virtual network functions (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 Data Plane Development Kit (DPDK) to ensure high availability (VNFs bond the data plane interfaces using the DPDK library). The image also has two redundant provider networks. The Compute node has two regular NICs bonded together and shared between the VNF management and the Red Hat OpenStack Platform API management.
The image shows a VNF that leverages DPDK at an application level and has access to single root I/O virtualization (SR-IOV) virtual functions (VFs) and physical functions (PFs), together for better availability or performance (depending on the fabric configuration). DPDK improves performance, while the VF/PF DPDK bonds provide support for failover (availability). The VNF vendor must ensure their DPDK poll mode driver (PMD) supports the SR-IOV card that is being exposed as a VF/PF. The management network uses Open vSwitch (OVS) so the VNF sees a "mgmt" network device using the standard virtIO drivers. Operators can use that device to initially connect to the VNF and ensure that their DPDK application bonds properly the two VF/PFs.

5.2.1. NFV SR-IOV without HCI

You can see the topology for single root I/O virtualization (SR-IOV) without hyper-converged infrastructure (HCI) for the NFV use case in the image below. It consists of compute and controller nodes with 1 Gbps NICs, and the Director node.
CHAPTER 6. DEPLOYING SR-IOV TECHNOLOGIES

Single root I/O virtualization (SR-IOV) allows near bare metal performance by allowing instances from OpenStack direct access to a shared PCIe resource through virtual resources.

6.1. PREREQUISITES

- Install and configure the undercloud before deploying the overcloud. See the Director Installation and Usage Guide for details.

**NOTE**

Do not manually edit values in `/etc/tuned/cpu-partitioning-variables.conf` that are modified by Director heat templates.

6.2. CONFIGURING SR-IOV

**NOTE**

The CPU assignments, memory allocation and NIC configurations of the following examples may differ from your topology and use case.

1. Generate the built-in `ComputeSriov` to define nodes in the OpenStack cluster that will run `NeutronSriovAgent`, `NeutronSriovHostConfig` and default compute services.

   ```bash
   # openstack overcloud roles generate \
   -o /home/stack/templates/roles_data.yaml \
   Controller ComputeSriov
   ```

2. Include the `neutron-sriov.yaml` and `roles_data.yaml` files when generating `overcloud_images.yaml` so that SR-IOV containers are prepared.

   ```bash
   SERVICES=\
   /usr/share/openstack-tripleo-heat-templates/environments/services
   openstack overcloud container image prepare \
   --namespace=registry.access.redhat.com/rhosp14 \
   --push-destination=192.168.24.1:8787 \
   --prefix=openstack- \
   --tag-from-label {version}-{release} \
   -e ${SERVICES}/neutron-sriov.yaml \
   --roles-file /home/stack/templates/roles_data.yaml \
   --output-env-file=/home/stack/templates/overcloud_images.yaml \
   --output-images-file=/home/stack/local_registry_images.yaml
   ```

**NOTE**

The push-destination IP address is the address that you previously set with the `local_ip` parameter in the `undercloud.conf` configuration file.

For more information on container image preparation, see Director Installation and Usage.
3. To apply the `KernelAgs` and `TunedProfile` parameters, include the `host-config-and-reboot.yaml` file from `/usr/share/openstack-tripleo-heat-templates/environments` to your deployment script.

```bash
openstack overcloud deploy --templates \
... \ 
-e /usr/share/openstack-tripleo-heat-templates/environments/host-config-and-reboot.yaml \ 
... 
```

4. Configure the parameters for the SR-IOV nodes under `parameter_defaults` in accordance with the needs of your cluster, and the configuration of your hardware. These settings are typically added to the `network-environment.yaml` file.

```yaml
NeutronNetworkType: 'vlan'
NeutronNetworkVLANRanges:
  - tenant:22:22
  - tenant:25:25
NeutronTunnelTypes: "
```

5. In the same file, configure role specific parameters for SR-IOV compute nodes.

```
NOTE
The `NeutronSriovNumVFs` parameter will soon be deprecated in favor of the `numvfs` attribute in the network configuration templates. Red Hat does not support modification of the `NeutronSriovNumVFs` parameter, nor the `numvfs` parameter, after deployment. Changing either parameter within a running environment is known to cause a permanent outage for all running instances which have an SR-IOV port on that PF. Unless you hard reboot these instances, the SR-IOV PCI device will not be visible to the instance.

```
```yaml
ComputeSriovParameters:
  IsolCpusList: "1-19,21-39"
  KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt
              intel_iommu=on isolcpus=1-19,21-39"
  TunedProfileName: "cpu-partitioning"
  NeutronBridgeMappings:
    - tenant:br-link0
  NeutronPhysicalDevMappings:
    - tenant:p7p1
    - tenant:p7p2
  NeutronSriovNumVFs:
    - p7p1:5
    - p7p2:5
  NovaPCIPassthrough:
    - devname: "p7p1"
      physical_network: "tenant"
    - devname: "p7p2"
      physical_network: "tenant"
  NovaVcpuPinSet: '1-19,21-39'
  NovaReservedHostMemory: 4096
```
6. Configure the SR-IOV enabled interfaces in the `compute.yaml` network configuration template. Ensure the interfaces are configured as standalone NICs for the purposes of creating SR-IOV virtual functions (VF):

   ```yaml
   - type: interface
     name: p7p3
     mtu: 9000
     use_dhcp: false
     defroute: false
     nm_controlled: true
     hotplug: true
   
   - type: interface
     name: p7p4
     mtu: 9000
     use_dhcp: false
     defroute: false
     nm_controlled: true
     hotplug: true
   
   7. Ensure that the list of default filters includes the value `AggregateInstanceExtraSpecsFilter`.

      ```
      NovaSchedulerDefaultFilters:
      ```

8. Deploy the overcloud.

   ```
   templates_home="/usr/share/openstack-tripleo-heat-templates"
   custom_templates="/home/stack/templates"

   openstack overcloud deploy --templates \
   -r ${custom_templates}/roles_data.yaml \
   -e ${templates_home}/environments/host-config-and-reboot.yaml \
   -e ${templates_home}/environments/services/neutron-sriov.yaml \
   -e ${custom_templates}/network-environment.yaml
   ```

### 6.3. Configuring Hardware Offload (Technology Preview)

Open vSwitch (OVS) hardware offload is a technology preview and not recommended for production deployments. For more information about technology preview features, see [Scope of Coverage Details](#).

OVS hardware offload takes advantage of single root I/O virtualization (SR-IOV), therefore some of the same configuration steps apply.

#### 6.3.1. Open vSwitch Hardware Offload

To enable OVS hardware offload, complete the following steps.

**Procedure**

1. Generate the `ComputeSriov` role:
openstack overcloud roles generate -o roles_data.yaml Controller ComputeSriov

2. Configure the `physical_network` parameter to match your environment.

- For VLAN, set the `physical_network` parameter to the name of the network you create in neutron after deployment. This value should also be in `NeutronBridgeMappings`.
- For VXLAN, set the `physical_network` parameter to the string value `null`.
- Ensure the `OvsHwOffload` parameter under role specific parameters has a value of `true`.

Example:

```yaml
parameter_defaults:
  ComputeSriovParameters:
    IsolCpusList: 2-9,21-29,11-19,31-39
    KernelArgs: \"default_hugepagesz=1GB hugepagesz=1G hugepages=128 iommu=on \"
    OvsHwOffload: true
    TunedProfileName: \"cpu-partitioning\"
  NeutronBridgeMappings:
    - tenant:br-tenant
  NeutronPhysicalDevMappings:
    - tenant:p7p1
    - tenant:p7p2
  NovaPCIPassthrough:
    - devname: \"p7p1\"
      physical_network: \"null\"
    - devname: \"p7p2\"
      physical_network: \"null\"
  NovaReservedHostMemory: 4096
  NovaVcpuPinSet: 1-9,21-29,11-19,31-39

3. Ensure that the list of default filters includes the value `NUMATopologyFilter`:

```yaml
NovaSchedulerDefaultFilters:
  \[
  \'RetryFilter',\'AvailabilityZoneFilter',\'ComputeFilter',\'ComputeCapabilitiesFilter',\'ImagePropertiesFilter',\'ServerGroupAntiAffinityFilter',\'ServerGroupAffinityFilter',\'PciPassthroughFilter',\'NUMATopologyFilter'\]
```

4. Configure one or more network interfaces intended for hardware offload in the `compute-sriov.yaml` configuration file:

**NOTE**

Do not use the `NeutronSriovNumVFs` parameter when configuring Open vSwitch hardware offload. The number of virtual functions will be specified using the `numvfs` parameter in a network configuration file used by `os-net-config`. Red Hat does not support modifying the `numvfs` setting during update or redeployment.

```yaml
- type: ovs_bridge
  name: br-tenant
  mtu: 9000
  members:
```
- type: sriov_pf
  name: p7p1
  numvfs: 5
  mtu: 9000
  primary: true
  promisc: true
  use_dhcp: false
  link_mode: switchdev

NOTE

Do not configure Mellanox network interfaces as a nic-config interface type `ovs-vlan` because this prevents tunnel endpoints such as VXLAN from passing traffic due to driver limitations.

5. Include the following files during the deployment of the overcloud:

   - ovs-hw-offload.yaml
   - host-config-and-reboot.yaml

      TEMPLATES_HOME="/usr/share/openstack-tripleo-heat-templates"
      CUSTOM_TEMPLATES="/home/stack/templates"

      openstack overcloud deploy --templates \
      -r ${CUSTOM_TEMPLATES}/roles_data.yaml \
      -e ${TEMPLATES_HOME}/environments/ovs-hw-offload.yaml \
      -e ${TEMPLATES_HOME}/environments/host-config-and-reboot.yaml \
      -e ${CUSTOM_TEMPLATES}/network-environment.yaml

6.3.2. Verification

1. Confirm that a pci device has its mode configured as switchdev:

   ```bash
   # devlink dev eswitch show pci/0000:03:00.0
   pci/0000:03:00.0: mode switchdev inline-mode none encap enable
   ```

2. Confirm offload is enabled in OVS:

   ```bash
   # ovs-vsctl get Open_vSwitch . other_config:hw-offload
   "true"
   ```

3. Confirm hardware offload is enabled on the NIC:

   ```bash
   # ethtool -k $NIC | grep tc-offload
   hw-tc-offload: on
   ```

6.4. DEPLOYING AN INSTANCE FOR SR-IOV

It is recommended to use host aggregates to separate high performance compute hosts. For information on creating host aggregates and associated flavors for scheduling see Creating host aggregates.
NOTE
You should use host aggregates to separate CPU pinned instances from unpinned instances. Instances that do not use CPU pinning do not respect the resourcing requirements of instances that use CPU pinning.

Deploy an instance for single root I/O virtualization (SR-IOV) by performing the following steps:

1. Create a flavor.

```bash
# openstack flavor create <flavor> --ram <MB> --disk <GB> --vcpus <#>
```

2. Create the network.

```bash
openstack network create net1 --provider-physical-network tenant --provider-network-type vlan --provider-segment <VLAN-ID>
```

3. Create the port.

   - Use vnic-type `direct` to create an SR-IOV virtual function (VF) port.
     ```bash
     # openstack port create --network net1 --vnic-type direct sriov_port
     ```
   
   - Use the following to create a virtual function with hardware offload.
     ```bash
     openstack port create --network net1 --vnic-type direct --binding-profile '{"capabilities": ["switchdev"]}' sriov_hwoffload_port
     ```
   
   - Use vnic-type `direct-physical` to create an SR-IOV PF port.
     ```bash
     # openstack port create --network net1 --vnic-type direct-physical sriov_port
     ```

4. Deploy an instance

```bash
# openstack server create --flavor <flavor> --image <image> --nic port-id=<id> <instance name>
```

### 6.5. CREATING HOST AGGREGATES

It is recommended to deploy guests using cpu pinning and hugepages for increased performance. You can schedule high performance instances on a subset of hosts by matching aggregate metadata with flavor metadata.

#### 6.5.1. Procedure

1. Ensure that the `AggregateInstanceExtraSpecsFilter` value is included in the `scheduler_default_filters` parameter in the `nova.conf` configuration file. This configuration can be set through the heat parameter `NovaSchedulerDefaultFilters` under role-specific parameters before deployment.

   ```yaml
   ComputeOvsDpdkSriovParameters:
   NovaSchedulerDefaultFilters: ['AggregateInstanceExtraSpecsFilter',
   ```

NOTE
This parameter can be added to heat templates and the original deployment script re-run to add this to the configuration of an exiting cluster.

