XC™ Series System Administration Guide (CLE 6.0.UP04) S-2393

Contents

About the XC Series System Administration Guide....9
About the Cray Management System....12
Manage the System....13
Connect the SMW to the Console of a Service Node....13
Configure Remote Access to SMW with VNC....13
About the Integrated Dell Remote Access Controller (iDRAC)....14
Change the Default iDRAC Password....14
Dell R815 SMW: Change the BIOS and iDRAC Settings....14
Dell R630 SMW: Change the BIOS and iDRAC Settings....23
Use the iDRAC....33
Hardware Component Identification....34
Physical ID for Cray XC Series Systems....34
Node ID (NID) on Cray XC Series Systems....37
Extended Node ID (XNID)....38
Topology Class....38
Boot the System....38
Run Tests after Boot is Complete....39
Manually Boot the Boot Node and Service Nodes....40
Manually Boot the Compute Nodes....41
Reboot a Single Compute Node....42
Reboot Login or Network Nodes....42
Reboot Many Nodes....42
Boot the SMW in Rescue Mode....43
Debug Ansible Failures During System Boot....44
Examine System Logs....44
Look Up Configuration Details....45
Examine Ansible Changelogs....46
Debug Ansible Failures in `init`....48
Examine System Dumps....48
Log on to the Boot Node....49
Display Boot Configuration Information....49
Update the Boot Configuration....50
Display the Format of the SDB attributes Table....50
Update SDB Tables....51
Free Up Disk Space in the btrfs File System After Removing SMW Snapshots....52

Boot a Node or Set of Nodes Using the xtcli boot Command....52
Increase the Boot Manager Timeout Value....53
Reboot Controllers of a Cabinet or Blade....53
Bounce Blades Repeatedly Until All Blades Succeed....54
Request and Display System Routing....54
Initiate a Network Discovery Process....55
Configure IP Routes....55
Shut Down the System Using the auto.xtshutdown File....56
The xtshutdown Command....56
Shut Down the System or Part of the System Using the xtcli shutdown Command....57
Shut Down Service Nodes....57
Stop System Components....58
Restart a Blade or Cabinet....60
System Component States....60
Abort Active Sessions on the HSS Boot Manager....62
Display and Change Software System Status....62
Configure Current System Timezone....62
View and Change the Status of Nodes....65
Perform Parallel Operations on Compute Nodes....66
Perform Parallel Operations on Service Nodes....67
Mark a Compute Node as a Service Node....67
Find Node Information....67
Display and Change Hardware System Status....68
Recreate HSS Database File System After Corruption....69
Dynamic Fan Speed Control....73
    Enable Dynamic Fan Speed Control....73
    Configure and Validate Dynamic Cooling Control Variables....74
Disable Hardware Components....76
Enable Hardware Components....77
Check Current State of Compute Node SSDs....77
Set Hardware Components to `EMPTY`....78
Lock Hardware Components....78
Unlock Hardware Components....79
Over-provision an Intel P3608 SSD....79
xtbounce Error Message Indicates Cabinet Controller and Its Blade Controllers Not in Sync....82
Power-cycle a Component to Handle Bus Errors....82
When a Component Fails....83
Dump and Reboot Nodes Automatically....83

Collect Debug Information From Hung Nodes Using the xtnmi Command....83
Modify BIOS Parameters....84
Increase File System Size....84
Add New Hardware to a System....86
Add a New Disk to a Volume Group in a Storage Set....89
Reboot Controllers of a Cabinet or Blade....90
Bounce Blades Repeatedly Until All Blades Succeed....91
Shut Down the System Using the auto.xtshutdown File....91
The xtshutdown Command....91
Shut Down Service Nodes....92
Shut Down the System or Part of the System Using the xtcli shutdown Command....92
Stop System Components....93
Restart a Blade or Cabinet....95
Abort Active Sessions on the HSS Boot Manager....96
Display and Change Software System Status....96
View and Change the Status of Nodes....96
Mark a Compute Node as a Service Node....97
Find Node Information....98
Display and Change Hardware System Status....99
Generate HSS Physical IDs....99
Disable Hardware Components....99
Enable Hardware Components....100
Set Hardware Components to `EMPTY`....100
Lock Hardware Components....101
Unlock Hardware Components....101
Set the Turbo Boost Limit....102
Perform Parallel Operations on Service Nodes....103
Perform Parallel Operations on Compute Nodes....103
xtbounce Error Message Indicates Cabinet Controller and Its Blade Controllers Not in Sync....104
Reduce Impact of Btrfs Periodic Maintenance on SMW Performance....104
Power-cycle a Component to Handle Bus Errors....105
When a Component Fails....105
Capture and Analyze System-level and Node-level Dumps....105
Configure xtdumpsys for Systems Using passwordless ssh....106
cdump and crash Utilities for Node Memory Dump and Analysis....107
Dump and Reboot Nodes Automatically....107
The /etc/opt/cray-xt-dumpd/dumpd.conf Configuration File....108
The dumpd-dbadmin Tool....109

The dumpd-request Tool....109
Collect Debug Information From Hung Nodes Using the xtnmi Command....110
Modify BIOS Parameters....110
Set or Change the HSS Data Store (MariaDB) Root Password....111
Recover from a Corrupt or Missing SMW MariaDB Database....112
Restore the HSS MariaDB Database from a Backup....113
Regenerate the HSS MariaDB Database from Scratch....114
Troubleshoot Temperature Warnings Reported in an End Cabinet....115
Recover from SMW R630 Boot Disk Hardware RAIDS Failure....116
Recover from SMW R815 Boot Disk Software RAID1 Failure....117
About X.509 Certificates and How to Redistribute Them....119
Update X.509 Host Certificate After SMW Hostname Change....125
Manage System Access....127
Change Account Passwords on the SMW....127
Change Account Passwords on CLE Nodes....127
Configure the System....129
Cray XC System Configuration....129
About the Configurator....130
Create a Config Set....132
Update a Config Set....136
Validate a Config Set and List Validation Rules....140
Config Set Create/Update Process....142
Tips for Configurator Interactive Sessions....146
cfgset Troubleshooting Tips....151
Remove Shallow Checksum after Pushing a Config Set from One SMW to Another....152
Update cray_sysenv Worksheet....153
Prepare and Update the Global Config Set....153
About Simple Sync....161
Configure Files for Cray Simple Sync Service....165
About the Node Image Mapping Service (NIMS)....166
About Node Groups....166
About the Image Management and Provisioning System (IMPS)....169
Where to Place the Root File System—tmpfs versus netroot....172
Install Third-Party Software with a Custom Image Recipe....173
About Config Set Caching....180
Add Kernel Watch Descriptors to Improve Config Set Caching Performance....181
Change a File on a Compute Node....181
Use an Ansible Play to Change a File on a Compute Node....183

Use a Custom Image Recipe to Change a File on a Compute Node....184
About Custom Ansible Plays....188
Control a Service on Specific Nodes at Boot Time....190
Manage Node Configuration, Services, and Settings at Boot Time (boot.last Script)....191
About Secure Shell Configuration....193
Monitor the System....196
Manage Log Files Using CLE and HSS Commands....196
Check the Status of System Components....197
Check the Status of Compute Processors....198
Monitor the System with the System Environmental Data Collector (SEDC)....199
Monitor the Health of PCIe Channels....199
Examine Activity on the HSS Boot Manager....200
Poll a Response from an HSS Daemon, Manager, or the Event Router....200
Validate the Health of the HSS....200
Monitor Event Router Daemon (erd) Events....201
Monitor Node Console Messages....201
View Component Alert, Warning, and Location History....202
Display Component Information....202
Display Alerts and Warnings....203
Display System Network Congestion Protection Information....204
Clear Component Flags....204
Display Error Codes....205
Cray Lightweight Log Management (LLM) System....205
Debug the CLE System Debugger Using debugraw and debugmax....205
cdump and crash Utilities for Node Memory Dump and Analysis....206
Resource Utilization Reporting....206
Overview of RUR Configuration....207
Enable and Configure RUR....207
Configure the cray_alps Service for Per-application RUR....214
Configure a WLM to Enable Per-job RUR....215
Refresh Nodes with Updated Config Sets....216
Enable/Disable Plugins....217
The dws Data Plugin....219
The dws_job_server Data Plugin....219
The dws_server Data Plugin....222
The energy Data Plugin....225
The gpustat Data Plugin....227
The memory Data Plugin....227

The nodeuse Data Plugin....229
The taskstats Data Plugin....229
The timestamp Data Plugin....232
The file Output Plugin....232
The llm Output Plugin....232
The user Output Plugin....232
The database Example Output Plugin....233
Create Custom RUR Data Plugins....233
Create Custom RUR Output Plugins....235
Implement a Site-Written RUR Plugin....236
Additional Plugin Examples....238
Application Completion Reporting (ACR) to RUR Migration Tips....241
Application Resource Utilization (ARU) to RUR Migration Tips....243
CSA to RUR Migration Tips....243
Modify an Installed System....246
Configure a Boot Failover Node....246
Disable Boot Node Failover....249
Configure an SDB Failover Node....250
Perform SDB Node Failback....253
Perform Boot Node Failback....254
Configure Realm-Specific IP....254
Use the `xtrsipcfg_v2` Script for an Advanced RSIP Configuration....255
Update cray_net Worksheet for an Advanced RSIP Configuration....258
The Node ARP Management Daemon (rca_arpd)....259
Create Logical Machines for Cray XC Series Systems....260
Configure a Logical Machine....260
Boot a Logical Machine....261
Boot the System Using Another Snapshot....261
Configure the NFS client to Mount the Exported Lustre File System....262
Define Bind Mount Points Within a Configuration Set....263
Enable Multipath on an Installed XC System....264
Change Lustre Versions....270
Repurpose Compute Nodes....274
Node Attributes....274
View and Temporarily Set Node Attributes....274
The XTAdmin Database segment Table....275
Apply Rolling Patches to Compute Nodes with `cnat`....276
Apply Live Updates to Nodes....277

Reuse One or More Previously-failed HSN Links....278
Add or Remove from Service....279
Remove a Compute Blade from Service While the System is Running....279
Return a Compute Blade into Service....281
State Manager LLM Logging....282
Boot Manager LLM Logging....282
Configure Node Health Checker Tests....283
Guidance for the Accelerator Test....286
Guidance for the Application Exited Check and Apinit Ping Tests....286
Guidance for the Filesystem Test....287
Guidance for the Hugepages Test....287
Guidance for the NHC Lustre File System Test....288
NHC Control Variables....288
Global Configuration Variables that Affect all NHC Tests....289
Standard Variables that Affect Individual NHC Tests....290
NHC Suspect Mode....292
NHC Messages....293
Recover from a Login Node Crash when a Login Node will not be Rebooted....293

About the XC Series System Administration Guide

The *XC™ Series System Administration Guide* (S-2393), released June 22, 2017, includes administrative tasks for Cray XC series systems running SMW 8.0 UP04 and CLE 6.0 UP04. It is intended for use by experienced Cray system administrators.

New in this release

- *Perform Boot Node Failback* on page 254 and *Perform SDB Node Failback* on page 253.
- *Use the xtrsipcfg_v2 Script for an Advanced RSIP Configuration* on page 255

Updated in this release

- *Set the Turbo Boost Limit* and *Stop System Components* on page 58 reflect currently released processors.
- *Configure a Boot Failover Node* on page 246 and *Configure an SDB Failover Node* on page 250 have updates for new SDB and boot node failback functionality and config set settings.

Command Prompt Conventions

Host name and account in command prompts

The host name in a command prompt indicates where the command must be run. The account that must run the command is also indicated in the prompt.

- The `root` or super-user account always has the `#` character at the end of the prompt.
- Any non-`root` account is indicated with *`account@hostname>`*. A user account that is neither `root` nor `crayadm` is referred to as `user`.

`smw#`	Run the command on the SMW as `root`.
`cmc#`	Run the command on the CMC as `root`.
`sdb#`	Run the command on the SDB node as `root`.
`crayadm@boot>`	Run the command on the boot node as the `crayadm` user.
`user@login>`	Run the command on any login node as any non-`root` user.
`hostname``#`	Run the command on the specified system as `root`.
`user@`*`hostname`*`>`	Run the command on the specified system as any non-`root` user.

`smw1#` `smw2#`	For a system configured with the SMW failover feature there are two SMWs—one in an active role and the other in a passive role. The SMW that is active at the start of a procedure is *smw1*. The SMW that is passive is *smw2*.
`smwactive#` `smwpassive#`	In some scenarios, the active SMW is *smw1* at the start of a procedure—then the procedure requires a failover to the other SMW. In this case, the documentation will continue to refer to the formerly active SMW as *smw1*, even though *smw2* is now the active SMW. If further clarification is needed in a procedure, the active SMW will be called *smwactive* and the passive SMW will be called *smwpassive*.

