Telco features configuration

This section documents and explains the configuration of Telco-specific features on clusters deployed via SUSE Edge for Telco.

The directed network provisioning deployment method is used, as described in the Automated Provisioning section.

The following topics are covered in this section:

- Kernel image for real time: Kernel image to be used by the real-time kernel.
- Kernel arguments for low latency and high performance: Kernel arguments to be used by the real-time kernel for maximum performance and low latency running telco workloads.
- CPU Pinning via Tuned and kernel args: Isolating the CPUs via kernel arguments and Tuned profile.
- CNI configuration: CNI configuration to be used by the Kubernetes cluster.
- SR-IOV configuration: SR-IOV configuration to be used by the Kubernetes workloads.
- DPDK configuration: DPDK configuration to be used by the system.
- vRAN acceleration card: Acceleration card configuration to be used by the Kubernetes workloads.
- Huge pages: Huge pages configuration to be used by the Kubernetes workloads.
- CPU pinning on Kubernetes: Configuring Kubernetes and the applications to leverage CPU pinning.
- NUMA-aware scheduling configuration: NUMA-aware scheduling configuration to be used by the Kubernetes workloads.
- Metal LB configuration: Metal LB configuration to be used by the Kubernetes workloads.
- Private registry configuration: Private registry configuration to be used by the Kubernetes workloads.
- Precision Time Protocol configuration: Configuration files for running PTP telco profiles.

Kernel image for real time

The real-time kernel image is not necessarily better than a standard kernel. It is a different kernel tuned to a specific use case. The real-time kernel is tuned for lower latency at the cost of throughput. The real-time kernel is not recommended for general purpose use, but in our case, this is the recommended kernel for Telco Workloads where latency is a key factor.

There are four top features:

- Deterministic execution:

  Get greater predictability — ensure critical business processes complete in time, every time and deliver high-quality service, even under heavy system loads. By shielding key system resources for high-priority processes, you can ensure greater predictability for time-sensitive applications.

- Low jitter:

  The low jitter built upon the highly deterministic technology helps to keep applications synchronized with the real world. This helps services that need ongoing and repeated calculation.

- Priority inheritance:

  Priority inheritance refers to the ability of a lower priority process to assume a higher priority when there is a higher priority process that requires the lower priority process to finish before it can accomplish its task. SUSE Linux Enterprise Real Time solves these priority inversion problems for mission-critical processes.

- Thread interrupts:

  Processes running in interrupt mode in a general-purpose operating system are not preemptible. With SUSE Linux Enterprise Real Time, these interrupts have been encapsulated by kernel threads, which are interruptible, and allow the hard and soft interrupts to be preempted by user-defined higher priority processes.

  In our case, if you have installed a real-time image like `SUSE Linux Micro RT`, kernel real time is already installed. From the SUSE Customer Center, you can download the real-time kernel image.

  NOTE For more information about the real-time kernel, visit SUSE Real Time.

Kernel arguments for low latency and high performance

The kernel arguments are important to be configured to enable the real-time kernel to work properly giving the best performance and low latency to run telco workloads. There are some important concepts to keep in mind when configuring the kernel arguments for this use case:

- Remove `kthread_cpus` when using SUSE real-time kernel. This parameter controls on which CPUs kernel threads are created. It also controls which CPUs are allowed for PID 1 and for loading kernel modules (the kmod user-space helper). This parameter is not recognized and does not have any effect.
- Isolate the CPU cores using `isolcpus`, `nohz_full`, `rcu_nocbs`, and `irqaffinity`. For a comprehensive list of CPU pinning techniques, refer to CPU Pinning via Tuned and kernel args chapter.
- Add `domain,nohz,managed_irq` flags to `isolcpus` kernel argument. Without any flags, `isolcpus` is equivalent to specifying only the `domain` flag. This isolates the specified CPUs from scheduling, including kernel tasks. The `nohz` flag stops the scheduler tick on the specified CPUs (if only one task is runnable on a CPU), and the `managed_irq` flag avoids routing managed external (device) interrupts at the specified CPUs. Note that the IRQ lines of NVMe devices are fully managed by the kernel and will be routed to the non-isolated (housekeeping) cores as a consequence. For example, the command line provided at the end of this section will result in only four queues (plus an admin/control queue) allocated on the system:

```
for I in $(grep nvme0 /proc/interrupts | cut -d ':' -f1); do cat
/proc/irq/${I}/effective_affinity_list; done | column
39      0       19      20      39
```

This behavior prevents any disruption caused by disk I/O to any time sensitive application running on the isolated cores, but might require attention and careful design for storage focused workloads.

- Tune the ticks (kernel's periodic timer interrupts):
  - `skew_tick=1`: ticks can sometimes happen simultaenously. Instead of all CPUs receiving their timer tick at the exact same moment, `skew_tick=1` makes them occur at slightly offset times. This helps reduce system jitter, resulting in more consistent and lower interrupt response times (an essential requirement for latency-sensitive applications).
  - `nohz=on`: stops the periodic timer tick on idle CPUs.
  - `nohz_full=<cpu-cores>`: Stops the period timer tick on specified CPUs that are dedicated for real-time applications.
- Disable Machine Check Exception (MCE) handling by specifying `mce=off`. MCEs are hardware errors detected by the processor and disabling them can avoid noisy logs.
- Add `nowatchdog` to disable the soft-lockup watchdog which is implemented as a timer running in the timer hard-interrupt context. When it expires (i.e. a soft lockup is detected), it will print a warning (in the hard interrupt context), running any latency targets. Even if it never expires, it goes onto the timer list, slightly increasing the overhead of every timer interrupt. This option also disables the NMI watchdog, so NMIs cannot interfere.
- `nmi_watchdog=0` disables the NMI (Non-Maskable Interrupt) watchdog. This can be omitted when `nowatchdog` is used.
- RCU (Read-Copy-Update) is a kernel mechanism that enables concurrent, lock-free access for many readers to shared data. An RCU callback, a function triggered after a 'grace period', ensures all previous readers have finished so old data can be safely reclaimed. We fine-tune RCU, particularly for sensitive workloads, to offload these callbacks from dedicated (pinned) CPUs, preventing kernel operations from interfering with critical, time-sensitive tasks.
  - Specify the pinned CPUs in `rcu_nocbs` so that RCU callbacks do not run on them. This helps reducing jitter and latency for the real-time workloads.
  - `rcu_nocb_poll` makes the no-callback CPUs regularly 'poll' to see if callback handling is required. This can reduce the interrupt overhead.
  - `rcupdate.rcu_cpu_stall_suppress=1` suppresses RCU CPU stall warnings, which can sometimes be false positives in heavily loaded real-time systems
  - `rcupdate.rcu_expedited=1` speeds up the grace period for RCU operations, making read-side critical sections more responsive
  - `rcupdate.rcu_normal_after_boot=1` When used with rcu_expedited, it allows RCU to rever to normal (non-expedited) operation after the system boot.
  - `rcupdate.rcu_task_stall_timeout=0` disables the RCU task stall detector, preventing potential warnings or system halts from long-running RCU tasks.

    - `rcutree.kthread_prio=99` sets the priority of the RCU callback kernel thread to the highest possible (99), ensuring it gets scheduled and handles RCU callbacks promptly, when needed.

- Add `ignition.platform.id=openstack` for Metal3 and Cluster API to successfully provision/deprovision the cluster. This is used by Metal3 Python agent, which originated from Openstack Ironic.
- Remove `intel_pstate=passive`. This option configures `intel_pstate` to work with generic cpufreq governors, but to make this work, it disables hardware-managed P-states (`HWP`) as a side effect. To reduce the hardware latency, this option is not recommended for real-time workloads.
- Replace `intel_idle.max_cstate=0 processor.max_cstate=1` with `idle=poll`. To avoid C-State transitions, the `idle=poll` option is used to disable the C-State transitions and keep the CPU in the highest C-State. The `intel_idle.max_cstate=0` option disables `intel_idle`, so `acpi_idle` is used, and `acpi_idle.max_cstate=1` then sets max C-state for acpi_idle. On {x86-64} architectures, the first ACPI C-State is always `POLL`, but it uses a `poll_idle()` function, which may introduce some tiny latency by reading the clock periodically, and restarting the main loop in `do_idle()` after a timeout (this also involves clearing and setting the `TIF_POLL` task flag). In contrast, `idle=poll` runs in a tight loop, busy-waiting for a task to be rescheduled. This minimizes the latency of exiting the idle state, but at the cost of keeping the CPU running at full speed in the idle thread.
- Disable C1E in BIOS. This option is important to disable the C1E state in the BIOS to avoid the CPU from entering the C1E state when idle. The C1E state is a low-power state that can introduce latency when the CPU is idle.