2. Create an aggregate group for single root I/O virtualization (SR-IOV), and add relevant hosts. Define metadata, for example, \texttt{sriov=true}, that matches defined flavor metadata.

```bash
# openstack aggregate create sriov_group
# openstack aggregate add host sriov_group compute-sriov-0.localdomain
# openstack aggregate set sriov_group sriov=true
```

3. Create a flavor.

```bash
# openstack flavor create <flavor> --ram <MB> --disk <GB> --vcpus <#>
```

4. Set additional flavor properties. Note that the defined metadata, \texttt{sriov=true}, matches the defined metadata on the SR-IOV aggregate.

```bash
openstack flavor set --property sriov=true --property hw:cpu_policy=dedicated --property hw:mem_page_size=1GB <flavor>
```
CHAPTER 7. PLANNING YOUR OVS-DPDK DEPLOYMENT

To optimize your Open vSwitch with Data Plane Development Kit (OVS-DPDK) deployment for NFV, you should understand how OVS-DPDK uses the Compute node hardware (CPU, NUMA nodes, memory, NICs) and the considerations for determining the individual OVS-DPDK parameters based on your Compute node.

See NFV performance considerations for a high-level introduction to CPUs and NUMA topology.

7.1. OVS-DPDK WITH CPU PARTITIONING AND NUMA TOPOLOGY

OVS-DPDK partitions the hardware resources for host, guests, and OVS-DPDK. The OVS-DPDK Poll Mode Drivers (PMDs) run DPDK active loops, which require dedicated cores. This means a list of CPUs and Huge Pages are dedicated to OVS-DPDK.

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

NOTE

DPDK PMD threads must be reserved on both NUMA nodes even if a NUMA node does not have an associated DPDK NIC.

OVS-DPDK performance also depends on reserving a block of memory local to the NUMA node. Use NICs associated with the same NUMA node that you use for memory and CPU pinning. Ensure that both interfaces in a bond are from NICs on the same NUMA node.

7.2. OVERVIEW OF WORKFLOWS AND DERIVED PARAMETERS
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.

You can use the OpenStack Workflow (mistral) service to derive parameters based on the capabilities of your available bare-metal nodes. OpenStack workflows use a .yaml file to define a set of tasks and actions to perform. You can use a pre-defined workbook, **derive_params.yaml**, in the `tripleo-common/workbooks/` directory. This workbook provides workflows to derive each supported parameter from the results retrieved from Bare Metal introspection. The **derive_params.yaml** workflows use the formulas from `tripleo-common/workbooks/derive_params_formulas.yaml` to calculate the derived parameters.

**NOTE**

You can modify the formulas in **derive_params_formulas.yaml** to suit your environment.

The **derive_params.yaml** workbook assumes all nodes for a given composable role have the same hardware specifications. The workflow considers the flavor-profile association and nova placement scheduler to match nodes associated with a role and uses the introspection data from the first node that matches the role.

See Troubleshooting Workflows and Executions for details on OpenStack workflows.

You can use the `-p` or `--plan-environment-file` option to add a custom **plan_environment.yaml** file to the `openstack overcloud deploy` command. The custom **plan_environment.yaml** file provides the list of workbooks and any input values to pass into the workbook. The triggered workflows merge the derived parameters back into the custom **plan_environment.yaml**, where they are available for the overcloud deployment. You can use these derived parameter results to prepare your overcloud images.

See Plan Environment Metadata for details on how to use the `--plan-environment-file` option in your deployment.

### 7.3. DERIVED OVS-DPDK PARAMETERS

The workflows in **derive_params.yaml** derive the DPDK parameters associated with the matching role that uses the ComputeNeutronOvsDpdk service.

The following is the list of parameters the workflows can automatically derive for OVS-DPDK:

- IsolCpusList
- KernelArgs
- NovaReservedHostMemory
- NovaVcpuPinSet
- OvsDpdkCoreList
- OvsDpdkSocketMemory
- OvsPmdCoreList
The `OvsDpdkMemoryChannels` parameter cannot be derived from the introspection memory bank data since the format of memory slot names are not consistent across different hardware environments.

In most cases, `OvsDpdkMemoryChannels` should be 4 (default). Use your hardware manual to determine the number of memory channels per socket and use this value to override the default.

See Section 8.1, “Deriving DPDK parameters with workflows” for configuration details.

### 7.4. OVERVIEW OF MANUALLY CALCULATED OVS-DPDK PARAMETERS

This section describes how Open vSwitch with Data Plane Development Kit (OVS-DPDK) uses parameters within the director `network_environment.yaml` HEAT templates to configure the CPU and memory for optimum performance. Use this information to evaluate the hardware support on your Compute nodes and how best to partition that hardware to optimize your OVS-DPDK deployment.

**NOTE**

You do not need to manually calculate these parameters if you use the `derived_parameters.yaml` workflow to generate these values automatically. See Overview of workflows and derived parameters

**NOTE**

Always pair CPU sibling threads (logical CPUs) together for the physical core when allocating CPU cores.

See Discovering your NUMA node topology to determine the CPU and NUMA nodes on your Compute nodes. You use this information to map CPU and other parameters to support the host, guest instance, and OVS-DPDK process needs.

### 7.4.1. CPU parameters

OVS-DPDK uses the following CPU partitioning parameters:

**OvsPmdCoreList**

Provides the CPU cores that are used for the DPDK poll mode drivers (PMD). Choose CPU cores that are associated with the local NUMA nodes of the DPDK interfaces. `OvsPmdCoreList` is used for the `pmd-cpu-mask` value in Open vSwitch.

- Pair the sibling threads together.
- Exclude all cores from the `OvsDpdkCoreList`
- Avoid allocating the logical CPUs (both thread siblings) of the first physical core on both NUMA nodes as these should be used for the `OvsDpdkCoreList` parameter.
- Performance depends on the number of physical cores allocated for this PMD Core list. On the NUMA node which is associated with DPDK NIC, allocate the required cores.
- For NUMA nodes with a DPDK NIC:
  - Determine the number of physical cores required based on the performance requirement and include all the sibling threads (logical CPUs) for each physical core.
For NUMA nodes without DPDK NICs:

- Allocate the sibling threads (logical CPUs) of one physical core (excluding the first physical core of the NUMA node). You need a minimal DPDK poll mode driver on the NUMA node even without DPDK NICs present to avoid failures in creating guest instances.

**NOTE**

DPDK PMD threads must be reserved on both NUMA nodes even if a NUMA node does not have an associated DPDK NIC.

**NovaVcpuPinSet**

Sets cores for CPU pinning. The Compute node uses these cores for guest instances. **NovaVcpuPinSet** is used as the *vcpu_pin_set* value in the *nova.conf* file.

- Exclude all cores from the **OvsPmdCoreList** and the **OvsDpdkCoreList**.
- Include all remaining cores.
- Pair the sibling threads together.

**NovaComputeCpuSharedSet**

Sets the cores to be used for emulator threads. This will define the value of the *nova.conf* parameter *cpu_shared_set*. The recommended value for this parameter matches the value set for **OvsDpdkCoreList**.

**IsolCpusList**

A set of CPU cores isolated from the host processes. This parameter is used as the *isolated_cores* value in the *cpu-partitioning-variable.conf* file for the *tuned-profiles-cpu-partitioning* component.

- Match the list of cores in **OvsPmdCoreList** and **NovaVcpuPinSet**.
- Pair the sibling threads together.

**OvsDpdkCoreList**

Provides CPU cores for non data path OVS-DPDK processes, such as handler and revalidator threads. This parameter has no impact on overall data path performance on multi-NUMA node hardware. This parameter is used for the *dpdk-lcore-mask* value in Open vSwitch, and these cores are shared with the host.

- Allocate the first physical core (and sibling thread) from each NUMA node (even if the NUMA node has no associated DPDK NIC).
- These cores must be mutually exclusive from the list of cores in **OvsPmdCoreList** and **NovaVcpuPinSet**.

### 7.4.2. Memory parameters

OVS-DPDK uses the following memory parameters:

**OvsDpdkMemoryChannels**
Maps memory channels in the CPU per NUMA node. The **OvsDpdkMemoryChannels** parameter is used by Open vSwitch as the `other_config:dpdk-extra=-n <value>" value.

- 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.

**NovaReservedHostMemory**

Reserves memory in MB for tasks on the host. This value is used by the Compute node as the `reserved_host_memory_mb` value in `nova.conf`.

- Use the static recommended value of 4096 MB.

**OvsDpdkSocketMemory**

Specifies the amount of memory in MB to pre-allocate from the hugepage pool, per NUMA node. This value is used by Open vSwitch as the `other_config:dpdk-socket-mem` value.

- Provide as a comma-separated list. Calculate the `OvsDpdkSocketMemory` value from the MTU value of each NIC on the NUMA node.
- For a NUMA node without a DPDK NIC, use the static recommendation of 1024 MB (1GB)
- The following equation approximates the value for `OvsDpdkSocketMemory`:
  - `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 512 MB as buffer. Round the value up to a multiple of 1024.

**Sample Calculation - MTU 2000 and MTU 9000**

DPDK NICs dpdk0 and dpdk1 are on the same NUMA node 0 and configured with MTUs 9000 and 2000 respectively. The sample calculation to derive the memory required is as follows:

1. Round off the MTU values to the nearest 1024 bytes.
   - The MTU value of 9000 becomes 9216 bytes.
   - The MTU value of 2000 becomes 2048 bytes.

2. Calculate the required memory for each MTU value based on these rounded byte values.
   - Memory required for 9000 MTU = (9216 + 800) * (4096*64) = 2625634304
   - Memory required for 2000 MTU = (2048 + 800) * (4096*64) = 746586112

3. Calculate the combined total memory required, in bytes.
   - 2625634304 + 746586112 + 536870912 = 3909091328 bytes.
This calculation represents (Memory required for MTU of 9000) + (Memory required for MTU of 2000) + (512 MB buffer).

4. Convert the total memory required into MB.

\[ \frac{3909091328}{(1024\times1024)} = 3728 \text{ MB} \]

5. Round this value up to the nearest 1024.

3724 MB rounds up to 4096 MB.

6. Use this value to set \texttt{OvsDpdkSocketMemory}.

\texttt{OvsDpdkSocketMemory: "4096,1024"}

Sample Calculation - MTU 2000

DPDK NICs dpdk0 and dpdk1 are on the same NUMA node 0 and configured with MTUs 2000 and 2000 respectively. The sample calculation to derive the memory required is as follows:

1. Round off the MTU values to the nearest 1024 bytes.

   The MTU value of 2000 becomes 2048 bytes.

2. Calculate the required memory for each MTU value based on these rounded byte values.

   \[ \text{Memory required for 2000 MTU} = (2048 + 800) \times (4096\times64) = 746586112 \]

3. Calculate the combined total memory required, in bytes.

   \[ 746586112 + 536870912 = 1283457024 \text{ bytes.} \]

   This calculation represents (Memory required for MTU of 2000) + (512 MB buffer).

4. Convert the total memory required into MB.

   \[ \frac{1283457024}{(1024\times1024)} = 1224 \text{ MB} \]

5. Round this value up to the nearest 1024.

   1224 MB rounds up to 2048 MB.

6. Use this value to set \texttt{OvsDpdkSocketMemory}.

   \texttt{OvsDpdkSocketMemory: "2048,1024"}

7.4.3. Networking parameters

\texttt{NeutronDpdkDriverType}

Sets the driver type used by DPDK. Use the default of \texttt{vfio-pci}.

\texttt{NeutronDatapathType}
Datapath type for OVS bridges. DPDK uses the default value of `netdev`.

**NeutronVhostuserSocketDir**
Sets the vhost-user socket directory for OVS. Use `/var/lib/vhost_sockets` for vhost client mode.

### 7.4.4. Other parameters

**NovaSchedulerDefaultFilters**
Provides an ordered list of filters that the Compute node uses to find a matching Compute node for a requested guest instance.

**VhostuserSocketGroup**
Sets the vhost-user socket directory group. The default value is `qemu`. `VhostuserSocketGroup` should be set to `hugetlbfs` so that the `ovs-vswitchd` and `qemu` processes can access the shared hugepages and unix socket used to configure the virtio-net device. This value is role specific and should be applied to any role leveraging OVS-DPDK.

**KernelArgs**
Provides multiple kernel arguments to `/etc/default/grub` for the Compute node at boot time. Add the following based on your configuration:

- **hugepagesz**: Sets the size of the huge pages on a CPU. This value can vary depending on the CPU hardware. Set to 1G for OVS-DPDK deployments (`default_hugepagesizez=1GB hugepagesizez=1G`). Check for the `pdpe1gb` CPU flag to ensure your CPU supports 1G.

  ```
  lshw -class processor | grep pdpe1gb
  ```

- **hugepages count**: Sets the number of huge pages available. This value depends on the amount of host memory available. Use most of your available memory (excluding `NovaReservedHostMemory`). You must also configure the huge pages count value within the OpenStack flavor associated with your Compute nodes.

