RedHawk *KVM-RT ™ User's Guide*

concurrent REAL-TIME

0898604-1.1
January 2020

Printed in U. S. A.

Revision History:	Level:	Effective With:
July 2019	1.0	RedHawk Linux 7.5
January 2020	1.1	RedHawk Linux 8.0

Preface

Scope of Manual

This manual provides information and instructions for using Concurrent Real-Time's RedHawk KVM-RT™.

Structure of Manual

This manual consists of:

- Chapter 1, which introduces you to KVM-RT.
- Chapter 2, which explains the steps in setting up and booting virtual machines in KVM-RT.
- Chapter 3, which discusses configuring KVM-RT.
- Chapter 4, which lists known issues.

Syntax Notation

The following notation is used throughout this manual:

italic	Books, reference cards, and items that the user must specify appear in *italic* type. Special terms may also appear in *italic*.
`list bold`	User input appears in **`list bold`** type and must be entered exactly as shown. Names of directories, files, commands, options and man page references also appear in **`list bold`** type.
	Operating system and program output such as prompts, messages and listings of files and programs appears in `list` type.
`[]`	Brackets enclose command options and arguments that are optional. You do not type the brackets if you choose to specify these options or arguments.
hypertext links	When viewing this document online, clicking on chapter, section, figure, table and page number references will display the corresponding text. Clicking on Internet URLs provided in blue type will launch your web browser and display the web site. Clicking on publication names and numbers in red type will display the corresponding manual PDF, if accessible.

Related Publications

The following table lists Concurrent Real-Time documentation. Depending upon the document, they are available online on RedHawk Linux systems or from Concurrent Real-Time's documentation web site at **http://redhawk.concurrent-rt.com/docs**.

RedHawk KVM-RT	**Pub. Number**
RedHawk KVM-RT Release Notes	0898603
RedHawk KVM-RT User's Guide	0898604
RedHawk Architect	
RedHawk Architect Release Notes	0898600
RedHawk Architect User's Guide	0898601
RedHawk Linux	
RedHawk Linux Release Notes	0898003
RedHawk Linux User's Guide	0898004
RedHawk Linux Cluster Manager User's Guide	0898016
RedHawk Linux FAQ	N/A
NightStar RT Development Tools	
NightView User's Guide	0898395
NightTrace User's Guide	0898398
NightProbe User's Guide	0898465
NightTune User's Guide	0898515

Contents

1
Introduction to KVM-RT

This chapter provides a general overview and requirements for using RedHawk KVM-RT.

Introduction

RedHawk KVM-RT is a Real-Time Hypervisor solution that utilizes QEMU/KVM and RedHawk real-time features to extend RedHawk's real-time determinism to guest RedHawk virtual machines.

It supports multiple guests, both real-time and non-real-time, running in virtual machines on a single host system.

Host System Requirements and Installation

Refer to the *RedHawk KVM-RT Release Notes* for hardware host system requirements and software installation instructions.

Though not a requirement, it is highly recommended that the entire host system be dedicated to running the Real-Time Hypervisor. Administrators of the KVM-RT host system must be careful not to disturb CPU shielding or CPU affinities on the system, or else real-time performance of virtual machines may be compromised.

Once KVM-RT is installed, the following command can be run to test the suitability of the of the host system.

```
$ sudo kvmrt-validate-host
```

Host Kernel Configuration

KVM-RT requires that a RedHawk kernel is booted on the host system while KVM-RT is being used. Additionally, the following kernel boot parameters may be required:

intel_iommu = on | amd_iommu = on

> This parameter enables device-level remapping of memory regions used for DMA in virtual machines. This parameter is required if any virtual machines will use PCI passthrough of physical PCI devices.

workqueue.pri = 3

Virtual machines that are enabled as real-time virtual machines will effectively "own" physical CPUs. This parameter allows host daemons and processes to run at a higher priority to allow the host to maintain control of the virtual machines.This parameter is required to guarantee real-time performance to real-time VMs.

These parameters may be added to the system's boot time parameters with the command **blscfg(1)** in RedHawk Linux version 8.0 and **ccur-grub2(1)** in RedHawk Linux version 7.5. Note that a reboot is necessary for the parameters to take effect.

2
Getting Started

This chapter explains the steps in setting up and booting virtual machines in KVM-RT. Also discussed are the various QEMU/KVM threads that run on the host for each virtual machine.