Command prompt inside chroot

If the `chroot` command is used, the prompt changes to indicate that it is inside a chroot environment on the system.

```
smw# chroot /path/to/chroot
chroot-smw#
```

Directory path in command prompt

Example prompts do not include the directory path, because long paths can reduce the clarity of examples. Most of the time, the command can be executed from any directory. When it matters which directory the command is invoked within, the `cd` command is used to change into the directory, and the directory is referenced with a period (`.`) to indicate the current directory.

For example, here are actual prompts as they appear on the system:

```
smw:~ # cd /etc
smw:/etc# cd /var/tmp
smw:/var/tmp# ls ./file
smw:/var/tmp# su - crayadm
crayadm@smw:~> cd /usr/bin
crayadm@smw:/usr/bin> ./command
```

And here are the same prompts as they appear in this publication:

```
smw# cd /etc
smw# cd /var/tmp
smw# ls ./file
smw# su - crayadm
crayadm@smw> cd /usr/bin
crayadm@smw> ./command
```

Typographic Conventions

`Monospace`	Indicates program code, reserved words, library functions, command-line prompts, screen output, file/path names, key strokes (e.g., `Enter` and `Alt-Ctrl-F`), and other software constructs.
`Monospaced Bold`	Indicates commands that must be entered on a command line or in response to an interactive prompt.
`Oblique` or *`Italics`*	Indicates user-supplied values in commands or syntax definitions.

Proportional Bold	Indicates a graphical user interface window or element.
\ (backslash)	At the end of a command line, indicates the Linux® shell line continuation character (lines joined by a backslash are parsed as a single line). Do not type anything after the backslash or the continuation feature will not work correctly.

Scope and Audience

This publication covers a wide range of system management topics and is intended for experienced Cray system administrators.

Trademarks

About the Cray Management System

With Cray Linux Environment (CLE) 6.0, Cray introduces a new management system built on these core principles:

- Separation of configuration data and software content
- Separation of the management infrastructure from the product content
- Modularity
- Prescriptive results
- Scalability

This Cray Management System (CMS) is intended to improve uptime through staging, reduce the risk associated with updates and changes, and enable users to extend functionality.

The CMS comprises these primary components:

IMPS Image Management and Provisioning System.

IMPS enables sites to manage software content in a prescriptive way. It leverages and extends industry-standard tools such as `zypper` and `rpm`. IMPS is used to create and distribute repository content (RPMs) and to create and update standard or custom images. Cray provides image recipes for different node types: service, login, compute, DAL, etc. The image recipes tie together the collections of software defined in the package collections and the repositories that contain the software. From them, IMPS builds a list of all the software and repositories referenced, and passes it to `zypper` or `yum`, which resolves the RPM dependencies and installs the software into the specified image root. See the IMPS man page for more information.

CMF Configuration Management Framework

The CMF is a combination of software and conventions that enable the modular management and application of configuration. Each application comes with the software needed to configure that application. All configuration information needed to operate the logical system is stored in a central repository called a config set. It is made available to every node in the system by means of the IMPS Distribution System (IDS), a read-only network share. Cray provides a configurator to enable sites to create, change, or add new configuration information. Configuration for all applications installed in an image is applied at boot time using cray-ansible, a wrapper that finds all Ansible plays installed on the system and executes them with Ansible.

NIMS Node Image Mapping Service

NIMS enables site administrators to assign any node or group of nodes any boot image. It also provides a mechanism for passing additional kernel parameters to the nodes on boot. See the NIMS man page for more information.

Ansible is installed into each image. During boot, each node runs all Ansible plays, pulling in the configuration information needed to self-configure ("pull" mode). Ansible is called twice during system boot—once from `initrd /init` before Linux has started up (*in_init*) and once after normal Linux startup with `systemd` (*multiuser*)—to cover both early and run level 3 use cases. Ansible can be run in “push” mode after boot to support reconfiguration.

Manage the System

Caution is encouraged when executing system management commands and procedures; hasty actions can result in down time and lost data.

IMPORTANT: Use persistent SCSI device names.

This does not apply to SMW disks: SCSI device names (`/dev/sd*`) are not guaranteed to be numbered the same from boot to boot. This inconsistency can cause serious system problems following a reboot. When installing CLE, the administrator **must** use persistent device names for file systems on the Cray system.

Cray recommends using the `/dev/disk/by-id/` persistent device names. Use `/dev/disk/by-id/` for the root file system in the `initramfs` image and in the `/etc/sysset.conf` installation configuration file as well as for other file systems, including Lustre (as specified in `/etc/fstab` and `/etc/sysset.conf`). For more information, see *CLE Installation and Configuration Guide*.

Alternatively, the administrator can define persistent names using a site-specific `udev` rule or `cray-scsidev-emulation`. However, only the `/dev/disk/by-id` method has been verified and tested.

CAUTION: The administrator must use `/dev/disk/by-id` when specifying the root file system. There is no support in the `initramfs` for `cray-scsidev-emulation` or custom `udev` rules.

Connect the SMW to the Console of a Service Node

The `xtcon` command is a console interface for service nodes. When it is executing, the `xtcon` command provides a two-way connection to the console of any running node.

With the release CLE 6.x, all service and compute nodes enable the `xtcon` console by default. If a node fails to boot, then the init boot sequence halts and drops into a console bash session waiting for the administrator to take action, such as debug the node. With release CLE 5.x, `xtcon` and the enablement of console on nodes is required via the kernel parameters.

See the `xtcon(8)` man page for additional information.

Configure Remote Access to SMW with VNC

Virtual network computing (VNC) software enables a user to view and interact with the SMW from another computer.

VNC is optional and enabling VNC is a site choice. With the DRAC on the SMW, many system administrators may prefer to use DRAC and not configure VNC.

To obtain a VNC client to connect to the server, download a VNC client from a reuptable website such as these:

- RealVNC™: *http://www.realvnc.com/*
- TightVNC™: *http://www.tightvnc.com/*

The VNC software requires a TCP/IP connection between the server and the viewer. Be aware that VNC is considered to be an insecure protocol, therefore Cray recommends that the VNC client only connect to the VNC server on the SMW via an SSH tunnel.

About the Integrated Dell Remote Access Controller (iDRAC)

The iDRAC is a systems management hardware and software solution that provides remote management capabilities, crashed system recovery, and power control functions for the System Management Workstation (SMW). The iDRAC alerts administrators to server issues, helps them perform remote server management, and reduces the need for physical access to the server. The iDRAC also facilitates inventory management and monitoring, deployment and troubleshooting. To help diagnose the probable cause of a system crash, the iDRAC can log event data and capture an image of the screen when it detects that the system has crashed.

Change the Default iDRAC Password

About this task

This procedure describes how to log in to the iDRAC web interface and change a user password.

Procedure

1. Log in to the web interface as `root`.
2. Select **iDRAC settings** on the left navigation bar.
3. Expand **iDRAC settings** on the left navigation bar.
4. Select **User Authentication**.
5. Select the user whose password is changing. To change the root password, select userid `2`.
6. Select **Next**.
7. Select the **Change Password** box and enter the new password in the boxes below it.
8. Select **Apply** to complete the password change.

The password change is complete.

Alternative. Another approach to changing the iDRAC root password is to use `ipmitool` on the SMW command line interface.

```
smw# ipmitool -U root -I lanplus -H <drac-ip-addr> -P <old-drac-password> \
user set password 2 <new-drac-password>
```

Dell R815 SMW: Change the BIOS and iDRAC Settings

Prerequisites

This procedure assumes the following:

- The SMW is disconnected from the boot RAID.

- The SMW is connected to a keyboard, monitor, and mouse (without this direct connection, some procedure instructions will not work as intended).

About this task

This procedure changes the system setup for a Dell R815 SMW: the network connections, remote power control, and the remote console. Depending on the server model and version of BIOS configuration utility, there may be minor differences in the steps to configure the system. For more information, refer to the documentation for the Dell server used at this site. Because Cray ships systems with most of the installation and configuration completed, some of these steps may have been done already.

For a Dell R630 SMW, see *Dell R630 SMW: Change the BIOS and iDRAC Settings* on page 23.

Procedure

1. Remove SMW non-boot internal drives.

   Eject all the internal disk drives from the SMW except for the primary boot disk in slot 0 and the secondary boot disk in slot 1.

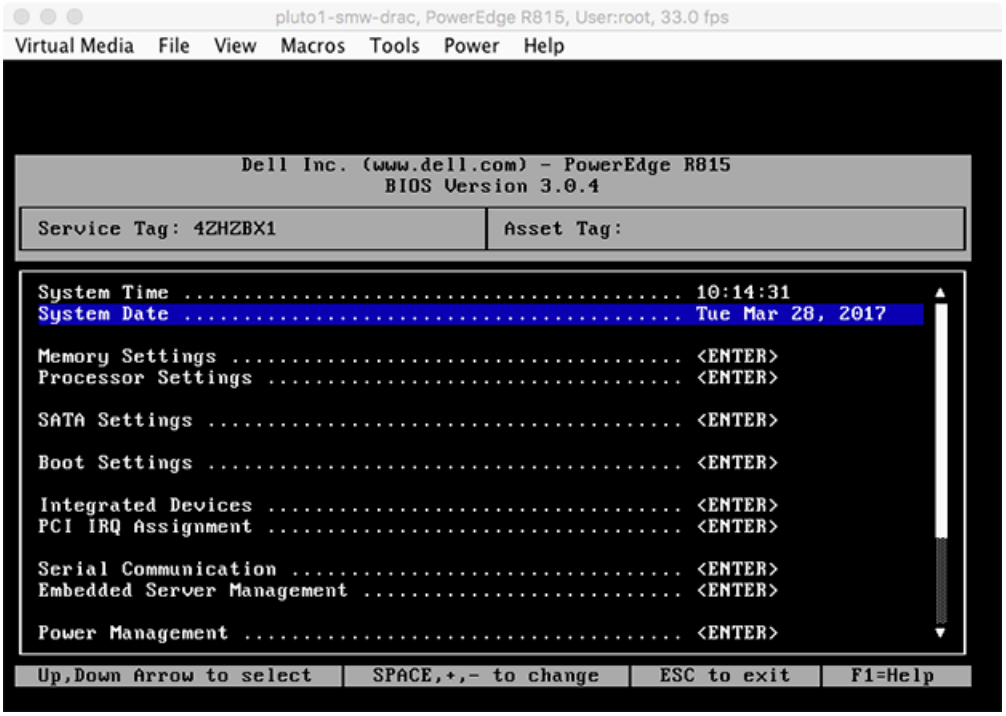
2. Power up the SMW. When the BIOS power-on self-test (POST) process begins, **quickly press the F2 key** after the following messages appear in the upper-right of the screen.

   ```
           F2 = System Setup
     F10 = System Services
   F11 = BIOS Boot Manager
           F12 = PXE Boot
   ```

   When the **F2** keypress is recognized, the **F2 = System Setup** line changes to **Entering System Setup**.

   After the POST process completes and all disk and network controllers have been initialized, the BIOS **System Setup** menu appears.

Figure 1. Dell R815 SMW BIOS System Setup Menu



3. Change system time.

The system time should be in UTC, not in the local timezone.

a. Select **System Time** in the **System Setup** menu.

The hours will be highlighted in blue.

b. Set the correct time.

1. Press the space key to change hours.

2. Use the right-arrow key to select minutes, then change minutes with the space key.

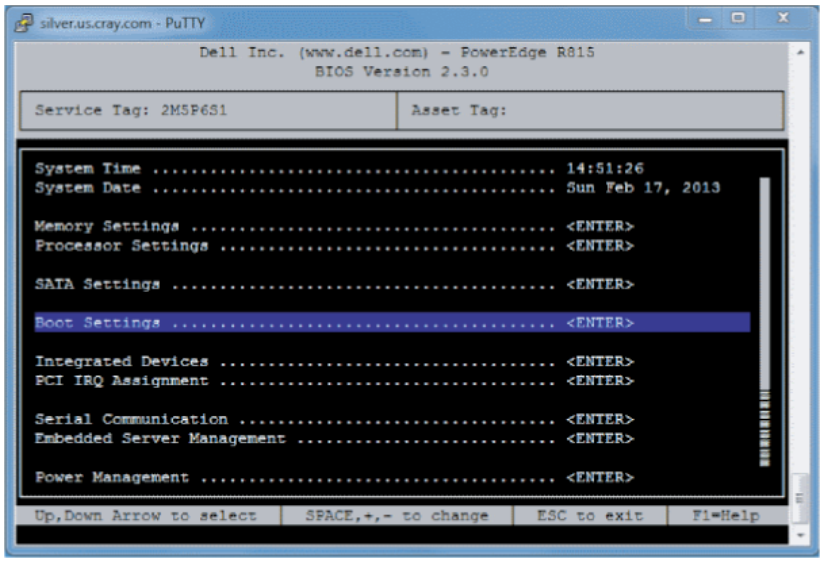
3. Use the right-arrow key to select seconds, then change seconds with the space key.

c. Press **Esc** when the correct time is set.

4. Change boot settings.

a. Select **Boot Settings** in the **System Setup** menu, then press **Enter**.

