VPC-DI Installation Notes

This guide assumes that components of VPC-DI have been properly installed to run in virtual machines (VMs) on commercial off-the shelf (COTS) servers. This chapter provides some installation notes that may assist in the installation process.

Creating a Boot Parameters File

The boot parameters file provides a means to pass configuration items to StarOS before it boots. The parameters are typically necessary to successfully load StarOS and specify items such as virtual slot number, VM type, NIC assignment and network bonding configuration.

By default, VPC-DI assigns the vNIC interfaces in the order offered by the hypervisor. To configure your vNICs manually according to a specific order, you need to create a boot parameters file. You also must create a boot parameters file if you want to enable a VNFM interface.

The boot parameters are sourced in multiple ways, with all methods using the same parameter names and usage. The first location for the boot parameters file is on the first partition of the first VM drive, for example, */boot1/param.cfg*. The second location searched is on the configuration drive, which is a virtual CD-ROM drive. If you are using OpenStack, specify the target boot parameters file name as *staros_param.cfg*. If you are not using OpenStack, create an ISO image with *staros_param.cfg* in the root directory and attach this ISO to the first virtual CD-ROM drive of the VM.

As the VM boots, the param.cfg file is parsed first by the preboot environment known as CFE. Once the VM starts Linux, the virtual CD-ROM drive is accessed to parse the *staros_param.cfg* file. If there are any conflicts with values stored in the */boot1/param.cfg* file, parameters in staros_param.cfg take precedence.

If you do not create a boot parameters file, the default file is used. If you create a boot parameters file, all parameters described in Configuring Boot Parameters, on page 7 must be defined.

Format of the Boot Parameters File

The structure of the boot parameters file is:

VARIABLE_NAME = VALUE

Specify one variable per line with a newline as the end of the line terminator (UNIX text file format). Variable names and values are case insensitive. Invalid values are ignored and an error indication is displayed on the VM console. If there are duplicate values for a variable (two different values specified for the same variable name), the last value defined is used.

Numeric values do not need to be zero padded. For example a PCI_ID of 0:1:1.0 is treated the same as 0000:01:01.0.

Network Interface Roles

Network interfaces serve specific roles depending on whether the VM is used for a CF or SF.

All system VMs have a network interface connection to the DI internal network. This network links all the VMs in a VPC-DI instance together. This network must be private to a VPC-DI instance and is configured by the system software.

All VMs have the option of configuring a network interface that is connected to the virtual network function (VNF) manager (VNFM) if it exists. This interface can be configured via DHCP or static IP assignment and is used to talk to a VNFM or higher level orchestrator. This interface is enabled before the main application starts.

On CFs, one additional interface connects to the management network interface. This interface is typically configured in StarOS and should be part of the Day 0 configuration. The management interface supports static address assignment through the main StarOS configuration file.

On SFs, an additional 0 to 12 network interfaces serve as service ports. These interfaces are configured by StarOS. Typically these ports are configured as trunk ports in the VNF infrastructure (VNFI).

Table 1: Network Interface Roles

Interface Role	Description
DI_INTERFACE	Interface to the DI internal network, required for all VM types
MGMT_INTERFACE	Interface to the management port on the CF VM
SERVICE#_INTERFACE	Service port number # on the SF VM, where # can be from 1 to 12.
VNFM_INTERFACE	Optional network interface to the VNFM or orchestrator, valid for all VM types

Note Although VIRTIO interfaces can be used for the DI_INTERFACE role and the SERVICE#_INTERFACE roles, they are not recommended.

Network Interface Identification

By default the first NIC found by a VPC-DI VM is assigned the DI internal network role. Additional ports serve as either the management interface on the CF or service ports on the SF. No interface is used as the VNFM interface by default.

VPC-DI assigns the vNIC interfaces in the order offered by the hypervisor. You cannot be guaranteed that the order of the vNICs as listed in the hypervisor CLI/GUI is the same as how the hypervisor offers them to the VM.

The order that VPC-DI finds the vNICs is subject to the PCI bus enumeration order and even paravirtual devices are represented on the PCI bus. The PCI bus is enumerated in a depth first manner where bridges are explored before additional devices at the same level. If all the network interfaces are of the same type then knowing the PCI topology is sufficient to get the vNIC order correct. If the network interfaces are of different types, then the order is dependent on the PCI topology plus the device driver load order inside the VM. The device driver load order is not guaranteed to be the same from software release to release but in general paravirtual devices are prior to pass-through devices.

There are several methods available to identify NICs.

- MAC address: MAC address of the interface
- Virtual PCI ID
- Bonded interfaces: When using network device bonding, network interfaces are identified to serve as the slave interface role. The slave interfaces in the bond are identified using MAC, PCI ID, or Interface type.
- Interface type and instance number.

Virtual PCI ID

Devices on a PCI bus are identified by a unique tuple known as the domain, bus, device, and function numbers. These identifiers can be identified in several ways.

Inside the guest, the **lspci** utility shows the bus configuration:

```
# lspci

00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.2 USB controller: Intel Corporation 82371SB PIIX3 USB [Natoma/Triton II] (rev 01)
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 VGA compatible controller: Cirrus Logic GD 5446
00:03.0 System peripheral: Intel Corporation 6300ESB Watchdog Timer
00:04.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
00:05.0 Ethernet controller: Red Hat, Inc Virtio network device
00:06.0 Ethernet controller: Red Hat, Inc Virtio network device
```

The domain, bus, device, and function numbers for this virtual bus are shown here:

Table 2: Virtual PCI IDs

Line	Domain	Bus	Device	Function
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)	0	0	0	0
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]	0	0	1	0

Line	Domain	Bus	Device	Function
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]	0	0	1	1
00:01.2 USB controller: Intel Corporation 82371SB PIIX3 USB [Natoma/Triton II] (rev 01)	0	0	1	2
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)	0	0	1	3
00:02.0 VGA compatible controller: Cirrus Logic GD 5446	0	0	2	0
00:03.0 System peripheral: Intel Corporation 6300ESB Watchdog Timer	0	0	3	0
00:04.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon	0	0	4	0
00:05.0 Ethernet controller: Red Hat, Inc Virtio network device	0	0	5	0
00:06.0 Ethernet controller: Red Hat, Inc Virtio network device	0	0	6	0

For libvirt-based virtual machines, you can get the virtual PCI bus topology from the **virsh dumpxml** command. Note that the libvirt schema uses the term *slot* for the device number. This is a snippet of the xml description of the virtual machine used in the previous example:

```
<interface type='bridge'>
  <mac address='52:54:00:c2:d0:5f'/>
  <source bridge='br3043'/>
  <target dev='vnet0'/>
  <model type='virtio'/>
  <driver name='vhost' queues='8'/>
  <alias name='net0'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>
</interface>
<interface type='bridge'>
  <mac address='52:54:00:c3:60:eb'/>
  <source bridge='br0'/>
  <target dev='vnet1'/>
  <model type='virtio'/>
  <alias name='net1'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
</interface>
```

Interface Type and Instance Number

Here the NIC is identified by its type using its Linux device driver name (virtio_net, vmxnet3, ixgbe, i40e, etc) and its instance number. The instance number is based on PCI enumeration order for that type of interface starting at instance number 1. The interface type is available to identify both paravirtual types as well as pass-through interfaces and SR-IOV virtual functions. The PCI enumeration order of devices on the PCI bus can be seen from the **lspci** utility, which is on the host OS.

For example, a CF with the following guest PCI topology indicates that virtio_net interface number1 is the Ethernet controller at 00:05.0 and virtio_net interface number 2 is the Ethernet Controller at 00:06.0. The output is from the **lspci** command executed in the guest:

```
# lspci

00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.2 USB controller: Intel Corporation 82371SB PIIX3 USB [Natoma/Triton II] (rev 01)
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 VGA compatible controller: Cirrus Logic GD 5446
00:03.0 System peripheral: Intel Corporation 6300ESB Watchdog Timer
00:04.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
00:05.0 Ethernet controller: Red Hat, Inc Virtio network device
00:06.0 Ethernet controller: Red Hat, Inc Virtio network device
```

Here is the complete list of the supported Linux drivers:

Table 3: Supported Linux Drivers

Type	PCI Vendor / Device ID	Driver Name
VIRTIO (paravirtual NIC for KVM)	0x10af / 0x1000	virtio_net
VMXNET3 (paravirtual NIC for VMware)	0x15ad / 0x07b0	vmxnet3

Type	PCI Vendor / Device ID	Driver Name
Intel 10 Gigabit Ethernet	0x8086 / 0x10b6 0x8086 / 0x10c6 0x8086 / 0x10c7 0x8086 / 0x10c8 0x8086 / 0x150b 0x8086 / 0x10dd 0x8086 / 0x10ec 0x8086 / 0x10f1 0x8086 / 0x10e1 0x8086 / 0x10db 0x8086 / 0x1508 0x8086 / 0x10f7 0x8086 / 0x10fc 0x8086 / 0x1517 0x8086 / 0x10fb 0x8086 / 0x1507 0x8086 / 0x1514 0x8086 / 0x10f9 0x8086 / 0x152a 0x8086 / 0x1529 0x8086 / 0x151c 0x8086 / 0x10f8 0x8086 / 0x1528 0x8086 / 0x154d 0x8086 / 0x154f 0x8086 / 0x1557	ixgbe
Intel 10 Gigabit NIC virtual function	0x8086 / 0x10ed 0x8086 / 0x1515	ixgbevf
Cisco UCS NIC	0x1137 / 0x0043 0x1137 / 0x0044 0x1137 / 0x0071	enic