Building Virtual Machines

KVM-RT works with virtual machines that have been created and configured within the libvirt framework. A virtual machine may be created and configured within libvirt in several ways, including:

- with Virtual Machine Manager
- with RedHawk Architect
- by cloning another virtual machine

Detailed instructions on how to build virtual machines are beyond the scope of this book but are well documented. General instructions and references to documentation are given in the following sections.

Real-time virtual machines must contain a guest OS of RedHawk Linux 7.5 or later. The guest CPU architecture must match that of the host.

Using Virtual Machine Manager to Create a Virtual Machine

The Virtual Machine Manager is a GUI tool that can be used to create, configure, and manage virtual machines within the libvirt framework.

Start Virtual Machine Manager by running:

```
$ sudo run virt-manager
```

See the **virt-manager(1)** man page for more information.

Using RedHawk Architect to Create a Virtual Machine

RedHawk Architect is an optional product offered by Concurrent Real-Time that specializes in creating, customizing and deploying RedHawk Linux disk images.

Architect can be used to create a RedHawk virtual machine and to export it to the Virtual Machine Manager. Detailed instructions can be found in the documentation that comes with RedHawk Architect. Below are the general steps required:

- run architect
- create a new session and configure the image as desired
- build the image
- deploy the image to a virtual machine
- export the virtual machine to Virtual Machine Manager

Cloning a Virtual Machine Image

Any existing virtual machine within the libvirt framework can be cloned by using the virt-clone command. For example:

```
$ sudo virt-clone -o old_vm -n new_vm
```

See the **virt-clone(1)** man page for more information.

Importing Virtual Machines into KVM-RT

Once virtual machines have been created within the libvirt framework, they can be imported into KVM-RT.

All libvirt virtual machines can be imported into KVM-RT with the following command:

```
$ sudo kvmrt-import
```

This command may be run at any time new VMs are created. Run **kvmrt-import --help** for more information and options.

When a VM is imported into KVM-RT it inherits the VM configuration settings from libvirt. Once this is done a VM may be further configured with KVM-RT as needed. See "Configuring Virtual Machines" in Chapter 3 for more information.

Booting and Shutting Down Virtual Machines

Once virtual machines have been imported into KVM-RT, the following KVM-RT tools can be used to boot, shutdown, and view the status of VMs.

Run the following command to start up all configured VMs:

```
$ sudo kvmrt-startup
```

Run the following command to shutdown all configured VMs:

```
$ sudo kvmrt-shutdown
```

Run the following command to query the state of all VMs:

```
$ sudo kvmrt-stat
```

Individual VMs can be booted or shutdown with the **kvmrt-start-vm** and **kvmrt-shutdown-vm** commands. For example:

```
$ sudo kvmrt-start-vm RedHawk-8.0
$ sudo kvmrt-shutdown-vm RedHawk-8.0
```

Run any of the above commands with the **--help** option for more information and options.

Understanding QEMU/KVM Threads

QEMU/KVM runs multiple threads for each virtual machine. The names and purpose of these threads are as follows:

qemu-kvm

These are emulator threads. There may be two or more of these.

qemu-system-x86

This is an alternate name for *qemu-kvm* in some distributions.

worker

These are dynamically created threads for long I/O operations being performed by the emulator.

SPICE Worker

This is a thread for a virtual console.

IO mon_ioth

This is an optional thread used for some I/O.

CPU n/KVM

These are virtual CPU (vCPU) threads. There will be one per virtual CPU, where *n* is the vCPU ID.

Use the **kvmrt-stat -t** command to display information about all currently running VM threads.

3
Configuring Virtual Machines

Virtual machines that are configured within the libvirt framework have an XML configuration file that controls all attributes of the virtual machine.

This file usually exists as "**/etc/libvirt/qemu/{DOMAIN}.xml**" for the given VM domain name and is created when the VM is created or imported into the libvirt framework. This file gets updated when VM configuration changes are made in Virtual Machine Manager.

KVM-RT uses a simplified configuration file, explained below, to manage multiple VMs. KVM-RT updates libvirt XML configuration files as needed to keep the two files in sync.

The KVM-RT Configuration File

The default location of the KVM-RT configuration file is **/etc/kvmrt.cfg**, but all **kvmrt-*** tools accept a **-f** option that allows the user to specify an alternate configuration file.