Figure 2. Dell R815 SMW Boot Settings Menu

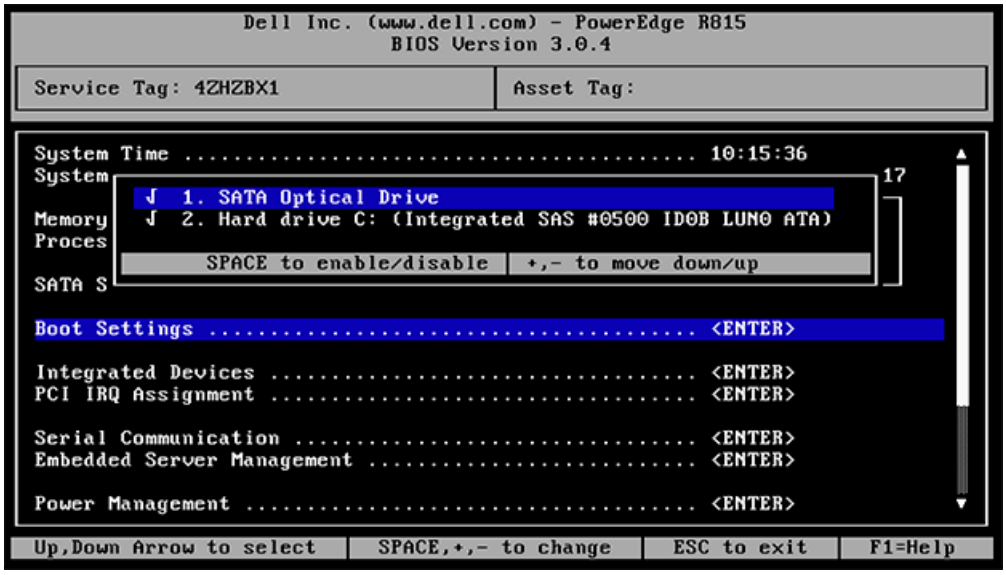


A pop-up menu with the following list appears:

```
Boot Mode ..................................... BIOS
Boot Sequence ................................. <ENTER>
USB Flash Drive Emulation Type................. <ENTER>
Boot Sequence Retry ........................... <Disabled>
```

b. Select **Boot Sequence**, then press **Enter**.

Figure 3. Dell R815 SMW Boot Sequence Settings



c. Change the order of items in the **Boot Sequence** list so that the optical (DVD) drive appears first, then the hard drive. If **Embedded NIC** appears in the list, it should end up below the optical drive and hard drive in the list.

d. Disable embedded NIC.

If **Embedded NIC** is in the list, select it and press **Enter**, then use the space key to disable it.

e. Press **Esc** to exit the **Boot Sequence** menu.

f. Press **Esc** again to exit the **Boot Settings** menu.

5. Change serial communication.

a. Select **Serial Communication** in the **System Setup** menu, then press **Enter**.

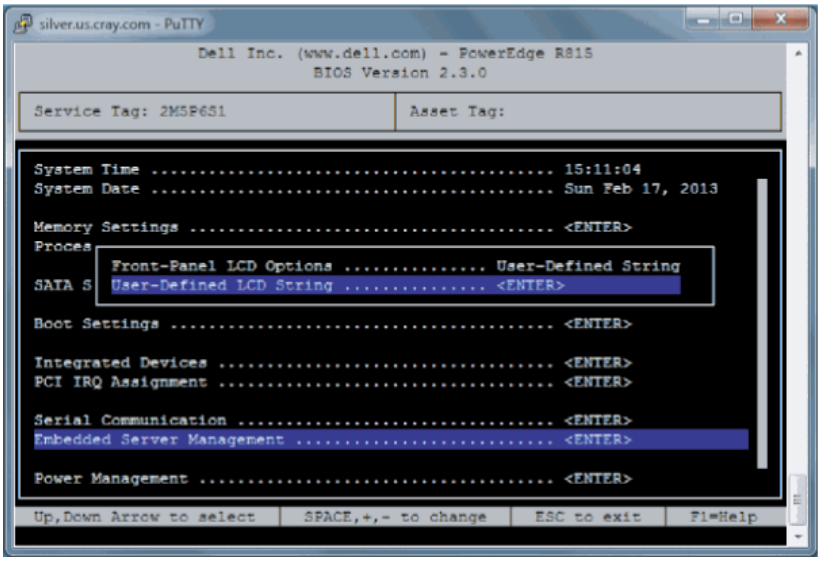
b. Confirm these settings in the **Serial Communication** menu.

- **Serial Communication** is set to **On with Console Redirection via COM2**
- **Serial Port Address** is set to **Serial Device1=COM2, Serial Device2=COM1**
- **External Serial Connector** is set to **Serial Device2**
- **Failsafe Baud Rate** is set to **115200**

c. Press **Esc** to exit the **Serial Communication** menu.

6. Select **Embedded Server Management** in the **System Setup** menu, then press **Enter**.

Figure 4. Dell R815 SMW Embedded Server Management Settings



a. Set **Front-Panel LCD Options** to **User-Defined LCD String** in the **Embedded Server Management** menu. Use the space key to cycle through the choices, then use the down-arrow key.

b. Set **User-Defined LCD String** to the login hostname (e.g., cray-drac), then press **Enter**.

c. Press **Esc** to exit the **Embedded Server Management** menu.

7. Insert base operating system DVD into SMW.

Insert the base OS DVD labeled into the DVD drive. (The DVD drive on the front of the SMW may be hidden by a removable decorative bezel.)

8. Save BIOS changes and exit.

a. Press **Esc** to exit the BIOS **System Setup** menu.

A menu with a list of exit options appears.

```
Save changes and exit
Discard changes and exit
Return to Setup
```

b. Ensure that **Save changes and exit** is selected, then press **Enter**.

The SMW resets automatically.

9. Enter BIOS boot manager.

a. When the BIOS POST process begins again, **quickly press the F11 key** within 5 seconds of when the following messages appear in the upper-right of the screen.

```
               F2 = System Setup
         F10 = System Services
       F11 = BIOS Boot Manager
                F12 = PXE Boot
```

When the **F11** keypress is recognized, the **F11 = BIOS Boot Manager** line changes to **Entering BIOS Boot Manager**.

10. Change the integrated Dell Remote Access Controller (iDRAC) settings.

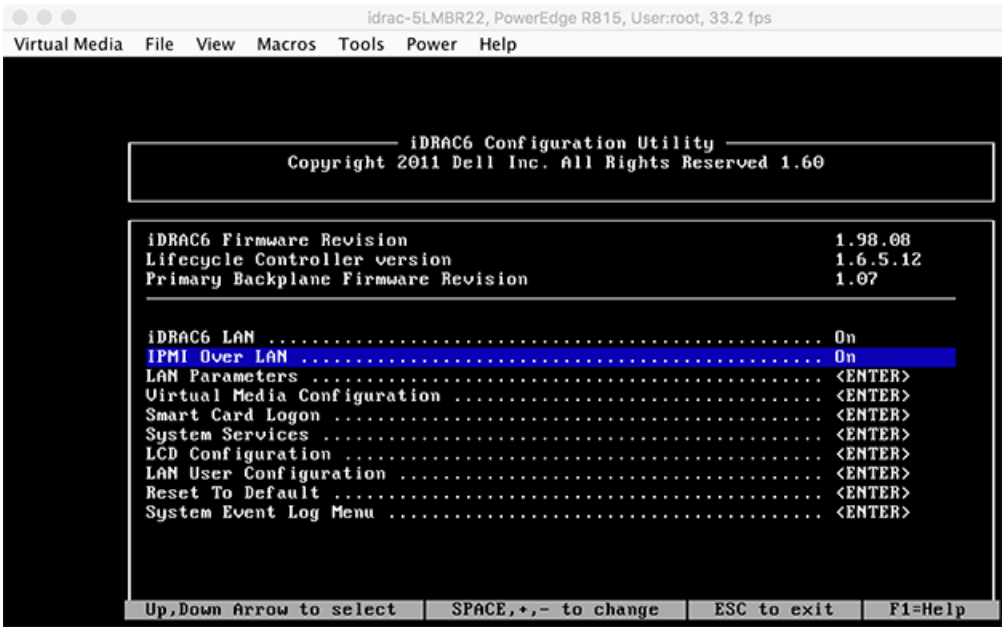
Watch the screen carefully as text scrolls until the **iDRAC6 Configuration Utility 1.57** line is visible. When the line **Press <Ctrl-E> for Remote Access Setup within 5 sec...** displays, press **Ctrl-E** within 5 seconds.

```
0 5 0 ATA WDC WD5000BPVT-0 1A01 465 GB
LSI Corporation MPT2 boot ROM successfully installed!
iDRAC6 Configuration Utility 1.57
Copyright 2010 Dell Inc. All Rights Reserved
iDRAC6 Firmware Revision version: 1.54.15
Primary Backplane Firmware Revision 1.07
-------------------------------
IPv6 Settings
-------------------------------
IPv6 Stack : Disabled
Address 1 : ::
Default Gateway : ::
-------------------------------
IPv4 Settings
-------------------------------
IPv4 Stack : Enabled
IP Address : 172. 31. 73.142
Subnet mask : 255.255.255. 0
Default Gateway : 172. 31. 73. 1
Press <Ctrl-E> for Remote Access Setup within 5 sec...
```

The **iDRAC6 Configuration Utility** menu appears.

11. Set **iDRAC6 LAN** to **ON**.

Figure 5. Dell R815 SMW iDRAC6 Configuration Utility Menu



12. Set **IPMI Over LAN** to **ON**.

13. Configure the iDRAC LAN parameters.

Select **LAN Parameters**, then press **Enter**.

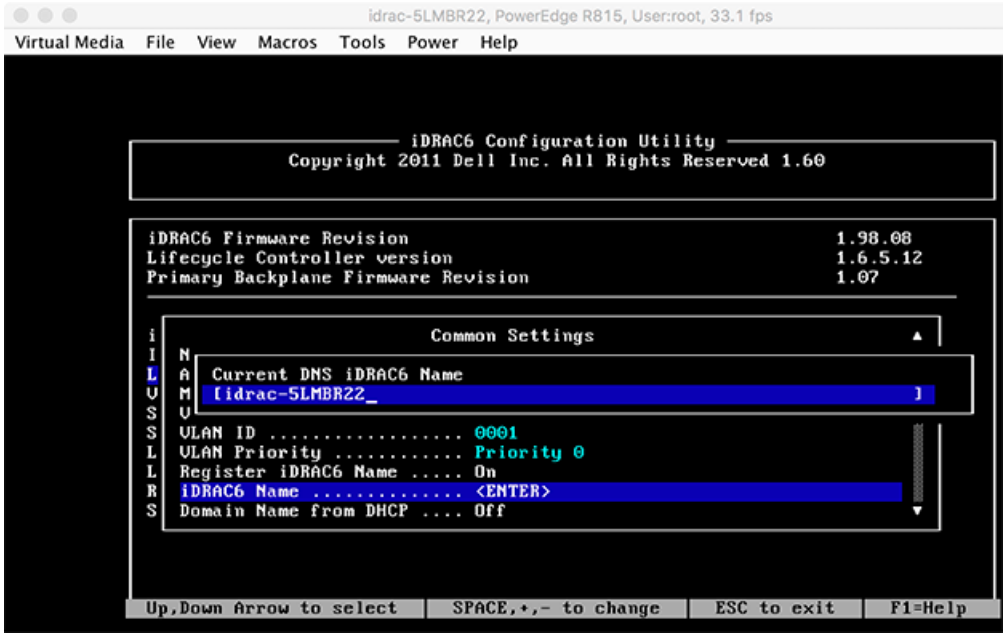
a. Configure iDRAC6 name.

Use the arrow key to scroll down and select **iDRAC6 Name**, then press **Enter**. Enter a value for **Current DNS iDRAC6 Name** (e.g., smw-drac), then press **Esc**.

Trouble? If unable to set the iDRAC6 name, try this:

1. Temporarily set **Register iDRAC6 Name** to "On."
2. Press **Enter** to set **iDRAC6 Name**. Select current or suggested name (edit enabled). When done, press **Esc**.
3. Return to **Register iDRAC6 Name** and set it to "Off."

Figure 6. Dell R815 SMW iDRAC6 LAN Parameters: iDRAC6 Name



b. Configure domain name.

   Use the arrow key to scroll down and select **Domain Name**, then press **Enter**. Enter a value for **Current Domain Name** (e.g., us.cray.com), then press **Enter**.

c. Configure hostname string.

