Table des matières

[chapter 4: Contrail DPDK vRoutersetup 2](#_Toc51764133)

[DPDK vRouterphysical network interface 2](#_Toc51764134)

[DPDK vRouterCPU setup 3](#_Toc51764135)

[CPU kept for Linux Operating System 4](#_Toc51764136)

[CPU allocated to the DPDK vRouter 5](#_Toc51764137)

[CPU allocated to Virtual Machines 6](#_Toc51764138)

[vRouter memory setup 7](#_Toc51764139)

[Hugepage memory configuration on the compute node operating system 7](#_Toc51764140)

[Hugepage allocation for the DPDK vrouter 7](#_Toc51764141)

[DPDK physical interface rings setup 8](#_Toc51764142)

[DPDK vRouterinternal queues rings setup 9](#_Toc51764143)

[DPDK Virtual Machine interface rings setup 11](#_Toc51764144)

[Virtual Machine vif multi-queue setup 12](#_Toc51764145)

[vRouter routing and switching object tables dimensioning parameters 13](#_Toc51764146)

**Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.Error! Hyperlink reference not valid.**

# chapter 4: Contrail DPDK vRouter setup

Contrail DPDK vRouter setup mainly consist to:

* Define NIC cards to be used by the vRouter for its interconnection with the physical network
* Define CPU resources to be allocated to the DPDK vRouter
* Define the huge pages memory to be used by the DPDK vRouter to create vRouter interface DPDK rings for physical and Virtual Machine network interface cards.
* Configure the number of queues of DPDK vRouter physical and Virtual Machine network interface cards. Queues will be configured automatically mapping 1:1 vRouter and physical NIC. For Virtual Machine NIC configured queues vRouter provides 1:1 queues till number of VM queues inf not bigger that vRouter allocated cores.

Contrail vRouter DPDK specific setup is defined into /etc/sysconfig/network-scripts/ifcfg-vhost0 configuration file. Then the vRouter agent vhost0 network interface has to be recreated to get modified setup enforced:

$ sudo ifdown vhost0

$ sudo ifup vhost0

## DPDK vRouter physical network interface

Only one physical interface can be plugged onto the vif0/0 port of the vRouter. Usually for resiliency purpose, a bond interface is created to group two physical interfaces in a single entity which is plugged onto the vRouter.

Physical NICs used into the bond interface are defined in BIND\_INT parameter:

$ vi /etc/sysconfig/network-scripts/ifcfg-vhost0

DEVICE=vhost0

DEVICETYPE=vhost

TYPE=dpdk

BIND\_INT=0000:02:01.0,0000:02:02.0

As well as others paramiters like, bond mode, policy and dirver:

BOND\_MODE=4

BOND\_POLICY=layer3+4

DRIVER=uio\_pci\_generic

Using the following command, we can display PCI identifier of physical interfaces which are available onto the Linux Operating system:

$ sudo lshw -class network | grep pci@

bus info: pci@0000:02:01.0

bus info: pci@0000:02:02.0

bus info: pci@0000:03:00.0

Once the Contrail DPDK vRouter has been started, we can see the actual physical interfaces used for the underlay network interconnection:

$ sudo docker exec contrail-vrouter-agent-dpdk /opt/contrail/bin/dpdk\_nic\_bind.py -s

Network devices using DPDK-compatible driver

============================================

0000:02:01.0 '82540EM Gigabit Ethernet Controller' drv=uio\_pci\_generic unused=e1000

0000:02:02.0 '82540EM Gigabit Ethernet Controller' drv=uio\_pci\_generic unused=e1000

Network devices using kernel driver

===================================

0000:03:00.0 'Virtio network device' if= drv=virtio-pci unused=virtio\_pci,uio\_pci\_generic

Other network devices

=====================

<none>

## DPDK vRouter CPU setup

When DPDK is used, some CPUs have to be booked for an exclusive use to the DPDK application for packet polling purpose - Poll Mode Driver (PMD). CPU allocation planning is a very important task to be done by the customer architecture team in charge of the virtual infrastructure.