The KVM-RT configuration file uses the INI file format, where each section name is a VM (domain) name. An example configuration is shown below:

```
[RedHawk-8.0]
autostart = True
title = RedHawk 8.0
description = A RedHawk 8.0 VM.
nr_vcpus = 2
cpu_topology = auto
cpuset =
rt = False
rt_memory = auto
numatune = auto
hide_kvm = False

[RedHawk-8.0-RT]
autostart = True
title = Real-Time RedHawk 8.0
description = A RedHawk 8.0 VM configured real-time.
nr_vcpus = 4
cpu_topology = auto
cpuset = 1-5
rt = True
rt_memory = auto
numatune = auto
hide_kvm = False
```

Defined below are the field types used in the attribute description that follows:

{string}: any string

{int}: any integer

{bool}: **true**|**false**|**on**|**off** |**yes**|**no**|**1**|**0**
(case-insensitive)

{ID-set}: a string that describes a set of ranges of integers in a human-readable form such as "0,2,4-7,12-15"

Each VM may be configured with the following attributes. Note that if an attribute is not set or it is missing from the file, the default value is used.

autostart = {bool}

This attribute enables auto-starting of the VM with **kvmrt-startup**.
The default value is **true** (enabled).

title = {string}

This attribute sets the VM title.
The default value is "".

description = {string}

This attribute sets the VM description.
The default value is "".

nr_vcpus = {int}

This attribute defines the number of virtual CPUs in the VM.
The default value is **1**.

cpu_topology = {int}, {int}, {int} | **auto**

This attribute defines the CPU topology that is seen by the VM.

If not **auto**, the value must be a string of three positive integers separated by commas ("sockets, cores, threads"), to describe the CPU topology. sockets is the number of CPU sockets, cores is the number of cores per socket, and threads is the number of threads per core.

When the value is **auto**, the topology is set to one socket, *nr_vcpus* cores per socket, and one thread per core.

The default value is **auto**.

NOTE

If the guest virtual machine is running a Windows operating system, the *cpu_topology* attribute may have been set to a default value that will not work well in KVM-RT. It is best to change this setting to **auto**. See the item labeled "VMs running the Windows operating system" in Chapter 4 for more information.

cpuset = {ID-set}

This attribute defines host CPUs IDs to which all VM threads are biased. The default value is "" (no CPU biasing). See the section "Understanding the cpuset Configuration Attribute" later in this chapter for more information.

rt_memory = {bool} | **auto**

This attribute enables memory locking of all pages used by the VM.

When the value is **auto**, this option is enabled if the *rt* attribute is enabled and disabled if *rt* is disabled.

The default value is **auto**.

numatune = {ID-set} | **auto**

This attribute sets the host NUMA nodes to be used for memory allocation to the VM.

If not **auto**, the value must describe a set of host NUMA node IDs. The set may be empty, in which case memory will not be restricted to any host NUMA nodes. If *cpuset* is empty then memory will not be restricted to any host NUMA nodes.

When the value is **auto**, all NUMA nodes used by cpuset will be used.

The default value is **auto**.

hide_kvm = {bool}

This attribute hides KVM from the view of the guest OS in the VM. The default value is **false** (disabled).

rt = {bool}

This attribute configures the VM for real-time. The default value is **false** (disabled).

The *cpuset* and *rt_memory* attributes must be configured (enabled) when this attribute is enabled. It is also recommended to configure and enable *numatune* when this attribute is enabled.

Configuration Tools

A KVM-RT configuration can be edited by running the command:

```
$ sudo kvmrt-edit-config
```

Note that KVM-RT configuration files should not be edited directly. **kvmrt-edit-config** validates and also synchronizes the configuration with libvirt.

A KVM-RT configuration, as interpreted by KVM-RT, can be displayed by running the command:

```
$ sudo kvmrt-show-config
```

The **kvmrt-validate-config** and **kvmrt-sync-config** commands can be run to validate and synchronize, respectively, a configuration. Users do not normally need to run these commands directly when using **kvmrt-edit-config**.

Run any of the above commands with the **--help** option for more information and options.

Advanced Libvirt Configuration

Advanced configuration that is beyond the scope of the KVM-RT configuration file may be made to the libvirt XML files, using Virtual Machine Manager or 'virsh edit', but additional synchronization and validation steps are required for KVM-RT.