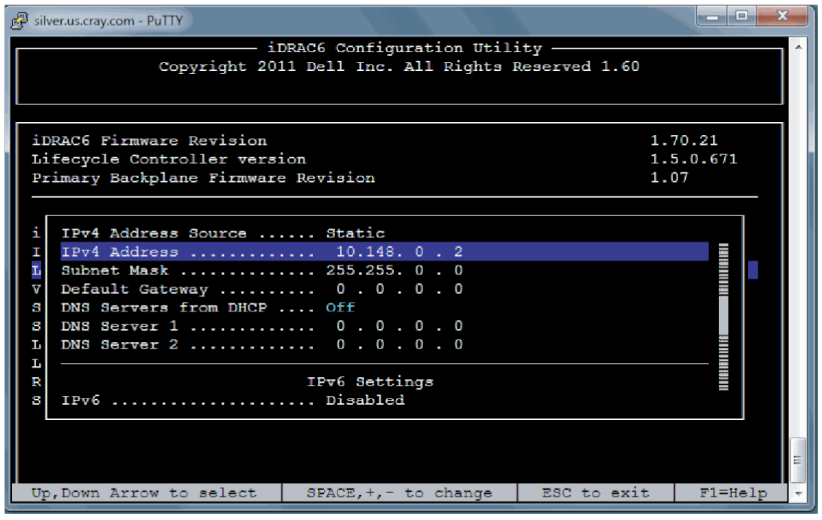
   Use the arrow key to scroll down and select **Host Name String**, then press **Enter**. Enter a value for **Current Host Name String** (e.g., smw-drac), then press **Esc**.

d. Configure IPv4 settings.

   Use the arrow key to scroll down into the **IPv4 Settings** group and confirm that the **IPv4 Address Source** is set to **static**. Then enter values for the following:

   IPv4 Address (the SMW DRAC IP address)
   Subnet Mask (the SMW iDRAC subnet mask)
   Default Gateway (the SMW iDRAC default gateway)
   DNS Server 1 (the first site DNS server)
   DNS Server 2 (the second site DNS server)

Figure 7. Dell R815 SMW DRAC IPv4 Parameter Settings



e. Configure IPv6 settings.

   Use the arrow key to scroll down into the **IPv6 Settings** group and ensure that **IPv6** is disabled.

f. Press **Esc** to exit **LAN Parameters** and return to the **iDRAC6 Configuration Utility** menu.

14. Configure iDRAC virtual media.

a. Select **Domain Name**, then press **Enter**.

b. Select **Virtual Media Configuration**, then press **Enter**.

c. Select the **Virtual Media** line and press the space key until it indicates **Detached**.

d. Press **Esc** to exit the **Virtual Media Configuration** menu.

15. Set the password for the iDRAC LAN root account.

Using the arrow keys, select **LAN User Configuration**, then press **Enter**. The following configuration is for both SSH and web browser access to the iDRAC.

a. Select **Account User Name** and enter the account name "root."

b. Select **Enter Password** and enter the intended password.

c. Select **Confirm Password** and enter the intended password again.

d. Press **Esc** to return to the **iDRAC6 Configuration Utility** menu.

16. Exit the iDRAC configuration utility.

a. Press **Esc** to exit the **iDRAC6 Configuration Utility** menu.

b. Select **Save Changes and Exit**.

The **BIOS Boot Manager** menu appears.

17. Choose to boot from SATA Optical Drive.

Using the arrow keys, select the **SATA Optical Drive** entry, then press **Enter**.

Dell R630 SMW: Change the BIOS and iDRAC Settings

Prerequisites

This procedure assumes the following:

- The *Configure the Dell R630 SMW RAID Virtual Disks* procedure has been completed.
- The SMW is rebooting. If the SMW is not rebooting, press **Ctrl-Alt-Delete** to reboot when ready to begin this procedure.

About this task

This procedure describes how to change the system setup for the SMW: the network connections, remote power control, and the remote console. This procedure includes detailed steps for the Dell R630 server. Depending on the server model and version of BIOS configuration utility, there could be minor differences in the steps to configure the system. For more information, refer to the documentation for the Dell server used at this site. Because Cray ships systems with most of the installation and configuration completed, some of the steps may have been done already.

For a Dell R815 server, see *Dell R815 SMW: Change the BIOS and iDRAC Settings* on page 14.

Procedure

Watch as the system reboots and the BIOS power-on self-test (POST) process begins. Be prepared to press **F2**, when prompted, to change the system setup.

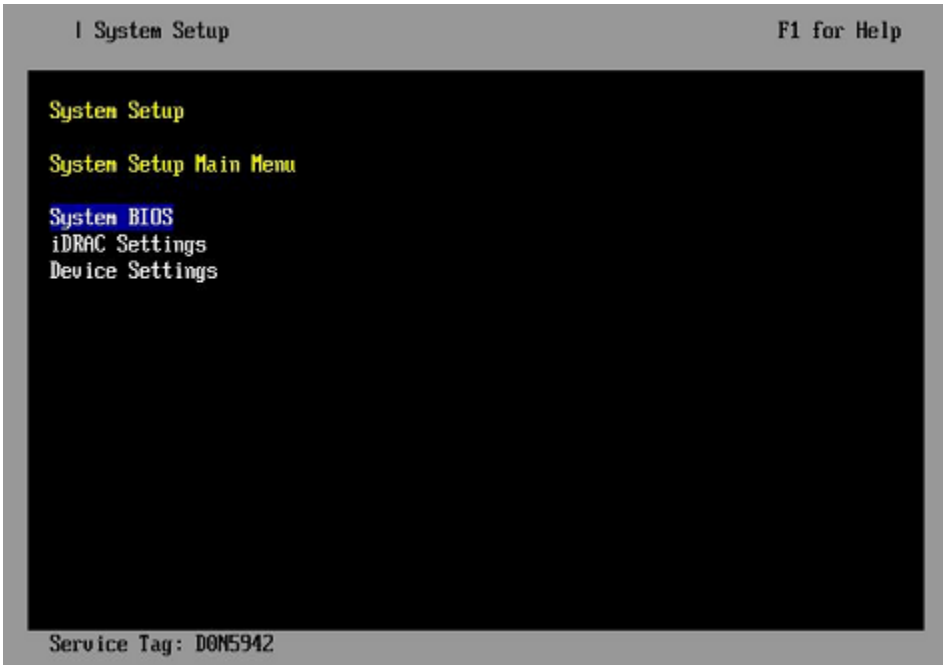
1. Press the **F2** key immediately after the following messages appear in the upper-left of the screen:

```
F2 = System Setup
F10 = Lifecycle Controller (Config iDRAC, Update FW, Install OS)
F11 = Boot Manager
F12 = PXE Boot
```

When the **F2** keypress is recognized, the **F2 = System Setup** line changes color from white-on-black to white-on-blue.

After the POST process completes and all disk and network controllers have been initialized, the Dell **System Setup** screen appears. The following submenus are available on the **System Setup Main Menu** and will be used in subsequent steps: **System BIOS**, **iDRAC Settings**, and **Device Settings**.

Figure 8. Dell R630 System Setup Main Menu



TIP: In system setup screens,

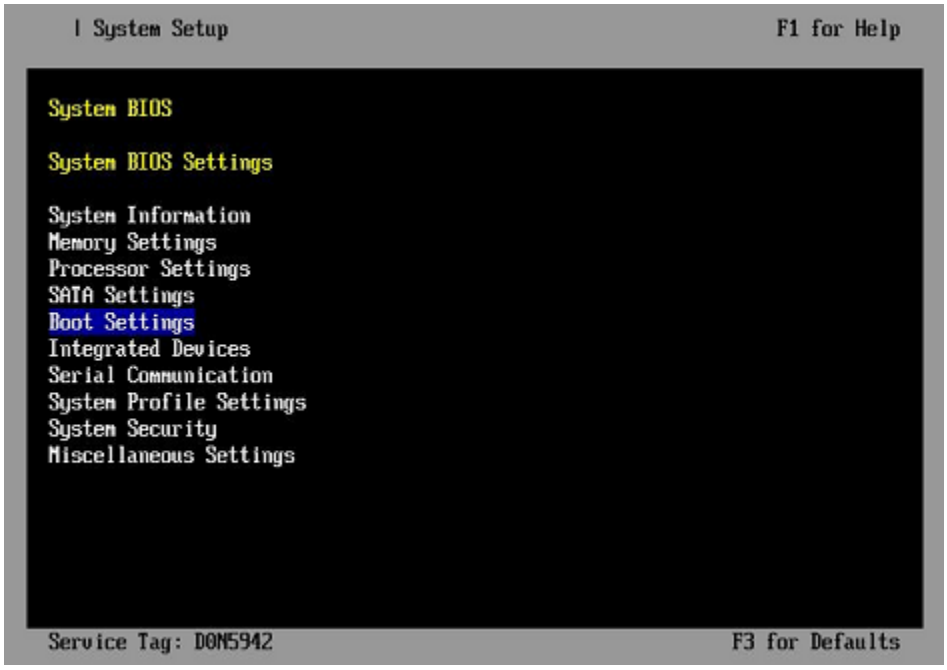
- Use the **Tab** key to move to different areas on the screen.
- Use the up-arrow and down-arrow keys to highlight or select an item in a list, then press the **Enter** key to enter or apply the item.
- Press the **Esc** key to exit a submenu and return to the previous screen.

2. Change the BIOS settings.

a. Select **System BIOS** on the **System Setup Main Menu**, then press **Enter**.

The **System BIOS Settings** screen appears.

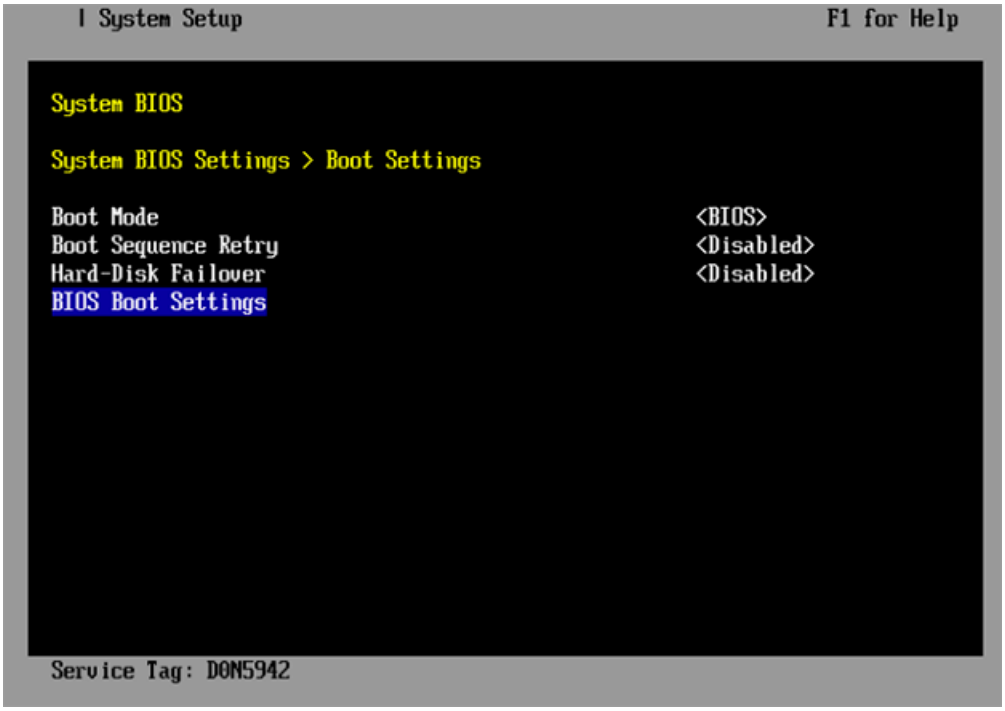
Figure 9. Dell R630 System BIOS Settings Screen



b. Change Boot Settings.

1. Select **Boot Settings** on the **System BIOS Settings** screen, then press **Enter**. The **Boot Settings** screen appears.

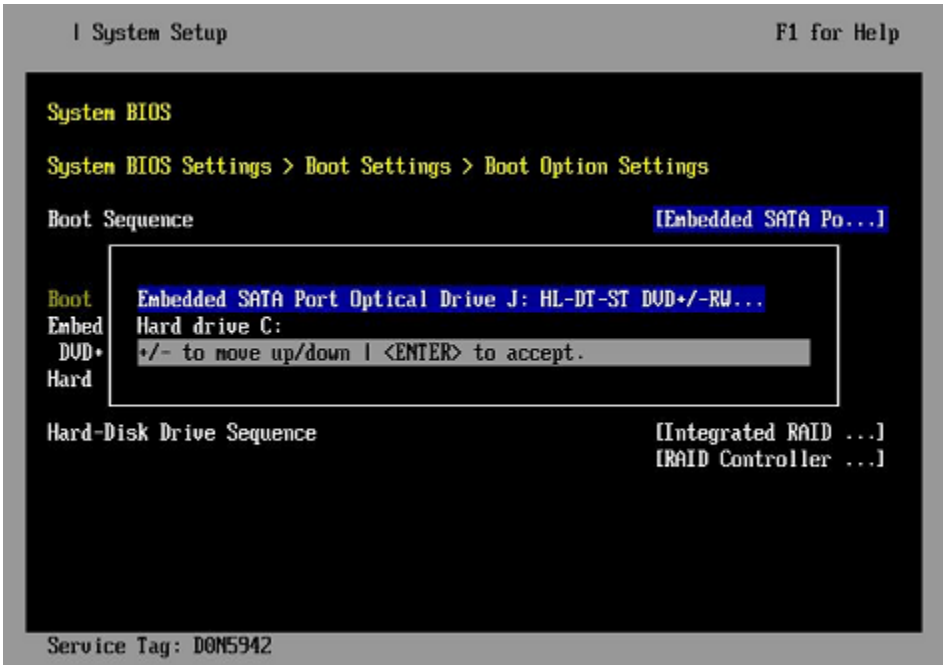
Figure 10. Dell R630 Boot Settings Screen



2. Ensure that **Boot Mode** is **BIOS** and not **UEFI**.

3. Select **BIOS Boot Settings**, then press **Enter**.
4. Select **Boot Sequence** on the **Boot Option Settings** screen, then press **Enter** to view a pop-up window with the boot sequence.