On each compute node we have to allocate:

* Some CPUs to be kept available for the Linux Operating System
* Some CPUs for the Virtual Machines. Generally, this is the main purpose of the customer virtual infrastructure creation.
* Some CPUs for the vRouter high speed packet processing (PMD).

The customer virtual infrastructure architect is starting to define the number of CPUs to be allocated in each group described above:

![Compute Node CPU capacity planning](data:image/png;base64;base64,)

In order to get the best performance, CPUs allocated to Virtual Machines and to vRouter have to be isolated from those that are kept to Linux Operating system. CPU isolation is the first setup to be done to define the CPUs that will no more be used by the Linux Operating system. Those CPUs will be dedicated to DPDK vRouter or used by OpenStack Nova to spawn Virtual Machines.

Below is the CPU core topology of a 2 sockets system with 2\*12 physical cores, with hyper-threading enabled:

NUMA node0 CPU(s):

PHY cores: **0 2 4 6 8 10 12 14 16 18 20 22**

HT cores : **24 26 28 30 32 34 36 38 40 42 44 46**

NUMA node1 CPU(s):

PHY cores: **1 3 5 7 9 11 13 15 17 19 21 23**

HT cores : **25 27 29 31 33 35 37 39 41 43 45 47**

This topology will be used in the configuration examples provided in next sections.

### CPU kept for Linux Operating System

By default, all CPUs are included in the group of CPUs available for the Operating System needs. These CPUs are told: *“isolated”* because they are no more used to process all tasks. In order to get a CPU isolated several mechanisms can be used:

* remove this CPU from the “common” CPU list used to process all tasks
* change the scheduling algorithm (cooperative to preemptive)
* participate or not to interrupt processing

It is possible to remove some CPUs using ***isolcpus*** kernel parameter. This kernel parameter has to be provisioned at the system startup. GRUB configuration is updated to define *isolcpus* parameter and the system restarted.

In the example below we are keeping only CPU 0,1,24 and 25 for the Linux Operating System excluding them from isolcpus list. Strongly recommend is use minimum first cores on each NUMA including their siblings.

$ vi /etc/default/grub

GRUB\_CMDLINE\_LINUX="console=tty0 console=ttyS0,115200n8 crashkernel=auto rhgb quiet default\_hugepagesz=1GB hugepagesz=1G hugepages=28 iommu=pt intel\_iommu=on isolcpus=**2-23,26-47**"

$ grub2-mkconfig -o /etc/grub2.cfg

We also need to specify the CPUs that have to be used by the Linux Operating system in Systemd configuration file:

$ vi /systemd/system.conf

CPUAffinity=**0-1,24-25**

$ sudo systemctl deamon-reexec

$ sudo systemctl system.slice restart

When a RedHat Linux Operating System is used, it’s also recommended to configure tuned to get a stronger CPU isolation.

$ vi /etc/tuned/cpu-partitioning-variables.conf

**isolated\_cores=2-23,26-47**

$ sudo systemctl restart tuned

This is important to keep enough CPUs for the Linux Operating system. Not isolated CPUs are used by all tasks started and managed by the Linux Operating system, they are:

* System configuration and control tasks
* Contrail vRouter agent (SDN control plane)
* Hypervisor configuration and control tasks (Virtual Machine configuration for instance)

### CPU allocated to the DPDK vRouter

#### Packet polling and processing threads

DPDK vRouter speed is depending on the number of CPUs allocated for packet polling and processing. There is a trade-off to be found by each customer on how many CPU he will use for its own applications running on Virtual Machines and how many CPUs he will book for the vRouter to increase network packets processing speed.

We have first to define how many CPUs will be booked for DPDK vRouter polling and packet processing threads. We can first consider that each allocated CPU will bring up to 3 MPPS per core packets network processing speed to the vrouter. This 3 MPPS value is depending on lots of factors, mainly CPU speed, number of CPUs, NUMA usage, packet size, vRouter mode (packet or flow mode). It can range between 1 MPPS to 3 MPPS per core.

A Kernel mode vRouter is generally providing 1 MPPS per core packet speed in the best case, more generally Kernel mode vRouter is providing less than 1 MPPS.

