Red Hat Enterprise Linux 8.0 Beta

Managing, monitoring and updating the kernel

A guide to managing the Linux kernel on Red Hat Enterprise Linux 8

Last Updated: 2018-12-04

Red Hat Enterprise Linux 8.0 Beta Managing, monitoring and updating the kernel

A guide to managing the Linux kernel on Red Hat Enterprise Linux 8

Legal Notice

Abstract

This document provides the users and administrators with necessary information about configuring their workstations on the Linux kernel level. Such adjustments bring performance enhancements, easier troubleshooting or optimized system.

Table of Contents

THIS IS A BETA VERSION!

Thank you for your interest in Red Hat Enterprise Linux 8.0 Beta. Be aware that:

- Beta code should not be used with production data or on production systems.
- Beta does not include a guarantee of support.
- Feedback and bug reports are welcome. Discussions with your account representative, partner contact, and Technical Account Manager (TAM) are also welcome.
- Upgrades to or from a Beta are not supported or recommended.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages, make sure you are viewing the documentation in the Multi-page HTML format. Highlight the part of text that you want to comment on. Then, click the **Add Feedback** pop-up that appears below the highlighted text, and follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
  1. Go to the Bugzilla website.
  2. As the Component, use **Documentation**.
  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click **Submit Bug**.

CHAPTER 1. UPDATING KERNEL WITH YUM

The following sections bring information about the Linux kernel provided and maintained by Red Hat (Red Hat kernel), and how to keep the Red Hat kernel updated. As a consequence, the operating system will have all the latest bug fixes, performance enhancements, and patches ensuring compatibility with new hardware.

1.1. WHAT IS KERNEL

A kernel is a core part of a Linux operating system, which manages the system resources, and provides interface between hardware and software applications. The Red Hat kernel is custom-build and based on the upstream Linux kernel, with focus on stability and compatibility with the latest technologies and hardware.

Before Red Hat releases a new kernel version, the kernel needs to pass a set of rigorous quality assurance tests.

The Red Hat kernels are packaged in the RPM format so that they are easy to upgrade and verify by the **yum** package manager.

WARNING

Custom kernels are **not** supported by Red Hat.

1.2. WHAT IS YUM

This section refers to description of the **yum** *package manager*.

Additional resources

- For more information on **yum** see the relevant sections of *Configuring and managing system administration* guide.

1.3. UPDATING KERNEL

The following procedure describes how to update the kernel using the **yum** package manager.

Procedure

1. To update the kernel, use the following:

```
# yum update kernel
```

   This command updates the kernel along with all dependencies to the latest available version.

2. Reboot your system for the changes to take effect.

IMPORTANT

When upgrading from Red Hat Enterprise Linux 7 to Red Hat Enterprise Linux 8, Red Hat strongly recommends that you reinstall the whole OS. While it is theoretically possible to update all necessary packages, it could easily result in the system being unusable.

1.4. INSTALLING KERNEL

The following procedure describes how to update or install the kernel using the **yum** package manager.

Procedure

- To install a specific kernel version, use the following:

```
# yum install kernel-{version}
```

NOTE

The **yum** package manager always **installs** a new kernel instead of replacing the current one, which could potentially leave your system unbootable.

Additional resources

- For a list of available kernels, refer to this *Red Hat labs page*.
- For a list of release dates of specific kernel versions, see *this article*.

CHAPTER 2. CONFIGURING KERNEL COMMAND LINE PARAMETERS

As a system administrator, you can configure the kernel command line parameters to make sure that they are in place and loaded as soon as possible. Also, certain kernel command line parameters are only adjustable in such a way.

2.1. WHAT ARE BOOT ENTRIES

A boot entry is a collection of options forming together a configuration file, which is usually tied to a particular kernel version. In practice, you have at least as many boot entries as your system has installed kernels. The boot entry configuration file is located in the **/boot/loader/entries/** directory and can look like this:

```
6f9cc9cb7d7845d49698c9537337cedc-4.18.0-5.el8.x86_64.conf
```

The file name above consists of a machine ID stored in the **/etc/machine-id** file, and a kernel version.

The boot entry configuration file comprises, among others, information about kernel version, initial ramdisk image, and the **kernelopts** variable, which contains kernel command line parameters. The contents of a boot entry config can be seen below:

```
title Red Hat Enterprise Linux (4.18.0-5.el8.x86_64) 8.0 (Ootpa)
version 4.18.0-5.el8.x86_64
linux /vmlinuz-4.18.0-5.el8.x86_64
initrd /initramfs-4.18.0-5.el8.x86_64.img
options $kernelopts
id rhel-20181029164945-4.18.0-5.el8.x86_64
grub_users $grub_users
grub_arg --unrestricted
grub_class kernel
```

NOTE

For Red Hat Enterprise Linux 7, the boot entry configuration is defined in **grub.cfg**, and for IBM Z the boot entry configuration is defined in the **zipl.conf** file.

2.2. WHAT ARE KERNEL COMMAND LINE PARAMETERS

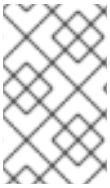
This module explains the concept of kernel command line parameters and their role in the system administration.

The kernel command line parameters, also known as kernel arguments, are used for boot time configuration of:

- The Linux kernel
- The initial RAM disk
- The user space features

The kernel command line parameters are often used to overwrite the default values and for informing the kernel about hardware parameters where the kernel would have problems to obtain such information.

By default, the kernel command line parameters for systems using the GRUB2 bootloader are defined in the **kernelopts** variable of the **/boot/grub2/grubenv** file for all kernel boot entries.



NOTE

For IBM Z, the kernel command line parameters are stored in the boot entry config file because the zipl bootloader does not support environment variables. Therefore **kernelopts** cannot be used.

2.3. WHAT IS GRUBBY

grubby is a utility for manipulating bootloader-specific configuration files.

You can use **grubby** also for changing the default boot entry, and for adding/removing arguments from a GRUB2 menu entry.

2.4. SETTING KERNEL COMMAND LINE PARAMETERS

2.4.1. Changing kernel command line parameters for all boot entries

This procedure describes how to change kernel command line parameters for all boot entries on your system.

Prerequisites

- Introduction to kernel command line parameters.

Procedure

1. To set a new value to the required parameter, use the **grub2-editenv** command as in the following example:

   ```
   # grub2-editenv - set kernelopts=”rd.debug=1 rhgb”
   ```

   This command sets a new value to the global variable **kernelopts** by replacing the old value. As a result, the kernel command line parameter **debug** is set for all boot entries on your system.

2. Reboot your system for the changes to take effect.

Now the boot loader is reconfigured, and the kernel command line parameters that you specified are applied.

2.4.2. Changing kernel command line parameters for a single boot entry

This procedure describes how to change kernel command line parameters for a single boot entry on your system.

Prerequisites

- Introduction to kernel command line parameters.