Figure 11. Dell R630 BIOS Boot Sequence

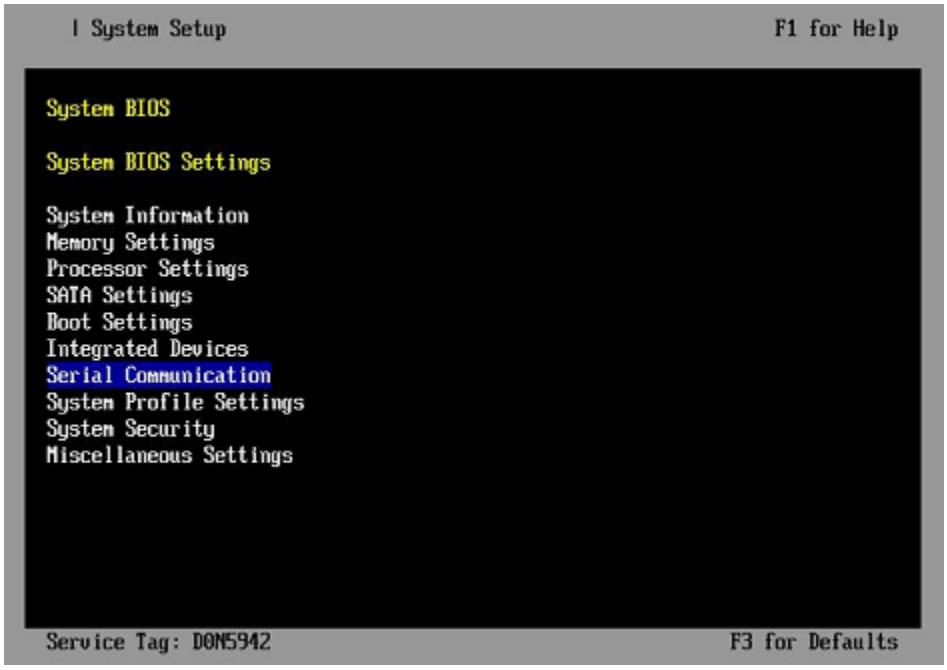


5. Change the boot order in the pop-up window so that the optical drive appears first, then the hard drive. If **Integrated NIC** appears in the list, it should end up below the optical drive and hard drive in the list.

   TIP: Use the up-arrow or down-arrow key to highlight or select an item, then use the **+** and **-** keys to move the item up or down.
6. Select **OK**, then press **Enter** to accept the change.
7. Click the box next to **Hard drive C:** under the **Boot Option/Enable/Disable** section to enable it. Do the same for the optical drive, if necessary.
8. Select **integrated NIC**, then press **Enter** to disable it.
9. Press **Esc** to exit **Boot Option Settings**.
10. Press **Esc** to exit **Boot Settings** and return to the **System BIOS Settings** screen.

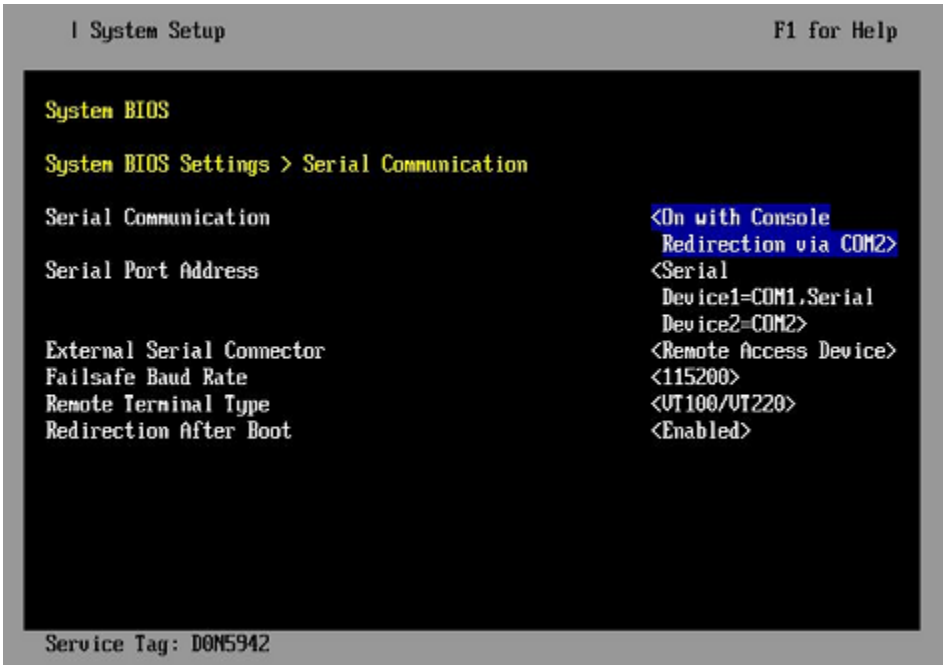
c. Change Serial Communication Settings.

Figure 12. Dell R630 System BIOS Settings: Serial Communication



1. Select **Serial Communication** on the **System BIOS Settings** screen. The **Serial Communication** screen appears.

   Figure 13. Dell R630 Serial Communication Screen



2. Select **Serial Communication** on the **Serial Communication** screen, then press **Enter**. A pop-up window displays the available options.
3. Select **On with Console Redirection via COM2** in the pop-up window, then press **Enter** to accept the change.

4. Select **Serial Port Address**, then select **Serial Device1=COM1, Serial Device2=COM2**, then press **Enter**.
5. Select **External Serial Connector**, then press **Enter**. A pop-up window displays the available options.
6. Select **Remote Access Device** in the pop-up window, then press **Enter** to return to the previous screen.
7. Select **Failsafe Baud Rate**, then press **Enter**. A pop-up window displays the available options.
8. Select **115200** in the pop-up window, then press **Enter** to return to the previous screen.
9. Press the **Esc** key to exit the **Serial Communication** screen.
10. Press **Esc** to exit the **System BIOS Settings** screen. A "Settings have changed" message appears.
11. Select **Yes** to save changes. A "Settings saved successfully" message appears.
12. Select **Ok**.

3. Change the iDRAC (Integrated Dell Remote Access Controller) settings.

   Select **iDRAC Settings** on the **System Setup Main Menu**, then press **Enter**.

   The **iDRAC Settings** screen appears.

   Figure 14. Dell R630 iDRAC Settings Screen

4. Change the iDRAC network.

   a. Select **Network** to display a long list of network settings.

   b. Change the DNS DRAC name.

      Use the arrow key to scroll down to **DNS DRAC Name**, then enter an iDRAC hostname that is similar to the SMW node hostname (e.g., cray-drac).

   c. Change the static DNS domain name.

Use the arrow key to scroll down to **Static DNS Domain Name**, then enter the DNS domain name and press **Enter**.

d. Change the IPv4 settings.

Use the arrow key to scroll down to the **IPV4 SETTINGS** list.

1. Ensure that IPv4 is enabled.
   a. If necessary, select **Enable IPV4**, then press **Enter**.
   b. Select **<Enabled>** in the pop-up window, then press **Enter** to return to the previous screen.
2. Ensure that DHCP is disabled.
   a. If necessary, select **Enable DHCP**, then press **Enter**.
   b. Select **<Disabled>** in the pop-up window, then press **Enter** to return to the previous screen.
3. Change the IP address.
   a. Select **Static IP Address**.
   b. Enter the IP address of the iDRAC interface (`ipmi0`) for the SMW, then press **Enter**.
4. Change the gateway.
   a. Select **Static Gateway**.
   b. Enter the appropriate value for the gateway of the network to which the iDRAC is connected, then press **Enter**.
5. Change the subnet mask.
   a. Select **Subnet Mask**.
   b. Enter the subnet mask for the network to which the iDRAC is connected (such as `255.255.255.0`), then press **Enter**.
6. Change the DNS server settings.
   a. Select **Static Preferred DNS Server**, enter the IP address of the primary DNS server, then press **Enter**.
   b. Select **Alternate DNS Server**, enter the IP address of the alternate DNS server, then press **Enter**.

e. Change the IPMI settings.

Change the IPMI settings to enable the Serial Over LAN (SOL) console.

1. Use the arrow key to scroll down to the **IPMI SETTINGS** list.
2. Ensure that **Enable IPMI over LAN** is selected.

   TIP: Use the left-arrow or right-arrow to switch between two settings.
3. Ensure that **Channel Privilege Level Limit** is set to **Administrator**.

f. Exit Network screen.

Press the **Esc** key to exit the **Network** screen and return to the **iDRAC Settings** screen.

5. Change hostname in iDRAC LCD display.

Change front panel security to show the hostname in LCD display.

a. Use the arrow key to scroll down and highlight **Front Panel Security** on the **iDRAC Settings** screen, then press **Enter**.

b. Select **Set LCD message**, then press **Enter**.

c. Select **User-Defined String**, then press **Enter**.

d. Select **User-Defined String**, then enter the SMW hostname and press **Enter**.

e. Press the **Esc** key to exit the **Front Panel Security** screen.

6. (Optional) Change the iDRAC **System Location** fields.

Change the **System Location** configuration on the **iDRAC Settings** screen to set any of these fields: **Data Center Name**, **Aisle Name**, **Rack Name**, and **Rack Slot**.

7. Configure iDRAC virtual media.

a. Select **Domain Name**, then press **Enter**.

b. Select **Virtual Media Configuration**, then press **Enter**.

c. Select the **Virtual Media** line and press the space key until it indicates **Detached**.

d. Press **Esc** to exit the **Virtual Media Configuration** menu.

8. Set the password for the iDRAC root account.

a. Use the arrow key to highlight **User Configuration** on the **iDRAC Settings** screen, then press **Enter**.

b. Confirm that User Name is root. Select **User Name**, then enter the "root" user name.

c. Select **Change Password**, then enter a new password.

d. Reenter the new password in the next pop-up window to confirm it (the default password is "calvin").

e. Press the **Esc** key to exit the **User Configuration** screen.

9. Exit iDRAC settings.

a. Press the **Esc** key to exit the **iDRAC Settings** screen.

A "Settings have changed" message appears.

b. Select **Yes**, then press **Enter** to save the changes.

A "Success" message appears.

c. Select **Ok**, then press **Enter**.

The main screen (**System Setup Main Menu**) appears.

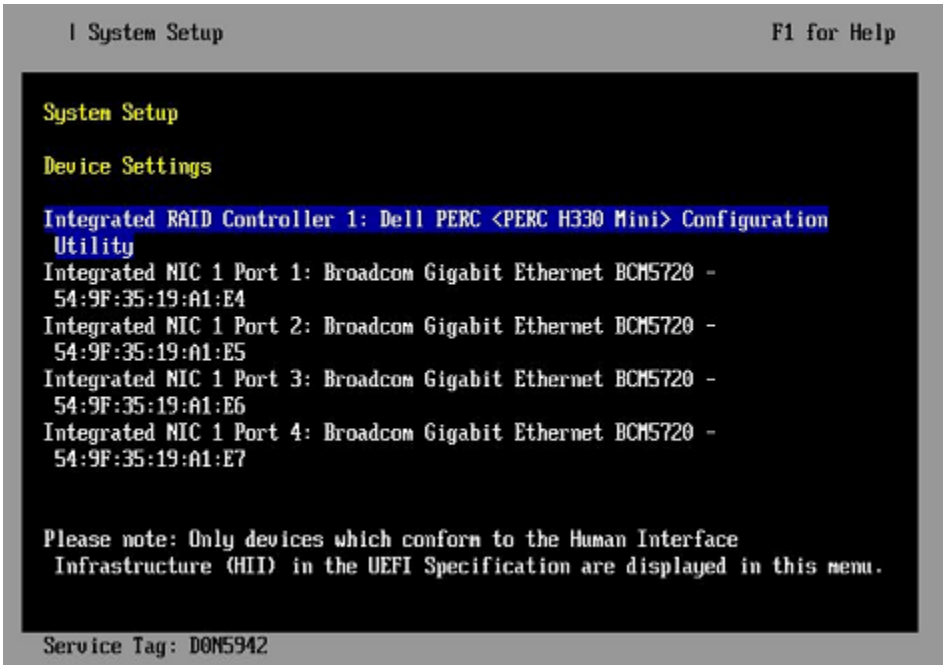
10. Change device settings.