With DPDK vRouter we are usually allocating from 4 to 8 network packets processing CPU (physical cores with their siblings).

A higher CPU number (more than 8) is not bringing much more performance due to some side effects of inter core communication or the multi queue setup it would require on Virtual Machines (cf: multi queue section).

CPU allocated to packet polling and processing are defined into CPU\_LIST parameter. This CPU parameter can use two different syntaxes: mask or list.

Here 4 physical CPUs (8 logical including second thread/siblings) are allocated to the vRouter for packet polling and processing:

$ vi /etc/sysconfig/network-scripts/ifcfg-vhost0

**CPU\_LIST**=2,4,6,8,26,28,30,32

PS: mask for CPUs 2,4,6,8,26,28,30,32 maps to binary value:  
b0000 0000 0000 0001 0101 0100 0000 0000 0000 0001 0101 0100 (0x154000154h).

#### vRouter DPDK dataplane configuration and control threads

Two DPDK vRouter parameters are allowing to define CPUs to be allocated for dataplane control and configuration threads:

* ***DPDK\_CTRL\_THREAD\_MASK:*** defines which CPUs will be allocated for DPDK initialization setup.
* ***SERVICE\_CORE\_MASK:*** defines which CPUs will be allocated for vRouter dataplane setup (vRouter interface setup).

DPDK initialization setup is done only at vRouter startup while vRouter dataplane setup task are done at vRouter initialization and each time a new interface is plugged or remove onto the vRouter. Same CPUs can be shared for these two tasks.

Here we are allocating CPU 10 and 34 dataplane control and configuration threads:

$ vi /etc/sysconfig/network-scripts/ifcfg-vhost0

**DPDK\_CTRL\_THREAD\_MASK**=0x400000400

**SERVICE\_CORE\_MASK**=0x400000400

PS: mask for CPUs 10,34 allocated maps to binary value:  
b0000 0000 0000 0100 0000 0000 0000 0000 0000 0100 0000 0000 (0x400000400h).

### CPU allocated to Virtual Machines

Host compute CPUs used for user Virtual Machines are defined into Nova configuration file. Below, we are enforcing Nova CPU assignment:

$ openstack-config --set /etc/nova/nova.conf DEFAULT vcpu\_pin\_set 3,5,7,9,11-23,27,29,31,33,35-47

$ cat /etc/nova/nova.conf | grep vcpu\_pin\_set

vcpu\_pin\_set=3,5,7,9,11-23,27,29,31,33,35-47

In order to get these changes taken into consideration, Nova compute service has to be restarted:

$ sudo docker restart nova\_compute

## vRouter memory setup

DPDK library need to get hugepages allocated by the Linux Operating system for vNIC rings setup. This is why, Contrail DPDK vRouterdataplane is requiring:

* Hugepage memory setup on the compute node
* Some of these huge pages to be allocated for the vRouter Physical NIC
* Some of these huge pages to be allocated for the Virtual Machine NICs
* Allocated hugepages to be visible from both DPDK vRouter application and Virtual Machines.

### Hugepage memory configuration on the compute node operating system

Only small hugepages (2MB) can be configured dynamically configured using sysctl. Bigger hugepages (1GB) must be configured at the system startup.

Following parameters are used:

* **default\_hugepagesz** defines which huge page size is a default (this size will appear in /proc/meminfo, and this size will be mounted by default when pagesize mounting option will not be used)
* **hugepagesz** followed by **hugepages** defines size and amount respectively and the pair can be repeated to configure different sizes of huge pages.

For instance, in order to configure 40 1GB hugepages and 40 2M hugepages at Linux System startup, we are proceeding like described hereafter and we are restarting the system:

$ vi /etc/default/grub

default\_hugepagesz=1GB hugepagesz=1G hugepages=40 hugepagesz=2M hugepages=40

$ grub2-mkconfig -o /etc/grub2.cfg

### Hugepage allocation for the DPDK vrouter

Some of the available operating system hugepages have to be allocated to the vRouterDPDK application to be used to create DPDK rings for the physical NIC.