Note that some combinations of configuration may be invalid and users are encouraged to make configuration changes by editing the KVM-RT configuration file with **kvmrt-edit-config** whenever possible.

If libvirt XML files are modified by the user outside of KVM-RT, then it is necessary to run **kvmrt-sync-config -r** and **kvmrt-validate-config**, like so:

```
$ sudo kvmrt-sync-config -r
$ sudo kvmrt-validate-config
```

Also note that **kvmrt-import -u** may be used instead of **kvmrt-sync-config -r**, as in:

```
$ sudo kvmrt-import -u
$ sudo kvmrt-validate-config
```

The **kvmrt-validate-config** command will display appropriate errors or warnings for any invalid configuration.

Run any of the above commands with the **--help** option for more information and options.

Understanding the cpuset Configuration Attribute

The *cpuset* attribute controls host-CPU-biasing of the QEMU/KVM threads of a virtual machine.

The *cpuset* attribute may be used for both real-time and non-real-time VMs. If *cpuset* is empty then the VM will not be bound to any particular host CPUs.

The first CPU in *cpuset* will be allocated to all non-vCPU threads. The remaining CPUs in *cpuset* will be used by the vCPU threads as follows:

- The CPU placement policy for all virtual CPU threads is round-robin on the host CPUs defined by *cpuset* (after one CPU is allocated to all non-vCPU VM threads).
- Over-provisioning of host CPUs (more CPUs in *cpuset* than *nr_vcpus* + 1) results in each virtual CPU being biased to more than one host CPU.
- Under-provisioning of host CPUs (less CPUs in *cpuset* than *nr_vcpus* +1) results in more than one virtual CPU being biased to each host CPU.

Configuring Real-Time Virtual Machines

Perform the following steps to configure a VM for real-time:

- enable the *rt* configuration attribute
- enable the *rt_memory* attribute (**auto** is recommended)
- consider enabling the *numatune* attribute (**auto** is recommended)
- configure the *cpuset* attribute as described below

Configuring the *cpuset* attribute for a real-time VM requires some understanding of the host system's CPU topology. Use the **cpu-topology** command to see a display of the host system's CPU topology:

```
$ cpu-topology
```

Run **cpu-topology --help** for more information and options.

cpu-topology displays the layout of CPU sockets, NUMA nodes, CPU cores, and logical CPUs. Example output follows:

```
NUMA Node 0:
        Core  0:              [Socket 0]
            CPU 0
        Core  1:              [Socket 0]
            CPU 1
        Core  2:              [Socket 0]
            CPU 2
        Core  3:              [Socket 0]
            CPU 3
```

If the system has a threaded-CPU architecture such as Intel's Hyper-Threading then the output may look something like this:

```
NUMA Node 0:
         Core  0:              [Socket 0]
             CPU 0
             CPU 4
         Core  1:              [Socket 0]
             CPU 1
             CPU 5
         Core  2:              [Socket 0]
             CPU 2
             CPU 6
         Core  3:              [Socket 0]
             CPU 3
             CPU 7
```

NUMA systems may have more than one NUMA node, for example:

```
NUMA Node 0:
         Core  0:              [Socket 0]
             CPU 0
             CPU 16
         Core  1:              [Socket 0]
             CPU 1
             CPU 17
         Core  2:              [Socket 0]
             CPU 2
             CPU 18
         ...

NUMA Node 1:
         Core  8:              [Socket 1]
             CPU 8
             CPU 24
         Core  9:              [Socket 1]
             CPU 9
             CPU 25
         Core 10:              [Socket 1]
             CPU 10
             CPU 26
         ...
```

The following rules should be observed when configuring a real-time VM:

- The *cpuset* of a real-time VM cannot overlap the *cpuset* of any other VM.
- The *cpuset* of a real-time VM must not be under-provisioned for the number of CPUs configured in the *nr_vcpus* attribute.
- Careful consideration should be given if the *cpuset* of a real-time VM spans multiple NUMA nodes.
- Careful consideration should be given if the *cpuset* of any other VM shares NUMA nodes with a real-time VM.