These steps disable an integrated NIC device by changing the setting for the integrated NIC on a port from **PXE** to **None**.

a. Change Integrated NIC 1 Port 1

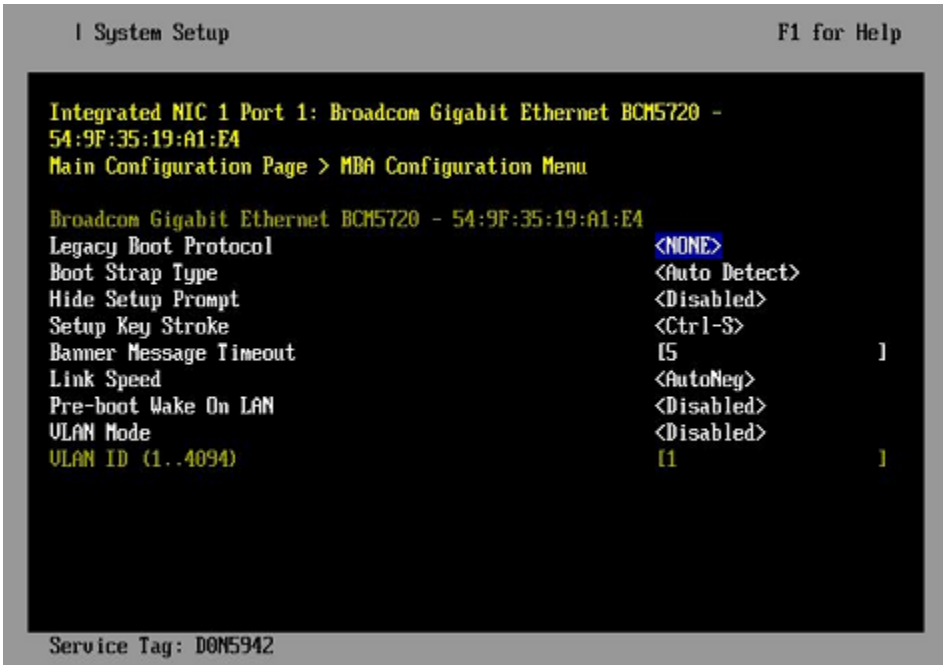
1. Select **Device Settings** on the **System Setup Main Menu**, then press **Enter**. The **Device Settings** screen appears.

Figure 15. Dell R630 Device Settings Screen



2. Select **Integrated NIC 1 Port 1: ...** on the **Device Settings** screen, then press **Enter**.
3. Select **MBA Configuration Menu** on the **Main Configuration Page** screen, then press **Enter**.

Figure 16. Dell R630 BIOS MBA Configuration Settings



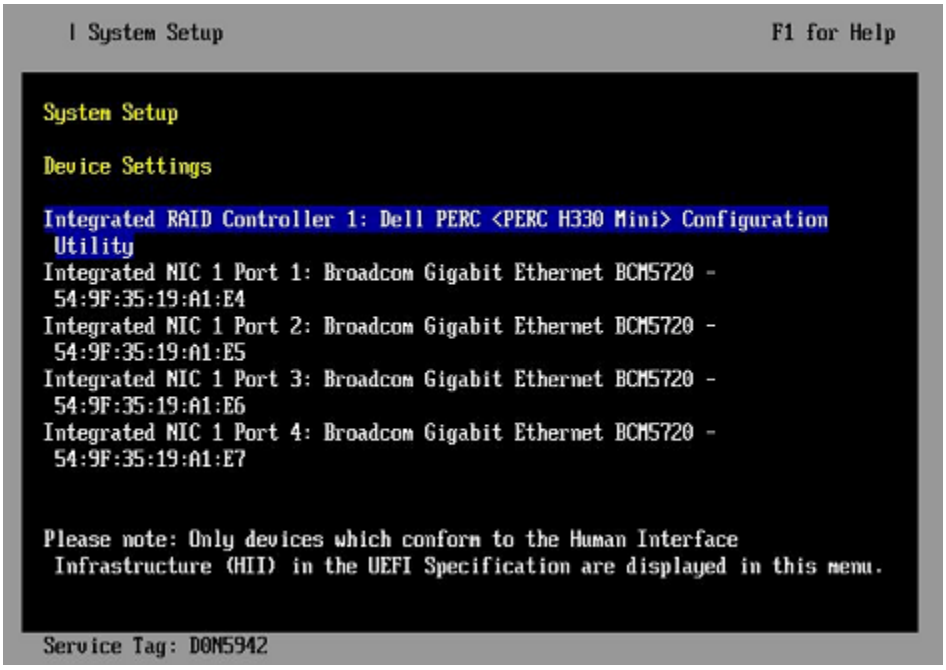
4. Select **Legacy Boot Protocol** on the **MBA Configuration Menu** screen, use the right-arrow or left-arrow key to highlight **None**, then press **Enter**.
5. Press the **Esc** key to exit the **MBA Configuration Menu** screen.

6. Press **Esc** to exit the **Main Configuration Page** screen. A "Warning Saving Changes" message appears.
7. Select **Yes**, then press **Enter** to save the changes. A "Success" message appears.
8. Select **OK**, then press **Enter**. The **Device Settings** screen appears.
9. Press **Esc** to exit the **Device Settings** screen. A "Settings have changed" message appears.
10. Select **Yes**, then press **Enter** to save the changes. A "Settings saved successfully" message appears.
11. Select **OK**, then press **Enter**. The main screen (**System Setup Main Menu**) appears.

b. Change Integrated NIC 1 Port 2

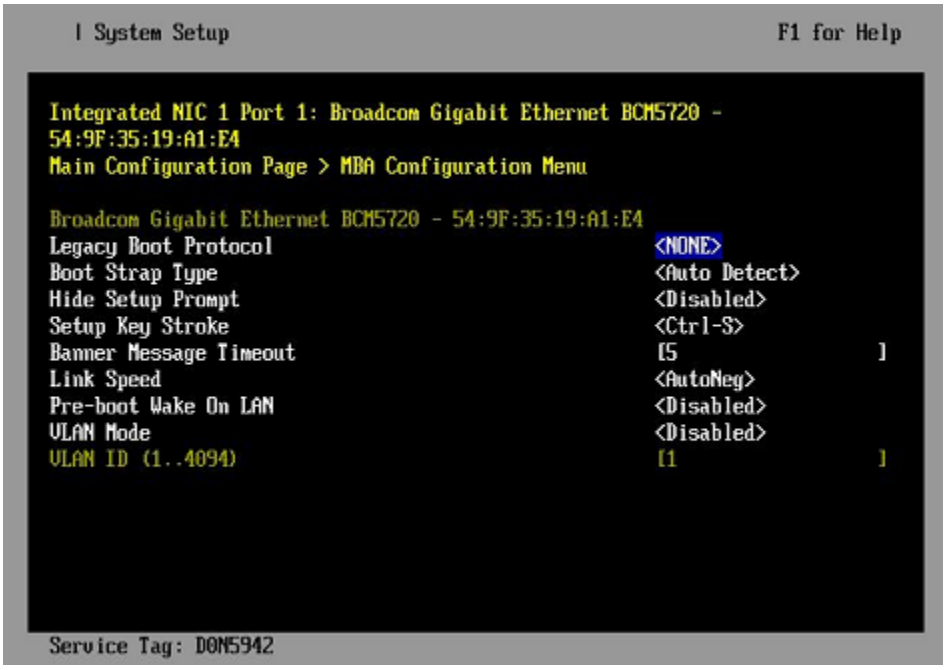
1. Select **Device Settings** on the **System Setup Main Menu**, then press **Enter**. The **Device Settings** screen appears.

Figure 17. Dell R630 Device Settings Screen



2. Select **Integrated NIC 1 Port 2: ...** on the **Device Settings** screen, then press **Enter**.
3. Select **MBA Configuration Menu** on the **Main Configuration Page** screen, then press **Enter**.

Figure 18. Dell R630 BIOS MBA Configuration Settings



4. Select **Legacy Boot Protocol** on the **MBA Configuration Menu** screen, use the right-arrow or left-arrow key to highlight **None**, then press **Enter**.
5. Press the **Esc** key to exit the **MBA Configuration Menu** screen.
6. Press **Esc** to exit the **Main Configuration Page** screen. A "Warning Saving Changes" message appears.
7. Select **Yes**, then press **Enter** to save the changes. A "Success" message appears.
8. Select **OK**, then press **Enter**. The **Device Settings** screen appears.
9. Press **Esc** to exit the **Device Settings** screen. A "Settings have changed" message appears.
10. Select **Yes**, then press **Enter** to save the changes. A "Settings saved successfully" message appears.
11. Select **OK**, then press **Enter**. The main screen (**System Setup Main Menu**) appears.

Use the iDRAC

Prerequisites

This procedure assumes an integrated Dell Remote Access Controller (iDRAC) has been set up for use with the SMW.

About this task

An iDRAC enables remote management of a Cray System Management Workstation (SMW). This procedure describes how to access the SMW console through the iDRAC.

Procedure

1. Bring up a web browser.

2. Go to: `https://cray-drac`, where *`cray-drac`* is the name assigned to the iDRAC during setup. The iDRAC login screen appears.
3. Enter the account user name and password set up in *Change the Default iDRAC Password* on page 14 or an iDRAC setup procedure.

   The **System Summary** window appears.
4. Select **Submit**.
5. To access the SMW console, select the **Console Media** tab.

   The **Virtual Console and Virtual Media** window appears.
6. Select **Launch Virtual Console**.

   TIP: By default, the console window has two cursors: one for the console and one for the administrator's window environment. To switch to single-cursor mode, select **Tools**, then **Single Cursor**. This single cursor will not move outside the console window. To exit single-cursor mode, press the **F9** key.

   TIP: To log out of the virtual console, kill the window or select **File**, then **Exit**. The web browser is still logged into the iDRAC.

For detailed information, see the iDRAC documentation at: *http://www.dell.com/support*.

Hardware Component Identification

System components (nodes, blades, chassis, cabinets, etc.) are named and located by node ID (NID), IP address, or physical ID. Physical IDs are often referred to as *cnames*.

Physical ID for Cray XC Series Systems

The physical ID identifies the cabinet's location on the floor and the component's location in the cabinet as seen by the HSS. Descriptions within the table below assume the reader is facing the front of the system cabinets.

Table 1. Physical ID Naming Conventions

Component	Format	Description
SMW	`s0`, `all`	All components attached to the SMW. `xtcli power up s0` powers up all components attached to the SMW.
cabinet	`c`*`X`*`-`*`Y`*	Compute/service cabinet, cabinet controller hostname. Not used for blower cabinets. For example: `c12-3` is cabinet 12 in row 3.
compute/service cabinet controller HSS microcontroller	`c`*`X`*`-`*`Y`*`m`*`M`*	Compute/Service cabinet controller HSS microcontroller; *`M`* is 0.
power rectifier module within a cabinet	`c`*`X`*`-`*`Y`*`r`*`R`*	Power rectifier module within a cabinet; *`R`* is 0 to 63.

Component	Format	Description
cabinet controller (CC) FPGA	c*X*-*Y*f*F*	Cabinet controller (CC) FPGA; *F* is 0.
blower cabinet	b*X*-*Y*	Blower cabinet, cabinet controller hostname (if applicable). *X* is 0 to 63; *Y* is 0 to 15. For example: b12-3 is blower cabinet 12 in row 3.
blower cabinet controller	b*X*-*Y*m*M*	Blower cabinet, cabinet controller; *M* is 0.
blower within a blower cabinet	b*X*-*Y*b*B*	Blower within a blower cabinet; *B* is 0-5.
chassis	c*X*-*Y*c*C*	Physical unit within cabinet: c*X*-*Y*; c*C* is the chassis number and *C* is 0-2. Chassis are numbered bottom to top. For example: c0-0c2 is chassis 2 of cabinet c0-0.
chassis host controller	c*X*-*Y*c*C*m*M*	Chassis host controller; *M* is 0.
optical connectors	c*X*-*Y*c*C*j*J*	Optical connectors per chassis; there are 40 optical connectors per chassis. *J* is 0-63.
chassis host FPGA	c*X*-*Y*c*C*f*F*	Chassis host FPGA; *F* is 0.
blade or slot	c*X*-*Y*c*C*s*S*	Physical unit within a slot of a chassis c*X*-*Y*c*C*; s*S* is the slot number of the blade and *S* is 0-15. For example: c0-0c2s4 is slot 4 of chassis 2 of cabinet c0-0. For example: c0-0c2s* is all slots (0...15) of chassis 2 of cabinet c0-0.
optical controller groups	c*X*-*Y*c*C*o*O*	Optical controller groups -- controller groups are associated with slots by multiplying controller number by 2 (and optionally adding 1); *O* is 0-7.
individual optical controller	c*X*-*Y*c*C*o*O*x*X*	Individual optical controller within an optical controller group; *X* is 0-4.
L0D FPGA within a base blade	c*X*-*Y*c*C*s*S*f*F*	L0D FPGA within a base blade; *F* is 0.
Aries™ ASIC	c*X*-*Y*c*C*s*S*a*A*	Aries ASIC within a base blade. There is only one Aries ASIC per blade, and all nodes on the blade connect to it. a*A* is the location of the ASIC within the blade and *A* is 0. For example: c0-1c2s3a0.