The rest of this documentation covers additional parameters, including huge pages and IOMMU.

This provides an example of kernel arguments for a 32-core Intel server, including the aforementioned adjustments:

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt root=UUID=77b713de-5cc7-4d4c-8fc6-f5eca0a43cf9
skew_tick=1 rd.timeout=60 rd.retry=45 console=ttyS1,115200 console=tty0
default_hugepagesz=1G hugepagesz=1G hugepages=40 hugepagesz=2M hugepages=0
ignition.platform.id=openstack intel_iommu=on iommu=pt irqaffinity=0,31,32,63
isolcpus=domain,nohz,managed_irq,1-30,33-62 nohz_full=1-30,33-62 nohz=on mce=off
net.ifnames=0 nosoftlockup nowatchdog nmi_watchdog=0 quiet rcu_nocb_poll rcu_nocbs=1-
30,33-62 rcupdate.rcu_cpu_stall_suppress=1 rcupdate.rcu_expedited=1
rcupdate.rcu_normal_after_boot=1 rcupdate.rcu_task_stall_timeout=0
rcutree.kthread_prio=99 security=selinux selinux=1 idle=poll
```

Here is another configuration example for a 64-core AMD server. Among the 128 logical processors (`0-127`), first 8 cores (`0-7`) are designated for housekeeping, while the remaining 120 cores (`8-127`) are pinned for the applications:

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt root=UUID=575291cf-74e8-42cf-8f2c-408a20dc00b8
skew_tick=1 console=ttyS1,115200 console=tty0 default_hugepagesz=1G hugepagesz=1G
hugepages=40 hugepagesz=2M hugepages=0 ignition.platform.id=openstack amd_iommu=on
iommu=pt irqaffinity=0-7 isolcpus=domain,nohz,managed_irq,8-127 nohz_full=8-127
```

```
rcu_nocbs=8-127 mce=off nohz=on net.ifnames=0 nowatchdog nmi_watchdog=0 nosoftlockup
quiet rcu_nocb_poll rcupdate.rcu_cpu_stall_suppress=1 rcupdate.rcu_expedited=1
rcupdate.rcu_normal_after_boot=1 rcupdate.rcu_task_stall_timeout=0
rcutree.kthread_prio=99 security=selinux selinux=1 idle=poll
```

CPU pinning via Tuned and kernel args

`tuned` is a system tuning tool that monitors system conditions to optimize performance using various predefined profiles. A key feature is its ability to isolate CPU cores for specific workloads, like real-time applications. This prevents the OS from utilizing these cores and potentially increasing latency.

To enable and configure this feature, the first thing is to create a profile for the CPU cores we want to isolate. In this example, among 64 cores, we dedicate 60 cores (`1-30,33-62`) for the application and remaining 4 cores are used for housekeeping. Note that the design of isolated CPUs heavily depends on the real-time applications.

```
$ echo "export tuned_params" >> /etc/grub.d/00_tuned

$ echo "isolated_cores=1-30,33-62" >> /etc/tuned/cpu-partitioning-variables.conf

$ tuned-adm profile cpu-partitioning
Tuned (re)started, changes applied.
```

Then we need to modify the GRUB option to isolate CPU cores and other important parameters for CPU usage. The following options are important to be customized with your current hardware specifications:

parameter	value	description
isolcpus	domain,nohz,managed_irq,1-30,33-62	Isolate the cores 1-30 and 33-62. `domain` indicates these CPUs are part of isolation domain. `nohz` enables tickless operation on these isolated CPUs when they are idle, to reduce interruptions. `managed_irq` isolates pinned CPUs from being targeted by IRQs. This contemplates `irqaffinity=0-7`, which already directs mosts IRQs to the housekeeping cores.
skew_tick	1	This option allows the kernel to skew the timer interrupts across the isolated CPUs.

parameter	value	description
nohz	on	When enabled, kernel's periodic timer interrupt (the 'tick') will stop on any CPU core that is idle. This primary benefits the housekeeping CPUs (`0,31,32,63`). This conserves power and reduces unnecessary wake-ups on those general-purpose cores.
nohz_full	1-30,33-62	For the isolated cores, this stops the tick and it does so even when the CPU is running a single active task. It means it makes the CPU run in full tickless mode (or 'dyntick'). The kernel will only deliver timer interrupts when they are actually needed.
rcu_nocbs	1-30,33-62	This option offloads the RCU callback processing from specified CPU cores.
rcu_nocb_poll		When this option is set, no-RCU-callback CPUs will regularly 'poll' to see if callback handling is required, rather than being explicitly woken up by other CPUs. This can reduce the interrupt overhead.
irqaffinity	0,31,32,63	This option allows the kernel to run the interrupts to the housekeeping cores.
idle	poll	This minimizes the latency of exiting the idle state, but at the cost of keeping the CPU running at full speed in the idle thread.
nmi_watchdog	0	This option disables only the NMI watchdog. This can be omitted when `nowatchdog` is set.
nowatchdog		This option disables the soft-lockup watchdog which is implemented as a timer running in the timer hard-interrupt context.

The following commands modify the GRUB configuration and apply the changes mentioned above to be present on the next boot:

Edit the `/etc/default/grub` file with above parameters and the file will look like this:

```
GRUB_CMDLINE_LINUX="BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt root=UUID=77b713de-5cc7-4d4c-
8fc6-f5eca0a43cf9 skew_tick=1 rd.timeout=60 rd.retry=45 console=ttyS1,115200
console=tty0 default_hugepagesz=1G hugepagesz=1G hugepages=40 hugepagesz=2M
hugepages=0 ignition.platform.id=openstack intel_iommu=on iommu=pt
irqaffinity=0,31,32,63 isolcpus=domain,nohz,managed_irq,1-30,33-62 nohz_full=1-30,33-
62 nohz=on mce=off net.ifnames=0 nosoftlockup nowatchdog nmi_watchdog=0 quiet
rcu_nocb_poll rcu_nocbs=1-30,33-62 rcupdate.rcu_cpu_stall_suppress=1
rcupdate.rcu_expedited=1 rcupdate.rcu_normal_after_boot=1
rcupdate.rcu_task_stall_timeout=0 rcutree.kthread_prio=99 security=selinux selinux=1
idle=poll"
```

Update the GRUB configuration:

```
$ transactional-update grub.cfg
$ reboot
```

To validate that the parameters are applied after the reboot, the following command can be used to check the kernel command line:

```
$ cat /proc/cmdline
```

There is another script that can be used to tune the CPU configuration, which basically is doing the following steps:

- Set the CPU governor to `performance`.
- Unset the timer migration to the isolated CPUs.
- Migrate the kdaemon threads to the housekeeping CPUs.
- Set the isolated CPUs latency to the lowest possible value.
- Delay the vmstat updates to 300 seconds.

The script is available at SUSE Edge for Telco Examples repository.

CNI Configuration

Cilium

`Cilium` is the default CNI plug-in for SUSE Edge for Telco. To enable Cilium on RKE2 cluster as the default plug-in, the following configurations are required in the `/etc/rancher/rke2/config.yaml` file:

```
cni:
- cilium
```

This can also be specified with command-line arguments, that is, `--cni=cilium` into the server line in `/etc/systemd/system/rke2-server` file.

To use the `SR-IOV` network operator described in the next section, use `Multus` with another CNI plug-in, like `Cilium` or `Calico`, as a secondary plug-in.

```
cni:
- multus
- cilium
```

NOTE For more information about CNI plug-ins, visit Network Options.