Type	PCI Vendor / Device ID	Driver Name
Mellanox ConnectX-5 **Note** . • Mellanox is supported on the User Plane. • Mellanox is supported on management interface for VPC-DI.	0x15b3 / 0x1017 0x15b3 / 0x1018	mlx5_core
Intel XL 710 family NIC (PF)	0x8086 / 0x1572 (40 gig) 0x8086 / 0x1574 (40 gig) 0x8086 / 0x1580 (40 gig) 0x8086 / 0x1581 (40 gig) 0x8086 / 0x1583 (40 gig) 0x8086 / 0x1584 (40 gig) 0x8086 / 0x1585 (40 gig) 0x8086 / 0x158a (25 gig) 0x8086 / 0x158b (25 gig)	i40e**
Intel XL 710 family NIC virtual function	0x8086 / 0x154c	i40evf

** Note: A known issue exists where MAC address assignment does not occur dynamically for SRIOV VFs created on the host when using the **i40e** driver. MAC address assignment is necessary to boot the StarOS VM. As a workaround, MAC address assignment must be configured from the host. Refer to the following link for more information:https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/xl710-sr-iov-config-guide-gbe-linux-brief.pdf

Configuring Boot Parameters

If you do not create a boot parameters file, the default file is used. If you create a boot parameters file, all parameters described in this task must be defined.

Before you begin

Refer to Network Interface Roles, on page 2 and Network Interface Identification, on page 2 for more information on determining the interface identifiers for your VM interfaces.

Step 1 **CARDSLOT**=*slot-number*

slot_number is an integer between 1 and 32 that indicates the slot number or VM. CF slots can be 1 or 2. SF slots can range from 3 to 48.

Step 2 **CARDTYPE**=*card-type*

card-type identifies whether the VM is a CF or SF.

- Use 0x40010100 for Control Function.
- Use 0x42020100 for Service Function.

Step 3 *interface-role*_**INTERFACE=***interface-id*

Valid values for *interface-role* are:

- DI
- MGMT
- SERVICE#, where # can range from 1 to 12
- VNFM

For example, DI_INTERFACE=interface-id.

Refer to Network Interface Roles, on page 2 for more information on interface roles.

Valid values for *interface-id* are:

- MAC: xx:xx:xx:xx:xx:xx
- PCI_ID:xxxx:xx:xx.x (Domain:Bus:Device.Function)
- TYPE:*drive-name–instance-number*
- BOND: *slave-interface-A*,*slave-interface-B*

Refer to Network Interface Identification, on page 2 for information on determining the interface identifier.

Example:

This example identifies the interface by its MAC address:

```
DI_INTERFACE=MAC:00:01:02:03:04:05
```

This example identifies the interface by its guest PCI address:

```
DI_INTERFACE=PCI_ID:0000:01:02.0
```

This example identifies the interface by its interface type (1st virtio interface):

```
DI_INTERFACE=TYPE:enic-1
```

Example:

This example identifies the interfaces as a network bond interface. The example illustrates identifying the interface using MAC address, PCI identifier and interface type:

```
DI_INTERFACE=BOND:MAC:00:01:02:03:04:05,MAC:00:01:02:03:04:06
# or
DI_INTERFACE=BOND:PCI_ID:0000:01:01.0,PCI_ID:0000:01:02.0
# or
DI_INTERFACE=BOND:TYPE:enic-1,TYPE:enic-2
```

Configuring Network Interface Bonding

The system supports configuring pairs of network interfaces into an active/standby bonded interface. Only one interface is active at a time and failure detection is limited to the loss of the physical link. Use this task to configure bonded interfaces.

All bonding variable names use the format *interface-role*_BOND. Refer to Network Interface Roles, on page 2 for information on interface roles.

Before you begin

All boot parameters described in this task are optional. If these parameters are required, add them to the boot parameters file together with the required parameters described in Configuring Boot Parameters, on page 7.

Step 1 *interface-role*_BOND_PRIMARY=*interface-id*

Configures the primary slave interface if you have a preference for a particular interface to be active the majority of the time. The default bond configuration does not select a primary slave.

Refer to Network Interface Roles, on page 2 for information on interface roles; refer to Network Interface Identification, on page 2 for information regarding interface identifiers.

Note By default, the reversion policy is that the bond only reverts back to the primary interface on a subsequent failure of the new active link.

By default, the failure detection method is that the bond uses the driver state to poll link status of the underlying interfaces.

Example:

This example specifies the primary interface using a MAC address:

```
DI_INTERFACE_BOND_PRIMARY=MAC:00:01:02:03:04:05
```

This example specifies the primary interface using a PCI identifier:

```
DI_INTERFACE_BOND_PRIMARY=BOND:PCI_ID:0000:01:01.0
```

This example specifies the primary interface using an interface type identifier:

Example:

```
DI_INTERFACE_BOND_PRIMARY=BOND:TYPE:enic-1
```

Step 2 *interface-role*_BOND_MII_POLL = *poll-interval*

Specifies the poll interval, in milliseconds, to use when MII is used for link detection. The poll interval can range from 0 to 1000. The default is 100.

Step 3 *interface-role*_BOND_MII_UPDELAY=*slave-enable-delay*

Specifies how long to wait for the link to settle before enabling a slave interface after a link failure, when MII is used for link detection. The link state can bounce when it is first detected. This delay allows the link to settle before trying to use the interface and thereby avoids excessive flips in the active slave for the bond interface.

The slave enable delay must be a multiple of the MII poll interval. Values are in milliseconds and the default is 0.

Step 4 *interface-role*_BOND_MII_DOWNDELAY=*slave-disable-delay*

Optional. When used, it allows the bond to wait before declaring that the slave interface is down, when MII is used for link detection. The slave disable delay must be a multiple of the MII poll interval. Values are in milliseconds and the default is 0.

Configuring a VNFM Interface

A virtual network function management (VNFM) interface is designed to communicate between each VM and a VNFM. This interface is brought up before the main application and can be configured only using the boot parameters. The VNFM interface is disabled by default.

Use this task to configure a VNFM interface:

Before you begin

All boot parameters described in this task are optional. If these parameters are required, add them to the boot parameters file together with the required parameters described in Configuring Boot Parameters, on page 7.

Step 1 VNFM_IPV4_ENABLE={true | false}

Enables the VNFM interface.

Step 2 VNFM_CARTRIDGE_AGENT={true | false}

Enables the cartridge agent. This must be enabled if the VNFM is using the cartridge agent.

Step 3 VNFM_IPV4_DHCP_ENABLE={true | false}

Enables DHCP on the VNFM.

Step 4 VNFM_IPV4_ADDRESS=*x.x.x.x*

Specifies the IP address for the VNFM where DHCP is not used.

Step 5 VNFM_IPV4_NETMASK=*x.x.x.x*

Specifies the netmask for the IP address of the VNFM where DHCP is not used.

Step 6 VNFM_IPV4_GATEWAY=*x.x.x.x*

Specifies the gateway for the IP address of the VNFM where DHCP is not used.

VNFM Interface Options

Note These configuration options are optional.

The virtual network functions manager (VNFM) interface is designed to communicate between each VM and a VNFM. The VNFM interface initializes before the main application and only boot parameters can configure the interface.

The VNFM interface is disabled by default.

Enable VNFM IPv4 Interface

The default value is False (disabled).

Variable	Valid Values
VNFM_IPV4_ENABLE	True or False

Configure IPv4 DHCP Client

Variable	Valid Values
VNFM_IPV4_DHCP_ENABLE	True or False

Configure IPv4 Static IP

Note If IPv4 DHCP client is enabled, static configuration parameters are ignored.

Variable	Valid Values
VNFM_IPV4_ADDRESS	x.x.x.x
VNFM_IPV4_NETMASK	x.x.x.x
VNFM_IPV4_GATEWAY	x.x.x.x

Enable VNFM IPv6 Interface.

Variable	Valid Values
VNFM_IPV6_ENABLE	True or False

Enable IPv6 Static IP Configuration

Variable	Valid Values
VNFM_IPV6_STATIC_ENABLE	True or False

If set to true, static IP parameters configuration applies to the interface as shown in the following section. If set to false, the interface attempts to use both stateless autoconfiguration (RFC4862) and DHCPv6 to configure the address of the interface.

Configure IPv6 Static IP

Note If the "VNFM_IPV6_ENABLE" parameter value is set to false, the static configuration parameters are ignored. The IPv6 address field should conform to RFC 5952. Prefix is fixed at /64.

