Oracle Linux Advanced Administration

Student Guide – Volume II

D84498GC10
Edition 1.0
April 2014
D86118

ORACLE

Author

Craig McBride

Technical Contributors and Reviewers

Michele Dady

Avi Miller

Elena Zannoni

Wim Coekaerts

Al Flournoy

Harald Van Breederode

Joel Goodman

Manish Kapur

Yasar Akthar

Ozgur Yuksel

Antoinette O'Sullivan

Gavin Bowe

Nick Alcock

Dwight Engen

Wayne Lewis

Herbert Van Den Bergh

Tim Hill

Kris Van Hees

John Haxby

Graphic Editor

Maheshwari Krishnamurthy

Editors

Smita Kommini

Daniel Milne

Publishers

Veena Narasimhan

Michael Sebastian Almeida

Jobi Varghese

Contents

Oracle Systems Learning Stream

A New Innovative Continuous Learning Solution

Ideal for supplementing what you learned in the classroom

Oracle Learning Streams

Videos from Oracle Experts

Continuously Updated Content

Connect Live with Instructors

ORACLE UNIVERSITY

Oracle Systems Learning Stream

Keep Your Skills Current through Continuous Learning

Expert Delivered

Access to hundreds of instructional videos delivered by Oracle subject matter experts

Training Across Your Infrastructure

For technical and business professionals looking to regularly broaden and deepen their knowledge

Continuously Refreshed Content

Covers technical, new features, how-to information and more…. on Oracle Hardware, Software, Operating System and Virtualization Solutions

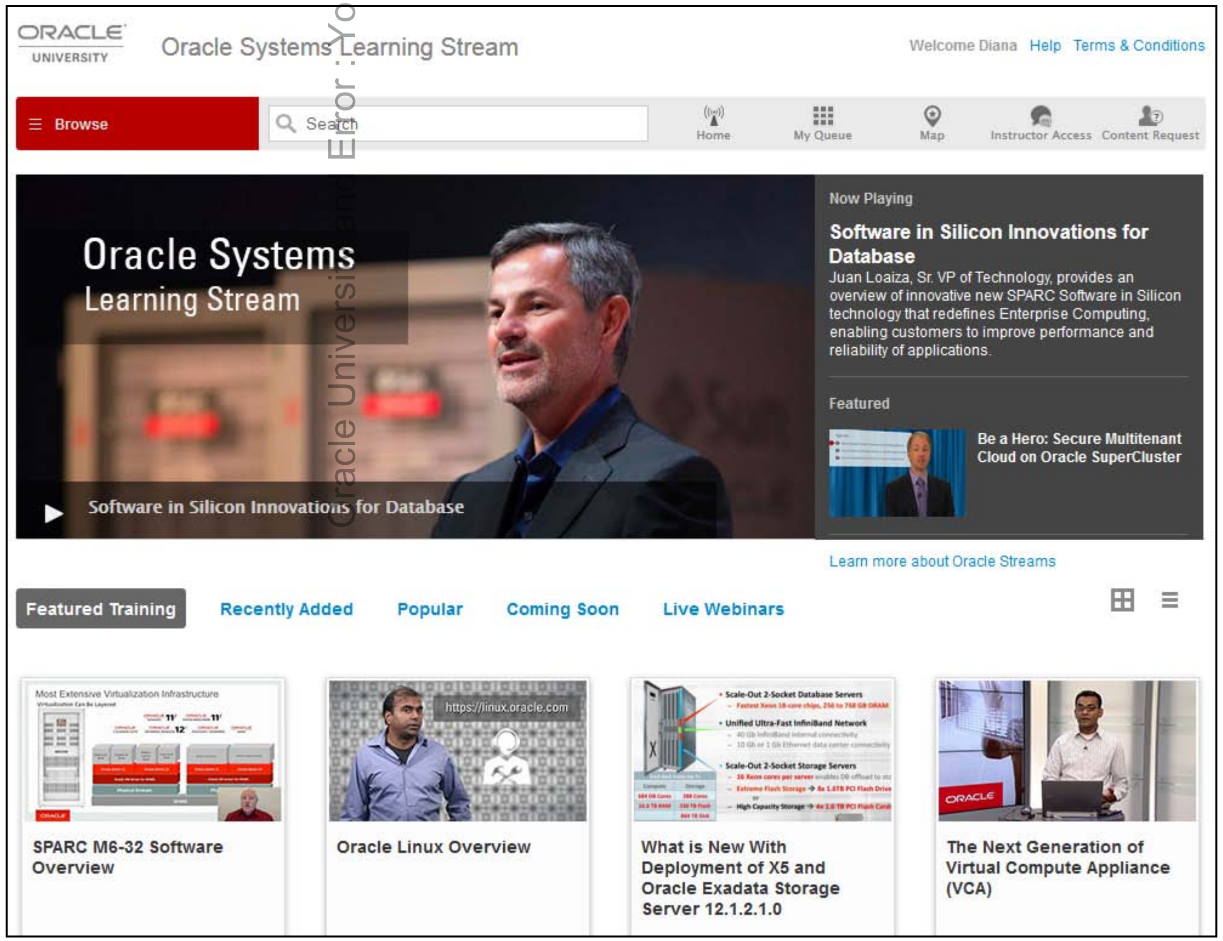
Request Topics that Interest You

Subscription Service

Preview the Oracle Systems Learning Stream

NOW!

education.oracle.com/streams/systems



Software in Silicon Innovations for Database

Now Playing

Software in Silicon Innovations for Database

Juan Loaiza, Sr. VP of Technology, provides an overview of innovative new SPARC Software in Silicon technology that redefines Enterprise Computing, enabling customers to improve performance and reliability of applications.

Featured

Be a Hero: Secure Multitenant Cloud on Oracle SuperCluster

Learn more about Oracle Streams

SPARC M6-32 Software Overview

Oracle Linux Overview

What is New With Deployment of X5 and Oracle Exadata Storage Server 12.1.2.1.0

The Next Generation of Virtual Compute Appliance (VCA)

ORACLE

11

XFS File System

Objectives

After completing this lesson, you should be able to:

- Describe XFS for Oracle Linux
- Create an XFS file system
- Use the `xfs_growfs` utility
- Use the `xfs_admin` utility
- Enable disk quotas on an XFS file system
- Use the `xfs_quota` utility
- Set project quotas
- Use the `xfsdump` and `xfsrestore` utilities
- Use XFS file system maintenance utilities

XFS File System

- XFS is a high-performance journaling file system.
- XFS is available for Oracle Linux x86_64 architecture.
  - Requires UEK R2 or later UEK
- It supports a maximum file size of 16 TB.
- It supports a maximum file system size of 100 TB.
- XFS can be created on a regular disk partition and on a logical volume.
  - XFS is not supported on the root (`/`) or `/boot` file systems.
- XFS supports extended attributes.
  - Used by Access Control Lists (ACL) and SELinux
- XFS supports user, group, and project disk quotas.

The XFS file system is a high-performance journaling file system. XFS in Oracle Linux is available for the x86_64 architecture and requires the Unbreakable Enterprise Kernel Release 2 (`2.6.39`) or UEK Release 3 (`3.8.13`). XFS supports a maximum file system size of 100 TB and a maximum file size of 16 TB. You can create an XFS file system on a regular disk partition and on a logical volume. XFS is not supported for use with the root (`/`) or `/boot` file systems.

The data section of an XFS file system contains the file system metadata (inodes, directories, and indirect blocks) and the user file data. The data section is partitioned into allocation groups, which are virtual storage regions of fixed size. Any files and directories that you create can span multiple allocation groups. Each allocation group manages its own set of inodes and free space independently of other allocation groups to provide both scalability and parallelism of I/O operations.

The XFS journal (or log) can be located internally in the data section of the file system, or externally on a separate device to reduce the number of disk seeks. The journal stores changes to the file system metadata while the file system is running until those changes are written to the data section. XFS journaling guarantees the consistency of the file system following loss of power or a system crash. When mounting a file system after a crash, the journal is read to complete operations that were in progress at the time of the crash.

Creating an XFS File System

- Log in to ULN, register your systems, and subscribe to the `ol6_x86_64_latest` channel.
  - You can also obtain the XFS packages from Public Yum.
- Use the `yum` command to install the `xfsprogs` package:

```
# yum install xfsprogs
```

- Use the `mkfs.xfs` command to create an XFS file system:

```
# mkfs.xfs /dev/xvdb1
```

- By default, the log area is contained within the data section.
- See the `mkfs.xfs` man page for more information.

To create an XFS file system on your system, install the `xfsprogs` software package:

```
    # yum install xfsprogs
```

Use the `mkfs.xfs` command or the `mkfs -t xfs` command to create an XFS file system. The following example creates an XFS file system with an internal log on the `/dev/xvdb1` partition:

```
    # mkfs.xfs /dev/xvdb1
```

The next example creates an XFS file system on `/dev/xvdb1` but places the journal on another device, `/dev/xvdd1`. The `size` option specifies a 10000 block journal:

```
    # mkfs.xfs -l logdev=/dev/xvdd1,size=10000b /dev/xvdb1
```

The next example creates an XFS file system with a stripe-unit size of 32 KB and 6 units per stripe on a logical volume:

```
    # mkfs.xfs -d su=32k,sw=6 /dev/myvolg/myvol
```

XFS uses the stripe-unit size and the number of units per stripe information to align data, inodes, and the journal appropriately for the storage. On LVM and `md` volumes and some hardware RAID configurations, XFS can automatically select the optimal stripe parameters.

The next example includes the output of the `mkfs.xfs` command. The `-f` option forces the overwrite of an existing file system type. The `–L` option sets the file system label to "`XFS`". The `-b size=512` option sets the logical block size to 512 bytes.

```
# mkfs.xfs –f –L XFS –b size=512 /dev/xvdb1
meta-data=/dev/xvdb1   isize=256   agcount=4, agzise=530129 blks
         =             sectsz=512  attr=2, projid32bit=1
         =             crc=0
data     =             bsize=512   blocks=2120516, imaxpct=25
         =             sunit=0     swidth=0 blks
naming   =version 2    bsize=4096  ascii-ci=0
log      =internal log bsize=512   blocks=20480, version=2
         =             sectsz=512  sunit=0 blks, lazy-count=1
realtime =none         extsz=4096  blocks=0, rtextents=0
```

The output shows that an XFS file system has up to three parts: a `data` section, a `log` section (journal), and a `realtime` section. Using the default `mkfs.xfs` options, the `realtime` section is absent, and the `log` area is contained within the `data` section. The `naming` area specifies settings for the file system directory.

The following are some additional options to the `mkfs.xfs` command:

- **`-b <`*`block_size`*`>`:** Each section of the file system is divided into a certain number of blocks. XFS allows you to choose the logical block size for each section of the file system. The physical disk blocks are always 512 bytes. The default value of the logical block size is 4 KB. This is the recommended block size for file systems larger than 100 MB. The minimum logical block is 512 bytes and is recommended for file systems smaller than 100 MB and for file systems with many small files. The maximum block size is the page size of the kernel.
- **`-d <`*`data_section_options`*`>`:** These options specify the location, size, and other parameters of the data section of the file system. The `data` section of the file system is divided into allocation groups to improve the performance of XFS. More allocation groups imply that you can achieve more parallelism when allocating blocks and inodes. Use the `–d agcount=<`*`value`*`>` option to select the number of allocation groups. The default number of allocation groups is 8 when the file system size is between 128 MB and 8 GB. Alternatively you can use the `–d agsize=<`*`value`*`>` option to select the size of allocation groups. The `agcount` and `agsize` parameters are mutually exclusive. The minimum allocation group size is 16 MB; the maximum size is just under 1 TB. Increase the number of allocation groups from the default if there is sufficient memory and a lot of allocation activity. Do not set the number of allocation groups too high, because this can cause the file system to use large amounts of CPU time, especially when the file system is nearly full.
- **`-n <`*`naming_options`*`>`:** These options specify the version and size parameters for the file system directory (or naming area). This allows you to choose a logical block size for the file system directory that is greater than the logical block size of the file system. For example, in a file system with many small files, the file system logical block size could be small (512 bytes) and the logical block size for the file system directory could be large (4 KB). This can improve the performance of directory lookups, because the tree storing the index information has larger blocks.

Refer to the man page for `mkfs.xfs` to view a description of all available options.

`xfs_growfs` Utility

- Use this to increase the size of a mounted XFS file system.
- There must be space available on the underlying device.
  - For example, XFS file system on a logical volume
- The syntax of the `xfs_growfs` command is as follows:
  - `xfs_growfs [options] mount-point`
- Options include:
  - `-d`: Expand the data section to use all available space.
  - `-D <size>`: Specify the size, in number of blocks, to expand the data section of the file system.
  - `-L <size>`: Specify the new size of the log area.
- See the `xfs_growfs` man page for more information.

Use the `xfs_growfs` command to increase the size of an XFS file system. The XFS file system must be mounted and there must be space available on the underlying device. The `xfs_growfs` utility is most often used with logical volumes. The syntax of the `xfs_growfs` command is as follows:

```
    xfs_growfs [options] mount-point
```

The following options are available for the `xfs_growfs` command:

- `-d`: Expand the data section of the file system to the maximum size of the underlying device.
- `-D <size>`: Specify the size to expand the data section of the file system. The `<size>` argument is expressed in the number of file system blocks.
- `-L <size>`: Specify the new size of the log area. This does not expand the size, but specifies the new size of the log area. Therefore, this option can be used to shrink the size of the log area. You cannot shrink the size of the data section of the file system.
- `-m <maxpct>`: Specify the new value for the maximum percentage of space in the file system that can be allocated as inodes. With the `mkfs.xfs` command, this option is specified with the `–i maxpct=<value>` option.

For more information, see the `xfs_growfs` manual page.

`xfs_admin` Utility

- Use this to change and view the parameters of an XFS file system.
  - Unmount the XFS file system before changing parameters with `xfs_admin`.
- You can change the file system label and the UUID:
  - File system label (`-L <new_label`*)*
  - File system UUID (`-U <new_UUID`)
- You can enable or disable XFS lazy counters.
  - To enable lazy counters (`-c 1`)
  - To disable lazy counters (`-c 0`)
- XFS enables lazy counters by default.
- Lazy counters improve performance by not requiring superblock update when other counters are changed.

Use the `xfs_admin` command to change the parameters of an XFS file system. You can also use the `xfs_admin` command to view the file system label and UUID:

```
# xfs_admin -lu /dev/xvdb1
label = "XFS"
UUID = ...
```

You must unmount the file system before changing parameters with the `xfs_admin` command. With the file system unmounted, you can change the following parameters:

- `-L <`*`label`*`>`: Use this option to change the file system label.
- `-U <`*`UUID`*`>`: Use this option to change the file system UUID.

You can also use the `xfs_admin` command to enable or disable lazy counters. With lazy counters enabled, the superblock is not modified or logged when changes are made to the free-space and inode counters. Information is stored in other parts of the file system to maintain the counter values. This provides significant performance improvements in some configurations. Enabling and disabling lazy counters is time-consuming on large file systems because the entire file system needs to be scanned. To enable and disable lazy counters:

- `-c 1`: Enables lazy counters
- `-c 0`: Disables lazy counters

For more information, see the `xfs_admin` manual page.

Enabling Disk Quotas on an XFS File System

- XFS supports quotas by user, by group, and by project.
  - Project quotas set limits on directory hierarchies.
- Limit disk space (blocks) and/or number of files (inodes).
  - Hard limits and soft limits on blocks and inodes
- Enable quotas by using XFS file system mount options:
  - `quota|uquota|usrquota`: Enable user quotas and enforce usage limits.
  - `gquota|grpquota`: Enable group quotas and enforce usage limits.
  - `pquota|prjquota`: Enable project quotas and enforce usage limits.
  - `uqnoenforce|gqnoenforce|pqnoenforce`: Enable quotas but do not enforce usage limits.

XFS supports disk quotas by user, by group, and by project. Project disk quotas allow you to limit the amount of disk space on individual directory hierarchies. You can configure both hard and soft limits on the number of disk blocks (or disk space), and the number of inodes, which limit the number of files a user can create. Quotas do not apply to the `root` user.

You must first enable quotas for users, groups, and/or projects by using a mount option when mounting for the XFS file system. After enabling quotas, use the `xfs_quota` command to set limits, to view quota information.

Enabling Quotas

To enable quotas for users on an XFS file system, include the `quota` option in the `/etc/fstab` entry for the file system, or mount the file system with the `quota` option:

```
# mount -o quota /dev/xvdb1 /xfs
```

To enable quotas for groups, include the `gquota` option in the `/etc/fstab` entry for the file system, or mount the file system with the `gquota` option:

```
# mount -o gquota /dev/xvdb1 /xfs
```

To enable quotas for projects, include the `prjquota` option in the `/etc/fstab` entry for the file system, or mount the file system with the `prjquota` option:

```
# mount -o prjquota /dev/xvdb1 /xfs
```

Alternatively, you can include the quota mount options in the `/etc/fstab` file. The following example shows entries in the `/etc/fstab` file to enable quotas for users, groups, and projects, respectively, on an XFS file system. These examples also mount the file system with read-write permissions:

```
/dev/xvdb1    /xfs    xfs    rw,quota    0  0
/dev/xvdb1    /xfs    xfs    rw,gquota    0  0
/dev/xvdb1    /xfs    xfs    rw,prjquota    0  0
```

XFS Quota Mount Options

Other “quota” mount options for XFS file systems are available. The following is a complete list of mount options to enable user quotas on XFS file systems:

- `quota|uquota|usrquota`: Enable user quotas and enforce usage limits.
- `uqnoenforce`: Enable user quotas. Report usage but do not enforce usage limits.

Group quota mount options include the following:

- `gquota|grpquota`: Enable group quotas and enforce usage limits.
- `gqnoenforce`: Enable group quotas. Report usage but do not enforce usage limits.

Project quota mount options include the following:

- `pquota|prjquota`: Enable project quotas and enforce usage limits.
- `pqnoenforce`: Enable project quotas. Report usage but do not enforce usage limits.

Report Quota State Information

You can use the following `xfs_quota` command to report the overall quota state information:

```
# xfs_quota -x -c state
User quota state on /xfs (/dev/xvdb1)
  Accounting: ON
  Enforcement: ON
  Inode: #37 (1 blocks, 1 extents)
Group quota state on /xfs (/dev/xvdb1)
  Accounting: OFF
  Enforcement: OFF
  Inode: N/A
Project quota state on /xfs (/dev/xvdb1)
Accounting: OFF
  Enforcement: OFF
  Inode: N/A
Blocks grace time: [7 days 00:00:30]
Inodes grace time: [7 days 00:00:30]
Realtime Blocks grace time: [7 days 00:00:30]
```

This command reports whether user, group, and project disk quota accounting is enabled and whether limits are being enforced. The grace period for blocks and inodes is also reported. The timer for the grace period is enabled whenever the soft limit is exceeded. If soft limits continue to be exceeded after the grace period expires, no more disk space or inodes are allocated.

`xfs_quota` Utility

- Use to report file system quota information and perform quota management operations on XFS file systems.
  – Set block and inode limits.
  – Enable or disable quota enforcement.
  – Modify the quota enforcement timeout information.
- It includes an interactive interface:

```
# xfs_quota
xfs_quota>
```

- Include `-x` and `-c` options to modify and report quota information from the command line:

```
# xfs_quota -x -c 'limit -u bsoft=5m bhard=6m john' /xfs
# xfs_quota -x -c state
```

After enabling quotas through the use of XFS file system mount options, use the `xfs_quota` command to report file system quota information, set block and inode limits, enable or disable quota enforcement, modify the quota enforcement timeout information (grace period), and perform other quota management operations on XFS file systems.

The `xfs_quota` utility provides a number of user and administrator subcommands. These subcommands can be issued in interactive mode or included as arguments to the `xfs_quota` command. Enter `xfs_quota` without any options or arguments to enter interactive mode. An `xfs_quota>` prompt appears in interactive mode. Enter `help` or `?` to view the available subcommands. You can also enter `help` *`commandname`* to display additional information on a specific subcommand.

```
# xfs_quota
xfs_quota>
```

When including `xfs_quota` subcommands from the command line, use the `-c` *`<command>`* option. Any modifications to the quota system from the command line also require the `-x` option (enable expert mode). The following example uses the `limit` subcommand to set a soft limit of 5 MB, and a hard limit of 6 MB on the XFS file system for user `john`:

```
# xfs_quota -x -c 'limit -u bsoft=5m bhard=6m john' /xfs
```

The following example sets a soft limit of 100 inodes and a hard limit of 150 inodes for the `students` group:

```
# xfs_quota -x -c ‘limit -g isoft=100 ihard=150 students’ /xfs
```

Displaying Quota Information

Use the `xfs_quota` command to display information about disk quotas. To list all paths with devices and identifiers:

```
# xfs_quota -x -c print
Filesystem     Pathname
/xfs           /dev/xvdb1 (uquota)
```

To report file system usage for blocks (`-b`) and inodes (`-i`):

```
# xfs_quota -x -c ‘free -hb’
Filesystem  Size   Used    Avail Use%  Pathname
/dev/xvdb1  1.0G  12.4M  1013.1M   1%  /xfs
# xfs_quota -x -c ‘free -hi’
Filesystem  Inodes  Used   Free  Use%  Pathname
/dev/xvdb1    1.1m     7   1.1m    0%  /xfs
```

To report file system quota information:

```
# xfs_quota -x -c report
User quota on /xfs (/dev/xvdb1)
                       Blocks
User ID     Used    Soft    Hard    Warn/Grace
----------  ---------------------------------------
root        8513       0       0     00 [--------]
oracle         0    4096    5120     00 [--------]
```

To report quota information in human-readable form on `/xfs`:

```
# xfs_quota -x -c ‘report -h’ /xfs
```

For more information, see the `xfs_quota` manual page.

Setting Project Quotas

- XFS allows you to set quotas on directory hierarchies.
- Project quotas are initially enabled by using a mount option:
  - `pquota` or `prjquota` or `pqnoenforce`
- Associate a unique project ID with an XFS directory hierarchy in the `/etc/projects` file. Example:
  - `50:/xfs`
- Associate a project name to the project ID in the `/etc/projid` file. Example:
  - `test:50`
- Use the project name when setting limits. Example:

```
# xfs_quota -x -c ‘limit -p bsoft=5m bhard=6m test’ /xfs
```

XFS allows you to set quotas on individual directory hierarchies. You can create an entry in the `/etc/projects` file that associates the XFS file system directory hierarchy with a unique project ID. For example, the following entry in `/etc/projects` associates a project ID of `50` with the `/xfs` directory:

```
50:/xfs
```

You can optionally use the `/etc/projid` file to associate a project name to a project ID. For example, the following entry in `/etc/projid` associates the project name `test` with project ID `50`:

```
test:50
```

After defining a project, use the `xfs_quota` command to set limits. The following example sets a soft limit of 5 MB and a hard limit of 6 MB for the `test` project:

```
# xfs_quota -x -c ‘limit -p bsoft=5m bhard=6m test’ /xfs
```

Defining projects in the `/etc/projects` file and the `/etc/projid` file is optional. Instead of defining projects in these files, you can use the `-p` option to the `xfs_quota` command to specify a project root directory for each operation. The next page shows a sample sequence of commands that use `/etc/projects` and `/etc/projid` to set project quotas and a sequence of commands that accomplishes the same function without the use of these files.

The following sequence of commands enables project quota on an XFS file system. This example restricts the `/var` directory to 5 GB of space:

```
# mount –o prjquota /dev/xvda3 /var
# echo 60:/var >> /etc/projects
# echo vardir:60 >> /etc/projid
# xfs_quota –x –c ‘project –s vardir’ /var
# xfs_quota –x –c ‘limit –p bhard=5g vardir’ /var
```

The following sequence of commands also restricts the `/var` directory to 5 GB of space without using the `/etc/projects` file and the `/etc/projid` file:

```
# mount –o prjquota /dev/xvda3 /var
# xfs_quota –x –c ‘project –s –p /var 60’ /var
# xfs_quota –x –c ‘limit –p bhard=5g 60’ /var
```

For more information, see the `projects`, `projid`, and `xfs_quota` manual pages.

Backing up and Restoring XFS File Systems

- Use `xfsdump` to back up an XFS file system.
  - You can back up entire XFS file systems or selected files and directories from an XFS file system.
- Use `xfsrestore` to restore files to an XFS file system.
  - You can restore entire XFS file systems or selected files and directories to an XFS file system.
- Install the `xfsdump` software package:

```
# yum install xfsdump
```

- Example of using `xfsdump`:

```
# xfsdump -l 0 -f /dev/st0 /xfs
```

- Example of using `xfsrestore`:

```
# xfsrestore -f /usr/tmp/backup /xfs
```

Use the `xfsdump` and `xfsrestore` utilities to back up and restore files in an XFS file system. You can back up files to directly attached tape drives or hard drives, or to remote drives that are accessible over the network. You can back up an entire XFS file system, only the files that have changed since a previous backup, or selected directories or files.

You can restore all files from a full or incremental backup, or selected files and directories. You can restore data to its original location or to another location within an XFS file system. The `xfsrestore` utility can also be run interactively, allowing you to select files that you want to restore.

To use `xfsdump` and `xfsrestore`, install the `xfsdump` software package:

```
    # yum install xfsdump
```

Using `xfsdump`

Use the `-l` *`<level>`* option to specify a full or incremental backup. Level 0 is a full backup of an entire XFS file system. Levels `1-9` are incremental backups that back up all files that have changed since a backup with a lower level number. The following example performs a level 0 backup of the XFS file system mounted on `/xfs` to a local SCSI tape device, `/dev/st0`. The `-L <session label>` option allows you to assign a label to the backup.

```
    # xfsdump -l 0 -L "Level 0 backup of /xfs" -f /dev/st0 /xfs
```

Backups can span multiple tape media if necessary. If the end of the tape media is reached before the backup is complete, `xfsdump` prompts you to insert additional media. Multiple backups can also be stored on the same media. The tape is automatically advanced to the end of any existing backups before beginning a new backup.

The following example performs a level 1 backup to a tape device attached to a remote system (`host01`). Use a colon between the remote host name (or IP address) and the tape device.

```
# xfsdump -l 1 -f host01:/dev/st0 /xfs
```

You can also use `xfsdump` to back up data to a file instead of a tape device. The following example performs a full backup (level 0) of the XFS file system mounted on `/xfs` to a local file, `/usr/tmp/full_Monday`. Note that if a level is not specified, a full backup is performed.

```
# xfsdump -f /usr/tmp/full_Monday /xfs
```

Use the `-s` option to back up specific files or directories in an XFS file system. The following example backs up `file` and `directory` to a file on a remote host, `host01:/usr/tmp/back`. Both `file` and `directory` are located in the XFS file system mounted on `/xfs`.

```
# xfsdump -f host01:/usr/tmp/back -s file -s directory /xfs
```

Examining `xfsdump` Inventory

The `xfsdump` utility keeps an inventory in the `/var/lib/xfsdump` directory of all backups. You can examine the inventory contents by using the `-I` option.

```
# xfsdump -I
```

The inventory records are in sequential order and are indented for readability and to emphasize the hierarchical nature of the `xfsdump` information.

Using `xfsrestore`

The following example restores an `xfsdump` from a SCSI tape device to an XFS file system mounted on `/xfs`.

```
# xfsrestore -f /dev/st0 /xfs
```

The following example restores the contents of an `xfsdump` that was written to the `/usr/tmp/backup` file to the `/xfs` directory.

```
# xfsrestore -f /usr/tmp/backup /xfs
```

You can perform cumulative restores from tape media that contains full (level 0) and incremental backups. Contents of the level 0 `xfsdump` are restored first, then contents are restored from the next higher level, and so forth until all incremental backups are restored. Use the `-r` option to perform a cumulative restore.

The following example performs a cumulative restore from `xfsdump` backups on a SCSI tape device to an XFS file system mounted on `/xfs_restore`.

```
# xfsrestore -f /dev/st0 -r /xfs_restore
```

A cumulative restore creates an `xfsrestorehousekeepingdir` directory in the directory that is restored. Files in this directory pass information from one execution of `xfsrestore` to the next. This directory can be removed after the cumulative restore is complete.

For more information, see the `xfsdump` and `xfsrestore` manual pages.

XFS File System Maintenance

- `xfs_fsr`: Improve performance of an XFS file system by re-organizing and improving the layout of the file extents.
- `xfs_check`: Check for file system corruption or consistency problems. This utility is deprecated; run `xfs_repair -n` instead.
- `xfs_repair`: Repair a corrupted or damaged XFS file system. Unmount the file system before running this command.
- `xfs_db`: Debug an XFS file system. This utility provides a command set that allows you to perform scans on the file system and to navigate and display its data structures.

Additional XFS utilities are available to perform file system maintenance. These utilities include the following:

- `xfs_fsr`: XFS is an extent-based file system. The `xfs_fsr` utility reorganizes and improves the layout of the file extents, which improves overall performance. Run this command on a mounted XFS file system or on individual files in the file system.
- `xfs_check`: Check for file system corruption or consistency problems. Run this on an unmounted XFS file system, using the device name as an argument. This utility is deprecated and scheduled for removal from the operating system distribution in June 2014. Use the `xfs_repair -n` command instead.
- `xfs_repair`: Repair a corrupted or damaged XFS file system. Unmount the file system before running this command. If the file system cannot be repaired, restore files from a backup with `xfsrestore`.
- `xfs_db`: Debug an XFS file system. This utility provides a command set that allows you to perform scans on the file system and to navigate and display its data structures.

For more information, see the `xfs_fsr`, `xfs_check`, `xfs_repair`, and `xfs_db` manual pages. The `xfs_db` utility also provides a `help` command.

Quiz

Which of the following statements are true?

a. XFS in Oracle Linux is available for the x86_64 architecture.

b. XFS in Oracle Linux requires the Unbreakable Enterprise Kernel Release 3 (3.8.13).

c. XFS in Oracle Linux supports a maximum file system size of 100 TB and a maximum file size of 100 TB.

d. XFS is supported for use with the root (`/`) and `/boot` file systems.

e. All of the above.

Quiz

Which of the following statements are true?

a. XFS supports quotas by user, by group, and by project.

b. Disk quotas are enabled by using XFS file system mount options.

c. Use the `xfs_admin` command to set limits on disk space and number of files.

d. Use the `xfs_quota` command to report file system quota information.

e. All of the above.

Quiz

Which of the following statements are true?

a. The `xfsdump` and `xfsrestore` utilities can be used to access remote storage devices.
b. The `xfsdump` and `xfsrestore` utilities support full and incremental backups and restores.
c. The `xfsdump` and `xfsrestore` utilities allow backups and restores of individual files on an XFS file system.
d. All of the above.

Summary

In this lesson, you should have learned how to:

- Describe XFS for Oracle Linux
- Create an XFS file system
- Use the `xfs_growfs` utility
- Use the `xfs_admin` utility
- Enable disk quotas on an XFS file system
- Use the `xfs_quota` utility
- Set project quotas
- Use the `xfsdump` and `xfsrestore` utilities
- Use XFS file system maintenance utilities

Practice 11: Overview

The practices for this lesson cover the following:

- Creating an XFS file system
- Setting disk quotas on an XFS file system
- Backing up and restoring an XFS file system

These practices assume that you completed Practice 8-1, during which you created a 1 GB partition on `/dev/xvdb`.

If you did not create the `/dev/xvdb1` partition in Practice 8-1, you are instructed to do so in Practice 11-1.

If you did create the `/dev/xvdb1` partition in Practice 8-1, you can skip the step to create the partition in Practice 11-1.

12

Btrfs File System

Objectives

After completing this lesson, you should be able to:

- Describe the features of the Btrfs file system
- Create a Btrfs file system
- Create Btrfs subvolumes and snapshots
- Take a snapshot of a file in a Btrfs subvolume
- Mount Btrfs subvolumes and snapshots
- Defragment and resize a Btrfs file system
- Add and remove devices in a Btrfs file system
- Check and repair the integrity of a Btrfs file system
- Convert ext file systems to Btrfs
- Create a Btrfs root file system by installing Oracle Linux from the UEK Boot ISO

Btrfs: Introduction

- Jointly developed by a number of companies
- Oracle Linux 6.3 with UEK2: first release with Btrfs
  - Alternative boot ISO media uses Btrfs as the root file system.
- Extent-based file storage, 16 EB maximum file size
- All data and metadata written via copy-on-write
- CRCs for all metadata and data
- Readable and writable snapshots
- Integrated volume management and RAID capabilities
- Online resizing and defragmentation
- Transparent compression
- Efficient storage for small files
- SSD optimizations and TRIM support

Btrfs is an open-source, general-purpose file system for Linux. The name derives from the use of B-trees to store internal file system structures. Different names are used for the file system, including "Butter F S" and "B-tree F S." Development of Btrfs began at Oracle in 2007, and now a number of companies (including Red Hat, Fujitsu, Intel, SUSE, and many others) are contributing to the development effort. Btrfs is included in the mainline Linux kernel.

Oracle Linux 6.3 with the Unbreakable Enterprise Kernel (UEK) R2 is the first release that officially supports Btrfs. The standard installation media does not have support for creating a Btrfs root file system on the initial installation. Oracle does provide an alternative boot ISO image that allows you to create a Btrfs root file system. The boot ISO installs Oracle Linux with the UEK. It requires a network installation source, accessible via FTP, HTTP, or NFS, that provides the actual RPM packages.

Btrfs provides extent-based file storage with a maximum file size of 16 Exabytes. All data and metadata is copy-on-write; this means that blocks of data are not changed on disk. Btrfs just copies the blocks and then writes out the copies to a different location. Not updating the original location eliminates the risk of a partial update or data corruption during a power failure. The copy-on-write nature of Btrfs also facilitates file system features such as replication, migration, backup, and restoration of data.

Btrfs maintains CRCs for all metadata and data so everything is checksummed to preserve the integrity of data against corruption. With a RAID-1 or RAID-10 configuration, if checksum fails on the first read, data is pulled off from another copy.

Btrfs allows you to create both readable and writable snapshots. A snapshot is a copy of an entire Btrfs subvolume taken at a given point in time. The snapshots appear as normal directories and you can access the snapshot as you would any other directory. Writeable snapshots allow you to roll back a file system to a previous state. You can take a snapshot, perform a system upgrade, and reboot into the snapshot if the upgrade causes problems.

All snapshots are writeable by default but you also have the option to create read-only snapshots. Read-only snapshots are useful for a backup and then can be deleted when the backup completes.

Btrfs allows a file system to span multiple devices. This is different from logical volume management (LVM) style of volume management. Btrfs does not create block devices; it just creates subvolumes in the file system that can then be mounted like a regular file system.

Btrfs also has built-in RAID support for levels RAID-0, RAID-1, and RAID-10. Btrfs's RAID is not multi-disk RAID like the software RAID devices created using the `mdadm` command. It is not block RAID either because it does not mirror block devices. Btrfs's RAID just ensures that for every block, there are "x" amount of copies. For RAID-1, for example, Btrfs just stores two copies of everything on two different devices.

Btrfs has online resizing and defragmentation. You can add or remove devices while the file systems remain online. When a device is removed, the extents stored on it are redistributed to the other devices in the file system. You can also replace devices while Btrfs is online. Btrfs rebalances the extents across the new disk and then you can drop the old disk from a Btrfs array.

Btrfs has transparent compression and currently supports two compression methods: `zlib` and LZO (the default). LZO offers a better compression ratio, whereas `zlib` offers faster compression. Btrfs can determine whether the blocks can be compressed and, therefore, compresses only when possible. You enable compression and specify the compression method using a `mount` option. For example, to enable LZO or `zlib` compression:

```
# mount –o compress=lzo|zlib <device> <mount_point>
```

You can also force Btrfs to always compress data:

```
# mount -o compress-force <device> <mount_point>
```

Btrfs provides efficient storage for small files. All Linux file systems address storage in block sizes, for example 4 KB. With other file systems, a file that is smaller than 4 KB wastes the leftover space. Btrfs stores these smaller files directly into the metadata, thereby providing a significant performance advantage over other file systems when creating and reading small files.

Btrfs automatically detects solid state drives (SSD) and turns off all optimizations for rotational media. For example, on spinning disks, it is important to store related data close together to reduce seeking. This requires CPU cycles to get good data locality on spinning disks, which is not as important with SSDs. TRIM support is also an optimization for SSD. It tells the SSD which blocks are no longer needed and are available to be written over.

Creating a Btrfs File System

- Install the `btrfs-progs` software package:

```
# yum install btrfs-progs
```

- Load the `btrfs` kernel module:

```
# modprobe btrfs
```

- Use the `mkfs.btrfs` command to create a file system.

```
mkfs.btrfs [options] block_device [block_device ...]
```

- To create a Btrfs file system across two devices:

```
# mkfs.btrfs /dev/sdb /dev/sdc
```

- Mount the Btrfs file system using the `mount` command, referencing either device:

```
# mount /dev/sdb /btrfs
```

To create a Btrfs file system on your system, install the `btrfs-progs` software package:

```
# yum install btrfs-progs
```

Load the `btrfs` kernel module using the `modprobe` command. Use the `lsmod` command to ensure that the kernel module is loaded.

```
# modprobe btrfs
# lsmod | grep btrfs
```

Use the `mkfs.btrfs` command to create a Btrfs file system. The syntax is:

```
mkfs.btrfs [options] block_device [block_device ...]
```

You can create a Btrfs file system on a single device or on multiple devices. Devices can be disk partitions, loopback devices (disk images in memory), multipath devices, or LUNs that implement RAID in hardware.

The available options for the `mkfs.btrfs` command are:

- `-A` *`offset`* – Specify the offset from the start of the device for the file system. The default is `0`, which is the start of the device.
- `-b` *`size`* – Specify the size of the file system. The default is all the available storage.

- `-d` *`type`* – Specify how the file system data is spanned across the devices. The *`type`* argument must be `raid0`, `raid1`, `raid10`, or `single`.
- `-l` *`size`* – Specify the leaf size, the least data item in which Btrfs stores data. The default is the page size.
- `-L` *`name`* – Specify a label name for the file system.
- `-m` *`profile`* – Specify how the file system metadata is spanned across the devices. The *`profile`* argument must be `raid0`, `raid1`, `raid10`, `single`, or `dup`.
- `-M` – Mix data and metadata chunks together for more efficient space utilization. This option affects performance for larger file systems, and is recommended only for file systems that are 1 GB or smaller.
- `-n` *`size`* – Specify the node size. The default is the page size.
- `-s` *`size`* – Specify the sector size, which is the minimum block allocation.
- `-V` – Print the `mkfs.btrfs` version and exit.

`mkfs.btrfs` Examples

To create a Btrfs file system on a single block device (for example, `/dev/sdb`):

```
# mkfs.btrfs /dev/sdb
```

To create a Btrfs file system on two block devices (for example, `/dev/sdb` and `/dev/sdc`):

```
# mkfs.btrfs /dev/sdb /dev/sdc
```

The default configuration for a file system with multiple devices is:

- `-d raid0` – Stripe the file system data across all devices.
- `-m raid1` – Mirror the file system metadata across all devices.

To create a Btrfs file system with multiple devices (`/dev/sdb` and `/dev/sdc`) and stripe both the data and the metadata:

```
# mkfs.btrfs –m raid0 /dev/sdb /dev/sdc
```

To create a Btrfs file system with multiple devices (`/dev/sdb` and `/dev/sdc`) and mirror both the data and the metadata:

```
# mkfs.btrfs –d raid1 /dev/sdb /dev/sdc
```

When you specify a single device, metadata is duplicated on that device unless you specify only a single copy. To create a Btrfs file system on a single block device (for example, `/dev/sdb`) and to specify not to duplicate the metadata:

```
# mkfs.btrfs –m single /dev/sdb
```

For RAID-10 data or metadata, you must specify an even number of at least four devices. To create a Btrfs file system and stripe the data and metadata across mirrored devices (RAID-10):

```
# mkfs.btrfs –d raid10 –m raid10 /dev/sd[bcde]
```

Mounting the File System

Use the `mount` command or make an entry in `/etc/fstab` as you would when mounting any other type of Linux file system. You can reference either device when your file system contains multiple devices. You can also reference the file system label or the UUID. Example:

```
# mount /dev/sdb /btrfs
```

`btrfs` Utility

- The `btrfs` utility requires a subcommand.

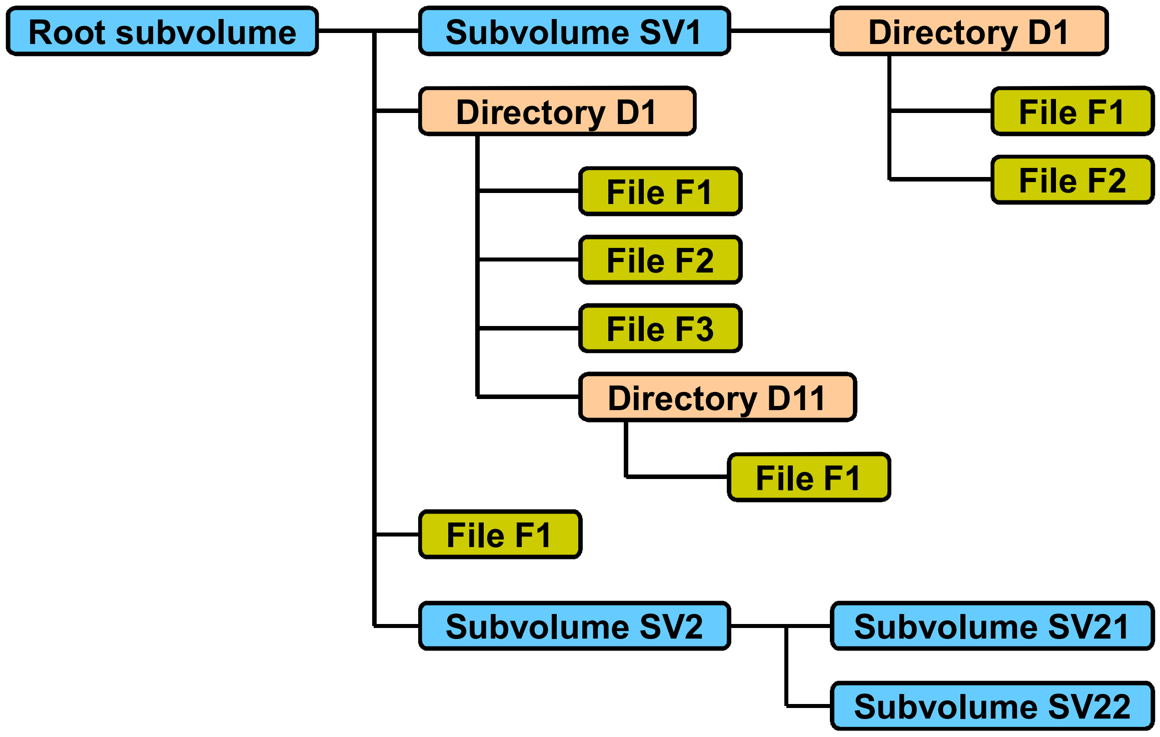
```
# btrfs
usage: btrfs [--help] [--version] <group> ... <command>
    btrfs subvolume create [<dest>/]<name>
...
```

- Available subcommands include:
  - `subvolume`
  - `filesystem`
  - `device`
  - `scrub`
  - `check | restore`
  - `inspect-internal`
  - `send | receive`
  - `quota | qgroup`

Use the `btrfs` command to manage and display information about a Btrfs file system. The command requires a subcommand. Enter `btrfs` without any arguments to list the subcommands:

```
# btrfs
Usage: btrfs [--help] [--version] <group> [<group>...] <command>
     btrfs subvolume create [<dest>/]<name>
          Create a subvolume
     btrfs subvolume delete <subvolume> [<subvolume>...]
          Delete subvolume(s)
...
     btrfs filesystem df <path>
          Show space usage information for a mount point
     btrfs filesystem show [--all-devices] [<uuid>|<label>]
          Show the structure of a filesystem
...
```

Btrfs Subvolumes



This slide illustrates a Btrfs file system hierarchy that consists of subvolumes, directories, and files. Btrfs subvolumes are named B-trees that hold files and directories. Subvolumes can also contain subvolumes, which are themselves named B-trees that can also hold files and directories. The top level of a Btrfs file system is also a subvolume, and is known as the *root subvolume*.