Bond CNI

SUSE Edge is shipping the bond cni plugin. In general terms bonding provides a method for aggregating multiple network interfaces into a single logical "bonded" interface. This is typically used to increase service availability by introducing redundant networking paths, but can also be used to increase bandwidth with certain bond modes. We support the bond cni plugin with multus for:

- MACVLAN
- Host Device
- SR-IOV

Bond CNI with MACVLAN

To use the bond cni plugin with MACVLAN two free interfaces are needed in the container. Here we have picked enp8s0 and enp9s0. Start by creating network attachment definitions for them:

NetworkAttachmentDefinition enp8s0

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: enp8s0-conf
spec:
  config: '{
      "cniVersion": "0.3.1",
      "plugins": [
        {
          "type": "macvlan",
          "capabilities": { "ips": true },
          "master": "enp8s0",
```

```
          "mode": "bridge",
          "ipam": {}
        }, {
          "capabilities": { "mac": true },
          "type": "tuning"
        }
      ]
    }'
```

NetworkAttachmentDefinition enp9s0

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: enp9s0-conf
spec:
  config: '{
      "cniVersion": "0.3.1",
      "plugins": [
        {
          "type": "macvlan",
          "capabilities": { "ips": true },
          "master": "enp9s0",
          "mode": "bridge",
          "ipam": {}
        }, {
          "capabilities": { "mac": true },
          "type": "tuning"
        }
      ]
    }'
```

After this you need a network attachment definition for the bond itself

NetworkAttachmentDefinition bond

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond-net1
spec:
  config: '{
  "type": "bond",
  "cniVersion": "0.3.1",
  "name": "bond-net1",
  "mode": "active-backup",
  "failOverMac": 1,
  "linksInContainer": true,
  "miimon": "100",
```

```
  "mtu": 1500,
  "links": [
     {"name": "net1"},
     {"name": "net2"}
  ],
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "192.168.200.100/24",
        "gateway": "192.168.200.1"
      }
    ],
    "subnet": "192.168.200.0/24",
    "routes": [{
      "dst": "0.0.0.0/0"
    }]
  }
}'
```

The IP addressing picked here is static and defines the address of the bond as 192.168.200.100 on a 24 network, with a gateway residing on the network's first available address. In the bond's network attachment we also define the type of bond we want. In this case it is active-backup.

To use this bond the pod needs to know about all interfaces. And example pod definition might look like this:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  annotations:
        k8s.v1.cni.cncf.io/networks: '[
{"name": "enp8s0-conf",
"interface": "net1"
},
{"name": "enp9s0-conf",
"interface": "net2"
},
{"name": "bond-net1",
"interface": "bond0"
}]'
spec:
  restartPolicy: Never
  containers:
  - name: bond-test
    image: alpine:latest
    command:
      - /bin/sh
      - "-c"
```

```
      - "sleep 60m"
    imagePullPolicy: IfNotPresent
```

Note how the annotation refers to all networks and how it defines the mapping between the interfaces enp8s0 → net1, and enp9s0→net2.

Bond CNI with Host Device

To use the bond cni plugin with Host Device two free interfaces are needed in the host. These interfaces will be mapped through to the container. Here we have picked enp8s0 and enp9s0. Start by creating network attachment definitions for them:

NetworkAttachmentDefinition enp8s0

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: enp8s0-hostdev
spec:
  config: '{
      "cniVersion": "0.3.1",
      "plugins": [
        {
          "type": "host-device",
          "name": "host0",
          "device": "enp8s0",
          "ipam": {}
        }]
    }'
```

NetworkAttachmentDefinition enp9s0

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: enp9s0-hostdev
spec:
  config: '{
      "cniVersion": "0.3.1",
      "plugins": [
        {
          "type": "host-device",
          "name": "host0",
          "device": "enp9s0",
          "ipam": {}
        }]
    }'
```

After this you need a network attachment definition for the bond itself. This is exactly the same as for the MACVLAN case.

NetworkAttachmentDefinition bond

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond-net1
spec:
  config: '{
  "type": "bond",
  "cniVersion": "0.3.1",
  "name": "bond-net1",
  "mode": "active-backup",
  "failOverMac": 1,
  "linksInContainer": true,
  "miimon": "100",
  "mtu": 1500,
  "links": [
     {"name": "net1"},
     {"name": "net2"}
  ],
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "192.168.200.100/24",
        "gateway": "192.168.200.1"
      }
    ],
    "subnet": "192.168.200.0/24",
    "routes": [{
      "dst": "0.0.0.0/0"
    }]
  }
}'
```

The IP addressing picked here is static and defines the address of the bond as 192.168.200.100 on a 24 network, with a gateway residing on the network's first available address. In the bond's network attachment we also define the type of bond we want. In this case it is active-backup.

To use this bond the pod needs to know about all interfaces. And example pod definition for bond with host devices might look like this:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
```

```
  annotations:
        k8s.v1.cni.cncf.io/networks: '[
{"name": "enp8s0-hostdev",
"interface": "net1"
},
{"name": "enp9s0-hostdev",
"interface": "net2"
},
{"name": "bond-net1",
"interface": "bond0"
}]'
spec:
  restartPolicy: Never
  containers:
  - name: bond-test
    image: alpine:latest
    command:
      - /bin/sh
      - "-c"
      - "sleep 60m"
    imagePullPolicy: IfNotPresent
```

Bond CNI with SR-IOV

Using the bond cni with SR-IOV is fairly straight forward. For more details on how to set up SR-IOV look in the SR-IO section. As described there you will create SriovNetworkNodePolicies that defines resourceNames, as well as number of virtual functions and such. The resourceNames are being used by the SriovNetwork which is used as interfaces in the pod definition. The bond definition is exactly the same as for the MACVLAN and host device cases.

SR-IOV

SR-IOV allows a device, such as a network adapter, to separate access to its resources among various `PCIe` hardware functions. There are different ways to deploy `SR-IOV`, and here, we show two different options:

- Option 1: using the `SR-IOV` CNI device plug-ins and a config map to configure it properly.
- Option 2 (recommended): using the `SR-IOV` Helm chart from Rancher Prime to make this deployment easy.

Option 1 - Installation of SR-IOV CNI device plug-ins and a config map to configure it properly

- Prepare the config map for the device plug-in

Get the information to fill the config map from the `lspci` command:

```
$ lspci | grep -i acc
8a:00.0 Processing accelerators: Intel Corporation Device 0d5c
```

```
$ lspci | grep -i net
19:00.0 Ethernet controller: Broadcom Inc. and subsidiaries BCM57504 NetXtreme-E
10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet (rev 11)
19:00.1 Ethernet controller: Broadcom Inc. and subsidiaries BCM57504 NetXtreme-E
10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet (rev 11)
19:00.2 Ethernet controller: Broadcom Inc. and subsidiaries BCM57504 NetXtreme-E
10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet (rev 11)
19:00.3 Ethernet controller: Broadcom Inc. and subsidiaries BCM57504 NetXtreme-E
10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet (rev 11)
51:00.0 Ethernet controller: Intel Corporation Ethernet Controller E810-C for QSFP
(rev 02)
51:00.1 Ethernet controller: Intel Corporation Ethernet Controller E810-C for QSFP
(rev 02)
51:01.0 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
02)
51:01.1 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
02)
51:01.2 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
02)
51:01.3 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
02)
51:11.0 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
02)
51:11.1 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
02)
51:11.2 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
02)
51:11.3 Ethernet controller: Intel Corporation Ethernet Adaptive Virtual Function (rev
02)
```

The config map consists of a `JSON` file that describes devices using filters to discover, and creates groups for the interfaces. The key is understanding filters and groups. The filters are used to discover the devices and the groups are used to create the interfaces.

It could be possible to set filters:

- vendorID: `8086` (Intel)
- deviceID: `0d5c` (Accelerator card)
- driver: `vfio-pci` (driver)
- pfNames: `p2p1` (physical interface name)

It could be possible to also set filters to match more complex interface syntax, for example:

- pfNames: `["eth1#1,2,3,4,5,6"]` or `[eth1#1-6]` (physical interface name)

Related to the groups, we could create a group for the `FEC` card and another group for the `Intel` card, even creating a prefix depending on our use case:

- resourceName: `pci_sriov_net_bh_dpdk`

- resourcePrefix: `Rancher.io`

There are a lot of combinations to discover and create the resource group to allocate some `VFs` to the pods.

NOTE	For more information about the filters and groups, visit sr-iov network device plug-in.