Variable	Valid Values
VNFM_IPV6_ADDRESS	x:x:x:x:x:x:x:x
VNFM_IPV6_GATEWAY	x:x:x:x:x:x:x:x

Configuring the DI Network VLAN

The DI network requires a unique and isolated network available for its use. When using pass-through interfaces, a VLAN ID can be configured to allow for easier separation of the VPC-DI instances in the customer network. Optionally, the DI Network VLAN can also be tagged on the host or even the L2 switch, if there are dedicated ports on the host.

Use this task to configure the VLAN.

Before you begin

All boot parameters described in this task are optional. If these parameters are required, add them to the boot parameters file together with the required parameters described in Configuring Boot Parameters, on page 7.

DI_Internal_VLANID=*vlan-id*

Specifies a VLAN ID for the internal DI network. Values can range from 1 to 4094.

Example:

```
DI_INTERNAL_VLANID=10
```

Configuring IFTASK Tunable Parameters

By default, DPDK allocates 30% of the CPU cores to the Internal Forwarder Task (IFtask) process. You can configure the resources allocated to IFTASK using these boot parameters. Use the **show cpu info** and **show cpu verbose** commands to display information regarding the CPU core allocation for IFTASK.

Note These are optional parameters that should be set with extreme care.

Step 1 (Optional) **IFTASK_CORES**=*percentage-of-cores*

Specify the percentage of CPU cores to allocate to IFTASK. Values can range from 0 to 100 percent. The default is 30.

Step 2 (Optional) **MCDMA_THREAD_DISABLE=***percentage-of-iftask-cores*

Set the MCDMA_THREAD_DISABLE parameter to 1 to run PMDs on all cores, rather than using an MCDMA - VNPU split.

Step 3 (Optional) **IFTASK_SERVICE_TYPE=***value*

Specifies the service type being deployed in order to calculate the service memory and enable service-specific features. The following service types can be specified:

- 0 = VPC service type
- 1 = GiLAN service type
- 2 = ePDG service type
- 3 = CUPS controller service type
- 4 = CUPS forwarder service type

The default is 0.

Step 4 (Optional) **IFTASK_CRYPTO_CORES=***value*

When **IFTASK_SERVICE_TYPE** is configured to "2" (EPDG), this parameter specifies the percentages of iftask cores to allocate to crypto processing. Values can range from 0 to 50 percent, though the cores dedicate will be capped at 4. The default is 0.

Note This parameter should only be used if the **IFTASK_SERVICE_TYPE** is set to "2" (EPDG). If it is set to any other service type, then this parameter should be set to "0".

Step 5 (Optional) **IFTASK_DISABLE_NUMA_OPT=***value*

Use this setting to disable the NUMA optimizations, even though more than 1 NUMA node is presented to the VM by the host. This option can be set when NUMA optimizations are not desirable for whatever reason.

- NO = enabled (default)
- YES = disabled

NUMA optimization is enabled by default, except for the following cases:

- The number of NUMA nodes/cells does not equal 2.
- Card type is Control Function (CF), Application Function (AF), or Network Function (NF). Only Service Function (SF) VMs support NUMA.
- The service type of the VM is not VPC. NUMA is only supported for VPC service type.
- This setting is explicitly set to YES (**IFTASK_DISABLE_NUMA_OPT=YES**).

Step 6 (Optional) **IFTASK_VNPU_TX_MODE=***value*

The compute nodes in an Ultra M deployment have 28 cores. Two of these cores are reserved for use by the host. When 26 cores are utilized, this results in an unequal distribution of MCDMA channels across the cores used to perform MCDMA work.

When this setting is enabled, the MCDMA function cores in iftask are split equally as MCDMA cores and VNPU TX lookup cores.

- 0 = disabled (Default)
- 1 = enabled

Step 7 (Optional) **MULTI_SEG_MBUF_ENABLE=***value*

By default in release 21.6 and higher, the system enables the use of multi-segmented transmission/reception with smaller size buffers in all memory pools for Ixgbe pf/vf drivers. This feature reduces the overall memory size of IFTASK and makes it more suitable for small deployments.

- 1 = true (default for Ixgbe NICs).
- 0 = false (default for all other NICs).

Important Care must be taken when upgrading to 21.6 on systems which use Ixgbe NICs as this feature by default is enabled.

This feature is automatically disabled for any system not using the Ixgbe vf/pf NICs NICs.

Example

Use the StarOS command **show cloud hardware iftask** *card_number* to verify that the boot parameters took effect:

```
[local]mySystem# show cloud hardware iftask 4
Card 4:
  Total number of cores on VM:      24
  Number of cores for PMD only:     0
  Number of cores for VNPU only:    0
  Number of cores for PMD and VNPU: 3
  Number of cores for MCDMA:        4
  Number of cores for Crypto        0
Hugepage size:               2048 kB
Total hugepages:           3670016 kB
NPUSHM hugepages:                0 kB
CPU flags: avx sse sse2 ssse3 sse4_1 sse4_2
Poll CPU's: 1 2 3 4 5 6 7
KNI reschedule interval: 5 us
```

Increased Maximum IFtask Thread Support

Feature Summary and Revision History

Summary Data

Applicable Product(s) or Functional Area	All
Applicable Platform(s)	VPC-DI
Feature Default	Enabled - Always-on
Related Changes in This Release	Not applicable

Related Documentation	*VPC-DI System Administration Guide*

Revision History

Important Revision history details are not provided for features introduced before releases 21.2 and N5.1.

Revision Details	Release
From this release, the maximum number of IFtask threads configuration supported is increased to 22 cores.	21.8
First introduced.	Pre 21.2

Feature Changes

When the number of DPDK Internal Forwarder (IFTask) threads configured (in /tmp/iftask.cfg) are greater than 14 cores, the IFTask drops packets or displays an error.

Previous Behavior: Currently, the maximum number of IFtask threads configuration is limited to only 14 cores.

New Behavior: From Release 21.8, the maximum number of IFtask threads configuration supported is increased to 22 cores.

Configure MTU Size

By default, the IFTASK process sets the maximum interface MTU as follows:

- Service interfaces: 2100 bytes
- DI network interface: 7100 bytes

These default can be modified by setting the following parameters in the param.cfg file:

Table 4: MTU Size Parameters

Parameter Name	Range	Default Value
DI_INTERFACE_MTU=	576-9100	7100
SERVICE_INTERFACE_MTU=	576-9100	2100

Refer to Configure Support for Traffic Above Supported MTU, on page 15 for configuring the MTU size for a system which does not support jumbo frames.

Configure Support for Traffic Above Supported MTU

By default, jumbo frame support is required for the system to operate. If your infrastructure does not support jumbo frames, you can still run the system, however you must specify the MTU for the DI internal network

to be 1500 in the boot parameters file. This allows the IFTASK to process DI network traffic that is above the supported MTU.

Before you begin

All boot parameters described in this task are optional. If these parameters are required, add them to the boot parameters file together with the required parameters described in Configuring Boot Parameters, on page 7.

DI_INTERFACE_MTU=*1500*

Specifies that the DI internal network does not support jumbo frames so that the software handles jumbo frames appropriately.

Boot Parameters File Examples

This example shows a boot parameters file for a CF in slot 1 with two VIRTIO interfaces:

```
CARDSLOT=1
CARDTYPE=0x40010100
DI_INTERFACE=TYPE:enic-1
MGMT_INTERFACE=TYPE:virtio_net-2
```

This example shows a boot parameters file for an SF in slot 3 with three VIRTIO interfaces:

```
CARDSLOT=3
CARDTYPE=0x42020100
DI_INTERFACE=TYPE:enic-1
SERVICE1_INTERFACE=TYPE:enic-3
SERVICE2_INTERFACE=TYPE:enic-4
```

This example shows a boot parameters file for a CF with pass-through NICs, bonding configured and a DI internal network on a VLAN:

```
CARDSLOT=1
CARDTYPE=0x40010100
DI_INTERFACE=BOND:TYPE:enic-1,TYPE:enic-2
MGMT_INTERFACE=BOND:TYPE:ixgbe-3,TYPE:ixgbe-4
DI_INTERNAL_VLANID=10
```

VPC-DI Onboarding using ESC

You can use ESC to start an instance of the VPC-DI.

Onboarding the VPC-DI with ESC on OpenStack

This procedure describes how to onboard the VPC-DI on an instance of ESC in an OpenStack environment.

Before you begin

This procedure assumes that you have ESC created in a functioning OpenStack environment running release Juno or later with network access. For the detailed ESC installation procedure, refer to the *Cisco Elastic Services Controller 2.3 Install and Upgrade Guide*: http://www.cisco.com/c/en/us/td/docs/net_mgmt/elastic_services_controller/2-3/install/guide/Cisco-Elastic-Services-Controller-Install-Upgrade-Guide-2-3.html. For a description of the ESC configuration, refer to the *Cisco Elastic Services Controller 2.3 User Guide* : http://www.cisco.com/c/en/us/td/docs/net_mgmt/elastic_services_controller/2-3/user/guide/Cisco-Elastic-Services-Controller-User-Guide-2-3.html

Refer to the *Release Notes* to determine the version of Elastic Services Controller supported for this release.