In order the vRouter DPDK application to be able to use Linux System available hugepages, a hugetlbfs pseudo filesystem needs to be mounted. The following line needs to be added to /etc/fstab.

$ vi /etc/fstab

hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,seclabel,pagesize=**1G**)

DPDK vRouter detects the huge pages size hugetlbfs mount point. Here, the DPDK vRouter will try to use 1GB huge pages. If no page size is specified, the DPDK vRouter is assuming 2 MB hugepages have to be used. If no available hugepages of the specified (or 2MB if not size is specified) are available, the contrail DPDK vRouter will fail to start.

Amount of huge page memory requested by the vRouter at startup for its physical NIC DPDK rings setup is specified in ***socket-mem*** parameter.

In order the vRouter to request hugepages memory on the first NUMA socket only, we are using this option with only one parameter:

--socket-mem <value>

In order the vRouter to request hugepages memory on both NUMA0 and NUMA1 socket, we are using this option with only two parameters:

--socket-mem <value>,<value>

It is important to allocate hugepage memory to all NUMA nodes that will have DPDK interfaces associated with them. If memory is not allocated on a NUMA node associated with a physical NIC or VM, they cannot be used. If you are using 2 or more ports from different NICs, it is best to ensure that these NICs are on the same CPU socket.

Here we are configuring the vRouter to request 1GB hugepages memory on both NUMA nodes:

$ vi /etc/sysconfig/network-scripts/ifcfg-vhost0

DPDK\_COMMAND\_ADDITIONAL\_ARGS="--socket-mem 1024,1024"

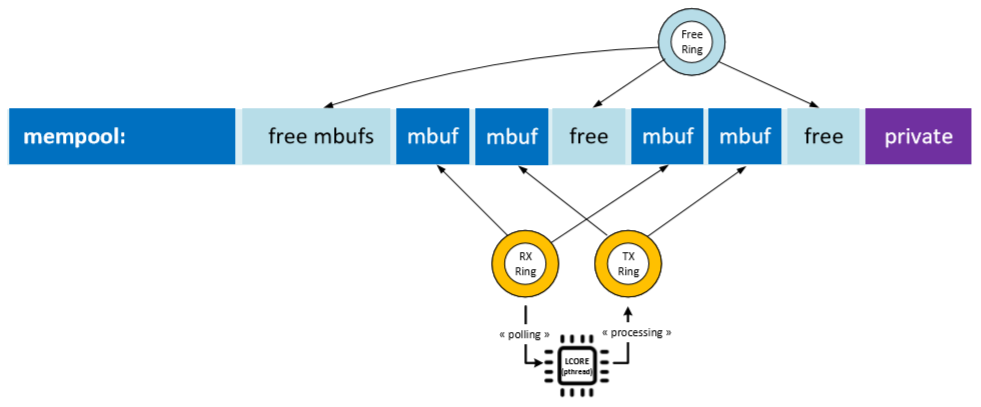
$ sudo ifdown vhost0

$ sudo ifup vhost0

### DPDK physical interface rings setup

In the previous section we’ve described how some hugepages memory is allocated to Contrail DPDK vRouter. This memory is mainly used by the DPDK vRouter to create DPDK rings for the physical interface.

Contrail DPDK vRouter will create 2 DPDK rings for each polling core (which are defined into CPU\_LIST parameter). DPDK rings are circular arrays of RX and TX descriptors that a pointing mbufs in which the packet content is stored. All mbufs for each TX/RX pair are stored into a single mempool memory area, they are representing an interface queue.



Following parameters are used for DPDK vRouter physical NIC configuration:

**--vr\_mempool\_sz** : is used to define mempool memory size. Default value is 16384.

**--dpdk\_txd\_sz** : is used to define Physical NIC TX Ring descriptor size. Default value is 256.

**--dpdk\_rxd\_sz** : is used to define Physical NIC RX Ring descriptor size. Default value is 256.

Here we are configuring the vRouter physical NIC DPDK rings with 512 RX and TX descriptors size and use 32MB mempool:

$ vi /etc/sysconfig/network-scripts/ifcfg-vhost0

DPDK\_COMMAND\_ADDITIONAL\_ARGS="--dpdk\_rxd\_sz 512 --dpdk\_txd\_sz 512 --vr\_mempool\_sz 32768"

$ sudo ifdown vhost0

$ sudo ifup vhost0

PS: Physical NIC DPDK ring size modification can lead to some unexpected side effect (packet loss). Needed mempool size depend on the configured maximum packet size (physical NIC MTU) and also the number of NIC using into the physical bond, and also the configured number of RX and TX descriptors.