Component	Format	Description
Aries NIC	c*X*-*Yc**Cs**Sa**An**NIC*	NIC (Network Interface Controller) within an Aries ASIC; *NIC* is 0-3. For example: c0-1c2s3a0n1
LCB tile row/column	c*X*-*Yc**Cs**Sa**Al**RCol*	LCB tile row/column. Row 5 is all processor tiles; all other rows contain only HSN tiles. Note the octal numbering. *R* is 0-5 and *Col* is 0-7.
SerDes macro associated with an LCB	c*X*-*Yc**Cs**Sa**Am**RCol*	SerDes macro associated with an LCB. Note the octal numbering. *R* is 0-5 and *Col* is 0-7.
SerDes macro network processor associated with an LCB	c*X*-*Yc**Cs**Sa**Ap**RCol*	SerDes macro network processor associated with an LCB. Note the octal numbering. *R* is 0-5 and *Col* is 0-7.
Aries ASIC VRM	c*X*-*Yc**Cs**Sa**Av**V*	Aries ASIC VRM; *V* is 0.
Processor Daughter Card (PDC) within a base blade	c*X*-*Yc**Cs**Sp**P*	Processor Daughter Card within a base blade; *P* is 0-3.
quad Processor Daughter Card (QPDC) within a base blade	c*X*-*Yc**Cs**Sq**Q*	Quad Processor Daughter Card within a base blade; *Q* is 0-1.
general-purpose-accelerator Processor Daughter Card (GPDC) within a base blade	c*X*-*Yc**Cs**Sk**K*	General-purpose-accelerator Processor Daughter Card (GPDC) within a base blade; *K* is 0-1.
L0C FPGA within a PDC	c*X*-*Yc**Cs**Sp**Pf**F*	L0C FPGA within a PDC; *F* is 0.
L0C FPGA within a QPDC	c*X*-*Yc**Cs**Sq**Qf**F*	L0C FPGA within a Quad PDC; *F* is 0.
L0C FPGA within a GPDC	c*X*-*Yc**Cs**Sk**Kf**F*	L0C FPGA within a GPDC; *F* is 0.
VRM within a PDC associated with a processor socket	c*X*-*Yc**Cs**Sp**Pv**V*	VRM within a PDC associated with a processor socket; *V* is 0-1.
SouthBridge chip within a PDC	c*X*-*Yc**Cs**Sp**Ps**SouthBridge*	SouthBridge chip within a PDC; *SouthBridge* is 0.
SouthBridge chip within a QPDC	c*X*-*Yc**Cs**Sq**Qs**SouthBridge*	SouthBridge chip within a Quad PDC; *SouthBridge* is 0-1.
SouthBridge chip within a GPDC	c*X*-*Yc**Cs**Sk**Ks**SouthBridge*	SouthBridge chip within a GPDC; *SouthBridge* is 0-1.
blade controller HSS microcontroller within a base blade	c*X*-*Yc**Cs**Sm**M*	Blade controller HSS microcontroller within a base blade (not the blade controller CPU); *M* is 0.
node	c*X*-*Yc**Cs**Sn**N*	Physical node on a base blade; n*N* is the location of the node and *N* is 0-3. For example: c0-0c2s4n0 is node 0 on blade 4 of chassis 2 in cabinet c0-0.

Component	Format	Description
		For example: `c0-0c2s4n*` is all nodes on blade 4 of chassis 2 of cabinet `c0-0`.
accelerator	c*X*-*Y*c*C*s*S*n*N*a*A*	Accelerator associated with a node; may be any type of supported accelerator. *A* is 0-7.
processor socket associated with a physical node	c*X*-*Y*c*C*s*S*n*N*s*Socket*	Processor socket associated with a physical node; *Socket* is 0-1.
DIMM associated with a processor socket	c*X*-*Y*c*C*s*S*n*N*s*Socket*m*M*	DIMM associated with a processor socket; *M* is 0-7.
VDD VRM associated with processor socket	c*X*-*Y*c*C*s*S*n*N*s*Socket*v*V*	VDD VRM associated with processor socket; *V* is 0.
VDR VRM associated with processor socket	c*X*-*Y*c*C*s*S*n*N*s*Socket*r*R*	VDR VRM associated with processor socket; *R* is 0.
die within a processor socket	c*X*-*Y*c*C*s*S*n*N*s*Socket*d*D*	Die within a processor socket; *D* is 0-3.
core within a die	c*X*-*Y*c*C*s*S*n*N*s*Socket*d*D*c*Core*	Core within a die; *Core* is 0-63.
memory controller within a die	c*X*-*Y*c*C*s*S*n*N*s*Socket*d*D*m*M*	Memory controller within a die; *M* is 0-3.
logical machine (partition)	p#	A partition is a group of components that make up a logical machine. Logical systems are numbered from 0 to the maximum number of logical systems minus one. Because `p0` is reserved to refer to the entire machine as a partition a configuration with 31 logical machines would be numbered `p1` through `p31` and `p0` would need to be deactivated or removed as it would no longer be valid.

Node ID (NID) on Cray XC Series Systems

The node ID (NID) is a decimal numbering of all CLE nodes. NIDs are sequential numberings of the nodes starting in cabinet `c0-0`. Each additional cabinet continues from the highest value of the previous cabinet; therefore, cabinet 0 has NIDs 0-191, and cabinet 1 has NIDs 192 - 383, and so on.

With the exception of Cray XC-AC (air-cooled) systems, all Cray XC Series cabinets contain three chassis; chassis 0 is the lower chassis in the cabinet. Each chassis contains sixteen blades and each blade contains four nodes. The lowest numbered NID in the cabinet is in chassis 0 slot 0 (lower left corner); slots are numbered from 0 (bottom) to 7 (top) on the left side and 8 (bottom) to 15 (top) on the right side (when facing the front of the cabinet). NID numbering begins in cabinet 0, slot 0 with NIDs 0, 1, 2, and 3; NIDs 4, 5, 6, 7 are in slot 1; this numbering scheme continues to slot 15 and then moves up to chassis 1 and so on.

Cray XC-AC systems only have one chassis, which is rotated 90 degrees counter-clockwise. Therefore, slot 0 is on the bottom right and slot 7 is on the bottom left; slot 8 (right) through 15 (left) are in the top row of the chassis (when facing the front of the cabinet).

Use the `xtnid2str` command to convert a NID to a physical ID. For information about using the `xtnid2str` command, see the `xtnid2str(8)` man page. To convert a physical ID to a NID number, use the `rtr --system-map` command and filter the output. For example:

```
crayadm@smw:~> rtr --system-map | grep c1-0c0s14n3 | awk '{ print $1 }'
251
```

Use the `nid2nic` command to print the *nid*-to-*nic_address* mappings, *nic_address*-to-*nid* mappings, and a specific *physical_location*-to-*nic_address* and *nid* mappings. For information about using the `nid2nic` command, see the `xtnid2str(8)` man page.

Extended Node ID (XNID)

An extended node ID (XNID) provides a means of addressing host nodes and their coprocessors independently even though a host and coprocessor share the same network interface. An XNID provides a handle for common communication interfaces within the system, such as PMI, LNET, TCP/IP, and DVS, to access coprocessors. This direct access permits direct (autonomous) execution of coprocessor-targeted executables.

During the installation of a system with coprocessors, the CLE installer prompts for a base extended node identifier offset value for the system. For example, assume that base is set to `50000`. That number is added to the host NID for a node containing a coprocessor. If the host node is `nid00032`, then the coprocessor is `nid50032`.

Topology Class

Each Cray system is given a topology class based on the number of cabinets and their cabling. Some commands, such as `xtbounce`, enable the administrator to specify topology class as an option.

The follow commands display the topology class of a system in their output:

- `xtcli status`
- `rca-helper -o`
- `xtclass` (executed on the SMW)

For example:

```
smw:~> xtclass
1
```

Boot the System

The `xtbootsys` command is used to manually boot the boot node, service nodes, and CNL compute nodes. An administrator can also boot the system using both user-defined and built-in procedures in automation files (e.g., `/opt/cray/hss/default/etc/auto.generic`).

```
crayadm@smw> xtbootsys -a auto.myautobootfile
```

Before modifying the `auto.generic` file, Cray recommends making a copy because it will be replaced by an SMW software upgrade. Avoid strict boot ordering of service nodes in an automated boot file. For related procedures, see .

The `xtbootsys` command prevents unintentional booting of currently booted partitions. If a boot automation file is being used, `xtbootsys` checks that file to determine if the string `shutdown` exists within any actions defined in the file. If it does, `xtbootsys` assumes that a shutdown is being done, and no further verification of operating on a booted partition occurs. If the partition is not being shut down and the boot node is in the `ready` state, `xtbootsys` announces this fact and queries for confirmation to proceed. By default, confirmation is enabled. To disable or enable confirmation when booting booted partitions, use the `xtbootsys config,confirm_booting_booted` and the `config,confirm_booting_booted_last_session` global TCL variables, the `--config` *`name=value`* on the `xtbootsys` command line, or the `XTBOOTSYS_CONFIRM_BOOTING_BOOTED` and `XTBOOTSYS_CONFIRM_BOOTING_BOOTED_LAST_SESSION` environment variables.

Run Tests after Boot is Complete

Prerequisites

This procedure assumes the following:

- The system has completed booting.
- The compute nodes are “interactive," not under workload manager (WLM) control.
- ALPS is available.

If ALPS is not available and Slurm is used as the WLM, then the compute nodes can be either "interactive" or "batch," and `srun` (the equivalent Slurm command) should be used instead of the `aprun` commands in the steps that follow.

About this task

Log in to the login node as crayadm. This can be done from the SMW to the boot node to the login node or directly from another computer to the login node without passing through the SMW and boot node. Then perform these rudimentary functionality checks.

Procedure

1. Run `apstat` to get the number of nodes to use for the following commands.

```
crayadm@login> NUMNODES=$(($(apstat -v | grep XT | awk "{print \$3}")))
crayadm@login> echo NUMNODES is $NUMNODES
```

2. Verify that all nodes run (from `/tmp`).

```
crayadm@login> cd /tmp
crayadm@login> aprun -b -n $NUMNODES -N 1 /bin/cat /proc/sys/kernel/hostname
```

3. Verify that the home directory is working by running a job.

```
crayadm@login> cd ~
crayadm@login> aprun -b -n $NUMNODES -N 1 /bin/cat /proc/sys/kernel/hostname
```

4. Verify that the Lustre directory is working by running a job.

```
crayadm@login> cd /lustre_file_system
crayadm@login> aprun -b -n $NUMNODES -N 1 /bin/cat /proc/sys/kernel/hostname
```

Manually Boot the Boot Node and Service Nodes

Prerequisites

The Lustre file system should start up before the compute nodes, and compute node Lustre clients should be unmounted before shutting down the Lustre file system.

About this task

If more than one boot image is set up to run, the administrator can check which image is set up to boot with the `xtcli boot_cfg show` or `xtcli part_cfg show p`*N* commands. To change which image is booting, see *Update the Boot Configuration* on page 50

Procedure

1. Log on to the SMW as `crayadm`.

2. Invoke the `xtbootsys` command to boot the boot node. If the system is partitioned, invoke `xtbootsys` with the `--partition p`*N* option.

   ```
   crayadm@smw:~> xtbootsys
   ```

   The `xtbootsys` command prompts with a series of questions. Cray recommends answering **`yes`** by typing **`Y`** to each question.

   ```
   Enter your boot choice:
          0) boot bootnode ...
          1) boot sdb ...
          2) boot compute ...
          3) boot service ...
          4) boot all (not supported) ...
          5) boot all_comp ...
          10) boot bootnode and wait ...
          11) boot sdb and wait ...
          12) boot compute and wait ...
          13) boot service and wait ...
          14) boot all and wait (not supported) ...
          15) boot all_comp and wait ...
          17) boot using a loadfile ...
          18) turn console flood control off ...
          19) turn console flood control on ...
          20) spawn off the network link recovery daemon (xtnlrd)...
          q) quit.
   ```

3. Select option 10 (boot bootnode and wait).

   A prompt to confirm the selection is displayed. Press the **`Enter`** key or type **`Y`** to each question to confirm.

   ```
   Do you want to boot the boot node ? [Yn] Y
   Do you want to send the ec_boot event ('no' means to only load memory) ? [Yn] Y
   ```

   After the boot node is booted, the process returns to the boot choice menu.