Procedure

- To add a parameter execute the following:

```
# grubby --update-kernel=/boot/vmlinuz$(uname -r) --args="
<YOUR_ARGUMENT>"
```

- To remove a parameter use the following:

```
# grubby --update-kernel=/boot/vmlinuz$(uname -r) --remove-
args="rd.debug=1 rhgb"
```

NOTE

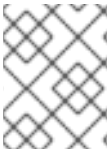
By default, there is the **options** parameter for each kernel boot entry which is set to the **kernelopts** variable. This variable is stored in the **/boot/loader/entries/<YOUR_KERNEL_BOOT_ENTRY>** configuration file. When a kernel command line parameter for a specific boot entry is changed, its **kernelopts** is expanded and the updated kernel command line parameter for the particular boot entry is stored in **/boot/loader/entries/<YOUR_KERNEL_BOOT_ENTRY>**.

CHAPTER 3. CONFIGURING KERNEL PARAMETERS WITH SYSCTL

As a system administrator, you can configure kernel parameters for various reasons, such as improving performance of your system. In order to do so, you use the **sysctl** command to adjust configuration for your workload.

3.1. WHAT ARE KERNEL TUNABLES

Kernel tunables are kernel parameters, which are addressed by the **sysctl** command through the **/proc/sys/** interface. The parameters are adjustable while the system is running. There is no need to reboot or recompile the kernel for the changes to take effect.



NOTE

Not all kernel parameters are under control of **sysctl**. Certain hardware specific options need to be set through the kernel command line parameters.

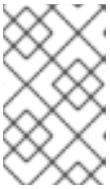
3.2. CONFIGURING KERNEL TUNABLES WITH SYSCTL

The following procedure describes how to use the **sysctl** command to list, and set kernel tunable parameters. The command is also able to filter the list or to set the tunables temporarily or permanently.

Procedure

1. To list all parameters, use the following:

```
# sysctl -a
```



NOTE

 # sysctl -a displays kernel parameters, which can be adjusted both at runtime and at boot time as well. It is up to the user to identify the tunables they wish to configure.

2. To configure the tunable temporarily, use the command as in the following example:

```
# sysctl vm.swappiness=20
```

The sample command above changes the tunable value while the system is running. The changes take effect immediately, without a need for restart.

NOTE

The changes return back to default after your system reboots.

CHAPTER 4. INSTALLING AND CONFIGURING KDUMP

4.1. WHAT IS KDUMP

Kdump is a kernel crash dumping mechanism that enables you to save the contents of the system's memory for later analysis. It relies on the **kexec** system call, which can be used to boot a Linux kernel from the context of another kernel, bypass BIOS, and preserve the contents of the first kernel's memory that would otherwise be lost.

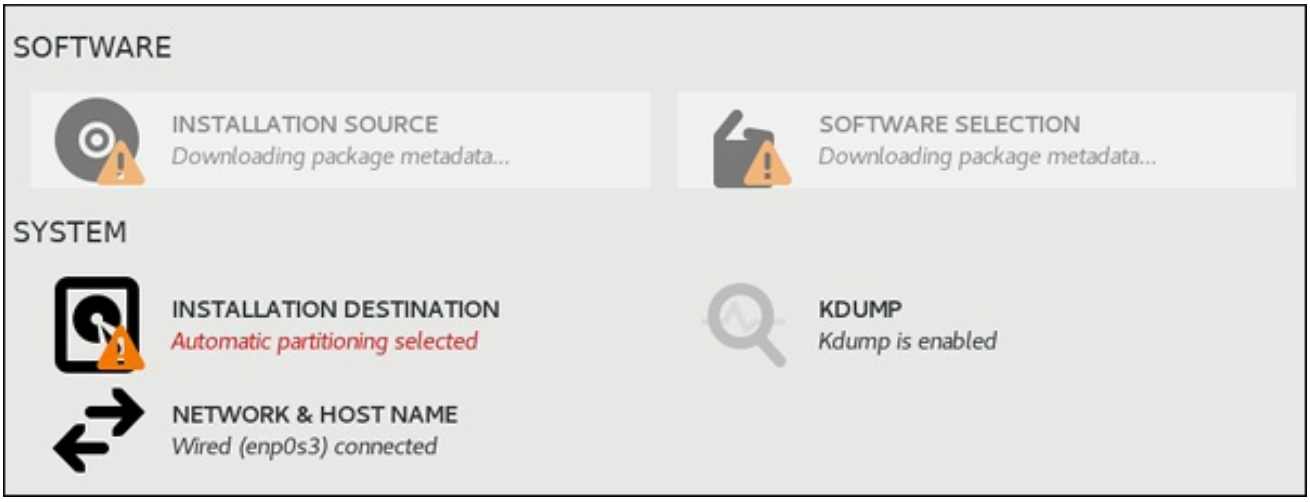
In case of a system crash, **kdump** uses **kexec** to boot into a second kernel (a *capture kernel*). This second kernel resides in a reserved part of the system memory that is inaccessible to the first kernel. The second kernel then captures the contents of the crashed kernel's memory (a *crash dump* or *vmcore*) and saves it.

IMPORTANT

A kernel crash dump can be the only information available in the event of a failure. Therefore, ensuring that kdump can be generated is extremely important in mission-critical environments. Red Hat advise that system administrators regularly update and test **kexec-tools** in your normal kernel update cycle. This is especially important when new kernel features are implemented.

4.2. INSTALLING KDUMP

In many cases, the **kdump** service is installed and activated by default on the new Red Hat Enterprise Linux installations. The **Anaconda** installer provides a screen for **kdump** configuration when performing an interactive installation using the graphical or text interface. The installer screen is titled **Kdump** and is available from the main **Installation Summary** screen, and only allows limited configuration - you can only select whether **kdump** is enabled and how much memory is reserved.



Some installation options, such as custom Kickstart installations, in some cases do not install or enable **kdump** by default. If this is the case on your system, follow the procedure below to install **kdump**.

Prerequisites

- An active RHEL subscription.
- A repository containing the **kexec-tools** package for your system CPU architecture.

- Fulfilled **kdump** requirements.

Procedure

1. Execute the following command to check whether **kdump** is installed on your system:

```
$ rpm -q kexec-tools
```

   Output if the package is installed: **$ kexec-tools-2.0.17-11.el8.x86_64**

   Output if the package is not installed: **$ package kexec-tools is not installed**

2. Install **kdump** and other necessary packages by:

```
# yum install kexec-tools
```

IMPORTANT

Starting with Red Hat Enterprise Linux 7.4 the **Intel IOMMU** driver is supported with **kdump**. When running kernels from version 7.3 or earlier, it is advised that **Intel IOMMU** support is disabled, otherwise kdump kernel is likely to become unresponsive.

Additional resources

- Information about memory requirements for **kdump** is available in Section 4.5.1, “Memory requirements for kdump”.

4.3. CONFIGURING KDUMP ON THE COMMAND LINE

4.3.1. Configuring kdump memory usage

The memory reserved for the **kdump** feature is always reserved during the system boot. The amount of memory is specified in the system’s Grand Unified Bootloader (GRUB) 2 configuration. The procedure below describes how to configure the memory reserved for **kdump** through the command line.

Prerequisites

- Fulfilled **kdump** requirements.

Procedure

1. Edit the **/etc/default/grub** file using the root permissions.

2. Set the **crashkernel=** option to the required value.
   For example, to reserve 128 MB of memory, use the following:

```
crashkernel=128M
```

   Alternatively, you can set the amount of reserved memory to a variable depending on the total amount of installed memory. The syntax for memory reservation into a variable is **crashkernel=*<range1>*:*<size1>*,*<range2>*:*<size2>***. For example:

```
crashkernel=512M-2G:64M,2G-:128M
```

The above example reserves 64 MB of memory if the total amount of system memory is 512 MB or higher and lower than 2 GB. If the total amount of memory is more than 2 GB, 128 MB is reserved for **kdump** instead.