Step 1 Get the qcow images of the VPC-DI instances for CF and SF.

The tarball file with the images is named something similar to `production.xxxxx.qvpc-di.qcow2.tgz`, depending on the release number. This archive includes two images for the CF and SF respectively: qvpc-di-cf.qcow2 and qvpc-di-xf.qcow2.

Step 2 Create the VPC-DI images in glance using the command **glance image-create**.

Example:

```
$ glance image-create --file qvpc-di-cf.qcow2 --container-format bare --disk-format
    qcow2 --is-public true --name cisco-qvpc-cf

$ glance image-create --file qvpc-di-xf.qcow2 --container-format bare --disk-format
    qcow2 --is-public true --name cisco-qvpc-xf
```

Step 3 Get the VPC-DI sample initialization tarball (vpc_esc_sample.tgz).

Step 4 Copy the VPC-DI sample initialization tarball to the ESC VM at the admin home (/home/admin/).

Step 5 Untar the VPC-DI sample initialization tarball to the vnf directory: opt/cisco/vnfs/cisco-qvpc/.

Step 6 Create the artifacts using the command line API command **esc_nc_cli edit-config** .

Example:

```
esc_nc_cli edit-config  /opt/cisco/vnfs/cisco-qvpc/dep/artifacts.xml
```

Step 7 Deploy the VPC-DI using the command **esc_nc_cli edit-config**.

Note Make sure to delete an existing deployment before redeploying the ESC. See Step 10.

The VPC-DI is deployed using the dep.xml file located in `/opt/cisco/vnfs/cisco-qvpc/dep/`. In general, you can use the default dep.xml file. If you need to customize the deployment, make any required changes to this file. For example, you can edit the chassis key that is used to create the chassis ID for your VPC-DI by editing the appropriate section in the dep.xml file:

```
<property>
  <name>CHASSIS_KEY</name>
  <value>164c03a0-eebb-44a8-87fa-20c791c0aa6d</value>
</property>
```

For a complete description of the dep.xml file, refer to: http://www.cisco.com/c/en/us/td/docs/net_mgmt/elastic_services_controller/2-2/deployment/Cisco-Elastic-Services-Controller-2-2-Deployment-Attributes.pdf

Example:

```
esc_nc_cli edit-config /opt/cisco/vnfs/cisco-qvpc/dep/dep.xml
```

This command may create a new tenant Core, if it does not already exist. Depending on which dep*.xml is used, the deployment might hit a "quota exceeded" error. If all SF and CFs do not boot, verify the default tenant quota and quota for tenant Core in OpenStack. The command to do this is `$ nova quota-defaults; nova quota-show --tenant Core`

Step 8 Verify the deployment status in the log file `var/log/esc/yangesc.log`.

Step 9 Wait for the VPC-DI to converge.

Step 10 To delete the VPC-DI deployment, use the ESC CLI command: **esc_nc_cli delete-dep** *Tenant deployment-name*

Example:

```
$ ./esc_nc_cli delete-dep Core cisco-qvpc
```

Step 11 (Optional) To use custom monitoring, before deploying the VPC-DI (Step 7) do the following:

a) Copy the mib file.

Example:

```
sudo cp /opt/cisco/vnfs/cisco-qvpc/config/starent.my /usr/share/snmp/mibs/
```

b) Add dynamic mapping metrics for the VPC-DI to any existing mappings. Merge the content of the file /opt/cisco/vnfs/cisco-qvpc/config/dynamic_mappings_snippet.xml to /opt/cisco/esc/esc-dynamic-mapping/dynamic_mappings.xml. Extra care should be taken for this step. For each SF, there needs to be one dynamic mapping in /opt/cisco/esc/esc-dynamic-mapping/dynamic_mappings.xml.

Customizing the VPC-DI Onboarding with ESC

After onboarding the VPC-DI with ESC, all VPC-DI files are located in the directory /opt/cisco/vnfs/cisco-qvpc/. There are numerous changes that can be made to the installation of your VPC-DI system. The following files can be altered as required.

VNF Deployment File

The deployment file is copied to your disk to the location: /opt/cisco/vnfs/cisco-qvpc/dep/.

You can create additional files as required.

Boot Parameter Files

You have several different boot parameter files for the various CF and SF VMs placed on your disk when you onboard the VPC-DI with ESC. These files are copied to: /opt/cisco/vnfs/cisco-qvpc/config/.

- param.cfg
- param_sf.cfg

These can be customized according to your needs. Refer to Creating a Boot Parameters File, on page 1 for more information on the boot parameters file format.

An example boot parameter file for a CF is shown here:

```
CARDSLOT=$SLOT_CARD_NUMBER
CPUID=0
```

```
CARDTYPE=$CARD_TYPE_NUM

DI_INTERFACE=TYPE:enic-1
VNFM_INTERFACE=TYPE:virtio_net-2
MGMT_INTERFACE=TYPE:virtio_net-3

VNFM_IPV4_ENABLE=true
VNFM_IPV4_DHCP_ENABLE=true
```

An example boot parameter file for an SF is shown here:

```
CARDSLOT=$SLOT_CARD_NUMBER
CPUID=0
CARDTYPE=$CARD_TYPE_NUM

DI_INTERFACE=TYPE:enic-1
VNFM_INTERFACE=TYPE:enic-2
SERVICE1_INTERFACE=TYPE:enic-3

VNFM_IPV4_ENABLE=true
VNFM_IPV4_DHCP_ENABLE=true
```

Configuration File

You can customize the configuration file located at /cisco/images/system.cfg. An example of a standard configuration file is shown here:

```
config
system hostname $VPC_HOSTNAME
clock timezone $TIMEZONE
context local
administrator admin password $ADMIN_PASS ftp

interface LOCAL1
ip address $CF_VIP_ADDR $CF_VIP_NETMASK
ip route 0.0.0.0 0.0.0.0 $NICID_1_GATEWAY LOCAL1
ip domain-lookup
ip domain-name $CF_DOMAIN_NAME
ip name-servers $CF_NAME_SERVER
ssh generate key
server sshd
subsystem sftp
port ethernet 1/1
bind interface LOCAL1 local
no shutdown
snmp community $SNMP_COMMUNITY read-only
end
```

Refer to Understanding Configuration Files for more information.

OpenStack Performance Optimizations

Cisco ESC allows a number of hypervisor optimizations using OpenStack Kilo release, such as non-uniform memory access (NUMA) node configuration, support for huge pages and the pinning of guest vCPUs.

- vCPU pinning—ability for guest vCPUs to be strictly pinned to a set of host physical CPUs. This prevents the potential wait by a vCPU for physical resources to become available.

- Large pages—allocation of larger blocks of memory to virtual resources.
- PCI-based NUMA scheduling—ability to assign a PCI device to an instance in OpenStack.

Note These OpenStack performance optimizations are supported using OpenStack Kilo version only.

OpenStack Kilo exposes various hardware acceleration features using a variety of mechanisms. One such mechanism involves the setting of key-value attributes on flavor objects. VM instantiation requests which reference a given flavor effectively request the corresponding hardware acceleration features. Provided the necessary OpenStack configuration is in place on the OpenStack control and compute nodes, the OpenStack compute service ("nova") selects an appropriate compute node, and assigns the corresponding resources.

To use hardware acceleration on the VPC-DI you must create new flavors in Cisco ESC and add the required metadata attributes using the NETCONF or REST interface. For information regarding flavors, refer to *Managing Flavors* in the *Cisco Elastic Services Controller User Guide*: http://www.cisco.com/c/en/us/support/cloud-systems-management/elastic-services-controller-2-1/model.html. You may also require a number of OpenStack configuration changes. The procedures for implementing hardware acceleration are described in these tasks.

Note Adding metadata is only supported for new flavors, not for existing ones. If metadata attributes need to be added to existing flavors, you must interact with OpenStack directly.

Configuring CPU Pinning

Step 1 On each OpenStack control node, configure the scheduler.

a) Configure the scheduler filters to include the NUMAToplogy and AggregateInstanceExtraSpec filters:

Example:

```
$ sudo vim /etc/nova/nova.conf
...
scheduler_default_filters=RetryFilter,AvailabilityZoneFilter,RamFilter,ComputeFilter,
ComputeCapabilitiesFilter,ImagePropertiesFilter,CoreFilter,NUMATopologyFilter,
AggregateInstanceExtraSpecsFilter
```

b) Restart the nova scheduler service.

Example:

```
$ sudo systemctl restart openstack-nova-scheduler.service
```

Step 2 On each relevant OpenStack compute node, configure which hypervisor processes are used for guests and which are not to be used for guests.

a) Ensure that hypervisor processes do not run on cores reserved for guests.

For example, to reserve cores 2, 3, 6 and 7 for guests, update the grub bootloader and reboot.

Example:

```
$ sudo grubby --update-kernel=ALL --args="isolcpus=2,3,6,7"
$ sudo grub2-install /dev/sda
$ sudo reboot
```

b) Verify the kernel command line reflects the change.