### DPDK vRouter internal queues rings setup

In some scenarios, Contrail DPDK vRouter is using a DPDK pipeline model in order to split packet polling and processing task in two different threads. When this DPDK pipeline mode is used, some internal queues are created in order to store packets that have been polled by the polling lcore thread before to be processed by the processing lcore thread.



Two parameters are used for DPDK vRouterinternal queue (software rings) configuration:

**--vr\_dpdk\_tx\_ring\_sz**: is used to define forwarding lcores TX Ring descriptor size (1024 by default)

**--vr\_dpdk\_rx\_ring\_sz**: is used to define forwarding lcores RX Ring descriptor size (1024 by default).

Here we are configuring the vRouterinternal rings with 2048 RX and TX descriptors:

$ vi /etc/sysconfig/network-scripts/ifcfg-vhost0

DPDK\_COMMAND\_ADDITIONAL\_ARGS="--vr\_dpdk\_rx\_ring\_sz 2048 --vr\_dpdk\_tx\_ring\_sz 2048"

$ sudo ifdown vhost0

$ sudo ifup vhost0

### DPDK Virtual Machine interface rings setup

Virtual Machines NIC queues are not configured by Contrail vRouter, they are managed by OpenStack. By default, Nova is configuring 256 rx and tx descriptor size virtIO interfaces on the Virtual Machine.

Virtual Machines NIC queue size is defined at OpenStack level in */etc/nova/nova.conf* configuration file. They are configured using rx\_queue\_size and tx\_queue\_size parameters.

$ cat /etc/nova/nova.conf | grep x\_queue\_size

rx\_queue\_size=512

tx\_queue\_size=512

In order to get these changes taken into consideration, Nova compute service has to be restarted.

$ sudo docker restart nova\_compute

Virtual Machine NIC and vRouter vif on which each interface is connected to are sharing the same queues (DPDK rings):

* a vRouter vif tx ring is the same as the virtual NIC rx ring it is connected to.
* a vRouter vif rx ring is the same as the virtual NIC tx ring it is connected to.

It avoids duplicating the same information and to add processing overhead (that would be generated to manage data copy between vRouter vif and the Virtual Machine queues).

This is why Virtual Machine NIC queues have to be accessible from both vRouter and the Virtual Machine it belongs to. Virtual Machines have to be created by the QEMU/KVM hypervisor with a specific property which is allowing them to access the host Operating system hugepages and to request hugepages allocations.

Hugepages size to be allocated by the Hypervisor to the Virtual Machine has to be specified with *hw:mem\_page\_size* property. The configured hugepages memory size must be the same as those used by the DPDK vRouter (defined into huge pages size hugetlbfs mount point).

Here we are configuring an OpenStack flavor named m1.large which define 1GB size hugepages in *hw:mem\_page\_size* property:

$ openstack flavor set m1.large --property hw:mem\_page\_size=1GB

Then this flavor is used at the instance creation:

$ openstack server create --flavor FLAVOR\_ID --image IMAGE\_ID INSTANCE\_NAME

PS: *hw:mem\_page\_size* property can also be defined at image level

$ openstack image set --property hw:mem\_page\_size=1GB < IMAGE\_ID>

## Virtual Machine vif multi-queue setup

As explained earlier, when supported, it is suitable to enable multiqueues on Virtual Machines NIC. The most suitable scenario is to configure the same number of queues on virtual NIC than the number of polling cores defined on Contrail vRouter.

So, if contrail DPDK vRouter is configured with 4 queues the best scenario is to configure 4 queues on Virtual Machines network inferfaces.