The root subvolume is mounted by default and Btrfs subvolumes appear as regular directories within the file system. However, a subvolume can be mounted and only files and directories in the subvolume are accessible. The following example lists the hierarchy displayed in the slide, with the default root subvolume mounted on `/btrfs`:

```
# ls -l /btrfs
drwx------ ... SV1
drwx------ ... D1
-rwxr-xr-x ... F1
drwx------ ... SV2
```

Mounting the `SV1` subvolume or the `SV2` subvolume on `/btrfs` allows access only to the files and directories within the respective subvolumes. Remount the root subvolume to gain access to the entire hierarchy.

Use the `btrfs subvolume` command to manage and report on Btrfs subvolumes. A list of the available subvolume commands is as follows:

```
# btrfs subvolume
usage: btrfs subvolume <command> <args>
     btrfs subvolume create [<dest>/]<name>
          Create a subvolume
     btrfs subvolume delete <subvolume> [<subvolume>...]
          Delete a subvolume
     btrfs subvolume list [-agopurts] [-G [+|-]value] [--
sort=gen,ogen,rootid,path] <path>
          List subvolumes (and snapshots)
     btrfs subvolume snapshot [-r] <source> [<dest>/]<name>
          Create a snapshot of the subvolume
     btrfs subvolume get-default <path>
          Get the default subvolume of a filesystem
     btrfs subvolume set-default <subvolid> <path>
          Set the default subvolume of a filesystem
     btrfs subvolume find-new <path> <lastgen>
          List the recently modified files in a filesystem
     btrfs subvolume show <subvol-path>
          Show more information of the subvolume
```

The word “`subvolume`” in the `btrfs` command can be abbreviated to “`sub`”. For example, both of the following commands are valid:

```
# btrfs subvolume create /btrfs/SV1
# btrfs sub create /btrfs/SV1
```

The abbreviation applies to other `btrfs` subcommands as well. For example, both of the following subcommands are valid:

```
# btrfs filesystem df /btrfs
# btrfs file df /btrfs
```

`btrfs subvolume` Utilities

- Use the `btrfs subvolume create` command to create a subvolume on a mounted Btrfs file system, such as:

```
# btrfs subvolume create /btrfs/SV1
```

- The subvolume appears as a normal directory when the `ls` command is used (only a partial output is shown):

```
# ls -l /btrfs
drwxr-xr-x  ...  SV1
```

- Use the `btrfs subvolume list` command to view the subvolumes in a Btrfs file system, as in this example:

```
# btrfs subvolume list /btrfs
ID 258 gen 10 top level 5 path SV1
```

Use the `btrfs subvolume create` command to create a subvolume. The following example creates a subvolume named `SV1` on a Btrfs file system mounted on `/btrfs`:

```
# btrfs subvolume create /btrfs/SV1
Create subvolume '/btrfs/SV1'
```

The subvolume appears as a regular directory. The following example creates a regular directory in `/btrfs` and then displays the content:

```
# mkdir /btrfs/D1
# ls -l /btrfs
drwxr-xr-x ... D1
drwxr-xr-x ... SV1
```

Use the `btrfs subvolume list` command to view only the subvolumes in a Btrfs file system, as in this example:

```
# btrfs subvolume list /btrfs
ID 258 gen 10 top level 5 path SV1
```

This command also displays the subvolume ID (`258`), root ID generation of the B-tree (`10`), and the top-level ID (`5`). These fields are described later in this lesson.

Btrfs Snapshots

- A snapshot is a point-in-time copy of a subvolume.
- Snapshots are created quickly and initially consume very little disk space.
- Use the `btrfs subvolume snapshot` command to create a snapshot of a subvolume.
- The following example creates a writable/readable snapshot named `SV1-snap` of the `SV1` subvolume:

```
# btrfs subvolume snapshot /btrfs/SV1 /btrfs/SV1-snap
```

- Use the `-r` option to create a read-only snapshot:

```
# btrfs subvolume snapshot -r /btrfs/SV1 /btrfs/SV1-
  rosnap
```

Btrfs subvolumes can be snapshotted and cloned, which creates additional B-trees. A snapshot starts as a copy of a subvolume taken at a point in time. You can make a snapshot writable and use it as an evolving clone of the original subvolume. Or you can use the snapshot as a stable image of a subvolume for backup purposes or for migration to other systems. Snapshots can be created quickly and they initially consume very little disk space.

Use the `btrfs subvolume snapshot` command to create a writable/readable snapshot of a subvolume. The following example creates a snapshot of the `SV1` subvolume:

```
# btrfs subvolume snapshot /btrfs/SV1 /btrfs/SV1-snap
Create a snapshot of '/btrfs/SV1' in '/btrfs/SV1-snap'
```

Use the `btrfs subvolume snapshot -r` option to create a read-only snapshot:

```
# btrfs subvolume snapshot -r /btrfs/SV1 /btrfs/SV1-rosnap
Create a readonly snapshot of '/btrfs/SV1' in '/btrfs/SV1-
rosnap'
```

The snapshots appear as a regular directory when the `ls` command is used. Snapshots also appear in the output of the `btrfs subvolume list` command.

Taking a Snapshot of a File

- Use the `cp --reflink` command to take a snapshot of a file.
- A new file shares the same disk blocks as the original file.
- The copy operation is almost instantaneous and also saves disk space.
- This operation works only within the boundaries of the same Btrfs file system and within the same subvolume.
- Example:

```
# cp --reflink /btrfs/SV1/file /btrfs/SV1/copy_of_file
```

You can use the `cp --reflink` command to take a snapshot of a file. With this option, the file system does not create a new link pointing to an existing inode, but instead creates a new inode that shares the same disk blocks as the original copy. The new file appears to be a copy of the original file but the data blocks are not duplicated. This allows the copy to be almost instantaneous and also saves disk space. As the file's content diverges over time, its amount of required storage grows. One restriction is that this operation can work only within the boundaries of the same file system and within the same subvolume.

The following example copies a file using the `cp --reflink` command. The space used is given both before and after the copy operation. Note that the space used does not increase.

```
# df -h /btrfs
Filesystem   Size  Used  Avail  Use%  Mounted on
/dev/sdb      16G  8.2M    14G    1%  /btrfs
# cp --reflink /btrfs/SV1/vmlinuz* /btrfs/SV1/copy_of_vmlinuz
# df -h
Filesystem   Size  Used  Avail  Use%  Mounted on
/dev/sdb      16G  8.2M    14G    1%  /btrfs
```

Mounting a Subvolume or Snapshot

- To mount a subvolume or snapshot, you must first determine the ID number.
- Use the `btrfs subvolume list` command to display the ID numbers, as in this example:

```
# btrfs subvolume list /btrfs
ID 258 gen 12 top level 5 path SV1
ID 259 gen 9 top level 5 path SV1-snap
```

- Use the `btrfs subvolume set-default` command to change the ID number to the entity to be mounted:

```
# btrfs subvolume set-default 259 /btrfs
```

- Unmount and remount the file system.
- Alternatively, use `-o subvolid=#` when mounting the file system, but this does not change the default subvolume ID.

By default, Linux mounts the parent Btrfs volume, which has an ID of `0`. In this example, the following `mount` command was issued before creating any subvolumes and snapshots:

```
# mount /dev/sdb /btrfs
```

The subvolume `SV1` was created in `/btrfs`. The `ls` command shows the subvolume:

```
# ls -l /btrfs
drwx------   ...   SV1
```

The following example copies files into `SV1`, creates a snapshot of `SV1`, and verifies that both the subvolume and the snapshot contain the same files:

```
# cp /boot/vmlinuz-3.8.13* /btrfs/SV1
# btrfs sub snapshot /btrfs/SV1 /btrfs/SV1-snap
# ls /btrfs/SV1*
/btrfs/SV1:
vmlinuz-3.8.13-16.2.1.el6uek.x86_64
/btrfs/SV1-snap:
vmlinuz-3.8.13-16.2.1.el6uek.x86_64
```

If you unmount `/btrfs` and remount it, the parent Btrfs volume is mounted by default:

```
# ls /btrfs
SV1  SV1-snap
# umount /btrfs
# mount /dev/sdb /btrfs
# ls /btrfs
SV1  SV1-snap
```

You can, however, mount a `btrfs` subvolume or snapshot as though it were a disk device. If you mount a snapshot instead of its parent subvolume, you effectively roll back the state of the file system to the time that the snapshot was taken.

The following example copies a file to `SV1` so that the content is different from `SV1-snap`:

```
# cp ~/test-file /btrfs/SV1
# ls /btrfs/SV1*
/btrfs/SV1:
test-file       vmlinuz-3.8.13-16.2.1.el6uek.x86_64
/btrfs/SV1-snap:
vmlinuz-3.8.13-16.2.1.el6uek.x86_64
```

To mount a subvolume or snapshot, you must first determine the ID number of the subvolume that you want to mount. Use the `btrfs subvolume list` command to display the ID numbers. In the following example, the ID of the root subvolume is `5`:

```
# btrfs subvolume list /btrfs
ID 258 gen 12 top level 5 path SV1
ID 259 gen 9 top level 5 path SV1-snap
```

Use the `btrfs subvolume set-default` command to set the default subvolume of a file system. For example, to mount the `SV1` Btrfs subvolume, which has an ID of `258`:

```
# btrfs subvolume set-default 258 /btrfs
```

You then need to unmount and remount the Btrfs file system. The root level then contains the contents of the `SV1` subvolume and the root subvolume is no longer visible:

```
# umount /btrfs
# mount /dev/sdb /btrfs
# ls /btrfs
test-file       vmlinuz-3.8.13-16.2.1.el6uek.x86_64
```

You can also use the `-o subvolid` option to the `mount` command to mount the root subvolume or a subvolume or snapshot. For example, to mount the root subvolume:

```
# umount /btrfs
# mount -o subvolid=5 /dev/sdb /btrfs
# ls /btrfs
SV1  SV1-snap
```

## `btrfs filesystem` Utilities

- Use the `btrfs filesystem` command to manage and report on Btrfs file systems.
- Available commands include:
  - `btrfs filesystem df`
  - `btrfs filesystem show`
  - `btrfs filesystem sync`
  - `btrfs filesystem defragment`
  - `btrfs filesystem resize`
  - `btrfs filesystem balance`
  - `btrfs filesystem label`
- For example, to display the file system label:

```
# btrfs filesystem label /btrfs
Btrfs
```

Use the `btrfs filesystem` command to manage and report on Btrfs file systems. A partial list of the available commands is as follows:

```
# btrfs filesystem
Usage: btrfs filesystem [<group>] <command> [<args>]
        btrfs filesystem df <path>
                Show space usage information for a mount point
        btrfs filesystem show [--all-devices] [<uuid>|<label>]
                Show the structure of a filesystem
        btrfs filesystem sync <path>
                Force a sync on a filesystem
        btrfs filesystem defragment [options] <file>|<dir> [...]
                Defragment a file or a directory
        btrfs filesystem resize [devid:][+/-]<newsize>[gkm]|...
                Resize a filesystem
...
```

`btrfs filesystem df` Utility

- Use the `btrfs filesystem df` command to show accurate space usage information for a mount point:

```
# btrfs filesystem df /btrfs
Data, RAID1: total=1.00GB, used 4.02MB
Data, total=8.00MB, used=0.00
System, RAID1: total=8.00MB, used=4.00KB
System, total=4.00MB, used=0.00
Metadata, RAID1: total=8.00MB, used=4.00KB
Metadata, total=8.00MB, used=0.00
```

- Btrfs allocates space on disks in chunks.
  - A chunk is 1 GB for data and 256 MB for metadata.
  - A chunk also has a specific RAID profile associated with it.

Some information is presented when you create a Btrfs file system. The following example creates a Btrfs file system with two 8 GB devices (`/dev/sdb` and `/dev/sdc`) and mirrors both the data and the metadata (metadata is mirrored by default):

```
# mkfs.btrfs -L Btrfs -d raid1 /dev/sdb /dev/sdc
...adding device /dev/sdc id 2
fs created label Btrfs on /dev/sdb
     nodesize 4096 leafsize 4096 sectorsize 4096 size 16.00GB
...
```

The preceding output shows that the block size is 4 KB with a total of 16 GB of space. But because the array is RAID1, you can fit only 8 GB of data on this file system. You actually have less than 8 GB because space is needed for the metadata as well. The example continues with creating a mount point and mounting the file system:

```
# mkdir /btrfs
# mount /dev/sdb /brtfs
```

As previously discussed, you can mount by referencing either device in the array, the LABEL, or the UUID.

Even the `/proc/mounts` file does not show the second device for the Btrfs file system:

```
# grep btrfs /proc/mounts
/dev/sdb /btrfs btrfs rw,seclabel,relatime,space_cache 0 0
```

For example, the following command copies a file to the Btrfs file system:

```
# cd /btrfs
# cp /boot/vmlinuz-3.8.13-16.2.1.el6uek.x86_64 .
# ls -l
-rwxr-xr-x ... vmlinuz...
```

When the file system is mounted and has a file copied to it, the output of the `df` command produces inaccurate information for the Btrfs file system:

```
# sync
# df –h
Filesystem  Size  Used  Avail  Use%  Mounted on
...
/dev/sdb     16G  8.2M    14G    1%  /btrfs
```

This output shows that the file system has a size of 16 G, which is not accurate because this is a RAID-1 array. To get accurate space information for a Btrfs file system, use the `btrfs filesystem df` command:

```
# btrfs filesystem df /btrfs
Data, RAID1: total=1.00GB, used 4.02MB
Data, total=8.00MB, used=0.00
System, RAID1: total=8.00MB, used=4.00KB
System, total=4.00MB, used=0.00
Metadata, RAID1: total=1.00GB, used=32.00KB
Metadata, total=8.00MB, used=0.00
```

Btrfs allocates space on disks in chunks. A chunk is 1 GB for data and 256 MB for metadata. A chunk also has a specific RAID profile associated with it, which allows Btrfs to have different allocation profiles for data and for metadata. The output of the `btrfs filesystem df` command shows that it has allocated only a 1 GB chunk of RAID-1 at this time.

Btrfs is not yet actually "RAIDing" the entire device. For example, if you specify RAID-1 for metadata and RAID-0 for data, metadata writes are mirrored across all the disks and data writes are striped across the disks.

The output of the `btrfs filesystem df` command shows that you are currently using 4.02 MB. The disk (system RAID1) has a total allocated space of 8 MB and has used 4 KB. Metadata is allocated 1 GB of space as well; it has used 32 KB of it.

`btrfs filesystem show|sync` Utilities

- Use the `btrfs filesystem show` command to display the structure of a file system, as in this example:

```
# btrfs filesystem show Btrfs
Label: 'btrfs' uuid: 8bb5...
   Total devices 2 FS bytes used 28.00KB
    devid    2 size 8.00GB used 1.81GB path /dev/sdc
    devid    1 size 8.00GB used 1.83GB path /dev/sdb
```

- Use the `btrfs filesystem sync` command to force a sync for the file system:

```
# btrfs filesystem sync /btrfs
FSSync '/btrfs'
```

Use the `btrfs filesystem show` command to display the structure of a file system. The syntax follows:

```
btrfs filesystem show [--all-devices] [<uuid>|<label>]
```

If you omit the optional `uuid` and `label`, the command shows information about all the Btrfs file systems. If you provide the `--all-devices` argument, all devices under `/dev` are scanned. Otherwise, the device list is obtained from `/proc/partitions`.

The following example displays the structure of the Btrfs file system with a label of `Btrfs`:

```
# btrfs filesystem show Btrfs
Label: 'Btrfs' uuid: 8bb5...
        Total devices 2 FS bytes used 4.05MB
        devid    2 size 8.00GB used 2.01GB path /dev/sdc
        devid    1 size 8.00GB used 2.03GB path /dev/sdb
```

Use the `btrfs filesystem sync` command to force a sync for the file system. The file system must be mounted. To force a sync of the file system mounted on `/btrfs`:

```
# btrfs filesystem sync /btrfs
FSSync '/btrfs'
```

`btrfs filesystem defragment` Utility

- Use the `btrfs filesystem defragment` command to defragment a file system, file, or directory.
- To defragment a file system:

```
# btrfs filesystem defragment /btrfs
```

- To defragment and compress a file system:

```
# btrfs filesystem defragment -c /btrfs
```

- Set up automatic defragmentation by specifying the `autodefrag` option with the `mount` command:

```
# mount -o autodefrag /dev/sdb /btrfs
```

ORACLE

Btrfs provides online defragmentation of a file system, file, or directory. The online defragmentation facility re-organizes data into contiguous chunks wherever possible to create larger sections of available disk space and to improve read and write performance. Use the `btrfs filesystem defragment` command to defragment a file or a directory.

```
btrfs filesystem defragment [options] <file>|<dir> [...]
```

The available options include the following:

- `-v` – Verbose
- `-c` – Compress file contents while defragmenting.
- `-f` – Flush file system after defragmenting.
- `-s` *`start`* – Defragment only from byte *`start`* onward.
- `-l` *`len`* – Defragment only up to *`len`* bytes.
- `-t` *`size`* – Defragment files only at least *`size`* bytes.

You can set up automatic defragmentation by specifying the `-o autodefrag` option when you mount the file system. Do not defragment with kernels up to version `2.6.37` if you have created snapshots or made snapshots of files using the `cp --reflink` option. Btrfs in these earlier kernels unlinks the copy-on-write copies of data.

`btrfs filesystem resize` Utility

- Use the `btrfs filesystem resize` command to resize a file system.
- To accommodate the resizing, you must have space available on the underlying devices.
- To reduce the file system by 2 GB:

```
# btrfs filesystem resize -2G /btrfs
```

- To increase the file system by 2 MB:

```
# btrfs filesystem resize +2M /btrfs
```

- To have the file system occupy all available space:

```
# btrfs filesystem resize max /btrfs
```

Btrfs provides online resizing of a file system. Use the `btrfs filesystem resize` command to resize a file system. You must have space available to accommodate the resizing because the command has no effect on the underlying devices. The syntax is as follows:

```
    btrfs filesystem resize [+/-]<newsize>[gkm]|max <path>
```

Descriptions of the parameters:

- `+ `*`newsize`* – Increases the file system size by *`newsize`* amount
- `- `*`newsize`* – Decreases the file system size by *`newsize`* amount
- *`newsize`* – Specifies the *`newsize`* amount
- `g`, `k`, or `m` – Specifies the unit of *`newsize`*, either GB or KB or MB. If no units are specified, the parameter defaults to bytes.
- `max` – Specifies that the file system occupy all available space

For example, to reduce the size of the file system by 2 GB:

```
    # btrfs filesystem resize -2G /btrfs
    Resize '/btrfs/' of '-2G'
```

`btrfs device` Utilities

- Use the `btrfs device` command to manage devices on Btrfs file systems.
- Available commands include:
  - `btrfs device add|delete|scan|ready|stats`
- The `btrfs device scan` command scans physical devices looking for members of a Btrfs volume.
  - This allows a multiple-disk Btrfs file system to be mounted without specifying all the disks on the `mount` command.
- `Udev` automatically runs `btrfs device scan` on boot.
- The `btrfs device ready` command checks whether all devices are in cache for mounting.
- The `btrfs device stats` command shows IO stats.

Use the `btrfs device` command to manage devices on Btrfs file systems. A list of the available commands is as follows:

```
# btrfs device
Usage: btrfs device <command> [<args>]
     btrfs device add <device> [<device>...] <path>
          Add a device to a filesystem
     btrfs device delete <device> [<device>...] <path>
          Remove a device from a filesystem
     btrfs device scan [--all-devices|<device>| [<device>...]
          Scan devices for a btrfs filesystem
...
```

The `btrfs device scan` command scans physical devices looking for members of a Btrfs volume. This command allows a multiple-disk Btrfs file system to be mounted without specifying all the disks on the `mount` command.

You do not need to run `btrfs device scan` from the command line, because `udev` automatically runs `btrfs device scan` on boot.

`btrfs device` Utility: Examples

- Use the `btrfs device add` command to add a device to a mounted file system, as in this example:

```
# btrfs device add /dev/sdd /btrfs
```

- Use the `btrfs filesystem balance` command after adding a device:

```
# btrfs filesystem balance /btrfs
```

- Use the `btrfs device delete` command to remove a device from a file system:

```
# btrfs device delete /dev/sdd /btrfs
```

Use the `btrfs device add` command to add a device to a file system. In this example, the current file system structure is as follows:

```
# btrfs filesystem show Btrfs
Label: 'Btrfs' uuid: 8bb5...
        Total devices 2 FS bytes used 4.05MB
        devid    2 size 8.00GB used 2.01GB path /dev/sdc
        devid    1 size 8.00GB used 2.03GB path /dev/sdb
```

The `btrfs filesystem df` command shows:

```
# btrfs filesystem df /btrfs
Data, RAID1: total=1.00GB, used 4.02MB
Data, total=8.00MB, used=0.00
System, RAID1: total=8.00MB, used=4.00KB
System, total=4.00MB, used=0.00
Metadata, RAID1: total=1.00GB, used=32.00KB
Metadata, total=8.00MB, used=0.00
```

The output of the `df` command shows:

```
# df -h
Filesystem  Size  Used  Avail  Use%  Mounted on
...
/dev/sdb     16G  8.2M    14G    1%  /btrfs
```

Add an 8 GB disk, `/dev/sdd`, to the file system mounted on `/btrfs` using the `btrfs device add` command:

```
# btrfs device add /dev/sdd /btrfs
```

The output of the `btrfs filesystem show` command shows the newly added device:

```
Total devices 3 FS bytes used 4.05MB
        devid    3 size 8.00GB used 0.00 path /dev/sdd
        devid    2 size 8.00GB used 2.01GB path /dev/sdc
        devid    1 size 8.00GB used 2.03GB path /dev/sdb
```

The output of the `btrfs filesystem df` command shows no difference after adding the new device:

```
# btrfs filesystem df /btrfs
Data, RAID1: total=1.00GB, used 4.02MB
Data, total=8.00MB, used=0.00
System, RAID1: total=8.00MB, used=4.00KB
System, total=4.00MB, used=0.00
Metadata, RAID1: total=1.00GB, used=32.00KB
Metadata, total=8.00MB, used=0.00
```

There is no difference in the output because the newly added device has not yet been allocated for either data or metadata.

The additional size is reflected in the output of `df`:

```
# df -h
Filesystem  Size  Used  Avail  Use%  Mounted on
...
/dev/sdb     24G  8.2M    14G    1%  /btrfs
```

After adding a device, it is recommended that you run the following `balance` command on the file system:

```
# btrfs filesystem balance /btrfs
```

Running this command redistributes space by balancing the chunks of the file system across all the devices. This command also reclaims any wasted space.

Use the `btrfs device delete` command to remove a device from a file system.

Example:

```
# btrfs device delete /dev/sdd /btrfs
```

`btrfs scrub` Utilities

- Use the `btrfs scrub` command to manage scrubbing on Btrfs file systems.
- Scrubbing is performed in the background by default. It attempts to report and repair bad blocks on the file system.
- Available commands include:
  - `btrfs scrub start`
  - `btrfs scrub cancel`
  - `btrfs scrub resume`
  - `btrfs scrub status`

You can initiate a check of the entire file system by triggering a file system scrub job. The scrub job runs in the background by default and scans the entire file system for integrity. It automatically attempts to report and repair any bad blocks that it finds along the way. Instead of going through the entire disk drive, the scrub job deals only with data that is actually allocated. Depending on the allocated disk space, this is much faster than performing an entire surface scan of the disk.

Scrubbing involves reading all the data from all the disks and verifying checksums. If any values are not correct, the data can be corrected by reading a good copy of the block from another drive. The scrubbing code also scans on read automatically. It is recommended that you scrub high-usage file systems once a week and all other file systems once a month.

The following is a partial list of the available `btrfs scrub` commands:

```
# btrfs scrub
Usage: btrfs scrub <command> [options] <path>|<device>
     btrfs scrub start [-Bdqr] <path>|<device>
          Start a new scrub
...
     btrfs scrub status [-dR] <path>|<device>
          Show status of running or finished scrub
```

`btrfs scrub` Utility: Examples

- Use the `btrfs scrub start` command to start a scrub on all the devices of a file system or on a single device.

```
# btrfs scrub start /btrfs
```

- Use the `btrfs scrub status` command to get the status of a scrub job. The following example includes detailed scrub information about each device in the file system:

```
# btrfs scrub status -dR /btrfs
```

- Use the `btrfs scrub cancel` command to cancel a running scrub job:

```
# btrfs scrub cancel /btrfs
```

- Use the `btrfs scrub resume` command to resume a previously cancelled or interrupted scrub:

```
# btrfs scrub resume /btrfs
```

Use the `btrfs scrub start` command to start a scrub on all the devices of a file system or on a single device. The syntax is as follows:

```
btrfs scrub start [-Bdqr] <path>|<device>
```

Description of options:

- `-B` – Do not run in the background and print statistics when finished.
- `-d` – Print separate statistics for each device of the file system. This option is used in conjunction with the `-B` option.
- `-q` – Run in quiet mode, omitting error messages and statistics.
- `-r` – Run in read-only mode, not correcting any errors.

The following example starts a scrub on the Btrfs file system that is mounted on `/btrfs`.

```
# btrfs scrub start -B /btrfs
scrub started on /brtfs, fsid ... (pid=...)
```

Use the `btrfs scrub status` command to get the status of a scrub job. Two options are available:

- `-d` – Print separate statistics for each device of the file system.
- `-R` – Print detailed scrub statistics.

The following is a partial output from the `btrfs scrub status` command with the `-dR` options:

```
# btrfs scrub status -dR /btrfs
scrub status for ...
scrub device /dev/sdb (id 1) history
     scrub started at ... and finished after ... seconds
     data_extends_scrubbed: 997
     tree_extends_scrubbed: 9
     data_bytes_scrubbed: 4083712
     tree_bytes_scrubbed: 36864
     read_errors: 0
     csum_errors: 0
     verify_errors: 0
     no_csum: 32
     csum_discards: 0
     super_errors: 0
     malloc_errors: 0
     uncorrectable_errors: 0
     unverified_errors: 0
     corrected_errors: 0
     last_physical: 1707081728
scrub device /dev/sdb (id 1) history
     scrub started at ... and finished after ... Seconds
     ...
```

You can also cancel a running scrub job. Progress is saved in the scrub progress file and you can resume scrubbing later.

To cancel a scrub:

```
# btrfs scrub cancel /btrfs
```

To resume a canceled or interrupted scrub job:

```
# btrfs scrub resume /btrfs
```

The `scrub resume` command has the same options as the `scrub start` command.

Btrfs stores the last two minutes, at 30-second intervals, of root ID generations. Btrfs continues to keep rolling these generations, even if there are no changes in the file system.

If a scrub does not correct errors, you can use the following mount option to roll back to a known good B-tree, given that the rest of the tree is available because of copy-on-write:

```
# mount -o recovery /dev/xvdb /btrfs
```

Converting Ext File Systems to Btrfs

- Use the `btrfs-convert` utility to convert an ext2, ext3, or ext4 file system to a Btrfs file system.
- To convert a non-`root` ext file system:
  1. Unmount the ext file system.
  2. Use `fsck` to check the integrity of the ext file system.
  3. Use the `btrfs-convert` utility to convert the file system.
  4. Edit `/etc/fstab` and change the file system type to `btrfs`.
  5. Mount the converted file system on the original mount point.
- The syntax of the `btrfs-convert` utility is as follows:

```
btrfs-convert <device>
```

- Refer to the *Oracle Linux Administrator's Guide* for procedures to convert the `root` file system.

Btrfs supports the conversion of ext2, ext3, and ext4 file systems to Btrfs file systems. The original ext file system metadata is stored in a snapshot named `ext#_saved` so that the conversion can be reversed if necessary.

Use the `btrfs-convert` utility to convert an ext file system. Always make a backup copy before converting a file system. To convert an ext file system other than the `root` file system, perform the steps listed in the slide.

Refer to the *Oracle Linux Administrator's Guide* for details about converting the `root` file system: http://docs.oracle.com/cd/E37670_01/

UEK Boot ISO

- An alternative installation image supports the installation of Oracle Linux by using the UEK as the installation kernel.
- The installation allows you to create a Btrfs root file system.
- To install Oracle Linux 6.5 using the UEK Boot ISO:
  1. Download the full Oracle Linux 6 update 5 DVD image from http://edelivery.oracle.com/linux.
  2. Create an installation source that is accessible via NFS or HTTP.
  3. Download the UEK Book ISO image from http://edelivery.oracle.com/linux.
  4. Replace the contents of the `images` directory.
  5. Boot the target machine by using the UEK Book ISO.
  6. Supply the `askmethod` parameter on the command line.
  7. Point to the installation source created in steps 2–4.

Beginning with Oracle Linux 6.3, Oracle provides an alternative installation image (UEK Boot ISO) that supports the installation of Oracle Linux using the Unbreakable Enterprise Kernel (UEK) as the installation kernel. This installation method allows you to create a Btrfs root file system. This ISO image contains only the bootable installation image. It requires a network installation source that provides the actual RPM packages. Perform the steps listed in the slide to install from the UEK Boot ISO.

In step 4, you replace the contents of the `images` directory in the installation source created in step 2 with the contents of the `images` directory from the UEK Book ISO. The default Oracle Linux Media follows Red Hat Enterprise Linux (RHEL) Boot Media to maintain compatibility. The UEK Boot ISO is available only with Oracle Linux and contains images to allow creation of a Btrfs root file system. If you use the default Media images directory, an ext4 root file system is created.

Quiz

Which of the following statements are true?

a. Btrfs is a general-purpose file system.
b. All Btrfs data and metadata is written via copy-on-write.
c. Btrfs supports only readable snapshots.
d. Btrfs supports online resizing and defragmentation.

Quiz

Which of the following statements are true?

a. Btrfs has built-in RAID support for RAID-0, RAID-1, RAID-5, RAID-6, and RAID-10.
b. Btrfs supports transparent compression.
c. Btrfs automatically detects and optimizes solid state drives.
d. Oracle Linux 6.3 with UEK2 is the first release to officially support Btrfs.

Quiz

Which of the following are valid `btrfs` commands?

a. `btrfs subvolume create`
b. `btrfs snapshot create`
c. `btrfs filesystem show`
d. `btrfs device create`

Summary

In this lesson, you should have learned how to:

- Describe the features of the Btrfs file system
- Create a Btrfs file system
- Create Btrfs subvolumes and snapshots
- Take a snapshot of a file in a Btrfs subvolume
- Mount Btrfs subvolumes and snapshots
- Defragment and resize a Btrfs file system
- Add and remove devices in a Btrfs file system
- Check and repair the integrity of a Btrfs file system
- Convert ext file systems to Btrfs
- Create a Btrfs root file system by installing Oracle Linux from the UEK Boot ISO

Practice 12: Overview

This practice covers the following topics:

- Creating Btrfs file systems with different specifications
- Resizing a Btrfs file system
- Adding a disk to and removing a disk from a Btrfs file system
- Creating a Btrfs subvolume and snapshot
- Mounting a subvolume and a snapshot
- Taking a snapshot of a file by using the `cp --reflink` command
- Corrupting data on a Btrfs file system and recovering from data corruption
- Installing Oracle Linux 6.5 from the UEK Boot ISO
- Exploring the Btrfs root file system

13

Control Groups (Cgroups)

Objectives

After completing this lesson, you should be able to:

- Describe the purpose of control groups
- Describe control group subsystems and parameters
- Describe the control group hierarchy model
- Configure control groups by using the `/etc/cgconfig.conf` file
- Use the `lssubsys` utility
- Use the `cgcreate` and `cgdelete` utilities
- Use the `cgset` utility
- Assign tasks to control groups
- Use the `cgrules.conf` file
- Enable PAM to use control group rules
- Obtain information about control groups and parameters

Control Groups: Introduction

Control groups (cgroups) enable you to allocate computing resources to specific processes or tasks. You can:

- Assign a set of CPU cores and memory nodes to a specific group of tasks
- Specify the relative share of CPU time available to the tasks in a cgroup
- Specify memory limits for the tasks in a cgroup
- Suspend the tasks in a cgroup
- Report the CPU time and memory used by the tasks in a cgroup
- Assign different priorities to network packets from different cgroups
- Deny or allow tasks in a cgroup access to specific devices

Control groups (cgroups) provide a mechanism to put Linux processes (tasks) into groups to ensure that critical workloads get the system resources (CPU, memory, and I/O) that they need. You can allocate system resources, track usage, and impose limits on the cgroups.

Cgroups provide more fine-grained control of CPU, I/O, and memory resources. You can associate a set of CPU cores and memory nodes with a group of processes that make up an application or a group of applications. This enables the subsetting of larger systems; more fine-grained control over memory, CPUs, and devices; and the isolation of applications.

For example, with very large NUMA systems, you make the best use of system resources by compartmentalizing. Cgroups give you a great deal of control over how to set up a system, which memory to give, and which CPUs to give to an individual task. You can pin processes to the same NUMA node and use NUMA-local memory. Cgroups facilitate database consolidation on large NUMA servers, I/O throttling support, and device whitelisting. Cgroups work inside virtual guests as well.

Some capabilities (specifically the `blkio` subsystem that provides I/O throttling) require the Oracle Linux Unbreakable Enterprise Kernel release 2 (`2.6.39.el6uek`) or the Red Hat compatible kernel in Oracle Linux 6.1 or later (`2.6.32.el6`)

A sample implementation is shown on the following page.

The following example defines a root cgroup named `cpu_ram` that includes the `cpuset`, `cpu`, and `memory` subsystems. It also defines a child cgroup, named `dbgrp`, for database processes. It allows the `oracle` user to add tasks and also defines certain parameters for CPU and memory usage. This configuration is contained in the cgroup configuration file `/etc/cgconfig.conf`:

```
mount {
      cpuset  = /cgroup/cpu_ram;
      cpu     = /cgroup/cpu_ram;
      memory  = /cgroup/cpu_ram;
}
group dbgrp {
      perm {
            task {
                  uid = oracle;
                  gid = dba;
            }
      }
      cpu {
            # Tasks in cgroup access CPU resources for 4 out of
            # every 5 seconds
            cpu.rt_period_us="5000000";
            cpu.rt_runtime_us="4000000";
      }
      cpuset {
            cpuset.mems="0";
            # Allocate CPU cores 4 through 7 to tasks in cgroup
            cpuset.cpus="4-7";
      }
      memory {
            # Allocate up to 4GB of memory to tasks in the cgroup
            memory.limit_in_bytes="4G";
            # Allocate up to 8GB of memory plus swap to the tasks
            memory.memsw.limit_in_bytes="8G";
            # Apply a soft limit of 2GB to tasks in the cgroup
            memory.soft_limit_in_bytes="2G";
      }
}
```

Cgroup Subsystems (Resource Controllers)

A subsystem is a kernel resource controller that applies limits or acts on a group of processes. Subsystems include:

- `cpuset` – Assigns individual CPUs and memory nodes to cgroup tasks
- `cpu` – Schedules CPU access to cgroup tasks
- `cpuacct` – Reports the total CPU time used by cgroup tasks
- `memory` – Reports or limits memory use of cgroup tasks
- `devices` – Grants or denies access to devices
- `freezer` – Suspends or resumes cgroup tasks
- `net_cls` – Tags outgoing network packets with an identifier, which can then be assigned a different priority
- `blkio` – Reports or controls I/O bandwidth for block devices

A subsystem is a kernel resource controller that applies limits or acts on a group of Linux processes. A subsystem represents a single kernel resource, such as memory or CPU cores. A cgroup associates a group of processes or tasks with a set of parameters for one or more subsystems. Processes assigned to each group are subject to the subsystem parameters.

Here are brief descriptions of the cgroup subsystems:

- `cpuset` – Assigns individual CPUs and memory nodes (for systems with NUMA architectures) to cgroup tasks
- `cpu` – Schedules CPU access to cgroup tasks using two schedulers: completely fair scheduler (CFS) and real-time scheduler (RT)
- `cpuacct` – Reports the total CPU time used by cgroup tasks
- `memory` – Reports or limits memory use of cgroup tasks
- `devices` – Grants or denies cgroup tasks access to devices
- `freezer` – Suspends or resumes cgroup tasks
- `net_cls` – Tags outgoing network packets with an identifier. You can configure the Linux traffic controller (`tc`) to assign different priorities to packets from different cgroups.
- `blkio` – Reports or controls I/O bandwidth for block devices. You can assign proportional weights to specific cgroups or set an upper limit for the number of I/O operations performed by a device.

Cgroup Subsystem Parameters

- Each subsystem has specific parameters that enable resource control and reporting mechanisms.
- Subsystem parameters allow you to set limits, restrict access, or define allocations for each subsystem.
- Understanding the specific parameters helps you understand the possibilities for controlling resources with cgroups.
- Each subsystem cgroup also has a set of common files, including the `tasks` file:
  - The `tasks` file keeps track of processes associated with the cgroup and the associated subsystem parameter settings.
  - The `tasks` file contains all the process IDs (PIDs) assigned to the cgroup.

Each subsystem has specific parameters that enable resource control and reporting mechanisms. The subsystem parameters are the heart of cgroup resource controls. They allow you to set limits, restrict access, or define allocations for each subsystem. Understanding the specific parameters helps you to understand the possibilities for controlling resources with cgroups.

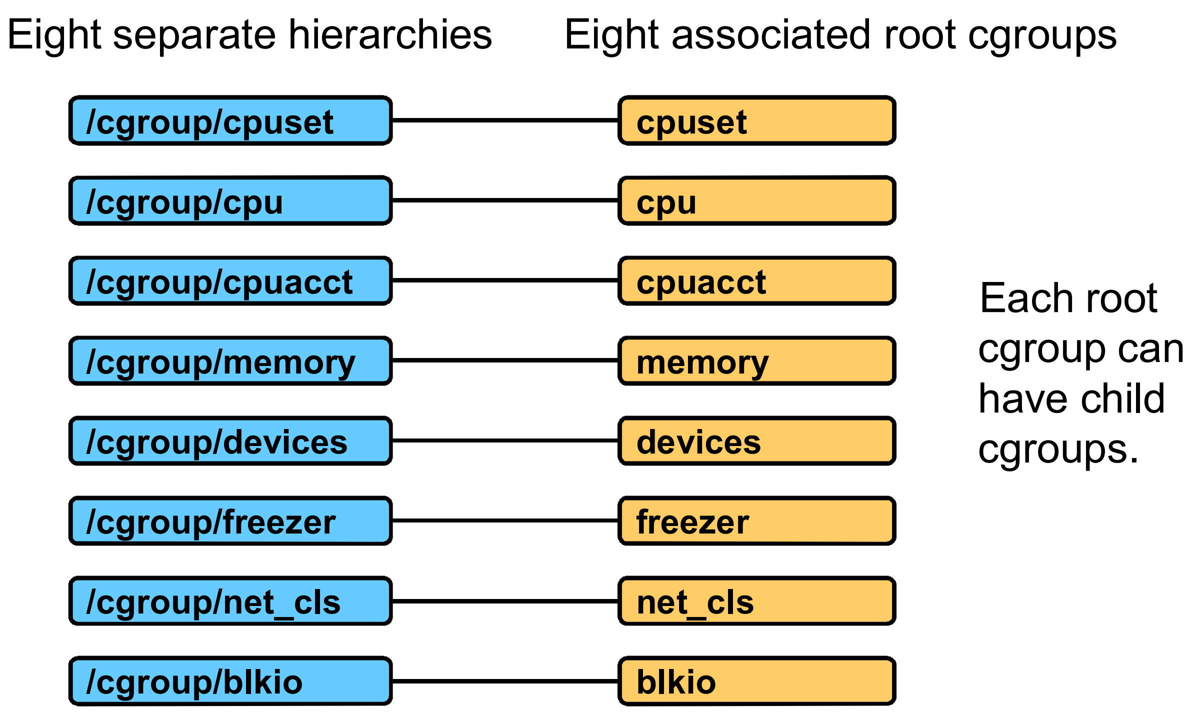
In addition to the subsystem parameters, each subsystem directory contains the following files:

- `cgroup.clone_children`
- `cgroup.event_control`
- `cgroup.procs`
- `notify_on_release`
- `release_agent`
- `tasks`

The `tasks` file keeps track of the processes associated with the cgroup and the associated subsystem parameter settings. After the setup is complete, the tasks file contains all the process IDs (PIDs) assigned to the cgroup.

Refer to the appendix titled “Cgroup Subsystem Parameters” for a description of each subsystem parameter.

Cgroup Hierarchy



This slide illustrates the default cgroup hierarchy. In the default hierarchy, each of the subsystems is attached to separate hierarchies, `/cgroup/<subsystem>`. Each hierarchy has an associated cgroup, known as the *root cgroup*. All the processes, or tasks, on the system are initially members of the root cgroup.

Cgroups are implemented using a file system–based model. You can traverse the `/cgroup` hierarchy to view current control group hierarchies, parameter assignments, and associated tasks. In this file system hierarchy, children inherit characteristics from their parents.

A hierarchy is a set of subsystems and cgroups arranged in a tree, so that every system process is in exactly one of the cgroups in the hierarchy. Groups can be hierarchical, where each group inherits characteristics from its parent group.

Many different hierarchies of cgroups can exist simultaneously on a system. Whereas the Linux process model is a single tree of processes (all processes are child processes of a common parent: the init process), the cgroup model is one or more separate, unconnected trees of tasks (that is, processes). Multiple separate hierarchies of cgroups are necessary because each hierarchy is attached to one or more subsystems.

Cgroup Configuration Rules and Constraints

- A hierarchy can have multiple subsystems attached.
  - For example, the `/cgroup/cpu-mem` hierarchy can have both `cpu` and `memory` subsystems attached to it.
  - Both `cpu` and `memory` subsystem parameters reside in the single `/cgroup/cpu-mem` hierarchy.
- A subsystem cannot be attached to a second hierarchy if the second hierarchy has a different subsystem attached.
  - The `cpu` subsystem is attached to the `/cgroup/cpu` hierarchy.
  - The `memory` subsystem is attached to the `/cgroup/memory` hierarchy.
  - Attaching the `cpu` subsystem to the `/cgroup/memory` hierarchy fails.
- A task cannot be a member of two different cgroups in the same hierarchy.

A single hierarchy can have one or more subsystems attached to it. For example, both the `cpu` and `memory` subsystems can be attached to a single hierarchy, `/cgroup/cpu-mem`. These two subsystems (`cpu` and `memory`) cannot be attached to any other hierarchy that has other subsystems already attached to it.

A subsystem cannot be attached to a second hierarchy if the second hierarchy has a different subsystem already attached to it. For example, if the `cpu` subsystem is attached to the `/cgroup/cpu` hierarchy and the `memory` subsystem is attached to the `/cgroup/memory` hierarchy, an attempt to attach the `cpu` subsystem to the `/cgroup/memory` hierarchy fails. A single subsystem can be attached to two hierarchies if both those hierarchies have only that subsystem attached.