- **iommu**: For Intel CPUs, add “`intel_iommu=on iommu=pt`”
- **isolcpus**: Sets the CPU cores to be tuned. This value matches `IsolCpusList`.

### 7.4.5. Instance extra specifications

Before deploying instances in an NFV environment, create a flavor that will utilize CPU pinning, huge pages, and emulator thread pinning.

**hw:cpu_policy**
Set the value of this parameter to `dedicated`, so that a guest will use pinned CPUs. Instances created from a flavor with this parameter set will have an effective overcommit ratio of 1:1. The default is `shared`.

**hw:mem_page_size**
Set the value of this parameter to a valid string of a specific value with standard suffix (For example, `4KB`, `8MB`, or `1GB`). Use 1GB to match the hugepagesz boot parameter. The number of huge pages available for the virtual machines is the boot parameter minus the `OvsDpdkSocketMemory`. Other valid parameter values include:

- `small` (default) - The smallest page size is used
- `large` - Only use large page sizes. (2MB or 1GB on x86 architectures)
- any - The compute driver may attempt large pages, but default to small if none available.

**hw:emulator_threads_policy**

Set the value of this parameter to `share` so that emulator threads are locked to CPUs that you’ve identified in the heat parameter, `NovaComputeCpuSharedSet`. If an emulator thread is running on a vCPU being used for the poll mode driver (PMD) or real-time processing, you can experience packet loss or missed deadlines.

### 7.5. TWO NUMA NODE EXAMPLE OVS-DPDK DEPLOYMENT

This sample Compute node includes two NUMA nodes as follows:

- **NUMA 0** has cores 0-7. 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 (NIC1 on NUMA 0 and NIC2 on NUMA 1).

![NUMA node diagram](image)

#### NOTE

Reserve the first physical cores (both thread pairs) on each NUMA node (0,1 and 8,9) for non data path DPDK processes (`OvsDpdkCoreList`).

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 there is no DPDK enabled on the NIC for that NUMA node. The remaining cores (not reserved for `OvsDpdkCoreList`) are allocated for guest instances. The resulting parameter settings are:

- **OvsPmdCoreList**: “2,3,10,11”
  - **NovaVcpuPinSet**: “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 there is no DPDK enabled on the NIC for that NUMA node. The remaining cores (not reserved for `OvsDpdkCoreList`) are allocated for guest instances. The resulting
parameter settings are:

```
OvsPmdCoreList: "2,3,4,5,10,11"
NovaVcpuPinSet: "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 there is no DPDK enabled on the NIC for that NUMA node. The remaining cores (not reserved for `OvsDpdkCoreList`) are allocated for guest instances. The resulting parameter settings are:

```
OvsPmdCoreList: "2,3,10,11"
NovaVcpuPinSet: "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 there is no DPDK enabled on the NIC for that NUMA node. The remaining cores (not reserved for `OvsDpdkCoreList`) are allocated for guest instances. The resulting parameter settings are:

```
OvsPmdCoreList: "2,3,10,11,12,13"
NovaVcpuPinSet: "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 (not reserved for `OvsDpdkCoreList`) are allocated for guest instances. The resulting parameter settings are:

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

**NOTE**

Red Hat recommends using 1 physical core per NUMA node.

### 7.6. TOPOLOGY OF AN NFV OVS-DPDK DEPLOYMENT

This sample deployment shows an Open vSwitch with Data Plane Development Kit (OVS-DPDK) configuration and consists of two virtual network functions (VNFs) each with two interfaces: the management interface represented by `mgt` and the data plane interface. In the OVS-DPDK deployment, the VNFs run with inbuilt DPDK that supports the physical interface. OVS-DPDK takes care of the bonding at the vSwitch level. In an OVS-DPDK deployment, it is recommended that you do **not** mix kernel and OVS-DPDK NICs as it can lead to performance degradation. To separate the management (`mgt`) network, connected to the Base provider network for the virtual machine, you need to ensure you have additional NICs. The Compute node consists of two regular NICs for the OpenStack API management that can be reused by the Ceph API but cannot be shared with any OpenStack tenant.
NFV OVS-DPDK topology

The following image shows the topology for OVS_DPK for the NFV use case. It consists of Compute and Controller nodes with 1 or 10 Gbps NICs, and the Director node.
CHAPTER 8. CONFIGURING AN OVS-DPDK DEPLOYMENT

This section deploys DPDK with Open vSwitch (OVS-DPDK) within the Red Hat OpenStack Platform environment. The overcloud usually consists of nodes in predefined roles such as Controller nodes, Compute nodes, and different storage node types. Each of these default roles contains a set of services defined in the core Heat templates on the director node.

You must install and configure the undercloud before you can deploy the overcloud. See the Director Installation and Usage Guide for details.

**IMPORTANT**

You must determine the best values for the OVS-DPDK parameters that you set in the network-environment.yaml file to optimize your OpenStack network for OVS-DPDK.

**NOTE**

Do not edit or change isolated_cores or other values in etc/tuned/cpu-partitioning-variables.conf that are modified by these director Heat templates.

8.1. DERIVING DPDK PARAMETERS WITH WORKFLOWS

**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.

See Section 7.2, “Overview of workflows and derived parameters” for an overview of the Mistral workflow for DPDK.

**Prerequisites**

You must have Bare Metal introspection, including hardware inspection extras (inspection_extras) enabled to provide the data retrieved by this workflow. Hardware inspection extras are enabled by default. See Inspecting the Hardware of Nodes.

**Define the Workflows and Input Parameters for DPDK**

The following lists the input parameters you can provide to the OVS-DPDK workflows:

- **num_phy_cores_per_numa_node_for_pmd**
  This input parameter specifies the required minimum number of cores for the NUMA node associated with the DPDK NIC. One physical core is assigned for the other NUMA nodes not associated with DPDK NIC. Ensure that this parameter is set to 1.

- **huge_page_allocation_percentage**
  This input parameter specifies the required percentage of total memory (excluding NovaReservedHostMemory) that can be configured as huge pages. The KernelArgs parameter is derived using the calculated huge pages based on the huge_page_allocation_percentage specified. Ensure that this parameter is set to 50.
The workflows use these input parameters along with the bare-metal introspection details to calculate appropriate DPDK parameter values.

To define the workflows and input parameters for DPDK:

1. Copy the `/usr/share/openstack-tripleo-heat-templates/plan-samples/plan-environment-derived-params.yaml` file to a local directory and set the input parameters to suit your environment.

   ```yaml
   workflow_parameters:
   tripleo.derive_params.v1.derive_parameters:
   # DPDK Parameters
   # Specifies the minimum number of CPU physical cores to be allocated for DPDK
   # PMD threads. The actual allocation will be based on network config, if
   # the a DPDK port is associated with a numa node, then this configuration
   # will be used, else 1.
   num_phy_cores_per_numa_node_for_pmd: 1
   # Amount of memory to be configured as huge pages in percentage. Out the
   # total available memory (excluding the NovaReservedHostMemory), the
   # specified percentage of the remaining is configured as huge pages.
   huge_page_allocation_percentage: 50
   ```

2. Run the `openstack overcloud deploy` command and include the following:

   - The `update-plan-only` option
   - The role file and all environment files specific to your environment
   - The `plan-environment-derived-parms.yaml` file with the `--plan-environment-file` optional argument

   ```bash
   $ openstack overcloud deploy --templates --update-plan-only \
   -r /home/stack/roles_data.yaml \
   -e /home/stack/<environment-file> \
   ... #repeat as necessary ... \
   -p /home/stack/plan-environment-derived-params.yaml
   ```

The output of this command shows the derived results, which are also merged into the `plan-environment.yaml` file.
isolcpus=1-7,17-23,9-15,25-31
NovaReservedHostMemory: 4096
NovaVcpuPinSet: 2-7,18-23,10-15,26-31
OvsDpdkCoreList: 0,16,8,24
OvsDpdkMemoryChannels: 4
OvsDpdkSocketMemory: 1024,1024
OvsPmdCoreList: 1,17,9,25

NOTE

The OvsDpdkMemoryChannels parameter cannot be derived from introspection details. In most cases, this value should be 4.

Deploy the Overcloud with the Derived Parameters

To deploy the overcloud with these derived parameters:

1. Copy the derived parameters from the deploy command output to the network-environment.yaml file.

```
# DPDK computer node.
ComputeOvsDpdkParameters:
  KernelArgs: default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt
  intel_iommu=on isolcpus=1-7,17-23,9-15,25-31
  TunedProfileName: "cpu-partitioning"
  IsolCpusList: "1-7,17-23,9-15,25-31"
  NovaVcpuPinSet: [2-7,18-23,10-15,26-31]
  NovaReservedHostMemory: 4096
  OvsDpdkSocketMemory: "1024,1024"
  OvsDpdkMemoryChannels: "4"
  OvsDpdkCoreList: "0,16,8,24"
  OvsPmdCoreList: "1,17,9,25"
```

NOTE

You must assign at least one CPU (with sibling thread) on each NUMA node with or without DPDK NICs present for DPDK PMD to avoid failures in creating guest instances.

NOTE

These parameters apply exclusively to the role ComputeOvsDpdk, and will not apply to other roles, including Compute or ComputeSriov that may exist on the same cluster. You can apply these parameters globally, but any global parameters are overwritten by role-specific parameters.

2. Deploy the overcloud using the role file and all environment files specific to your environment. See Deploying the Overcloud for details.

8.2. OVS-DPDK TOPOLOGY
With Red Hat OpenStack Platform, you can create custom deployment roles, using the composable roles feature, adding or removing services from each role. For more information on Composable Roles, see Composable Roles and Services.

This image shows a sample Open vSwitch with Data Plane Development Kit (OVS-DPDK) topology with two bonded ports for the control plane and data plane:

Configuring OVS-DPDK comprises the following tasks:

- If you use composable roles, copy and modify the `roles_data.yaml` file to add the custom role for OVS-DPDK.
- Update the appropriate `network-environment.yaml` file to include parameters for kernel arguments and DPDK arguments.
- Update the `compute.yaml` file to include the bridge for DPDK interface parameters.
- Update the `controller.yaml` file to include the same bridge details for DPDK interface parameters.
Run the `overcloud_deploy.sh` script to deploy the overcloud with the DPDK parameters.

**NOTE**

This guide provides examples for CPU assignments, memory allocation, and NIC configurations that can vary from your topology and use case. See the Network Functions Virtualization Product Guide and Chapter 2, Hardware requirements to understand the hardware and configuration options.

**Prerequisites**

- OVS 2.10
- DPDK 17
- Tested NIC. For a list of tested NICs for NFV, see Section 2.1, “Network Adapter support”.

**NOTE**

The Red Hat OpenStack Platform operates in OVS client mode for OVS-DPDK deployments.

### 8.3. SETTING THE MTU VALUE FOR OVS-DPDK INTERFACES

Red Hat OpenStack Platform supports jumbo frames for Open vSwitch with Data Plane Development Kit (OVS-DPDK). To set the maximum transmission unit (MTU) value for jumbo frames you must:

- Set the global MTU value for networking in the `network-environment.yaml` file.
- Set the physical DPDK port MTU value in the `compute.yaml` file. This value is also used by the vhost user interface.
- Set the MTU value within any guest instances on the Compute node to ensure that you have a comparable MTU value from end to end in your configuration.

**NOTE**

VXLAN packets include an extra 50 bytes in the header. Calculate your MTU requirements based on these additional header bytes. For example, an MTU value of 9000 means the VXLAN tunnel MTU value is 8950 to account for these extra bytes.

**NOTE**

You do not need any special configuration for the physical NIC because the NIC is controlled by the DPDK PMD and has the same MTU value set by the `compute.yaml` file. You cannot set an MTU value larger than the maximum value supported by the physical NIC.

To set the MTU value for OVS-DPDK interfaces:

NOTE

Ensure that the NeutronDpdkSocketMemory value in the network-environment.yaml file is large enough to support jumbo frames. See Section 7.4.2, “Memory parameters” for details.

2. Set the MTU value on the bridge to the Compute node in the controller.yaml file:

```yaml
- type: ovs_bridge
  name: br-link0
  use_dhcp: false
  members:
    - type: interface
      name: nic3
      mtu: 9000
```

3. Set the MTU values for an OVS-DPDK bond in the compute.yaml file:

```yaml
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  members:
    - type: ovs_dpdk_bond
      name: dpdkbond0
      mtu: 9000
      rx_queue: 2
      members:
        - type: ovs_dpdk_port
          name: dpdk0
          mtu: 9000
          members:
            - type: interface
              name: nic4
            - type: ovs_dpdk_port
              name: dpdk1
              mtu: 9000
              members:
                - type: interface
                  name: nic5
```

8.4. CONFIGURING A FIREWALL FOR SECURITY GROUPS

Dataplane interfaces need a high degree of performance in a stateful firewall. To protect these interfaces, consider deploying a telco grade firewall as a virtual network function (VNF).
Controlplane interfaces can be configured by setting the NeutronOVSFirewallDriver parameter to openvswitch. This configures OpenStack Networking to use the flow-based OVS firewall driver. This is set in the network-environment.yaml file under parameter_defaults.

Example:

```
parameter_defaults:
    NeutronOVSFirewallDriver: openvswitch
```

When the OVS firewall driver is used, it is important to disable it for dataplane interfaces. This can be done with the openstack port set command.

Example:

```
openstack port set --no-security-group --disable-port-security ${PORT}
```

### 8.5. SETTING MULTIQUEUE FOR OVS-DPDK INTERFACES

To set set same number of queues for interfaces in Open vSwitch with Data Plane Development Kit (OVS-DPDK) on the Compute node, modify the compute.yaml file as follows:

```
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  members:
    - type: ovs_dpdk_bond
      name: dpdkbond0
      mtu: 9000
      rx_queue: 2
      members:
        - type: ovs_dpdk_port
          name: dpdk0
          mtu: 9000
          members:
            - type: interface
              name: nic4
          - type: ovs_dpdk_port
            name: dpdk1
            mtu: 9000
            members:
              - type: interface
                name: nic5
```

### 8.6. DEPLOYING THE OVERCLOUD

1. Assign at least one CPU (with sibling thread) on each NUMA node with or without DPDK NICs present for DPDK PMD to avoid failures in creating guest instances.

2. Ensure parameters for your DPDK compute role are populated in network-environment.yaml. These can be copied from derived OVS-DPDK parameters if needed:

```
# DPDK compute node.
ComputeOvsDpdkParameters:
```
3. Deploy the overcloud using the `openstack overcloud deploy` command.

   - Include the role file and all environment files specific to your environment.

   - Apply the `KernelArgs` and `TunedProfile` parameters by including the `host-config-and-reboot.yaml` file from `/usr/share/openstack-tripleo-heat-templates/environments` to your deployment script:

     ```bash
     TEMPLATES_HOME="/usr/share/openstack-tripleo-heat-templates"
     CUSTOM_TEMPLATES="/home/stack/templates"
     openstack overcloud deploy --templates
     -r ${CUSTOM_TEMPLATES}/roles_data.yaml
     -e ${TEMPLATES_HOME}/environments/host-config-and-reboot.yaml
     -e ${CUSTOM_TEMPLATES}/network-environment.yaml
     -e ${CUSTOM_TEMPLATES}/controller.yaml
     -e ${CUSTOM_TEMPLATES}/computeovsdpdk.yaml
     ...
     ```

### 8.7. KNOWN LIMITATIONS

There are certain limitations when configuring OVS-DPDK with Red Hat OpenStack Platform for the NFV use case:

- Use Linux bonds for control plane networks. Ensure that both PCI devices used in the bond are on the same NUMA node for optimum performance. Neutron Linux bridge configuration is not supported by Red Hat.

- Huge pages are required for every instance running on the hosts with OVS-DPDK. If huge pages are not present in the guest, the interface appears but does not function.

- With OVS-DPDK, there is a performance degradation of services that use tap devices, such as Distributed Virtual Routing (DVR). The resulting performance is not suitable for a production environment.

- When using OVS-DPDK, all bridges on the same Compute node must be of type `ovs_user_bridge`. The director may accept the configuration, but Red Hat OpenStack Platform does not support mixing `ovs_bridge` and `ovs_user_bridge` on the same node.

### 8.8. CREATING A FLAVOR AND DEPLOYING AN INSTANCE FOR OVS-DPDK
After you have completed configuring Open vSwitch with Data Plane Development Kit (OVS-DPDK) for your Red Hat OpenStack Platform deployment with NFV, you can create a flavor and deploy an instance with the following steps:

1. Create an aggregate group and add relevant hosts for OVS-DPDK. Define metadata, for example `dpdk=true`, that matches defined flavor metadata.

   ```
   # openstack aggregate create dpdk_group
   # openstack aggregate add host dpdk_group [compute-host]
   # openstack aggregate set --property dpdk=true dpdk_group
   
   NOTE
   Use host aggregates to separate CPU pinned instances from unpinned instances. Instances that do not use CPU pinning do not respect the resourcing requirements of instances that use CPU pinning.
   ```

2. Create a flavor.

   ```
   # openstack flavor create <flavor> --ram <MB> --disk <GB> --vcpus <#>
   ```

3. Set additional flavor properties. Note that the defined metadata, `dpdk=true`, matches the defined metadata in the DPDK aggregate.

   ```
   # openstack flavor set <flavor> --property dpdk=true --property hw:cpu_policy=dedicated --property hw:mem_page_size=1GB --property hw:emulator_threads_policy=isolate
   
   In this example, `m1.medium_huge_4cpu` is the flavor name and the remaining parameters set the other properties for the flavor.
   ```

See Configure Emulator Threads to run on a Dedicated Physical CPU for details on the emulator threads policy for performance improvements.

1. Create the network.

   ```
   # openstack network create net1 --provider-physical-network tenant --provider-network-type vlan --provider-segment <VLAN-ID>
   ```

2. Deploy an instance.

   ```
   # openstack server create --flavor <flavor> --image <glance image> --nic net-id=<network ID>
   ```

For using multi-queue with OVS-DPDK, set the `hw.vif_multiqueue_enabled` property on an image then set the `hw.vif_multiqueue_enabled` property on a flavor:

1. Set the image properties.

   ```
   # openstack image set --property hw.vif_multiqueue_enabled=true <image>
   ```

2. Set additional flavor properties.

   ```
   # openstack flavor set --property hw:vif_multiqueue_enabled=true <flavor>
   ```
8.9. TROUBLESHOOTING THE CONFIGURATION

This section describes the steps to troubleshoot the Open vSwitch with Data Plane Development Kit (DPDK-OVS) configuration.

1. Review the bridge configuration and confirm that the bridge was created with the `datapath_type=netdev`.

   ```
   # ovs-vsctl list bridge br0
   _uuid               : bdce0825-e263-4d15-b256-f01222df96f3
   auto_attach         : []
   controller          : []
   datapath_id         : "00002608cebd154d"
   datapath_type       : netdev
   datapath_version    : "<built-in>"
   external_ids        : {}
   fail_mode           : []
   flood_vlans         : []
   flow_tables         : {}
   ipfix               : []
   mcast_snooping_enable: false
   mirrors             : []
   name                : "br0"
   netflow             : []
   other_config        : {}
   ports               : [52725b91-de7f-41e7-bb49-3b7e50354138]
   protocols           : []
   rstp_enable         : false
   rstp_status         : {}
   sflow               : []
   status              : {}
   stp_enable          : false
   ```

2. Confirm that the docker container `neutron_ovs_agent` is configured to start automatically.

   ```
   # docker inspect neutron_ovs_agent | grep -A1 RestartPolicy
   "RestartPolicy": {
     "Name": "always",
   }
   ```

3. If the container is having trouble starting, you can view any related messages.

   ```
   # less /var/log/containers/neutron/openvswitch-agent.log
   ```

4. Confirm that the PMD CPU mask of the `ovs-dpdk` are pinned to the CPUs. In case of HT, use sibling CPUs.
   For example, take `CPU4`:

   ```
   # cat /sys/devices/system/cpu/cpu4/topology/thread_siblings_list
   4,20
   `````
   So, using CPU 4 and 20:

   ```
   # ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=0x100010
   ```
Display their status:

```
# tuna -t ovs-vswitchd -CP
thread ctxt_switches pid SCHED_rtpri affinity voluntary nonvoluntary cmd
3161 OTHER 0 6 765023  614 ovs-vswitchd
3219 OTHER 0 6 1 0 handler24
3220 OTHER 0 6 1 0 handler21
3221 OTHER 0 6 1 0 handler22
3222 OTHER 0 6 1 0 handler23
3223 OTHER 0 6 1 0 handler25
3224 OTHER 0 6 1 0 handler26
3225 OTHER 0 6 1 0 handler27
3226 OTHER 0 6 1 0 handler28
3227 OTHER 0 6 2 0 handler31
3228 OTHER 0 6 2 4 handler30
3229 OTHER 0 6 2 5 handler32
3230 OTHER 0 6 953538 431 revalidator29
3231 OTHER 0 6 1424258 976 revalidator33
3232 OTHER 0 6 1424693 836 revalidator34
3233 OTHER 0 6 951678 503 revalidator36
3234 OTHER 0 6 1425128 498 revalidator35
*3235 OTHER 0 4 151123 51 pmd37*
*3236 OTHER 0 20 298967 48 pmd38*
3164 OTHER 0 6 47575 0 dpdk_watchdog3
3165 OTHER 0 6 237634 0 vhost_thread1
3166 OTHER 0 6 3665 0 urcu2
```
CHAPTER 9. TUNING A RED HAT OPENSTACK PLATFORM ENVIRONMENT

9.1. PINNING EMULATOR THREADS

Emulator threads handle interrupt requests and non-blocking processes for virtual machine hardware emulation. These threads float across the vCPUs that the guest uses for processing. If threads used for the poll mode driver (PMD) or real-time processing run on these vCPUs, you can experience packet loss or missed deadlines.

You can separate emulator threads from VM processing tasks by pinning the threads to their own vCPUs, increasing performance as a result.

9.1.1. Configuring CPUs to host emulator threads

To improve performance, reserve a subset of pCPUs for hosting emulator threads. Red Hat recommends using pCPUs identified in the `OvsDpdkCoreList` parameter.

Procedure

1. Deploy an overcloud with `NovaComputeCpuSharedSet` defined for a given role. The value of `NovaComputeCpuSharedSet` applies to the `cpu_shared_set` parameter in the `nova.conf` file for hosts within that role.

   ```
   parameter_defaults:
   ComputeOvsDpdkParameters:
   OvsDpdkCoreList: "0-1,16-17"
   NovaComputeCpuSharedSet: "0-1,16-17"
   ```

2. Create a flavor to build instances with emulator threads separated into a shared pool.

   ```
   openstack flavor create --ram <size_mb> --disk <size_gb> --vcpus <vcpus> <flavor>
   ```

3. Add the `hw:emulator_threads_policy` extra specification, and set the value to `share`. Instances created with this flavor will use the vCPUs defined in the `cpu_share_set` parameter in the `nova.conf` file.

   ```
   openstack flavor set <flavor> --property hw:emulator_threads_policy=share
   ```

   **NOTE**

   You must set the `cpu_share_set` parameter in the `nova.conf` file manually or with Heat to enable the share policy for this extra specification.

9.1.2. Verify the emulator thread pinning

Procedure

1. Identify the host for a given instance and the name of the instance.

   ```
   openstack server show <instance_id>
   ```
2. Use SSH to log onto the identified host as heat-admin.

```
ssh heat-admin@compute-1
[compute-1]$ sudo virsh dumpxml instance-00001 | grep `emulatorpin cpuset`
```

9.2. ENABLING RT-KVM FOR NFV WORKLOADS

This section describes the steps to install and configure Red Hat Enterprise Linux 7.5 Real Time KVM (RT-KVM) for the Red Hat OpenStack Platform. Red Hat OpenStack Platform provides real-time capabilities with a new Real-time Compute node role that provisions Red Hat Enterprise Linux for Real-Time, as well as the additional RT-KVM kernel module, and automatic configuration of the Compute node.

9.2.1. Planning for your RT-KVM Compute nodes

You must use Red Hat certified servers for your RT-KVM Compute nodes. See Red Hat Enterprise Linux for Real Time 7 certified servers for details.

See Registering and updating your undercloud for details on how to enable the rhel-7-server-nfv-rpms repository for RT-KVM, and ensuring your system is up to date.

NOTE
You need a separate subscription to a Red Hat OpenStack Platform for Real Time SKU before you can access this repository.

Building the real-time image

Use the following steps to build the overcloud image for Real-time Compute nodes.

1. To initialize the stack user to use the director command line tools, run the following command:

   `[stack@undercloud-0 ~]$ source ~/.stackrc`

2. Install the libguestfs-tools package on the undercloud to get the virt-customize tool:

   `(undercloud) [stack@undercloud-0 ~]$ sudo yum install libguestfs-tools`

3. Extract the images:

   `(undercloud) [stack@undercloud-0 ~]$ tar -xf /usr/share/rhosp-director-images/overcloud-full.tar`
   `(undercloud) [stack@undercloud-0 ~]$ tar -xf /usr/share/rhosp-director-images/ironic-python-agent.tar`

4. Copy the default image:

   `(undercloud) [stack@undercloud-0 ~]$ cp overcloud-full.qcow2 overcloud-realtime-compute.qcow2`

5. Register your image to enable Red Hat repositories relevant to your customizations. Replace [username] and [password] with valid credentials in the following example.
6. Find a list of pool IDs from your account’s subscriptions, and attach the appropriate pool ID to your image.