- Offset the reserved memory.
  Some systems require to reserve memory with a certain fixed offset since crashkernel reservation is very early, and it wants to reserve some area for special usage. If the offset is set, the reserved memory begins there. To offset the reserved memory, use the following syntax:

  ```
  crashkernel=128M@16M
  ```

  The example above means that **kdump** reserves 128 MB of memory starting at 16 MB (physical address 0x01000000). If the offset parameter is set to 0 or omitted entirely, **kdump** offsets the reserved memory automatically. This syntax can also be used when setting a variable memory reservation as described above; in this case, the offset is always specified last (for example, **crashkernel=512M-2G:64M,2G-:128M@16M**).

3. Use the following command to update the GRUB2 configuration file:

   ```
   # grub2-mkconfig -o /boot/grub2/grub.cfg
   ```

Additional resources

- The **crashkernel=** option can be defined in multiple ways. The **auto** value enables automatic configuration of reserved memory based on the total amount of memory in the system, following the guidelines described in Section 4.5.1, “Memory requirements for kdump”.

4.3.2. Configuring the kdump target

When a kernel crash is captured, the core dump can be either stored as a file in a local file system, written directly to a device, or sent over a network using the **NFS** (Network File System) or **SSH** (Secure Shell) protocol. Only one of these options can be set at a time, and the default behavior is to store the **vmcore** file in the **/var/crash/** directory of the local file system.

Prerequisites

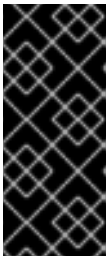
- Fulfilled **kdump** requirements.

Procedure
To change the local directory in which the core dump is to be saved, as **root**, edit the **/etc/kdump.conf** configuration file as described below.

1. Remove the hash sign ("#") from the beginning of the **#path /var/crash** line.

2. Replace the value with the intended directory path. For example:

   ```
   path /usr/local/cores
   ```



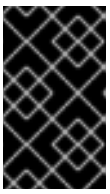
IMPORTANT

In Red Hat Enterprise Linux 8, the directory defined as the kdump target using the **path** directive must exist when the **kdump** systemd service is started - otherwise the service fails. This behavior is different from earlier releases of Red Hat Enterprise Linux, where the directory was being created automatically if it did not exist when starting the service.

To write the file to a different partition, as **root**, edit the **/etc/kdump.conf** configuration file as described below.

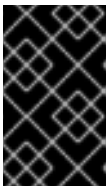
1. Remove the hash sign ("#") from the beginning of the **#ext4** line, depending on your choice.
   - device name (the **#ext4 /dev/vg/lv_kdump** line)
   - file system label (the **#ext4 LABEL=/boot** line)
   - UUID (the **#ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937** line)
2. Change the file system type as well as the device name, label or UUID to the desired values. For example:

```
ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937
```



IMPORTANT

It is recommended to specify storage devices using a **LABEL=** or **UUID=**. Disk device names such as **/dev/sda3** are not guaranteed to be consistent across reboot.



IMPORTANT

When dumping to Direct Access Storage Device (DASD) on IBM Z hardware, it is essential that the dump devices are correctly specified in **/etc/dasd.conf** before proceeding.

To write the dump directly to a device:

1. Remove the hash sign ("#") from the beginning of the **#raw /dev/vg/lv_kdump** line.
2. Replace the value with the intended device name. For example:

```
raw /dev/sdb1
```

To store the dump to a remote machine using the **NFS** protocol:

1. Remove the hash sign ("#") from the beginning of the **#nfs my.server.com:/export/tmp** line.
2. Replace the value with a valid hostname and directory path. For example:

```
nfs penguin.example.com:/export/cores
```

To store the dump to a remote machine using the **SSH** protocol:

1. Remove the hash sign ("#") from the beginning of the **#ssh user@my.server.com** line.
2. Replace the value with a valid username and hostname.
3. Include your **SSH** key in the configuration.
   - Remove the hash sign from the beginning of the **#sshkey /root/.ssh/kdump_id_rsa** line.
   - Change the value to the location of a key valid on the server you are trying to dump to. For example:

     ```
     ssh john@penguin.example.com
     sshkey /root/.ssh/mykey
     ```

Additional resources

- For a complete list of currently supported and unsupported targets sorted by type, see Section 4.5.3, “Supported kdump targets”.

4.3.3. Configuring the core collector

To reduce the size of the **vmcore** dump file, you can specify an external application (a *core collector*) that saves the dump file in a more compact format. Optionally, omit unwanted information. Currently, the only fully supported *core collector* is the **makedumpfile** utility. To enable and configure the *core collector*, follow the procedure below.

Prerequisites

- Fulfilled **kdump** requirements.

Procedure

- As **root**, edit the **/etc/kdump.conf** configuration file and remove the hash sign ("#") from the beginning of the **#core_collector makedumpfile -l --message-level 1 -d 31**.
- Add the **-c** parameter. For example:

  ```
  core_collector makedumpfile -c
  ```

  The command above enables the dump file compression.
- Add the **-d *value*** parameter. For example:

  ```
  core_collector makedumpfile -d 17 -c
  ```

  The command above removes both zero and free pages from the dump. The *value* is a sum of values of pages you want to omit as described in Section 4.5.4, “Supported kdump filtering levels”.

Additional resources

- See the **makedumpfile(8)** man page for a complete list of available options.

4.3.4. Configuring the kdump default failure responses

By default, when **kdump** fails to create a vmcore dump file at the target location specified in Section 4.3.2, “Configuring the kdump target”, the system reboots, and the dump is lost in the process. To change this behavior, follow the procedure below.

Prerequisites

- Fulfilled **kdump** requirements.

Procedure

1. As **root**, remove the hash sign ("#") from the beginning of the **#default shell** line in the **/etc/kdump.conf** configuration file.
2. Replace the value with a desired action as described in Section 4.5.5, “Supported default failure responses”. For example:

```
default poweroff
```

4.3.5. Enabling and disabling the kdump service

To start the **kdump** service at boot time, follow the procedure below.

Prerequisites

- Fulfilled **kdump** requirements.
- All configuration is set up according to your needs.

Procedure

1. To enable the **kdump** service, use the following command:

```
# systemctl enable kdump.service
```

This enables the service for **multi-user.target**.