For any single hierarchy you create, each task on the system can be a member of exactly one cgroup in that hierarchy. A single task can be in multiple cgroups as long as each of those cgroups is in a different hierarchy. If you assign a task as a member of a second cgroup in the same hierarchy, it is removed from the first cgroup in that hierarchy. At no time is a task ever in two different cgroups in the same hierarchy. For example, a running `sshd` process can be a member of any one cgroup in the `/cgroup/cpu` hierarchy and can be a member of any one cgroup in the `/cgroup/memory` hierarchy. The process cannot be a member of two different cgroups in the same hierarchy.

Cgroup Configuration

- Install the `libcgroup` software package:

```
# yum install libcgroup
```

- Enable the `cgconfig` service to start at boot time:

```
# chkconfig cgconfig on
```

- The main configuration file for cgroups is `/etc/cgconfig.conf`. It is used to define control groups, their parameters, and mount points.
- The `/etc/cgconfig.conf` file contains two types of definitions.
  – `mount`: Defines the virtual file systems to mount subsystems
  – `group`: Defines a cgroup, access permissions, subsystems to use, and parameter values for those subsystems
- Start the `cgconfig` service to read the configuration file.

To enable the cgroup services on your system, install the `libcgroup` software package:

```
# yum install libcgroup
```

To ensure that the `cgconfig` service starts at boot time, enter the following command to enable the service for run levels 2, 3, 4, and 5:

```
# chkconfig cgconfig on
```

The main configuration file for cgroups is `/etc/cgconfig.conf`. When you start the `cgconfig` service, it reads this configuration file. Restart the `cgconfig` service after making any configuration changes:

```
# service cgconfig restart
```

The `/etc/cgconfig.conf` file is used to define control groups, their parameters, and mount points. The file contains two types of definitions.

- `mount`: Defines the virtual file systems that you use to mount resource subsystems before you attach them to cgroups. The configuration file can contain only one mount definition.
- `group`: Defines a cgroup, its access permissions, the resource subsystems that it uses, and the parameter values for these subsystems. The configuration file can contain more than one group definition.

`/etc/cgconfig.conf` “mount” Section

```
mount {
    cpuset  = /cgroup/cpuset;
    cpu     = /cgroup/cpu;
    cpuacct = /cgroup/cpuacct;
    memory  = /cgroup/memory;
    devices = /cgroup/devices;
    freezer = /cgroup/freezer;
    net_cls = /cgroup/net_cls;
    blkio   = /cgroup/blkio;
}
```

The syntax of the mount section in the `/etc/cgconfig.conf` file is as follows:

```
mount {
      <controller> = <path>;
}
```

By default, all resource controllers (subsystems) are mounted to `/cgroup/<controller>`:

```
mount {
      cpuset  = /cgroup/cpuset;
      cpu     = /cgroup/cpu;
      cpuacct = /cgroup/cpuacct;
      memory  = /cgroup/memory;
      devices = /cgroup/devices;
      freezer = /cgroup/freezer;
      net_cls = /cgroup/net_cls;
      blkio   = /cgroup/blkio;
}
```

Starting the `cgconfig` service with the default configuration file creates and mounts an individual hierarchy for each subsystem, and attaches the subsystems to these hierarchies.

```
# service cgconfig start
# ls -R /cgroup/
/cgroup/:
blkio  cpu  cpuacct  cpuset  devices  freezer  memory  net_cls

/cgroup/blkio:
blkio.weight   blkio.weight_device   blkio.time
...

/cgroup/cpu:
cpu.shares   cpu.rt_period_us   cpu.rt_runtime_us
...

/cgroup/cpuacct:
cpuacct.usage   cpuacct.stat   cpuacct.usage_percpu
...

/cgroup/cpuset:
cpuset.cpus   cpuset.mems   cpuset.cpu_exclusive
...

/cgroup/devices:
devices.allow   devices.deny   devices.list
...

/cgroup/freezer:
freezer.state
...

/cgroup/memory:
memory.limit_in_bytes   memory.soft_limit_in_bytes
...

/cgroup/net_cls
net_cls.classid
...
```

`lssubsys` Utility

Use the `lssubsys` command to list the hierarchies.

```
# lssubsys -m
cpuset   /cgroup/cpuset
cpu      /cgroup/cpu
cpuacct  /cgroup/cpuacct
memory   /cgroup/memory
devices  /cgroup/devices
freezer  /cgroup/freezer
net_cls  /cgroup/net_cls
blkio    /cgroup/blkio
```

Use the `lssubsys` command to list the hierarchies. The following example includes the mount points (`-m`):

```
# lssubsys -m
cpuset   /cgroup/cpuset
cpu      /cgroup/cpu
cpuacct  /cgroup/cpuacct
memory   /cgroup/memory
devices  /cgroup/devices
freezer  /cgroup/freezer
net_cls  /cgroup/net_cls
blkio    /cgroup/blkio
```

Include the subsystem(s) arguments to list specific hierarchies, as in this example:

```
# lssubsys -m blkio cpu
blkio    /cgroup/blkio
cpu      /cgroup/cpu
```

Sample “mount” Implementation

- Edit `/etc/cgconfig.conf` to create the `cpu_ram` hierarchy and attach the `cpu`, `cpuset`, and `memory` subsystems.

```
mount {
   cpuset  = /cgroup/cpu_ram;
   cpu     = /cgroup/cpu_ram;
   memory  = /cgroup/cpu_ram;
}
```

- Alternative command-line method (not persistent):

```
# mkdir /cgroup/cpu_ram
# mount -t cgroup -o cpu,cpuset,memory cpu_ram
   /cgroup/cpu_ram
```

The following sample content of the `/etc/cgconfig.conf` file creates the `cpu_ram` hierarchy, and attaches the `cpu`, `cpuset`, and `memory` subsystems:

```
mount {
     cpuset   = /cgroup/cpu_ram;
     cpu      = /cgroup/cpu_ram;
     memory   = /cgroup/cpu_ram;
}
```

Restart the `cgconfig` service to read the `/etc/cgconfig` file and create the `/cgroup/cpu_ram` hierarchy:

```
# service cgconfig restart
```

Command-Line Method

Alternatively, you can create a mount point for the hierarchy, and then attach the appropriate subsystems. The following commands create the same configuration as the preceding one:

```
# mkdir /cgroup/cpu_ram
# mount -t cgroup -o cpu,cpuset,memory cpu_ram /cgroup/cpu_ram
```

Use the `mount` command to view the mounted cgroup file system:

```
# mount | grep cgroup
cpu_ram on /cgroup/cpu_ram type cgroup (rw,cpu,cpuset,memory)
```

The `/cgroup/cpu_ram` hierarchy contains the following subsystem parameters after the `cpu`, `cpuset`, and `memory` subsystems are attached:

```
# ls /cgroup/cpu_ram
cpu.rt_period_us                 memory.force_empty
cpu.rt_runtime_us                memory.limit_in_bytes
cpuset.cpu_exclusive             memory.max_usage_in_bytes
cpuset.cpus                      memory.memsw.failcnt
cpuset.mem_exclusive             memory.memsw.limit_in_bytes
cpuset.mem_hardwall              memory.memsw.max_usage_in_bytes
cpuset.memory_migrate            memory.memsw.usage_in_bytes
cpuset.memory_pressure           memory.move_charge_at_immigrate
cpuset.memory_pressure_enabled   memory.numa_stat
cpuset.memory_spread_page        memory.oom_control
cpuset.memory_spread_slab        memory.soft_limit_in_bytes
cpuset.mems                      memory.stat
cpuset.sched_load_balance        memory.swappiness
cpuset.sched_relax_domain_level  memory.usage_in_bytes
cpu.shares                       memory.use_hierarchy
memory.failcnt
```

The output of the `lssubsys` command shows that the `cpu`, `cpuset`, and `memory` subsystems are attached to a hierarchy mounted on `/cgroup/cpu_ram`:

```
# lssubsys -m
cpuacct
devices
freezer
net_cls
blkio
cpuset,cpu,memory  /cgroup/cpu_ram
```

The remaining subsystems are not attached to any hierarchy, as shown by their lack of a corresponding mount point.

/etc/cgconfig.conf "group" Section

```
group <cgroup_name> {
   [perm {
      task {
         uid = <task_user>;
         gid = <task_group>;
      }
      admin {
         uid = <admin_user>;
         gid = <admin_group>;
      }
   }
   <subsystem> {
      <subsystem.parameter> = <value>;
   }
}
```

The group section defines a cgroup, its access permissions, the resource subsystems that it uses, and the parameter values for these subsystems. You can define multiple groups in the `/etc/cgconfig.conf` file. The syntax is shown in the slide.

The *cgroup_name* argument defines the name of the cgroup. The `perm` (permissions) section is optional but allows you to define a `task` section and an `admin` section:

- `task` – Defines the user and group combination that can add tasks to the cgroup
- `admin` – Defines the user and group combination that can modify subsystem parameters and create subgroups

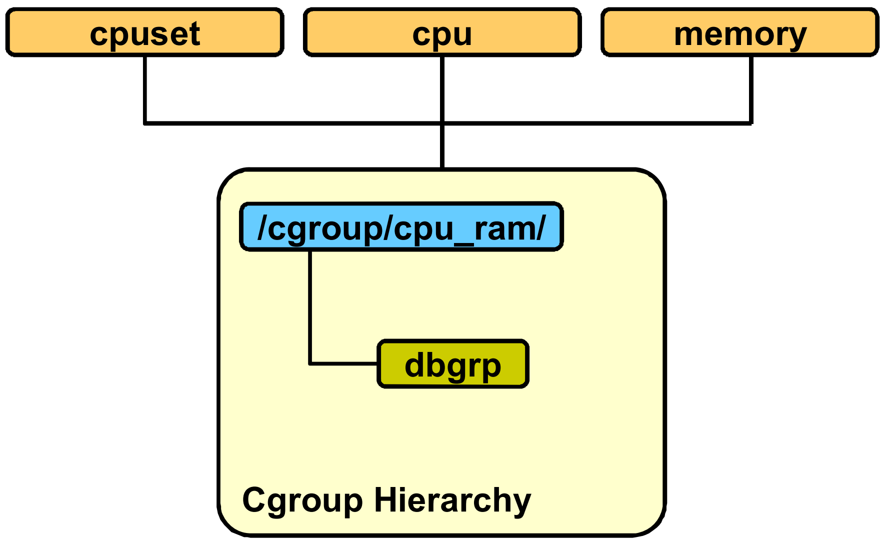
The `root` user always has permission to add tasks, modify subsystem parameters, and create subgroups.

The `subsystem` section allows you to define the parameter settings for the cgroup. You can have multiple `subsystem` sections and provide values for specific parameter settings within each section. If several subsystems are grouped in the same hierarchy, include definitions for all the subsystems in the section. For example, if the `/cgroup/cpu_ram` hierarchy includes the `cpu`, `cpuset`, and `memory` subsystems, include definitions for all these subsystems.

The following example was presented earlier in this lesson. This cgroup definition defines the root cgroup as `cpu_ram` and defines a child cgroup (`dbgrp`) for database processes. It allows the `oracle` user to add tasks and also defines certain parameters for CPU and memory usage:

```
mount {
     cpuset  = /cgroup/cpu_ram;
     cpu     = /cgroup/cpu_ram;
     memory  = /cgroup/cpu_ram;
}
group dbgrp {
     perm {
          task {
               uid = oracle;
               gid = dba;
          }
     }
     cpu {
          # Tasks in cgroup access CPU resources for 4 out of
          # every 5 seconds
          cpu.rt_period_us="5000000";
          cpu.rt_runtime_us="4000000";
     }
     cpuset {
          cpuset.mems="0";
          # Allocate CPU cores 4 through 7 to tasks in cgroup
          cpuset.cpus="4-7";
     }
     memory {
          # Allocate up to 4GB of memory to tasks in the cgroup
          memory.limit_in_bytes="4G";
          # Allocate up to 8GB of memory plus swap to the tasks
          memory.memsw.limit_in_bytes="8G";
          # Apply a soft limit of 2GB to tasks in the cgroup
          memory.soft_limit_in_bytes="2G";
     }
}
```

View of the New Cgroup Hierarchy



This slide illustrates the hierarchy created by the configuration on the previous page. Three subsystems—`cpuset`, `cpu`, `memory`—are attached to a single hierarchy, `/cgroup/cpu_ram`. This parent cgroup has one child group, `dbgrp`.

The child inherits all characteristics from the parent with the exception of those explicitly set in the configuration file. For example, the parent group contains default values:

```
# cat /cgroup/cpu_ram/cpu.rt_period_us
1000000
# cat /cgroup/cpu_ram/cpu.rt_runtime_us
950000
```

The child group contains the values set in the configuration file:

```
# cat /cgroup/cpu_ram/dbgrp/cpu.rt_period_us
5000000
# cat /cgroup/cpu_ram/dbgrp/cpu.rt_runtime_us
4000000
```

The child does not inherit the processes. Moving processes to a child control group is covered later in this lesson.

`cgcreate` and `cgdelete` Utilities

- Alternatively, you can use the `cgcreate` command to create a cgroup from the command line. The syntax is:

```
cgcreate [-t uid:gid] [-a uid:gid] -g subsystems:path
```

- For example, to create a new cgroup named `group2` in the `cpu` subsystem hierarchy:

```
# cgcreate -g cpu:group2
```

- To create a single cgroup named `cpu_mem` in two different subsystem hierarchies (`cpu` and `memory`):

```
# cgcreate -g cpu,memory:cpu_mem
```

- Use the `cgdelete` command to remove a cgroup:

```
# cgdelete cpu:group2
# cgdelete cpu,memory:cpu_mem
```

Alternatively, you can use the `cgcreate` command to create a cgroup. The syntax for `cgcreate` is:

```
cgcreate [-t uid:gid] [-a uid:gid] -g subsystems:path
```

The options are described as follows:

- `-t <uid:gid>` – Specifies the user and group IDs of those allowed to add tasks to the cgroup. The default values are the IDs of the parent cgroup.
- `-a <uid:gid>` – Specifies the user and group IDs of those allowed to modify subsystem parameters and create subgroups. The default values are the IDs of the parent cgroup.
- `-g <subsystems:cgroup-path>` – Specifies the subsystems to add and the relative path to the subsystems. You can specify this option multiple times.

For example, to create a new cgroup named `group2` in the `cpu` subsystem hierarchy:

```
# cgcreate -g cpu:group2
```

Use the `cgdelete` command to remove a cgroup:

```
# cgdelete cpu:group2
```

`cgset` Utility

- Alternatively, use the `cgset` command to set the parameters of cgroups from the command line. The syntax is:

```
cgset [-r <name=value>] <cgroup-path>
cgset --copy-from <source_cgroup-path>] <cgroup-path>
```

- Example: Suppose that the `cpuset` subsystem is attached to the `group1-web` and `group2-db` cgroups. To allocate CPU and memory nodes for the cgroups:

```
# cgset -r cpuset.cpus='0,2,4,6,8,10,12,14' group1-web
# cgset -r cpuset.cpus='1,3,5,7,9,11,13,15' group2-db
# cgset -r cpuset.mems='0' group1-web
# cgset -r cpuset.mems='1' group2-db
```

Alternatively, use the `cgset` command to set the parameters of cgroups from the command line. The syntax is as follows:

```
cgset [-r <name=value>] <cgroup-path>
cgset --copy-from <source_cgroup-path>] <cgroup-path>
```

Options:

- `-r <name=value>` – Specifies the name of the file to set and the value that is written to that file. You can use this parameter multiple times.
- `--copy-from <source_cgroup-path>` – Specifies the cgroup whose parameters are copied to the input cgroup

The following example assumes that the `cpuset` subsystem is attached to the `group1-web` and `group2-db` cgroups. It sets parameters for the cgroups, allocating CPU and memory nodes:

```
# cgset -r cpuset.cpus='0,2,4,6,8,10,12,14' group1-web
# cgset -r cpuset.cpus='1,3,5,7,9,11,13,15' group2-db
# cgset -r cpuset.mems='0' group1-web
# cgset -r cpuset.mems='1' group2-db
```

Assigning Processes to a Cgroup

- Use the `cgclassify` command to move existing processes to a cgroup. For example, to move PID `1683` into the `group1-web` cgroup:

```
# cgclassify -g cpuset:group1-web 1683
```

- Alternatively, you can also use the `echo` command:

```
# echo 1683 > /cgroup/cpuset/group1-web/tasks
```

- Use the `cgexec` command to launch a process in the specified cgroup. For example, to execute the `httpd` service in the `group1-web` cgroup:

```
# cgexec -g cpuset:group1-web httpd
```

- Alternatively, edit the associated file in `/etc/sysconfig` to allocate the service to a cgroup automatically.

There are several ways to assign a process or set of processes to a cgroup.

You can use the `cgclassify` command to move existing processes to a cgroup. The associated subsystem parameters are applied to these processes. The syntax is as follows:

```
    cgclassify [-g <subsystems:cgroup-path>] <pidlist>
```

This command moves the processes defined by the list of processes, *`pidlist`*, to the specified control groups.

Option:

- `-g <`*`subsystems`*`:`*`cgroup-path`*`>` – Specifies the cgroup where the *`pidlist`* is moved. You can use this option multiple times. If this option is not used, `cgclassify` moves *`pidlist`* to cgroup based on the rules defined in the `/etc/cgrules.conf` file.

The following example moves an existing process, PID `1683`, into the `group1-web` cgroup:

```
    # cgclassify -g cpuset:group1-web 1683
```

This command adds PID `1683` to the `tasks` file in `/cgroup/cpuset/group1-web`. You can also use the `echo` command to accomplish this:

```
    # echo 1683 > /cgroup/cpuset/group1-web/tasks
```

Use the `cgexec` command to launch a process in the specified cgroup. The syntax is as follows:

```
cgexec [-g <subsystems:cgroup-path>] <command> [arguments]
```

This `cgexec` command executes the *`<command>`* with optional *`[arguments]`* in the specified control groups. The following example executes the `httpd` service in the `group1-web` cgroup:

```
# cgexec -g cpuset:group1-web httpd
```

For services that have a configuration file in `/etc/sysconfig`, you can edit the configuration file to allocate the service to a cgroup automatically. For example, add the following line to `/etc/sysconfig/httpd`:

```
CGROUP_DAEMON="cpuset:group1-web"
```

Then start the service to automatically execute the processes in the specified cgroup:

```
# service httpd start
```

You can also set up rules so that processes are automatically assigned to particular groups by `cgred` (the cgroup rules engine daemon). The `cgred` daemon moves tasks into cgroups according to the settings in the `/etc/cgrules.conf` file.

Cgroup Rules Configuration File

- You can also set up rules so that processes are automatically assigned to particular groups by `cgred`.
- The `cgred` daemon moves tasks into cgroups according to the settings in the `/etc/cgrules.conf` file.
- The syntax to define a cgroup and associated subsystems for a user is as follows:

```
user_name [:command_name] subsystem_name [,...]
   cgroup_name
```

- For example, to assign the tasks run by the `oracle` user to the `cpu`, `cpuset`, and `memory` subsystems in the `dbgrp` cgroup:

```
oracle cpu,cpuset,memory dbgrp
```

- You can also use the `*` and `@` characters.

Use the cgroup rules definition file, `/etc/cgrules.conf`, to define the control groups to which the kernel assigns processes when they are created. Use the following syntax to define a cgroup and associated subsystems for a user:

```
user_name [:command_name] subsystem_name [,...] cgroup_name
```

The optional *`command_name`* specifies the name or absolute path name of a command. The following example assigns the tasks run by the `oracle` user to the `cpu`, `cpuset`, and `memory` subsystems in the `dbgrp` cgroup:

```
oracle cpu,cpuset,memory dbgrp
```

You can also use an asterisk (`*`) for *`user_name`* (for all users) and for *`subsystem_name`* to associate all subsystems for *`user_name`* in the cgroup. The following example specifies all users and all subsystems for the `allgrp` cgroup:

```
* * allgrp
```

Use the `@`*`group_name`* argument to define a cgroup and subsystems for all users in a group. The following example assigns the tasks run by users in the guest group to the `devgrp` cgroup:

```
@guest devices devgrp
```

Use the `service` command to start the `cgred` service after updating `/etc/cgrules.conf`.

Enabling PAM to Use Cgroup Rules

- You can configure Pluggable Authentication Modules (PAM) to use the rules that you define in the `/etc/cgrules.conf` file.
- Install the `libcgroup-pam` software package:

```
# yum install libcgroup-pam
```

- Edit the `/etc/pam.d/su` configuration file and add the following line:

```
session optional pam_cgroup.so
```

- For a service that has a configuration file in `/etc/sysconfig`, add the following line to start the service in a specified cgroup:

```
CGROUP_DAEMON="*:cgroup_name"
```

You can configure PAM to use the rules that you define in the `/etc/cgrules.conf` file to associate processes with cgroups. You must first install the `libcgroup-pam` software package:

```
    # yum install libcgroup-pam
```

Installing this package installs the `pam_cgroup.so` module in `/lib64/security` on 64-bit systems or in `/lib/security` on 32-bit systems.

Edit the `/etc/pam.d/su` configuration file and add the following line:

```
    session optional pam_cgroup.so
```

For a service that has a configuration file in `/etc/sysconfig`, add the following line to start the service in a specified cgroup:

```
    CGROUP_DAEMON="*:cgroup_name"
```

Getting Information About Cgroups

- To determine which process belongs to which cgroup, use the following command:

```
# ps -O cgroup
```

- If the PID is known, use the following command:

```
# cat /proc/PID/cgroup
```

- The `/proc/cgroups` file contains subsystem information:

```
# cat /proc/cgroup
```

- The `tree` command lists all hierarchies and cgroups:

```
# tree /cgroups
```

- The `lscgroup` command lists the cgroups on a system:

```
# lscgroup
```

Use the following command to determine which process belongs to which cgroup:

```
# ps -O cgroup
 PID CGROUP ...        COMMAND
2692 memory,cpu:/ ... su -
2700 memory,cpu:/ ... -bash
...
```

If the process ID (PID) is known, you can use the following command as well:

```
# cat /proc/PID/cgroup
```

For example, for PID `2692`:

```
# cat /proc/2692/cgroup
1:memory,cpu:/
```

You can also view the `/proc/cgroups` file to view information about the subsystems:

```
# cat /proc/cgroup
#subsys_name  hierarchy  num_cgroups  enabled
cpu           1          3            1
...
```

You can use the `tree` command (if installed) to view all the hierarchies and their cgroups:

```
    # tree /cgroups
    /cgroup/
    |---blkio
    |---cpu
    |---cpu-n-ram
    |   |---cgroup
    |   |   |---cpu.shares
    |   |   |---cpu.rt_period_us
    ...
    11 directories, 73 files
```

Use the `lscgroup` command to list the cgroups on a system:

```
    # lscgroup
    cpu,memory:/
    cpu,memory:/cgroup
    cpu,memory:/dbgrp
```

`cgget` Utility

- Use the `cgget` command to view the value of a subsystem parameter. The syntax is:

```
cgget -r cgroup_file cgroups_list
```

- For example, to view the memory statistics for the cgroup `dbgrp`:

```
# cgget -r memory.stat dbgrp
```

- Use the following syntax to list the values of all the subsystem parameters:

```
cgget -r subsystem /
```

- For example, to list all the `memory` subsystem parameter values:

```
# cgget -r memory /
```

You can use the `cgget` command to view the value of a subsystem parameter:

```
    cgget -r cgroup_file cgroups_list
```

The *`cgroup_file`* argument is the virtual file that contains the values for a subsystem. The *`cgroups_list`* argument is a list of cgroups separated with spaces. For example, to view the memory statistics for the cgroup `dbgrp`:

```
    # cgget -r memory.stat dbgrp
```

You can also use the `cgget` command to list the values of all the subsystem parameters:

```
    cgget -r subsystem /
```

For example, to list all the memory subsystem parameter values:

```
    # cgget -r memory /
```

Quiz

Cgroups enable you to do which of the following?

a. Specify the relative share of CPU time available to the tasks in a cgroup.
b. Specify memory limits for the tasks in a cgroup.
c. Suspend the tasks in a cgroup.
d. Report the CPU time and memory used by the tasks in a cgroup.

Quiz

Which of the following statements are true?

a. Cgroups can be hierarchical: each group inherits characteristics from its parent group.

b. Many different hierarchies of cgroups can exist simultaneously on a system.

c. Each time a new hierarchy is created on the system, all the tasks on the system are initially members of the default cgroup of that hierarchy, which is known as the *root cgroup*.

d. When you create a child cgroup, the child inherits the parent’s processes.

Quiz

Which of the following commands allow you to assign processes to a cgroup?

a. `cgcreate`

b. `cgclassify`

c. `cgexec`

d. `echo`

Summary

In this lesson, you should have learned how to:

- Create control groups
- Assign tasks to control groups
- Set subsystem parameter values for control groups
- Use the configuration files for control groups and control group rules
- Enable PAM to use control group rules
- Obtain information about control groups

Practice 13: Overview

This practice covers the following topics:

- Exploring the default cgroup hierarchy
- Creating cgroups in the default hierarchy
- Creating a custom cgroup hierarchy
- Using cgroups to allocate CPU resources
- Configuring cgroup rules

14

Virtualization with Linux

ORACLE

Objectives

After completing this lesson, you should be able to:

- Describe virtualization and its benefits
- Explain how Linux as a virtual guest supports the different virtualization modes
- Outline the support for Linux as a guest operating system (OS) with various virtualization solutions
- Describe the KVM hypervisor
- Use the `libvirt` tools to create and manage KVM virtual guests

This lesson begins with a general discussion of virtualization technology. There is an overview of the major virtualization products. For each virtualization product, this lesson discusses the features, the major components, the integration package and, if it exists, the cloud product associated with the virtualization product. There are many virtualization products that are not discussed in this lesson. The following URL provides a comparison of platform virtual machine packages: http://everything.explained.at/Comparison_of_platform_virtual_machines/.

This lesson concludes with a discussion of Kernel-based Virtual machine (KVM). In the practice associated with this lesson, you create and manage virtual guests in a KVM environment.

What Is Virtualization?

- The ability to run a virtual instance of:
  - An operating system
  - An application environment
  - A storage or network resource
- Virtualization provides the following benefits:
  - Better hardware utilization
  - Server consolidation
  - Faster deployments
  - Foundation for cloud computing

Virtualization is usually associated with running one or more virtual instances of an operating system on a physical host.

But virtualization is not limited to running an operating system (OS) on a virtual machine. Virtualization extends to:

- Creating virtual application environments, for example with Linux Containers or Oracle Solaris Zones
- I/O virtualization: Virtualizing a physical Host Bus Adapter (HBA) by using NPIV, which stands for N-Port ID Virtualization or virtualizing a network card by using SR-IOV, which stands for single root I/O virtualization

Virtualization provides many benefits including:

- Increasing your hardware utilization
- Consolidating your servers into fewer, more powerful, and better-utilized platforms
- The ability to quickly provide environments for development and test operations
- Providing the foundation for creating private or public cloud environments

In this topic, you explore how you can:

- Deploy Linux in a virtual environment
- Create a virtual environment with Oracle Linux

Virtualization Concepts

- The virtual environment runs on a physical host.
  - The virtual instances on the physical host are called virtual machines.
  - Virtual machines run a guest OS like Oracle Linux or Microsoft Windows.
- Virtual machines are managed by a layer called the hypervisor.
  - A type-1 hypervisor runs directly on the hardware of the physical host.
  - A type-2 hypervisor runs in the operating system installed on the physical host.

A hypervisor provides a virtualized environment for the guests running on the physical platform. The hypervisor creates and runs the virtual machines. The hypervisor can be:

- A software layer
- In the firmware of the physical host, for example, Oracle VM Server for SPARC
- Partly in the hardware, to supplement the software layer with virtualization hardware extensions (Intel VT-x or AMD-V)

Server virtualization products, such as Oracle VM Server for x86 or Microsoft Hyper-V, run directly on the physical platform. These products contain a hypervisor layer that communicates directly with the hardware. For this reason, this kind of hypervisor is called a **type-1 hypervisor** or bare-metal hypervisor.

A **type-2 hypervisor**, also known as a host-based hypervisor**,** is designed to run within a traditional operating system. This type of hypervisor adds a distinct layer to the OS and the running virtual guest becomes a third software layer. Examples of this type of virtualization include Parallels and Oracle VM VirtualBox.

KVM is considered both a type-1 and type-2 (type-1/2) hypervisor, because KVM turns the Linux kernel into a bare-metal hypervisor, but the OS running on the virtualization host is a full OS.

Virtualization Modes

- Virtualization must handle:
  - Disk devices and network devices
  - Privileged instructions
  - Memory access and paging
  - Buses, interrupts, and timekeeping
  - BIOS and the booting process
- With full virtualization, all aspects of a guest OS are emulated.
- With paravirtualization, the guest OS communicates with the hypervisor by using hypercalls.

Full Virtualization

For a fully hardware-virtualized machine (HVM) type of guest, all aspects of the virtual machine are virtualized. The guest OS running on the virtual machine does not need to know that it is running in a virtual environment and can run unmodified. Privileged instructions issued by the guest OS are trapped by the hypervisor and translated into safe, emulated instructions on the system hardware.

Full virtualization takes advantage of the virtualization hardware extensions offered by the physical server. These hardware extensions are required to run fully virtualized virtual machines.

Many products such as VMware and Microsoft's virtualization offerings use full virtualization, although each vendor also supports some type of virtualization-aware interface in the virtual machine to optimize access to I/O devices.

Paravirtualization

With paravirtualization, the guest running on the virtual machine is virtualization aware.

Paravirtualization is not an all-or-nothing mode: It represents a spectrum of support for the virtual environment.

Full Paravirtualization

Paravirtualized or PV guests run a modified version of the guest operating system, which is virtualization aware. With full paravirtualization, the PV guests do not require the presence of hardware virtualization extensions on the host processor. This type of guest is supported by just a few hypervisors, such as Xen-based virtualization products.

You can find the list of operating systems that can run as Xen PV guests at this location: http://wiki.xen.org/wiki/DomU_Support_for_Xen. This list includes major Linux distributions and Oracle Solaris 11.

Note: Xen is a type-1 hypervisor that allows guests to run either as fully paravirtualized (PV guests) or as hardware virtualized (HVM guests), with or without paravirtualized drivers. Xen is available as open source and is used by virtualization products such as Citrix XenServer and Oracle VM Server for x86. Find more information about Xen and virtualization at this site: http://wiki.xenproject.org/wiki/Xen_Overview.

Paravirtualized Drivers

You can install paravirtualized drivers in your hardware-virtualized (HVM) guest OS to optimize access to disk and network devices. Paravirtualized (PV) drivers are idealized device drivers that map operations to the real device drivers in the virtualization host. Oracle VM Server for x86, which is Xen-based, refers to this type of guest as Hardware Virtualized Machine (HVM) with paravirtualized (PV) drivers. Most virtualization vendors offer this type of paravirtualization support for selected operating systems.

In the Xen virtual environment, an HVM guest that can also support virtualized interrupts and timers in addition to running PV drivers is said to run in PVHVM mode. The Oracle Linux kernel provides this type of paravirtualization support, called paravirt-ops. Paravirt-ops is discussed further in the next slide.

You can find more information about the virtualization spectrum available for Xen-type virtual environments like Oracle VM Server for x86 and Citrix XenServer at this site: http://wiki.xen.org/wiki/Virtualization_Spectrum.

As mentioned previously, most virtualization solutions offer paravirtualization support for selected operating systems. This paravirtualization support by the major virtualization providers is highlighted throughout this lesson.

Linux and Xen Integration

- Pvops stands for paravirt-ops.
- The pvops kernel:
  - Contains all paravirtualized drivers
  - Can determine whether the underlying system supports full virtualization or paravirtual operations
  - Allows the guest running Linux to switch into PV, HVM, or HVM with PV drivers mode at boot time
- The Oracle Linux Unbreakable Enterprise Kernel (UEK) is a pvops kernel.

Paravirt-ops, or pvops, provides the support in the Linux kernel for virtual guests to run as paravirtualized guests on a Xen hypervisor, including Xen-based Oracle VM, with its Oracle VM Server for x86 platform.

With its pvops kernel, a Linux OS can boot natively on a physical host or boot as a guest OS in a virtual machine. If booting in a virtual machine, the Linux with pvops kernel can support HVM operation if the underlying hypervisor supports only full virtualization.

Several Linux distributions offer pvops support, including Oracle Linux, Red Hat Enterprise Linux (RHEL), Fedora, and Debian. RHEL provides the network and block storage paravirtualized drivers, but the tighter integration with Xen, which provides support for the management domain (called control domain or dom0), was removed starting with RHEL 6. Red Hat is now adopting KVM as its virtualization platform. You learn about dom0 with Oracle VM Server for x86 and KVM later in this lesson.

Oracle Linux 5 and later versions provide pvops support.

Running Linux in a Virtual Machine

- You can run Linux in a virtual machine with:
  - Oracle VM Server for x86
  - Microsoft Hyper-V
  - Oracle VM VirtualBox
  - VMware vSphere
  - Citrix XenServer
  - KVM
- Most virtualization providers offer integration packages to optimize Linux operations in the virtual machine.

You can run Linux as a guest operating system with most virtualization products. The list in the slide contains only a partial list of virtualization solutions. If you plan to run Oracle Linux in a virtual environment, you can access information about support for Oracle Linux with the major virtualization products at My Oracle Support, and search for Oracle Linux support policies.

Running Linux in a Virtual Environment

All virtualization products offer integration and services when running Linux as a virtual OS. This support can be provided through the emulation layer, with specialized drivers, or by support already present in the Linux distribution.

Virtualization providers offer integration packages to provide additional support such as:

- Heartbeat, which detects whether the virtual machine is running
- Integrated shutdown, where you can shut down the virtual machine from the virtualization management component
- Mouse support to help with mouse synchronization
- Messaging, which allows communication as key/value pairs between the virtual machine and the management layer

For Hyper-V, this integration package is called Linux Integration Services. For Oracle VM, it is called Guest Additions, and for VMware, it is called VMware Tools.

Oracle VM Server for X86

- Oracle VM is an enterprise-class server virtualization solution.
- Oracle VM Server for x86 is the component of Oracle VM to deploy virtual workloads on the x86 platform.
  - It includes a type-1 Xen hypervisor.
- Oracle VM Server for x86 is the x86 virtualization platform for Oracle's cloud computing solutions.

Oracle VM

Oracle VM is Oracle's server virtualization solution for both x86 and SPARC architectures and supports a variety of workloads such as Linux, Windows, and Oracle Solaris.

Oracle VM Server for x86

Oracle VM Server for x86 is part of the Oracle VM virtualization solution and provides the x86 server virtualization component for both Oracle and non-Oracle workloads. In addition to Oracle Linux and Oracle Solaris, Oracle VM Server for x86 supports Red Hat, CentOS, and SUSE Linux Enterprise Server. You can find a complete list of supported guest operating systems in the Oracle VM Release Notes, Part Number E35329-08 or newer.

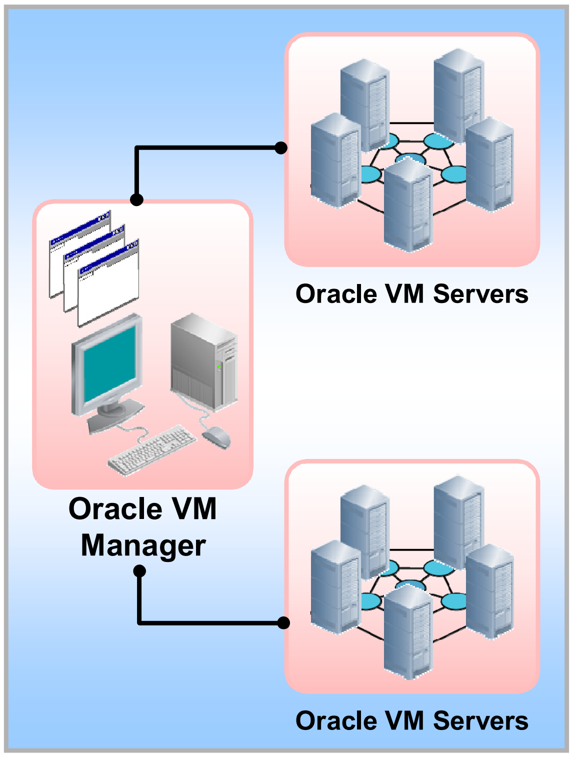
Oracle Cloud Solution

Enterprise Manager Cloud Control (EMCC) offers a solution that enables you to create, manage, and monitor a private cloud. Oracle VM Server for x86 is the virtualization platform for Oracle cloud computing service models:

- Infrastructure as a Service (IaaS), which makes available resources such as processing, networking, and storage
- Platform as a Service (PaaS), which makes available platforms onto which you can deploy applications
- Software as a Service (SaaS), which makes available an application that you can use and customize

Oracle VM Server for x86 Components

- Each Oracle VM server in the environment is a separate virtualization platform.
- The Oracle VM Manager tracks and manages the resources in your virtual environment.
- Client access to Oracle VM includes:
  - Web-based UI
  - CLI
  - Web Services API



Oracle VM Servers

Oracle VM Server for x86 is installed on a physical host. The Xen hypervisor runs directly on the host hardware. A special virtual machine, called dom0, is the host operating system running on top of the Xen hypervisor, and acts as the control domain. Each virtual guest runs in a domain called domU. Dom0, as opposed to domU-type domains, is allowed to run privileged instructions and provides drivers for the hardware on the host platform.

Oracle VM Manager

The Oracle VM Manager tracks and manages the resources available in your virtual environment. These resources include the resources in each of the Oracle VM servers, as well as the connected networks and storage. If an action is required on the resources, the Oracle VM Manager delegates an Oracle VM server to carry out the task.

You use the Oracle VM Manager to create virtual machines, and the virtual machines run on Oracle VM servers.

The diagram in the slide shows Oracle VM servers in groupings called server pools. You use the Oracle VM Manager to create pools. Most virtualization products offer a pooling feature that is the basis for other features such as high availability and live migration. With live migration, you can migrate a running virtual machine from one virtualization host to another virtualization host. Oracle VM Server for x86 offers this feature.

Oracle VM Management Interfaces

Oracle VM Server for x86 offers several ways to access and manage your environment:

- The Oracle VM Manager user interface (UI)

  You access the Oracle VM Manager by using a browser-based UI. The Oracle VM Manager UI is the most widely used management interface. From the UI, you can perform nearly all administrative functions.
- The Oracle VM command-line interface (CLI)

  With the Oracle VM CLI, you can automate configuration and operational functions by writing scripts that include embedded Oracle VM CLI commands.
- The Oracle VM Web Services

  The Oracle VM Web Services offer you a programmatic interface to Oracle VM. You can use the Representational State Transfer (REST) or Simple Object Access Protocol (SOAP) communication protocols from within Java, Python, or any other language that supports access to web services to configure, manage, or monitor your virtual environment.

Oracle VM Training

Oracle University offers several courses for Oracle VM. Go to http://education.oracle.com/virtualization and click Server Virtualization.

Linux as a Guest OS with Oracle VM Server for X86

- Supported Linux guests can run using PV, HVM, or HVM with PV driver virtualization modes.
- Install Linux from an ISO file or a template from the Oracle Software Delivery Cloud.
- Install the Oracle VM Guest Additions package in the virtual machine for bi-directional messaging and first-boot configuration.
- Manage the life cycle of your Linux guests from the Oracle VM Manager.
- Deploy Linux within a cloud infrastructure with Enterprise Manager Cloud Control.

Oracle Linux as a Guest with Oracle VM Server for x86

Oracle Linux is very well suited for running in a virtual machine with Oracle VM Server for x86. Oracle Linux supports paravirtualization (PV) modes and full virtualization mode (HVM).

Oracle VM Server for x86 has been designed and tested to handle mission-critical enterprise workloads.

Oracle VM Guest Additions

The Oracle VM Guest Additions form a bi-directional messaging channel between Oracle VM Manager and the guest OS. This communication channel allows first-boot installation configuration. Nearly all of the Oracle VM templates available from the Oracle Software Delivery Cloud already have Guest Additions installed.

Oracle VM Server for x86 and Enterprise Manager Cloud Control

You can use Oracle Enterprise Manager Cloud Control to build cloud solutions based on Oracle VM Server for x86. Oracle Enterprise Manager Cloud Control extends the functionality of Oracle VM by offering centralized monitoring, self-service provisioning, and configuration management, including a facility to store cloud resources such as templates and patches in a software library. Find out more about Oracle Enterprise Manager Cloud Control at http://www.oracle.com/technetwork/oem/cloud-mgmt-496758.html.

Linux as a Guest OS with Oracle VM VirtualBox

Oracle VM VirtualBox:

- Is the most popular virtualization software
- Incorporates a type-2 (host-based) hypervisor
- Runs on Mac OS X, Windows, Linux, or Oracle Solaris
- Supports several flavors of Linux, running as guest OS on a virtual machine:
  - Oracle Linux, RHEL, Ubuntu, Debian, SUSE, Mandriva, Fedora, CentOS, Oracle Solaris, and more
- Is a great choice for desktop virtualization

Oracle VM VirtualBox is a general-purpose virtualization product for x86 hardware. It is a type-2 hypervisor virtualizer that you install in an already existing OS, such as Windows, Linux, MAC OS X, or Oracle Solaris. It provides full virtualization only. Though you can use Oracle VM VirtualBox for server provisioning, its desktop extension pack makes it an excellent choice for desktop virtualization. You can find more information about VirtualBox extension packs at this location: https://www.virtualbox.org/manual/ch01.html#intro-installing.

VirtualBox supports several operating systems running as guests in a virtual machine. Find the list of supported guest operating systems at https://www.virtualbox.org/wiki/Guest_OSes.

VirtualBox is a great choice for evaluating Oracle Linux as a platform for running your business applications, because you can run Oracle Linux in a VirtualBox virtual machine on a desktop. At this site, http://www.oracle.com/technetwork/articles/servers-storage-admin/evaluating-linux-vb-1934676.html, you can find information about evaluating Oracle Linux, plus a link to an Oracle Linux VM download for VirtualBox.

VMware vSphere

- vSphere is a family of products providing VMware's virtualization platform.
- ESXi is vSphere's bare-metal (type-1) hypervisor.
  - You install ESXi on a 64-bit processor host.
- vCenter is the management layer for vSphere.
  - You install vCenter on a 64-bit Microsoft Windows platform.
- vSphere is the building block for VMware's vCloud offering.

VMware vSphere

VMware vSphere is a virtualization platform that comprises several virtualization products, associated tools, and components.

VMware ESXi

A VMware virtualization environment includes one or more ESXi physical hosts that form the virtualization layer. The ESXi host is a type-1 or bare-metal hypervisor. You attach a storage and networking infrastructure to your ESXi hosts and use the vSphere management layer to make these resources available to deploy virtual machines.

VMware vCenter Server

vCenter is the management layer for vSphere. It provides a single point of control to manage the storage and network resources configured to the ESXi hosts and to assign these resources to the virtual machines. vCenter also provides:

- User access control by connecting to Active Directory
- A repository of management information in an Oracle, Microsoft SQL Server, or IBM DB2 database. Stored information includes host and virtual machine configurations, user permissions and roles, resource inventory such as storage resources, and performance statistics.

You install the vCenter Server on a 64-bit Microsoft Windows platform. You can have a single, stand-alone instance of vCenter or join the instances into a group. You can also deploy your vCenter instances into a highly available configuration.

VMware Cloud Offering

With VMware vCloud, you can build a vSphere-based private cloud. You can find more information about vCloud at this location: http://www.vmware.com/products/vcloud-suite/.