```
sudo subscription-manager list --all --available | less
```

```
virt-customize -a overcloud-realtime-compute.qcow2 --run-command
'subscription-manager attach --pool [pool-ID]' 
```

7. Add repositories necessary for Red Hat OpenStack Platform with NFV.

```
virt-customize -a overcloud-realtime-compute.qcow2 --run-command
'subscription-manager repos --enable=rhel-7-server-nfv-rpms
--enable=rhel-7-server-rpms
--enable=rhel-7-server-rh-common-rpms
--enable=rhel-7-server-extras-rpms
--enable=rhel-7-server-openstack-14-rpms'
```

8. Create a script to configure real-time capabilities on the image.

```
(undercloud) [stack@undercloud-0 ~]$ cat <<'EOF' > rt.sh
#!/bin/bash
set -eux
yum -v -y --setopt=protected_packages= erase kernel.$(uname -m)
yum -v -y install kernel-rt kernel-rt-kvm tuned-profiles-nfv-host
EOF
```

9. Run the script to configure the RT image:

```
(undercloud) [stack@undercloud-0 ~]$ virt-customize -a overcloud-realtime-compute.qcow2 -v --run rt.sh 2>&1 | tee virt-customize.log
```

NOTE

You may see the following error in the `rt.sh` script output: **grubby fatal error: unable to find a suitable template.** You can safely ignore this error.

10. You can check that the packages installed using the `rt.sh` script installed correctly by examining the `virt-customize.log` file that was created from the previous command.

```
(undercloud) [stack@undercloud-0 ~]$ cat virt-customize.log | grep Verifying
```


11. Relabel SELinux:

   (undercloud) [stack@undercloud-0 ~]$ virt-customize -a overcloud-realtime-compute.qcow2 - -selinux-relabel

12. Extract vmlinuz and initrd:

   
   **NOTE**
   
   The software version in the vmlinuz and initramfs filenames vary with the kernel version. Use the relevant software version in the filename, for example image/boot/vmlinuz-3.10.0-862.rt56.804.el7x86_64, or use the wildcard symbol * instead.

   (undercloud) [stack@undercloud-0 ~]$ mkdir image
   (undercloud) [stack@undercloud-0 ~]$ guestmount -a overcloud-realtime-compute.qcow2 -i - -ro image
   (undercloud) [stack@undercloud-0 ~]$ cp image/boot/vmlinuz-*x86_64 ./overcloud-realtime-compute.vmlinuz
   (undercloud) [stack@undercloud-0 ~]$ cp image/boot/initramfs-*x86_64.img ./overcloud-realtime-compute.initrd
   (undercloud) [stack@undercloud-0 ~]$ guestunmount image

13. Upload the image:

   (undercloud) [stack@undercloud-0 ~]$ openstack overcloud image upload --update-existing - -os-image-name overcloud-realtime-compute.qcow2

You now have a real-time image you can use with the **ComputeOvsDpdkRT** composable role on your selected Compute nodes.

**Modifying BIOS settings on RT-KVM Compute nodes**

To reduce latency on your RT-KVM Compute nodes, you must modify the BIOS settings. Disable all options for the following in your Compute node BIOS settings:

- Power Management
- Hyper-Threading
- CPU sleep states
- Logical processors
See Setting BIOS parameters for descriptions of these settings and the impact of disabling them. See your hardware manufacturer documentation for complete details on how to change BIOS settings.

9.2.2. Configuring OVS-DPDK with RT-KVM

NOTE

You must determine the best values for the OVS-DPDK parameters that you set in the network-environment.yaml file to optimize your OpenStack network for OVS-DPDK. See Section 8.1, “Deriving DPDK parameters with workflows” for details.

9.2.2.1. Generating the ComputeOvsDpdk composable role

You use the ComputeOvsDpdkRT role to specify Compute nodes that use the real-time compute image.

Generate roles_data.yaml for the ComputeOvsDpdkRT role.

# (undercloud) [stack@undercloud-0 ~]$ openstack overcloud roles generate -o roles_data.yaml

Controller ComputeOvsDpdkRT

9.2.2.2. Configuring the OVS-DPDK parameters

IMPORTANT

Attempting to deploy Data Plane Development Kit (DPDK) without appropriate values causes the deployment to fail or lead to unstable deployments. You must determine the best values for the OVS-DPDK parameters set in the network-environment.yaml file to optimize your OpenStack network for OVS-DPDK. See Section 8.1, “Deriving DPDK parameters with workflows” for details.

1. Add the nic configuration for the OVS-DPDK role you use under resource_registry:

resource_registry:

# Specify the relative/absolute path to the config files you want to use for override the default.
OS::TripleO::ComputeOvsDpdkRT::Net::SoftwareConfig: nic-configs/compute-ovs-dpdk.yaml
OS::TripleO::Controller::Net::SoftwareConfig: nic-configs/controller.yaml

2. Under parameter_defaults, set the OVS-DPDK and RT-KVM parameters:

# DPDK compute node.
ComputeOvsDpdkRTParameters:
  KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt intel_iommu=on isolcpus=1-7,17-23,9-15,25-31"
  TunedProfileName: "realtime-virtual-host"
  IsolCpusList:
    "1,2,3,4,5,6,7,9,10,17,18,19,20,21,22,23,11,12,13,14,15,25,26,27,28,29,30,31"
  NovaVcpuPinSet: [2,3,4,5,6,7,10,11,12,13,14,15,16,23,24,25,26,27,28,29,30,31]
  NovaReservedHostMemory: 4096
  OvsDpdkSocketMemory: "1024,1024"
9.2.2.3. Deploying the overcloud

Deploy the overcloud for ML2-OVS:

```bash
(undercloud) [stack@undercloud-0 ~]$ openstack overcloud deploy
   --templates
   -r /home/stack/ospd-14-vlan-dpdk-ctlplane-bonding-rt/roles_data.yaml
   -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml
   -e /usr/share/openstack-tripleo-heat-templates/environments/host-config-and-reboot.yaml
   -e /home/stack/ospd-14-vxlan-dpdk-data-bonding-rt-hybrid/containers-prepare-parameter.yaml
```

9.2.3. Launching an RT-KVM Instance

To launch an RT-KVM instance on a real-time enabled Compute node:

1. Create an RT-KVM flavor on the overcloud:

   ```bash
   # openstack flavor create --ram 4096 --disk 20 --vcpus 4 <flavor-name>
   # openstack flavor set --property hw:cpu_policy=dedicated <flavor-name>
   # openstack flavor set --property hw:cpu_realtime=yes <flavor-name>
   # openstack flavor set --property hw:mem_page_size=1GB <flavor-name>
   # openstack flavor set --property hw:cpu_realtime_mask=^0-1 <flavor-name>
   # openstack flavor set --property hw:emulator_threads_policy=isolate <flavor-name>
   ```

2. Launch an RT-KVM instance:

   ```bash
   # openstack server create --image <rhel> --flavor <flavor-name> --nic net-id=<dpdk-net>
   test-rt
   ```

3. Optionally, verify that the instance uses the assigned emulator threads:

   ```bash
   # virsh dumpxml <instance-id> | grep vcpu -A1
   <vcpu placement='static'>4</vcpu>
   <cputune>
     <vcupin vcpu="0" cpuset="1"/>
     <vcupin vcpu="1" cpuset="3"/>
     <vcupin vcpu="2" cpuset="5"/>
     <vcupin vcpu="3" cpuset="7"/>
     <emulatorpin cpuset="0-1"/>
     <vcpusched vcpu="2-3" scheduler="fifo"
                   priority="1"/>
   </cputune>
   ```
9.3. TRUSTED VIRTUAL FUNCTIONS

You can configure physical functions (PFs) to trust virtual functions (VFs) so that VFs can perform some privileged actions. For example, you can use this configuration to allow VFs to enable promiscuous mode or to change a hardware address.

9.3.1. Providing trust

Prerequisites

- An operational installation Red Hat OpenStack Platform director

Procedure

Complete the following steps to deploy the overcloud with the parameters necessary to enable physical function trust of virtual functions:

1. Add the `NeutronPhysicalDevMappings` parameter under the `parameter_defaults` section to make the link between the logical network name and the physical interface.

   ```
   parameter_defaults:
       NeutronPhysicalDevMappings: "sriov2:p5p2"
   ```

2. Add the new property "trusted" to the existing parameters related to SR-IOV.

   ```
   parameter_defaults:
       NeutronPhysicalDevMappings: "sriov2:p5p2"
       NeutronSriovNumVFs: ["p5p2:8"]
       NovaPCIPassthrough:
           - devname: "p5p2"
             physical_network: "sriov2"
             trusted: "true"
   ```

   **NOTE**
   
   You must include quotation marks around the value "true".

   **IMPORTANT**
   
   Complete the following step only in trusted environments. This step will allow non-administrative accounts the ability to bind trusted ports.

3. Modify permissions to allow users the capability of creating and updating port bindings.

   ```
   parameter_defaults:
       NeutronApiPolicies: {
           operator_create_binding_profile: { key: 'create_port:binding:profile', value: 'rule:admin_or_network_owner' },
           operator_get_binding_profile: { key: 'get_port:binding:profile', value: 'rule:admin_or_network_owner' },
           operator_update_binding_profile: { key: 'update_port:binding:profile', value: 'rule:admin_or_network_owner' }
       }
   ```
9.3.2. Utilizing trusted virtual functions

Execute the following on a fully deployed overcloud to utilize trusted virtual functions.

Creating a trusted VF network

1. Create a network of type vlan.

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

2. Create a subnet.

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

3. Create a port, setting the `vnic-type` option to direct, and the `binding-profile` option to true.

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

4. Create an instance binding it to the previously created trusted port.

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

Verify the trusted virtual function configuration on the hypervior

On the compute node that hosts the newly created instance, run the following command:

```bash
# ip link
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
```

View the output of the `ip link` command and verify that the trust status of the virtual function is trust on. The example output contains details of an environment that contains two ports. Note that `vf 6` contains the text trust on.

9.4. CONFIGURING RX/TX QUEUE SIZE

You can experience packet loss at high packet rates above 3.5 million packets per second (mpps) for many reasons, such as:

- a network interrupt
- a SMI
- packet processing latency in the Virtual Network Function
To prevent packet loss, increase the queue size from the default of 512 to a maximum of 1024.

**Procedure**

- To increase the RX and TX queue size, include the following lines to the `parameter_defaults:` section of a relevant director role. Here is an example with ComputeOvsDpdk role:

```yaml
parameter_defaults:
  ComputeOvsDpdkParameters:
    - NovaLibvirtRxQueueSize: 1024
    - NovaLibvirtTxQueueSize: 1024
```

**Testing**

- You can observe the values for RX queue size and TX queue size in the nova.conf file:

```ini
[libvirt]
rx_queue_size=1024
tx_queue_size=1024
```

- You can check the values for RX queue size and TX queue size in the VM instance XML file generated by libvirt on the compute host.

```xml
<devices>
  <interface type='vhostuser'>
    <mac address='56:48:4f:4d:5e:6f'/>
    <source type='unix' path='/tmp/vhost-user1' mode='server'/>
    <model type='virtio'/>
    <driver name='vhost' rx_queue_size='1024' tx_queue_size='1024'/>
  </interface>
</devices>
```

To verify the values for RX queue size and TX queue size, use the following command on a KVM host:

```bash
$ virsh dumpxml <vm name> | grep queue_size
```

- You can check for improved performance, such as 3.8 mpps/core at 0 frame loss.

### 9.5. CONFIGURING A NUMA-AWARE VSWITCH

**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](#).

Before you implement a NUMA-aware vSwitch, examine the following components of your hardware configuration:

- The number of physical networks.
- The placement of PCI cards.
- The physical architecture of the servers.

Memory-mapped I/O (MMIO) devices, such as PCIe NICs, are associated with specific NUMA nodes. When a VM and the NIC are on different NUMA nodes, there is a significant decrease in performance. To increase performance, align PCIe NIC placement and instance processing on the same NUMA node.

Use this feature to ensure that instances that share a physical network are located on the same NUMA node. To optimize datacenter hardware, you can leverage load-sharing VMs by using multiple networks, different network types, or bonding.

**IMPORTANT**

To architect NUMA-node load sharing and network access correctly, you must understand the mapping of the PCIe slot and the NUMA node. For detailed information on your specific hardware, refer to your vendor's documentation.

To prevent a cross-NUMA configuration, place the VM on the correct NUMA node, by providing the location of the NIC to Nova.

**Prerequisites**

- You have enabled the filter “NUMATopologyFilter”

**Procedure**

- Set a new `NeutronPhysnetNUMANodesMapping` parameter to map the physical network to the NUMA node that you associate with the physical network.
- If you use tunnels, such as VxLAN or GRE, you must also set the `NeutronTunnelNUMANodes` parameter.