- To start the service in the current session, use the following command:

```
# systemctl start kdump.service
```

2. To stop the **kdump** service, type the following command:

```
# systemctl stop kdump.service
```

3. To disable the **kdump** service, execute the following command:

```
# systemctl disable kdump.service
```

4.4. CONFIGURING KDUMP IN THE WEB CONSOLE

The following sections provide an overview of how to setup and test the **kdump** configuration through a web console called Cockpit. The console is contained in a default installation of Red Hat

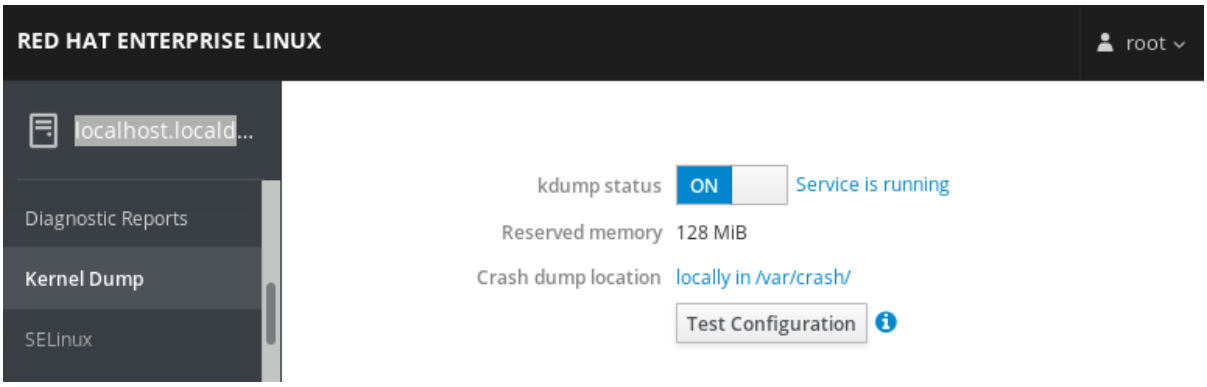
Enterprise Linux 8. Cockpit allows you to enable or disable starting the service at boot time, configure the reserved memory for **kdump**, and conveniently select the *vmcore* saving location in an uncompressed or compressed format.

4.4.1. Configuring kdump memory usage and target location in Cockpit

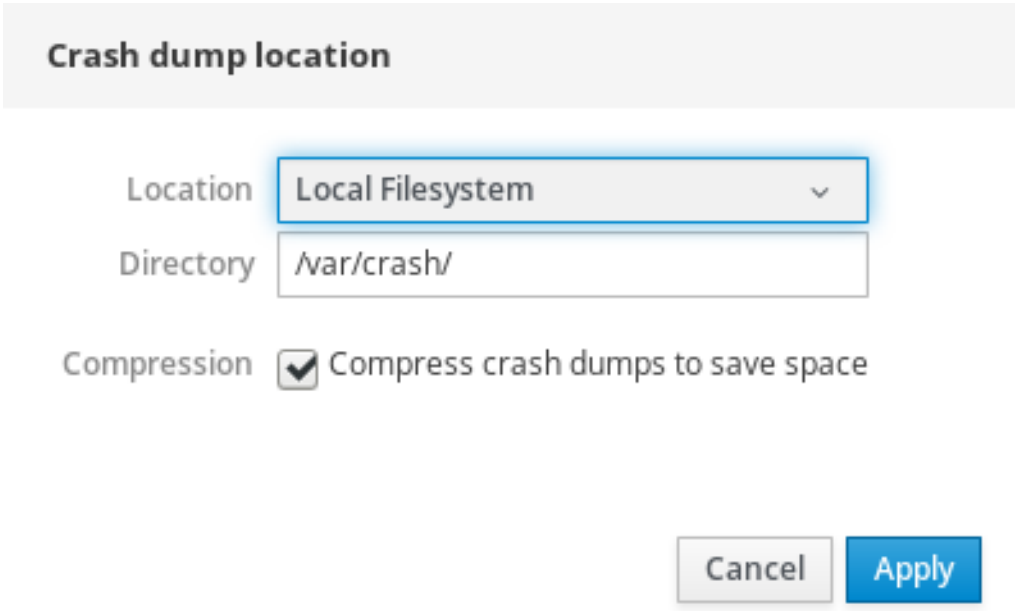
The procedure below shows you how to use the **Kernel Dump** tab in the Cockpit interface to configure the amount of memory that is reserved for the kdump kernel. The procedure also describes how to specify the target location of the vmcore dump file and how to test your configuration.

Procedure

1. Open the **Kernel Dump** tab and start the **kdump** service.
2. Configure the **kdump** memory usage through the command line.
3. Click the link next to the **Crash dump location** option.



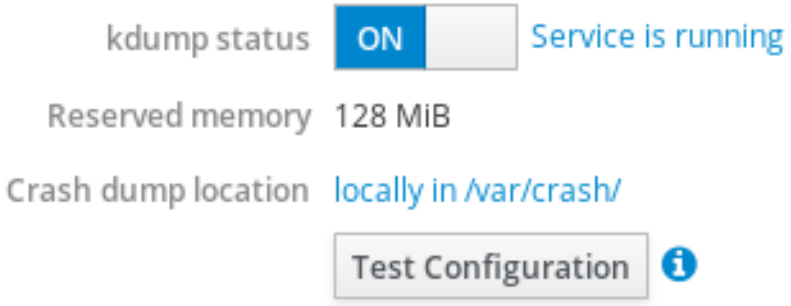
4. Select the **Local Filesystem** option from the drop-down and specify the directory you want to save the dump in.



NOTE

Tick the **Compression** check box to reduce the size of the vmcore dump file.

5. Test your configuration by crashing the kernel.



Additional resources

- For a complete list of currently supported targets for **kdump**, see Supported kdump targets.

4.5. SUPPORTED KDUMP CONFIGURATIONS AND TARGETS

4.5.1. Memory requirements for kdump

In order for **kdump** to be able to capture a kernel crash dump and save it for further analysis, a part of the system memory has to be permanently reserved for the capture kernel. When reserved, this part of the system memory is not available to the main kernel.

The memory requirements vary based on certain system parameters. One of the major factors is the system's hardware architecture. To find out the exact machine architecture (such as Intel 64 and AMD64, also known as x86_64) and print it to standard output, use the following command:

```
$ uname -m
```

The table below contains a list of minimum memory requirements to automatically reserve a memory size for **kdump**. The size changes according to the system's architecture and total available physical memory.

Table 4.1. Minimum Amount of Reserved Memory Required for kdump

Architecture	Available Memory	Minimum Reserved Memory
AMD64 and Intel 64 (**x86_64**)	1 GB to 64 GB	160 MB of RAM.
	64 GB to 1 TB	256 MB of RAM.
	1 TB and more	512 MB of RAM.
64-bit ARM architecture (**arm64**)	2 GB and more	512 MB of RAM.
IBM Power Systems (**ppc64le**)	2 GB to 4 GB	384 MB of RAM.
	4 GB to 16 GB	512 MB of RAM.

Architecture	Available Memory	Minimum Reserved Memory
	16 GB to 64 GB	1 GB of RAM.
	64 GB to 128 GB	2 GB of RAM.
	128 GB and more	4 GB of RAM.
IBM Z (**s390x**)	4 GB to 64 GB	160 MB of RAM.
	64 GB to 1 TB	256 MB of RAM.
	1 TB and more	512 MB of RAM.

On many systems, **kdump** is able to estimate the amount of required memory and reserve it automatically. This behavior is enabled by default, but only works on systems that have more than a certain amount of total available memory, which varies based on the system architecture.

NOTE

The automatic configuration of reserved memory based on the total amount of memory in the system is a best effort estimation. The actual memory may vary due to other factors such as I/O devices.