Linux as a Guest OS with VMware vSphere

- vSphere provides several client interfaces to create virtual machines:
  - A vSphere local client and web client
  - Several CLI tools (for example `vmkfstools` and `vmware-cmd`)
  - A vSphere Web Services API
- Provision a guest OS in the virtual machine by:
  - Installing from media
  - Deploying from a template
  - Cloning an existing virtual machine
- VMware offers paravirtual drivers: PVSCSI and VMXNET 3

vSphere Clients

You manage your vSphere environment by accessing the vCenter server using vSphere clients:

- vSphere Client: A locally installed program to connect remotely to the vCenter Server
- vSphere Web Client: A web program to connect remotely to the vCenter Server
- vSphere CLI: Includes commands like `vmkfstools`, a tool to manage virtual disks and physical storage on an ESXi server, and `vmware-cmd`, a command used to perform virtual machine operations remotely, like starting or stopping a virtual machine
- vSphere PowerCLI: Based on Microsoft PowerShell, provides `cmdlets` to create, manage, and monitor virtual machines
- vSphere Web Client SDK: Provides a programmatic interface using web services to create customized tasks for your vSphere environment

Creating a Virtual Machine from Installation Media

Using the vSphere client or web client, you can:

- Create a virtual machine
- Assign it a name
- Specify a guest operating system, for example, Oracle Linux (64-bit)

- Create a network connection
- Create a virtual disk from an already configured storage source (datastore). A datastore is storage space across multiple ESXi servers.

After creating the virtual machine, connect to an ISO image on your local disk and install the OS in the virtual machine.

Creating a Virtual Machine from a Template

Using the vSphere client or web client, you can deploy a virtual machine from a template or clone an existing virtual machine. The new virtual machine inherits the software and configured properties present in the template or virtual machine. You can further customize your Linux guest during the template deployment or clone operation.

Paravirtualization with vSphere

VMware offers these paravirtual drivers:

- PVSCSI, a paravirtual SCSI adapter
- VMXNET 3, a paravirtual network driver

You select PVSCSI adapters to achieve greater throughput and lower CPU utilization for storage operations. PVSCSI adapters yield the best performance in SAN environments.

VMXNET is a paravirtual network interface that is designed to reduce the I/O virtualization overhead and therefore increase performance.

In most cases, you obtain paravirtual drivers by installing VMware Tools in your virtual machine. VMware Tools is VMware’s integration package for virtual machines.

Microsoft Hyper-V and Windows Azure

- Hyper-V is a Windows Server role that turns the server into a type-1 virtualization provider.
- In addition to a UI, Hyper-V provides Windows Powershell `cmdlets` to manage the virtual environment.
- Linux Integration Services (LIS) provides integration between the OS running in the Hyper-V virtual machine.
  - The Hyper-V LIS package is already built in Oracle Linux 6.4 and newer releases.
- Windows Azure is Microsoft’s cloud solution, based on Hyper-V.

Microsoft Hyper-V

Microsoft Hyper-V is a Windows server role that turns a Windows server, such as Windows Server 2012 R2 into a type-1 hypervisor. Hyper-V requires a 64-bit processor and hardware-assisted virtualization.

Manage Hyper-V with:

- Hyper-V Manager, a GUI tool
- A Hyper-V module for Windows Powershell, providing `cmdlets` for management tasks.

Hyper-V provides a software package called Linux Integration Services (LIS) that provides integration between the OS running in the Hyper-V virtual machine and the physical host. This package is already available in Oracle Linux, starting with Oracle Linux 6.4. In addition to timekeeping, virtual machine heartbeat detection, and integrated shutdown features, the package provides an information exchange capability between the running Linux virtual machine and the Hyper-V server. This functionality is similar to the messaging function available for Oracle Linux with the Oracle VM Guest Additions package.

Windows Azure

Windows Azure is a cloud platform that allows you to quickly build, deploy, and manage scalable solutions.

As part of Windows Azure Compute services, you create virtual machines that use either Windows Server or the Linux operating system, including Oracle Linux. If the applications that you want to deploy run on Hyper-V, they also run on Windows Azure.

Windows Azure is based on a customized version of Microsoft Hyper-V called the Windows Azure Hypervisor. This hypervisor provides the virtualization services. Windows Azure contains additional components that manage the storage and computing resources in the Microsoft datacenters that are hosting Windows Azure.

Linux as a Guest OS with Microsoft Hyper-V and Windows Azure

- The Linux guest OS runs in an isolated environment called a child partition.
  - The parent partition (or Management OS) loads the hypervisor.
- Hyper-V offers a form of paravirtualization called Enlightened I/O.
  - Recent releases of many Linux distributions contain the paravirtual drivers for Hyper-V's Enlightened I/O.
- To build Linux virtual machines in Azure, access the Management Portal and select a template from the Image Gallery.

Hyper-V Partitions

With Hyper-V, a virtual guest OS runs in a child partition, which isolates the guest OS from other partitions on the physical host. The child partition is also called a virtual machine. The parent partition, also called the Management OS, loads the hypervisor and contains the virtualization stack and the virtualization tools. The virtualization stack in the parent partition has direct access to the hardware devices.

Paravirtualization with Hyper-V

Hyper-V supports a form of paravirtualization called Enlightened I/O, for networking, storage, graphics and other input devices. Enlightened I/O provides increased performance by bypassing a layer of emulated hardware. The guest OS must support Enlightened I/O. The drivers for Enlightened I/O are now delivered directly in recent Linux distributions, including Oracle Linux, Red Hat, and CentOS. For other distributions, you can obtain the Enlightened I/O drivers by downloading the Linux Integration Services from the Microsoft Download Center at http://www.microsoft.com/en-us/download/default.aspx.

How to Create an Oracle Linux Virtual Machine with Windows Azure

After signing up for a particular program (for example, the Compute services program), create a virtual machine by using an image from the Image Gallery in the Windows Azure Management portal. Or you can upload a `.vhd` disk image as a file to Windows Azure. This vhd-type disk must already contain a bootable OS.

You can create stand-alone virtual machines or place your virtual machines in the same cloud service to allow the virtual machines to communicate or to provide load balancing to your applications.

Linux as a Virtualization Provider

- With Linux Containers (LXC)
  - Containers allow you to run multiple user-space versions of Linux on the same host without the need of a hypervisor.
- With KVM, your Linux host becomes a type-1/2 hypervisor.
  - Guests run as ordinary user-space processes.
  - Guests are hardware accelerated and fully virtualized.
- KVM requires hardware virtualization extensions on Intel or AMD physical platforms.
- Use `libvirt` API and tools to manage KVM guests.

With Oracle Linux, there are two ways to configure virtual environments, Linux Containers (LXC) and KVM.

Linux Containers (LXC)

LXC is a virtualization technology that allows you to partition system resources on the control host into virtual instances called containers, which have their own process and network space. This technology provides isolation for the application(s) running in the container, while allowing resource adjustment to the container. The container environment is similar to a standard Linux OS but depends on the control host's kernel. LXC is covered in a separate lesson titled "Linux Containers (LXC)."

KVM

KVM was first developed at Qumranet, which Red Hat bought in 2008. With KVM, your Oracle Linux host becomes a type-1/2 hypervisor. KVM requires hardware virtualization extensions on your Intel or AMD physical platform. KVM is a full virtualization solution, mainly for x86 hardware.

- Full virtualization means that the virtualization solution provides full emulation for the guest operating system running in the virtual machine including emulation for networking, interrupts and timers, and even provides an emulated BIOS.
- With full virtualization, you can run unmodified versions of the guest operating system in the virtual machine, either Linux or Windows.

`libvirt`

- `libvirt` is a toolkit to manage your virtualized environment.
- `libvirt` is not specific to KVM:
  - Other virtualization technologies such as LXC, Xen, and VirtualBox use `libvirt`.
- `libvirt` offers an API and tools such as:
  - `virt-install`: Create KVM guests
  - `virsh`: Perform operational tasks on KVM guests
  - `virt-manager`: UI to manage KVM guests
- Guest metadata is stored in XML format.

KVM uses `libvirt`, an API and toolkit, to manage your virtualized environment. The `libvirt` toolkit can interact with several virtualizers, including these hypervisors:

- KVM/QEMU
- Xen hypervisor
- Linux Container System (LXC)
- VirtualBox hypervisor
- VMware ESX hypervisor

Find the complete list at http://libvirt.org/.

`libvirt` provides local and remote management of virtual machines. For secure remote management, you can use `libvirt` with TLS encryption and x509 certificates. For authentication, you can select Kerberos and SASL. Simple Authentication and Security Layer (SASL) allows applications to exchange information securely.

The `libvirt` toolkit provides the tools and APIs to manage virtual machines: Provision, start, stop, modify, migrate, monitor, and delete.

With `libvirt` tools, the virtual machine or domain metadata is described using the XML format. By default, the virtual machine XML configuration files reside in the `/etc/libvirt/qemu` directory.

Installing KVM and `libvirt`

- The KVM modules are present in mainline Linux, as of `2.6.20`.
  - With Oracle and Red Hat Linux, KVM is already available.
- Get started by installing the following `libvirt` packages:

```
# yum install qemu-kvm virt-manager libvirt libvirt-
   python python-virtinst  virt-viewer libguestfs-tools
```

- Or if installing Oracle Linux:
  - Select the Virtualization Host server role.
  - Customize your host by selecting the Virtualization package group and the General Purpose Desktop package group.

KVM support is already present in Red Hat and Oracle Linux. If using other distributions, consult the documentation for a list of packages to install for KVM support.

The package providing this support is `qemu-kvm`.

To verify that `kvm` is present, use the `lsmod` command:

```
# lsmod | grep kvm
kvm_intel              55356  3
kvm                   372790  1 kvm_intel
```

For administering your KVM guests, install the `libvirt` tools.

Installing the Packages Separately

The selection of packages to install depends on what you plan to do. Install the following packages to get started with `libvirt`:

```
# yum install qemu-kvm virt-manager libvirt libvirt-python
python-virtinst virt-viewer libguestfs-tools
```

- `libvirt`: The API library for interacting with the KVM hypervisor. The `libvirtd` daemon is part of this package. This daemon runs on the virtualization server and performs management tasks for virtualized guests.
- `python-virtinst`: Required for the `virt-install` command to create virtual machines

- `libvirt-python`: For applications written in the Python programming language
- `virt-manager`: Provides a graphical tool for administering virtual machines

Using the Software Groups

You can use the `yum` command to install the virtualization software groups to your existing Linux host:

- Virtualization Client: Clients for installing and managing virtualization instances
- Virtualization Platform: Provides an interface for accessing and controlling virtualized guests and containers
- Virtualization Tools: Tools for offline and live virtual image management

Example:

```
# yum groupinstall “Virtualization Platform”
```

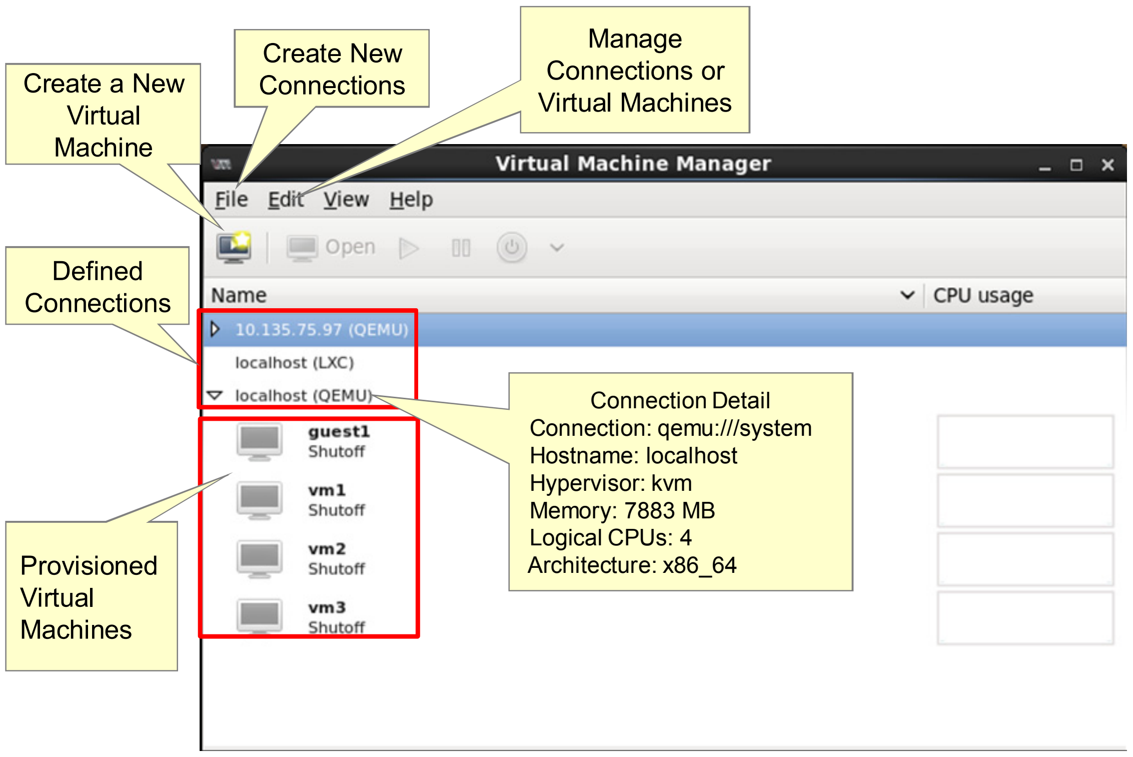
Adding Virtualization Support When Installing Linux

If installing Oracle Linux:

- Select the Virtualization Host server role in the package selection screen.
- Click the “Customize now” button.
- Optionally, select the Virtualization Tools package group.
- Select the General Purpose Desktop package group if it is not already selected.

The General Purpose Desktop package group is required if you plan to use the `virt-manager` graphical user interface.

Getting Started with `virt-manager`: Connections



Connection Detail
Connection: qemu:///system
Hostname: localhost
Hypervisor: kvm
Memory: 7883 MB
Logical CPUs: 4
Architecture: x86_64

Use the `virt-manager` command to launch the graphical user interface.

Because `virt-manager` supports several types of hypervisors, you must create one or more connections to specify which hypervisor to use for your virtual machines.

You can access the hypervisor locally or remotely.

Creating a Connection

When creating a connection to the KVM hypervisor, you specify the hypervisor as QEMU/KVM.

Note: QEMU is used with KVM to provide emulation for components like a NIC, disk device, or graphics adapter.

If accessing the hypervisor remotely, you must specify the necessary `ssh` parameters for the connection to succeed.

Creating a Virtual Machine

After creating the connection, you can provision virtual machines for that connection using one of the following two methods:

- Click the “Create a new virtual machine” icon, located in the toolbar.
- Highlight the target connection and select New from the shortcut menu.

You can also create new virtual machines with the `virsh` or `virt-install` commands. If you do not specify parameters with the `virsh` command, you start an interactive session:

```
# virsh
Welcome to virsh, the virtualization interactive terminal.

Type:  'help' for help with commands
       'quit' to quit
virsh # version
Compiled against library: libvirt 0.10.2
Using library: libvirt 0.10.2
Using API: QEMU 0.10.2
Running hypervisor: QEMU 0.12.1
virsh # quit
```

Example of creating a virtual machine with `virt-install`:

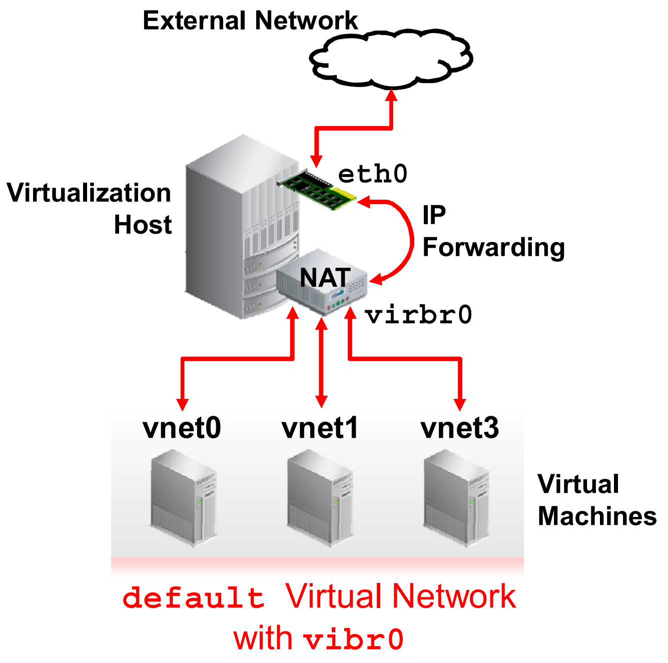
```
# virt-install --connect qemu:///system --virt-type kvm --name
test --ram 500 --disk
path=/var/lib/libvirt/images/test.img,size=4 --cdrom
/home/user01/Downloads/OracleLinux-R6-U5-Server-x86_64-dvd.iso -
-os-variant oel6

Starting install...
Allocating 'test.img'
| 4.0 GB     00:00
Creating domain...
|    0 B     00:00
Connected to domain test
Escape character is ^]

Google, Inc.
Serial Graphics Adapter 10/14/11
...
```

Virtual Networks

- A virtual network acts like a virtual network switch.
- The `default` virtual network is created when the `libvirt` daemon is started for the first time.
- Guest NICs connect to the virtual network `virbr0` bridge.
- Use `virt-manager` or `virsh net-*` commands to manage virtual networks.



Before creating your virtual guests, you must create virtual networks to enable networking for the guests.

Virtual Network Using NAT and IP Forwarding

Virtual machines connect to virtual networks. When the `libvirtd` daemon starts for the first time, it defines a virtual network called `default`. This virtual network uses a virtual network switch, or bridge, called `virbr0`, to which the guests attach.

The diagram in the slide shows the virtual interfaces, `vnet0`, `vnet1`, and `vnet3`, connecting to the virtual switch, `virbr0`, which is a bridge. These virtual interfaces are the back-end NICs created on the virtualization host, and they correspond to virtual network interfaces in the virtual guests. The `virbr0` bridge is not attached to any physical NIC on the virtualization host. Instead, NAT and IP forwarding are used to forward packets from the virtual guests to the external network. You can find more information about virtual networking at this site: http://wiki.libvirt.org/page/VirtualNetworking.

Virtual Network in Routed Mode

In addition to virtual networks using NAT and IP forwarding, you can set up a virtual network that uses a virtual switch (or bridge) in routed mode, where the virtual switch is connected to a physical NIC on the virtualization host.

Private Virtual Network

If virtual guests running on a virtualization host need to communicate with each other but do not require a connection to an external network, you can create a virtual network in private or isolated mode. This type of network still allows virtual guests to acquire an IP address but does not support traffic into or out of the virtualization host.

Working with Storage

- Use `libvirt` tools to create storage pools and allocate volumes for virtual guests.
  - Storage pools can be iSCSI, NFS, disk devices, file systems or directories, or even LVM Volume Group.
  - Volumes support several format types, including raw, `qcow2`, and `vmdk`.
- You can also assign storage outside of storage pools.
  - Ensure that the storage (for example, an NFS share) is available when the virtual guest boots.
- Consider using `virtio`, which offers a paravirtualized driver to accelerate disk operations.

You do not need to create storage pools to assign storage to your virtual guests. If you use storage pools, the `livbirt` tools ensure that the assigned storage residing in a storage pool is available when you attempt to boot your virtual guest.

`libvirt` supports several storage pool types. A few of these types are described here:

- Directory pool: You create volumes in the directory, selecting among several format types, such as `qcow2`, which is a file format used by QEMU. `qcow2` uses a disk optimization scheme to delay storage allocation until needed. The `vmdk` file format listed in the slide is a file format developed by VMware.
- iSCSI pool: The storage pool is created on an existing iSCSI target.
- NFS pool: The storage pool is created on an existing NFS share.
- Disk device: You can use a disk device, such as a USB stick, as a storage pool. A volume is created in the storage pool by creating partitions on the disk. For example, you create a storage pool with device `/dev/sdb` and create a volume as `/dev/sdb1`.

`virtio` is a paravirtualization standard for both storage and networking. The `virtio` paravirtualized drivers reduce I/O latency. The guest must support paravirtualization to use `virtio`.

To enable virtio operations, you specify virtio when allocating a virtual disk (or a virtual network interface), or you can change the model type for the virtual disk to virtio at a later time.

For example, if you examine XML file vm3.xml in /etc/libvirt/qemu (on the virtualization host) that describes virtual guest vm3, you find the following disk section under the devices section:

```
<disk type='file' device='disk'>
   <driver name='qemu' type='raw' cache='none'/>
   <source file='/var/lib/libvirt/images/vm3.img'/>
   <target dev='vda' bus='virtio'/>
   <address type='pci' domain='0x0000' bus='0x00' slot='0x06'
           function='0x0'/>
 </disk>
```

Note that virtio is selected as the bus type.

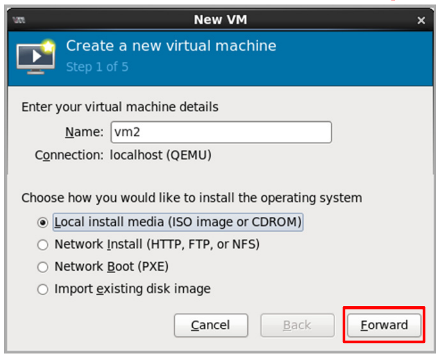
Creating Virtual Machines

- Using `virt-install`

```
virt-install \
     --name=vm1 \
     --file=/var/lib/libvirt/images/vm1.dsk \
     --file-size=8 --nonsparse
     ...
     --network bridge=br0
     --os-type=linux --os-variant=oel6
```

- Using `virt-manager`

Step 1



Enter your virtual machine details

Name: vm2

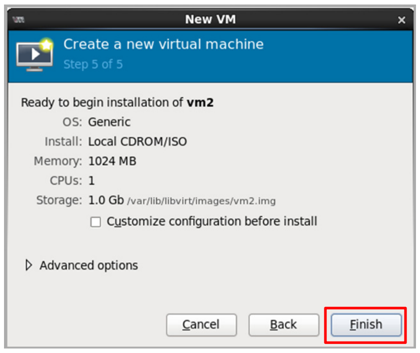
Connection: localhost (QEMU)

Choose how you would like to install the operating system

- [✓ Local install media (ISO image or CDROM)]
- Network Install (HTTP, FTP, or NFS)
- Network Boot (PXE)
- Import existing disk image

...

Step 5



Ready to begin installation of **vm2**

OS: Generic
Install: Local CDROM/ISO
Memory: 1024 MB
CPUs: 1
Storage: 1.0 Gb /var/lib/libvirt/images/vm2.img

☐ Customize configuration before install

Use the `virt-install` command from the command line or in a script to create virtual machines. You need `root` privileges to use the `virt-install` command.

Make sure that you use the appropriate `os-type` and `os-variant` for the OS in your virtual machines. Use the following command to list the `os-variant`:

```
#  virt-install --os-variant list
```

You can create a virtual machine using the following installation methods:

- ISO image or CDROM
- Network install: You must make the installation tree accessible by using HTTP, FTP, or NFS.
- PXE network install: For this installation method to succeed, the virtual machine must be able to acquire an IP address.
- Using an existing disk image: The disk image must contain an already installed, bootable OS.

Hot plugging operations are allowed with KVM when also supported by the guest OS. This means that you can dynamically increase the number of virtual CPUs (vCPUs), or add virtual disks or network interfaces to your virtual machines. Virtual disks hotplug operations are supported for only some bus types.

Managing the Life Cycle of a Virtual Machine

The following operations are supported for virtual machines:

- Pause
- Start and shutdown, including force shutdown
- Cloning
- Migration
- Delete

Shutdown and Pause Operations

You can use the pause function before attempting to clone your virtual machine. Use the resume function to un-pause the virtual machine.

To shut down a virtual machine using a `libvirt` tool, the `acpid` daemon must be running in the guest OS. If not running, use force shutdown from `virt-manager` or shut down the virtual machine from its console.

Cloning

The cloning operation copies the disk images of an existing machine to create a new virtual machine. It automatically assigns a new MAC address and updates the virtual machine unique ID called UUID. You can clone by using the `virt-manager` UI or use the `virt-clone` command. Pause the virtual machine before attempting to clone it.

Migration

You can migrate a virtual machine to another virtualization host, if both the source and target hosts have the same architecture. You must also ensure that both hosts have access to the guest storage. Find more information about KVM migration at this location: http://www.linux-kvm.org/page/Migration

Delete Operation

When you delete a virtual machine, you have the choice to retain the virtual machine's storage files.

Quiz

What statements are true regarding paravirtualized drivers?

a. Paravirtualized drivers are idealized drivers installed in the virtual guest.
b. Paravirtualized drivers optimize I/O operations.
c. Paravirtualized drivers map operations to real drivers on the virtualization host.
d. Most virtualization solutions offer paravirtualized drivers.
e. All of the above.

Summary

In this lesson, you should have learned how to:

- Describe virtualization and its benefits
- Explain how Linux as a virtual guest supports the different virtualization modes
- Outline the support for Linux as a guest OS with various virtualization solutions
- Describe the KVM hypervisor
- Use the `libvirt` tools to create and manage KVM virtual guests

Practice 14: Overview

This practice covers the following topics:

- Preparing your virtualization host for KVM operations
- Starting the Virtual Machine Manager
- Creating a virtual machine with an existing virtual disk
- Accessing the virtual machine's console
- Cloning the virtual machine's disk
- Adding paravirtualization to the virtual machine's NIC operations

15

Linux Containers (LXC)

Objectives

After completing this lesson, you should be able to:

- Describe the purpose of Linux Containers
- Describe container configuration parameters
- Install the required Linux Container software packages
- Describe Linux Container template scripts
- Create a Linux Container by using the Oracle template script
- Use Linux Container utilities to start and stop a container
- Use additional Linux Container utilities
- Install an Oracle VM template as a base environment
- Create a Linux Container from an existing rootfs

Linux Containers: Introduction

- Linux Containers (LXC) are the next step up from cgroups.
- LXC provides application and operating system isolation.
- Containers do not require a hypervisor.
- They are similar to Oracle Solaris Zones in that:
  - They both provide virtualization at the application level
  - One kernel is shared by many zones or containers
- Containers rely on the cgroups functionality.
- Containers rely on namespace isolation that is similar to `chroot`.
- Processes within a container can have their own process ID space, file system structure, and network interfaces.

Linux Containers (LXC) are the next step up from cgroups for using system resources more efficiently. Whereas cgroups allow you to isolate system resources, containers provide application and operating-system isolation. Containers allow you to run multiple user-space versions of Linux on the same host without the need of a hypervisor. You can isolate environments and control how resources are allocated without the virtualization overhead.

Linux Containers are similar to Oracle Solaris Zones in that they are virtualization at the application level, above the kernel. One operating system kernel is shared by many zones or containers. Because the kernel is shared, you are limited to the modules and drivers that it has loaded. The difference between zones and containers is more at the implementation level and in the way it is integrated into the operating system.

Containers rely on the cgroups functionality but also rely on namespace isolation, similar to `chroot`. Within each container, processes can have their own private view of the operating system with its own process ID space, file system structure, and network interfaces.

Containers can be useful for:

- Running different copies of application configurations on the same server
- Running multiple versions of Oracle Linux on the same server
- Creating sandbox environments for testing and development
- Controlling the resources allocated to user environments

You can also use Btrfs subvolumes as a way to quickly create containers.

Linux Container Resource Isolation

- Containers provide:
  - Resource management through control groups (cgroups)
  - Resource isolation through namespaces
- A user-space container object provides resource isolation and control for an application or a system.
- Containers rely on a set of kernel functionalities to be active.
  - LXC is fully functional beginning with kernel `2.6.29`.
- Use the `lxc-checkconfig` command to get information about your kernel configuration.
  - This utility reads `/proc/config.gz` or `/boot/config*`.
- Define the resources to isolate for an application.
- Running a system within a container is easier than an application.

Containers provide resource management through control groups (cgroups) and resource isolation through namespaces. They use these functionalities to provide a user-space container object that can then provide full resource isolation and resource control for an application or a system.

Containers rely on a set of kernel functionalities, such as namespaces, control groups, networking, and file capabilities, to be active. Beginning with kernel `2.6.29` or later, LXC is fully functional. Running with an older kernel version causes LXC to work with a restricted number of functionalities, or can even cause LXC to fail. You can use the `lxc-checkconfig` command to get information about your kernel configuration. This utility reads the `/proc/config.gz` file if it is found, or reads the `/boot/config*` file for your active kernel version and displays the status (enabled/disabled) of kernel functionalities.

Before running an application in a container, identify the resources to isolate. By default, the process IDs, the SysV IPCs, and the mount points are isolated. With the default configuration, you can run a simple shell command within a container. When running an application, for example `sshd`, provide a new network stack and a new host name. To avoid container conflicts, specify a `root` file system for the container. Running a system in a container is easier than running an application because you do not care about specific resource isolation when running a system. Everything is isolated when running a system.

The following is sample output from the `lxc-checkconfig` command:

```
# lxc-checkconfig
Kernel configuration not found at /proc/config.gz; search...
Kernel configuration found at /boot/config-3.8.13-26.1.1...
--- Namespaces ---
Namespaces: enabled
Utsname namespaces: enabled
Ipc namespaces: enabled
Pid namespaces: enabled
User namespaces: missing
Network namespaces: enabled
Multiple /dev/pts instances: enabled

--- Control groups ---
Cgroup: enabled
Cgroup clone_children flag: enabled
Cgroup device: enabled
Cgroup sched: enabled
Cgroup cpu account: enabled
Cgroup memory controller: enabled
Cgroup cpuset: enabled

--- Misc  ---
Veth pair device: enabled
Macvlan: enabled
Vlan: enabled
File capabilities: enabled
...
```

Notice that the UEK R3 (`3.8.13`) does not have the necessary support for user namespaces. Without this support, you cannot set `lxc.id_map` entries, which allow you to run a container under another UID instead of `root`. Support for user namespaces is available beginning with kernel version `3.13`.

Linux Container Configuration File

Create a file to define the system resources for the container.

- Architecture and Utsname:
  - Architecture and host name for the container
- Network:
  - Virtual network interfaces
- TTY:
  - Pseudo tty, console output, and number of available ttys
- Mount points:
  - Different places to be mounted
- Root file system:
  - Root file system for the container
- Control groups:
  - Configuration for the different cgroup subsystems

Creating a container defines a set of system resources to be virtualized and isolated by a process that uses the container. As mentioned, PIDs, SysV IPCs, and mount points are virtualized and isolated by default. Other system resources are shared across containers until they are explicitly defined in the Linux Container configuration file. For example, if a network is defined in the configuration file, a network stack is created for the container. Otherwise, the container shares the same network stack with the host that creates the container.

Use the Linux Container configuration file to define the different system resources to be assigned to the container. The following system resources are currently supported:

- **Architecture** – Specifies the architecture for the container
- **Utsname** – Specifies the host name for the container
- **Network** – Specifies how the network is virtualized in the container. Network virtualization works at layer 2. You can define multiple virtual network interfaces.
- **TTY** – Specifies the pseudo tty, console output, and available ttys
- **Mount points** – Specifies the different places to be mounted
- **Root file system** – Specifies the root file system for the container, which can be different from the root file system for the host system
- **Control groups** – Specifies the configuration for the different cgroup subsystems

You can define the following configuration keys. Define each key on a separate line using the `key = value` format.

Architecture / Utsname

- `lxc.arch`: Specifies the architecture for the container. Valid values are `x86`, `i686`, `x86_64`, and `amd64`.
- `lxc.utsname`: Specifies the host name for the container

Network

- `lxc.network.type`: Specifies the type of network virtualization to be used for the container. Valid values are `empty`, `veth`, `vlan`, `macvlan`, and `phys`.
- `lxc.network.flags`: Specifies an action to do for the network. A value of `up` activates the network.
- `lxc.network.link`: Specifies the interface to be used for the real network traffic
- `lxc.network.name`: Specifies the virtual network interface in the container
- `lxc.network.hwaddr`: Specifies the MAC address of the container's network interface
- `lxc.network.ipv4`: Specifies the IPv4 address for the virtualized interface. You also specify the broadcast address on the same line, immediately after the IPv4 address.
- `lxc.network.ipv4.gateway`: Specifies the IPv4 address to use as the gateway in the container
- `lxc.network.ipv6`: Specifies the IPv6 address for the virtualized interface
- `lxc.network.ipv6.gateway`: Specifies the IPv6 address to use as the gateway in the container
- `lxc.network.script.up`: Specifies a script to be executed after creating and configuring the network used from the host side

TTY

- `lxc.pts`: Specifies the number of pseudo ttys allowed for a `pts` instance
- `lxc.console`: Specifies a path to a file where the console output is written
- `lxc.tty`: Specifies the number of ttys available to the container

Mount Points

- `lxc.mount`: Specifies a file location `fstab` format that contains the mount information
- `lxc.mount.entry`: Specifies a mount point corresponding to a line in the `fstab` format

Root File System

- `lxc.rootfs`: Specifies the `root` file system for the container. If it is not specified, the container shares its `root` file system with the host system.
- `lxc.rootfs.mount`: Specifies where to recursively bind `lxc.rootfs` before pivoting. This ensures success of the `pivot_root(8)` system call.
- `lxc.pivotdir`: Specifies where to pivot the original root file system under `lxc.rootfs`. The default is `/mnt`.

Control Groups

- `lxc.cgroup.[subsystem name]`: Specifies the control group subsystem parameter to be set. An example is `lxc.cgroup.cpuset.cpus`.

See `man lxc.conf` for complete descriptions. See `/usr/share/doc/lxc/` for examples.

Required Linux Container Packages

- Install the `libcgroup` software package, configure the `cgconfig` service to start at boot time, and start the `cgconfig` service:

```
# yum install libcgroup
# chkconfig cgconfig on
# service cgconfig start
```

- Install the LXC packages:

```
# yum install lxc lxc-libs
```

- Install the Btrfs package to use the Btrfs file system for container repositories:

```
# yum install btrfs-progs
```

The Linux Containers feature takes the cgroups resource management facility as its basis and depends on the cgroup file system being mounted. This requires that you install the `libcgroup` software package, configure the `cgconfig` service to start at boot time, and start the `cgconfig` service. Use the following commands to perform these tasks:

```
# yum install libcgroup
# chkconfig cgconfig on
# service cgconfig start
```

The LXC packages are also required to create and use Linux Containers. Install these packages as follows:

```
# yum install lxc lxc-libs
```

The Btrfs file system can be used for a container repository. New instances can then be cloned and spawned quickly, without requiring significant additional disk space. Btrfs allows you to create a subvolume that contains the base template for the containers, and to create containers from writable snapshots of the template. Install the `btrfs-progs` package to use Btrfs as follows:

```
# yum install btrfs-progs
```

Linux Container Template Scripts

- Template scripts define the settings for the system resources that are assigned to a container.
- Several template scripts are available:

```
# ls /usr/share/lxc/templates
lxc-alpine    lxc-busybox  lxc-opensuse lxc-ubuntu
lxc-altlinux  lxc-debian   lxc-oracle   lxc-ubuntu-cloud
lxc-archlinux lxc-fedora   lxc-sshd
```

- The settings are stored in `/container/<name>`.

```
# lxc-create -n bb_cont -t busybox
# ls /container/bb_cont
config   rootfs
```

Template scripts define the settings for the different system resources that are assigned to a running container. Each template script defines different resources but examples include the container host name, network configuration, mount points, the root file system, number of available ttys, and other settings. The following is a list of the current template scripts:

```
# ls /usr/share/lxc/templates
lxc-alpine      lxc-busybox     lxc-opensuse    lxc-ubuntu
lxc-altlinux    lxc-debian      lxc-oracle      lxc-ubuntu-cloud
lxc-archlinux   lxc-fedora      lxc-sshd
```

Each template script is used to generate a specific type of Linux Container. The `lxc-altlinux` script is for generating an ALT Linux Container, the `lxc-busybox` script is for generating a BusyBox container, the `lxc-oracle` script is for generating an Oracle Linux Container, and so on.

The following example uses the `lxc-busybox` template to create a BusyBox container named `bb_cont`. The `-n` option provides the container name and the `-t` option specifies the template to use. Omit the `lxc-` portion of the template script:

```
# lxc-create -n bb_cont -t busybox
```

Each script creates a system object directory in `/container/` that has the same name as the `–n` argument passed to the `lxc-create` command. Each template script is different, but for the “BusyBox” example, the following files and directories are created:

```
# ls –l /container/bb_cont
-rw-r--r-- ... config
drw-r--r-- ... rootfs
```

The template scripts create the `config` file. The template scripts are not used after the container is created; they are used only during `lxc-create`.

Configuration settings from the template script are written to the `config` file, as in this example:

```
# cat /container/bb_cont/config
lxc.utsname = bb_cont
lxc.tty = 1
lxc.pts = 1
lxc.rootfs = /container/bb_cont/rootfs
lxc.mount.entry=/lib lib none ro,bind 0 0
lxc.mount.entry=/usr/lib usr/lib none ro,bind 0 0
lxc.mount.entry=/lib64 lib64 none ro,bind 0 0
lxc.mount.entry=/usr/lib64 usr/lib64 none ro,bind 0 0
```

These settings are taken directly from the `lxc-busybox` template script:

```
# grep lxc. /usr/share/lxc/templates/lxc-busybox
lxc.utsname = $name
lxc.tty = 1
lxc.pts = 1
lxc.rootfs = $rootfs
echo “lxc.mount.entry=/$dir $dir none ro,bind 0 0” >>
$path/config
```

These settings in this `config` file are defined as:

- `lxc.utsname` – Specifies the host name for the container
- `lxc.tty` – Specifies the number of ttys available to the container
- `lxc.pts` – Specifies the maximum number of pseudo ttys allowed for a `pts` instance
- `lxc.rootfs` – Specifies the `root` file system for the container. This can be an image file, a directory, or a block device. If it is not specified, the container shares its `root` file system with the host.
- `lxc.mount.entry` – Specifies a mount point corresponding to a line in the `fstab` file

The contents of `rootfs` in this “BusyBox” example are:

```
# ls /container/bb_cont/rootfs
bin  dev  etc  home  lib  lib64  mnt  proc  root  sbin  selinux
tmp  usr  var
```

`lxc-create` Utility

- Use the `lxc-create` command to create a container. The syntax is:

```
lxc-create -n name [-f config] [-t template] [-B
   <backing-store>] [-- template-options]
```

- The command creates an object directory that defines the resources that the container can use or can see:

```
/container/<name>/
```

- A configuration file is optional. The default isolation is:
  - Processes
  - SysV inter-process communication (IPC)
  - Mount points
- Template scripts also provide configuration information:

```
# lxc-create -n ol-test -t oracle
```

Use the `lxc-create` command to create a container. This command creates a system object directory in `/container`, which stores configuration and user information. The object provides a definition of the different resources that an application can use or can see. The syntax is:

```
lxc-create -n <container_name> [-f <config_file>] [-t
<template>] [-B <backing-store>] [-- template-options]
```

The options are described as follows:

- `-n <container_name>` – Specifies the name of the container. This name becomes the name of the system object directory, `/container/<container_name>`.
- `-f <config_file>` – Specifies the file used to configure the virtualization and isolation functionality of the container. If a configuration file is not specified, the container is created with the default isolation: processes, SysV IPC, and mount points.
- `-t <template>` – Specifies the short name of an existing template script. Template scripts are located in `/usr/share/lxc/templates`. Configuration information is provided in these template scripts.
- `-B <backing-store>` – Specifies either `none`, `btrfs`, or `lvm` as the backing-store
- `-- template-options` – Specifies arguments to pass to the template

The `-B` *`<backing-store>`* options are described further.

- `none`: This is the default. The container root file system is a directory under `/container/`*`container_name`*`/rootfs`.
- `btrfs`: This parameter need not be specified if the `/container` file system exists and `/container` is a Btrfs file system. If `/container` is a Btrfs file system, the `rootfs` directory is a subvolume when it is created by `lxc-create`.
- `lvm`: This specifies to use an LVM block device. Additional options are available when using the `lvm` parameter.
  - `--lvname lvname1`: Specifies creation of a logical volume named `lvname1` rather than the container name, which is the default logical volume
  - `--vgname vgname1`: Specifies creation of the logical volume in the volume group `vgname1` rather than `lxc`, which is the default volume group
  - `--fstype FSTYPE`: Specifies creation of an `FSTYPE` file system on the logical volume, rather than ext4, which is the default file system type
  - `--fssize SIZE`: Specifies creation of a logical volume and file system of `SIZE` rather than the default size of 1G

Template scripts are located in `/usr/share/lxc/templates`. Each script name begins with `lxc-`. To create a container named `ol-test` using the template script named `lxc-oracle`, use the short name (omit the `lxc-`) as follows:

```
# lxc-create –n ol-test –t oracle
```

Use the following command to display the list of options supported by the template:

```
lxc-create –t <template> -h
```

Use the short name when requesting template options. For example, to display options supported by the `lxc-oracle` template:

```
# lxc-create –t oracle –h
...
Template-specific options (TEMPLATE_OPTIONS):
-a|--arch=<arch>           architecture of the container
-R|--release=<release>     release to download for the container
-u|--url=<url>             replace yum repo url
-t|--templatefs=<path>     copy/clone rootfs at path
-h|--help
```

`lxc-oracle` Container Template

- This template simplifies the creation of Oracle Linux containers.
- The template has been accepted upstream for inclusion in the `lxc-0.9` RPM.
- This template creates the container by using packages from `public-yum.oracle.com` to create a new rootfs.
- To create an Oracle Linux 6.5 container with i386 packages:

```
# lxc-create -n ol65-32 -t oracle -- -a i386 -R 6.5
```

- To create the latest Oracle Linux 6 container with x86_64 packages:

```
# lxc-create -n ol6L-64 -t oracle -- -a x86_64 -R
   6.latest
```

The `lxc-oracle` template supports creating an Oracle Linux container using packages from `public-yum.oracle.com` to create a new rootfs. In addition, this template can create a container based on an existing rootfs using an Oracle VM template. The `lxc-oracle` template is accepted upstream for inclusion in `lxc-0.9`.

The following example uses the `lxc-oracle` template to create an Oracle Linux 6.5 container named `ol65-32` from `public-yum` with i386 packages:

```
    # lxc-create -n ol65-32 -t oracle -- -a i386 -R 6.5
```

To use the template and to create the latest Oracle Linux version 6 container from `public-yum` with x86_64 packages:

```
    # lxc-create -n ol6L-64 -t oracle -- -a x86_64 -R 6.latest
```

The following example creates an Oracle Linux container from an existing rootfs. The container rootfs is a Btrfs snapshot if `/path/to/rootfs` is a Btrfs subvolume on the same Btrfs file system.

```
    # lxc-create -n ol6L-64 -t oracle -- -t /path/to/rootfs
```

If you do not specify template options, the container defaults to the same architecture and Oracle Linux version as the host.

The `lxc-oracle` template uses the `virbr0` bridge set up by `libvirtd`. Before using this template, install `libvirt`, configure the service to start at boot time, and start the service as follows:

```
# yum install libvirt
# chkconfig libvirtd on
# service libvirtd start
```

Create a container named `ol-test` using the `lxc-oracle` template. If you do not specify template options, the container defaults to the same architecture and Oracle Linux version as the host.

```
# lxc-create -n ol-test -t oracle
...
Host is OracleServer 6.5
Create configuration file /container/ol-test/config
Downloading release 6.5 for x86_64
...
Transaction Summary
==============================================================
...
Complete!
Rebuilding rpm database
Configuring container for Oracle Linux 6.5
Added container user:oracle password:oracle
Added container user:root password:root
Container : /container/ol-test/rootfs
Config    : /container/ol-test/config
Network   : eth0 () on virbr0
'oracle' template installed
'ol-test'created
```

As you can see from the output, the `lxc-oracle` template creates an `oracle` user. It sets the initial `root` and `oracle` user passwords for the container and prints them out so that you can log in.