```
parameter_defaults:
  NeutronPhysnetNUMANodesMapping: {<physnet_name>: [<NUMA_NODE>],}
  NeutronTunnelNUMANodes: <NUMA_NODE>,<NUMA_NODE>
```

Here is an example with 2 physical networks tunneled to NUMA node 0:

- one tenant network associated with NUMA node 0
- one management network without any affinity

```
parameter_defaults:
  NeutronBridgeMappings:
    - tenant:br-link0
  NeutronPhysnetNUMANodesMapping: {tenant: [1], mgmt: [0,1]}
  NeutronTunnelNUMANodes: 0
```

**Testing**

- Observe the configuration in the file `/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf`

```
[neutron_physnet_tenant]
```
numa_nodes=1
[neutron_tunnel]
numa_nodes=1

- Confirm the new configuration with the `lscpu` command:
  
  ```
  $ lscpu
  
  ```

- Launch a VM, with the NIC attached to the appropriate network

### 9.6. CONFIGURING QUALITY OF SERVICE (QOS) IN AN NFVI ENVIRONMENT

For details on Configuring QoS, see Configure Quality-of-Service (QoS). Support is limited to QoS rule type `bandwidth-limit` on SR-IOV and OVS-DPDK egress interfaces.
CHAPTER 10. EXAMPLE: CONFIGURING OVS-DPDK AND SR-IOV WITH VXLAN TUNNELLING

This section describes how to deploy Compute nodes with both OVS-DPDK and SR-IOV interfaces. The cluster installs with ML2/OVS and VXLAN tunnelling.

IMPORTANT

You must determine the best values for the OVS-DPDK parameters that you set in the network-environment.yaml file to optimize your OpenStack network for OVS-DPDK. See Deriving DPDK parameters with workflows for details.

10.1. CONFIGURING ROLES DATA

Red Hat OpenStack Platform provides a set of default roles in the roles_data.yaml file. You can create your own roles_data.yaml file to support the roles you require.

For the purposes of this example, the ComputeOvsDpdkSriov role is created. For information on creating roles in Red Hat OpenStack Platform, see Advanced Overcloud Customization. For details on the specific role used for this example, see roles_data.yaml.

10.2. CONFIGURING OVS-DPDK PARAMETERS

IMPORTANT

You must determine the best values for the OVS-DPDK parameters that you set in the network-environment.yaml file to optimize your OpenStack network for OVS-DPDK. See https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/14/html/network_functions_virtualization_planning_and_configuration_guide/part-dpdk-configure#proc_derive-dpdk for details.

1. Add the custom resources for OVS-DPDK under resource_registry:

```
resource_registry:
  # Specify the relative/absolute path to the config files you want to use for override the default.
  OS::TripleO::ComputeOvsDpdkSriov::Net::SoftwareConfig: nic-configs/computeovsdpdksriov.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: nic-configs/controller.yaml
```

2. Under parameter_defaults, set the tunnel type to vxlan and the network type to vxlan,vlan:

```
NeutronTunnelTypes: 'vxlan'
NeutronNetworkType: 'vxlan,vlan'
```

3. Under parameters_defaults, set the bridge mapping:

```
# The OVS logical->physical bridge mappings to use.
NeutronBridgeMappings:
  - dpdk-mgmt:br-link0
```
4. Under `parameter_defaults`, set the role-specific parameters for the `ComputeOvsDpdkSriov` role:

### OVS DPDK configuration #

**ComputeOvsDpdkSriovParameters:**
- **KernelArgs:** "default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt intel_iommu=on isolcpus=2-19,22-39"
- **TunedProfileName:** "cpu-partitioning"
- **IsolCpusList:** "2-19,22-39"
- **NovaVcpuPinSet:** [4-19,24-39]
- **NovaReservedHostMemory:** 4096
- **OvsDpdkSocketMemory:** "3072,1024"
- **OvsDpdkMemoryChannels:** "4"
- **OvsDpdkCoreList:** "0,20,1,21"
- **OvsPmdCoreList:** "2,22,3,23"
- **NovaComputeCpuSharedSet:** [0,20,1,21]
- **NovaLibvirtRxQueueSize:** 1024
- **NovaLibvirtTxQueueSize:** 1024

**NOTE**

You must assign at least one CPU (with sibling thread) on each NUMA node with or without DPDK NICs present for DPDK PMD to avoid failures in creating guest instances. This is expressed in the example through the `OvsPmdCoreList` parameter, having cores 2 and 22 from NUMA 1, and having cores 3 and 23 from NUMA 2.

**NOTE**

These huge pages are consumed by the virtual machines, and also by OVS-DPDK using the `OvsDpdkSocketMemory` parameter as shown in this procedure. The number of huge pages available for the virtual machines is the `boot` parameter minus the `OvsDpdkSocketMemory`.

You must also add `hw:mem_page_size=1GB` to the flavor you associate with the DPDK instance.

**NOTE**

`OvsDPDKCoreList` and `OvsDpdkMemoryChannels` are the required settings for this procedure. Attempting to deploy DPDK without appropriate values causes the deployment to fail or lead to unstable deployments.

5. Configure the role-specific parameters for SR-IOV:

**NovaPCIPassthrough:**
- **devname:** "p7p3"
  - trusted: "true"
  - physical_network: "sriov-1"
- **devname:** "p7p4"
  - trusted: "true"
  - physical_network: "sriov-2"
10.3. CONFIGURING THE CONTROLLER NODE

1. Create the control plane Linux bond for an isolated network.

   ```yaml
   - type: linux_bond
     name: bond_api
     bonding_options: "mode=active-backup"
     use_dhcp: false
     dns_servers:
       get_param: DnsServers
     members:
       - type: interface
         name: nic2
         primary: true
   ```

2. Assign VLANs to this Linux bond.

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

   - type: vlan
     vlan_id:
       get_param: StorageNetworkVlanID
     device: bond_api
     addresses:
       - ip_netmask:
         get_param: StorageIpSubnet

   - type: vlan
     vlan_id:
       get_param: StorageMgmtNetworkVlanID
     device: bond_api
     addresses:
       - ip_netmask:
         get_param: StorageMgmtIpSubnet

   - type: vlan
     vlan_id:
       get_param: ExternalNetworkVlanID
     device: bond_api
     addresses:
       - ip_netmask:
         get_param: ExternalIpSubnet
     routes:
       - default: true
         next_hop:
           get_param: ExternalInterfaceDefaultRoute
   ```

3. Create the OVS bridge for access to neutron-dhcp-agent and neutron-metadata-agent services.
10.4. CONFIGURING THE COMPUTE NODE FOR DPDK AND SR-IOV

Create the `computeovsdpdksrivov.yaml` file from the default `compute.yaml` file and make the following changes:

1. Create the control plane Linux bond for an isolated network.

   ```yaml
   - 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
   ```

2. Assign VLANs to this Linux bond.

   ```yaml
   - type: vlan
     vlan_id:
       get_param: InternalApiNetworkVlanID
     device: bond_api
     addresses:
     - ip_netmask:
       get_param: InternalApiIpSubnet
   
   - type: vlan
     vlan_id:
       get_param: StorageNetworkVlanID
     device: bond_api
     addresses:
     - ip_netmask:
       get_param: StoragelpSubnet
   ```
3. Set a bridge with a DPDK port to link to the controller.

```yaml
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  ovs_extra:
    - str_replace:
        template: set port br-link0 tag=_VLAN_TAG_
        params:
          _VLAN_TAG_: get_param: TenantNetworkVlanID
  addresses:
    - ip_netmask:
        get_param: TenantIpSubnet
  members:
    - type: ovs_dpdk_bond
      name: dpdkbond0
      mtu: 9000
      rx_queue: 2
      members:
        - type: ovs_dpdk_port
          name: dpdk0
          members:
            - type: interface
              name: nic7
        - type: ovs_dpdk_port
          name: dpdk1
          members:
            - type: interface
              name: nic8
```

**NOTE**

To include multiple DPDK devices, repeat the `type` code section for each DPDK device that you want to add.

**NOTE**

When using OVS-DPDK, all bridges on the same Compute node must be of type `ovs_user_bridge`. The director may accept the configuration, but Red Hat OpenStack Platform does not support mixing `ovs_bridge` and `ovs_user_bridge` on the same node.

### 10.5. DEPLOYING THE OVERCLOUD

Run the `overcloud_deploy.sh` script to deploy the overcloud.

```bash
openstack overcloud deploy \
--templates \
-r /home/stack/ospd-14-vxlan-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/roles_data.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/host-config-and-reboot.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/neutron-ovs-dpdk.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/neutron-sriov.yaml \
```
-e /usr/share/openstack-tripleo-heat-templates/environments/collectd-environment.yaml \ 
- e /home/stack/ospd-14-vxlan-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/api-policies.yaml \ 
- e /home/stack/ospd-14-vxlan-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/network-environment.yaml \ 
- e /home/stack/overcloud_images.yaml \ 
--log-file overcloud_install.log &> overcloud_install.log
CHAPTER 11. UPGRADING RED HAT OPENSTACK PLATFORM WITH NFV

There are additional considerations and steps required to upgrade Red Hat OpenStack Platform when you have OVS-DPDK configured. The steps are covered in *Preparing an NFV-Configured Overcloud*. 
CHAPTER 12. PERFORMANCE

Red Hat OpenStack Platform director configures 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. OVS 2.10 embeds support for DPDK 17 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 characteristics, 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.

See NFV Performance Considerations and Configure Emulator Threads to run on a Dedicated Physical CPU for a high-level introduction to CPUs and NUMA topology.
CHAPTER 13. FINDING MORE INFORMATION

The following table includes additional Red Hat documentation for reference:

The Red Hat OpenStack Platform documentation suite can be found here: Red Hat OpenStack Platform Documentation Suite

Table 13.1. List of Available Documentation

<table>
<thead>
<tr>
<th>Component</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Hat Enterprise Linux</td>
<td>Red Hat OpenStack Platform is supported on Red Hat Enterprise Linux 7.5. For information on installing Red Hat Enterprise Linux, see the corresponding installation guide at: Red Hat Enterprise Linux Documentation Suite.</td>
</tr>
<tr>
<td>Red Hat OpenStack Platform</td>
<td>To install OpenStack components and their dependencies, use the Red Hat OpenStack Platform director. The director uses a basic OpenStack installation as the undercloud to install, configure, and manage the OpenStack nodes in the final overcloud. You need one extra host machine for the installation of the undercloud, in addition to the environment necessary for the deployed overcloud. For detailed instructions, see Red Hat OpenStack Platform Director Installation and Usage. For information on configuring advanced features for a Red Hat OpenStack Platform enterprise environment using the Red Hat OpenStack Platform director such as network isolation, storage configuration, SSL communication, and general configuration method, see Advanced Overcloud Customization.</td>
</tr>
<tr>
<td>NFV Documentation</td>
<td>For a high level overview of the NFV concepts, see the Network Functions Virtualization Product Guide.</td>
</tr>
</tbody>
</table>
APPENDIX A. SAMPLE DPDK SRIOV HYBRID YAML FILES

This section provides sample YAML files as a reference for adding single root I/O virtualization (SR-IOV) and Data Plane Development Kit (DPDK) interfaces on the same compute node.

NOTE

These templates are from a fully configured environment and include parameters unrelated to NFV, that may not be relevant or appropriate for your deployment.