- Careful consideration should be given if *numatune* is not enabled for a real-time VM, or if the *numatune* node set is not contained within the NUMA nodes used by the *cpuset*.
- Careful consideration should be given if the *numatune* node set of any other VM overlaps with the NUMA nodes used by a real-time VM's *cpuset*.
- The cpusets of all real-time VMs must not consume all host CPUs. This is because some CPUs must be available for the KVM-RT host OS.

The KVM-RT tools will display appropriate errors or warnings for any of the above conditions.

Adhering to the following recommendations will help simplify real-time VM configuration:

- Always configure *cpuset* with a least *nr_vcpus* + 1 host CPUs.
- Do not configure the *cpuset* of any other VM to conflict with this VM's *cpuset*, or to use any other CPUs in a NUMA node used by this VM.
- Do not let the *cpuset* span multiple NUMA nodes.
- When configuring *cpuset*, allocate VM CPUs starting from the highest numbered core in each socket and count down, making sure to dedicate for the KVM-RT host the first core(s) in each socket. For example, in a CPU configuration with 2 sockets and 10 cores per socket, CPU allocations for the VMs should start with CPU #9 (last core in first socket) and count down; and with CPU #19 (last core in second socket) and count down. At the very least, CPU #0 (first core in first socket) and CPU #10 (first core in second socket) should be left unallocated for the KVM-RT host to use. See the item labeled "IRQs that cannot be migrated" in Chapter 4 for an explanation for this recommendation.
- Set *numatune* to **auto**.
- Do not configure the *numatune* of any other VM to include the NUMA node used by this VM.
- Use the **kvmrt-show-config** command to view the real-time policy configured for all VMs.
- Use the **kvmrt-stat -t** command to display the cpu-biasing of all currently running VM threads.

Understanding KVM-RT Use of Threaded CPUs

On host systems having a threaded-CPU architecture such as Intel's Hyper-Threading, KVM-RT gives special treatment to multi-threaded CPU cores when a real-time VM is in use.

Real-time demands that only one threaded sibling CPU be in use to avoid contention of CPU core resources (e.g. caches, etc.). To ensure this, KVM-RT shuts down all but one threaded sibling CPU for each CPU core allocated to a real-time VM. This requires some consideration when assigning VM *cpuset*s.

A real-time VM will be given ownership of all threaded sibling CPUs that are related to the CPUs specified in its *cpuset*. This may result in the VM consuming but not using

more CPUs than it has specified in it's *cpuset*. Only one CPU per threaded core will be used for real-time and the others will be shutdown.

No special treatment is given to threaded cores hosting non-real-time VMs.

4
Known Issues

Special consideration should be given to the following areas:

Known Issues in KVM-RT version 1.1

Graphics intensive programs

If a graphics intensive VM does not use dedicated GPU hardware, emulated graphics or any graphics intensive programs can cause VMs to affect the real-time performance of other virtual machines. The onboard graphics of a CPU rely on the memory controllers on the chip for accessing VRAM memory for the host and the VMs. If the memory controllers are overloaded with graphical accesses, the real-time VMs can suffer from performance hits.

VMs running the Windows operating system

There is a "per socket" license for Windows operating systems that will dramatically reduce the performance of a Windows virtual machine if you use the default libvirt CPU topology settings. The user should adjust the *cpu_topology* setting in the KVM-RT configuration to be many cores/threads on a single socket. It is recommended that the *cpu_topology* parameter be set to **auto** for VMs running Windows. If the CPU topology setting is not adjusted, the Windows VM will act like it is a single CPU system and system performance will be slow.

Per-CPU IRQs

Some device drivers use per-CPU IRQs. These IRQs may impact the performance of real-time VMs. They will also prevent the shutting down of sibling cpus in threaded-CPU architectures which also may impact real-time performance. Some per-CPU IRQs can be migrated using the shield system service. Changes to this service can be made by editing the shield configuration file **/etc/sysconfig/shield** and the changes can then be put into effect using **systemctl(1).**

IRQs that cannot be migrated

In some architectures, IRQs are bound to the first CPU of the first core in each socket and cannot be migrated to another CPU. Note that in a threaded-CPU architecture a core may have more than one CPU hence the term ‘first CPU’ is used. These IRQs may affect the performance of real-time VMs running on those CPUs, therefore, it is recommended that the first CPU of the first core in each socket be reserved for the KVM-RT host and not be allocated to a real-time VM.