Example:

```
$ cat /proc/cmdline
...  isolcpus=2,3,6,7 ...
```

c) Configure guest virtual machine instances so that they are only allowed to run on specific cores and reserve RAM for hypervisor processes. For example, to use cores 2, 3, 6 and 7 and reserve 512 MB for hypervisor processes:

Example:

```
$ sudo vim /etc/nova/nova.conf
...
vcpu_pin_set=2,3,6,7
...
reserved_host_memory_mb=512
...
```

d) Restart the nova compute service.

Example:

```
$ sudo systemctl restart openstack-nova-compute.service
```

Step 3 Configure the global parameters.

a) Create a `performance-pinned` host aggregate for hosts that received pinning requests, and add an arbitrary attribute `pinned=true` to identify it.

Example:

```
$ nova aggregate-create performance-pinned
$ nova aggregate-set-metadata performance-pinned pinned=true
```

b) Create the normal aggregate for all other hosts and add the same arbitrary attribute, but set `pinned=false` to identify it.

Example:

```
$ nova aggregate-create normal
$ nova aggregate-set-metadata normal pinned=false
```

c) Add the previously enabled compute nodes to the `performance-pinned` host aggregate, and add all other compute nodes to the `normal` host aggregate.

Example:

```
$ nova aggregate-add-host normal             compute100.cloud.com
$ nova aggregate-add-host normal             compute101.cloud.com
$ nova aggregate-add-host normal             compute102.cloud.com
$ nova aggregate-add-host performance-pinned compute103.cloud.com
$ nova aggregate-add-host performance-pinned compute104.cloud.com
$ nova aggregate-add-host performance-pinned compute105.cloud.com
```

Step 4 Configure the flavor attributes using the Cisco ESC northbound API.

a) Set the flavor attributes.

- hw:cpu_policy=dedicated
- aggregate_instance_extra_specs:pinned=true

All instances created using this flavor are sent to hosts in host aggregates with pinned=true in their aggregate metadata.

Example:

```
 version='1.0' encoding='ASCII'?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <flavors>
    <flavor>
      <name>testfl6</name>
      <vcpus>1</vcpus>
      <memory_mb>2048</memory_mb>
      <root_disk_mb>10240</root_disk_mb>
      <ephemeral_disk_mb>0</ephemeral_disk_mb>
      <swap_disk_mb>0</swap_disk_mb>
      <properties>
        <property>
          <name>hw:cpu_policy</name><value>dedicated</value>
          <name>aggregate_instance_extra_specs:pinned</name><value>true</value>
        </property>
      </properties>
    </flavor>
  </flavors>
</esc_datamodel>
```

b) Update all other flavors to so that their extra specifications match them to the compute hosts in the normal aggregate.

Example:

```
$ nova flavor-key <flavor_Id> set aggregate_instance_extra_specs:pinned=false
```

c) To verify: launch a VM instance using the modified flavor and locate the compute node where the VM instance was started.

Example:

```
$ nova boot --image <test-image> --flavor <modified-flavor> test-instance
$ nova show test-instance | egrep
'OS-EXT-SRV-ATTR:hypervisor_hostname|OS-EXT-SRV-ATTR:instance_name'
| OS-EXT-SRV-ATTR:hypervisor_hostname  | compute3.cloud.com
| OS-EXT-SRV-ATTR:instance_name        | instance-00000cee
```

d) Log into the returned compute node and use the **virsh** tool to extract the XML of the returned instance.

Example:

```
$ ssh compute3.cloud.com
...
$ sudo virsh dumpxml instance-00000cee
...
  <vcpu placement='static' cpuset='2-3,6-7'>1</vcpu>
```

Configuring Huge Pages

Use this procedure to configure huge pages on your OpenStack configuration.

Step 1 Edit /etc/sysctrl.conf to configure the number of pages.

Example:

```
vm.nr_hugepages = 32768
```

Step 2 Edit /proc/meminfo to set the page size.

Example:

```
Hugepagesize: 2048 kB
```

Step 3 Create a flavor with hw:mem_page_size=2048 using the Cisco ESC northbound API.

Example:

```
<?xml version='1.0' encoding='ASCII'?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
  <flavors>
    <flavor>
      <name>testfl6</name>
      <vcpus>1</vcpus>
      <memory_mb>2048</memory_mb>
      <root_disk_mb>10240</root_disk_mb>
      <ephemeral_disk_mb>0</ephemeral_disk_mb>
      <swap_disk_mb>0</swap_disk_mb>
      <properties>
        <property>
          <name>hw:mem_page_size</name><value>2048</value>
        </property>
      </properties>
    </flavor>
  </flavors>
</esc_datamodel>
```

Configuring PCI Passthrough

The Intel VT-d extensions provide hardware support for directly assigning a physical device to a guest. This allows virtual devices to avoid the throughput loss and reduced packet forwarding capacity that would be involved in traversing multiple switches or bridges to reach a physical interface.

The VT-d extensions are required for PCI passthrough with Red Hat Enterprise Linux. The extensions must be enabled in the BIOS. Some system manufacturers disable these extensions by default.

This procedure does not cover enabling VT-d from the BIOS perspective; refer to your server manufacture BIOS configuration guide to enable VT-d. From the Linux kernel perspective, VT-d is enabled by adding "intel_iommu=on" to grub config.

Before you begin

VT-d must be enabled on the Intel chipset to support PCI Passthrough.

Step 1 Enable VT-d support on Red hat Enterprise Linux.

a) Edit "/etc/default/grub" and add "intel_iommu=on" to the end of the line: GRUB_CMDLINE_LINUX.

Example:

```
GRUB_TIMEOUT=5
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="rd.lvm.lv=rhel/swap crashkernel=auto rd.lvm.lv=rhel/root rhgb quiet
intel_iommu=on"
GRUB_DISABLE_RECOVERY="true"
```

b) Regenerate grub.conf and reboot your server by running the grub2-mkconfig command.

Example:

```
grub2-mkconfig -o /boot/grub2/grub.cfg on BIOS systems
```

or

```
grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg on UEFI systems
```

c) Check that IOMMU gets activated by running the command **dmesg | grep -iE "dmar|iommu"**

Example:

Sample output from the **dmesg** command with IOMMU enabled:

```
[ 0.000000] Kernel command line: BOOT_IMAGE=/vmlinuz-3.10.0-229.el7.x86_64
root=/dev/mapper/rhel-root
ro rd.lvm.lv=rhel/swap crashkernel=auto rd.lvm.lv=rhel/root rhgb quiet intel_iommu=on
[ 0.000000] Intel-IOMMU: enabled
```

Step 2 Unbind the device from the Linux kernel.

a) For PCI passthrough to work, a device must be unbound from the Linux kernel driver. To accomplish this, pci_stub module is used.

Example:

```
Load pci_stub module "modprobe pci_stub"
```

b) Locate the network adapter you want to use for PCI passthrough. Run **lspci** and note the PCI address of the desired network card.

Example:

In this example, it is desired to use PCI device 15:00.0 for PCI passthrough.

```
12:00.0 Ethernet controller: Cisco Systems Inc VIC Ethernet NIC (rev a2)
13:00.0 Ethernet controller: Cisco Systems Inc VIC Ethernet NIC (rev a2)
14:00.0 Ethernet controller: Cisco Systems Inc VIC Ethernet NIC (rev a2)
15:00.0 Ethernet controller: Cisco Systems Inc VIC Ethernet NIC (rev a2)
```

c) Identify the Vendor ID and Device ID. Run **lspci -n**.

Example:

In this partial output, the Vendor ID 1137 and Device ID 0071 are identified.

```
11:00.0 0c04: 1137:0071 (rev a2)
12:00.0 0200: 1137:0071 (rev a2)
13:00.0 0200: 1137:0071 (rev a2)
14:00.0 0200: 1137:0071 (rev a2)
15:00.0 0200: 1137:0071 (rev a2)
```

d) Use this configuration to unbind the desired device from the Linux kernel driver.

Note that highlighted text must be changed for your device information.

Example:

```
echo "1137 0071" > /sys/bus/pci/drivers/pci-stub/new_id
echo 0000:15:00.0 > /sys/bus/pci/devices/0000:15:00.0/driver/unbind
echo 0000:15:00.0 > /sys/bus/pci/drivers/pci-stub/bind
```

e) Verify the success of these command by running **dmesg | grep stub**.

Example:

```
[ 276.705315] pci-stub 0000:15:00.0: claimed by stub
```

f) To make the changes persistent, add the pci-stub.ids to grub CMDLINE, update grub and reboot the host.

Note Note that this code applies to all vNICs with the specifiedVender/Device ID (1137:0071 in this example).

Example:

```
edit /etc/default/grub
GRUB_CMDLINE_LINUX="..pci-stub.ids=1137:0071"
grub2-mkconfig -o /boot/grub2/grub.cfg
reboot
```

Step 3 Configure nova.conf in OpenStack.

a) Specify which PCI devices are available for PCI passthrough, either using (vendorID, productID) combinations to allow any PCI device with that (vendorid, productid) combination to be passed through, or by specifying PCI addresses of devices allowed to be passed through.

```
pci_passthrough_whitelist={"vendor_id": "1137", "product_id": "0071"}
```

or

```
pci_passthrough_whitelist = [ {"address": "01:00.1"}, {"address": "02:00.1"} ]
```

b) Specify the PCI alias to (product ID, vendor ID) combination mapping. Note that this setting does currently not support PCI addresses, so all PCI devices on the whitelist have the same name.