Additional resources

- For information on how to change memory settings on the command line, see Section 4.3.1, “Configuring kdump memory usage”.
- For instructions on how to set up the amount of reserved memory through the web console, see Section 4.4.1, “Configuring kdump memory usage and target location in Cockpit”.
- For more information about various Red Hat Enterprise Linux technology capabilities and limits, see the *technology capabilities and limits tables*.

4.5.2. Minimum threshold for automatic memory reservation

On some systems, it is possible to allocate memory for **kdump** automatically, either by using the **crashkernel=auto** parameter in the boot loader configuration file, or by enabling this option in the graphical configuration utility. For this automatic reservation to work, however, a certain amount of total memory needs to be available in the system. The amount differs based on the system’s architecture.

The table below lists the thresholds for automatic memory allocation. If the system has less memory than specified in the table, the memory needs to be reserved manually.

Table 4.2. Minimum Amount of Memory Required for Automatic Memory Reservation

Architecture	Required Memory
AMD64 and Intel 64 (**x86_64**)	2 GB

Architecture	Required Memory
IBM Power Systems (**ppc64le**)	2 GB
IBM Z (**s390x**)	4 GB

Additional resources

- For information on how to manually change these settings on the command line, see Section 4.3.1, “Configuring kdump memory usage”.
- For instructions on how to manually change the amount of reserved memory through the web console, see Section 4.4.1, “Configuring kdump memory usage and target location in Cockpit”.

4.5.3. Supported kdump targets

When a kernel crash is captured, the vmcore dump file can be either written directly to a device, stored as a file on a local file system, or sent over a network. The table below contains a complete list of dump targets that are currently supported or explicitly unsupported by **kdump**.

Table 4.3. Supported kdump Targets

Type	Supported Targets	Unsupported Targets
Raw device	All locally attached raw disks and partitions.	
Local file system	**ext2**, **ext3**, **ext4**, and **xfs** file systems on directly attached disk drives, hardware RAID logical drives, LVM devices, and **mdraid** arrays.	Any local file system not explicitly listed as supported in this table, including the **auto** type (automatic file system detection).
Remote directory	Remote directories accessed using the **NFS** or **SSH** protocol over **IPv4**.	Remote directories on the **rootfs** file system accessed using the **NFS** protocol.
Remote directories accessed using the **iSCSI** protocol over both hardware and software initiators.	Remote directories accessed using the **iSCSI** protocol on **be2iscsi** hardware.	Multipath-based storages.
		Remote directories accessed over **IPv6**.
		Remote directories accessed using the **SMB** or **CIFS** protocol.

Type	Supported Targets	Unsupported Targets
		Remote directories accessed using the **FCoE** (*Fibre Channel over Ethernet*) protocol.
		Remote directories accessed using wireless network interfaces.

Additional resources

- For information on how to configure the target type on the command line, see Section 4.3.2, “Configuring the kdump target”.
- For information on how to configure the target through the web console, see Section 4.4.1, “Configuring kdump memory usage and target location in Cockpit”.

4.5.4. Supported kdump filtering levels

To reduce the size of the dump file, **kdump** uses the **makedumpfile** core collector to compress the data and optionally to omit unwanted information. The table below contains a complete list of filtering levels that are currently supported by the **makedumpfile** utility.

Table 4.4. Supported Filtering Levels

Option	Description
1	Zero pages
2	Cache pages
4	Cache private
8	User pages
16	Free pages

NOTE

The **makedumpfile** command supports removal of transparent huge pages and hugetlbfs pages. Consider both these types of hugepages User Pages and remove them using the **-8** level.

Additional resources

- For instructions on how to configure the core collector on the command line, see Section 4.3.3, “Configuring the core collector”.

4.5.5. Supported default failure responses

By default, when **kdump** fails to create a core dump, the operating system reboots. You can, however, configure **kdump** to perform a different operation in case it fails to save the core dump to the primary target. The table below lists all default actions that are currently supported.

Table 4.5. Supported Default Actions

Option	Description
dump_to_rootfs	Attempt to save the core dump to the root file system. This option is especially useful in combination with a network target: if the network target is unreachable, this option configures kdump to save the core dump locally. The system is rebooted afterwards.
reboot	Reboot the system, losing the core dump in the process.
halt	Halt the system, losing the core dump in the process.
poweroff	Power off the system, losing the core dump in the process.
shell	Run a shell session from within the initramfs, allowing the user to record the core dump manually.

Additional resources

- For detailed information on how to set up the default failure responses on the command line, see Section 4.3.4, “Configuring the kdump default failure responses”.

4.5.6. Estimating kdump size

When planning and building your kdump environment, it is necessary to know how much space is required for the dump file before one is produced.

The **makedumpfile --mem-usage** command provides a useful report about excludable pages, and can be used to determine which dump level you want to assign. Run this command when the system is under representative load, otherwise **makedumpfile --mem-usage** returns a smaller value than is expected in your production environment.

```
[root@hostname ~]# makedumpfile --mem-usage /proc/kcore

TYPE            PAGES                   EXCLUDABLE      DESCRIPTION
------------------------------------------------------------------------
ZERO            501635                  yes             Pages filled
with zero
CACHE           51657                   yes             Cache pages
CACHE_PRIVATE   5442                    yes             Cache pages +
private
USER            16301                   yes             User process
pages
```

```
FREE            77738211                yes             Free pages
KERN_DATA       1333192                 no              Dumpable kernel
data
```

IMPORTANT

The **`makedumpfile --mem-usage`** command reports in **pages**. This means that you have to calculate the size of memory in use against the kernel page size. The Red Hat Enterprise Linux kernel is 4 kilobytes for AMD64 and Intel 64 architectures, and 64 kilobytes for IBM POWER architecture.

4.6. TESTING THE KDUMP CONFIGURATION

The following procedure describes how to test that the kernel dump process works and is valid before the machine enters production.

WARNING

The commands below cause the kernel to crash. Use caution when following these steps, and never use them on a production system.

Procedure

1. Reboot the system with **`kdump`** enabled.
2. Make sure that **`kdump`** is running:

```
~]# systemctl is-active kdump
active
```

3. Force the Linux kernel to crash:

```
echo 1 > /proc/sys/kernel/sysrq
echo c > /proc/sysrq-trigger
```

The ***`address-YYYY-MM-DD-HH:MM:SS`*`/vmcore`** file is created at the location you have specified in **`/etc/kdump.conf`** (by default to **`/var/crash/`**).

NOTE

In addition to confirming the validity of the configuration, it is possible to use this action to record how long it takes for a crash dump to complete, while a representative load was running.

4.7. ANALYZING A CORE DUMP

To determine the cause of the system crash, you can use the **crash** utility, which provides an interactive prompt very similar to the GNU Debugger (GDB). This utility allows you to interactively analyze a core

dump created by **kdump**, **netdump**, **diskdump** or **xendump** as well as a running Linux system. Alternatively, you have the option to use the *Kdump Helper or the Kernel Oops Analyzer tool*.

4.7.1. Installing the crash utility

The following procedure describes how to install the **crash** analyzing tool.

Procedure

1. Install the **crash** package:

```
# yum install crash
```

2. Install the **kernel-debuginfo** package:

```
# yum install kernel-debuginfo
```

The package corresponds to your running kernel and provides the data necessary for the dump analysis.

4.7.2. Running and exiting the crash utility

The following procedure describes how to start the crash utility for analyzing the cause of the system crash.