The `lxc-oracle` template also creates the network configuration as follows:

```
# grep network /container/ol-test/config
lxc.network.type = veth
lxc.network.link = virbr0
lxc.network.name = eth0
...
```

The `veth` type of network virtualization is a peer network device with one side assigned to the container and the other side attached to a bridge specified by `lxc.network.link`.

Starting and Stopping a Container

- There are two ways to run an application or a system in a container:

```
lxc-start -n name -f configfile cmd
lxc-execute -n name -f configfile -- cmd
```

- The `lxc-execute` command runs the specified command using an intermediate process, `lxc-init`.
  - `lxc-init` has a PID of `1`. The first process of the application has a PID of `2`.
- The `lxc-start` command runs the specified command directly in the container.
  - The PID of the first process is `1`. If no command is specified, `lxc-start` runs `/sbin/init`.
- Use `lxc-stop` to kill all the processes in the container.

Use the `lxc-start` command or the `lxc-execute` command to run an application or a system inside a container. If you did not create the container before starting the application, the container uses the configuration file that is passed as an argument to the command. If there is no configuration file, the container uses the default isolation. The syntax for both `lxc-start` and `lxc-execute` is similar; they both accept a command to run within the container as an argument:

```
lxc-start -n <container_name> ... [command]
lxc-execute -n <container_name> ... [-- command]
```

The `lxc-execute` command runs the specified command using an intermediate process, `lxc-init`. In the container, `lxc-init` has a PID of `1` and the first process of the application has a PID of `2`.

The `lxc-start` command runs the specified command directly in the container. The PID of the first process is `1`. If no command is specified, `lxc-start` runs `/sbin/init`.

Therefore, `lxc-execute` is best suited for running an application within a container, whereas `lxc-start` is best suited for running a system in the container.

When the application stops, the container is also stopped. You can also use the `lxc-stop` command to kill all the processes in the container.

`lxc-start` Utility

- Is mainly used to run a system container. The syntax is:

```
lxc-start -n <name> [-f <config_file>] [-c
    <console_file>] [-d] [-s <KEY=VAL>] [<command>]
```

- Runs the specified `[<command>]` in the container
- If no command is specified, uses the default `/sbin/init` to run a system container
  - `-d`: Runs the container as a daemon
  - `-f`: Specifies a configuration file that overrides a container created using `lxc-create`
  - `-c`: Specifies a file to output the container console
  - `-s <KEY=VAL>`: Overrides configuration settings

System containers emulate an entire Linux OS booting from scratch. They provide their own `init` program, so `lxc-init` is not needed. Use `lxc-start` for system containers and use `lxc-execute` for running a single program, which generally shares the rootfs with the host system. The complete syntax of `lxc-start` is as follows:

```
lxc-start -n <name> [-f <config_file>] [-c <console_file>]
          [-d] [-s <KEY=VAL>] [<command>]
```

This command runs the specified `[<command>]` inside the container, which is identified by the `-n <name>` argument. If no `[<command>]` is specified, `lxc-start` uses the default `/sbin/init` to run a system container.

If the container does not exist, that is, it was not already created by the `lxc-create` command, the container is set up as defined in the `<config_file>`. If no configuration is defined, the default isolation (processes, SysV IPC, and mount points) is used.

The remaining options are described as follows:

- `-d` – Specifies to run the container as a daemon. If an error occurs, nothing is displayed.
- `-c <console_file>` – Specifies a file to output the container console. By default, output is written to the terminal if the `-d` option is not specified.
- `-s <KEY=VAL>` – Assigns a value (VAL) to a configuration key (KEY). This overrides assignments in the configuration file.

`lxc-execute` Utility

- Is mainly used to run an application in an isolated environment. The syntax is:

```
lxc-execute -n <name> [-f <config_file>] [-s <KEY=VAL>]
   [-- <command>]
```

- Runs the specified `[<command>]` inside the container via an intermediate process, `lxc-init`
  - The `lxc-init` process had a PID of 1.
  - The `lxc-init` process waits for the specified command and all child processes to end before `lxc-init` ends.
- Example:

```
# lxc-execute -n bbtest -s lxc.cgroup.cpuset.cpus=0
   /sbin/busybox httpd
```

The complete syntax of `lxc-execute` is as follows:

```
    lxc-execute -n <name> [-f <config_file>] [-s <KEY=VAL>]
    [-- <command>]
```

This command runs the specified `[<command>]` inside the container via an intermediate process, `lxc-init`. The `lxc-init` process has a PID of `1`, is executed when `lxc-execute` is run, and invokes the command that you pass to `lxc-execute`. After launching the specified command, `lxc-init` waits for the command and all the child processes to end before it ends. This allows it to support daemons inside the container. The options are similar to `lxc-start`, except that there is no `–c console_file` option and no option to run the specified command as a daemon using `–d`.

The following example starts BusyBox's simple `httpd` server in a container and lets it run only on CPU `0`. You do not need to pre-create the `bbtest` container with `lxc-create`, you can just use `lxc-execute` to run it.

```
    # lxc-execute -n bbtest -s lxc.cgroup.cpuset.cpus=0
    /sbin/busybox httpd
```

The following example runs a backup job, `/root/bkup-system.sh`, in a container:

```
# lxc-execute -n bkup \
  -s lxc.cgroup.cpuset.cpus=0 \
  -s lxc.cgroup.cpu.shares=256 \
  -s lxc.cgroup.memory.soft_limit_in_bytes=268435456 \
  -s lxc.cgroup.memory.limit_in_bytes=536870912 \
  /root/bkup-system.sh
```

The following defines the cgroup criteria:

- Limit the job to run on only CPU 0.
- Limit the job to only use 25% of the CPU (100% shares is 1024).
- Limit the container to use only 256 MB when there is memory pressure.
- Otherwise limit it to 512 MB regardless of memory pressure.

`lxc-ls` and `lxc-info` Utilities

- Use the `lxc-ls` command to display the containers that the host system is running.

```
# lxc-ls
ol6ctrl
```

- Use the `lxc-info` command to display the state of a container.

```
# lxc-info –n ol6ctrl
state:    STOPPED
pid:           -1
```

- A container can be in one of the following states: `ABORTING`, `RUNNING`, `STARTING`, `STOPPED`, `STOPPING`, or `FROZEN`.

Use the `lxc-ls` command to display the containers that the host system is running.

```
# lxc-ls
ol6ctrl
```

The command accepts the options of the `ls` command.

Use the `lxc-info` command to display the state of a container.

```
# lxc-info –n ol6ctrl
state:    STOPPED
pid:           -1
```

A container can be in one of the following states: `ABORTING`, `RUNNING`, `STARTING`, `STOPPED`, `STOPPING`, or `FROZEN`. Start the container, and then display the state, which shows that it is `RUNNING`.

```
# lxc-start –n ol6ctrl -d
# lxc-info –n ol6ctrl
state:    RUNNING
pid:        2677
```

`lxc-console` Utility

- Container must be in `RUNNING` state and must be configured with ttys:
  - `/sbin/mingetty` processes are running in the container.
- Use the `lxc-console` command to access the container through the ttys. The syntax is:

```
lxc-console -n <name> [-t <ttynum>]
```

- Specify the number of ttys available to the container in a configuration file:

```
lxc.tty = #
```

- To exit a `lxc-console` session, press Ctrl + A and then Q.
- You can also use `ssh` to log in if the container is running `/sbin/sshd` and has an IP address.

If the container is in `RUNNING` state and is configured with ttys (that is, `/sbin/mingetty` processes have started running in the container), you can use the `lxc-console` command to access the container through the ttys. The syntax of `lxc-console` is as follows:

```
lxc-console -n <name> [-t <ttynum>]
```

You can optionally specify the tty number to connect. If it is not specified, a tty number is automatically selected by the container. You are prompted for a login and a password:

```
# lxc-console -n ol6ctrl
ol6ctrl login: root
password:
```

Use the following configuration entry to define the number of ttys available:

```
lxc.tty = #
```

To exit the `lxc-console` session, type Ctrl + A followed by Q.

You can also log in using the `ssh` command if the container has an IP address assigned to the virtual network interface and the container has the `/usr/sbin/sshd` process running.

`lxc-freeze` and `lxc-unfreeze` Utilities

- Use the `lxc-freeze` command to stop all processes that belong to a container.

```
# lxc-freeze –n ol6ctrl
# lxc-info –n ol6ctrl
state:    FROZEN
pid:       2677
```

- Use the `lxc-unfreeze` command to resume all processes:

```
# lxc-unfreeze –n ol6ctrl
```

- This feature requires the cgroup `freezer` subsystem to be enabled in the kernel.

Use the `lxc-freeze` command to stop all processes that belong to a container (for example, to accommodate job scheduling). This command puts all the processes in an uninterruptible state. The state changes to `FROZEN`.

```
# lxc-freeze –n ol6ctrl
# lxc-info –n ol6ctrl
state:    FROZEN
pid:       2677
```

Use the `lxc-unfreeze` command to resume all processes.

```
# lxc-unfreeze –n ol6ctrl
# lxc-info –n ol6ctrl
state:    RUNNING
pid:       2677
```

This feature is enabled only if the cgroup `freezer` subsystem is enabled in the kernel.

`lxc-cgroup` Utility

- When a container is started, a control group (cgroup) is created and associated with the container.
- Use the `lxc-cgroup` command to view and modify the cgroup subsystem parameters set in the container.
- To display a cgroup subsystem parameter:

```
# lxc-cgroup -n ol6ctrl cpu.shares
```

- To set the value of a cgroup subsystem parameter:

```
# lxc-cgroup -n ol6ctrl cpu.shares 500
```

- To set a persistent value, add the entry to the container's configuration file (or template script before `lxc-create`):

```
lxc.cgroup.cpu.shares = 500
```

When a container is started, a control group (cgroup) is created and associated with the container. Use the `lxc-cgroup` command to view and modify the cgroup subsystem parameters set in the container. To display the current value of a cgroup subsystem parameter, provide the parameter as an argument.

```
# lxc-cgroup -n ol6ctrl cpu.shares
1024
```

To set the value of a cgroup subsystem parameter, pass the parameter and the value as arguments.

```
# lxc-cgroup -n ol6ctrl cpu.shares 500
# lxc-cgroup -n ol6ctrl cpu.shares
500
```

To make the cgroup subsystem values persistent, add the settings to the container's configuration file or add the settings to the template script before creating the container.

```
lxc.cgroup.cpu.shares = 500
lxc.cgroup.devices.allow = c 1:3 rwm
```

Additional Linux Container Utilities

- `lxc-destroy` – Destroys a container object
- `lxc-ps` – Lists the running processes in a container
- `lxc-netstat` – Displays network information for a container
- `lxc-monitor` – Monitors the state of a container
- `lxc-wait` – Waits for a container to change to a specified state

Available LXC utilities:

- `lxc-create` – Creates a persistent container object
- `lxc-destroy` – Destroys a container object
- `lxc-execute` – Runs an application inside a container
- `lxc-start` – Runs a system inside a container
- `lxc-stop` – Kills all the processes inside a container
- `lxc-console` – Logs in to a container
- `lxc-freeze` – Stops all the processes belonging to a container
- `lxc-unfreeze` – Resumes all the processes belonging to a container
- `lxc-ls` – Lists the containers belonging to a host system
- `lxc-ps` – Lists the processes running in a specific container
- `lxc-info` – Displays the state of a container
- `lxc-netstat` – Displays network information for a container
- `lxc-monitor` – Monitors the state of a container
- `lxc-wait` – Waits for a container to change to a specified state
- `lxc-cgroup` – Sets or displays the value of a cgroup subsystem parameter

Using Oracle VM Template as a Base Environment

1. Download Oracle VM template (ZIP archive) from http://edelivery.oracle.com/linux.
2. Unzip the Oracle Linux ZIP archive.
3. Extract the `.ova` file (using the `tar` command).
4. Rename the `System.img` file to `System.img.gz`.
5. Uncompress the `System.img.gz` file.
6. Use the `kpartx -a` command to add device mappings for the detected partitions on the `System.img` file.
7. Mount the `root` partition on `/mnt`.
8. Create a Btrfs file system (`/container`) and subvolume (`/container/ol#-template`).
9. Copy `/mnt` to `/container/ol#-template`.

Perform the following steps to use an Oracle VM template as a base environment for the `lxc-oracle` script:

1. Download an Oracle VM template from http://edelivery.oracle.com/linux into a temporary directory. This temporary directory needs to be large enough to accommodate a 12 GB `System.img` file.
2. The downloaded file name begins with a `V` and has a `.zip` extension. Use the `unzip` utility to unzip this file. The example Oracle VM template file name is `V42906-01.zip`.

```
# unzip V42906-01.zip
Archive: V42906-01.zip
  inflating: OVM_OL6U5_x86_64_PVM.ova
```

3. Extract the ZIP file. In this example, this results in the `OVM_OL6U5_x86_64_PVM.ova` file. Use the `tar` utility to untar the `.ova` file:

```
# tar xvf OVM_OL6U5_x86_64_PVM.ova
OVM_OL6U5_x86_64_PVM.ovf
OVM_OL6U5_x86_64_PVM.mf
System.img
```

4. Use the `mv` command to rename `System.img` to `System.img.gz`:

   ```
   # mv System.img System.img.gz
   ```

5. Use the `gunzip -d` command to uncompress the `System.img.gz` file:

   ```
   # gunzip -d System.img.gz
   ```

6. Use the `kpartx -l` command to list the partitions found on the `System.img` file:

   ```
   # kpartx -l System.img
   loop0p1 : 0 1028096 /dev/loop0 2048
   loop0p2 : 0 19941376 /dev/loop0 210944
   loop0p3 : 0 4194304 /dev/loop0 20971520
   ```

   The largest partition, `loop0p2` in this example, corresponds to the `root` partition. Use the `kpartx -a` command to add device mappings for the detected partitions on the `System.img` file:

   ```
   # kpartx -a System.img
   ```

   The new device mappings appear in `/dev/mapper`:

   ```
   # ls /dev/mapper
   control  loop0p1  loop0p2  loop0p3
   ```

7. Use the `mount` command to mount the `root` partition, `/dev/mapper/loop0p2`, on `/mnt`.

   ```
   # mount /dev/mapper/loop0p2 /mnt
   ```

8. Use the `mkfs.btrfs` command to create a Btrfs file system on a suitable device. This example uses `/dev/xvdb`:

   ```
   # mkfs.btrfs –L container /dev/xvdb
   ```

   Use the `btrfs sub create` command to create a subvolume named `ol6-template` (if you downloaded the Oracle Linux 6 OVM template) on the Btrfs file system:

   ```
   # btrfs sub create /container/ol6-template
   ```

9. Copy the contents of `/mnt` to this Btrfs subvolume, `/container/ol6-template`. There are several different methods to copy the files; the following command is only one example:

   ```
   # find /mnt –mount –depth | cpio –pdv /container/ol6-template
   ```

The `/container/ol6-template` Btrfs subvolume can now be used as a template with the `lxc-oracle` script because it contains the `root` file system for Oracle Linux 6:

```
# ls /container/ol6-template
bin  dev home lib64      media opt  root selinux sys u01 var
boot etc lib  lost+found mnt   proc sbin srv     tmp usr
```

When you use the `lxc-create` command with the `lxc-oracle` template and the `/path/to/rootfs` template option:

```
# lxc-create –n <name> -t oracle -- -t /container/ol6-template
```

The rootfs of the new container is a Btrfs snapshot if `/path/to/root` is a subvolume on the same Btrfs file system.

Quiz

Which of the following statements are true?

a. Containers provide application and operating system isolation.

b. Each container has its own copy of the Linux kernel.

c. Containers rely on the cgroups functionality but also rely on namespace isolation (similar to chroot).

d. By default, the process IDs, SysV IPCs, and mount points are isolated.

Quiz

Which of the following statements are true about LXC template scripts?

a. Template scripts define the settings for the different system resources that are assigned to a running container.
b. Template scripts are located in the `/usr/share/lxc/templates` directory.
c. Each template script is used to generate a specific type of Linux Container.
d. Template scripts are arguments to the `lxc-start` and `lxc-execute` commands.

Summary

In this lesson, you should have learned how to:

- Install the required Linux Container software packages
- Use Linux Container template scripts
- Create a Linux Container by using the Oracle template script
- Use Linux Container utilities to start and stop a container
- Use additional Linux Container utilities
- Create a base environment from an Oracle VM template
- Create a Linux Container from a base environment (an existing rootfs)

Practice 15: Overview

This practice covers the following topics:

- Preparing your system to create a Linux Container
- Creating a Linux Container from a template
- Exploring container configuration
- Using `lxc` commands
- Using an Oracle VM template as a base environment
- Creating a container from an existing rootfs

16

Security Enhanced Linux (SELinux)

Objectives

After completing this lesson, you should be able to:

- Describe SELinux
- Describe SELinux packages
- Use the SELinux Administration GUI
- Describe SELinux modes
- Describe SELinux policies
- Describe SELinux Booleans
- Describe SELinux file labeling, context, and users
- Use SELinux utilities

Introduction to SELinux

- SELinux:
  - Was created by the National Security Agency
  - Is an enhancement to the Linux kernel
  - Implements a type of security called MAC
- Standard Linux security is based on DAC:
  - DAC provides minimal protection.
  - Access to objects is based on user identity and ownership.
- MAC can define security policy:
  - It provides granular permissions for all users, programs, processes, files, and devices.
  - SELinux policy rule determines whether access is allowed.
- The Linux kernel checks and enforces MAC rules after it checks DAC rules.

Standard Linux security is based on Discretionary Access Control (DAC). DAC provides minimal protection from broken software or malware running as a normal user or `root`. Access to files and devices is based solely on user identity and ownership. Malicious or flawed software can do anything with the files and resources it controls through the user that started the process. If the user is `root` or the application is `setuid` or `setgid` to `root`, the process can have `root`-level control over the entire file system.

SELinux (Security Enhanced Linux) was created by the United States National Security Agency to provide a finer-grained level of control over files, processes, users, and applications in the system. It is an enhancement to the Linux kernel, and it implements a different type of security called Mandatory Access Control (MAC). MAC policy is centrally managed rather than being managed by the user.

MAC under SELinux allows you to define a security policy that provides granular permissions for all users, programs, processes, files, and devices. Access control decisions are based on all the security-relevant information available, and not just authenticated user identity. When security-relevant access takes place, such as when a process attempts to open a file, the operation is intercepted in the kernel by SELinux. If an SELinux policy rule allows the operation, it continues; otherwise, the operation is blocked and the process receives an error. The kernel checks and enforces MAC rules after it checks DAC rules. SELinux policy rules are not used if DAC rules deny access first.

SELinux Packages

- `policycoreutils`: Provides the policy core utilities that are required for basic operation of SELinux
- `libselinux-utils`: Provides additional SELinux tools
- `libselinux`: Provides an API for SELinux tools
- `libselinux-python`: Contains the Python bindings for developing SELinux applications
- `selinux-policy`: Provides the SELinux Reference Policy
- `selinux-policy-targeted`: Provides targeted policies. For MLS, install `selinux-policy-mls`.
- Other packages are available but not installed by default.

There are many different SELinux software packages, some of these are installed by default, and some are not. The following is a list of the SELinux packages that are installed by default:

- **`policycoreutils`:** Provides utilities such as `sestatus`, `restorecon`, `secon`, `setfiles`, `semodule`, `load_policy`, and `setsebool` for operating and managing SELinux
- **`libselinux-utils`:** Provides the `avcstat`, `getenforce`, `getsebool`, `matchpathcon`, `selinuxconlist`, `selinuxdefcon`, `selinuxenabled`, `setenforce`, and `togglesebool` tools
- **`libselinux`:** Provides an API for SELinux applications to get and set process and file security contexts and to obtain security policy decisions
- **`libselinux-python`:** Contains the Python bindings for developing SELinux applications
- **`selinux-policy`:** Provides the SELinux Reference Policy. The SELinux Reference Policy is a complete SELinux policy and is used as a basis for other policies, such as the SELinux targeted policy.
- **`selinux-policy-targeted`:** Provides the targeted policy. For MLS policy, install `selinux-policy-mls`.

The following is a partial list of SELinux packages that are not installed by default:

- **`setroubleshoot-server`**: Translates denial messages, produced when access is denied by SELinux, into detailed descriptions that are viewed with the `sealert` command
- **`policycoreutils-python`**: Provides additional utilities such as `semanage`, `audit2allow`, `audit2why`, and `chcat` for operating and managing SELinux
- **`policycoreutils-gui`**: Provides `system-config-selinux`, which is a graphical tool for managing SELinux
- **`setools-console`**: Provides the Tresys Technology SETools distribution, several tools and libraries for analyzing and querying policy, audit log monitoring and reporting, and file context management
- **`mcstrans`**: Translates levels, such as s0-s0:c0.c1023, to an easier-to-read form, such as `SystemLow-SystemHigh`

The `semanage` command is described in this lesson. Install the `policycoreutils-python` package as follows to run the `semanage` command:

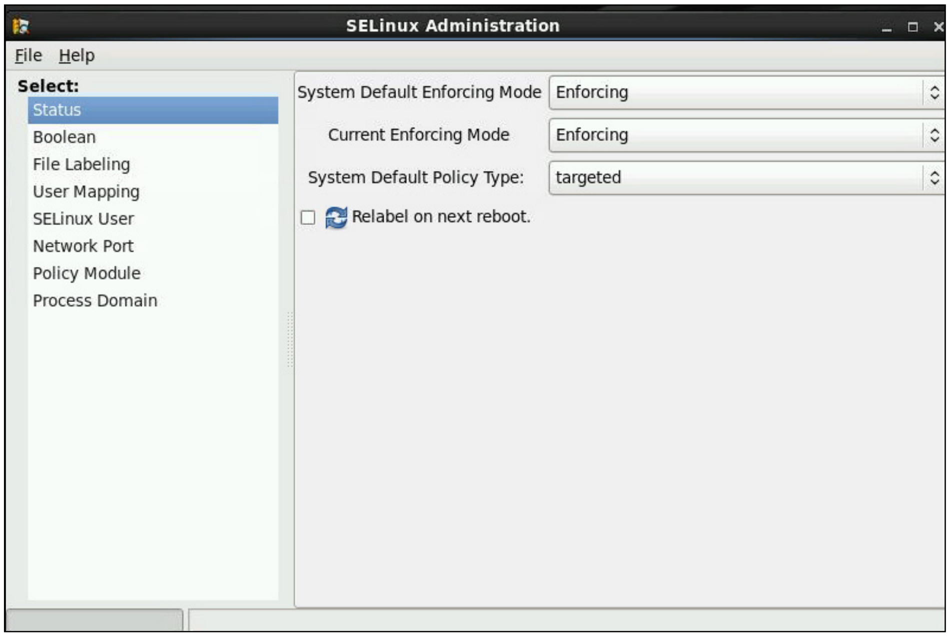
```
# yum install policycoreutils-python
```

This installs the following dependencies:

- `audit-libs-python`
- `libcgroup`
- `libsemanage-python`
- `setools-libs`
- `setools-libs-python`

SELinux Administration GUI

```
system-config-selinux
```



System Default Enforcing Mode: Enforcing
Current Enforcing Mode: Enforcing
System Default Policy Type: targeted
☐ Relabel on next reboot.

The SELinux Administration graphical user interface (GUI) is a tool used to manage SELinux. To run this GUI, install the `policycoreutils-gui` package:

```
# yum install policycoreutils-gui
```

This installs the following dependencies:

- `gnome-python2-gtkhtml2`
- `gtkhtml2`
- `setools-console`

Enter the following command to display the GUI:

```
# system-config-selinux
```

The GUI displays a list of options on the left side. The right side of the GUI changes as different options are selected from the left.

The example shown in the slide is the Status view. From this view, you can change:

- **Enforcing Mode:** Enforcing, Permissive, Disabled
- **Policy Type:** Targeted, MLS (Multi-Level Security)
- **Relabel on next reboot:** Labels all files at boot time with an SELinux context

The `sestatus` command displays the SELinux mode and the SELinux policy being used.

SELinux Modes

- Enforcing:
  - This is the default state. Access is denied to users and programs unless permitted by SELinux security policy rules.
- Permissive:
  - Security policy rules are not enforced but denial messages are logged.
- Disabled:
  - No security policy is loaded. Only DAC rules are used for access control.

SELinux runs in one of three modes (or states).

Enforcing

This is the default state that enforces SELinux security policy. Access is denied to users and programs unless permitted by SELinux security policy rules. All denial messages are logged as AVC (Access Vector Cache) Denials.

Permissive

This is a diagnostic state. The security policy rules are not enforced, but SELinux sends denial messages to a log file. This allows you to see what would have been denied if SELinux were running in enforcing mode.

Disabled

SELinux does not enforce a security policy because no policy is loaded in the kernel. Only DAC rules are used for access control.

Setting a Mode

- Set the mode from the SELinux GUI.
- Define the `SELINUX` directive in `/etc/selinux/config`:
  - `SELINUX=enforcing`
  - `SELINUX=permissive`
  - `SELINUX=disabled`
- Use the `setenforce` command to set either enforcing (`1`) or permissive (`0`) mode. Mode does not persist after a reboot:

```
# setenforce 1
# setenforce 0
```

- To view the current SELinux mode:

```
# getenforce
Enforcing
```

There are multiple ways of setting the SELinux mode. One way is to select the mode from the Status view in the SELinux GUI.

You can also edit the main configuration file for SELinux, `/etc/selinux/config`. Set the mode by changing the `SELINUX` directive in this file. For example, to set the mode to enforcing:

```
SELINUX=enforcing
```

The `setenforce` command is used to change between enforcing and permissive modes. Changes made with this command do not persist across reboots.

To change to enforcing mode:

```
# setenforce 1
```

To change to permissive mode:

```
# setenforce 0
```

Display the Current Mode

Use the `getenforce` command to view the current SELinux mode:

```
# getenforce
Enforcing
```

SELinux Policies

- SELinux implements one of two different policies:
  - Targeted: Applies access controls to targeted processes
  - MLS: Multi-Level Security
- Select the policy type from the SELinux GUI or define the `SELINUXTYPE` directive in `/etc/selinux/config`:
  - `SELINUXTYPE=targeted`
  - `SELINUXTYPE=mls`
- Targeted policy:
  - Targeted processes run in a confined domain.
  - All network services are targeted processes.
  - Access to files is limited.
- Non-targeted processes run in an unconfined domain.
  - Access to files is not limited.

The SELinux policy describes the access permissions for all users, programs, processes, files, and devices they act upon. SELinux implements one of two different policies:

- **Targeted:** This default policy applies access controls to certain (targeted) processes.
- **MLS:** Multi-Level Security

Select the policy type from the SELinux GUI, or set the `SELINUXTYPE` directive in the `/etc/selinux/config` file. Example:

```
SELINUXTYPE=targeted
```

With the targeted policy, targeted processes run in their own domain, called a confined domain. In a confined domain, the files that a targeted process has access to are limited. If a confined process is compromised by an attacker, the attacker's access to resources and the possible damage they can do is also limited. SELinux denies access to these resources and logs the denial.

Only specific services are placed into these distinct security domains that are confined by the policy. For example, a user runs in a completely unconfined domain while services that listen on a network for client requests, such as `named`, `httpd`, and `sshd`, run in a specific, confined domain tailored to its operation. Processes that run as the Linux `root` user and perform tasks for users, such as the `passwd` application, are also confined.

Processes that are not targeted run in an unconfined domain. SELinux policy rules allow processes running in unconfined domains almost all access. If an unconfined process is compromised, SELinux does not prevent an attacker from gaining access to system resources and data. DAC rules still apply in an unconfined domain.

The following are examples of unconfined domains:

- `initrc_t` domain: `init` programs run in this unconfined domain.
- `kernel_t` domain: Unconfined kernel processes run in this domain.
- `unconfined_t` domain: Linux users logged in to the system run in this domain.

Many domains that are protected by SELinux have `man` pages describing how to customize their policies.

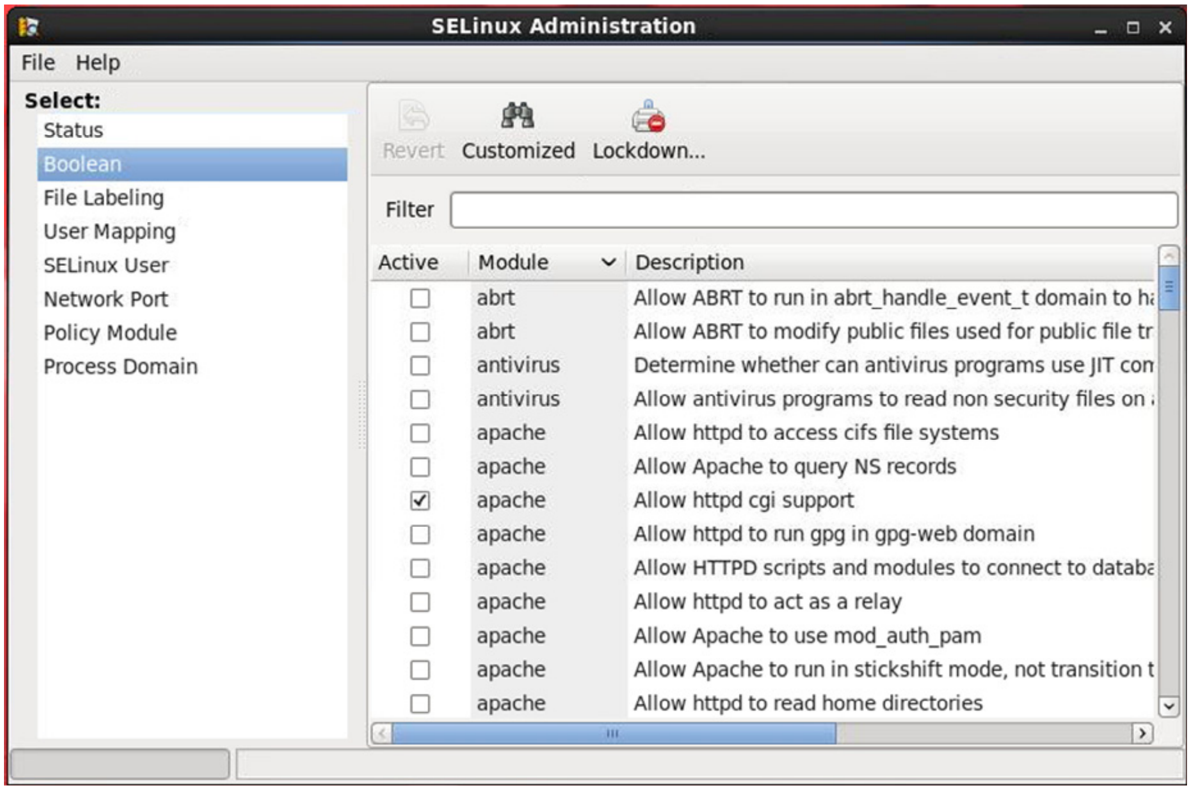
The configuration for each policy is installed in the `/etc/selinux/<`*`SELINUXTYPE`*`>` directories. The following example shows a partial listing of the `/etc/selinux` directory with both targeted and MLS policies installed:

```
# ls –l /etc/selinux
-rw-r--r--. root  root  config
drwxr-xr-x. root  root  mls
drwxr-xr-x. root  root  targeted
```

The targeted policy is installed by default, but the MLS policy is not. To use the MLS policy, install the `selinux-policy-mls` package:

```
# yum install selinux-policy-mls
```

SELinux Booleans



Active	Module	Description
☐	abrt	Allow ABRT to run in abrt_handle_event_t domain to ha
☐	abrt	Allow ABRT to modify public files used for public file tr
☐	antivirus	Determine whether can antivirus programs use JIT com
☐	antivirus	Allow antivirus programs to read non security files on a
☐	apache	Allow httpd to access cifs file systems
☐	apache	Allow Apache to query NS records
☑	apache	Allow httpd cgi support
☐	apache	Allow httpd to run gpg in gpg-web domain
☐	apache	Allow HTTPD scripts and modules to connect to databa
☐	apache	Allow httpd to act as a relay
☐	apache	Allow Apache to use mod_auth_pam
☐	apache	Allow Apache to run in stickshift mode, not transition t
☐	apache	Allow httpd to read home directories

A given SELinux policy can be customized by enabling or disabling a set of policy Booleans. Booleans allow parts of SELinux policy to be changed at run time, without any knowledge of SELinux policy writing. This allows changes without reloading or recompiling SELinux policy.

The `system-config-selinux` command displays the SELinux GUI, which, via the Boolean view (shown in the slide), allows customization of these Booleans. In the Boolean view, the Active check box indicates whether a Boolean is on or off. You can sort by any of the three columns (Active, Module, or Description).

You can also display this list from the command line by using the following command:

```
# semanage boolean -l
SELinux boolean  State  Default Description
ftp_home_dir      (off   ,  off) Allow ftp to read and write ...
smartmon_3ware    (off   ,  off) Enable additional permissions...
xdm_sysadm_login (off   ,  off) Allow xdm logins as sysadm
...
```

In the sample listing, the `ftp_home_dir` Boolean is off, which prevents the FTP daemon, `vsftpd`, from reading and writing to files in user home directories.

Displaying Booleans

You can also use the `getsebool` command to list Booleans. This command displays statuses but no descriptions.

To display all Booleans and their statuses:

```
# getsebool –a
abrt_anon_write --> off
abrt_handle_event --> off
allow_console_login --> on
...
```

Include the Boolean name as an argument to display the status of a specific Boolean. Multiple Boolean arguments are also allowed:

```
# getsebool allow_ftpd_use_nfs mozilla_read_content
allow_ftpd_use_nfs --> off
mozilla_read_content --> off
```

The `seinfo` command also displays the list of Booleans:

```
# seinfo –b
Conditional Booleans: 217
  allow_domain_fd_use
  allow_ftpd_full_access
  ...
```

Setting Booleans

Use the `setsebool` command to configure Booleans from the command line. The syntax is:

```
setsebool <Boolean> on|off
```

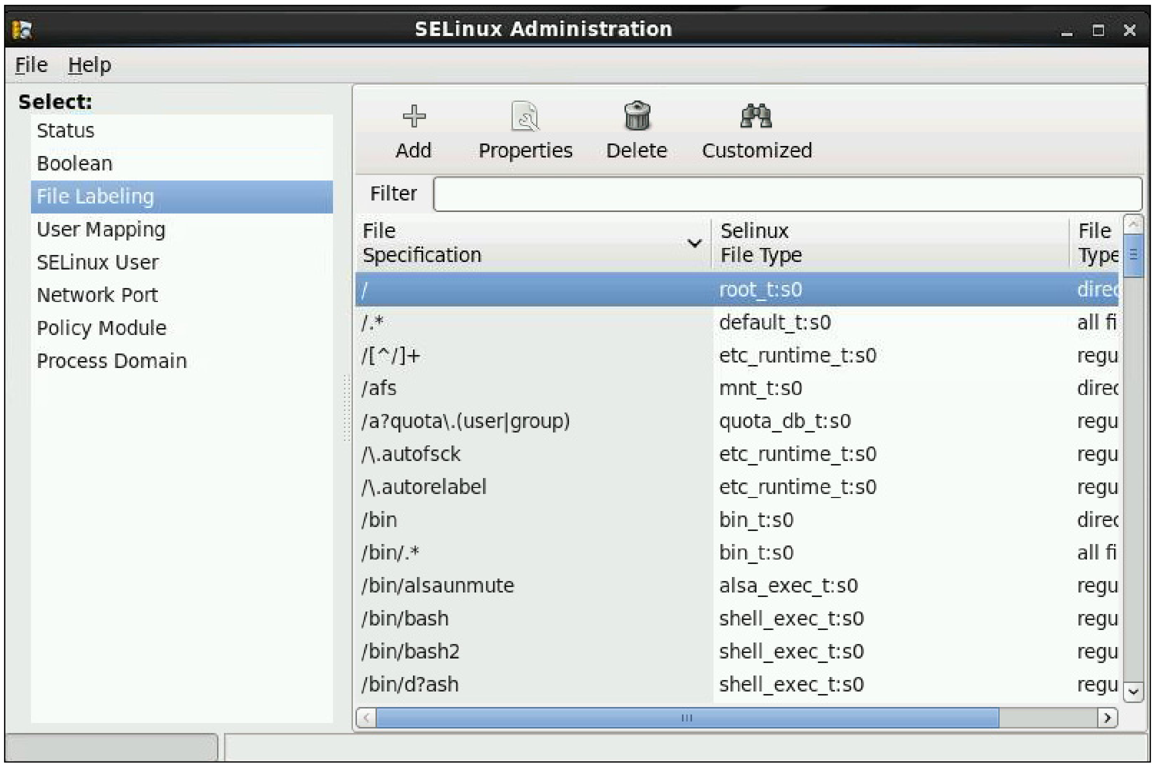
For example, the following sequence of commands displays the current status of a Boolean, then enables it to allow the `syslogd` daemon to send mail, and then displays the status again:

```
# getsebool logging_syslogd_can_sendmail
logging_syslogd_can_sendmail --> off
# setsebool logging_syslogd_can_sendmail on
logging_syslogd_can_sendmail --> on
```

To make the change persistent across reboots, use the `–P` option:

```
# setsebool -P logging_syslogd_can_sendmail on
```

SELinux File Labeling



File Specification	Selinux File Type	File Type
/	root_t:s0	direc
/.*	default_t:s0	all fi
/[^/]+	etc_runtime_t:s0	regu
/afs	mnt_t:s0	direc
/a?quota\.(user|group)	quota_db_t:s0	regu
/\.autofsck	etc_runtime_t:s0	regu
/\.autorelabel	etc_runtime_t:s0	regu
/bin	bin_t:s0	direc
/bin/.*	bin_t:s0	all fi
/bin/alsaunmute	alsa_exec_t:s0	regu
/bin/bash	shell_exec_t:s0	regu
/bin/bash2	shell_exec_t:s0	regu
/bin/d?ash	shell_exec_t:s0	regu

The SELinux GUI's File Labeling view is shown in the slide. All files, directories, devices, and processes have a security context (or label) associated with them. For files, this context is stored in the extended attributes of the file system. Problems with SELinux often arise from the file system being mislabeled. If you see an error message containing `file_t`, that is usually a good indicator that you have a problem with file system labeling.

There are several ways to relabel the file system:

- Create the `/.autorelabel` file and reboot.
- The Status view in the SELinux GUI provides an option to relabel on next reboot.
- Three command-line utilities, `restorecon`, `setfiles`, and `fixfiles`, relabel files.

To view the file system context information from the command line, use the `ls -Z` command:

```
# ls -Z
-rw-------. root root system_u:object_r:admin_home_t:s0 anaco...
-rw-r--r--. root root system_u:object_r:admin_home_t:s0 insta...
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 k...
```

This information is also stored in the `/etc/selinux/<`*`SELINUXTYPE`*`>/contexts/files` directory.

SELinux Context

- Context is additional information about users, files, and processes.
- Access control is based on this information:
  - SELinux user: Linux users are mapped to SELinux users.
  - Role: An attribute of RBAC that acts as an intermediary between domains and SELinux users
  - Type: An attribute of TE that defines a domain for processes
  - Level: Optional information; an attribute of MLS and MCS
- Use the `-Z` option to display context information:

```
# ls -Z
# ps -Z
# id -Z
```

- The format is `user:role:type:level`.

The SELinux context contains additional information such as SELinux user, role, type, and level. Access control decisions on processes, Linux users, and files are based on this context information.

To view the SELinux context information about files, use the `ls -Z` command:

```
# ls -Z
-rw-------. root root system_u:object_r:admin_home_t:s0 anaco...
```

To view the SELinux context information about processes, use the `ps -Z` command:

```
# ps -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 16445 su
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 16531 bash
```

To view the SELinux context associated with your Linux user, use the `id -Z` command:

```
# id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

SELinux context is displayed by using the following syntax:

```
user:role:type:level
```

SELinux User

There are SELinux users in addition to the regular Linux users. SELinux users are part of an SELinux policy. The policy is authorized for a specific set of roles and for a specific MLS (Multi-Level Security) range. Each Linux user is mapped to an SELinux user as part of the policy. This allows Linux users to inherit the restrictions and security rules and mechanisms placed on SELinux users.

To define what roles and levels they can enter, the mapped SELinux user identity is used in the SELinux context for processes in that session. Use the SELinux Administration GUI to display user mapping. You can also view a list of mappings between SELinux and Linux user accounts from the command line:

```
# semanage login -l
Login Name    SELinux User      MLS/MCS Range
_default_     unconfined_u      s0-s0:c0.c1023
root          unconfined_u      s0-s0:c0.c1023
system_u      system_u          s0-s0:c0.c1023
```

Linux users are mapped to the SELinux `_default_` login by default, which is mapped to the SELinux `unconfined_u` user. The last column, MLS/MCS Range, is the level used by MLS and MCS (Multi-Category Security).

Role

Role is an attribute of the Role-Based Access Control (RBAC) security model. The role serves as an intermediary between domains and SELinux users. SELinux users are authorized for roles, roles are authorized for domains, and processes run in their own separate domains. The roles determine which domains you can enter, and ultimately, which files you can access.

Type

Type is an attribute of Type Enforcement (TE). The type defines a type for files, and defines a domain for processes. Processes are separated from each other by running in their own domains. This separation prevents processes from accessing files used by other processes, as well as preventing processes from accessing other processes. SELinux policy rules define how types can access each other, whether it is a domain accessing a type, or a domain accessing another domain.

Level

Level is an attribute of MLS and MCS. An MLS range is a pair of levels, written as lowlevel-highlevel if the levels differ, or lowlevel if the levels are identical (s0-s0 is the same as s0). Each level is a sensitivity-category pair, with categories being optional. If there are categories, the level is written as sensitivity:category-set. If there are no categories, it is written as sensitivity.

If the category set is a contiguous series, it can be abbreviated. For example, c0.c3 is the same as c0,c1,c2,c3. The `/etc/selinux/targeted/setrans.conf` file is the Multi-Category Security translation table for SELinux and maps levels to human-readable form such as `s0:c0.c1023=SystemHigh`. Do not edit this file with a text editor; use the `semanage` command to make changes.

Use the `chcon` command to change the SELinux context for files. Changes made with the `chcon` command do not survive a file system relabel or the execution of the `restorecon` command. When using `chcon`, provide all or part of the SELinux context to change.

Changing the Context File Type

- SELinux requires that KVM image files have the `virt_image_t` label applied to them.