After setting the filters and groups to match the interfaces depending on the hardware and the use case, the following config map shows an example to be used:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: sriovdp-config
  namespace: kube-system
data:
  config.json: |
    {
        "resourceList": [
            {
                "resourceName": "intel_fec_5g",
                "devicetype": "accelerator",
                "selectors": {
                    "vendors": ["8086"],
                    "devices": ["0d5d"]
                }
            },
            {
                "resourceName": "intel_sriov_odu",
                "selectors": {
                    "vendors": ["8086"],
                    "devices": ["1889"],
                    "drivers": ["vfio-pci"],
                    "pfNames": ["p2p1"]
                }
            },
            {
                "resourceName": "intel_sriov_oru",
                "selectors": {
                    "vendors": ["8086"],
                    "devices": ["1889"],
                    "drivers": ["vfio-pci"],
                    "pfNames": ["p2p2"]
                }
            }
        ]
    }
```

- Prepare the `daemonset` file to deploy the device plug-in.

The device plug-in supports several architectures (`arm`, `amd`, `ppc64le`), so the same file can be used for different architectures deploying several `daemonset` for each architecture.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sriov-device-plugin
  namespace: kube-system
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: kube-sriov-device-plugin-amd64
  namespace: kube-system
  labels:
    tier: node
    app: sriovdp
spec:
  selector:
    matchLabels:
      name: sriov-device-plugin
  template:
    metadata:
      labels:
        name: sriov-device-plugin
        tier: node
        app: sriovdp
    spec:
      hostNetwork: true
      nodeSelector:
        kubernetes.io/arch: amd64
      tolerations:
      - key: node-role.kubernetes.io/master
        operator: Exists
        effect: NoSchedule
      serviceAccountName: sriov-device-plugin
      containers:
      - name: kube-sriovdp
        image: rancher/hardened-sriov-network-device-plugin:v3.7.0-build20240816
        imagePullPolicy: IfNotPresent
        args:
        - --log-dir=sriovdp
        - --log-level=10
        securityContext:
          privileged: true
        resources:
          requests:
            cpu: "250m"
```

```
          memory: "40Mi"
        limits:
          cpu: 1
          memory: "200Mi"
      volumeMounts:
      - name: devicesock
        mountPath: /var/lib/kubelet/
        readOnly: false
      - name: log
        mountPath: /var/log
      - name: config-volume
        mountPath: /etc/pcidp
      - name: device-info
        mountPath: /var/run/k8s.cni.cncf.io/devinfo/dp
    volumes:
      - name: devicesock
        hostPath:
          path: /var/lib/kubelet/
      - name: log
        hostPath:
          path: /var/log
      - name: device-info
        hostPath:
          path: /var/run/k8s.cni.cncf.io/devinfo/dp
          type: DirectoryOrCreate
      - name: config-volume
        configMap:
          name: sriovdp-config
          items:
          - key: config.json
            path: config.json
```

- After applying the config map and the `daemonset`, the device plug-in will be deployed and the interfaces will be discovered and available for the pods.

```
$ kubectl get pods -n kube-system | grep sriov
kube-system  kube-sriov-device-plugin-amd64-twjfl  1/1  Running  0  2m
```

- Check the interfaces discovered and available in the nodes to be used by the pods:

```
$ kubectl get $(kubectl get nodes -oname) -o jsonpath='{.status.allocatable}' | jq
{
  "cpu": "64",
  "ephemeral-storage": "256196109726",
  "hugepages-1Gi": "40Gi",
  "hugepages-2Mi": "0",
  "intel.com/intel_fec_5g": "1",
  "intel.com/intel_sriov_odu": "4",
  "intel.com/intel_sriov_oru": "4",
```

```
  "memory": "221396384Ki",
  "pods": "110"
}
```

- The `FEC` is `intel.com/intel_fec_5g` and the value is 1.
- The `VF` is `intel.com/intel_sriov_odu` or `intel.com/intel_sriov_oru` if you deploy it with a device plug-in and the config map without Helm charts.

IMPORTANT	If there are no interfaces here, it makes little sense to continue because the interface will not be available for pods. Review the config map and filters to solve the issue first.

Option 2 (recommended) - Installation using Rancher using Helm chart for SR-IOV CNI and device plug-ins

- Get Helm if not present:

```
$ curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

- Install SR-IOV.

```
helm install sriov-crd oci://registry.suse.com/edge/charts/sriov-crd -n sriov-network-
operator
helm install sriov-network-operator oci://registry.suse.com/edge/charts/sriov-network-
operator -n sriov-network-operator
```

- Check the deployed resources crd and pods:

```
$ kubectl get crd
$ kubectl -n sriov-network-operator get pods
```

- Check the label in the nodes.

With all resources running, the label appears automatically in your node:

```
$ kubectl get nodes -oyaml | grep feature.node.kubernetes.io/network-sriov.capable

feature.node.kubernetes.io/network-sriov.capable: "true"
```

- Review the `daemonset` to see the new `sriov-network-config-daemon` and `sriov-rancher-nfd-worker` as active and ready:

```
$ kubectl get daemonset -A
NAMESPACE            NAME                          DESIRED   CURRENT   READY   UP-
```

```
TO-DATE   AVAILABLE   NODE SELECTOR                                          AGE
calico-system            calico-node                    1         1         1       1
1           kubernetes.io/os=linux                                  15h
sriov-network-operator   sriov-network-config-daemon    1         1         1       1
1           feature.node.kubernetes.io/network-sriov.capable=true   45m
sriov-network-operator   sriov-rancher-nfd-worker       1         1         1       1
1           <none>                                                  45m
kube-system              rke2-ingress-nginx-controller  1         1         1       1
1           kubernetes.io/os=linux                                  15h
kube-system              rke2-multus-ds                 1         1         1       1
1           kubernetes.io/arch=amd64,kubernetes.io/os=linux         15h
```

In a few minutes (can take up to 10 min to be updated), the nodes are detected and configured with the SR-IOV capabilities:

```
$ kubectl get sriovnetworknodestates.sriovnetwork.openshift.io -A
NAMESPACE              NAME     AGE
sriov-network-operator   xr11-2   83s
```

- Check the interfaces detected.

The interfaces discovered should be the PCI address of the network device. Check this information with the `lspci` command in the host.

```
$ kubectl get sriovnetworknodestates.sriovnetwork.openshift.io -n kube-system -oyaml
apiVersion: v1
items:
- apiVersion: sriovnetwork.openshift.io/v1
  kind: SriovNetworkNodeState
  metadata:
    creationTimestamp: "2023-06-07T09:52:37Z"
    generation: 1
    name: xr11-2
    namespace: sriov-network-operator
    ownerReferences:
    - apiVersion: sriovnetwork.openshift.io/v1
      blockOwnerDeletion: true
      controller: true
      kind: SriovNetworkNodePolicy
      name: default
      uid: 80b72499-e26b-4072-a75c-f9a6218ec357
    resourceVersion: "356603"
    uid: e1f1654b-92b3-44d9-9f87-2571792cc1ad
  spec:
    dpConfigVersion: "356507"
  status:
    interfaces:
    - deviceID: "1592"
      driver: ice
```

```
      eSwitchMode: legacy
      linkType: ETH
      mac: 40:a6:b7:9b:35:f0
      mtu: 1500
      name: p2p1
      pciAddress: "0000:51:00.0"
      totalvfs: 128
      vendor: "8086"
    - deviceID: "1592"
      driver: ice
      eSwitchMode: legacy
      linkType: ETH
      mac: 40:a6:b7:9b:35:f1
      mtu: 1500
      name: p2p2
      pciAddress: "0000:51:00.1"
      totalvfs: 128
      vendor: "8086"
    syncStatus: Succeeded
kind: List
metadata:
  resourceVersion: ""
```

NOTE

If your interface is not detected here, ensure that it is present in the next config map:

```
$ kubectl get cm supported-nic-ids -oyaml -n sriov-network-operator
```

If your device is not there, edit the config map, adding the right values to be discovered (should be necessary to restart the `sriov-network-config-daemon` daemonset).

- Create the `NetworkNode Policy` to configure the `VFs`.

Some `VFs` (`numVfs`) from the device (`rootDevices`) will be created, and it will be configured with the driver `deviceType` and the `MTU`:

NOTE

The `resourceName` field must not contain any special characters and must be unique across the cluster. The example uses the `deviceType: vfio-pci` because `dpdk` will be used in combination with `sr-iov`. If you don't use `dpdk`, the deviceType should be `deviceType: netdevice` (default value).

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-dpdk
  namespace: sriov-network-operator
```

```
spec:
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  resourceName: intelnicsDpdk
  deviceType: vfio-pci
  numVfs: 8
  mtu: 1500
  nicSelector:
    deviceID: "1592"
    vendor: "8086"
    rootDevices:
    - 0000:51:00.0
```