Example:

```
pci_alias={"vendor_id":"1137", "product_id":"0071", "name":"nic1"}
```

c) Make the following additional changes to nova.conf

Example:

```
scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler
scheduler_available_filters=nova.scheduler.filters.all_filters
scheduler_available_filters=nova.scheduler.filters.pci_passthrough_filter.PciPassthroughFilter
scheduler_default_filters=RamFilter,ComputeFilter,AvailabilityZoneFilter,ComputeCapabilitiesFilter,ImagePropertiesFilter
```

d) Restart nova.

Example:

```
openstack-service restart nova
```

Step 4 Create a flavor with the attribute pci_passthrough:alias set to <PCI_DEVICE_ALIAS>:<NUM_DEVICES_REQUESTED>. The PCI_DEVICE_ALIAS references values from the pci_alias setting in /etc/nova/nova.conf.

Example:

```
$ cat fl.xml
<?xml version='1.0' encoding='ASCII'?>
<esc_datamodel xmlns="http://www.cisco.com/esc/esc">
    <flavors>
        <flavor>
            <name>testfl6</name>
              <vcpus>1</vcpus>
              <memory_mb>2048</memory_mb>
              <root_disk_mb>10240</root_disk_mb>
              <ephemeral_disk_mb>0</ephemeral_disk_mb>
              <swap_disk_mb>0</swap_disk_mb>
              <properties><property>
                  <name>pci_passthrough:alias</name><value>nic1g:1</value>
              </property></properties>
        </flavor>
    </flavors>
</esc_datamodel>
$ sudo /opt/cisco/esc/esc-confd/esc-cli/esc_nc_cli edit-config ./fl.xml
```

Onboarding the VPC-DI with Heat Orchestration Templates (HOT) in OpenStack

VPC-DI can be deployed as a Virtual Network Function (VNF) in an Openstack environment. VPC-DI runs as a collection of Virtual machines and the VMs have specific requirements with respect to storage, networking, and configuration. In the Openstack environment, the Orchestrator is responsible for creating the objects required to bring up the VPC-DI VMs. The orchestrator is also responsible for creating and terminating the VMs and their associated objects. The orchestrator interfaces with Openstack services to create such entities in Openstack.

Openstack provides a service called HEAT orchestration templates (HOT) which defines the network, compute and storage topology of a VNF via a template. The HEAT template can be used as the blueprint for deploying an instance of the VNF.

The format of the templates and an example of the ENV parameters file are provided later in this section.

Step 1 Get the qcow images of the VPC-DI instances for CF and SF.

The tarball file with the images is named something similar to production.xxxxx.qvpc-di.qcow2.tgz, depending on the release number. When the tarball file is open there should be two images for the CF and SF: qvpc-di-cf.qcow2 and qvpc-di-xf.qcow2.

Step 2 Create all VPC-DI images in glance using the command **glance image-create**.

Example:

```
$ glance image-create --file qvpc-di-cf.qcow2 --container-format bare --disk-format
    qcow2 --is-public true --name cisco-qvpc-cf

$ glance image-create --file qvpc-di-sf.qcow2 --container-format bare --disk-format
    qcow2 --is-public true --name cisco-qvpc-sf
```

Step 3 Get the VPC-DI sample initialization tarball for HOT (vpc_HOT_sample.tgz).

Step 4 Copy the VPC-DI sample initialization tarball to your local machine.

Step 5 Untar the VPC-DI sample initialization tarball to any directory. There are two files with extensions **.yml** and **.env**.

Step 6 Edit the ENV file according to your openstack deployment.

Provide the values for your networks, availability zone, etc.

Browse to your local directory and click the **.yml** file for *Template source* and **.env** file for *Environment Source*

Step 7 Perform one of these:

- Deploy the VPC-DI using the OpenStack Dashboard by navigating to **Project>Orchestration>Stacks>Launch Stack**.
- Use the CLI to deploy the VPC-DI using HEAT with the command **heat stack-create -f di.yml -e di.env**.

Step 8 Verify the Status field is Complete, which indicates that there are Errors .

Step 9 Wait for the VPC-DI to converge.

Step 10 To delete the VPC-DI deployment, select the check box next to the stack name and click **Delete Stack**.

VPC-DI Heat Orchestration Templates

This section describes the format of the Heat templates. The VPC-DI HOT version is 2013-05-23. The template has these four sections: parameter-groups, parameters, resources, and outputs.

VPC-DI HOT Parameter Groups

The parameter_groups section allows for specifying how the input parameters should be grouped and the order to provide the parameters in. These groups are used to describe expected behavior for downstream user interfaces.

Table 5: HOT Parameters

Parameter Definition in Template	Notes

`- label: images`
`  description: CF and SF images in qvpc-di`
`  parameters:`
`  - qvpc_image_cf`
`  - qvpc_image_sf`
	List of images to be defined in the template

`- label: networks`
`  description: network configuration for DI`
`  parameters:`
`  - network_di_mgmt`
`  - network_di_internal`
`  - network_public`
`  - network_service1`
`  - network_service2`
`  - network_service3`
`  - network_service4`
	List of networks to be defined in the template
	Note This example lists four SF service ports. Up to 12 SF service portss can be defined.

VPC-DI HOT Parameters

The heat template defines a number of parameters for which you must provide values in an ENV file. Each of these parameters is described here. Each parameter definition is contained within the **parameters** section of the heat template. A sample ENV file follows the parameter descriptions.

Table 6: HOT Parameters

Parameter Definition in Template	Notes

`flavor_cf:`
`  type: string`
`  description: Flavor for Control Function VM`
`  default: m1.large`
	The name of the flavor to be used to create CFs. This can be one of the five default flavors, or a custom flavor that has been defined in OpenStack.

`flavor_sf:`
`  type: string`
`  description: Flavor for Service Function VM`
`  default: m1.large`
	The name of the flavor to be used to create SFs.

`availability_zone:`
`  type: string`
`  description: Availability_zone where the VNF`
`  should be created`
`  default: nova`
	Location in OpenStack where the VNF is created.

`qvpc_image_cf:`
`  type: string`
`  label: Active CF image file in glance`
`  description: Active CF image ID or file in glance`
`  default: qvpc-di-<`*`version`*`>-cf.qcow2`
`  constraints:`
`  - custom_constraint: glance.image`
	Name of the VPC-DI image file for the CFs. This file must have been uploaded to glance.

Parameter Definition in Template	Notes
qvpc_image_xf: type: string label: SF image file in glance description: SF image ID or file in glance default: qvpc-di-<*version*>-xf.qcow2 constraints: - custom_constraint: glance.image	Name of the VPC-DI image file for the service function VMs, that has been uploaded to glance.

Parameter Definition in Template	Notes
network_public: type: string description: Network ID or Network Name of external network default: public constraints: - custom_constraint: neutron.network	Network ID or name of the external network
network_cf_mgmt: type: string description: Management Network ID or Name default: private constraints: - custom_constraint: neutron.network	Name or identifier for the VPC-DI management network.
network_di_internal: type: string description: Unique QVPC-DI internal Network associated with this VNF default: private constraints: - custom_constraint: neutron.network	Name or identifier of the DI internal network. This is a private L2 network that interconnects the VMs in the VPC-DI.
network_service#: type: string description: Network ID or Network Name of network to use for SF service ports default: cflocal constraints: - custom_constraint: neutron.network	Name or identifier of the service port. You can define between one and 12 service ports in each SF, where # represents this number. Each service port can perform a different service.
network_core: type: string description: core network for keepalives default: core constraints: - custom_constraint: neutron.network	Name or identifier of the core network used for keepalive messages.
qvpc_vip_addr: type: string description: OAM IP Address shared between CF01 and CF02 default: <*value*> constraints: - custom_constraint: ip_addr	Virtual IP address used between the CFs.

Parameter Definition in Template	Notes
qvpc_vip_gateway: type: string description: IP Address of Default Gateway for OAM Network default: <*value*> constraints: - custom_constraint: ip_addr	The default gateway for the management network.
vnf_name: type: string description: Unique name for this VNF instance default: qvpc_di	Unique name for this VNF instance. This name is used to identify the VNF.
vnf_id: type: string description: Unique ID for this VNF instance default: 0	ID for the VNF instance.
admin_password: type: string description: Default Administrator password for DI Access default: Cisco123	
snmp_community: type: string description: READ SNMP string for this VPC instance default: public	
timezone: type: string description: TimeZone for this VF instance default: us-pacific	
cf_domain_name: type: string description: Domain for this VF instance default: localdomain	
az_cf<#>: type: string description: CF availability zone default: <*value*>	The availability zone for each of the two CF instances.
az_sf<#>: type: string description: CF availability zone default:<*value*>	The availability zone for each of the SF instances.