```
# # ls -dZ /var/lib/libvirt/images
drwx...  root root system_u:object_r:virt_image_t:s0
```

- To change the label on a new directory, `/virtstore`:

```
# semanage fcontext -a -t virt_image_t "/virtstore(/.* )?"
# restorecon -R -v /virtstore
# ls -dZ /virtstore
drwx...  root  root  system_u:object_r:virt_image_t:s0
```

- The `semanage` command adds the `/virtstore` directory to the SELinux targeted policy file.
- The `restorecon` command reads the policy file and updates the SELinux security contexts.

In "Lesson 14: Virtualization with Linux," you learned that KVM virtual machine disk images are created in the `/var/lib/libvirt/images` directory by default. SELinux requires that image files have the `virt_image_t` label applied to them. You can use the `ls -Z` command to confirm that this label is applied to the `/var/lib/libvirt/images` directory:

```
# ls -dZ /var/lib/libvirt/images
drwx...  root root system_u:object_r:virt_image_t:s0  /var/...
```

You can use a different directory for your virtual machine images but you need to add the new directory to your SELinux policy and relabel it first. The following steps are used to add the `/virtstore` directory to the `targeted` SELinux policy and relabel the directory:

```
# semanage fcontext -a -t virt_image_t "/virtstore(/.* )?"
```

The above command adds the `/virtstore` directory to the SELinux policy by appending a line to the following file:

```
# cat /etc/selinux/targeted/contexts/files/file_contexts.local
/virtstore(/.* )? system _u:object_r:virt_image_t:s0
```

You still need to set the new security context on the directory and all files in the directory.

You can use any of the following commands to change the SELinux contexts on the `/virtstore` directory:

- **`fixfiles`:** Fixes the security context on file systems
- **`restorecon`:** Resets the security context on one or more files
- **`setfiles`:** Initializes the security context on one or more files

Each of these commands reads the files in `/etc/selinux/targeted/contexts/files` directory.

The following example shows the SELinux contexts before running the `restorecon` command:

```
# ls -dZ /virtstore
drwx...  root  root  system_u:object_r:file_t:s0
```

Notice that the SELinux type is set to `file_t`. The following example runs the `restorecon` command to change the type as defined in the `/etc/selinux/targeted/contexts/files/file_contexts.local` file:

```
# restorecon -R -v /virtstore
# ls -dZ /virtstore
drwx...  root  root  system_u:object_r:virt_image_t:s0
```

There are also SELinux Booleans that affect KVM when launched by `libvirt`. Two of these Booleans are listed as follows:

- **`virt_use_nfs`**: Allow `virt` to manage NFS files.
- **`virt_use_samba`**: Allow `virt` to manage CIFS files.

These Booleans need to be enabled when using NFS or SAMBA shares respectively for storing virtual machine disk images. There are additional SELinux Booleans that affect KVM. For example:

```
# getsebool –a | grep virt
virt_use_comm --> off
virt_use_execmem --> off
virt_use_fusefs --> off
virt_use_nfs --> off
virt_use_samba --> off
virt_use_sanlock --> off
virt_use_sysfs --> on
virt_use_usb --> on
virt_use_xserver --> off
```

Confined SELinux Users

- Confined SELinux users and their associated domains:
  - `guest_u`: The domain for the user is `guest_t`.
  - `staff_u`: The domain for the user is `staff_t`.
  - `user_u`: The domain for the user is `user_t`.
  - `xguest_x`: The domain for the user is `xguest_t`.
- SELinux can confine Linux users by mapping Linux users to SELinux users.
- Use `semanage` to map a Linux user to an SELinux user:
  - `# semanage login -a -s user_u newuser`
- Booleans are available to change user behavior when running applications in home directories and in `/tmp`.

Linux users are mapped to the SELinux `_default_` login by default, which is mapped to the SELinux `unconfined_u` user. However, SELinux can confine Linux users, to take advantage of the security rules and mechanisms applied to them, by mapping Linux users to SELinux users.

A number of confined SELinux users exist in SELinux policy. The following is a list of confined SELinux users and their associated domains:

- **`guest_u`:** The domain for the user is `guest_t`.
- **`staff_u`:** The domain for the user is `staff_t`.
- **`user_u`:** The domain for the user is `user_t`.
- **`xguest_x`:** The domain for the user is `xguest_t`.

Linux users in the `guest_t`, `xguest_t`, and `user_t` domains can run set user ID (`setuid`) applications only if the SELinux policy permits it (such as `passwd`). They cannot run the `su` and `sudo` `setuid` applications to become the `root` user.

Linux users in the `guest_t` domain have no network access and can log in only from a terminal. They can log in with `ssh` but cannot use `ssh` to connect to another system.

The only network access Linux users in the `xguest_t` domain have is Firefox for connecting to webpages.

Linux users in the `xguest_t`, `user_t`, and `staff_t` domains can log in using the X Window System and a terminal.

By default, Linux users in the `staff_t` domain do not have permissions to execute applications with the `sudo` command.

By default, Linux users in the `guest_t` and `xguest_t` domains cannot execute applications in their home directories or `/tmp`, preventing them from executing applications in directories they have write access to. This helps prevent flawed or malicious applications from modifying files that users own.

By default, Linux users in the `user_t` and `staff_t` domains can execute applications in their home directories and `/tmp`.

Mapping Linux Users to SELinux Users

Use the `semanage login –a` command to map a Linux user to an SELinux user. For example, to map the Linux `newuser` user to the SELinux `user_u` user, run the following command:

```
    # semanage login -a -s user_u newuser
```

The `-a` option adds a new record and the `-s` option specifies the SELinux user. The last argument, `newuser`, is the Linux user that you want mapped to the specified SELinux user.

Booleans for Users Executing Applications

Some Booleans are available to change user behavior when running applications in their home directories and in `/tmp`. Use the `setsebool –P` *`<boolean>`* `on|off` command:

To allow Linux users in the `guest_t` domain to execute applications in their home directories and `/tmp`:

```
    # setsebool -P allow_guest_exec_content on
```

To allow Linux users in the `xguest_t` domain to execute applications in their home directories and `/tmp`:

```
    # setsebool -P allow_xguest_exec_content on
```

To prevent Linux users in the `user_t` domain from executing applications in their home directories and `/tmp`:

```
    # setsebool -P allow_user_exec_content off
```

To prevent Linux users in the `staff_t` domain from executing applications in their home directories and `/tmp`:

```
    # setsebool -P allow_staff_exec_content off
```

SELinux Utilities: Summary

- `sestatus`: Displays SELinux status
- `semanage`: Is an SELinux policy management tool
- `semodule`: Manages SELinux policy modules
- `setsebool`: Sets SELinux Boolean value
- `avcstat`: Displays SELinux AVC statistics
- `getenforce`: Reports the current SELinux mode
- `getsebool`: Reports SELinux Boolean values
- `seinfo`: Is an SELinux policy query tool
- `sesearch`: Is an SELinux policy query tool
- `fixfiles`: Fixes the security context on file systems
- `restorecon`: Resets the security context on files

This slide lists some of the more commonly used command-line utilities for managing and operating SELinux. More utilities are provided. The `policycoreutils` package installs the following utilities:

- **`fixfiles`:** Fixes the security context on file systems
- **`load_policy`:** Loads a new SELinux policy into the kernel
- **`restorecon`:** Resets the security context on one or more files
- **`setfiles`:** Initializes the security context on one or more files
- **`secon`:** Displays the SELinux context from a file, program, or user input
- **`semodule_deps`:** Displays the dependencies between SELinux policy packages
- **`semodule_expand`:** Expands an SELinux policy module package
- **`semodule_link`:** Links SELinux policy module packages together
- **`semodule_package`:** Creates an SELinux policy module package
- **`restorecond`:** Is a daemon that watches for file creation and sets the default file context
- **`semodule`:** Manages SELinux policy modules
- **`sestatus`:** Displays SELinux status
- **`setsebool`:** Sets SELinux Boolean value

The `libselinux-utils` package installs the following utilities:

- **`avcstat`:** Displays SELinux AVC statistics
- **`getenforce`:** Reports the current SELinux mode
- **`getsebool`:** Reports SELinux Boolean values
- **`matchpathcon`:** Queries the system policy and displays the default security context associated with the file path
- **`selinuxconlist`:** Displays all of the SELinux context reachable for a user
- **`selinuxdefcon`:** Displays the default SELinux context for a user
- **`selinuxenabled`:** Indicates whether SELinux is enabled
- **`setenforce`:** Modifies the SELinux mode
- **`togglesebool`:** Flips the current value of an SELinux Boolean

The `setools-console` package installs the following utilities:

- **`findcon`:** An SELinux file context search tool
- **`indexcon`:** An SELinux file context indexing tool
- **`replcon`:** An SELinux file context replacement tool
- **`seaudit-report`:** An SELinux audit log reporting tool
- **`sechecker`:** An SELinux policy checking tool
- **`sediff`:** An SELinux policy difference tool
- **`seinfo`:** An SELinux policy query tool
- **`sesearch`:** An SELinux policy query tool

The `policycoreutlis-python` package installs the following utilities:

- **`semanage`:** Is an SELinux policy management tool
- **`audit2allow`, `audit2why`:** Generates SELinux policy `allow/don’t_audit` rules from logs of denied operations
- **`chcat`:** Changes or removes the security category for each file or user
- **`sandbox`:** Runs a command in an SELinux sandbox

The `policycoreutlis-gui` package installs the following utilities:

- **`system-config-selinux`:** SELinux Administration GUI

Quiz

A given SELinux policy can be customized by enabling or disabling which of the following?

a. Modes
b. Policies
c. Booleans
d. Contexts
e. Labels
f. Users

Summary

In this lesson, you should have learned how to:

- Describe SELinux
- Describe SELinux packages
- Use the SELinux Administration GUI
- Describe SELinux modes
- Describe SELinux policies
- Describe SELinux Booleans
- Describe SELinux file labeling, context, and users
- Use SELinux utilities

Practice 16: Overview

The practices for this lesson cover the following:

- Exploring SELinux files and directories
- Configuring SELinux Booleans
- Configuring SELinux Context

17

Core Dump Analysis

Objectives

After completing this lesson, you should be able to:

- Describe Kexec and Kdump
- Configure Kdump to capture kernel vmcore dump
- Describe kernel parameters that can cause a panic
- Use magic SysRq keys
- Use the `crash` utility for analyzing core dumps

System Core Collection: Kexec and Kdump

- Use Kexec and Kdump to ensure the creation of reliable kernel vmcores for diagnostic purposes.
- To use Kdump, install the following package:

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

- To enable Kdump, reserve a portion of the system memory for the capture kernel.
  – Use the `crashkernel=<size>` boot parameter.

  Append to the kernel line in the GRUB configuration file.

```
kernel /vmlinuz-3.8.13-26.1.1.el6uek ...
   crashkernel=128M
```

- Turn on the Kdump init script and start the service:

```
# chkconfig kdump on
# service kdump start
```

Kdump is the Linux kernel crash dumping mechanism. In the event of a system crash, Kdump provides a memory dump (vmcore) image. This image can assist in determining the cause of the crash. It is highly recommended that you enable the Kdump feature.

Kexec and Kdump together ensure faster bootup and the creation of reliable kernel vmcores for diagnostic purposes. Kexec is a fast-boot mechanism that allows booting a Linux kernel from the context of an already running kernel without going through BIOS. Kdump uses Kexec to boot into a second kernel whenever the system crashes. The crash dump is captured from the context of a freshly booted kernel and not from the context of the crashed kernel. This second kernel boots with very little memory and captures the dump image.

To enable and use Kdump, install the following package:

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

Enabling Kdump requires you to reserve a portion of the system memory for the capture kernel. This portion of memory is unavailable for other uses. The amount of memory that is reserved for the Kdump kernel is represented by the `crashkernel` boot parameter. This is appended to the kernel line in the GRUB configuration file, `/boot/grub/grub.conf`. The following example enables Kdump and reserves 128 MB of memory:

```
kernel /vmlinuz-3.8.13-26.1.1.el6uek ... crashkernel=128M
```

If Kdump fails to start, the following error appears in `/var/log/messages`:

```
kdump: No crashkernel parameter specified for running kernel
```

In addition to reserving memory, you can designate the starting address (offset) of this reserved memory. For example, adding the following option to the kernel line reserves 128 MB of memory, starting at physical address 0x01000000 (16 MB):

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

To set the offset to `48M`:

```
crashkernel=128M@48M
```

If you have more than 128 GB RAM, use the following setting:

```
crashkernel=512M@64M
```

If more control is needed over the size and placement of the reserved memory, use the following format:

```
crashkernel=range1:size1[,range2:size2,...][@offset]
```

The `range<n>` value specifies a range of values that are matched against the amount of physical RAM present in the system. The corresponding `size<n>` value specifies the amount of Kexec memory to reserve.

The following example tells Kexec to reserve 64 MB of RAM if the system contains between 512 MB and 2 GB of memory. If the system contains more than 2 GB of physical memory, reserve 128 MB:

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

To simplify Kdump configuration, support has been added for the `crashkernel=auto` kernel parameter. For Xen, this parameter is supported only for Domain 0. If this parameter is enabled, the output of the `dmesg` command shows `crashkernel=XM@0M`. This is normal.

After adding the `crashkernel` parameter to the `/boot/grub/grub.conf` file, reboot your system so that memory is reserved for the capture kernel. The `free -m` command correctly shows that less memory is available for the system.

Use the `chkconfig` command to start the Kdump service at boot time. Use the `service` command to start the Kdump service immediately:

```
# chkconfig kdump on
# service kdump start
```

This loads your kernel-kdump image via Kexec, leaving your system ready to capture a vmcore on crashing. You can test by force-crashing your system using the following command:

```
# echo c > /proc/sysrq-trigger
```

This causes panic output to be displayed, followed by the system restarting into the Kdump kernel. When the boot process gets to the point where it starts the Kdump service, the vmcore is copied to disk to the default location, `/var/crash/<YYYY-MM-DD-HH:MM>/vmcore`. The system then reboots back into the normal kernel.

Note that Kdump is not supported on Xen domU guests. Virtualized systems can use `xm dump-core` command for panics.

Kdump Configuration File

- The configuration file for Kdump is `/etc/kdump.conf`.
- By default, the target location to write the vmcore is the `/var/crash` directory on the local system.
  - Set the `path` directive to change this.
- Capture only the bare essentials for kernel fault debugging and compress the core:

```
core_collector makedumpfile -d 31 -c
```

  - This strips zero pages, pagecache pages, and user pages.
  - Compressing turns a 64 GB core into around 500 MB.
  - Always compress before uploading to Oracle Support.
- The default action to take, in case dumping to an intended target fails, is to reboot.
  - Set the `default` directive to change this.

The configuration file for Kdump is `/etc/kdump.conf`. The default target location for the vmcore is the `/var/crash` directory on the local file system, which is represented as follows:

```
path /var/crash
```

To change the target location, edit this file and specify the new target. To write to a different local directory, edit the `path` directive and provide the directory path. To write to a different partition, specify the type of file system followed by an identifier.
Example:

```
path /
ext4 UUID=bff248...
```

To write directly to a device, edit the `raw` directive and specify the device name.
Example:

```
raw /dev/sda1
```

To write to a remote system by using NFS, use the `net` directive followed by the FQDN of the remote system, then a colon (`:`), and then the directory path. Example:

```
net host01.example.com:/export/crash
```

To write to a remote machine by using SSH, use the `net` directive followed by a valid username, the `@` sign, and the host name, in that order.
Example:

```
    net root@host02.example.com
```

Modify the filtering level for the vmcore dump using the `core_collector` directive in the `/etc/kdump.conf` file. To exclude certain pages from the dump, use the `-d <value>` parameter where `<value>` is a sum of values of the pages that you want to exclude. Use the following values for the pages:

- **`1:`** zero page
- **`2:`** cache page
- **`4:`** cache private
- **`8:`** user data
- **`16:`** free page

The recommendation is to exclude all these pages as follows. Add the values (the total for all is `31`) and provide the sum as the argument:

```
    core_collector makedumpfile -d 31 -c
```

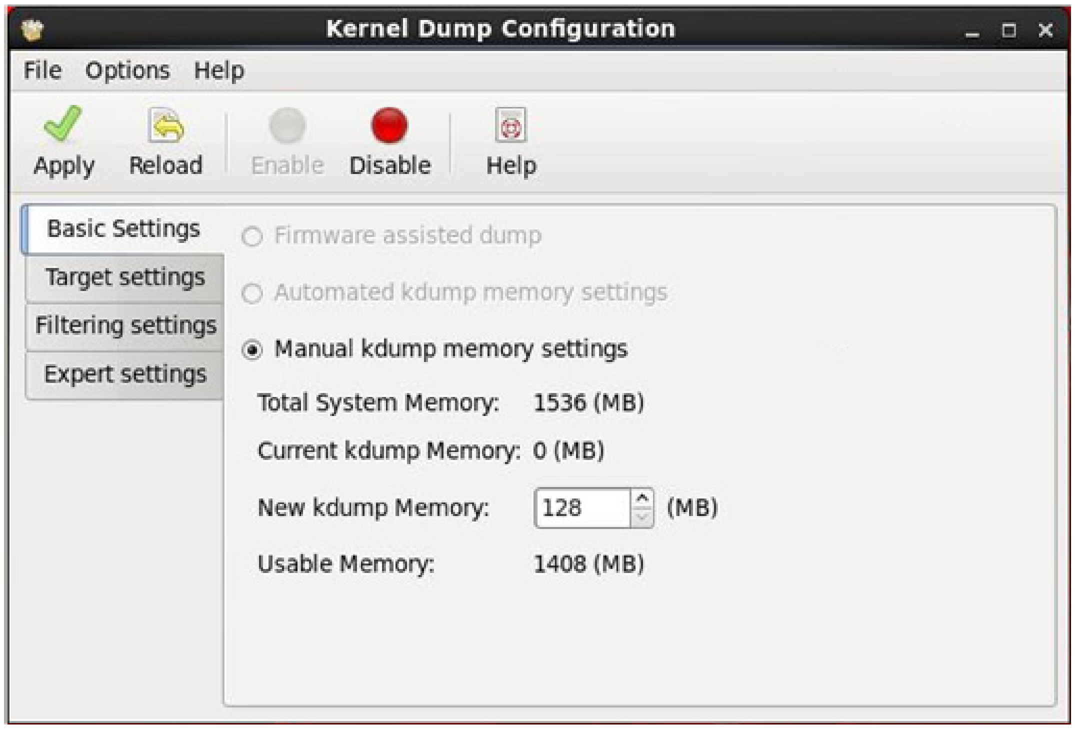
The `-c` option enables dump file compression. To exclude only zero (`1`) and free (`16`) pages:

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

The default action to take if dumping to the intended target fails is to reboot. Other possible actions are `halt`, `poweroff`, `shell`, or `mount_root_run_init`, which means mount the root file system and run `/sbin/init`. To change this, set the `default` directive in `/etc/kdump.conf`, as in this example:

```
    default poweroff
```

Kdump Setup Configuration GUI



○ Firmware assisted dump

○ Automated kdump memory settings

● Manual kdump memory settings

Total System Memory: 1536 (MB)

Current kdump Memory: 0 (MB)

New kdump Memory: 128 (MB)

Usable Memory: 1408 (MB)

You can also enable Kdump from a GUI. Enter the following command to use the Kernel Dump Configuration GUI:

```
# system-config-kdump
```

The GUI appears as shown in the slide. Click the Enable button to configure the `kdump` daemon to start at boot time for run levels 2, 3, 4, and 5.

Four tabs appear on the left side of the GUI. The Basic Settings tab allows you to select the amount of memory to reserve for Kdump.

Use the Target Settings tab to specify the target location for the vmcore dump. You can store the dump image in a local file system or store it remotely using NFS or SSH. The default is to store the vmcore file in the `/var/crash` directory of the local file system. The following targets are supported:

- **Raw device:** All locally attached raw disks and partitions
- **Local file system:** Any ext2, ext3, ext4, or minix file system on directly attached disk drives, hardware RAID logical drives, LVM devices, and mdraid arrays
- **Remote directory:** Remote directories accessed using NFS or SSH over IPv4 and remote directories accessed using iSCSI over hardware initiators

Unsupported targets include:

- **Local file system:** The `eCryptfs` file system
- **Remote directory:** Remote directories on the `rootfs` file system accessed using NFS, remote directories accessed using iSCSI over software initiators, remote directories accessed over IPv6, remote directories accessed using SMB/CIFS or FCoE (Fibre Channel over Ethernet), remote directories accessed using wireless NICs, and multipath-based storages

The Filtering Settings tab allows you to select the filtering level for the vmcore dump. You can choose to exclude any or all of the following from the dump:

- zero page
- cache page
- cache private
- user data
- free page

The Expert Settings tab allows you to choose which kernel and initial RAM disk to use. From this tab you can also customize the options that are passed to the kernel and the core collector program. And you can choose what to do when the kernel crash is captured. The following options are available:

- **`mount rootfs and run /sbin/init`:** This is the default action.
- **`reboot`:** Reboot the system.
- **`shell`:** Present a user with an interactive shell prompt.
- **`halt`:** Halt the system.
- **`poweroff`:** Power the system off.

Click Apply to save any changes.

`netdump` Utility

- A kernel crash log is written to a server on the network.
- A `netdump` server listens on the network for crashed kernels to contact it.
- Configure the server:
  - Install the `netdump-server` package.
  - `# service netdump-server start`
  - `# passwd netdump`
- Configure the client:
  - Install the `netdump` package.
  - Edit `/etc/sysconfig/netdump`.
    - `NETDUMPADDR=<`*`netdump-server IP`*`>`
  - `# service netdump propagate`
  - `# service netdump start`

The `netdump` utilities allow you to configure a `netdump` server that listens on the network for crashed kernels to contact it. The server then writes the oops log and a memory dump to `/var/netdump/crash` before asking the crashed machine to reboot. The system that crashes is a `netdump` client. Following are a couple of scenarios that would benefit from using `netdump`.

An oops is a deviation from correct behavior of the kernel that produces a certain error log. When the Linux kernel detects a problem, it prints an oops message and kills any offending process. Diagnosing the cause of a panic requires that the messages sent by the kernel be captured for analysis. If the serial console is overwhelmed or the disks are not responding, then these messages can be lost.

Under a few specific circumstances, Oracle Clusterware might evict a cluster, resulting in a kernel panic. This is to prevent a node from trying to update data without proper synchronization. Loss of heartbeat in a shared disk subsystem can also cause a panic. OCFS2 causes a panic if communication is lost to the cluster or to the shared disk subsystem.

Configuring a `netdump` Server

The `netdump` server is the location where the failing machine writes the dump file and associated messages. Install the `netdump-server` package on the server.

There is no `netdump-server` package in the base Oracle Linux 6 repository. You can get the package from Extra Packages for Enterprise Linux (EPEL) at:

http://dl.fedoraproject.org/pub/epel/6/x86_64/repoview/netdump-server.html

After installing the server package, start the server by using the following command:

```
# service netdump-server start
```

You must then set the `netdump` user password by entering the following command as `root`:

```
# passwd netdump
Changing password for user netdump.
New UNIX password: *******
Retype new UNIX password: ******
passwd: all authentication tokens updated successfully.
```

Set up `netdump-server` to start at boot time by entering the following command:

```
# chkconfig netdump-server on
```

Configuring a `netdump` Client

Install the `netdump-<version>.rpm` on the client machine. Edit the configuration file, `/etc/sysconfig/netdump` and set the `NETDUMPADDR` directive to the IP address of the `netdump` server. Example:

```
NETDUMPADDR=<netdump-server IP>
```

To allow the `netdump` client to connect to the `netdump` server, propagate the key by running the following command (be prepared to provide the `netdump` user password):

```
# service netdump propagate
The authenticity of host ... can't be established
RSA key fingerprint is ...
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added ... (RSA) to the list of known hosts.
netdump@... password:*****
```

Set up the `netdump` client to start at boot as follows:

```
# chkconfig netdump on
```

Start the `netdump` client service as follows:

```
# service netdump start
initializing netdump                                   [ OK ]
initializing netconsole                                [ OK ]
...
```

Notice that the `netconsole` service is started with `netdump` client. This module provides logging of kernel events over UDP allowing debugging of problems where disk logging fails and serial consoles are impractical.

Kernel Tuning Parameters

- The following parameters can conditionally bring down the system.
- Out of memory:
  - `vm.panic_on_oom` – Panics when out of memory
  - `kernel.panic_on_oops` – Panics when a BUG occurs
- Hung Task Monitoring
  - `kernel.hung_task_timeout_secs` – Displays a message if tasks are not scheduled within the timeout period
  - `kernel.hung_task_panic` – Panics if the preceding condition is hit
  - Gives a point-in-time crash core for further analysis

The `vm.panic_on_oom` parameter (`/proc/sys/vm/panic_on_oom`) enables or disables a kernel panic in an out-of-memory (OOM) situation. When set to `0` (default), the kernel's OOM-killer scans through the entire task list and attempts to kill some rogue memory-hogging process to avoid a panic. When set to `2`, the kernel always panics when an OOM condition occurs. When set to `1`, the kernel normally panics but can survive in certain conditions. If a process limits allocations to certain nodes using memory policies or cpusets, and those nodes reach memory exhaustion status, one process can be killed by the OOM-killer. No panic occurs in this case because other nodes' memory might be free. This means that the system as a whole might not have reached an out-of-memory condition yet. Settings of `1` and `2` are for failover of clustering. Select the setting according to your failover policy.

The `kernel.panic_on_oops` parameter (`/proc/sys/kernel/panic_on_oops`) controls the kernel's behavior when an oops or BUG is encountered. When set to `0`, the system tries to continue operations. When set to `1` (default), the system delays a few seconds (to give the `klogd` kernel log daemon time to record the oops output), and then panics.

The `kernel.hung_task_timeout_secs` parameter (in the `/proc/sys/kernel/` directory) is set to `120` by default. This causes a message to be generated when a task is stuck in D (disk sleep) state for 120 seconds. A process is put in D state while waiting for `read()` or `write()` return. A process cannot be killed or interrupted while in D state.

If the `kernel.hung_task_panic` parameter (in the `/proc/sys/kernel/` directory) is enabled by setting the value to `1`, it causes the kernel to panic if any user or kernel thread sleeps in state TASK_UNINTERRUPTIBLE for more than `kernel.hung_task_timeout_secs` seconds. The default setting for the `kernel.hung_task_panic` parameter is `0`, or disabled.

Magic SysRq Keys

Magic SysRq Key	How to Invoke Magic SysRq
M: Dump **system memory** statistics.	**Console**: Alt + Sys Rq + <cmd>
P, W: Dump the stack for all **processors.**	**Serial Console:** <Break> <cmd>
T: Dump the **kernel stack trace** for all processes.	**Command Line:** echo t > /proc/sysrq-trigger
C: Immediately cause a system **crash.**	**Oracle VM dom0:** xm sysrq <domain ID> <cmd>
S … U… B: Emergency Sync all disks, Unmount disks, reboot	**Ensure kernel.sysrq = 1**
Some of these operations (such as stack trace) dump a lot of data (1 MB or more).	**These operations take full priority in the kernel. Use these carefully. Do not run them in your monitoring scripts.**

The magic SysRq (system request) key is a key combination that is understood by the Linux kernel, which allows you to execute low-level commands regardless of the system's state. It is often used to recover from, or diagnose, a system hang. The slide lists some of the common uses and the method to invoke. The left column lists the Magic SysRq keys and right column lists the different methods of invoking the keys. For example, to cause a system crash of an OVM guest, determine the domain ID of the guests as follows:

```
[dom0]# xm list
```

Assuming that the domain ID is `60`, use the following command from dom0 to crash the VM guest:

```
[dom0]# xm sysrq 60 c
```

The magic SysRq feature is controlled by the `kernel.sysrq` kernel parameter (`/proc/sys/kernel/sysrq`). Set the value to `1` to enable, or to `0` (default) to disable.

`crash` Utility

- Use the `crash` utility to analyze the state of the system while it is running or after a kernel crash has occurred.
  - Crash is merged with the GNU Debugger (`gdb`).
  - Crash can analyze core dumps created by multiple facilities.
- Syntax for crash:

```
# crash <vmcore> <kernel-debug data>
```

  - `vmcore` – The memory image
  - `vmlinux` – Part of the `kernel-debuginfo` package
- Download the `kernel-debuginfo` package from oss.oracle.com.
- The `vmlinux` file is installed in the `/usr/lib/debug/lib/modules/<kernel>` directory.

The `crash` utility allows you to analyze the state of the Linux system while it is running or after a kernel crash has occurred. The utility has been merged with `gdb`, the GNU Debugger, so `crash` includes source code-level debugging capabilities. The `crash` utility is used to analyze core dumps created by the `kdump`, `netdump`, `diskdump`, `xm dump-core`, Linux Kernel Crash Dumps (LKCD), and `vissh dump` facilities. The syntax for `crash` is as follows:

```
crash [options] [vmlinux] [vmcore]
```

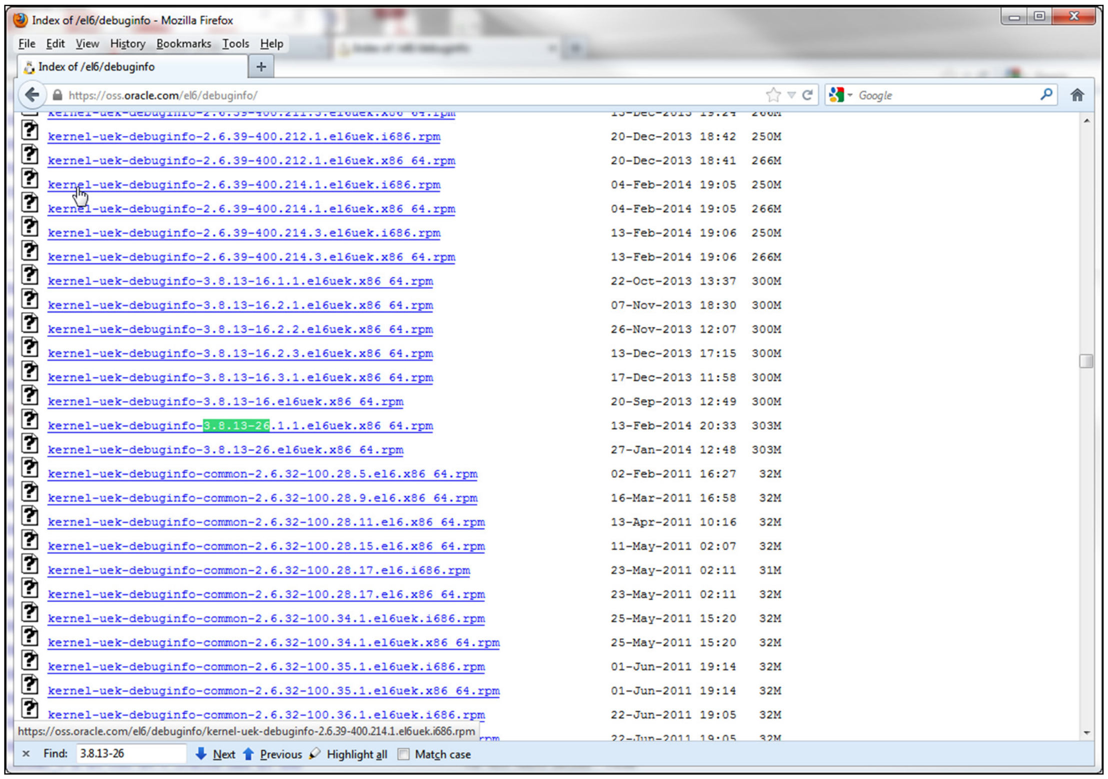
Optional <*vmlinux*> and <*vmcore*> arguments:

- `vmlinux` – A `vmlinux` kernel object file, often referred to as the *namelist*. The `vmlinux` file is part of the `kernel-debuginfo` package. This file is installed in the `/usr/lib/debug/lib/modules/<kernel>` directory.
- `vmcore` – The memory image, which is a kernel crash dump file created by any of the supported core dump facilities. Omit the `vmcore` argument to invoke the `crash` utility on a live system.

The following is an example with both <*vmlinux*> and <*vmcore*> arguments:

```
# crash /usr/lib/debug/lib/modules/3.8.13-
26.1.1.el6uek.x86_64/vmlinux /root/2014-0220-0525.00-
host03.249.core
```

Downloading `kernel-debuginfo` RPM Packages


Index of /el6/debuginfo - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Index of /el6/debuginfo
https://oss.oracle.com/el6/debuginfo/
Google

```
kernel-uek-debuginfo-2.6.39-400.212.1.el6uek.i686.rpm              20-Dec-2013 18:42  250M
kernel-uek-debuginfo-2.6.39-400.212.1.el6uek.x86_64.rpm            20-Dec-2013 18:41  266M
kernel-uek-debuginfo-2.6.39-400.214.1.el6uek.i686.rpm              04-Feb-2014 19:05  250M
kernel-uek-debuginfo-2.6.39-400.214.1.el6uek.x86_64.rpm            04-Feb-2014 19:05  266M
kernel-uek-debuginfo-2.6.39-400.214.3.el6uek.i686.rpm              13-Feb-2014 19:06  250M
kernel-uek-debuginfo-2.6.39-400.214.3.el6uek.x86_64.rpm            13-Feb-2014 19:06  266M
kernel-uek-debuginfo-3.8.13-16.1.1.el6uek.x86_64.rpm               22-Oct-2013 13:37  300M
kernel-uek-debuginfo-3.8.13-16.2.1.el6uek.x86_64.rpm               07-Nov-2013 18:30  300M
kernel-uek-debuginfo-3.8.13-16.2.2.el6uek.x86_64.rpm               26-Nov-2013 12:07  300M
kernel-uek-debuginfo-3.8.13-16.2.3.el6uek.x86_64.rpm               13-Dec-2013 17:15  300M
kernel-uek-debuginfo-3.8.13-16.3.1.el6uek.x86_64.rpm               17-Dec-2013 11:58  300M
kernel-uek-debuginfo-3.8.13-16.el6uek.x86_64.rpm                   20-Sep-2013 12:49  300M
kernel-uek-debuginfo-3.8.13-26.1.1.el6uek.x86_64.rpm               13-Feb-2014 20:33  303M
kernel-uek-debuginfo-3.8.13-26.el6uek.x86_64.rpm                   27-Jan-2014 12:48  303M
kernel-uek-debuginfo-common-2.6.32-100.28.5.el6.x86_64.rpm         02-Feb-2011 16:27   32M
kernel-uek-debuginfo-common-2.6.32-100.28.9.el6.x86_64.rpm         16-Mar-2011 16:58   32M
kernel-uek-debuginfo-common-2.6.32-100.28.11.el6.x86_64.rpm        13-Apr-2011 10:16   32M
kernel-uek-debuginfo-common-2.6.32-100.28.15.el6.x86_64.rpm        11-May-2011 02:07   32M
kernel-uek-debuginfo-common-2.6.32-100.28.17.el6.i686.rpm          23-May-2011 02:11   31M
kernel-uek-debuginfo-common-2.6.32-100.28.17.el6.x86_64.rpm        23-May-2011 02:11   32M
kernel-uek-debuginfo-common-2.6.32-100.34.1.el6uek.i686.rpm        25-May-2011 15:20   32M
kernel-uek-debuginfo-common-2.6.32-100.34.1.el6uek.x86_64.rpm      25-May-2011 15:20   32M
kernel-uek-debuginfo-common-2.6.32-100.35.1.el6uek.i686.rpm        01-Jun-2011 19:14   32M
kernel-uek-debuginfo-common-2.6.32-100.35.1.el6uek.x86_64.rpm      01-Jun-2011 19:14   32M
kernel-uek-debuginfo-common-2.6.32-100.36.1.el6uek.i686.rpm        22-Jun-2011 19:05   32M
https://oss.oracle.com/el6/debuginfo/kernel-uek-debuginfo-2.6.39-400.214.1.el6uek.i686.rpm
```

To use the `crash` utility, you must install the `crash` and `kernel-debuginfo` RPM packages. The `crash` RPM is included in the Oracle Linux distribution. Download the `kernel-debuginfo` RPM packages from https://oss.oracle.com/ as shown in the slide.

You need to determine the kernel version and architecture of the vmcore dump file to know which `kernel-debuginfo` RPM packages to download and install. If you have access to the system that produced the vmcore, use the `uname -r` command to determine the kernel version and architecture as follows:

```
# uname -r
3.8.13-26.1.1.el6uek.x86_64
```

In this example, the kernel version is `3.8.13-26.1.1.el6uek` and the architecture is `x86_64`. If you have only the vmcore file, use the `strings <vmcore>` command as follows:

```
# strings <vmcore> | less
...
/vmlinuz-3.8.13-26.1.1.el6uek.x86_64 ro root=/dev/mapper...
...
```

Browse through the output until you see the kernel version and system architecture.

Download the matching `debuginfo` and `debuginfo-common` RPMs for the desired kernel version and architecture. You might need to check multiple URLs to locate the correct kernel version and architecture. For Oracle Linux 4.5 and earlier, browse to:

- http://kernel.us.oracle.com/kernels

For Oracle Linux 4 (version 4.6 and later), browse to:

- http://oss.oracle.com/el4/debuginfo

For Oracle Linux 5, browse to:

- http://oss.oracle.com/el5/debuginfo

For Oracle Linux 6, browse to:

- http://oss.oracle.com/el6/debuginfo

The slide highlights the `debuginfo` RPM package for kernel version `3.8.13-26.1.1.el6uek.x86_64`. For this kernel version and architecture, download the following two packages:

```
kernel-uek-debuginfo-3.8.13-26.1.1.el6uek.x86_64
kernel-uek-debuginfo-common-3.8.13-26.1.1.el6uek.x86_64
```

You can either extract only the `vmlinux` executable from the `kernel-debuginfo` RPM, or install both RPMs on the system on which you are analyzing the core dump file.

Use the `yum update` command to install the latest version of `crash`, which supports dumps of systems running UEK2 and above:

```
# yum update crash
```

Initial `crash` Output

```
      KERNEL: /usr/lib/debug/lib/modules/3.8.../vmlinux
    DUMPFILE: /root/2014-0220-0525.00-host03.249.core
        CPUS: 1
        DATE: Thu Feb 20 05:15:36 2014
      UPTIME: 11:19:18
LOAD AVERAGE: 0.00, 0.01, 0.05
       TASKS: 215
    NODENAME: host03.example.com
     RELEASE: 3.8.13-26.1.1.el6uek.x86_64
     VERSION: #2 SMP Thu Feb 13 19:42:43 PST 2014
     MACHINE: x86_64  (2992 Mhz)
      MEMORY: 1.5 GB
       PANIC: ""
         PID: 0
     COMMAND: "swapper"
        TASK: ffffffff818ae420 [THREAD_INFO: ...
         CPU: 0
       STATE: TASK_RUNNING (ACTIVE)
     WARNING: panic task not found
```

The slide displays an example of information that is initially displayed when `crash` is run. You can learn a lot about the state of a system from this initial output. The number of CPUs and the load average over the last 1 minute, last 5 minutes, and last 15 minutes are displayed. The number of tasks running, the amount of memory, the panic string, the command executing at the time the core dump was created, and additional information are displayed.

In this example, the system did not panic. The core dump file was created using the `xm dump-core` command. The following example displays some of the initial `crash` information for an actual panic:

```
DUMPFILE: tmp/vmcore
   PANIC: "Oops: 0002" (check log for details)
     PID: 1696
 COMMAND: "insmod"
    TASK: c74de000
     CPU: 0
   STATE: TASK_RUNNING (PANIC)
```

In this example, an `insmod` attempt to install a module resulted in an “oops” violation.

Using the `crash` Utility

- The `crash` utility provides an interactive prompt from which commands are executed:

```
crash>
```

- Use the `?` or `help` command to view a list of commands.
- Crash commands fall into the following categories:
  - Symbolic display – Displays kernel text and data structures
  - System state – Includes kernel-aware commands that examine various kernel subsystems on a system-wide or per-task basis
  - Utility functions – Include useful helper commands such as a calculator, translator, and search function.
  - Session control – Includes commands that control the crash session itself

After the initial `crash` output is displayed, the `crash` utility provides an interactive prompt from which `crash` commands are executed:

```
# crash /usr/lib/debug/lib/modules/3.8.13-
26.1.1.el6uek.x86_64/vmlinux /root/2014-0220-0525.00-
host03.249.core

...

crash>
```

From the `crash>` prompt, you can enter `help` or `?` to display the `crash` commands. You can also enter `help` *`<command>`* to display usage information for a specific command. Each `crash` command falls into one of the categories listed in the slide.

Symbolic Display crash Commands

This category of crash commands displays kernel data structures. Available commands are:

- struct – Displays a structure definition or the contents of a structure at a specified address
- union – Is the same as struct, but used for kernel data types that are defined as unions instead of structures
- * – Is the “Pointer-to” command that is used instead of struct or union. The gdb module determines whether the argument is a structure or a union, and then calls the appropriate function.
- p – Displays the contents of a kernel variable
- sym – Translates a kernel symbol name to its kernel virtual address or vice versa
- dis – Disassembles the source code instructions of a kernel function

The following symbolic display crash commands take advantage of the gdb integration to display kernel data structures symbolically:

- struct – Displays a structure definition or provides a formatted display of the contents of a structure at a specified address. Example:

```
crash> struct cpu
struct cpu {
     int node;
     int hotpluggable;
     struct sys_device sysdev;
}
```

- union – Is the same as the struct command, but is used for kernel data types that are defined as unions instead of structures
- * – Is the “Pointer-to” command, which can be used instead of entering struct or union. The gdb module first determines whether the argument is a structure or a union, and then calls the appropriate function.

The following is an example of using the “Pointer-to” command:

```
crash> *page
struct page {
     long unsigned int flags;
     struct address_space *mapping;
     struct {
          union {
               long unsigned int index;
          };
     ...
}
SIZE: 64
```

- `p` – Displays the contents of a kernel variable. The arguments are passed on to the `gdb print` command for proper formatting. Example:

```
crash> p init_mm
init_mm = $7 = {
     mmap = 0x0,
     mm_rb = {
          rb_node = 0x0
     };
mmap_cache = 0x0,
...
```

- `whatis` – Displays the definition of structures, unions, typedefs, or text/data symbols. Example:

```
crash> whatis linux_binfmt
struct linux_binfmt {
     struct linux_binfmt *next;
     struct module *module;
     int (*load_binary) ();
     int (*load_shlib) ();
     int (*core_dump) ();
};
```

- `sym` – Translates a kernel symbol name to its kernel virtual address and section, or a kernel virtual address to its symbol name and section. You can also use this command to dump the complete list of kernel symbols (`-l`), or to query (`-q`) the symbol list for all symbols containing a given substring.
- `dis` – Disassembles the source code instructions of a complete kernel function, or from a specified address for a given number of instructions, or from the beginning of a function up to a specified address

System State `crash` Commands

These `crash` commands are "kernel-aware" commands that display various kernel subsystems on a system-wide or per-task basis. Available commands are:

- `bt` – Displays a kernel stack backtrace of the current context. This is probably the most useful `crash` command.
- `dev` – Displays character and block device data, I/O port and memory usage (`-i`), and disk I/O statistics (`-d`)
- `files` – Displays information about open files
- `fuser` – Displays file users, or the tasks that reference a specified file name or inode address
- `irq` – Displays IRQ (interrupt request) data
- `kmem` – Displays several kernel memory subsystems such as `/proc/slabinfo` data at the time of the crash (`-s`)

The following system state `crash` commands are "kernel-aware" commands that display various kernel subsystems on a system-wide or per-task basis:

- `bt` – Displays a kernel stack backtrace of the current context. This is probably the most useful `crash` command. Because the initial context is the panic context, it shows the function trace leading up to the kernel panic. Example:

```
crash> bt
PID: 0  TASK: ffffffff818ae420  CPU: 0  COMMAND: "swapper/0"
 #0 [ffffffff8189be38] __schedule at ffffffff81591d12
 #1 [ffffffff8189bec0] default_idle at ffffffff8101e35d
 #2 [ffffffff8189bee0] cpu_idle at ffffffff8101dc19
```

- `dev` – Displays character and block device data. Example:

```
crash> dev
CHRDEV  NAME              CDEV        OPERATIONS
   1    mem        ffff88005c803840  memory_fops
   4    /dev/vc/0  ffffffff81dc7560  console_fops
...
```

- `files` – Displays information about open files. This command is context-sensitive, meaning that it acts on the current context unless a PID or task address is specified as an argument. Example:

```
crash> files
PID: 0  TASK: ffff88005b290680  CPU: 0  COMMAND: “crash”
ROOT: /  CWD: /
FD         FILE              DENTRY          INODE
0 ffff8800599c... Ffff88003e65... Ffff88005a11... CHR /dev/pts/0
...
```

- `fuser` – Displays the tasks that reference a specified file name or inode address as the current root or working directory, an open file descriptor, or that mmap the file. Example:

```
crash> fuser /dev/pts/0
PID         TASK        COMM       USAGE
2394  ffff88005b3544c0  “bash”     fd
3470  ffff88005b3544c0  “crash”    fd
```

- `irq` – Displays interrupt request (IRQ) data. Example:

```
crash> irq
IRQ   IRQ_DESC/_DATA     IRQACTION      NAME
 0   ffff88005da74080 ffffffff81786000  “timer”
...
```

- `kmem` – Displays the state of several kernel memory subsystems. This command accepts a number of options. For example, the `-i` option displays general memory usage information:

```
crash> kmem -i
             PAGES     TOTAL      PERCENTAGE
 TOTAL MEM  384258    1.5 GB         -----
      FREE   22955   89.7 MB     5% of TOTAL MEM
      USED  361303    1.4 GB    94% of TOTAL MEM
...
TOTAL SWAP  770047    2.9 GB         -----
 SWAP USED      45    180 KB    0% of TOTAL SWAP
 SWAP FREE  770002    2.9 GB   99% of TOTAL SWAP
crash> kmem -s
CACHE            NAME        OBJSIZE ALLOCATED TOTAL SLABS SSIZE
ffff88005b762fc0 rpc_buffers    2048         8     8     4    4k
ffff88005ae62f80 rpc_tasks       256         8    15     1    4k
...
```

System State `crash` Commands

Additional “kernel-aware” commands:

- `log` – Displays the kernel message buffer chronologically. This command gives more data than the `dmesg` command.
- `mach` – Displays machine-specific data such as cpuinfo structure (`-c`) and physical memory map (`-m`)
- `mod` – Displays the list of currently installed kernel modules and can also load (`-s` or `-S`) and unload (`-d`) debug data
- `mount` – Displays information about currently mounted file systems
- `net` – Displays network-related data
- `ps` – Displays process status information
- `pte` – Translates the contents of a page table entry (PTE)
- `runq` – Displays the list of tasks on the run queue

Additional system state category of `crash` commands:

- `log` – Displays the kernel message buffer chronologically. This is the same data that is displayed with the `dmesg` command but this output can include console logs that never reached the console ringbuffer and messages that were not written to syslog/disk.
- `mach` – Displays machine-specific data such as cpuinfo structure (`-c`) and physical memory map (`-m`). Example:

```
crash> mach
   MACHINE TYPE: x86_64
    MEMORY SIZE: 1.5 GB
           CPUS: 1
PROCESSOR SPEED: 2992 Mhz
...
```

- `mod` – Displays the list of currently installed kernel modules. With the `-s` or `-S` options, it loads the debug data from the module object files if they are available, allowing the symbolic debugging capability of kernel modules. The `-d` option deletes the symbolic and debugging data of the specified module.

- `mount` – Displays information about currently mounted file systems such as `vfsmount` structure address, `super_block` structure address, file system type, device name, and mount point. Example:

```
crash> mount
   VFSMOUNT          SUPERBLK      TYPE    DEVNAME  DIRNAME
ffff88005c735d80 ffff88005c738800  rootfs  rootfs   /
ffff88005874cdc0 ffff88005c741c00  proc    proc     /proc
...
```

- `net` – Displays network-related data such as ARP cache, open network socket addresses, `net_device` address, IP addresses of each network device, and other information. Example:

```
crash> net
   NET_DEVICE     NAME     IP ADDRESS(ES)
ffff88005c30e000  lo       127.0.0.1
ffff88005bb28000  eth0     192.0.2.105
...
```

- `ps` – Displays process status information, but not as comprehensively as the `bt` command. The active task is highlighted by the `>`. Example:

```
crash> ps
PID  PPID CPU       TASK          ST  %MEM    VSZ  RSS  COMM
> 0     0  0   ffffffff81781020  RU   0.0      0    0  [swapper]
  1     0  0   ffff88005c09a040  IN   0.1  19396 1424  init
...
```

- `pte` – Translates the hexadecimal contents of a page table entry (PTE) into its physical page address and page bit settings. If the PTE references a swap location, it displays the swap device and offset. Example:

```
crash> pte ffff88005b762fc0
      PTE           PHYSICAL     FLAGS
ffff88005b762fc0  f88005b762000  (DIRTY|PROTNONE|GLOBAL|NX)
crash> pte ffff88005ae62f80
      PTE           PHYSICAL     FLAGS
ffff88005ae62f80  f88005ae62000  (DIRTY|GLOBAL|NX)
```

- `runq` – Displays the list of tasks on the run queue of each CPU. Example:

```
crash> runq
CPU 0 RUNQUEUE: ffff88005fc121c0
  CURRENT: PID: 0  TASK: ffffffff81781020  COMMAND: “swapper”
  RT PRIO_ARRAY: ffff88005fc12310
    [no tasks queued]
```

System State `crash` Commands

Additional "kernel-aware" commands:

- `sig` – Displays a task's signal-handling data. You can include multiple task or PID numbers as arguments.
- `swap` – Displays information for each configured swap device. The `swapon -s` command displays the same data.
- `sys` – Displays the system information shown during crash initialization, or the system call table entries (`-c`)
- `task` – Displays the contents of `task_struct`. Multiple task or PID numbers can be entered.
- `timer` – Displays the timer queue entries
- `vm` – Displays a task's virtual memory data
- `vtop` – Translates a user or kernel virtual address to its physical address
- `waitq` – Displays tasks blocked on the specified wait queue

The remaining system state category of `crash` commands:

- `sig` – Displays a task's signal-handling data. You can include multiple task or PID numbers as arguments. Example:

```
crash> sig
PID: 0   TASK: ffffffff81781020   COMMAND: "swapper"
SIGNAL_STRUC: ffffffff81782960   NR_THREADS: 1
 SIG     SIGACTION       HANDLER         MASK          FLAGS
  [1]  ffffffff81782d88  SIG_DFL  0000000000000000  0
  [2]  ffffffff81782da8  SIG_DFL  0000000000000000  0
...
```

- `swap` – Displays information for each configured swap device. The `swapon -s` command displays the same data in a slightly different format. Example:

```
crash> swap
FILENAME         TYPE         SIZE     USED    PCT    PRIORITY
/dm-1          PARTITION   3080188k     0k     0%        -1
```

- sys – Displays the system information shown during crash initialization, or the system call table entries (-c). Example:

```
crash> sys -c
NUM  SYSTEM CALL
  0  sys_read             ../fs/read)write.c: 403
  1  sys_write            ../fs/read)write.c: 421
```

- task – Displays the contents of task_struct. Each task_struct data structure describes a process or task in the system. You can enter multiple task or PID numbers. Example:

```
crash> task
PID: 0  TASK: ffffffff81781020  CPU: 0   COMMAND: "swapper"
struct task_struct {
     state = 0,
     stack = 0xffffffff81776000,
     usage = {
          counter = 2
     },
     flags = 4202752,
...
```

- timer – Displays the timer queue entries in chronological order
- vm – Displays a task's virtual memory data, including mm_struct address, page directory address, resident set size, and total virtual memory size. Example:

```
crash> vm
PID: 0  TASK: ffff88005b352680  CPU: 0   COMMAND: "crash"
      MM                PGD             RSS    TOTAL_CM
ffff880037fa62c0  ffff8800589bb000  165980k  301640k
      VMA          START      END       FLAGS   FILE
ffff88005bb69608   400000    a07000  8001875  /usr/bin/crash
...
```

- vtop – Translates a user or kernel virtual address to its physical address. Other information such as the PTE translation, the vm_area_struct data, and the mem_map page data is displayed. Example:

```
crash> vtop ffff88005b352680
VIRTUAL           PHYSICAL
ffff88005b352680  5b352680
...
```

- waitq – Displays tasks blocked on the specified wait queue

Utility `crash` Commands

The utility `crash` commands are helper commands. Available commands are:

- `ascii` – Translates a hexadecimal value to ASCII
- `btop` – Translates a hexadecimal value to its page number
- `eval` – Evaluates an expression
- `list` – Displays the contents of a linked list of structures
- `ptob` – Translates a page number to its physical address
- `ptov` – Translates a hexadecimal physical address into a kernel virtual address
- `search` – Searches memory space for a specified value
- `rd` – Reads or displays a specified amount of memory
- `wt` – Writes or modifies the contents of memory

The following utility `crash` commands are helper commands that serve a variety of functions:

- `ascii` – Translates a hexadecimal value to ASCII. If no argument is entered, an ASCII chart is displayed.
- `btop` – Translates a hexadecimal address to its page number
- `eval` – Evaluates an expression and displays the result in hexadecimal, decimal, octal, and binary. This command can also function as a calculator.
- `list` – Displays the contents of a linked list of structures
- `ptob` – Translates a page number to its physical address (byte value)
- `ptov` – Translates a hexadecimal physical address into a kernel virtual address
- `search` – Searches for a specified value within the user virtual, kernel virtual, or physical memory space. You can also provide a starting and ending point to search.
- `rd` – Reads or displays a specified amount of user, kernel, or physical memory in a specified format
- `wt` – Writes or modifies the contents of memory. Use this command with great care.

Session Control `crash` Commands

The session control `crash` commands help control the execution of a `crash` session. Available commands are:

- `alias` – Creates an alias for a command string
- `exit` – Exits the crash session
- `extend` – Extends the `crash` command set by dynamically loading crash extension shared object libraries
- `foreach` – Allows you to execute a `crash` command on multiple tasks in the system
- `gdb` – Passes arguments to the GNU Debugger
- `repeat` – Repeats a command indefinitely
- `set` – Changes the `crash` context to a new task, or displays the current context
- `q` – Exits the crash session

The following session control `crash` commands control the `crash` session itself:

- `alias` – Creates an alias for a command string. Several built-in aliases are provided. Enter the `alias` command with no arguments to display the current list of aliases.
- `exit` – Exits the `crash` session. This command is the same as the `q` command.
- `extend` – Extends the `crash` command set by dynamically loading crash extension shared object libraries. Use the `-u` option to unload shared object libraries.
- `foreach` – Allows you to execute a `crash` command on multiple tasks in the system. It can be used with `bt`, `vm`, `task`, `files`, `net`, `set`, `sig`, and `vtop` commands.
- `gdb` – Passes arguments to the GNU Debugger for processing. Use the `gdb help` command to list classes of commands.
- `repeat` – Repeats a command indefinitely until stopped with Ctrl + C. This command is useful only on a live system.
- `set` – Changes the `crash` context to a new task, or is used to display the current context
- `q` – Exits the `crash` session. This command is the same as the `exit` command.

General Guidelines for Using `crash`

- The `bt` command is often the first command you use after starting a crash session.
- To show the stack traces of an active task on each CPU:

```
crash> bt -a
```

- To see source file line numbers:

```
crash> bt -l
```

- To check for D state stuck processes:

```
crash> ps | grep UN
```

- To change the context to a different PID:

```
crash> set <PID>
```

- Other useful commands include `files`, `mount`, and `net`.
- Use `help <command>` to see options and examples.

The steps to debug a kernel crash vary according to the problem. The following provides some basic steps to follow. The `bt` command is often the first command you use after starting a `crash` session. The `bt` command shows the function trace leading up to the kernel panic. Use the `bt -a` command to show the stack traces of an active task on each CPU because there is often a relationship between the panicking tasks on one CPU and the running tasks on the other CPUs. If you see `cpu_idle` and `swapper`, it means nothing is running on that CPU. You can also use `bt` as an argument to the `foreach` command to display backtraces of all tasks. Use the `bt -l` command to see source file line numbers.

The `kmem -i` command provides a good summary of memory and swap use. Look for SLAB greater than 500 MB and SWAP in use. Use the command `ps | grep UN` to check for D state stuck processes. These processes contribute to the load average.

You can also redirect output to a file as follows. Look through the output for a process that is hung to set the PID. Use the `set` command to change the context to that PID:

```
crash> ps > ps.txt
crash> set <PID>
```

Commands that show open files (`files`), mount points (`mount`), and network devices with IP addresses (`net`) are often helpful. Use the `help <command>` in `crash` to see options.

Quiz

Which of the following statements are true?

a. To use `kdump`, first install the `kdump-tools` package.

b. To enable `kdump`, append the `crashkernel=<`*`mem_size`*`>` parameter to the kernel line in `grub.conf`.

c. The configuration file for `kdump` is `/etc/kdump.conf`.

d. You cannot write a vmcore dump to a remote system.

Quiz

Which of the following statements are true?

a. To use the `crash` utility, you must install the `crash` RPM and the `kernel-debuginfo` RPM packages.

b. The `kernel-debuginfo` RPM packages for Oracle Linux 6 are available from https://oss.oracle.com/.

c. The `crash` utility can analyze core dumps created by the `kdump`, `netdump`, `diskdump`, `xm dump-core`, Linux Kernel Crash Dumps (LKCD), and `vissh dump` facilities.

d. The `crash` utility has been merged with `gdb` (the GNU Debugger), so `crash` includes source code–level debugging capabilities.

Quiz

Which of the following statements are true?

a. The `crash` utility provides commands to display kernel data structures.

b. The `crash` utility provides commands to display various kernel subsystems on a system-wide or per-task basis.

c. The `crash` utility provides useful helper commands such as a calculator, translator, and search function.

d. The most useful `crash` command is `bt` (backtrace) because it shows the function trace leading up to the kernel panic.

Summary

In this lesson, you should have learned how to:

- Configure kdump to capture kernel vmcore dump
- Set kernel parameters that can cause a panic
- Use magic SysRq keys
- Use the `crash` utility for analyzing core dumps

Practice 17: Overview

This practice covers the following topics:

- Configuring `kdump`
- Creating a core dump file
- Preparing your system to analyze the vmcore
- Using the `crash` utility to analyze the vmcore

18

Dynamic Tracing with DTrace

Objectives

After completing this lesson, you should be able to:

- Describe the purpose of DTrace
- Enable DTrace on Oracle Linux
- Describe and view DTrace providers and probes
- Use the D programming language to enable probes and corresponding actions
- Use built-in D variables
- Use built-in D functions
- Create D scripts to explore your system

DTrace: Introduction

- DTrace is a performance analysis and troubleshooting tool.
- It has been available in the Oracle Solaris operating system since 2005, and was ported to Oracle Linux.
- DTrace allows dynamic tracing by defining probe points dynamically.
- Probes and actions at probe points are defined by commands and scripts written in the D language.
  – Probes are made available through providers.
- The current list of DTrace Oracle Linux providers is as follows:
  – `dtrace`, `profile`, `syscall`, `proc`, `sched`, `io`, `sdt`, and `fasttrap`

DTrace is a comprehensive dynamic tracing facility originally developed for Oracle Solaris, and is now available with Oracle Linux. DTrace is a performance analysis and troubleshooting tool designed to give operational insights that allow you to tune and troubleshoot the OS and user-space programs. Use DTrace to explore your system and to understand how it works, to track down performance problems across many layers of software, or to locate the cause of a multitude of abnormal behavior problems. DTrace also provides Oracle Linux developers with a tool to analyze performance, and to better see how their systems work. DTrace enables higher quality application development, reduced down time, lower cost, and greater utilization of existing resources.

DTrace stands for Dynamic Tracing and provides an instrumentation technique that dynamically patches live running instructions with instrumentation code. At specific points of execution in the code, you can activate probes and designate actions, such as collecting and displaying information. DTrace allows you to dynamically define probe points. This means that they are not precompiled into the kernel. DTrace has providers, which are basically categories of probes. The providers of DTrace for Linux are listed in the slide.

DTrace provides a D programming language to enable probe points and associated actions. Use the D language from the command line, or create D script files when performing complex tracing. The D language is similar to the C programming language and `awk`.

Reasons to Use DTrace on Linux

- Many tracing and profiling tools exist for Linux, for varying use cases:
  - Inconsistent syntax, scripting languages, and output formats
  - Lack of integration of both kernel and application tracing in a single tool
- DTrace provides an integrated solution.
- Administrators and developers know DTrace from Oracle Solaris.
- Existing D scripts can be used.
- Customers ask for it.

DTrace provides an integrated solution to the variety of tracing tools currently available for Linux. Available tracing and profiling tools include:

- `strace` – This tool traces system calls and signals made by a program.
- `pstack` – This tool prints a stack trace of a running process.
- `oprofile` – This is a system profiling tool that is capable of profiling all running code, including hardware and software interrupt handlers, kernel modules, the kernel, shared libraries, and applications. It uses hardware counters and can also count cache activity.
- `perf` – This tool is part of the Performance Counters for Linux (PCL) kernel-based subsystem. It tracks hardware events and can also measure software events.
- `stap` – This tool is the front end to Systemtap, which can collect information about a running Linux system for use in the diagnosis of performance or functional problems.
- `valgrind` – This is a suite of tools for debugging and profiling Linux executables. The tool suite includes memory leak checking, thread error detectors, and other profiler tools.
- `blktrace` – This tool is a block layer I/O tracing mechanism, which traces I/O traffic on block devices.

Each of these tools uses a different syntax, a different scripting language, and provides different output formats. There is a lack of integration of both kernel and application tracing. DTrace provides the integrated tracing solution in a single tool that customers have asked for.

Another reason for porting DTrace to Linux is that administrators and developers know DTrace from Oracle Solaris. These skills are transferable to DTrace for Oracle Linux. Existing Oracle Solaris D scripts can be used. A DTrace book titled *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X, and FreeBSD (*by Brendan Gregg and Jim Mauro, Prentice Hall 2011) contains hundreds of D scripts. There is also the DTraceToolkit, which is a collection of over 200 useful documented scripts. Some of the many tools developed using DTrace and the D programming language include the following:

- `dexplorer` – Automatically runs a collection of DTrace scripts to examine many areas of the system, and places the output in a meaningful directory structure that is tar'd and gzip'd
- `iosnoop` – Displays disk I/O activity in real time, allowing you to observe what is happening on your disks, including the PID and the responsible command. The output includes the block address and size of the disk operation.
- `iotop` – Displays top disk I/O events by process. This tracks disk I/O by process and prints a summary report that is refreshed every interval.
- `execsnoop` – Displays process activity. As processes are executed on the server, their details are printed out. This is especially useful in troubleshooting short-lived processes that are otherwise difficult to spot.
- `opensnoop` – Displays file opens. The file name and file handle are traced along with some process details.
- `rwtop` – Displays top read/write bytes by process. It prints a summary report that is refreshed at intervals. This measures reads and writes at the application level.
- `tcpsnoop` – Displays TCP network packets by process. This analyses TCP network packets and prints the responsible PID and UID, plus standard details such as IP address and port. This captures traffic of newly created TCP connections that were established while this program was running. It helps to identify processes causing TCP traffic.

The DTraceToolkit, the code for the previously mentioned D scripts, and many other D scripts are available at the following website: http://www.brendangregg.com/dtrace.html. Not all of the scripts in the DTrace book and in the DTrace Toolkit work in Linux. This is because DTrace for Linux continues to evolve and not all of the providers used in these scripts are available in DTrace for Linux. Another reason is where function or system calls are used (named), they probably differ between Solaris and Linux and the user must use the corresponding Linux kernel function name. However, many of the scripts do work in Linux and all the scripts are helpful in learning the capabilities of DTrace and the D programming language.

DTrace 0.4 in UEK R3

- In UEK R3, DTrace support is integrated with the kernel.
- You also need to install the `dtrace-utils` and `dtrace-modules` packages that are available on UNL.
- Meta-provider support has been implemented:
  - Allows DTrace to instantiate providers dynamically on demand.
  - An example of a meta-provider is the `fasttrap` provider that is used for user-space tracing.
- User-space statically defined tracing (USDT) supports SDT-like probes in user-space executable and libraries.
- The SDT provider has been improved so that probes for kernel modules can now be enabled.
  - Previously you could enable probes only in the core kernel.

As of this writing, the latest release is DTrace 0.4 in UEK R3. The following lists some of the additional features compared with DTrace 0.3.2 in UEK R2. Refer to the Unbreakable Enterprise Kernel Release 3 Release Notes at http://docs.oracle.com/cd/E37670_01/ for a complete list of additional features.

- In addition to using the UEK R3, to use DTrace 0.4 you must install the `dtrace-utils` and `dtrace-modules` packages that are available on the `ol6_x86_64_UEKR3_latest` and `ol6_x86_64_Dtrace_userspace_latest` channels on the Unbreakable Linux Network (ULN).
- Meta-provider support has been implemented, which allows DTrace to instantiate providers dynamically on demand. An example of a meta-provider is the `fasttrap` provider that is used for user-space tracing.
- User-space statically defined tracing (USDT) supports Statically Defined Tracing (SDT)-like probes in user-space executable and libraries.
- USDT requires programs to be modified to include embedded static probe points. The `sys/sdt.h` header file is provided to support USDT, but you can also use the `-h` option to `dtrace` to generate a suitable header file from a provider description file.
- The SDT provider has been improved so that probes for kernel modules can now be enabled. Previously you could enable probes only in the core kernel.

The list of the additional features in DTrace 0.4 in UEK R3 compared with DTrace 0.3.2 in UEK R2 continues:

- To enable the use of USDT probes in DTrace-enabled programs, you must load the new `fasttrap` module:

  ```
  # modprobe fasttrap
  ```
- Currently, the `fasttrap` provider supports the use of USDT probes. It is not used to implement the `pid` provider.
- For more information, refer to the “Statically Defined Tracing for User Applications” chapter of the *Oracle Linux 6 Dynamic Tracing Guide*, at http://docs.oracle.com/cd/E37670_01/E38608/html/index.html.

DTrace-Enabled Applications

- A number of DTrace-enabled applications are available with the release of DTrace 0.4, including MySQL and PHP.
  - These applications have been instrumented to contain statically defined DTrace probes.
- DTrace-enabled versions of user-space applications are available in the playground repository of Oracle Public Yum.
  - Oracle does not offer support for these packages and does not accept any liability for their use.
- PostgreSQL 9.2.4 includes support for DTrace.

A number of DTrace-enabled applications are available with the release of DTrace 0.4, including MySQL and PHP. These applications have been instrumented to contain statically defined DTrace probes. You can find details about the probes for MySQL at: http://dev.mysql.com/doc/refman/5.5/en/dba-dtrace-mysqld-ref.html.

Details about the probes for PHP are available at http://php.net/manual/features.dtrace.php.

DTrace-enabled versions of user-space applications are available in the playground repository of Oracle Public Yum:

http://public-yum.oracle.com/repo/OracleLinux/OL6/playground/latest/x86_64/

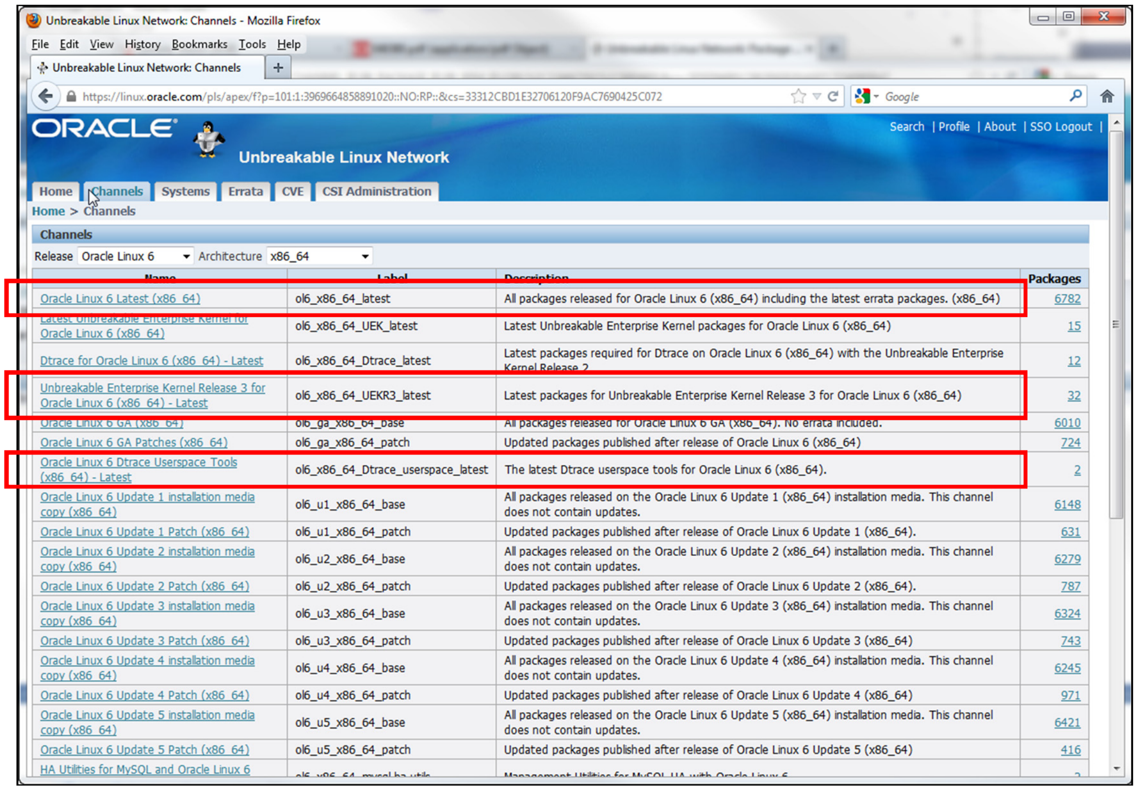
The packages that are provided in the playground repository are intended for experimentation only and you should not use them with production systems. Oracle does not offer support for these packages and does not accept any liability for their use.

PHP 5.4.20, PHP 5.5.4, and later versions can be built with DTrace support on Oracle Linux. See https://blogs.oracle.com/opal/entry/using_php_dtrace_on_oracle.

PostgreSQL 9.2.4 includes support for DTrace as described in http://www.postgresql.org/docs/9.2/static/dynamic-trace.html.

You can build a DTrace-enabled version of PostgreSQL by specifying the `--enable-dtrace` option to configure as described in http://www.postgresql.org/docs/9.2/static/install-procedure.html.

ULN Channels for DTrace 0.4



Unbreakable Linux Network: Channels - Mozilla Firefox

https://linux.oracle.com/pls/apex/f?p=101:1:3969664858891020::NO:RP::&cs=33312CBD1E32706120F9AC7690425C072

Unbreakable Linux Network

Home > Channels

Channels

Release: Oracle Linux 6 · Architecture: x86_64

Name	Label	Description	Packages
Oracle Linux 6 Latest (x86_64)	ol6_x86_64_latest	All packages released for Oracle Linux 6 (x86_64) including the latest errata packages. (x86_64)	6782
Latest Unbreakable Enterprise Kernel for Oracle Linux 6 (x86_64)	ol6_x86_64_UEK_latest	Latest Unbreakable Enterprise Kernel packages for Oracle Linux 6 (x86_64)	15
Dtrace for Oracle Linux 6 (x86_64) - Latest	ol6_x86_64_Dtrace_latest	Latest packages required for Dtrace on Oracle Linux 6 (x86_64) with the Unbreakable Enterprise Kernel Release 2	12
Unbreakable Enterprise Kernel Release 3 for Oracle Linux 6 (x86_64) - Latest	ol6_x86_64_UEKR3_latest	Latest packages for Unbreakable Enterprise Kernel Release 3 for Oracle Linux 6 (x86_64)	32
Oracle Linux 6 GA (x86_64)	ol6_ga_x86_64_base	All packages released for Oracle Linux 6 GA (x86_64). No errata included.	6010
Oracle Linux 6 GA Patches (x86_64)	ol6_ga_x86_64_patch	Updated packages published after release of Oracle Linux 6 (x86_64)	724
Oracle Linux 6 Dtrace Userspace Tools (x86_64) - Latest	ol6_x86_64_Dtrace_userspace_latest	The latest Dtrace userspace tools for Oracle Linux 6 (x86_64).	2
Oracle Linux 6 Update 1 installation media copy (x86_64)	ol6_u1_x86_64_base	All packages released on the Oracle Linux 6 Update 1 (x86_64) installation media. This channel does not contain updates.	6148
Oracle Linux 6 Update 1 Patch (x86_64)	ol6_u1_x86_64_patch	Updated packages published after release of Oracle Linux 6 Update 1 (x86_64).	631
Oracle Linux 6 Update 2 installation media copy (x86_64)	ol6_u2_x86_64_base	All packages released on the Oracle Linux 6 Update 2 (x86_64) installation media. This channel does not contain updates.	6279
Oracle Linux 6 Update 2 Patch (x86_64)	ol6_u2_x86_64_patch	Updated packages published after release of Oracle Linux 6 Update 2 (x86_64).	787
Oracle Linux 6 Update 3 installation media copy (x86_64)	ol6_u3_x86_64_base	All packages released on the Oracle Linux 6 Update 3 (x86_64) installation media. This channel does not contain updates.	6324
Oracle Linux 6 Update 3 Patch (x86_64)	ol6_u3_x86_64_patch	Updated packages published after release of Oracle Linux 6 Update 3 (x86_64)	743
Oracle Linux 6 Update 4 installation media copy (x86_64)	ol6_u4_x86_64_base	All packages released on the Oracle Linux 6 Update 4 (x86_64) installation media. This channel does not contain updates.	6245
Oracle Linux 6 Update 4 Patch (x86_64)	ol6_u4_x86_64_patch	Updated packages published after release of Oracle Linux 6 Update 4 (x86_64)	971
Oracle Linux 6 Update 5 installation media copy (x86_64)	ol6_u5_x86_64_base	All packages released on the Oracle Linux 6 Update 5 (x86_64) installation media. This channel does not contain updates.	6421
Oracle Linux 6 Update 5 Patch (x86_64)	ol6_u5_x86_64_patch	Updated packages published after release of Oracle Linux 6 Update 5 (x86_64)	416
HA Utilities for MySQL and Oracle Linux 6	ol6_x86_64_mysql_ha_utils	Management Utilities for MySQL HA with Oracle Linux 6	2

To install and configure DTrace 0.4, subscribe your system to the following ULN channels:

- Oracle Linux 6 Latest (x86_64) (`ol6_x86_64_latest`)
- Unbreakable Enterprise Kernel Release 3 for Oracle Linux 6 (x86_64) – Latest (`ol6_x86_64_UEKR3_latest`)
- Oracle Linux 6 Dtrace Userspace Tools (x86_64) – Latest (`ol6_x86_64_Dtrace_userspace_latest`)

Make sure that your system is not subscribed to the following channels. The following channels are applicable to UEK R2, DTrace for UEK R2, the beta release of DTrace for UEK R2, and the beta release of UEK R3.

- Latest Unbreakable Enterprise Kernel for Oracle Linux 6 (x86_64) (`ol6_x86_64_UEK_latest`)
- DTrace for Oracle Linux 6 (x86_64) - Latest (`ol6_x86_64_Dtrace_latest`)
- DTrace for Oracle Linux 6 (x86_64) - Beta release (`ol6_x86_64_Dtrace_BETA`)
- Unbreakable Enterprise Kernel Release 3 (3.8 based) for Oracle Linux 6 (x86_64) - Beta release (`ol6_x86_64_UEK_BETA`)

Enabling DTrace

- Register your systems with the appropriate ULN channels.
- To update your system to UEK R3:

```
# yum update
```

- To install the DTrace utilities package:

```
# yum install dtrace-utils
```

- The DTrace kernel modules are installed in
  – `/lib/modules/`uname -r`/kernel/drivers/dtrace`
  – Use the `modprobe` command to load the kernel modules.
- Release notes are located in `/usr/share/doc/dtrace*`.
- DTrace guides and UEK R3 release notes are available at http://docs.oracle.com/cd/E37670_01/.

After registering your systems with the appropriate channels on ULN, use the `yum` command to install the packages.

If your system is not already running the Unbreakable Enterprise Kernel Release 3 (UEK R3), use the `yum` command as follows to update your system to use UEK R3:

```
    # yum update
```

After the `yum update` command completes, reboot your system. Your system should automatically boot the UEK R3 (version `3.8.13`). Otherwise, select the Oracle Linux Server (`3.8.13`) kernel in the GRUB menu.

Use the `yum` command as follows to install the DTrace utilities package:

```
    # yum install dtrace-utils
```

If the appropriate `dtrace-modules` package for the running kernel is not present on your system, running any `dtrace` command downloads and installs the package from ULN, for example:

```
    # dtrace -l
```

Alternatively, you can use the `yum` command to install the `dtrace-modules-`uname -r`` package.

The DTrace modules are installed in `/lib/modules/<UEK version>.x86_64 /kernel/drivers/dtrace`.

```
# ls -l /lib/modules/`uname -r`/kernel/drivers/dtrace
-rw-r--r--.  ...  dt_perf.ko
-rw-r--r--.  ...  dtrace.ko
-rw-r--r--.  ...  dt_test.ko
-rw-r--r--.  ...  fasttrap.ko
-rw-r--r--.  ...  profile.ko
-rw-r--r--.  ...  sdt.ko
-rw-r--r--.  ...  systrace.ko
```

The `dt_test` module is included because it is used by the developer's test suite. It is not for general use. There is some consideration to not even distribute it. The main reason it is currently distributed is that there might be a reason in the future for making the test suite available as a package. In that case, the `dt_test` module would be needed.

The `dt_perf` module is for internal testing only and has been provided as a "curiosity" item. It has limited usefulness for anyone other than developers of DTrace.

Use the `modprobe` command to load the modules. It is not necessary to manually load the `dtrace` module. Loading any of the other modules automatically loads the `dtrace` module.

```
# modprobe fasttrap
# modprobe profile
# modprobe sdt
# modprobe systrace
# lsmod | grep dtrace
dtrace     136502  4 systrace,sdt,profile,fasttrap
ctf           941  1 dtrace
```

Additional resources include the following guides located at: http://docs.oracle.com/cd/E37670_01/:

- *Oracle Linux 6 Dynamic Tracing Guide*
- *Oracle Linux 6 Administrator's Solutions Guide*
- *Oracle Linux 6 DTrace Tutorial*

There is also a forum for DTrace on Oracle Linux at: https://forums.oracle.com/forums/forum.jspa?forumID=1398.

DTrace Probes

- Instrumentation points are called *probes*.
- Each probe is associated with an action.
- Actions determine what data to record when a probe fires.
  - Any argument to the function
  - Any global variable in the kernel
  - A nanosecond time stamp of when the function was called
  - A stack trace to indicate what code called this function
- A probe:
  - Is made available by a provider
  - Identifies the instrumented function (for function-bound probes) and, in most cases, the containing module
  - Has a name
  - Has a unique integer identifier

DTrace probes are tracing points, or instrumentation points, that enable you to record data at various points of interest. Each probe is associated with an action. When the probe fires, the action is executed. For example, you can define probes that fire upon entry into a kernel function, when a specific file opens, when a particular process starts, or when a line of code executes. Examples of data that you can display upon entry into a kernel function include:

- Any argument to the function
- Any global variable in the kernel
- A nanosecond time stamp of when the function was called
- A stack trace to indicate what code called this function

A probe:

- Is made available by a provider
- (For function-bound probes) identifies the instrumented function and, in most cases, the containing module
- Has a name
- Has a unique integer identifier

DTrace Providers

- A probe is defined and implemented by a provider.
- A provider is a kernel module that enables its probes to trace data.
- The following providers are available:
  - `dtrace`, `fasttrap`, `io`, `proc`, `profile`, `sched`, `sdt`, and `syscall`
- To list all probes for all providers:

```
# dtrace -l
ID  PROVIDER    MODULE   FUNCTION NAME
1    dtrace                        BEGIN
...
```

- Each line of output identifies a specific probe in the following form:

  provider:*module*:*function*:*name*

DTrace probes are implemented by providers. A provider is a kernel module that enables a requested probe to fire when it is hit. A provider receives information from DTrace about when a probe is to be enabled and transfers control to DTrace when an enabled probe is hit.

Use the `dtrace -l` command to list all probes and their respective providers:

```
# dtrace -l
ID    PROVIDER    MODULE    FUNCTION NAME
1     dtrace                         BEGIN
2     Dtrace                         END
3     Dtrace                         ERROR
4     syscall    vmlinux        read entry
5     syscall    vmlinux        read return
...
```

Each line of output identifies a specific probe. Use the following syntax to uniquely identify each probe:

provider:*module*:*function*:*name*

The DTrace providers are summarized as follows:

- `dtrace` – Provides probes that relate to DTrace itself, such as `BEGIN`, `ERROR`, and `END`. You can use these probes to initialize DTrace's state before tracing begins, process its state after tracing has completed, and handle unexpected execution errors in other probes.
- `profile` – Provides probes associated with an interrupt that fires at a fixed, specified time interval. These probes are associated with the asynchronous interrupt event rather than with any particular point of execution. You can use these probes to sample some aspect of a system's state.
- `syscall` – Provides probes at the entry to and return from every system call. Because system calls are the primary interface between user-level applications and the operating system kernel, these probes can offer you an insight into the interaction between applications and the system.
- `sdt` – Creates probes at sites that a software programmer has formally designated. The Statically Defined Tracing (SDT) mechanism allows programmers to choose locations of interest to users of DTrace and to convey information about each location through the probe name.
- `proc` – Provides probes for monitoring process creation and termination, LWP (light-weight process) creation and termination, executing new program images, and sending and handling signals.
- `sched` – Provides probes related to CPU scheduling. Because CPUs are the one resource that all threads must consume, the sched provider is very useful for understanding systemic behavior.
- `io` – Provides probes that relate to data input and output. The `io` provider enables quick exploration of behavior observed through I/O monitoring tools such as `iostat`.
- `fasttrap` – Supports user-space tracing of DTrace-enabled applications. The `fasttrap` provider makes available a single probe that fires each time that a DTrace-enabled user process executes an instruction.

`dtrace` Provider

- The `dtrace` provider enables pre-processing, post-processing, and D program error-processing capabilities.
- To list all probes for the `dtrace` provider:

```
# dtrace -l -P dtrace
ID  PROVIDER    MODULE   FUNCTION NAME
1    dtrace                       BEGIN
2    dtrace                       END
3    dtrace                       ERROR
```

- Three probes are provided by the `dtrace` provider:
  - `dtrace:::BEGIN` – Fires each time you start a new tracing request
  - `dtrace:::END` – Fires after all other probes
  - `dtrace:::ERROR` – Fires when a runtime error occurs in executing a clause for a probe

Use `dtrace` probes to initialize state (pre-processing) before tracing begins, process state after tracing has completed (post-processing), and handle unexpected execution errors in other probes.

Only three probes are provided by the `dtrace` provider:

- `BEGIN` – Fires each time you start a new tracing request. Use this probe to initialize any state that is needed in other probes.
- `END` – Fires after all other probes. Use this probe to process state that has been gathered or to format the output.
- `ERROR` – Fires when a runtime error occurs in executing a clause for a probe

The syntax to uniquely identify each probe is as follows:

```
provider:module:function:name
```

The `dtrace` probes are not bound to a function or module; therefore, these probes are identified as follows:

- `dtrace:::BEGIN`
- `dtrace:::END`
- `dtrace:::ERROR`

`profile` Provider

- The `profile` provider provides dynamic probes that fire at specific time intervals.
- You can use `profile` probes to sample an aspect of system state and provide periodic output or take periodic action.
- To list all probes for the `profile` provider:

```
# dtrace -l -P profile
 ID PROVIDER   MODULE                   FUNCTION NAME
635  profile                                     tick-1
635  profile                                     tick-10
636  profile                                     tick-100
637  profile                                     tick-500
638  profile                                     tick-1000
640  profile                                     tick-5000
```

The `profile` provider provides probes associated with a time-based interrupt that fires at a fixed, specified time interval. These probes are not associated with any particular point of execution. Use these probes to sample some aspect of system state such as the state of the current thread, the state of the CPU, or the current machine instruction.

The `tick-N` probes fire at fixed intervals at a high interrupt level on only one CPU per interval. The actual CPU might change over time.

The value of `N` defaults to rate-per-second. You can include an optional time suffix as follows:

- `nsec|ns` – Nanoseconds; for example, `tick-1ns`
- `usec|us` – Microseconds; for example, `tick-10us`
- `msec|ms` – Milliseconds; for example, `tick-100ms`
- `sec|s` – Seconds (this is the default)
- `min|m` – Minutes; for example, `tick-1m`
- `hour|h` – Hours; for example, `tick-1h`
- `day|d` – Days; for example, `tick-1d`
- `hz` – Hertz; the highest supported tick frequency is 5000 HZ (`tick-5000hz`).

`syscall` Provider

- The `syscall` provider dynamically instruments the entry and return of every system call.
- A majority of the probes are provided by `syscall`.
- To list all probes for the `syscall` provider:

```
# dtrace -l -P syscall
ID   PROVIDER     MODULE    FUNCTION NAME
4     syscall    vmlinux        read entry
5     syscall    vmlinux        read return
6     syscall    vmlinux       write entry
7     syscall    vmlinux       write return
8     syscall    vmlinux        open entry
9     syscall    vmlinux        open return
10    syscall    vmlinux       close entry
11    syscall    vmlinux       close return
...
```

The `syscall` provider makes available a probe at the entry to and return from every system call in the system. A system call changes the processor state from user mode to kernel mode, so that the CPU can access protected kernel memory. Because system calls are the primary interface between user-level applications and the operating system kernel, the `syscall` provider offers tremendous insight into application behavior with respect to the system.

A majority of the probes are provided by the `syscall` provider. The `syscall` provider provides a pair of probes for each system call:

- `entry` – Fires before the system call is entered
- `return` – Fires after the system call has completed but before control has transferred back to user level

For all `syscall` probes, the function name is set to the name of the instrumented system call and the module name is `vmlinux`. As an example, the name for `syscall` probe ID `4` is:

```
syscall:vmlinux:read:entry
```

`sdt` Provider

- The `sdt` (Statically Defined Tracing) provider allows programmers to insert explicit probes in their applications.
- SDT probes are declared using the following macros from `<sys/sdt.h>`:
  - `DTRACE_PROBE`, `DTRACE_PROBE1`, `DTRACE_PROBE2`, `DTRACE_PROBE3`, and `DTRACE_PROBE4`.
- The module name and function name corresponds to the kernel module and function of the probe.
- The name of the probe depends on the name given in the `DTRACE_PROBEn` macro.
- To list the SDT probes you have installed:

```
# dtrace -l -P sdt -m <module>
```

The `sdt` (Statically Defined Tracing) provider allows programmers to insert explicit probes in their applications. Programmers can select locations of interest to users of DTrace and provide some information about each location through the probe name. Because the SDT probes that are defined for Oracle Linux kernel are likely to change over time, they are not listed here.

SDT probes are declared using the following macros from `<sys/sdt.h>`: `DTRACE_PROBE`, `DTRACE_PROBE1`, `DTRACE_PROBE2`, `DTRACE_PROBE3`, and `DTRACE_PROBE4`. The module name and function name of an SDT-based probe correspond to the kernel module and function of the probe. The name of the probe depends on the name given in the `DTRACE_PROBEn` macro.

DTrace includes the kernel module name and function name as part of the tuple identifying a probe; therefore, you do not need to include this information in the probe name. Use the following command on your driver module to list the probes you have installed and the full names that are seen by DTrace users:

```
    # dtrace -l -P sdt -m <module>
```

`proc` Provider

- The `proc` provider makes available all probes pertaining to the following activities:
  - Process creation and termination
  - LWP creation and termination
  - Executing new program images
  - Sending and handling signals
- To list all probes for the `proc` provider:

```
# dtrace -l -P proc
 ID PROVIDER   MODULE             FUNCTION NAME
608     proc  vmlinux        __send_signal signal-discard
609     proc  vmlinux        __send_signal signal-send
613     proc  vmlinux    do_execve_common exec-success
614     proc  vmlinux    do_execve_common exec
615     proc  vmlinux    do_execve_common exec-failure
...
```

The `proc` provider makes available probes pertaining to the following activities: process creation and termination, lightweight process (LWP) creation and termination, executing new program images, and sending and handling signals.

There are currently 13 `proc` probes, including the following:

- `start` – Fires in the context of a newly created process, before any user-level instructions are executed in the process
- `create` – Fires when a process (or process thread) is created using `fork()` or `vfork()`
- `exec` – Fires whenever a process loads a new process image using a variant of the `execve()` system call
- `exec-failure` – Fires when an `exec()` variant has failed
- `exec-success` – Fires when an `exec()` variant has succeeded
- `exit` – Fires when the current process is exiting
- `lwp-create` – Fires when a process thread is created
- `lwp-start` – Fires within the context of a newly created process or process thread
- `lwp-exit` – Fires when a process or process thread is exiting

sched Provider

- The sched provider makes available all probes related to CPU scheduling.
- Probes provided by sched allow you to trace key scheduling events.
- To list all probes for the sched provider:

```
# dtrace -l -P sched
 ID   PROVIDER    MODULE                 FUNCTION NAME
604       sched   vmlinux              __schedule remain-cpu
605       sched   vmlinux              __schedule sleep
606       sched   vmlinux              __schedule preempt
607       sched   vmlinux              __schedule off-cpu
612       sched   vmlinux            dequeue_task dequeue
623       sched   vmlinux            enqueue_task enqueue
624       sched   vmlinux      finish_task_switch on-cpu
...
```

The sched provider makes available probes related to CPU scheduling. The sched provider dynamically traces key scheduling events. Because CPUs are the one resource that all threads must consume, the sched provider is very useful for understanding systemic behavior. For example, using the sched provider, you can understand when and why threads sleep, run, change priority, or wake other threads.

There are currently 12 sched probes, including the following:

- on-cpu – Fires when a CPU begins to execute a thread
- off-cpu – Fires when a thread is about to be taken off a CPU
- surrender – Fires when a CPU has been instructed by another CPU to make a scheduling decision, often because a higher-priority thread has become runnable
- change-pri – Fires whenever a thread's priority is about to be changed
- enqueue – Fires immediately before a runnable thread is enqueued to a run queue
- dequeue – Fires immediately before a runnable thread is dequeued from a run queue
- tick – Fires as a part of clock tick–based accounting
- wakeup – Fires immediately before the current thread wakes a thread sleeping on a synchronization object

`io` Provider

- The `io` provider makes available all probes related to device input and output (I/O).
- To list all probes for the `io` provider:

```
# dtrace -l -P io
 ID  PROVIDER      MODULE                   FUNCTION NAME
610        io     vmlinux           __wait_on_buffer wait-done
611        io     vmlinux           __wait_on_buffer wait-start
622        io     vmlinux     end_bio_bh_io_sync done
630        io     vmlinux               submit_bh start
```

The `io` provider makes available probes related to device input and output (I/O). You can explore behavior observed through I/O monitoring tools such as `iostat`.

The `io` probes are defined as follows:

- `start` – Fires when an I/O request is about to be made either to a peripheral device or to an NFS server
- `done` – Fires after an I/O request has been fulfilled
- `wait-start` – Fires immediately before a thread begins to wait, pending completion of a given I/O request
- `wait-done` – Fires when a thread finishes waiting for the completion of a given I/O request

Enabling Probes

- Use the `dtrace` command without the `-l` option.
- You can specify probes to enable by provider (`-P`), by name (`-n`), by function (`-f`), and by module (`-m`).
- To enable all probes provided by the `syscall` provider:

```
# dtrace -P syscall
```

- To enable the `BEGIN` probe in the `dtrace` provider:

```
# dtrace -n dtrace:::BEGIN
```

- To enable probes in the `syscall` provider whose function name begins with "`open`":

```
# dtrace -n syscall::open*:
```

- To enable every probe in the `vmlinux` module:

```
# dtrace -m vmlinux
```

Probes are enabled with the `dtrace` command by specifying them without the list (`-l`) option. DTrace performs the associated action when the probe fires. The default action indicates only that a probe fired. No other data is recorded. You can enable (and list) probes by provider (`-P`), by name (`-n`), by function (`-f`), and by module (`-m`).

To enable all probes provided by the `syscall` provider:

```
# dtrace -P syscall
dtrace: description ‘syscall’ matched 592 probes
CPU    ID           FUNCTION:NAME
0      36             ioctl:entry
0      37            ioctl:return
0      36             ioctl:entry
0      37            ioctl:return
0      30    rt_sigaction:entry
^C
```

From the output, you can see that the default action displays the CPU where the probe fired, the unique probe ID, the function where the probe fired, and the probe name.

The following example enables the `BEGIN` probe in the `dtrace` provider:

```
# dtrace -n dtrace:::BEGIN
dtrace: description 'dtrace:::BEGIN' matched 1 probe
CPU        ID         FUNCTION:NAME
0           1                :BEGIN
^C
```

The following example enables probes provided by the `syscall` provider in the `open` function:

```
# dtrace -n syscall::open:
dtrace: description 'syscall::open:' matched 2 probes
CPU        ID         FUNCTION:NAME
0           8            open:entry
0           9           open:return
0           8            open:entry
0           9           open:return
0           8            open:entry
^C
```

You can use the special characters `*`, `?`, and `[ ]` as wildcards in probe names. The preceding example matched two probes. The following example uses the `*` character to list all probes in the `syscall` provider whose function name begins with "`open`":

```
# dtrace –l -n syscall::open*:
 ID PROVIDER    MODULE              FUNCTION NAME
  8 syscall                             open entry
  9 syscall                             open return
492 syscall                           openat entry
493 syscall                           openat return
586 syscall                 open_by_handle_at entry
586 syscall                 open_by_handle_at return
```

The following example enables all probes in the `vmlinux` module, regardless of the provider to which they belong:

```
# dtrace -m vmlinux
dtrace: description 'vmlinux' matched 35 probes
CPU        ID                  FUNCTION:NAME
0         628              __schedule:sleep
0         620          dequeue_task:dequeue
0         626            __schedule:off-cpu
0         609     finish_task_switch:on-cpu
^C
```

DTrace Actions

- DTrace actions are taken when a probe fires.
- The default action indicates only that a probe fired.
  - No other data is recorded.
- Actions are programmable in the D language.
  - Enclose actions in braces `{}`.
  - Enclose probe and associated action in single quotes `‘ ’`.
- The following example uses the `trace()` function as the action:

```
# dtrace -n ‘syscall:::entry {trace(timestamp)}’
dtrace: description ‘syscall:::entry’ matched 296 probes
CPU   ID       FUNCTION:NAME
0      6          write:entry 173635407378228
^C
```

DTrace actions are taken when a probe fires. The default action displays the CPU where the probe fired, the unique probe ID, the function where the probe fired, and the probe name.

Use the D programming language to specify probes of interest and bind actions to those probes. The D programming language is similar to the C programming language and includes a set of functions and variables to help make tracing easier. The actions are listed as a series of statements enclosed in braces `{}`, following the probe name.

The following example uses the `trace()` function as the action to trace the time of entry to each system call:

```
# dtrace -n ‘syscall:::entry {trace(timestamp)}’
dtrace: description ‘syscall:::entry’ matched 296 probes
CPU        ID         FUNCTION:NAME
0           6           write:entry 173635407378228
0           6           write:entry 173635407399180
0          36           ioctl:entry 173635407416780
^C
```

Surround the probe name and associated action with single quotation marks (`‘ ’`).

In the following example, the probe of interest is the `BEGIN` probe provided by the `dtrace` provider. The actions are to call two functions, `trace()` and `exit()`. When specifying multiple actions, terminate each action with a semicolon (`;`).

```
# dtrace -n ‘dtrace:::BEGIN {trace(“hello, world”);exit(0);}’
dtrace: description ‘dtrace:::BEGIN’ matched 1 probe
CPU        ID         FUNCTION:NAME
0           1                :BEGIN  hello, world
```

The function `trace()` indicates that DTrace records the specified argument, the string ”`hello, world`”, when the `dtrace:::BEGIN` probe fires, and then prints it out.

The function `exit()` indicates that DTrace ceases tracing and exits the `dtrace` command.

Built-In D Variables

- `errno` – Current errno value for system calls
- `execname` – Name of the current process' executable file
- `pid` – Process ID of the current process
- `tid` – Thread ID of the current thread
- `timestamp` – Time since boot in nanoseconds
- `probeprov` – Provider field of the current probe
- `probemod` – Module field of the current probe
- `probefunc` – Function field of the current probe
- `probename` – Name field of the current probe
- `curpsinfo` – Process state information for the current thread
- `curthread` – Internal kernel thread structure for the current thread

The list in the slide gives some of the most useful built-in D variables. Refer to the *Oracle Linux 6 Dynamic Tracing Guide* for a complete list of built-in D variables.

`trace()` Built-in Function

- `trace(pid)` – Traces the current process ID
- `trace(execname)` – Traces the name of the current executable
- `trace(curthread->t_pri)` – Traces the `t_pri` field of the current thread
- `trace(probefunc)` – Traces the function name of the probe
- `trace(timestamp / 1000)` – Traces the time stamp variable divided by 1000
- `trace("hello, world")` – Traces the string "`hello, world`"
- `trace(`max_pfn)` – Traces the `max_pfn` system variable

There are many built-in functions that perform actions. Previous examples use the `trace()` function. This function takes a D expression as its argument and traces the result to the directed buffer. All the examples in the slide are valid `trace()` actions.

The last example, ``trace(`max_pfn)``, is an example of tracing the value of a system variable named `max_pfn`. DTrace instrumentation executes inside the Oracle Linux operating system kernel; therefore, in addition to accessing special DTrace variables and probe arguments, you can also access kernel data structures, symbols, and types. These capabilities enable advanced DTrace users, administrators, service personnel, and driver developers to examine the low-level behavior of the operating system kernel and device drivers.

D uses the backquote character (`` ` ``) as a special scoping operator for accessing symbols that are defined in the operating system and not in your D program. For example, the Oracle Linux kernel contains a C declaration of a system variable named `max_pfn`. This variable is declared in C in the kernel source code as follows:

```
unsigned long max_pfn
```

Use the following statement in a D program script to trace the value of this variable:

```
trace(`max_pfn)
```

DTrace: Examples

- Display commands executing on the system:

```
# dtrace -n 'proc:::exec {trace(stringof(arg0));}'
```

- Display new processes with arguments:

```
# dtrace -n 'proc:::exec-success {trace(curpsinfo->
   pr_psargs);}'
```

- Display files opened by processes using `printf()`:

```
# dtrace -n 'syscall::open*:entry {printf("%s %s",
   execname,copyinstr(arg0));}'
```

- Display the number of system calls using aggregations:

```
# dtrace -n 'syscall:::entry {@num[probefunc] =
   count();}'
```

  – With aggregation, totals appear after you press Ctrl + C.

Several examples are presented in subsequent slides. The following example displays commands that are executing on your system:

```
# dtrace -n 'proc:::exec {trace(stringof(arg0));}'
dtrace: description 'proc:::exec ' matched 1 probe
CPU  ID                  FUNCTION:NAME
0   617          do_execve_common:exec   /bin/ls
0   617          do_execve_common:exec   /bin/bash
^C
```

The following example displays new processes with arguments:

```
# dtrace -n 'proc:::exec-success {trace(curpsinfo->pr_psargs);}'
dtrace: description 'proc:::exec-success ' matched 1 probe
CPU  ID                  FUNCTION:NAME
0   619  do_execve_common:exec-success /usr/sbin/sshd -R
0   619  do_execve_common:exec-success /sbin/unix_chkpwd root...
^C
```

The following example displays files opened by process. The example also introduces the `printf()` function, which is used to trace data, as well as to output the data and other text in a specific format that you describe. The `printf()` function tells DTrace to trace the data associated with each argument after the first argument (`“%s %s”`), known as the *format string*, and then to format the results using the rules described by the format string:

```
# dtrace -n ‘syscall::open*:entry {printf(“%s
%s”,execname,copyinstr(arg0));}’
dtrace: description ‘syscall::open*:entry ‘ matched 3 probes
CPU   ID                    FUNCTION:NAME
0      8                       open:entry  vi /etc/ld.so.cache
0      8                       open:entry  vi /lib64/libm.so.6
0      8                       open:entry  crond /etc/passwd
0      8                       open:entry  automount /proc/mounts
0      8                       open:entry  man /etc/ld.so.cache
0      8                       open:entry  man /lib64/libc.so.6
^C
```

The following example displays the number of system calls by system call using aggregations. DTrace provides several built-in functions for aggregating data that individual probes gather. The name of the aggregation is prefixed with the `@` symbol. Press Ctrl + C to display the totals:

```
# dtrace -n ‘syscall:::entry {@num[probefunc] = count();}’
dtrace: description ‘syscall:::entry ‘ matched 296 probes
^C
getgid                                          1
getuid                                          1
inotify_add_watch                               1
newlstat                                        1
rt_sigreturn                                    1
setgid                                          1
setresgid                                       1
setuid                                          1
close                                           2
fcntl                                           2
getdents                                        2
geteuid                                         2
newfstat                                        2
open                                            2
setgroups                                       2
...
```

DTrace: Examples

- Display the number of `write()` system calls and the number of `read()` system calls invoked by processes:

```
# dtrace -n 'syscall::write:entry,syscall::read:entry
   {@[strjoin(probefunc,"() calls")] = count();}'
```

- Display disk size by process:

```
# dtrace -n 'io:::start {printf("%d %s %d",
   pid,execname,args[0]->b_bcount);}'
```

- Display the number of `read()` system calls, excluding those initiated by the `dtrace` process:

```
# dtrace -n 'syscall::read:entry /execname != "dtrace"/
   {@reads[execname, fds[arg0].fi_pathname] = count();}'
```

  - This example uses a predicate.

The following example counts the number of `write()` system calls and the number of `read()` system calls invoked by processes until you press Ctrl + C:

```
# dtrace -n 'syscall::write:entry,syscall::read:entry
{@[strjoin(probefunc,"() calls")] = count();}'
dtrace: description 'syscall::write:entry,syscall::read:entry '
matched 2 probes
^C
write() calls                                238
read() calls                                 431
```

The following example displays disk size by process:

```
# dtrace -n 'io:::start {printf("%d %s %d",pid,execname,args[0]-
>b_bcount);}'
CPU   ID                FUNCTION:NAME
  0  602              submit_bh:start 342 jdb2/dm-0-8 4096
  0  602              submit_bh:start 1184 flush-252:0 4096
^C
```

The following example uses a predicate to selectively aggregate the number of `read()` system calls. The `read()` system calls initiated by the `dtrace` process are excluded because of the `/execname != "dtrace"/` predicate:

```
# dtrace -n ‘syscall::read:entry /execname != "dtrace"/ {
@reads[execname, fds[arg0].fi_pathname] = count(); }’
^C
at-spi-registry   socket:[12950]                          1
bash              pipe:[15831]                            1
crond             /lib64/libnsl-2.12.so                   1
crond             /lib64/security/pam_access.so           1
fdisk             /sys/devices/virtual/block/dm-0/dm/na   1
...
```

The D programming language does not provide control-flow constructs such as if-statements and loops. However, D does provide the ability to conditionally trace data and modify control flow. D uses logical expressions called *predicates* that can be used to prefix program clauses.

A predicate expression is enclosed in `//` characters and is evaluated at probe firing time, before executing any of the statements associated with the corresponding clause. If the predicate evaluates to true (represented by any non-zero value), the statement list is executed. If the predicate is false (represented by a zero value), none of the statements are executed and the probe firing is ignored.

D Scripts

- Example of a simple D script:

```
# cat hello.d
dtrace:::BEGIN
{
   trace(“hello, world”);
   exit(0);
}
```

- Use the `dtrace -s <script>` command to execute:

```
# dtrace -s hello.d
dtrace: script ‘hello.d’ matched 1 probe
CPU       ID                 FUNCTION:NAME
  0        1                        :BEGIN  hello, world
```

Enabling multiple probes and multiple actions becomes difficult to manage on the command line. For complex tracing, DTrace supports D script files. Each D script file ends with a `.d` suffix and consists of a series of clauses. Each clause describes one or more probes to enable and an optional set of actions to perform when the probe fires. The actions are listed as a series of statements enclosed in braces `{}`, following the probe name. Each statement ends with a semicolon (`;`). The example D script in the slide provides the same functionality as the following command-line entry:

```
# dtrace -n ‘dtrace:::BEGIN {trace(“hello, world”);exit(0);}’
dtrace: description ‘dtrace:::BEGIN’ matched 1 probe
CPU        ID          FUNCTION:NAME
0           1                 :BEGIN  hello, world
```

Use the `dtrace` command with the `-s <script>` option to execute the script:

```
# dtrace -s hello.d
dtrace: script ‘hello.d’ matched 1 probe
CPU       ID                FUNCTION:NAME
  0        1                       :BEGIN  hello, world
```

D Scripts

- Use the `dtrace -q -s <script>` command to execute the script in “quiet” mode:

```
# dtrace -qs hello.d
hello, world
```

- Alternatively, you can create executable DTrace interpreter files by beginning the script with the following line:

```
#!/usr/sbin/dtrace -s
dtrace:::BEGIN
{
   trace(“hello, world”);
   exit(0);
}
```

- Give the script execute permission and you can run the script by entering its name on the command line.

The quiet mode option, `dtrace -q`, instructs DTrace to record only the actions that are explicitly stated. This option suppresses the default output that is normally produced by the `dtrace` command. The following example shows the use of the `-q` option in the `hello.d` script:

```
# dtrace -q -s hello.d
hello, world
```

Alternatively, you can create executable DTrace interpreter files. Interpreter files have execute permission and always begin with the following line:

```
#!/usr/sbin/dtrace -s
```

Add the preceding line as the first line in a script to invoke the interpreter. Give the script execute permission and you can run the script by entering its name on the command line as follows:

```
# vi hello.d
#!/usr/sbin/dtrace -qs
...
# chmod +x hello.d
# ./hello.d
hello, world
```

Using Predicates in a D Script

Example of predicates in a D script:

```
# cat countdown.d
dtrace:::BEGIN
{
   i = 5;
}
profile:::tick-1sec
/i > 0/
{
   trace(i--);
}
profile:::tick-1sec
/i == 0/
{
   trace("blastoff!");
   exit(0);
}
```

The example in the slide implements a predicate in a D script. When executed, the script counts down from 5 and then prints a message and exits. The script uses the `dtrace:::BEGIN` probe to initialize an integer `i` to `10`. The script then uses the `profile:::tick-1sec` probe to implement a timer that fires once per second. The first predicate, `/i > 0/`, checks whether the value of `i` is greater than `0`. If the value is greater, the action is to use the `trace()` function to decrement the value of `i` by 1.

The script uses the `profile:::tick-1sec` probe a second time and the predicate evaluates if `i` is equal to zero, `/i == 0/`. If this predicate is true, the action is to use the `trace()` function to display the string `"blastoff!"`. A second action is to exit `dtrace` and return to the shell prompt.

The following command executes the D script in the slide and displays the output:

```
# dtrace -s countdown.d
dtrace: script 'countdown.d' matched 3 probes
CPU     ID                    FUNCTION:NAME
0      638                       :tick-1sec  5
0      638                       :tick-1sec  4
0      638                       :tick-1sec  3
0      638                       :tick-1sec  2
0      638                       :tick-1sec  1
0      638                       :tick-1sec  blastoff!
```

The following example uses a predicate to evaluate the process ID (`pid`) that is passed as a command-line argument (`$1`), and uses DTrace to observe every time the process performs a `read()` or `write()` system call:

```
# cat rw.d
syscall::read:entry,
syscall::write:entry
/pid == $1/
{
}
```

The following example expects the `pid` of the shell to be `9618`. Executing the D script in one window and running shell commands in a second window reports `dtrace` probe firings as follows:

```
# dtrace -s rw.d 9618
dtrace: script ‘rw.d' matched 2 probes
CPU     ID                    FUNCTION:NAME
0        6                     write:entry
0        4                      read:entry
0        6                     write:entry
0        4                      read:entry
^C
```

Conditional Expressions

D provides support for simple conditional expressions using the `?` and `:` operators. These operators enable a triplet of expressions to be associated where the first expression is used to conditionally evaluate one of the other two. For example, the following D statement could be used to set a variable `x` to one of two strings depending on the value of `i`:

```
x = i == 0 ? "zero" : "non-zero";
```

In this example, the expression `i == 0` is first evaluated. If the expression is true, “`zero`” is returned as the value of `x`. If the expression is false, “`non-zero`” is returned. These return values do not invoke a tracking function such as `trace()` or `printf()`. If you want to conditionally trace data, use a predicate instead.

D Script: Example

The following script includes the use of an array:

```
# cat rwtime.d
syscall::read:entry,
syscall::write:entry
/pid == $1/
{
   ts[probefunc] = timestamp;
}
syscall::read:return,
syscall::write:return
/pid == $1 && ts[probefunc] != 0/
{
   printf("%d nsecs", timestamp - ts[probefunc]);
}
```

The example script in the slide uses an array to trace the elapsed time for each system call. The script instruments both the entry to and return from `read()` and `write()`, and samples the time at each point. Then, on return from a given system call, the script computes the difference between the first and second time stamp. You could use separate variables for each system call, but it is easier to use an associative array indexed by the probe function name.

The first clause defines an array named `ts` and assigns the appropriate member the value of the DTrace variable `timestamp`. This variable returns the value of an always-incrementing nanosecond counter. When the entry time stamp is saved, the corresponding return probe samples `timestamp` again and reports the difference between the current time and the saved value.

The predicate on the return probe requires that DTrace is tracing the appropriate process and that the corresponding entry probe has already fired and assigned `ts[probefunc]` a non-zero value. This trick eliminates invalid output when DTrace first starts. If your shell is already waiting in a `read()` system call for input when you execute `dtrace`, the `read:return` probe fires without a preceding `read:entry` for this first `read()` and `ts[probefunc]` evaluates to zero because it has not yet been assigned.

The following example runs the script and displays an output. The command-line argument uses command substitution to obtain the PID of the `bash` shell:

```
# dtrace -s rwtime.d '/usr/bin/pgrep - bash'
dtrace: script 'rwtime.d' matched 4 probe
CPU ID           FUNCTION:NAME
  0  7            write:return 15645 nsecs
  0  7            write:return 11175 nsecs
  0  7            write:return 10616 nsecs
  0  5             read:return 3347303916 nsecs
  0  7            write:return 13968 nsecs
  0  7            write:return 11175 nsecs
  0  7            write:return 10616 nsecs
^C
```

D Script: Example

- The `-C` option invokes the C preprocessor:

```
# dtrace -C -s activity.d
```

- Use D *pragmas* to enable DTrace options:

```
# pragma D option quiet
```

- Clause-local variables reuse storage for each D program clause that relates to a probe.
  - This is similar to automatic variables in C, C++, or Java.
  - Use `this->` notation to assign and reference.

```
    this->pid = ((struct task_struct *)arg0)->pid;
    time[this->pid] = timestamp;
    p_pid[this->pid] = pid;
    p_name[this->pid] = execname;
    p_exec[this->pid] = "";
```

The D script on the following page shows ongoing activity in terms of what program was executing, what its parent is, and how long it ran. Use the following command to run this script:

```
# dtrace -C -s activity.d
```

The `-C` option invokes the C preprocessor (see `man cpp`). You can use the C preprocessor in conjunction with your D programs by specifying the `dtrace -C` option.

This script introduces special D compiler directives called *pragmas*. DTrace is tuned by setting or enabling options and this is accomplished by using D pragmas. You can include D pragmas anywhere in a D script, including outside probe clauses. Begin the line with `#` as follows. This example sets the DTrace runtime `quiet` option (same as `dtrace -q`):

```
#pragma D option quiet
```

This script also introduces clause-local variables. These are variables whose storage is reused for each D program clause that relates to a probe. Clause-local variables are similar to automatic variables in a C, C++, or Java language program, which are active during each invocation of a function. Like all D program variables, clause-local variables are created on their first assignment. These variables are referenced and assigned by applying the `->` operator to the special identifier `this`.

```
# cat activity.d
#pragma D option quiet

proc::do_fork:create
{
        this->pid = ((struct task_struct *)arg0)->pid;
        time[this->pid] = timestamp;
        p_pid[this->pid] = pid;
        p_name[this->pid] = execname;
        p_exec[this->pid] = "";
}

proc::do_execve_common:exec
/p_pid[pid] != 0/
{
        p_exec[pid] = stringof(arg0);
}

proc::do_exit:exit
/p_pid[pid] != 0 && p_exec[pid] != ""/
{
        printf("%s (%d) executed %s (%d) for %d msecs\n",
           p_name[pid], p_pid[pid], p_exec[pid], pid,
               (timestamp - time[pid]) / 1000);
}

proc::do_exit:exit
/p_pid[pid] !=0 && p_exec[pid] == ""/
{
        printf("%s (%d) forked itself (as %d) for %d msecs\n",
           p_name[pid], p_pid[pid], pid,
               (timestamp - time[pid]) / 1000);
}
```

Quiz

Which of the following statements are true?

a. DTrace providers are categories of probes.

b. DTrace probes are tracing or instrumentation points that enable you to record data at various points of interest.

c. Oracle makes DTrace available through the ULN.

d. DTrace for Oracle Linux is being ported to Oracle Solaris.

Quiz

Which of the following statements are true?

a. DTrace probes are uniquely identified by using the `provider:module:function:name` format.

b. `dtrace` probes are used to initialize state before tracing begins, to process state after tracing has completed, and to handle unexpected execution errors by other probes.

c. The `syscall` probes are available for the entry to every system call and for the return from every system call.

d. Other providers include `sched`, `sdt`, `io`, `proc`, `profile`, and `fbt`.

Quiz

Which of the following statements are true?

a. DTrace probes can be enabled by provider (`-P`), by name (`-n`), by function (`-f`), by module (`-m`), and by label (`-l`).

b. Use the D programming language to specify probes of interest and to bind actions to those probes.

c. The following example uses the correct D programming language syntax:

```
# dtrace –n ‘syscall:::entry
   {trace(timestamp);exit(0);}’
```

d. The `trace()` built-in function takes a D expression as an argument and traces the result to the directed buffer.

Quiz

Which of the following statements are true?

a. DTrace provides several built-in functions for aggregating data that individual probes gather.

b. The name of the aggregation is prefixed with the `@` symbol.

c. DTrace predicates provide the ability to conditionally trace data and modify control flow.

d. A predicate expression is enclosed in `%%` characters.

Summary

In this lesson, you should have learned how to:

- Enable DTrace on Oracle Linux
- View DTrace providers and probes
- Use the D programming language to enable probes and corresponding actions
- Use built-in D variables
- Use built-in D functions
- Create D scripts to explore your system

Practice 18: Overview

This practice covers the following topics:

- Installing DTrace packages
- Enabling DTrace modules
- Using DTrace from the command line to list providers and probes
- Using DTrace from the command line to enable probes and actions
- Creating and running D scripts

A

Cgroup Subsystem Parameters

`cpuset` Subsystem Parameters

```
# ls /cgroup/cpuset/cpuset*
cpuset.cpu_exclusive
cpuset.cpus
cpuset.mem_exclusive
cpuset.mem_hardwall
cpuset.memory_migrate
cpuset.memory_pressure
cpuset.memory_pressure_enabled
cpuset.memory_spread_page
cpuset.memory_spread_slab
cpuset.mems
cpuset.sched_load_balance
cpuset.sched_relax_domain_level
```

`cpuset.cpus`

Is a comma-separated list of CPU cores to which a cgroup has access

`cpuset.mems`

Is a comma-separated list of memory nodes to which a cgroup has access

`cpuset.cpu_exclusive`

Specifies whether the CPUs listed in `cpuset.cpus` are exclusively allocated to this CPU set. The value must be either of the following:

- `0` – CPUs are not exclusively allocated. This is the default.
- `1` – This enables exclusive use of the CPUs by a CPU set.

`cpuset.mem_exclusive`

Specifies whether the memory nodes listed in `cpuset.mems` are exclusively allocated to this CPU set. The value must be either of the following:

- `0` – Memory nodes are not exclusively allocated. This is the default.
- `1` – This enables exclusive use of the memory nodes by a CPU set.

`cpuset.mem_hardwall`

Specifies whether kernel allocations of pages and buffers to the memory nodes listed in `cpuset.mems` are exclusive to this CPU set. The value must be either of the following:

- `0` – Page and buffer data is shared. This is the default.
- `1` – Page and buffer data is not shared across processes that belong to multiple users.

`cpuset.memory_migrate`

Specifies whether memory pages are allowed to migrate between memory nodes if the value of `cpuset.mems` changes. The value must be either of the following:

- `0` – Disables memory migration. This is the default.
- `1` – Allows pages to migrate between memory nodes

`cpuset.memory_pressure_enabled`

Specifies whether the memory pressure statistic is gathered. The value must be either of the following:

- `0` – Disables the counter. This is the default.
- `1` – Enables the counter

`cpuset.memory_pressure`

Reports the memory pressure (if enabled) created by the processes in the cgroup. This represents the number of attempts per second by processes to reclaim in-use memory, multiplied by 1000.

`cpuset.memory_spread_page`

Specifies whether file system buffers are distributed across the allocated memory nodes of the CPU set. The value must be either of the following:

- `0` – Disables distribution. This is the default.
- `1` – Allows distribution of file system buffers across memory nodes

`cpuset.memory_spread_slab`

Specifies whether I/O slab caches are distributed across the allocated memory nodes of the CPU set. The value must be either of the following:

- `0` – Disables distribution. This is the default.
- `1`: Allows the caches to be distributed across memory nodes

`cpuset.sched_load_balance`

Specifies whether the kernel balances CPU load by moving processes between the CPU cores allocated to a CPU set. The value must be either of the following:

- `1` – Enables load balancing. This is the default.
- `0` – Disables load balancing

`cpuset.sched_relax_domain_level`

Specifies the load-balancing scheme, if load balancing is enabled. The value must be either of the following:

- `-1` – Uses the system's default load balancing scheme. This is the default.
- `0` – Performs periodic load balancing
- `1` – Load balances across threads running on the same core
- `2` – Load balances across cores of the same CPU
- `4` – Load balances across a subset of CPU cores on a system with a NUMA architecture
- `5` – Load balances across all CPU cores on a system with a NUMA architecture

`cpu` Subsystem Parameters

```
# ls /cgroup/cpu/cpu*
cpu.cfs_period_us
cpu.cfs_quota_us
cpu.rt_period_us
cpu.rt_runtime_us
cpu.shares
cpu.stat
```

`cpu.cfs_period_us`

Specifies a period of time in microseconds that a cgroup's CPU access is rescheduled. For example, to give tasks in a cgroup access to a single CPU for 0.5 seconds out of every 1 second, set `cpu.cfs_quota_us` to 500000 and set `cpu.cfs_period_us` to 1000000.

`cpu.cfs_quota_us`

Specifies the total amount of time in microseconds that all tasks in a cgroup can run during one period (as defined by `cpu.cfs_period_us`). The default setting of `-1` indicates the cgroup has no CPU time restrictions.

`cpu.rt_period_us`

Applicable to real-time scheduling tasks only. Specifies a period of time in microseconds that a cgroup's CPU access is rescheduled. The default is 1000000 (1 second).

`cpu.rt_runtime_us`

Applicable to real-time scheduling tasks only. Specifies the longest period of time in microseconds that tasks in a cgroup have CPU access. The default is 950000 (0.95 seconds).

By default, real-time scheduling tasks in a cgroup access CPU resources 95% of each second. To give tasks less access, for example 60%, set `cpu.rt_runtime_us` to 600000 and leave `cpu.rt_period_us` set to 1000000.

`cpu.shares`

Specifies the relative share of CPU time available to the tasks in a cgroup. Two cgroups with the same value receive equal CPU time. One cgroup with a value of `2` receives twice the CPU time of another cgroup with a value of `1`.

`cpu.stat`

Reports the following CPU time statistics:

- `nr_periods` – Number of period intervals (as specified in `cpu.cfs_period_us`) that have elapsed.
- `nr_throttled` – Number of times tasks in a cgroup have been throttled (not allowed to run) because they have exhausted their quota (as specified in `cpu.cfs_quota_us`).
- `throttled_time` – Total time duration (in nanoseconds) for which tasks in a cgroup have been throttled.

`cpuacct` Subsystem Parameters

```
# ls /cgroup/cpuacct/cpuacct*
cpuacct.stat
cpuacct.usage
cpuacct.usage_percpu
```

`cpuacct.usage`

Reports the total CPU time in nanoseconds for all the tasks in the cgroup. Setting this parameter to `0` resets its value and the value of the **`cpuacct.usage_percpu`** parameter.

`cpuacct.stat`

Reports the total CPU time in nanoseconds that is spent in user and system (kernel) mode by all the tasks in the cgroup

`cpuacct.usage_percpu`

Reports the CPU time in nanoseconds that is spent for each CPU by all the tasks in the cgroup

`memory` Subsystem Parameters

```
# ls /cgroup/memory/memory*
memory.failcnt
memory.force_empty
memory.limit_in_bytes
memory.max_usage_in_bytes
memory.memsw.failcnt
memory.memsw.limit_in_bytes
memory.memsw.max_usage_in_bytes
memory.memsw.usage_in_bytes
memory.move_charge_at_immigrate
memory.numa_stat
memory.oom_control
memory.soft_limit_in_bytes
memory.stat
memory.swappiness
memory.usage_in_bytes
memory.use_hierarchy
```

`memory.limit_in_bytes`

Specifies the hard, upper limit permitted for user memory, including the file cache, in bytes. You cannot limit the root cgroup. You can only limit groups lower in the hierarchy.

`memory.soft_limit_in_bytes`

Specifies a soft, upper limit for user memory, including the file cache. Set the soft limit lower than the hard limit value because the hard limit always takes precedence.

`memory.memsw.limit_in_bytes`

Specifies the upper limit for user memory, plus the swap space in bytes. To avoid an out-of-memory error, set this value less than the amount of swap space, and set the value of the `memory.limit_in_bytes` parameter less than `memory.memsw.limit_in_bytes`. You must also set the `memory.limit_in_bytes` parameter before setting the `memory.memsw.limit_in_bytes` parameter.

Each of these limit parameters defaults to bytes. You can also specify the limits in k or K, m or M, and g or G. Set the parameters to `-1` to remove the limits.

`memory.failcnt`

Reports the number of times that the amount of memory used by a cgroup has reached the value of `memory.limit_in_bytes`

`memory.memsw.failcnt`

Reports the number of times that the amount of memory and swap space used by a cgroup has reached the value of `memory.memsw.limit_in_bytes`

`memory.max_usage_in_bytes`

Reports the maximum amount of memory in bytes used by the tasks in the cgroup

`memory.usage_in_bytes`

Reports the total size in bytes of the memory used by all the tasks in the cgroup

`memory.memsw.max_usage_in_bytes`

Reports the maximum amount of memory and swap space in bytes used by the tasks in the cgroup

`memory.memsw.usage_in_bytes`

Reports the total size in bytes of the memory and swap space used by the tasks in the cgroup

`memory.stat`

Reports the following memory statistics:

- `active_anon` – The size in bytes of the anonymous and swap cache on the active least-recently-used (LRU) list (includes `tmpfs`)
- `active_file` – The size in bytes of the file-backed memory on the active LRU list
- `cache` – The size in bytes of the page cache (includes `tmpfs`)
- `hierarchical_memory_limit` – The size in bytes of the memory limit for the cgroup hierarchy
- `hierarchical_memsw_limit` – The size in bytes of the memory limit plus swap for the cgroup hierarchy
- `inactive_anon` – The size in bytes of the anonymous and swap cache on the inactive LRU list (includes `tmpfs`)
- `inactive_file` – The size in bytes of the file-backed memory on the inactive LRU list
- `mapped_file` – The size in bytes of the memory-mapped files (includes `tmpfs`)
- `pgfault` – The number of page faults, where the kernel has to allocate and initialize physical memory for use in the virtual address space of a process
- `pgmajfault` – The number of major page faults, where the kernel has to actively free physical memory before allocation and initialization
- `pgpgin` – The number of paged-in pages of memory
- `pgpgout` – The number of paged-out pages of memory
- `rss` – The size in bytes of the anonymous and swap cache (does not include tmpfs). The actual resident set size is given by the sum of `rss` and `mapped_file`.
- `swap` – The size in bytes of used swap space
- `total_*` – The value of the appended statistic for the cgroup and all of its children
- `unevictable` – The size in bytes of memory that is not reclaimable

`memory.force_empty`

Is set to `0` to remove all pages from memory that were used by the tasks in a cgroup. Do this before removing a child cgroup; otherwise the memory pages are assigned to the parent cgroup.

`memory.swappiness`

Specifies a bias value for the kernel to swap out the memory pages used by the processes in the cgroup rather than reclaim pages from the page cache. Modify the default value of `60` to change the bias as follows:

- Set to less than `60` to reduce the kernel's preference for swapping out.
- Set to greater than `60` to increase the preference for swapping out.
- Set to greater than `100` to allow the system to swap out the pages that fall within the address space of the cgroup's tasks.

You cannot change the bias for the root cgroup or for a cgroup containing child cgroups.

`memory.use_hierarchy`

Specifies whether the kernel attempts to reclaim memory from other tasks in a cgroup's hierarchy that exceed their limit. The value must be either of the following:

- `0` – Disables reclaiming of memory from other tasks in the hierarchy. This is the default.
- `1` – Allows memory to be reclaimed from other tasks in the hierarchy

`memory.move_charge_at_immigrate`

Specifies whether a task's charges are moved when you migrate the task between cgroups. The value must be either of the following:

- `0` – Disables moving task charges
- `1` – Moves charges for an in-use or swapped-out anonymous page that is exclusively owned by the task
- `2` – Moves charges for the file pages that are memory-mapped by the task
- `3` – Is equivalent to specifying both `1` and `2`

`memory.oom_control`

Displays the values of the out-of-memory (OOM) notification control feature. The settings are:

- `oom_kill_disable` – Set to either `0` to enable or `1` to disable.
- `under_oom` – Set to either `1` to allow the tasks to be stopped or to `0` to remove the cgroup from under OOM control.

`memory.numa_stat`

Reports the NUMA memory usage in bytes for each memory node, as well as the following statistics:

- `anon` – The size in bytes of the anonymous and swap cache
- `file` – The size in bytes of file-backed memory
- `unevictable` – The size in bytes of non-reclaimable memory
- `total` – The sum of anon, file, and unevictable values

`devices` Subsystem Parameters

```
# ls /cgroup/devices/devices*
devices.allow
devices.deny
devices.list
```

`devices.allow`

Specifies a device that a cgroup is allowed to access. Devices are defined by the following:

- Type (`a` for any, `b` for block, or `c` for character)
- Major and minor numbers separated by a colon
- Access modes (`m` for create permission, `r` for read access, and `w` for write access)

You can use an asterisk, `*`, as a wildcard to represent any major or minor number. For example, specify `b 8:* rw` to allow read and write access to any SCSI disk drive.

`devices.deny`

Specifies a device that a cgroup is not allowed to access. Use the same syntax as `devices.allow`.

`devices.list`

Reports the devices with access controls. Output of `a *.* rwm` indicates that all devices are available in all access modes.

`freezer` Subsystem Parameters

```
# ls /cgroup/freezer/freezer*
freezer.state
```

`freezer.state`

The value of the parameter is one of the following:

- `FROZEN` – Tasks in the cgroup are suspended.
- `FREEZING` – Tasks in the cgroup are in the process of being suspended.
- `THAWED` – Tasks in the cgroup have resumed.

You perform the following steps to suspend a specific process:

1. Move the process to a cgroup in a hierarchy that has the freezer subsystem attached.
2. Freeze the particular cgroup to suspend the process contained in it.

You cannot move a process into a `FROZEN` cgroup. Only the `FROZEN` and `THAWED` values can be written to `freezer.state`. The `FREEZING` value can only be read, not written.

`net_cls` Subsystem Parameters

```
# ls /cgroup/net_cls/net_cls*
net_cls.classid
```

`net_cls.classid`

Specifies a single hexadecimal value that indicates a traffic control handle. The value is presented in decimal format to the Linux traffic controller (`tc`). You can configure the traffic controller to use the handles that the `net_cls` subsystem adds to network packets.

`blkio` Subsystem Parameters

```
# ls /cgroup/blkio/blkio*
blkio.io_merged
blkio.io_queued
blkio.io_service_bytes
blkio.io_serviced
blkio.io_service_time
blkio.io_wait_time
blkio.reset_stats
blkio.sectors
blkio.throttle.io_service_bytes
blkio.throttle.io_serviced
blkio.throttle.read_bps_device
blkio.throttle.read_iops_device
blkio.throttle.write_bps_device
blkio.throttle.write_iops_device
blkio.time
blkio.weight
blkio.weight_device
```

`blkio.weight`

Specifies a cgroup's share of access to block I/O. The range is from 100 to 1000, with a default value of `1000`.

`blkio.weight_device`

Specifies a cgroup's share of access to block I/O on a specific device. The range is from 100 to 1000 and the device is specified by its major and minor numbers, separated by a colon. For example, `8:16 100` specifies a value of 100 for `/dev/sdb`. The value of this parameter overrides the default value set by the `blkio.weight` parameter.

`blkio.time`

Reports the time in milliseconds that I/O access was available to a device specified by its major and minor numbers

`blkio.sectors`

Reports the number of disk sectors written to or read from the devices specified by their major and minor numbers

`blkio.io_serviced`

Reports the number of `async`, `read`, `sync`, or `write` I/O operations to or from the devices specified by their major and minor numbers, as recorded by the Completely Fair Queuing (CFQ) scheduler

`blkio.io_service_bytes`

Reports the number of bytes transferred by `async`, `read`, `sync`, or `write` I/O operations to or from the devices specified by their major and minor numbers, as recorded by the CFQ

`blkio.io_service_time`

Reports the time in nanoseconds taken to complete the `async`, `read`, `sync`, or `write` I/O operations to or from the devices specified by their major and minor numbers, as recorded by the CFQ

`blkio.io_wait_time`

Reports the total time in nanoseconds that a cgroup spent waiting for the `async`, `read`, `sync`, or `write` I/O operations to complete to or from the devices specified by their major and minor numbers

`blkio.io_merged`

Reports the number of BIOS requests that have been merged into the `async`, `read`, `sync`, or `write` I/O operations by a cgroup

`blkio.io_queued`

Reports the number of requests queued for `async`, `read`, `sync`, or `write` I/O operations by a cgroup

`blkio.throttle.read_iops_device`

Specifies the maximum number of read operations per second that a cgroup can perform on a device. The device is specified by its major and minor numbers, separated by a colon. For example, `8:16 100` specifies that a maximum of 100 read operations per second can be performed on `/dev/sdb`.

`blkio.throttle.read_bps_device`

Specifies the maximum number of bytes per second that a cgroup can read from a device. The device is specified by its major and minor numbers, separated by a colon. For example, `8:16 4194304` specifies that a maximum of 4 MB can be read per second from `/dev/sdb`.

`blkio.throttle.write_iops_device`

Specifies the maximum number of write operations per second that a cgroup can perform on a device. The device is specified by its major and minor numbers, separated by a colon. For example, `8:1 50` specifies that a maximum of 50 write operations per second can be performed on `/dev/sda1`.

`blkio.throttle.write_bps_device`

Specifies the maximum number of bytes per second that a cgroup can write to a device. The device is specified by its major and minor numbers, separated by a colon. For example, `8:1 2097152` specifies that a maximum of 2MB can be written per second to `/dev/sda1`.