Prerequisites

- Find out which kernel you are currently running (for example **4.18.0-5.el8.x86_64**).

Procedure

1. To start the **crash** utility, two necessary parameters need to be passed to the command:
   - The debug-info (a decompressed vmlinuz image), for example **/usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinux**
   - The actual vmcore file, for example **/var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore**
     The resulting **crash** command then looks like this:

```
# crash /usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinux
/var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore
```

Use the same *<kernel>* version that was captured by **kdump**.

Example 4.1. Running the crash utility

The following example shows analyzing a core dump created on October 6 2018 at 14:05 PM, using the 4.18.0-5.el8.x86_64 kernel.

```
...
WARNING: kernel relocated [202MB]: patching 90160 gdb
minimal_symbol values

      KERNEL: /usr/lib/debug/lib/modules/4.18.0-
```

```
5.el8.x86_64/vmlinux
    DUMPFILE: /var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore
[PARTIAL DUMP]
        CPUS: 2
        DATE: Sat Oct  6 14:05:16 2018
      UPTIME: 01:03:57
LOAD AVERAGE: 0.00, 0.00, 0.00
       TASKS: 586
    NODENAME: localhost.localdomain
     RELEASE: 4.18.0-5.el8.x86_64
     VERSION: #1 SMP Wed Aug 29 11:51:55 UTC 2018
     MACHINE: x86_64  (2904 Mhz)
      MEMORY: 2.9 GB
       PANIC: "sysrq: SysRq : Trigger a crash"
         PID: 10635
     COMMAND: "bash"
        TASK: ffff8d6c84271800  [THREAD_INFO:
ffff8d6c84271800]
         CPU: 1
       STATE: TASK_RUNNING (SYSRQ)

crash>
```

2. To exit the interactive prompt and terminate **crash**, type **exit** or **q**.

Example 4.2. Exiting the crash utility

```
crash> exit
~]#
```

NOTE

The **crash** command is also a great debugging tool that can be used without any parameters to inspect the live, currently running, system. However use it with caution so as not to break your system.

4.7.3. Displaying message buffer, backtrace, and other indicators in the crash utility

The following procedures describe how to use the crash utility and display various indicators, such as a message buffer, a backtrace, a process status, virtual memory information and open files.

Displaying the message buffer

- To display the kernel message buffer, type the **log** command at the interactive prompt as displayed in the example below:

Example 4.3. Displaying the kernel message buffer

```
crash> log
... several lines omitted ...
EIP: 0060:[<c068124f>] EFLAGS: 00010096 CPU: 2
EIP is at sysrq_handle_crash+0xf/0x20
```

```
EAX: 00000063 EBX: 00000063 ECX: c09e1c8c EDX: 00000000
ESI: c0a09ca0 EDI: 00000286 EBP: 00000000 ESP: ef4dbf24
 DS: 007b ES: 007b FS: 00d8 GS: 00e0 SS: 0068
Process bash (pid: 5591, ti=ef4da000 task=f196d560
task.ti=ef4da000)
Stack:
 c068146b c0960891 c0968653 00000003 00000000 00000002 efade5c0
c06814d0
<0> fffffffb c068150f b7776000 f2600c40 c0569ec4 ef4dbf9c 00000002
b7776000
<0> efade5c0 00000002 b7776000 c0569e60 c051de50 ef4dbf9c f196d560
ef4dbfb4
Call Trace:
 [<c068146b>] ? __handle_sysrq+0xfb/0x160
 [<c06814d0>] ? write_sysrq_trigger+0x0/0x50
 [<c068150f>] ? write_sysrq_trigger+0x3f/0x50
 [<c0569ec4>] ? proc_reg_write+0x64/0xa0
 [<c0569e60>] ? proc_reg_write+0x0/0xa0
 [<c051de50>] ? vfs_write+0xa0/0x190
 [<c051e8d1>] ? sys_write+0x41/0x70
 [<c0409adc>] ? syscall_call+0x7/0xb
Code: a0 c0 01 0f b6 41 03 19 d2 f7 d2 83 e2 03 83 e0 cf c1 e2 04
09 d0 88 41 03 f3 c3 90 c7 05 c8 1b 9e c0 01 00 00 00 0f ae f8 89
f6 <c6> 05 00 00 00 00 01 c3 89 f6 8d bc 27 00 00 00 00 8d 50 d0
83
EIP: [<c068124f>] sysrq_handle_crash+0xf/0x20 SS:ESP 0068:ef4dbf24
CR2: 0000000000000000
```

Type **`help log`** for more information on the command usage.

NOTE

The kernel message buffer includes the most essential information about the system crash and, as such, it is always dumped first in to the **`vmcore-dmesg.txt`** file. This is useful when an attempt to get the full **`vmcore`** file failed, for example because of lack of space on the target location. By default, **`vmcore-dmesg.txt`** is located in the **`/var/crash/`** directory.

4.7.3.1. Displaying a backtrace

- To display the kernel stack trace, use the **`bt`** command.

Example 4.4. Displaying the kernel stack trace

```
crash> bt
PID: 5591   TASK: f196d560  CPU: 2   COMMAND: "bash"
 #0 [ef4dbdcc] crash_kexec at c0494922
 #1 [ef4dbe20] oops_end at c080e402
 #2 [ef4dbe34] no_context at c043089d
 #3 [ef4dbe58] bad_area at c0430b26
 #4 [ef4dbe6c] do_page_fault at c080fb9b
 #5 [ef4dbee4] error_code (via page_fault) at c080d809
    EAX: 00000063  EBX: 00000063  ECX: c09e1c8c  EDX: 00000000
EBP: 00000000
```

```
    DS:  007b      ESI: c0a09ca0  ES:  007b      EDI: 00000286
GS:  00e0
    CS:  0060      EIP: c068124f  ERR: ffffffff  EFLAGS: 00010096
 #6 [ef4dbf18] sysrq_handle_crash at c068124f
 #7 [ef4dbf24] __handle_sysrq at c0681469
 #8 [ef4dbf48] write_sysrq_trigger at c068150a
 #9 [ef4dbf54] proc_reg_write at c0569ec2
#10 [ef4dbf74] vfs_write at c051de4e
#11 [ef4dbf94] sys_write at c051e8cc
#12 [ef4dbfb0] system_call at c0409ad5
    EAX: ffffffda  EBX: 00000001  ECX: b7776000  EDX: 00000002
    DS:  007b      ESI: 00000002  ES:  007b      EDI: b7776000
    SS:  007b      ESP: bfcb2088  EBP: bfcb20b4  GS:  0033
    CS:  0073      EIP: 00edc416  ERR: 00000004  EFLAGS: 00000246
```

Type **bt *<pid>*** to display the backtrace of a single process or type **help bt** for more information on **bt** usage.

4.7.3.2. Displaying a process status

- To display the status of processes in the system, use the **ps** command.