A.1. SAMPLE VXLAN DPDK SRIOV YAML FILES

A.1.1. roles_data.yaml

```yaml
# File generated by TripleO
# Role: Controller
# name: Controller
description: |
  Controller role that has all the controller services loaded and handles Database, Messaging and Network functions.
CountDefault: 1
tags:
  - primary
  - controller
networks:
  - External
  - InternalApi
  - Storage
  - StorageMgmt
  - Tenant
  # For systems with both IPv4 and IPv6, you may specify a gateway network for
  # each, such as ['ControlPlane', 'External']
default_route_networks: ['External']
HostnameFormatDefault: %stackname%-controller-%index%'
# Deprecated & backward-compatible values (FIXME: Make parameters consistent)
# Set uses_deprecated_params to True if any deprecated params are used.
uses_deprecated_params: True
deprecated_param_extraconfig: 'controllerExtraConfig'
deprecated_param_flavor: 'OvercloudControlFlavor'
deprecated_param_image: 'controllerImage'
deprecated_nic_config_name: 'controller.yaml'
ServicesDefault:
  - OS::TripleO::Services::Aide
  - OS::TripleO::Services::AodhApi
  - OS::TripleO::Services::AodhEvaluator
  - OS::TripleO::Services::AodhListener
  - OS::TripleO::Services::AodhNotifier
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::BarbicanApi
  - OS::TripleO::Services::BarbicanBackendSimpleCrypto
```
- OS::TripleO::Services::BarbicanBackendDogtag
- OS::TripleO::Services::BarbicanBackendKmip
- OS::TripleO::Services::BarbicanBackendPkcs11Crypto
- OS::TripleO::Services::CACerts
- OS::TripleO::Services::CeilometerAgentCentral
- OS::TripleO::Services::CeilometerAgentNotification
- OS::TripleO::Services::CephExternal
- OS::TripleO::Services::CephMds
- OS::TripleO::Services::CephMgr
- OS::TripleO::Services::CephMon
- OS::TripleO::Services::CephRbdMirror
- OS::TripleO::Services::CephRgw
- OS::TripleO::Services::CertmongerUser
- OS::TripleO::Services::CinderApi
- OS::TripleO::Services::CinderBackendDellPs
- OS::TripleO::Services::CinderBackendDellSc
- OS::TripleO::Services::CinderBackendDellEMCUUnity
- OS::TripleO::Services::CinderBackendDellEMCVMAXISCSI
- OS::TripleO::Services::CinderBackendDellEMCVNX
- OS::TripleO::Services::CinderBackendDellEMCXTRMIOISCSI
- OS::TripleO::Services::CinderBackendNetApp
- OS::TripleO::Services::CinderBackendScaleIO
- OS::TripleO::Services::CinderBackendVRTSHyperScale
- OS::TripleO::Services::CinderBackendNVMeOF
- OS::TripleO::Services::CinderBackup
- OS::TripleO::Services::CinderHPELeftHandISCSI
- OS::TripleO::Services::CinderScheduler
- OS::TripleO::Services::CinderVolume
- OS::TripleO::Services::Clustercheck
- OS::TripleO::Services::Collectd
- OS::TripleO::Services::Congress
- OS::TripleO::Services::ContainerImagePrepare
- OS::TripleO::Services::DesignateApi
- OS::TripleO::Services::DesignateCentral
- OS::TripleO::Services::DesignateProducer
- OS::TripleO::Services::DesignateWorker
- OS::TripleO::Services::DesignateMDNS
- OS::TripleO::Services::DesignateSink
- OS::TripleO::Services::Docker
- OS::TripleO::Services::Ec2Api
- OS::TripleO::Services::Etcd
- OS::TripleO::Services::ExternalSwiftProxy
- OS::TripleO::Services::Fluentd
- OS::TripleO::Services::GlanceApi
- OS::TripleO::Services::GlanceRegistry
- OS::TripleO::Services::GnocchiApi
- OS::TripleO::Services::GnocchiMetricd
- OS::TripleO::Services::GnocchiStatsd
- OS::TripleO::Services::HAproxy
- OS::TripleO::Services::HeatApi
- OS::TripleO::Services::HeatApiCloudwatch
- OS::TripleO::Services::HeatApiCfn
- OS::TripleO::Services::HeatEngine
- OS::TripleO::Services::Horizon
- OS::TripleO::Services::Ipsec
- OS::TripleO::Services::IronicApi
- OS::TripleO::Services::IronicConductor
- OS::TripleO::Services::IronicInspector
- OS::TripleO::Services::IronicPxe
- OS::TripleO::Services::IronicNeutronAgent
- OS::TripleO::Services::Iscsid
- OS::TripleO::Services::Keepalived
- OS::TripleO::Services::Kernel
- OS::TripleO::Services::Keystone
- OS::TripleO::Services::LoginDefs
- OS::TripleO::Services::ManilaApi
- OS::TripleO::Services::ManilaBackendCephFs
- OS::TripleO::Services::ManilaBackendIsilon
- OS::TripleO::Services::ManilaBackendNetapp
- OS::TripleO::Services::ManilaBackendUnity
- OS::TripleO::Services::ManilaBackendVNX
- OS::TripleO::Services::ManilaBackendVMAX
- OS::TripleO::Services::ManilaScheduler
- OS::TripleO::Services::ManilaShare
- OS::TripleO::Services::Memcached
- OS::TripleO::Services::MetricsQdr
- OS::TripleO::Services::MistralApi
- OS::TripleO::Services::MistralEngine
- OS::TripleO::Services::MistralExecutor
- OS::TripleO::Services::MistralEventEngine
- OS::TripleO::Services::MongoDb
- OS::TripleO::Services::MySQL
- OS::TripleO::Services::MySQLClient
- OS::TripleO::Services::NeutronApi
- OS::TripleO::Services::NeutronBgpVpnApi
- OS::TripleO::Services::NeutronSfcApi
- OS::TripleO::Services::NeutronCorePlugin
- OS::TripleO::Services::NeutronDhcpAgent
- OS::TripleO::Services::NeutronL2gwAgent
- OS::TripleO::Services::NeutronL2gwApi
- OS::TripleO::Services::NeutronL3Agent
- OS::TripleO::Services::NeutronLbaasv2Agent
- OS::TripleO::Services::NeutronLbaasv2Api
- OS::TripleO::Services::NeutronLinuxbridgeAgent
- OS::TripleO::Services::NeutronMetadataAgent
- OS::TripleO::Services::NeutronML2FujitsuCfab
- OS::TripleO::Services::NeutronML2FujitsuFossw
- OS::TripleO::Services::NeutronOvsAgent
- OS::TripleO::Services::NeutronVppAgent
- OS::TripleO::Services::NovaApi
- OS::TripleO::Services::NovaConductor
- OS::TripleO::Services::NovaConsoleauth
- OS::TripleO::Services::NovaIronic
- OS::TripleO::Services::NovaMetadata
- OS::TripleO::Services::NovaPlacement
- OS::TripleO::Services::NovaScheduler
- OS::TripleO::Services::NovaVncProxy
- OS::TripleO::Services::Ntp
- OS::TripleO::Services::ContainersLogrotateCron
- OS::TripleO::Services::OctaviaDeploymentConfig
- OS::TripleO::Services::OctaviaApi
- OS::TripleO::Services::OctaviaHealthManager

APPENDIX A. SAMPLE DPDK SRIOV HYBRID YAML FILES
- OS::TripleO::Services::OctaviaHousekeeping
- OS::TripleO::Services::OctaviaWorker
- OS::TripleO::Services::OpenDaylightApi
- OS::TripleO::Services::OpenDaylightOvs
- OS::TripleO::Services::OVNDBs
- OS::TripleO::Services::OVNController
- OS::TripleO::Services::Pacemaker
- OS::TripleO::Services::PankoApi
- OS::TripleO::Services::OsloMessagingRpc
- OS::TripleO::Services::OsloMessagingNotify
- OS::TripleO::Services::Redis
- OS::TripleO::Services::Rhsm
- OS::TripleO::Services::RsyslogSidecar
- OS::TripleO::Services::SaharaApi
- OS::TripleO::Services::SaharaEngine
- OS::TripleO::Services::Securetty
- OS::TripleO::Services::SensuClient
- OS::TripleO::Services::SkydiveAgent
- OS::TripleO::Services::SkydiveAnalyzer
- OS::TripleO::Services::Snmp
- OS::TripleO::Services::Sshd
- OS::TripleO::Services::SwiftProxy
- OS::TripleO::Services::SwiftDispersion
- OS::TripleO::Services::SwiftRingBuilder
- OS::TripleO::Services::SwiftStorage
- OS::TripleO::Services::Tacker
- OS::TripleO::Services::Timezone
- OS::TripleO::Services::TripleoFirewall
- OS::TripleO::Services::TripleoPackages
- OS::TripleO::Services::Tuned
- OS::TripleO::Services::Vpp
- OS::TripleO::Services::Zaqar
- OS::TripleO::Services::Ptp
- OS::TripleO::Services::Xinetd

# Role: ComputeOvsDpdkSriov

- name: ComputeOvsDpdkSriov
description: | Compute OvS DPDK Role
CountDefault: 1
networks:
  - InternalApi
  - Tenant
  - Storage
RoleParametersDefault:
  VhostuserSocketGroup: "hugetlbfs"
TunedProfileName: "cpu-partitioning"
ServicesDefault:
  - OS::TripleO::Services::Aide
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::BootParams
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CephClient
  - OS::TripleO::Services::CephExternal
  - OS::TripleO::Services::CertmongerUser
A.1.2. network-environment.yaml

```yaml
resource_registry:
  # Specify the relative/absolute path to the config files you want to use for override the default.
  OS::TripleO::_computeOvsDpdkSriov::Net::SoftwareConfig: nic-configs/computeovsdpksriov.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: nic-configs/controller.yaml

parameter_defaults:
  # Customize all these values to match the local environment
  InternalApiNetCidr: 10.10.10.0/24
  TenantNetCidr: 10.10.2.0/24
  StorageNetCidr: 10.10.3.0/24
  StorageMgmtNetCidr: 10.10.4.0/24
  ExternalNetCidr: 172.20.12.112/28
  # CIDR subnet mask length for provisioning network
  ControlPlaneSubnetCidr: '24'
  InternalApiAllocationPools: ["{'start': '10.10.10.10', 'end': '10.10.10.200'}"]
  TenantAllocationPools: ["{'start': '10.10.2.100', 'end': '10.10.2.200'}"]
```
StorageAllocationPools: [{'start': '10.10.3.100', 'end': '10.10.3.200'}]
StorageMgmtAllocationPools: [{'start': '10.10.4.100', 'end': '10.10.4.200'}]

# Use an External allocation pool which will leave room for floating IPs

# Set to the router gateway on the external network
ExternalInterfaceDefaultRoute: 172.20.12.126

# Gateway router for the provisioning network (or Undercloud IP)
ControlPlaneDefaultRoute: 192.168.24.1

# Generally the IP of the Undercloud
EC2MetadataIp: 192.168.24.1

InternalApiNetworkVlanID: 10
TenantNetworkVlanID: 11
StorageNetworkVlanID: 12
StorageMgmtNetworkVlanID: 13

ExternalNetworkVlanID: 14

# Define the DNS servers (maximum 2) for the overcloud nodes
DnsServers: ['8.8.8.8', '8.8.4.4']

# May set to br-ex if using floating IPs only on native VLAN on bridge br-ex
NeutronExternalNetworkBridge: 

# The tunnel type for the tenant network (vxlan or gre). Set to " to disable tunneling.
NeutronTunnelTypes: 'vxlan'

# The tenant network type for Neutron (vlan or vxlan).
NeutronNetworkType: 'vxlan,vlan'

# The OVS logical->physical bridge mappings to use.
NeutronBridgeMappings: 'dpdk-mgmt:br-link0'

# The Neutron ML2 and OpenVSwitch vlan mapping range to support.

# Nova flavor to use.
OvercloudControllerFlavor: controller

# Number of nodes to deploy.
ControllerCount: 3

# NTP server configuration.
NtpServer: clock.redhat.com

# MTU global configuration
NeutronGlobalPhysnetMtu: 9000

# Configure the classname of the firewall driver to use for implementing security groups.
NeutronOVSFirewallDriver: openvswitch

SshServerOptions:
  UseDns: 'no'

ControllerHostnameFormat: 'controller-%index%'

ComputeOvsDpdkSriovHostnameFormat: 'compute-%index%'

# Collectd server that will gather information from overcloud deployment - Service Assurance
CollectdServer: 192.0.90.1
CollectdConnectionType: "network"

#########################################################################
# OVS DPDK configuration #
#########################################################################