- Validate configurations:

```
$ kubectl get $(kubectl get nodes -oname) -o jsonpath='{.status.allocatable}' | jq
{
  "cpu": "64",
  "ephemeral-storage": "256196109726",
  "hugepages-1Gi": "60Gi",
  "hugepages-2Mi": "0",
  "intel.com/intel_fec_5g": "1",
  "memory": "200424836Ki",
  "pods": "110",
  "rancher.io/intelnicsDpdk": "8"
}
```

- Create the sr-iov network (optional, just in case a different network is needed):

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: network-dpdk
  namespace: sriov-network-operator
spec:
  ipam: |
    {
      "type": "host-local",
      "subnet": "192.168.0.0/24",
      "rangeStart": "192.168.0.20",
      "rangeEnd": "192.168.0.60",
      "routes": [{
        "dst": "0.0.0.0/0"
      }],
      "gateway": "192.168.0.1"
    }
  vlan: 500
  resourceName: intelnicsDpdk
```

- Check the network created:

```
$ kubectl get network-attachment-definitions.k8s.cni.cncf.io -A -oyaml

apiVersion: v1
items:
- apiVersion: k8s.cni.cncf.io/v1
  kind: NetworkAttachmentDefinition
  metadata:
    annotations:
      k8s.v1.cni.cncf.io/resourceName: rancher.io/intelnicsDpdk
    creationTimestamp: "2023-06-08T11:22:27Z"
    generation: 1
    name: network-dpdk
    namespace: sriov-network-operator
    resourceVersion: "13124"
    uid: df7c89f5-177c-4f30-ae72-7aef3294fb15
  spec:
    config: '{ "cniVersion":"0.4.0", "name":"network-
dpdk","type":"sriov","vlan":500,"vlanQoS":0,"ipam":{"type":"host-
local","subnet":"192.168.0.0/24","rangeStart":"192.168.0.10","rangeEnd":"192.168.0.60"
,"routes":[{"dst":"0.0.0.0/0"}],"gateway":"192.168.0.1"}
      }'
kind: List
metadata:
  resourceVersion: ""
```

DPDK

DPDK (Data Plane Development Kit) is a set of libraries and drivers for fast packet processing. It is used to accelerate packet processing workloads running on a wide variety of CPU architectures. The DPDK includes data plane libraries and optimized network interface controller (NIC) drivers for the following:

1. A queue manager implements lockless queues.
2. A buffer manager pre-allocates fixed size buffers.
3. A memory manager allocates pools of objects in memory and uses a ring to store free objects; ensures that objects are spread equally on all DRAM channels.
4. Poll mode drivers (PMD) are designed to work without asynchronous notifications, reducing overhead.
5. A packet framework as a set of libraries that are helpers to develop packet processing.

The following steps will show how to enable DPDK and how to create VFs from the NICs to be used by the DPDK interfaces:

- Install the DPDK package:

```
$ transactional-update pkg install dpdk dpdk-tools libdpdk-23
$ reboot
```

- Kernel parameters:

To use DPDK, employ some drivers to enable certain parameters in the kernel:

parameter	value	description
iommu	pt	This option enables the use of the vfio driver for the DPDK interfaces.
intel_iommu or amd_iommu	on	This option enables the use of vfio for VFs.

To enable the parameters, add them to the /etc/default/grub file:

```
GRUB_CMDLINE_LINUX="BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt root=UUID=77b713de-5cc7-4d4c-
8fc6-f5eca0a43cf9 skew_tick=1 rd.timeout=60 rd.retry=45 console=ttyS1,115200
console=tty0 default_hugepagesz=1G hugepagesz=1G hugepages=40 hugepagesz=2M
hugepages=0 ignition.platform.id=openstack intel_iommu=on iommu=pt
irqaffinity=0,31,32,63 isolcpus=domain,nohz,managed_irq,1-30,33-62 nohz_full=1-30,33-
62 nohz=on mce=off net.ifnames=0 nosoftlockup nowatchdog nmi_watchdog=0 quiet
rcu_nocb_poll rcu_nocbs=1-30,33-62 rcupdate.rcu_cpu_stall_suppress=1
rcupdate.rcu_expedited=1 rcupdate.rcu_normal_after_boot=1
rcupdate.rcu_task_stall_timeout=0 rcutree.kthread_prio=99 security=selinux selinux=1
idle=poll"
```

Update the GRUB configuration and reboot the system to apply the changes:

```
$ transactional-update grub.cfg
$ reboot
```

- Load vfio-pci kernel module and enable SR-IOV on the NICs:

```
$ modprobe vfio-pci enable_sriov=1 disable_idle_d3=1
```

- Create some virtual functions (VFs) from the NICs.

To create for VFs, for example, for two different NICs, the following commands are required:

```
$ echo 4 > /sys/bus/pci/devices/0000:51:00.0/sriov_numvfs
$ echo 4 > /sys/bus/pci/devices/0000:51:00.1/sriov_numvfs
```

- Bind the new VFs with the vfio-pci driver:

```
$ dpdk-devbind.py -b vfio-pci 0000:51:01.0 0000:51:01.1 0000:51:01.2 0000:51:01.3 \
                              0000:51:11.0 0000:51:11.1 0000:51:11.2 0000:51:11.3
```

- Review the configuration is correctly applied:

```
$ dpdk-devbind.py -s

Network devices using DPDK-compatible driver
============================================
0000:51:01.0 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci
unused=iavf,igb_uio
0000:51:01.1 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci
unused=iavf,igb_uio
0000:51:01.2 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci
unused=iavf,igb_uio
0000:51:01.3 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci
unused=iavf,igb_uio
0000:51:01.0 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci
unused=iavf,igb_uio
0000:51:11.1 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci
unused=iavf,igb_uio
0000:51:21.2 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci
unused=iavf,igb_uio
0000:51:31.3 'Ethernet Adaptive Virtual Function 1889' drv=vfio-pci
unused=iavf,igb_uio

Network devices using kernel driver
===================================
0000:19:00.0 'BCM57504 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet 1751'
if=em1 drv=bnxt_en unused=igb_uio,vfio-pci *Active*
0000:19:00.1 'BCM57504 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet 1751'
if=em2 drv=bnxt_en unused=igb_uio,vfio-pci
0000:19:00.2 'BCM57504 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet 1751'
if=em3 drv=bnxt_en unused=igb_uio,vfio-pci
0000:19:00.3 'BCM57504 NetXtreme-E 10Gb/25Gb/40Gb/50Gb/100Gb/200Gb Ethernet 1751'
if=em4 drv=bnxt_en unused=igb_uio,vfio-pci
0000:51:00.0 'Ethernet Controller E810-C for QSFP 1592' if=eth13 drv=ice
unused=igb_uio,vfio-pci
0000:51:00.1 'Ethernet Controller E810-C for QSFP 1592' if=rename8 drv=ice
unused=igb_uio,vfio-pci
```

vRAN acceleration (Intel ACC100/ACC200)

As communications service providers move from 4 G to 5 G networks, many are adopting virtualized radio access network (vRAN) architectures for higher channel capacity and easier deployment of edge-based services and applications. vRAN solutions are ideally located to deliver low-latency services with the flexibility to increase or decrease capacity based on the volume of

real-time traffic and demand on the network.

One of the most compute-intensive 4 G and 5 G workloads is RAN layer 1 (L1) FEC, which resolves data transmission errors over unreliable or noisy communication channels. FEC technology detects and corrects a limited number of errors in 4 G or 5 G data, eliminating the need for retransmission. Since the FEC acceleration transaction does not contain cell state information, it can be easily virtualized, enabling pooling benefits and easy cell migration.

- Kernel parameters

To enable the vRAN acceleration, we need to enable the following kernel parameters (if not present yet):

parameter	value	description
iommu	pt	This option enables the use of vfio for the DPDK interfaces.
intel_iommu or amd_iommu	on	This option enables the use of vfio for VFs.

Modify the GRUB file /etc/default/grub to add them to the kernel command line:

```
GRUB_CMDLINE_LINUX="BOOT_IMAGE=/boot/vmlinuz-6.4.0-9-rt root=UUID=77b713de-5cc7-4d4c-
8fc6-f5eca0a43cf9 skew_tick=1 rd.timeout=60 rd.retry=45 console=ttyS1,115200
console=tty0 default_hugepagesz=1G hugepagesz=1G hugepages=40 hugepagesz=2M
hugepages=0 ignition.platform.id=openstack intel_iommu=on iommu=pt
irqaffinity=0,31,32,63 isolcpus=domain,nohz,managed_irq,1-30,33-62 nohz_full=1-30,33-
62 nohz=on mce=off net.ifnames=0 nosoftlockup nowatchdog nmi_watchdog=0 quiet
rcu_nocb_poll rcu_nocbs=1-30,33-62 rcupdate.rcu_cpu_stall_suppress=1
rcupdate.rcu_expedited=1 rcupdate.rcu_normal_after_boot=1
rcupdate.rcu_task_stall_timeout=0 rcutree.kthread_prio=99 security=selinux selinux=1
idle=poll"
```

Update the GRUB configuration and reboot the system to apply the changes:

```
$ transactional-update grub.cfg
$ reboot
```

To verify that the parameters are applied after the reboot, check the command line:

```
$ cat /proc/cmdline
```

- Load vfio-pci kernel modules to enable the vRAN acceleration:

```
$ modprobe vfio-pci enable_sriov=1 disable_idle_d3=1
```

- Get interface information Acc100:

```
$ lspci | grep -i acc
8a:00.0 Processing accelerators: Intel Corporation Device 0d5c
```

- Bind the physical interface (PF) with vfio-pci driver:

```
$ dpdk-devbind.py -b vfio-pci 0000:8a:00.0
```

- Create the virtual functions (VFs) from the physical interface (PF).

Create 2 VFs from the PF and bind with vfio-pci following the next steps:

```
$ echo 2 > /sys/bus/pci/devices/0000:8a:00.0/sriov_numvfs
$ dpdk-devbind.py -b vfio-pci 0000:8b:00.0
```

- Configure acc100 with the proposed configuration file:

```
$ pf_bb_config ACC100 -c /opt/pf-bb-config/acc100_config_vf_5g.cfg
Tue Jun  6 10:49:20 2023:INFO:Queue Groups: 2 5GUL, 2 5GDL, 2 4GUL, 2 4GDL
Tue Jun  6 10:49:20 2023:INFO:Configuration in VF mode
Tue Jun  6 10:49:21 2023:INFO: ROM version MM 99AD92
Tue Jun  6 10:49:21 2023:WARN:* Note: Not on DDR PRQ version  1302020 != 10092020
Tue Jun  6 10:49:21 2023:INFO:PF ACC100 configuration complete
Tue Jun  6 10:49:21 2023:INFO:ACC100 PF [0000:8a:00.0] configuration complete!
```

- Check the new VFs created from the FEC PF:

```
$ dpdk-devbind.py -s
Baseband devices using DPDK-compatible driver
==============================================
0000:8a:00.0 'Device 0d5c' drv=vfio-pci unused=
0000:8b:00.0 'Device 0d5d' drv=vfio-pci unused=

Other Baseband devices
======================
0000:8b:00.1 'Device 0d5d' unused=
```

Huge pages

When a process uses RAM, the CPU marks it as used by that process. For efficiency, the CPU allocates RAM in chunks 4K bytes is the default value on many platforms. Those chunks are named pages. Pages can be swapped to disk, etc.

Since the process address space is virtual, the CPU and the operating system need to remember which pages belong to which process, and where each page is stored. The greater the number of pages, the longer the search for memory mapping. When a process uses 1 GB of memory, that is 262144 entries to look up (1 GB / 4 K). If a page table entry consumes 8 bytes, that is 2 MB (262144 * 8) to look up.

Most current CPU architectures support larger-than-default pages, which give the CPU/OS fewer entries to look up.

- Kernel parameters

To enable the huge pages, we should add the following kernel parameters. In this example, we configure 40 1G pages, though the huge page size and exact number should be tailored to your application's memory requirements:

parameter	value	description
hugepagesz	1G	This option allows to set the size of huge pages to 1 G
hugepages	40	This is the number of huge pages defined before
default_hugepagesz	1G	This is the default value to get the huge pages

Modify the GRUB file /etc/default/grub to add these parameters in GRUB_CMDLINE_LINUX:

```
default_hugepagesz=1G hugepagesz=1G hugepages=40 hugepagesz=2M hugepages=0
```

Update the GRUB configuration and reboot the system to apply the changes:

```
$ transactional-update grub.cfg
$ reboot
```

To validate that the parameters are applied after the reboot, you can check the command line:

```
$ cat /proc/cmdline
```

- Using huge pages

To use the huge pages, we need to mount them:

```
$ mkdir -p /hugepages
$ mount -t hugetlbfs nodev /hugepages
```

Deploy a Kubernetes workload, creating the resources and the volumes:

```
...
 resources:
   requests:
     memory: "24Gi"
     hugepages-1Gi: 16Gi
     intel.com/intel_sriov_oru: '4'
   limits:
     memory: "24Gi"
     hugepages-1Gi: 16Gi
     intel.com/intel_sriov_oru: '4'
...
```

```
...
volumeMounts:
  - name: hugepage
    mountPath: /hugepages
...
volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
...
```

CPU pinning on Kubernetes

Prerequisite

Must have the `CPU` tuned to the performance profile covered in this section.

Configure Kubernetes for CPU Pinning

Configure kubelet arguments to implement CPU management in the `RKE2` cluster. Add the following configuration block such as below example to your `/etc/rancher/rke2/config.yaml` file. Make sure specifying the housekeeping CPU cores in `kubelet-reserved` and `system-reserved` arguments:

```
kubelet-arg:
- "cpu-manager-policy=static"
- "cpu-manager-policy-options=full-pcpus-only=true"
- "cpu-manager-reconcile-period=0s"
- "kubelet-reserved=cpu=0,31,32,63"
- "system-reserved=cpu=0,31,32,63"
```

Leveraging Pinned CPUs for Workloads

There are three ways to use that feature using the `Static Policy` defined in kubelet depending on

the requests and limits you define on your workload:

1. `BestEffort` QoS Class: If you do not define any request or limit for `CPU`, the pod is scheduled on the first `CPU` available on the system.

   An example of using the `BestEffort` QoS Class could be:

   ```
   spec:
     containers:
     - name: nginx
       image: nginx
   ```

2. `Burstable` QoS Class: If you define a request for CPU, which is not equal to the limits, or there is no CPU request.

   Examples of using the `Burstable` QoS Class could be:

   ```
   spec:
     containers:
     - name: nginx
       image: nginx
       resources:
         limits:
           memory: "200Mi"
         requests:
           memory: "100Mi"
   ```

   or

   ```
   spec:
     containers:
     - name: nginx
       image: nginx
       resources:
         limits:
           memory: "200Mi"
           cpu: "2"
         requests:
           memory: "100Mi"
           cpu: "1"
   ```

3. `Guaranteed` QoS Class: If you define a request for CPU, which is equal to the limits.

   An example of using the `Guaranteed` QoS Class could be:

   ```
   spec:
     containers:
   ```

```
    - name: nginx
      image: nginx
      resources:
        limits:
          memory: "200Mi"
          cpu: "2"
        requests:
          memory: "200Mi"
          cpu: "2"
```

NUMA-aware scheduling

Non-Uniform Memory Access or Non-Uniform Memory Architecture (NUMA) is a physical memory design used in SMP (multiprocessors) architecture, where the memory access time depends on the memory location relative to a processor. Under NUMA, a processor can access its own local memory faster than non-local memory, that is, memory local to another processor or memory shared between processors.

Identifying NUMA nodes

To identify the NUMA nodes, on your system use the following command:

```
$ lscpu | grep NUMA
NUMA node(s):                    1
NUMA node0 CPU(s):               0-63
```

NOTE

For this example, we have only one NUMA node showing 64 CPUs.

NUMA needs to be enabled in the BIOS. If dmesg does not have records of NUMA initialization during the bootup, then NUMA-related messages in the kernel ring buffer might have been overwritten.

Metal LB

MetalLB is a load-balancer implementation for bare-metal Kubernetes clusters, using standard routing protocols like L2 and BGP as advertisement protocols. It is a network load balancer that can be used to expose services in a Kubernetes cluster to the outside world due to the need to use Kubernetes Services type LoadBalancer with bare-metal.

To enable MetalLB in the RKE2 cluster, the following steps are required:

- Install MetalLB using the following command:

```
$ kubectl apply <<EOF -f
apiVersion: helm.cattle.io/v1
kind: HelmChart
```

```
metadata:
  name: metallb
  namespace: kube-system
spec:
  chart: oci://registry.suse.com/edge/charts/metallb
  targetNamespace: metallb-system
  version: {version-metallb-chart}
  createNamespace: true
---
apiVersion: helm.cattle.io/v1
kind: HelmChart
metadata:
  name: endpoint-copier-operator
  namespace: kube-system
spec:
  chart: oci://registry.suse.com/edge/charts/endpoint-copier-operator
  targetNamespace: endpoint-copier-operator
  version: {version-endpoint-copier-operator-chart}
  createNamespace: true
EOF
```

- Create the `IpAddressPool` and the `L2advertisement` configuration:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: kubernetes-vip-ip-pool
  namespace: metallb-system
spec:
  addresses:
    - 10.168.200.98/32
  serviceAllocation:
    priority: 100
    namespaces:
      - default
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: ip-pool-l2-adv
  namespace: metallb-system
spec:
  ipAddressPools:
    - kubernetes-vip-ip-pool
```

- Create the endpoint service to expose the `VIP`:

```
apiVersion: v1
kind: Service
```

```
metadata:
  name: kubernetes-vip
  namespace: default
spec:
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: rke2-api
    port: 9345
    protocol: TCP
    targetPort: 9345
  - name: k8s-api
    port: 6443
    protocol: TCP
    targetPort: 6443
  sessionAffinity: None
  type: LoadBalancer
```

- Check the `VIP` is created and the `MetalLB` pods are running:

```
$ kubectl get svc -n default
$ kubectl get pods -n default
```

Private registry configuration

`Containerd` can be configured to connect to private registries and use them to pull private images on each node.

Upon startup, `RKE2` checks if a `registries.yaml` file exists at `/etc/rancher/rke2/` and instructs `containerd` to use any registries defined in the file. If you wish to use a private registry, create this file as root on each node that will use the registry.

To add the private registry, create the file `/etc/rancher/rke2/registries.yaml` with the following content:

```
mirrors:
  docker.io:
    endpoint:
      - "https://registry.example.com:5000"
configs:
  "registry.example.com:5000":
    auth:
      username: xxxxxx # this is the registry username
      password: xxxxxx # this is the registry password
    tls:
      cert_file:            # path to the cert file used to authenticate to the
```

```
registry
      key_file:             # path to the key file for the certificate used to
authenticate to the registry
      ca_file:              # path to the ca file used to verify the registry's
certificate
      insecure_skip_verify: # may be set to true to skip verifying the registry's
certificate
```

or without authentication:

```
mirrors:
  docker.io:
    endpoint:
      - "https://registry.example.com:5000"
configs:
  "registry.example.com:5000":
    tls:
      cert_file:            # path to the cert file used to authenticate to the
registry
      key_file:             # path to the key file for the certificate used to
authenticate to the registry
      ca_file:              # path to the ca file used to verify the registry's
certificate
      insecure_skip_verify: # may be set to true to skip verifying the registry's
certificate
```

For the registry changes to take effect, you need to either configure this file before starting RKE2 on the node, or restart RKE2 on each configured node.

NOTE For more information about this, please check containerd registry configuration rke2.

Precision Time Protocol

Precision Time Protocol (PTP) is a network protocol developed by the Institute of Electrical and Electronics Engineers (IEEE) to enable sub-microsecond time synchronization in a computer network. Since its inception and for a couple of decades now, PTP has been in use in many industries. It has recently seen a growing adoption in the telecommunication networks as a vital element to 5G networks. While being a relatively simple protocol, its configuration can change significantly depending on the application. For this reason, multiple profiles have been defined and standardized.

In this section, only telco-specific profiles will be covered. Consequently time-stamping capability and a PTP hardware clock (PHC) in the NIC will be assumed. Nowadays, all telco-grade network adapters come with PTP support in hardware, but you can verify such capabilities with the following command:

```
# ethtool -T p1p1
Time stamping parameters for p1p1:
Capabilities:
        hardware-transmit
        software-transmit
        hardware-receive
        software-receive
        software-system-clock
        hardware-raw-clock
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off
        on
Hardware Receive Filter Modes:
        none
        all
```

Replace `p1p1` with name of the interface to be used for PTP.

The following sections will provide guidance on how to install and configure PTP on SUSE Edge specifically, but familiarity with basic PTP concepts is expected. For a brief overview of PTP and the implementation included in SUSE Edge for Telco, refer to https://documentation.suse.com/sles/html/SLES-all/cha-tuning-ptp.html.

Install PTP software components

In SUSE Edge for Telco, the PTP implementation is provided by the `linuxptp` package, which includes two components:

- `ptp4l`: a daemon that controls the PHC on the NIC and runs the PTP protocol
- `phc2sys`: a daemon that keeps the system clock in sync with the PTP-synchronized PHC on the NIC

Both daemons are required for the system synchronization to fully work and must be correctly configured according to your setup. This is covered in Configure PTP for telco deployments.

The easiest and best way to integrate PTP in your downstream cluster is to add the `linuxptp` package under `packageList` in the Edge Image Builder (EIB) definition file. This way the PTP control plane software will be installed automatically during the cluster provisioning. See the EIB documentation for more information on installing packages.

Below find a sample EIB manifest with `linuxptp`:

```
apiVersion: 1.0
image:
  imageType: RAW
  arch: x86_64
  baseImage: {micro-base-rt-image-raw}
```

```
  outputImageName: eibimage-slmicrort-telco.raw
operatingSystem:
  time:
    timezone: America/New_York
  kernelArgs:
    - ignition.platform.id=openstack
    - net.ifnames=1
  systemd:
    disable:
      - rebootmgr
      - transactional-update.timer
      - transactional-update-cleanup.timer
      - fstrim
      - time-sync.target
    enable:
      - ptp4l
      - phc2sys
  users:
    - username: root
      encryptedPassword: ${ROOT_PASSWORD}
  packages:
    packageList:
      - jq
      - dpdk
      - dpdk-tools
      - libdpdk-23
      - pf-bb-config
      - open-iscsi
      - tuned
      - cpupower
      - linuxptp
    sccRegistrationCode: ${SCC_REGISTRATION_CODE}
```

NOTE

The `linuxptp` package included in SUSE Edge for Telco does not enable `ptp4l` and `phc2sys` by default. If their system-specific configuration files are deployed at provisioning time (see Cluster API integration), they should be enabled. Do so by adding them to the `systemd` section of the manifest, as in the example above.

Follow the usual process to build the image as described in the EIB Documentation and use it to deploy your cluster. If you are new to EIB, start from ../components/edge-image-builder.pdf instead.

Configure PTP for telco deployments

Many telco applications require strict phase and time synchronization with little deviance, which resulted in a definition of two telco-oriented profiles: the ITU-T G.8275.1 and ITU-T G.8275.2. They both have a high rate of sync messages and other distinctive traits, such as the use of an alternative Best Master Clock Algorithm (BMCA). Such behavior mandates specific settings in the configuration file consumed by `ptp4l`, provided in the following sections as a reference.

NOTE

- Both sections only cover the case of an ordinary clock in Time Receiver configuration.
- Any such profile must be used in a well-planned PTP infrastructure.
- Your specific PTP network may require additional configuration tuning, make sure to review and adapt the provided examples if needed.

PTP profile ITU-T G.8275.1

The G.8275.1 profile has the following specifics:

- Runs directly on Ethernet and requires full network support (adjacent nodes/switches must support PTP).
- The default domain setting is 24.
- Dataset comparison is based on the G.8275.x algorithm and its `localPriority` values after `priority2`.