Example 4.5. Displaying the status of processes in the system

```
crash> ps
   PID    PPID  CPU   TASK    ST  %MEM     VSZ    RSS  COMM
>     0      0   0  c09dc560  RU   0.0       0      0  [swapper]
>     0      0   1  f7072030  RU   0.0       0      0  [swapper]
      0      0   2  f70a3a90  RU   0.0       0      0  [swapper]
>     0      0   3  f70ac560  RU   0.0       0      0  [swapper]
      1      0   1  f705ba90  IN   0.0    2828   1424  init
... several lines omitted ...
   5566      1   1  f2592560  IN   0.0   12876    784  auditd
   5567      1   2  ef427560  IN   0.0   12876    784  auditd
   5587   5132   0  f196d030  IN   0.0   11064   3184  sshd
>  5591   5587   2  f196d560  RU   0.0    5084   1648  bash
```

Use **ps *<pid>*** to display the status of a single process. Use *help ps* for more information on **ps** usage.

4.7.3.3. Displaying virtual memory information

- To display basic virtual memory information, type the **vm** command at the interactive prompt.

Example 4.6. Displaying virtual memory information of the current context

```
crash> vm
PID: 5591   TASK: f196d560  CPU: 2   COMMAND: "bash"
   MM       PGD      RSS    TOTAL_VM
f19b5900  ef9c6000  1648k    5084k
  VMA       START      END    FLAGS  FILE
f1bb0310    242000    260000 8000875  /lib/ld-2.12.so
f26af0b8    260000    261000 8100871  /lib/ld-2.12.so
```

```
efbc275c    261000    262000 8100873  /lib/ld-2.12.so
efbc2a18    268000    3ed000 8000075  /lib/libc-2.12.so
efbc23d8    3ed000    3ee000 8000070  /lib/libc-2.12.so
efbc2888    3ee000    3f0000 8100071  /lib/libc-2.12.so
efbc2cd4    3f0000    3f1000 8100073  /lib/libc-2.12.so
efbc243c    3f1000    3f4000 100073
efbc28ec    3f6000    3f9000 8000075  /lib/libdl-2.12.so
efbc2568    3f9000    3fa000 8100071  /lib/libdl-2.12.so
efbc2f2c    3fa000    3fb000 8100073  /lib/libdl-2.12.so
f26af888    7e6000    7fc000 8000075  /lib/libtinfo.so.5.7
f26aff2c    7fc000    7ff000 8100073  /lib/libtinfo.so.5.7
efbc211c    d83000    d8f000 8000075  /lib/libnss_files-2.12.so
efbc2504    d8f000    d90000 8100071  /lib/libnss_files-2.12.so
efbc2950    d90000    d91000 8100073  /lib/libnss_files-2.12.so
f26afe00    edc000    edd000 4040075
f1bb0a18   8047000   8118000 8001875  /bin/bash
f1bb01e4   8118000   811d000 8101873  /bin/bash
f1bb0c70   811d000   8122000 100073
f26afae0   9fd9000   9ffa000 100073
... several lines omitted ...
```

Use **vm <*pid*>** to display information on a single process, or use **help vm** for more information on **vm** usage.

4.7.3.4. Displaying open files

- To display information about open files, use the **files** command.

Example 4.7. Displaying information about open files of the current context

```
crash> files
PID: 5591   TASK: f196d560  CPU: 2   COMMAND: "bash"
ROOT: /    CWD: /root
 FD    FILE     DENTRY    INODE    TYPE  PATH
  0  f734f640  eedc2c6c  eecd6048  CHR   /pts/0
  1  efade5c0  eee14090  f00431d4  REG   /proc/sysrq-trigger
  2  f734f640  eedc2c6c  eecd6048  CHR   /pts/0
 10  f734f640  eedc2c6c  eecd6048  CHR   /pts/0
255  f734f640  eedc2c6c  eecd6048  CHR   /pts/0
```

Use **files <*pid*>** to display files opened by only one selected process, or use **help files** for more information on **files** usage.

4.7.4. Using Kernel Oops Analyzer

The Kernel Oops Analyzer is a tool that analyzes the crash dump by comparing the oops messages with known issues in the knowledge base.

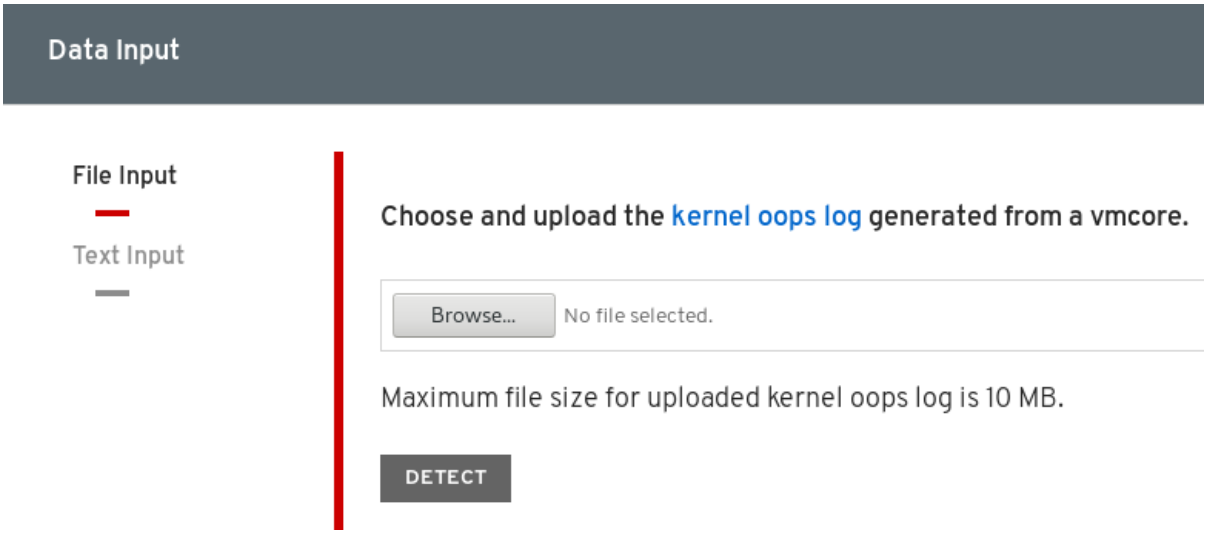
Prerequisites

- Secure an oops message to feed the Kernel Oops Analyzer by following instructions in *Red Hat*

Labs.

Procedure

1. Follow the *Kernel Oops Analyzer* link to access the tool.
2. Browse for the oops message by hitting the **Browse** button.



3. Click the **DETECT** button to compare the oops message based on information from **makedumpfile** against known solutions.

Additional resources

- **kdump.conf**(5) — a manual page for the **/etc/kdump.conf** configuration file containing the full documentation of available options.
- **zipl.conf**(5) — a manual page for the **/etc/zipl.conf** configuration file.
- **zipl**(8) — a manual page for the **zipl** boot loader utility for IBM System z.
- **makedumpfile**(8) — a manual page for the **makedumpfile** core collector.
- **kexec**(8) — a manual page for **kexec**.
- **crash**(8) — a manual page for the **crash** utility.
- **/usr/share/doc/kexec-tools/kexec-kdump-howto.txt** — an overview of the **kdump** and **kexec** installation and usage.
- For more information about the **kexec** and **kdump** configuration see the *Red Hat Knowledgebase article*.
- For more information about the supported **kdump** targets see the *Red Hat Knowledgebase article*.
- *The crash utility homepage*.
- *The GRUB2 boot loader homepage and documentation*.