In OpenStack, in order to get Virtual Machines configured with multiqueues, we have to enable multiqueue support on virtual instance image. It can be done with following command:

$ openstack image set --property hw\_vif\_multiqueue\_enabled="true" < IMAGE\_ID>

Then this image is used at the instance creation:

$ openstack server create --flavor <FLAVOR\_ID> --image <IMAGE\_ID> <INSTANCE\_NAME>

Then, we an instance is started with multiqueue vif property enabled, each interface is automatically configured with several queues. The number of queues to be enabled on each interface is automatically defined by Nova.

If the compute host (hypervisor node running qemu/kvm) is running Linux Kernel 3.X, the number of queues configured on the Virtual Machine interface is the same as the number of virtual CPUs configured on the Virtual Machine but can’t exceed 8 queues. It means for a Virtual Machine configured with 10 vCPUs, all its virtual network interface cards will be configured with 8 queues when multi queue is enabled.

If the compute host (hypervisor node running qemu/kvm) is running Linux Kernel 4.X, the number of queues configured on the Virtual Machine interface is the same as the number of virtual CPUs configured on the Virtual Machine but can’t exceed 256 queues. It means for a Virtual Machine configured with 10 vCPUs, all its virtual network interface cards will be configured with 10 queues when multi queue is enabled.

As explain earlier, Contrail vRouter is not able to process packets generated by connected virtual network interface cards configured with more queues than the number of CPU defined into its CPU\_LIST (number of polling and processing cores defined on Contrail vRouter).

Consequently, a Contrail vRouter configured with only 4 polling and processing cores won’t be able to collect a Virtual Machine configured with 10 vCPU with vif multiqueue property enable is connected to.

One of the following changes has to be performed:

* disable multiqueue on the Virtual Machine
* add more polling and processing cores on the vRouter (increase to 8 cores instead of only 4). In latest libvirt can be increased to 16.
* decrease the number of queues configured by Nova on the Virtual Machine

Unfortunately, no mechanism is provided by Nova to specify a specific value for the number of queues to be enabled on Virtual Machine network interface. Only the described algorithm is proposed.

In order to decrease the number of queues on the Virtual Machine network interfaces we have to run ethtool command inside this Virtual Machine. For instance, below we are configuring 4 logical queues on eth0 vNIC :

$ sudo ethtool -L eth0 combined 4

Virtual Machine initialization script has to be modified to automatically decrease the default value defined by Nova for the number of queues configured on its network interfaces to a lower one.

This is why, the most efficient setup today is to use Linux Kernel 3.X on OpenStack compute node running QEMU/KVM and to configure 8 CPU into the CPU\_LIST of the Contrail DPDK vRouter.

## vRouter routing and switching object tables dimensioning parameters

Some parameters supported as well as DPDK vRouter than Kernel one are allowing to define the size of internal objects tables. They are:

* **--vr\_flow\_entries**: maximum flow entries (default is 512K)
* **--vr\_oflow\_entries**: maximum overflow entries (default is 8K)
* **--vr\_bridge\_entries**: maximum bridge entries (default is 256K)
* **--vr\_bridge\_oentries**: maximum bridge overflow entries (default is 0)
* **--vr\_mpls\_labels**: maximum MPLS labels used in the node (default is 5K)
* **--vr\_nexthops**: maximum next hops in the node (defaut is 512K)
* **--vr\_vrfs**: maximum VRFs supported in the node (default is 4096)
* **--vr\_interfaces**: maximum interfaces that can be created (default is 4352)

In order to override their default values, we can configured an updated value using DPDK\_COMMAND\_ADDITIONAL\_ARGS parameter defined in vhost0 DPDK vRouter configuration file.

For instance; we can decrease the nexthops table size to 32K instead of 512K configured by default:

$ vi /etc/sysconfig/network-scripts/ifcfg-vhost0

DPDK\_COMMAND\_ADDITIONAL\_ARGS=”--vr\_nexthops=32768”

$ sudo ifdown vhost0

$ sudo ifup vhost0

All these parameters could increase vRouter performances but could also have a bad impact when not properly configured.

**vRouters DPDK fine tuning  parameters**

**--dpdk\_ctrl\_thread\_mask** : *(20.03 and later version)* CPUs to be used for vrouter control threads (CPU list or hexadecimal bitmask).