Copy the following content to a file named `/etc/ptp4l-G.8275.1.conf`:

```
# Telecom G.8275.1 example configuration
[global]
domainNumber                    24
priority2           255
dataset_comparison              G.8275.x
G.8275.portDS.localPriority     128
G.8275.defaultDS.localPriority  128
maxStepsRemoved                 255
logAnnounceInterval             -3
logSyncInterval                 -4
logMinDelayReqInterval          -4
announceReceiptTimeout      3
serverOnly                      0
ptp_dst_mac                     01:80:C2:00:00:0E
network_transport               L2
```

Once the file has been created, it must be referenced in `/etc/sysconfig/ptp4l` for the daemon to start correctly. This can be done by changing the `OPTIONS=` line to:

```
OPTIONS="-f /etc/ptp4l-G.8275.1.conf -i $IFNAME --message_tag ptp-8275.1"
```

More precisely:

- `-f` requires the file name of the configuration file to use; `/etc/ptp4l-G.8275.1.conf` in this case
- `-i` requires the name of the interface to use, replace `$IFNAME` with a real interface name.
- `--message_tag` allows to better identify the ptp4l output in the system logs and is optional.

Once the steps above are complete, the `ptp4l` daemon must be (re)started:

```
# systemctl restart ptp4l
```

Check the synchronization status by observing the logs with:

```
# journalctl -e -u ptp4l
```

PTP profile ITU-T G.8275.2

The G.8275.2 profile has the following specifics:

- Runs on IP and does not require full network support (adjacent nodes/switches may not support PTP).
- The default domain setting is 44.
- Dataset comparison is based on the G.8275.x algorithm and its `localPriority` values after `priority2`.

Copy the following content to a file named `/etc/ptp4l-G.8275.2.conf`:

```
# Telecom G.8275.2 example configuration
[global]
domainNumber                    44
priority2           255
dataset_comparison              G.8275.x
G.8275.portDS.localPriority     128
G.8275.defaultDS.localPriority  128
maxStepsRemoved                 255
logAnnounceInterval             0
serverOnly                      0
hybrid_e2e                      1
inhibit_multicast_service       1
unicast_listen                  1
unicast_req_duration            60
logSyncInterval                 -5
logMinDelayReqInterval          -4
announceReceiptTimeout      2
#
# Customize the following for slave operation:
#
[unicast_master_table]
table_id                        1
logQueryInterval                2
UDPv4                           $PEER_IP_ADDRESS
[$IFNAME]
unicast_master_table            1
```

Make sure to replace the following placeholders:

- `$PEER_IP_ADDRESS` - the IP address of the next PTP node to communicate with, such as the master or boundary clock that will provide synchronization.
- `$IFNAME` - tells `ptp4l` what interface to use for PTP.

Once the file has been created, it must be referenced, along with the name of the interface to use for PTP, in `/etc/sysconfig/ptp4l` for the daemon to start correctly. This can be done by changing the `OPTIONS=` line to:

```
OPTIONS="-f /etc/ptp4l-G.8275.2.conf --message_tag ptp-8275.2"
```

More precisely:

- `-f` requires the file name of the configuration file to use. In this case, it is `/etc/ptp4l-G.8275.2.conf`.
- `--message_tag` allows to better identify the ptp4l output in the system logs and is optional.

Once the steps above are complete, the `ptp4l` daemon must be (re)started:

```
# systemctl restart ptp4l
```

Check the synchronization status by observing the logs with:

```
# journalctl -e -u ptp4l
```

Configuration of phc2sys

Although not required, it is recommended that you fully complete the configuration of `ptp4l` before moving to `phc2sys`. `phc2sys` does not require a configuration file and its execution parameters can be solely controlled through the `OPTIONS=` variable present in `/etc/sysconfig/ptp4l`, in a similar fashion to `ptp4l`:

```
OPTIONS="-s $IFNAME -w"
```

Where `$IFNAME` is the name of the interface already set up in ptp4l that will be used as the source for the system clock. This is used to identify the source PHC.

Cluster API integration

Whenever a cluster is deployed through a management cluster and directed provisioning, both the configuration file and the two configuration variables in `/etc/sysconfig` can be deployed on the host at provisioning time. Below is an excerpt from a cluster definition, focusing on a modified `RKE2ControlPlane` object that deploys the same G.8275.1 configuration file on all hosts:

```
apiVersion: controlplane.cluster.x-k8s.io/v1beta1
kind: RKE2ControlPlane
metadata:
  name: single-node-cluster
  namespace: default
spec:
  infrastructureRef:
    apiVersion: infrastructure.cluster.x-k8s.io/v1beta1
    kind: Metal3MachineTemplate
    name: single-node-cluster-controlplane
  replicas: 1
  version: ${RKE2_VERSION}
  rolloutStrategy:
    type: "RollingUpdate"
    rollingUpdate:
      maxSurge: 0
  registrationMethod: "control-plane-endpoint"
  serverConfig:
    cni: canal
  agentConfig:
    format: ignition
    cisProfile: cis
    additionalUserData:
      config: |
        variant: fcos
        version: 1.4.0
        systemd:
          units:
            - name: rke2-preinstall.service
              enabled: true
              contents: |
                [Unit]
                Description=rke2-preinstall
                Wants=network-online.target
                Before=rke2-install.service
                ConditionPathExists=!/run/cluster-api/bootstrap-success.complete
                [Service]
                Type=oneshot
                User=root
                ExecStartPre=/bin/sh -c "mount -L config-2 /mnt"
                ExecStart=/bin/sh -c "sed -i \"s/BAREMETALHOST_UUID/$(jq -r .uuid
/mnt/openstack/latest/meta_data.json)/\" /etc/rancher/rke2/config.yaml"
                ExecStart=/bin/sh -c "echo \"node-name: $(jq -r .name
/mnt/openstack/latest/meta_data.json)\" >> /etc/rancher/rke2/config.yaml"
                ExecStartPost=/bin/sh -c "umount /mnt"
                [Install]
                WantedBy=multi-user.target
        storage:
          files:
            - path: /etc/ptp4l-G.8275.1.conf
```

```
              overwrite: true
              contents:
                inline: |
                  # Telecom G.8275.1 example configuration
                  [global]
                  domainNumber                      24
                  priority2                         255
                  dataset_comparison                G.8275.x
                  G.8275.portDS.localPriority       128
                  G.8275.defaultDS.localPriority    128
                  maxStepsRemoved                   255
                  logAnnounceInterval               -3
                  logSyncInterval                   -4
                  logMinDelayReqInterval            -4
                  announceReceiptTimeout            3
                  serverOnly                        0
                  ptp_dst_mac                       01:80:C2:00:00:0E
                  network_transport                 L2
              mode: 0644
              user:
                name: root
              group:
                name: root
            - path: /etc/sysconfig/ptp4l
              overwrite: true
              contents:
                inline: |
                  ## Path:           Network/LinuxPTP
                  ## Description:    Precision Time Protocol (PTP): ptp4l settings
                  ## Type:           string
                  ## Default:        "-i eth0 -f /etc/ptp4l.conf"
                  ## ServiceRestart: ptp4l
                  #
                  # Arguments when starting ptp4l(8).
                  #
                  OPTIONS="-f /etc/ptp4l-G.8275.1.conf -i $IFNAME --message_tag ptp-
8275.1"
              mode: 0644
              user:
                name: root
              group:
                name: root
            - path: /etc/sysconfig/phc2sys
              overwrite: true
              contents:
                inline: |
                  ## Path:           Network/LinuxPTP
                  ## Description:    Precision Time Protocol (PTP): phc2sys settings
                  ## Type:           string
                  ## Default:        "-s eth0 -w"
                  ## ServiceRestart: phc2sys
```

```
                #
                # Arguments when starting phc2sys(8).
                #
                OPTIONS="-s $IFNAME -w"
            mode: 0644
            user:
              name: root
            group:
              name: root
    kubelet:
      extraArgs:
        - provider-id=metal3://BAREMETALHOST_UUID
    nodeName: "localhost.localdomain"
```

Besides other variables, the above definition must be completed with the interface name and with the other Cluster API objects, as described in [atip-automated-provisioning].

NOTE

- This approach is convenient only if the hardware in the cluster is uniform and the same configuration is needed on all hosts, interface name included.
- Alternative approaches are possible and will be covered in future releases.

At this point, your hosts should have a working and running PTP stack and will start negotiating their PTP role.