CHAPTER 5. APPLYING KERNEL PATCHES WITH KPATCH

The **kpatch** live kernel patching solution allows you to patch a running kernel without rebooting or restarting any processes. **kpatch** enables system administrators to apply critical security patches to the kernel immediately, without having to wait for long-running tasks to complete, for users to log off, or for scheduled downtime. It gives more control over uptime without sacrificing security or stability.

WARNING

Some incompatibilities exist between **kpatch** and other kernel subcomponents. Read the Section 5.6, “Limitations of kpatch” carefully before using **kpatch**.

5.1. KPATCH SUPPORT

- Live kernel is supported for customers who have a Premium SLA subscription.
- Live kernel patching is only supported on the active Red Hat Enterprise Linux 8 maintenance stream that is within the current async errata phase. See Red Hat Enterprise Linux Life Cycle for information about current support phases.
- Live kernel patching is not available on the Extended Update Support (EUS) at this time.
- Live kernel patching is not supported on the Red Hat Enterprise Linux for Real Time (RT) kernel.
- Not all issues may be covered under live kernel patching, including hardware enablement.

5.2. ACCESS TO KERNEL PATCHES

Live kernel patching capability is implemented as a kernel module (kmod) that is delivered as an RPM package. The **kpatch** utility is used to install and remove the kernel modules for live kernel patching.

Customers with Premium subscriptions are eligible to request a live kernel patch as part of an accelerated fix solution from Red Hat Support.

Eligible customers who typically used **hotfix** kernels which required a reboot can now request a **kpatch** patch that requires no down time. The kpatch patch will be supported 30 days after the errata that contains the fix is released.

Customers who require accelerated fix options should open a support case Red Hat Customer Portal and discuss appropriate accelerated fix options. For fastest support, include the CVE id or Bug number as well as the precise kernel version(s) to be patched. The kernel version may obtained with **uname -r**.

5.3. SUPPORT FOR THIRD-PARTY LIVE PATCHING

kpatch is the only live kernel patching utility supported by Red Hat with the RPM modules supplied through your Red Hat support contract. Red Hat cannot support third-party live kernel patch.

If you require support for an issue that arises with a third-party live patch, Red Hat recommends that you open a case with the live kernel patching vendor at the outset of any investigation in which a root cause

determination is necessary. This allows the source code to be supplied if the vendor allows, and for their support organization to provide assistance in root cause determination prior to escalating the investigation to Red Hat Support.

For any system running with third-party live kernel patches, Red Hat reserves the right to ask for reproduction with Red Hat shipped and supported software. In the event that this is not possible, we require a similar system and workload be deployed on your test environment without live patches applied, to confirm if the same behavior is observed.

For more information about third-party software support policies, see How does Red Hat Global Support Services handle third-party software, drivers, and/or uncertified hardware/hypervisors or guest operating systems?

5.4. COMPONENTS OF KPATCH

The components of **kpatch** are as follows:

kpatch.service

A **systemd** service required by **multiuser.target** which loads the **kpatch** modules at boot time.

Patch Module

- The delivery mechanism for new kernel code.
- This is another kernel module that is named to match the **kpatch** being applied.
- The patch module contains the compiled code from the latest hotfixes introduced to the kernel.
- The patch modules register with the **livepatch** kernel subsystem and provide information about original functions to be replaced, with corresponding pointers to the replacement functions.

The kpatch Utility

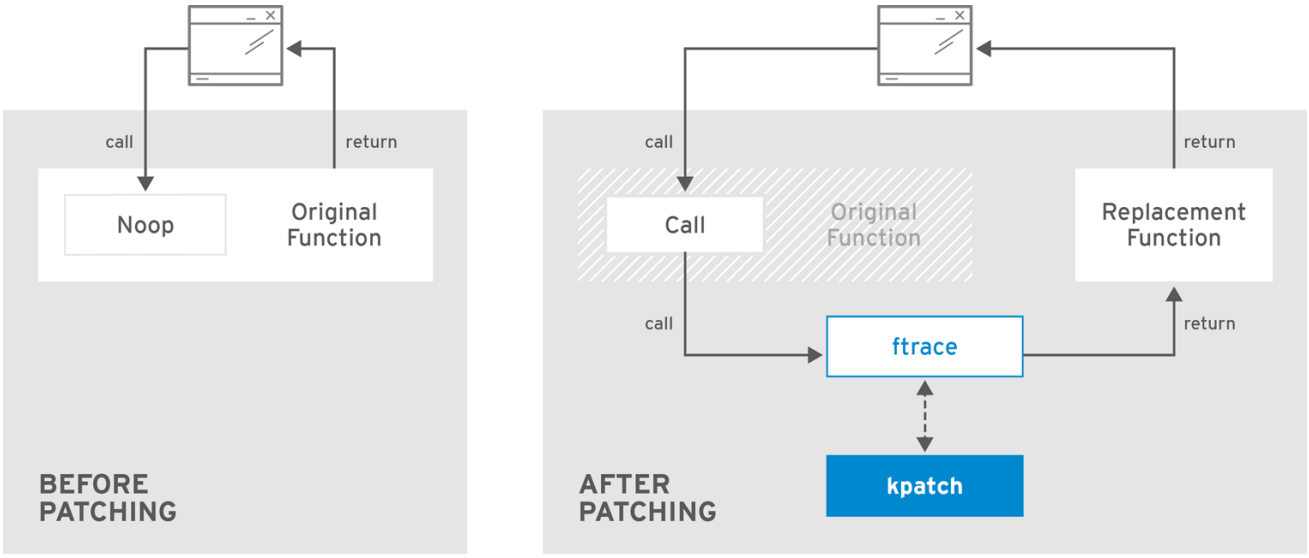
A command-line tool which allows you to manage patch modules.

5.5. HOW KPATCH WORKS

The **kpatch** kernel patching solution uses the **livepatch** kernel subsystem to redirect old functions to new ones. When a live kernel patch is applied to a system, the following things happen:

1. The new compiled code in the module is copied to **/var/lib/kpatch** and registered for re-application to the kernel via **systemd** on next boot.
2. The **kpatch** module is loaded into the running kernel and the new functions are registered to the **ftrace** mechanism with a pointer to the location in memory of the new code.
3. When the kernel accesses the patched function, it is redirected to the **ftrace** mechanism which bypasses the original functions and redirects the kernel to patched version of the function.

Figure 5.1. How kpatch Works



5.6. LIMITATIONS OF KPATCH

- **`kpatch`** is not a general-purpose kernel upgrade mechanism. It is used for applying simple security and bug fix updates when rebooting the system is not immediately possible.
- Do not use the **`SystemTap`** or **`kprobe`** tools during or after loading a patch. The patch could fail to take effect until after the probe has been removed.
- Do not suspend or hibernate the system when using **`kpatch`**. This can result in a patch being temporarily disabled for a small amount of time.

CHAPTER 6. SETTING LIMITS FOR APPLICATIONS

6.1. WHAT ARE CONTROL GROUPS

Linux Control Groups (cgroups) enable limits on the use of system resources, ensuring that an individual process running inside a **cgroup** only utilizes as much resources as has been allowed in the **cgroups** configuration.

Due to easier container-cgroup association, the containers have a much more coherent cgroup view. Control Groups also enable tasks inside the container to have a virtualized view of the cgroup the task belongs to.