Each of these parameters is defined for your VNF instance using an ENV file. An example ENV file is shown here:

```
parameters:
  # flavor defined for CF and SF in AIC
  flavor_cf: vsaegw_cf
  flavor_sf: vsaegw_sf

  # availability zone where the VNF instance should be deployed
  availability_zone: avzone-kvm-az01
```

```
  # vPC-DI glance images in qcow2
  qvpc_image_cf01: QVPCCF
  qvpc_image_sf: QVPCSF

  # Neutron Networks attached to vSAEGW instancenetwork_di_mgmt: oam_protected_net
  network_di_internal: saegw_di_internal_active_net
  network_service1: saegw_gn_net
  network_service2: saegw_sgi_net
  network_service3: saegw_support_net
  network_service4: saegw_icsr_li_net

  # VNF Instance Name
  vnf_name: qvpcDI_vsaegw

  # VNF Instance ID
  vnf_id: 01

  # Administrator user password
  admin_password: cisco123

parameters:
  flavor_cf:
    type: string
    description: Flavor for Control Function VM
    default: cisco-qvpc-cf
  flavor_sf:
    type: string
    description: Flavor for Service Function VM
    default: cisco-qvpc-xf
  qvpc_image_cf:
    type: string
    label: CF image file in glance
    description: CF image ID or file in glance
    default: qvpc-di-68031-cf.qcow2
    constraints:
      - custom_constraint: glance.image
  qvpc_image_sf:
    type: string
    label: SF image file in glance
    description: SF image ID or file in glance
    default: qvpc-di-68031-xf.qcow2
    constraints:
      - custom_constraint: glance.image
  network_public:
    type: string
    description: Network ID or Network Name of external network
    default: public
    constraints:
      - custom_constraint: neutron.network
  network_cf_mgmt:
    type: string
    description: Management Network ID or Name
    default: cf-mgmt
    constraints:
      - custom_constraint: neutron.network
  network_di_internal:
    type: string
    description: Unique QVPC-DI internal Network associated with this VNF
    default: di-internal
    constraints:
      - custom_constraint: neutron.network
  network_service1:
    type: string
    description: Transport Interface (Gn/S11/S1-u/S5) in to SAEGW Context
    default: service1
```

```
    constraints:
      - custom_constraint: neutron.network
  network_service2:
    type: string
    description: Transport Interface (Data, Voice, LI VLANs) in SGi Context
    default: service2
    constraints:
      - custom_constraint: neutron.network
   network_core:
    type: string
    description: core network for keepalives
    default: core
    constraints:
      - custom_constraint: neutron.network

# vip_addr and vip_gateway are automatically retrieved from the management network
  qvpc_vip_addr:
    type: string
    description: OAM IP Address shared between CF01 and CF02
    default: 172.16.181.2
    constraints:
      - custom_constraint: ip_addr
  qvpc_vip_gateway:
   type: string
   description: IP Address of Default Gateway for OAM Network
   default: 172.16.181.1
   constraints:
      - custom_constraint: ip_addr
  vnf_name:
    type: string
    description: Unique name for this VNF instance
    default: qvpc_di
  vnf_id:
    type: string
    description: Unique ID for this VNF instance
    default: 0
  admin_password:
    type: string
    description: Default Administrator password for DI Access
    default: Cisco123
  snmp_community:
    type: string
    description: READ SNMP string for this VPC instance
    default: public
  timezone:
    type: string
    description: TimeZone for this VF instance
    default: us-pacific
  cf_domain_name:
    type: string
    description: Domain for this VF instance
    default: localdomain
  az_cf1:
    type: string
    description: CF availability zone
    default: conway1
  az_cf2:
    type: string
    description: CF availability zone
    default: conway2
  az_sf3:
    type: string
    description: SF3 availability zone
    default: conway3
```

```
  az_sf4:
    type: string
    description: SF6 availability zone
    default: conway4
```

VPC-DI HOT Resources

The resources section of the template defines the control function (CF) and service function (SF) VMs as well as each of their ports.

Management Network

```
# Create port on management network and reserve a virtual IP address
qvpc_vip_port:
  type: OS::Neutron::Port
  properties:
      network: {get_param: network_di_mgmt}
      fixed_ips:
        - subnet_id: {get_param: subnet_id_di_mgmt}

# Associate a floating IP address to the virutal port
qvpc_vip_floating_ip:
  type: OS::Neutron::FloatingIP
  properties:
      floating_network: {get_param: network_public}
      port_id: {get_resource: qvpc_vip_port}
```

The VIP port is the virtual IP port used to access the VPC-DI. The VIP port IP address is configurable in the Day 0 configuration.

HOT Resources for CF

The heat template must define each of the two CF VMs being used by the VNF. This definition includes configuring the port that connects to the DI internal network, as well as the port that connects to the CF management network, specifying the StarOS boot parameter file and the StarOS Day 0 configuration file. The definition of the first CF is shown here with an explanation; the second CF is defined in a similar way.

CF DI Internal Network

This section creates the CF DI internal network. Use this section twice, once for each of the two CFs that must be configured. # is either 1 or 2.

```
# Port connected to unique DI-network
qvpc_cf_0#_port_int:
  type: OS::Neutron::Port
  properties:
    network: {get_param: network_di_internal}
    allowed_address_pairs:
      -ip_address: "172.16.0.0/18"
```

qvpc_cf_#_port_int is port connected to the DI internal network. The value of the network is extracted from the parameter **network_di_internal** which is retrieved from the ENV file.

The property allowed_address_pairs must be in each di-internal port. Because the di_internal port is assigned an IP address by the VPC-DI in the 17.16.0.0/18 network which is different from its address in neutron, we

need to configure the allowed_address_pairs property to allow traffic on those address to pass through the port. The allowed address pair extension extends the port attribute to enable you to specify arbitrary MAC address or IP address (CIDR) pairs that are allowed to pass through a port regardless of the subnet associated with the network.

CF Management Network

This section creates the CF management network. Use this section twice, once for each of the two CFs that must be configured. # is either 1 or 2.

```
# Port connected to the management network
qvpc_cf_0#_port_mgmt:
  type: OS::Neutron::Port
  properties:
    network: {get_param: network_di_mgmt}
    allowed_address_pairs:
      - ip_address: {get_param: qvpc_vip_addr}
```

qvpc_cf_#_port_mgmt represents the port definition of the port connected to the OAM network. The value is extracted from the parameter **network_di_mgmt** which is retrieved from the ENV file.

SSH Keys

DI inter-VM communication is now only possible via authentication through externally supplied SSH keys. These keys are passed as part of the HEAT deployment. Public and private keys are required.

Generate the public and private SSH keys. Create a file called *user_key.pub* containing the public key. Create a file called *user_key* containing the private key. Ensure that both of these files are stored on the configuration drive. These files are referenced by HEAT:

```
      personality:
        "user_key.pub": |
                    ssh-rsa
<public_key>
        "user_key": |
                    -----BEGIN RSA PRIVATE KEY-----
<private_key>
                    -----END RSA PRIVATE KEY-----
```

Create CF VM

This section creates the CF VM. Use this section twice, once for each of the two CFs that must be created. # is either 1 or 2.

```
qvpc_cf_0#:
  type: OS::Nova::Server
  properties:
    # Create VM of format "<vnf_name>_cf_0#"
    name:
      str_replace:
        template: ${VF_NAME}_cf_0#
        params:
          ${VF_NAME}: {get_param: vnf_name}
    # Use active CF image and CF Flavor
    image: {get_param: qvpc_image_cf1 }
    flavor: {get_param: flavor_cf }
    networks:
```

```
  - port: {get_resource: qvpc_cf_0#_port_int}
  - port: {get_resource: qvpc_cf_0#_port_mgmt}
config_drive: True
```

The CF VM (**qvpc_cf_#**) is created with the previously defined parameters and named according to the convention "<vnf_name>_cf_#". The **vnf_name** is retrieved from the ENV file as are the image and flavor to be used to create the VNF.