**--service\_core\_mask** : *(20.03 and later version)* CPUs to be used for vrouter service threads (CPU list or hexadecimal bitmask).

**--yield\_option** : *(20.03 and later version)* is used to enable or disable yield on forwarding cores (0 or 1 - enabled by default). Yield is an action that occurs in a computer program during multithreading, of forcing a processor (core) to relinquish control of the current running thread (vrouter polling and processing tasks), and sending it to the end of the running queue, of the same scheduling priority.As only one single thread is pinned onto vrouter allocated CPUs listed in CPU\_LIST, yield is useless (if the CPU isolation has properly be enforced).In the case below, yield is disabled onto forwarding cores.  
--yield\_option 0

**--vr\_no\_load\_balance** : *(20.08 and later version)* is used to disable packets processing pipeline model (internal load-balancing in which the processing and forwarding core is different from the polling one). When this parameter is present the internal load-balancing is disabled. When this parameter is absent, the internal load-balancing is enabled (default setup). In the case below, the internal load balancing is disabled:  
--vr\_no\_load\_balance

**--vr\_uncond\_close\_flow\_on\_tcp\_rst** : *(20.08 and later version)* is used to enable/disable unconditional closure of Flow on TCP RST (0 or 1 - disabled by default). In the case below, the unconditional closure of Flow on TCP RST is enabled:  
--vr\_uncond\_close\_flow\_on\_tcp\_rst 1

**---no-gro** : *(troubleshooting purpose)* is used to disable GRO (Generic Receive Offload) on DPDK vRouter dataplane. In the case below, GRO is disabled:  
--no-gro

**---no-gso** : *(troubleshooting purpose)* is used to disable GSO (Generic Segmention Offload) on DPDK vRouter dataplane. In the case below, GSO is disabled:  
--no-gso

**---no-mrgbuf** : *(troubleshooting purpose)* is used to turn off mergeable buffers on DPDK vRouter dataplane. In the case below, mergeable buffers are disabled:  
--no-mrgbuf

**--vr\_dpdk\_tx\_ring\_sz** : *(20.03 and later version)* is used to define forwarding lcores TX Ring descriptor size (1024 by default). In the case below, TX Ring descriptor size has been set to 2048.  
--vr\_dpdk\_tx\_ring\_sz 2048

**--vr\_dpdk\_rx\_ring\_sz** : *(20.03 and later version)* is used to define forwarding lcores RX Ring descriptor size (1024 by default). In the case below, RX Ring descriptor size has been set to 2048.  
--vr\_dpdk\_rx\_ring\_sz 2048

**--socket-mem** : is used to define the amount of memory pre-allocated for contrail vrouter. In the case below, 1GB of huge-page memory is pre-allocated on NUMA node 0 and NUMA node 1.  
--socket-mem 1024,1024

**--vr\_mempool\_sz** : is used to define mempool memory size. In the case below 128 MB mempool memory size is defined.

--vr\_mempool\_sz 131072

**--dpdk\_txd\_sz** : is used to define Physical NIC TX Ring descriptor size. In the case below 2048 bytes RX ring descriptor size is defined.

--dpdk\_txd\_sz 2048

**--dpdk\_rxd\_sz** : is used to define Physical NIC RX Ring descriptor size. In the case below 2048 bytes RX ring descriptor size is defined.

--dpdk\_rxd\_sz 2048

These values (especially --vr\_mempool\_sz, --dpdk\_txd\_sz and --dpdk\_rxd\_sz) have to be adjusted depending on :

* the inter NIC model used
* the number of NIC members of vhost0 bond
* the number of logical cores allocated to the vrouter

but also (especially **--vr\_no\_load\_balance**) depending on:

* multiqueue usage or not on virtual machine network interfaces
* encapsulation protocol (MPLSoGRE is not supported)
* once –vr\_no\_load\_balance there is no need to configure **--vr\_dpdk\_tx\_ring\_sz and --vr\_dpdk\_rx\_ring\_sz** as not used

A close up of a map

Description automatically generated