ComputeOvsDpdkSriovParameters:
  KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt intel_iommu=on isolcpus=2-19,22-39"
APPENDIX A. SAMPLE DPDK SRIOV HYBRID YAML FILES

```
TunedProfileName: "cpu-partitioning"
IsolCpusList: "2-19,22-39"
NovaVcpuPinSet: [4-19,24-39]
NovaReservedHostMemory: 4096
OvsDpdkSocketMemory: "3072,1024"
OvsDpdkMemoryChannels: "4"
OvsDpdkCoreList: "0,20,1,21"
OvsPmdCoreList: "2,22,3,23"
NovaComputeCpuSharedSet: [0,20,1,21]
OvsLibvirtRxQueueSize: 1024
OvsLibvirtTxQueueSize: 1024
NovaPCIPassthrough:
  - devname: "p7p3"
    trusted: "true"
    physical_network: "sriov-1"
  - devname: "p7p4"
    trusted: "true"
    physical_network: "sriov-2"
NeutronPhysicalDevMappings: "sriov-1:p7p3,sriov-2:p7p4"
NeutronSriovNumVFs: "p7p3:5,p7p4:5"

A.1.3. controller.yaml

heat_template_version: rocky

description: >
  Software Config to drive os-net-config to configure VLANs for the
  controller role.

parameters:
  ControlPlaneIp:
    default: 
    description: IP address/subnet on the ctlplane network
    type: string
  ExternalIpSubnet:
    default: 
    description: IP address/subnet on the external network
    type: string
  ExternalInterfaceRoutes:
    default: []
    description: >
      Routes for the external network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  InternalApiIpSubnet:
    default: 
    description: IP address/subnet on the internal_api network
    type: string
```
InternalApiInterfaceRoutes:
  default: []
  description: >
    Routes for the internal_api network traffic.
    JSON route e.g. [{'destination': '10.0.0.0/16', 'nexthop': '10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
StorageIpSubnet:
  default: ''
  description: IP address/subnet on the storage network
  type: string
StorageInterfaceRoutes:
  default: []
  description: >
    Routes for the storage network traffic.
    JSON route e.g. [{'destination': '10.0.0.0/16', 'nexthop': '10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
StorageMgmtIpSubnet:
  default: ''
  description: IP address/subnet on the storage_mgmt network
  type: string
StorageMgmtInterfaceRoutes:
  default: []
  description: >
    Routes for the storage_mgmt network traffic.
    JSON route e.g. [{'destination': '10.0.0.0/16', 'nexthop': '10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
TenantIpSubnet:
  default: ''
  description: IP address/subnet on the tenant network
  type: string
TenantInterfaceRoutes:
  default: []
  description: >
    Routes for the tenant network traffic.
    JSON route e.g. [{'destination': '10.0.0.0/16', 'nexthop': '10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
ManagementIpSubnet: # Only populated when including environments/network-management.yaml
  default: ''
  description: IP address/subnet on the management network
  type: string
ManagementInterfaceRoutes:
  default: []
  description: >
    Routes for the management network traffic.
    JSON route e.g. [{'destination': '10.0.0.0/16', 'nexthop': '10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
APPENDIX A. SAMPLE DPDK SRIOV HYBRID YAML FILES

BondInterfaceOvsOptions:
  default: 'bond_mode=active-backup'
  description: The ovs_options string for the bond interface. Set things like lACP=active and/or bond_mode=balance-slb using this option.
  type: string

ExternalNetworkVlanID:
  default: 10
  description: Vlan ID for the external network traffic.
  type: number

InternalApiNetworkVlanID:
  default: 20
  description: Vlan ID for the internal api network traffic.
  type: number

StorageNetworkVlanID:
  default: 30
  description: Vlan ID for the storage network traffic.
  type: number

StorageMgmtNetworkVlanID:
  default: 40
  description: Vlan ID for the storage_mgnt network traffic.
  type: number

TenantNetworkVlanID:
  default: 50
  description: Vlan ID for the tenant network traffic.
  type: number

ManagementNetworkVlanID:
  default: 60
  description: Vlan ID for the management network traffic.
  type: number

ExternalInterfaceDefaultRoute:
  default: '10.0.0.1'
  description: default route for the external network
  type: string

ControlPlaneSubnetCidr:
  default: ''
  description: >
    The subnet CIDR of the control plane network. (The parameter is automatically resolved from the ctlplane subnet's cidr attribute.)
  type: string

ControlPlaneDefaultRoute:
  default: ''
  description: The default route of the control plane network. (The parameter is automatically resolved from the ctlplane subnet's gateway_ip attribute.)
  type: string

DnsServers: # Override this via parameter_defaults
  default: []
  description: >
    DNS servers to use for the Overcloud (2 max for some implementations).
    If not set the nameservers configured in the ctlplane subnet's dns_nameservers attribute will be used.
  type: comma_delimited_list

EC2MetadataIp:
  default: ''
  description: The IP address of the EC2 metadata server. (The parameter is automatically resolved from the ctlplane subnet's host_routes attribute.)
  type: string
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
        $network_config:
          network_config:
            - type: interface
              name: nic1
              use_dhcp: false
              addresses:
                - ip_netmask:
                    list_join:
                      - / 
                      - get_param: ControlPlaneIp
                      - get_param: ControlPlaneSubnetCidr
              routes:
                - ip_netmask: 169.254.169.254/32
                  next_hop: get_param: EC2MetadataIP
            - type: linux_bond
              name: bond_api
              bonding_options: "mode=active-backup"
              use_dhcp: false
              dns_servers:
                get_param: DnsServers
              members:
                - type: interface
                  name: nic2
                  primary: true
            - type: vlan
              vlan_id:
                get_param: InternalApiNetworkVlanID
              device: bond_api
              addresses:
                - ip_netmask:
                    get_param: InternalApiIpSubnet
            - type: vlan
              vlan_id:
                get_param: StorageNetworkVlanID
              device: bond_api
              addresses:
                - ip_netmask:
                    get_param: StorageIpSubnet
            - type: vlan
              vlan_id:


```yaml
  - type: vlan
    vlan_id:
      get_param: StorageMgmtNetworkVlanID
    device: bond_api
    addresses:
      - ip_netmask:
          get_param: StorageMgmtIpSubnet
  - type: vlan
    vlan_id:
      get_param: ExternalNetworkVlanID
    device: bond_api
    addresses:
      - ip_netmask:
          get_param: ExternalIpSubnet
    routes:
      - default: true
        next_hop:
          get_param: ExternalInterfaceDefaultRoute
  - type: ovs_bridge
    name: br-link0
    use_dhcp: false
    mtu: 9000
    members:
      - type: interface
        name: nic3
        mtu: 9000
    - type: vlan
      vlan_id:
        get_param: TenantNetworkVlanID
      mtu: 9000
      addresses:
        - ip_netmask:
            get_param: TenantIpSubnet

outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value:
      get_resource: OsNetConfigImpl

A.1.4. compute-ovs-dpdk.yaml

heat_template_version: rocky

description: >
  Software Config to drive os-net-config to configure VLANs for the
  compute role.

parameters:
  ControlPlaneIp:
    default: "
    description: IP address/subnet on the ctlplane network
type: string
  ExternalIpSubnet:
    default: "
    description: "
```
description: IP address/subnet on the external network
type: string

ExternalInterfaceRoutes:
default: []
description: >
  Routes for the external network traffic.
  JSON route e.g. [{'destination': '10.0.0.0/16', 'nexthop': '10.0.0.1'}]
  Unless the default is changed, the parameter is automatically resolved
  from the subnet host_routes attribute.

type: json

InternalApiIpSubnet:
default: ""
description: IP address/subnet on the internal_api network
type: string

InternalApiInterfaceRoutes:
default: []
description: >
  Routes for the internal_api network traffic.
  JSON route e.g. [{'destination': '10.0.0.0/16', 'nexthop': '10.0.0.1'}]
  Unless the default is changed, the parameter is automatically resolved
  from the subnet host_routes attribute.

type: json

StorageIpSubnet:
default: ""
description: IP address/subnet on the storage network
type: string

StorageInterfaceRoutes:
default: []
description: >
  Routes for the storage network traffic.
  JSON route e.g. [{'destination': '10.0.0.0/16', 'nexthop': '10.0.0.1'}]
  Unless the default is changed, the parameter is automatically resolved
  from the subnet host_routes attribute.

type: json

StorageMgmtIpSubnet:
default: ""
description: IP address/subnet on the storage_mgmt network
type: string

StorageMgmtInterfaceRoutes:
default: []
description: >
  Routes for the storage_mgmt network traffic.
  JSON route e.g. [{'destination': '10.0.0.0/16', 'nexthop': '10.0.0.1'}]
  Unless the default is changed, the parameter is automatically resolved
  from the subnet host_routes attribute.

type: json

TenantIpSubnet:
default: ""
description: IP address/subnet on the tenant network
type: string

TenantInterfaceRoutes:
default: []
description: >
  Routes for the tenant network traffic.
  JSON route e.g. [{'destination': '10.0.0.0/16', 'nexthop': '10.0.0.1'}]
  Unless the default is changed, the parameter is automatically resolved
ManagementIpSubnet: # Only populated when including environments/network-management.yaml
default: 

description: IP address/subnet on the management network
type: string

ManagementInterfaceRoutes:
default: []
description: Routes for the management network traffic.
JSON route e.g. [{'destination': '10.0.0.0/16', 'nexthop': '10.0.0.1'}]
Unless the default is changed, the parameter is automatically resolved from the subnet host_routes attribute.
type: json

BondInterfaceOvsOptions:
default: 'bond_mode=active-backup'
description: The ovs_options string for the bond interface. Set things like lacp=active and/or bond_mode=balance-slb using this option.
type: string

InternalApiNetworkVlanID:
default: 10
description: Vlan ID for the internal_api network traffic.
type: number

StorageNetworkVlanID:
default: 30
description: Vlan ID for the storage network traffic.
type: number

StorageMgmtNetworkVlanID:
default: 40
description: Vlan ID for the storage_mgmt network traffic.
type: number

TenantNetworkVlanID:
default: 50
description: Vlan ID for the tenant network traffic.
type: number

ManagementNetworkVlanID:
default: 60
description: Vlan ID for the management network traffic.
type: number

ExternalInterfaceDefaultRoute:
default: '10.0.0.1'
description: default route for the external network
type: string

ControlPlaneSubnetCidr:
default: 

description: The subnet CIDR of the control plane network. (The parameter is automatically resolved from the ctlplane subnet's cidr attribute.)
type: string

ControlPlaneDefaultRoute:
default: 

description: The default route of the control plane network. (The parameter
is automatically resolved from the ctlplane subnet's `gateway_ip` attribute.

type: string

DnsServers: # Override this via parameter_defaults
default: []
description: >
  DNS servers to use for the Overcloud (2 max for some implementations).
  If not set the nameservers configured in the ctlplane subnet's
dns_nameservers attribute will be used.
type: comma_delimited_list

EC2MetadataIp:
default: "
description: The IP address of the EC2 metadata server. (The parameter
  is automatically resolved from the ctlplane subnet's `host_routes` attribute.)
type: string

resources:
OsNetConfigImpl:
type: OS::Heat::SoftwareConfig
properties:
group: script
config:
  str_replace:
    template:
    params:
      $network_config:
        network_config:
        - type: interface
          name: nic1
          use_dhcp: false
defroute: false
        - type: interface
          name: nic2
          use_dhcp: false
        addresses:
        - ip_netmask:
          list_join:
            - / 
            - - get_param: ControlPlaneIp 
          - - get_param: ControlPlaneSubnetCidr 
        routes:
        - ip_netmask: 169.254.169.254/32 
          next_hop:
            get_param: EC2MetadataIp 
        - default: true 
          next_hop:
            get_param: ControlPlaneDefaultRoute 
        - 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: InternalApilpSubnet

- type: vlan
  vlan_id:
    get_param: StorageNetworkVlanID
device: bond_api
addresses:
  - ip_netmask:
    get_param: StoragelpSubnet

- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  ovs_extra:
    - str_replace:
      template: set port br-link0 tag=_VLAN_TAG_
      params:
        _VLAN_TAG_:
          get_param: TenantNetworkVlanID
addresses:
  - ip_netmask:
    get_param: TenantIpSubnet
members:
  - type: ovs_dpdk_bond
    name: dpdkbond0
    mtu: 9000
    rx_queue: 2
    members:
      - type: ovs_dpdk_port
        name: dpdk0
        members:
          - type: interface
            name: nic7
      - type: ovs_dpdk_port
        name: dpdk1
        members:
          - type: interface
            name: nic8
      - type: interface
        name: p7p3
        mtu: 9000
        use_dhcp: false
defroute: false
nm_controlled: true
hotplug: true

- type: interface
  name: p7p4
  mtu: 9000
  use_dhcp: false
  defroute: false
  nm_controlled: true
  hotplug: true

outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value:
      get_resource: OsNetConfigImpl

A.1.5. overcloud_deploy.sh

#!/bin/bash

openstack overcloud deploy \\n  --templates \\n  -r /home/stack/ospd-14-vxlan-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/roles_data.yaml \\
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \\
  -e /usr/share/openstack-tripleo-heat-templates/environments/host-config-and-reboot.yaml \\
  -e /usr/share/openstack-tripleo-heat-templates/environments/neutron-ovs-dpdk.yaml \\
  -e /usr/share/openstack-tripleo-heat-templates/environments/neutron-sriov.yaml \\
  -e /usr/share/openstack-tripleo-heat-templates/environments/collectd-environment.yaml \\
  -e /home/stack/ospd-14-vxlan-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/api-policies.yaml \\
  -e /home/stack/ospd-14-vxlan-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/network-environment.yaml \\
  -e /home/stack/overcloud_images.yaml \\
  --log-file overcloud_install.log &> overcloud_install.log