StarOS Day 0 Configuration

The Day 0 configuration provided here configures the DI interface, system hostname and enables SSH and SFTP access using *personality* properties.

```
# Metadata to provide cloud-init capability to VPC-DI
  personality:
    "staros_param.cfg":
        str_replace:
            template: |
                CARDSLOT=$CARD_NUMBER
                CARDTYPE=$CARD_TYPE
                CPUID=$UUID
                DI_INTERFACE_MTU=1500
                DI_INTERFACE=TYPE:virtio_net-1
                MGMT_INTERFACE=TYPE:virtio_net-2
                VNFM_INTERFACE=TYPE:virtio_net-3
                VNFM_IPV4_ENABLE=true
                VNFM_IPV4_DHCP_ENABLE=true
                VNFM_PROXY_ADDRS=192.168.180.92,192.168.180.91,192.168.180.93
            params:
                $CARD_NUMBER: 1
                $CARD_TYPE: "0x40030100"
                $UUID: 0
    "staros_config.txt":
        str_replace:
            template: |
                config
                    system hostname $VF_NAME-cf-$CARD_NUMBER
                    clock timezone $TIMEZONE
                    ssh key-gen wait-time 0
                    context local
                        administrator admin password $ADMIN_PASS ftp
                            interface LOCAL1
                                ip address $CF_VIP_ADDR 255.255.255.0
                        #exit
                        ip route 0.0.0.0 0.0.0.0 $CF_VIP_GATEWAY LOCAL1
                        ip domain-lookup
                        ip domain-name $CF_DOMAIN_NAME
                        ip name-servers $CF_VIP_GATEWAY
                        ssh generate key
                        server sshd
                            subsystem sftp
                        #exit
                        server confd
                            confd-user admin
                    #exit
                    port ethernet 1/1
                        bind interface LOCAL1 local
                        no shutdown
                    #exit
                    snmp community $SNMP_COMMUNITY read-only
                end
```

```
                params:
                    $CARD_NUMBER: 1
                    $VF_NAME: {get_param: vnf_name}
                    $TIMEZONE: {get_param: timezone}
                    $ADMIN_PASS: {get_param: admin_password}
                    $SNMP_COMMUNITY: {get_param: snmp_community}
                    $CF_DOMAIN_NAME: {get_param: cf_domain_name}
                    $SLOT_CARD_NUMBER: 1
                    #$CF_VIP_ADDR: {get_attr: [qvpc_vip_port, fixed_ips, 0, ip_address]}
                    $CF_VIP_ADDR: 172.16.181.2
                    #$CF_VIP_GATEWAY: { get_attr: [qvpc_vip_port, subnets, 0, gateway_ip]
}
                    $CF_VIP_GATEWAY: 172.16.181.1
        "user_key.pub": |
                    ssh-rsa
<public_key>
        "user_key": |
                    -----BEGIN RSA PRIVATE KEY-----
<private_key>
                    -----END RSA PRIVATE KEY-----
```

$CARD_NUMBER refers to the number of the slot, which here is 1 but is 2 for the second CF.

HOT Resources for SF

Use the heat template to define each of the service function (SF) VMs that you want to deploy in the VPC-DI. For each SF you must configure the port to connect to the DI internal network as well as each of the service ports that you need for the SF. You can configure up to 12 service ports. This example creates a single SF that is used for an SAE gateway with four service ports. You must repeat a similar configuration for each SF required.

Define The Ports in the SF

```
# Create port for DI-Internal Network
  qvpc_sf_03_port_int:
    type: OS::Neutron::Port
    properties:
        network: {get_param: network_di_internal}
        allowed_address_pairs:
          - ip_address: "172.16.0.0/18"
```

qvpc_sf_#_port_int is the port that connects to the internal DI network. # is the number of the SF and can range from 3 to the maximum number of SFs allowed. The value of the network is extracted from the parameter **network_di_internal** which is retrieved from the ENV file.

```
# Create first service port (document as per your use)
  qvpc_sf_03_port_svc_01:
    type: OS::Neutron::Port
    properties:
      network: {get_param: network_service1}
```

qvpc_sf_#_port_svc_01 is the first service port. Ports are numbered consecutively from 1 to 12. The value of the network is extracted from the parameter **network_service1** which is retrieved from the ENV file.

```
# Create second service port (document as per your use)
  qvpc_sf_03_port_svc_02:
```

```
    type: OS::Neutron::Port
    properties:
      network: {get_param: network_service2}
      allowed_address_pairs:
        - ip_address: "192.168.10.0/24"
# Create third service port  (document as per your use)
  qvpc_sf_03_port_svc_03:
    type: OS::Neutron::Port
    properties:
      network: {get_param: network_service3}
# Create forth service port  (document as per your use)
  qvpc_sf_03_port_svc_04:
    type: OS::Neutron::Port
    properties:
      network: {get_param: network_service4}
```

The remaining three service ports are created - each retrieving the network information from the ENV file. Additional service ports can be created as required.

Create SF VM

```
qvpc_sf_03:
  type: OS::Nova::Server
  properties:
# Create VM name of format "<vnf_name>_sf_0<num>"
    name:
      str_replace:
          template: ${VF_NAME}_sf_03
          params:
            ${VF_NAME}: {get_param: vnf_name}
# Use SF image and SF Flavor
    image: { get_param: qvpc_image_sf }
    flavor: { get_param: flavor_sf }
    networks:
      - port: {get_resource: qvpc_sf_03_port_int}
      - port: {get_resource: qvpc_sf_03_port_svc_01}
      - port: {get_resource: qvpc_sf_03_port_svc_02}
      - port: {get_resource: qvpc_sf_03_port_svc_03}
      - port: {get_resource: qvpc_sf_03_port_svc_04}
    config_drive: True
```

The SF **qvpc_sf_#** is created with the name of the format 'vnf_name_sf_0#', where vnf_name is VNF name value retrieved from the ENV file and # is the slot of the SF. The values of the service ports are previously defined in the heat template. The image and flavor are also taken from the ENV file.

Each SF is defined similarly in the template.

Personality Configuration

Day 0 and Day 1 configurations are injected into the VNF using personality properties. The VPC-DI applies personality properties to the system, and expects this metadata from the HEAT template as shown here.

The personality defines the boot parameters file. Refer to Configuring Boot Parameters, on page 7 for more information on the boot parameters.

```
# Associate VM to unique slot (>2) and identify that its a SF
       config_drive: True
      personality:
        "staros_param.cfg":
```

```
            str_replace:
                template: |
                    CARDSLOT=$CARD_NUMBER
                    CARDTYPE=$CARD_TYPE
                    CPUID=$UUID
                    DI_INTERFACE_MTU=1500
                params:
                    $CARD_NUMBER: 3
                    $CARD_TYPE: "0x42070100"
                    $UUID: 0
        "user_key.pub": |
                    ssh-rsa
<public_key>
        "user_key": |
                    -----BEGIN RSA PRIVATE KEY-----
                    <private_key>
                    -----END RSA PRIVATE KEY-----
```

DI inter-VM communication is now only possible via authentication through externally supplied SSH keys. These keys are passed as part of the HEAT deployment. Public and private keys are required.

Generate the public and private SSH keys. Create a file called *user_key.pub* containing the public key. Create a file called *user_key* containing the private key. Ensure that both of these files are stored on the configuration drive. These files are referenced by HEAT as shown above.

VPC-DI HOT Outputs

The outputs section of the heat template defines the outputs from using the template. You can see the outputs by going to **Project>Orchestration>Stacks** and selecting the heat stack that you deployed. In the **Overview** tab you see any outputs from the heat stack.

You can also see output from the heat stack by running the **heat stack-show $[stack_name]** command at the command line.

Examples of types of output you might define for the VPC-DI are shown here:

```
qvpc_floating_ip:
  description: Floating IP of qvpc-di VIP
  value: { get_attr: [qvpc_vip_floating_ip, floating_ip_address]}
CF1_networks:
  description: The networks of the deployed CF-1
  value: { get_attr: [qvpc_cf_01, networks] }
CF2_networks_2:
  description: The networks of the deployed CF-2
  value: { get_attr: [qvpc_cf_02, networks] }
port_1_int:
  description: The port of the deployed server 1, di-internal
  value: { get_attr: [qvpc_cf_01_port_int, mac_address] }
port_1_mgmt:
  description: The port of the deployed server 1, cf-mgmt
  value: { get_attr: [qvpc_cf_01_port_mgmt, mac_address] }
port_2_int:
    description: The port of the deployed server 2, di-internal
    value: { get_attr: [qvpc_cf_02_port_int, mac_address] }
port_2_mgmt:
    description: The port of the deployed server 2, cf-mgmt
    value: { get_attr: [qvpc_cf_02_port_mgmt, mac_address] }
```

VMware Installation Notes

DI inter-VM communication is now only possible via authentication through externally supplied SSH keys. Public and private keys are required. These keys must be supplied prior to booting the VM as part of an ISO.

The keys must be generated on an external host and packaged in the ISO which must then be attached to the VM. The keys and the ISO files are generated as follows:

```
$ mkdir iso
$ ssh-keygen -t rsa -N "" -C "root@localhost" -f iso/user_key
$ genisoimage -o vpcdi_keys.iso iso
```

Once the ISO file is generated, power-up the VM and map to the CD-DVD ROM. From within vSphere, this is done by selecting the VM (CF or SF) from the list and clicking on the CD/DVD icon from the option bar near the top. Then select **Connect to ISO image on local disk** and choose the ISO. Repeat this for all of the VMs (CFs and SFs).

Once the keys are mapped, point the VPC-DI boot configuration to the image by setting the right boot priority and reload the VPC-DI.

Rules for VM Recovery

When you create a spare VM by cloning an old VM, you will encounter a mac address mismatch issue. The following are the VM Recovery rules for replacing to spare VM:

1. Do not use the VM suspend/resume.
2. You can shutdown/reboot the VM. It owns the slot as long as it is not deleted.
3. You can replace the faulty VM with the new one by recreating it only after deleting the faulty VM. The new VM can use the slot of the faulty VM.
4. If you want to bring back the faulty VM, you can recreate it and assign a slot that is not currently in use.