4. Select option 11 (boot sdb and wait).

   A prompt to confirm the selection is displayed. Press the **Enter** key or type **Y** to each question to confirm.

```
Do you want to boot the sdb node ? [Yn] Y
Do you want to send the ec_boot event ('no' means to only load memory) ? [Yn] Y
```

5. Select option 13 (boot service and wait).

   A prompt to enter a list of service nodes to be booted is displayed.

6. Type **p0** to boot the remaining service nodes in the entire system or **p*N*** (where *N* is the partition number) to boot a partition.

```
Do you want to boot service p0 ? [Yn] Y
Do you want to send the ec_boot event ('no' means to only load memory) ? [Yn] Y
```

   To confirm the selection, press the **Enter** key or type **Y** to each question.

7. Log on to any service nodes for which there are local configuration or startup scripts (such as starting Lustre) and run the scripts.

Manually Boot the Compute Nodes

Prerequisites

All service and login nodes are booted and Lustre, if configured at this time, has started.

Procedure

1. Invoke the `xtbootsys` command if it is not running.

```
crayadm@smw:~> xtbootsys
```

```
Enter your boot choice:
       0) boot bootnode ...
       1) boot sdb ...
       2) boot compute ...
       3) boot service ...
       4) boot all (not supported) ...
       5) boot all_comp ...
       10) boot bootnode and wait ...
       11) boot sdb and wait ...
       12) boot compute and wait ...
       13) boot service and wait ...
       14) boot all and wait (not supported) ...
       15) boot all_comp and wait ...
       17) boot using a loadfile ...
       18) turn console flood control off ...
       19) turn console flood control on ...
       20) spawn off the network link recovery daemon (xtnlrd)...
       q) quit.
```

2. Select option **17** (boot using a loadfile). A series of prompts are displayed. Type the responses indicated in the following example. For the `component list` prompt, type **p0** to boot the entire system, or **p***N* (where *N* is the partition number) to boot a partition. At the final three prompts, press the **Enter** key.

```
Enter your boot choice: 17
Enter a boot type string (or nothing to do nothing): CNL0
Enter a boot type option (or nothing to do nothing): compute
Enter a component list (or nothing to do nothing): p0
Enter 'any' to wait for any console output,
   or 'linux' to wait for a linux style boot,
   or anything else (or nothing) to not wait at all: Enter
Enter an alternative CPIO archive name (or nothing): Enter
Do you want to send the ec_boot event ('no' means to only load memory) ? [Yn]
Enter
```

3. Return to the `xtbootsys` menu after all compute nodes are booted. Type **`q`** to exit the `xtbootsys` program.

4. Remove the `/etc/nologin` file from all service nodes to permit a non-root account to log on.

```
smw:~# ssh root@boot
boot:~# xtunspec -r /rr/current -d /etc/nologin
```

Reboot a Single Compute Node

A system administrator can initiate a warm boot with the `xtbootsys` command's `--reboot` option. This operation performs minimal initialization followed by a boot of only the selected compute nodes. Unlike the sequence that is used by the `xtbounce` command, there is no power cycling of the Cray ASICs or of the node itself; therefore, the high-speed network (HSN) routing information is preserved. Do not specify a session identifier (`-s` or `--session` option) because `--reboot` continues the last session and adds the selected components to it.

Reboot a single comput node

For this example, reboot node `c1-0c2s1n2`:

```
crayadm@smw:~> xtbootsys --reboot c1-0c2s1n2
```

Reboot Login or Network Nodes

Login or network nodes cannot be rebooted through a `shutdown` or reboot command issued on the node; they must be restarted through the HSS system using the `xtbootsys --reboot` *`idlist`* SMW command. The HSS must be used so that the proper kernel is pushed to the node.

IMPORTANT: Do not attempt to warm boot nodes running other services in this manner.

For additional information, see the `xtbootsys(8)` man page.

Reboot login or network nodes

```
crayadm@smw:~> xtbootsys --reboot idlist
```

Reboot Many Nodes

When rebooting many CLE nodes, the default is to reboot nodes in chunks up to 96 at a time. To change this chunk size to a different value, the `reboot_maxids` variable in `xtbootsys` can be adjusted on the command line for the warm boot command. This example changes `reboot_maxids` from 96 to 512.

Reboot many nodes

```
crayadm@smw:~> xtbootsys --reboot -c reboot_maxids=512
```

Boot the SMW in Rescue Mode

Prerequisites

- Download the image to use for booting the SMW to the system that is running the browser accessing the iDRAC web interface.
- If the system running the web browser used for accessing iDRAC is *not* a Windows machine, determine how to type the equivalent of F11 key. For example, on a MacBook, the keystroke is **`fn-F11`**.
- Ensure that no physical media is loaded in the drive.
- Start the Dell iDRAC for the SMW.

About this task

If unable to boot the SMW through normal means, such as when file system corruption occurs, use the Dell iDRAC web interface to start the system in rescue mode.

Procedure

1. Launch the virtual console by selecting **Overview** → **Server** → **Properties**. The **System Summary** page displays. Under **Virtual Console Preview** section, click **Launch**. The **Virtual Console Viewer** launches.

2. From the Virtual Console Viewer, launch virtual media by selecting **Virtual Media** → **Launch Virtual Media**. The **Client View** window displays.

3. Select **Add Image** and select the SMW image to launch. The name of this image is (or is similar to) `SLE-12-SP2-Server-DVD-x85_64-RC3-DVD1.iso`. Click **Open**.

4. Select the Mapped check box, which is next the selected SMW image. Leave the **Client View** window open.

5. Reinitialize BIOS and boot the system by powering on the system or, if the system was not previously shutdown, resetting the system. From the **Virtual Console Viewer**, select **Power** → **Power On System** or **Power** → **Reset System**.

   As the BIOS hardware initialization proceeds, watch the Virtual Console Viewer for instructions to press F11 for the BIOS Boot Manager and press that key or its equivalent. If the opportunity is missed, reset the system and try again.

6. In the BIOS Boot Manager, select **Virtual CD**.

7. On the SUSE boot window, select the **More...** option. Then select **Rescue System**. A prompt is displayed for access to rescue system tools.

Debug Ansible Failures During System Boot

Ansible runs in `init` and Ansible runs a second time after `systemd` completes the boot process. Ansible failures in `init` cause the affected node to drop into a debug shell for node access via `xtcon` for troubleshooting. When the debug shell is exited, Ansible is re-executed in `init`. A node's boot does not proceed until the first run of `cray-ansible` in `init` is successful.

The Ansible callback plugin captures any file changes made by Ansible file modules and stores a record of these changes in log files located at `/var/opt/cray/log/ansible/changelog`. The plugin provides detailed failure information, including the path to the task file being executed and any config set variable references in the task file.

Ansible logs under `/var/opt/cray/log/ansible` are collected via `cdump` and `xtdumpsys`. In addition, `xtdumpsys` collects the files from running nodes, changed by Ansible according to the changelog callback plugin. When possible, Ansible Cray-provided plays create a backup of files modify by a play to let the administrator to perform a diff of these files to see the changes made by Ansible. Administrators can use the `ansible_cfg_search` command to examine an image and a config set. This command outputs a list of variables and the Ansible files that accessed each variable.

Examine System Logs

Various logs receive entries during the boot process that can indicate boot problems.

`systemd` Journal

The `systemd` init system takes over the boot process after `initrd`. Use the `journalctl -a` to display all kernel messages and other information in the `systemd` journal. Using `journalctl -f` displays the most recent journal entries and continuously prints new entries. `systemd` stores messages in a custom database, the `systemd` journal. The information available in the journal includes:

- `syslogd` messages
- Kernel log messages
- `initrd` messages
- Messages written to stdout/stderr for all services

HSS Daemon Logs

The HSS daemons and the `rsyslogd` daemon running on the SMW logs to files in the `/var/opt/cray/log` directory. These daemons include `nimsd`, `xtpmd`, `xtremoted`, `xtpowerd`, `xtsnmpd`, `xtdiagd`, `erfsd`, `state_manager`, `bootmanager`, `sedc_manager`, `nid_mgr`, `erdh`, and `erd`.

SMW Command Log

The `/var/opt/cray/log/commands` log lists the commands issued from the SMW console.

CLE Boot Logs

The output from booting CLE is in the `/var/opt/cray/log/p0-current` log. For more detailed information, go to the `p0-current` directory and examine these log files:

- `bootinfo.`*`timestamp`*

Contains output from the `xtbootsys` command. Timing information for how long sections of the boot process take is listed at the bottom of this file.

- `console-`*`YYYYMMDD`*

  Contains the combined console output from every node. To find Ansible failures for a node during `init`, search for `cray-ansible: /etc/ansible/site.yaml completed in init - FAILED.`

Look Up Configuration Details

The `ansible_cfg_search` command line tool lets an administrator on the SMW specify a config set, an IMPS image root, and optionally, an Ansible play to query for config set lookups and template locations. The intent is to provide a general understanding of which configuration files are used at specific points in the boot process. The command uses the playbook structure to inspect the plays, roles, templates, and task files for patterns that appear to be config set variable lookups. For each lookup found in the Ansible content, the command lists a path to the configuration template that holds the variable.

Before using `ansible_cfg_search`, load the `system-config` module.

```
ansible_cfg_search [-h] [-p PLAYBOOK] [-s CONFIG_SETTING] [-e LOOKUP_EXPRESSION]
            [-q] config_set image
```

Required arguments:

config_set	The config set to search for config variables.
image	The IMPS image root containing ansible content to search. If necessary, use the `image list` command to find the IMPS image root.

Optional arguments:

-h, --help	Display help information.
-p *PLAYBOOK*, --playbook *PLAYBOOK*	The Ansible playbook file contained in the IMPS image to search for configuration lookups.
-s *CONFIG_SETTING*, --config-setting *CONFIG_SETTING*	List the configuration templates and Ansible files that contain the specified setting.

Example

Examine a config set to determine the settings that the baseopts.yaml play is looking up:

```
smw: # module load system-config
smw: # ansible_cfg_search p0 \
service_cle_6.0.UP03-build6.0.3074_sles_12-created20170120 \
--play baseopts.yaml
```

Output:

```
/var/opt/cray/imps/image_roots/service_cle_6.0.UP03-build6.0.3074_sles_12-created20170120/
etc/ansible/baseopts.yaml:

  - /var/opt/cray/imps/image_roots/service_cle_6.0.UP03-build6.0.3074_sles_12-
created20170120/etc/ansible/roles/baseopts/tasks/smw.yaml:
    - /var/opt/cray/imps/config/sets/p0/config/cray_user_settings_config.yaml:
      - cray_user_settings.settings.default_modules.data.smw

  - /var/opt/cray/imps/image_roots/service_cle_6.0.UP03-build6.0.3074_sles_12-
```

```
created20170120/etc/ansible/roles/baseopts/tasks/main.yaml:
    - /var/opt/cray/imps/config/sets/p0/config/cray_login_config.yaml:
      - cray_login.settings.login_nodes.data.members
    - /var/opt/cray/imps/config/sets/p0/config/cray_user_settings_config.yaml:
      - cray_user_settings.settings.default_modules.data.login
      - cray_user_settings.settings.default_modules.data.service
      - cray_user_settings.settings.default_modules.data.smw

  - /var/opt/cray/imps/image_roots/service_cle_6.0.UP03-build6.0.3074_sles_12-
created20170120/etc/ansible/roles/baseopts/tasks/login.yaml:
    - /var/opt/cray/imps/config/sets/p0/config/cray_user_settings_config.yaml:
      - cray_user_settings.settings.default_modules.data.login

  - /var/opt/cray/imps/image_roots/service_cle_6.0.UP03-build6.0.3074_sles_12-
created20170120/etc/ansible/roles/baseopts/tasks/service.yaml:
    - /var/opt/cray/imps/config/sets/p0/config/cray_user_settings_config.yaml:
      - cray_user_settings.default_modules
      - cray_user_settings.default_modules.login
      - cray_user_settings.default_modules.service
      - cray_user_settings.default_modules.smw
      - cray_user_settings.settings.default_modules.data.service
```

Examine Ansible Changelogs

The Ansible changelog provides information about files created, modified, and deleted by Ansible. Changelogs are created for `cray-ansible` when it first runs during the `init` phase and again when `cray-ansible` runs for the second time, during the booted phase. These logs are on the SMW in `/var/opt/cray/log/ansible`.

Logs created in the first phase (init):

sitelog-init	Contains Ansible play output from each task in executed plays.
file-changelog-init	Human-readable listing of each file changed by an Ansible play.
file-changelog-init.yaml	Machine-readable listing of each file changed by an Ansible play.

Logs created in the second phase (booted):

sitelog-booted	Contains Ansible play output from each task in executed plays.
file-changelog-booted	Human-readable listing of each file changed by an Ansible play.
file-changelog-booted.yaml	Machine-readable listing of each file changed by an Ansible play.

This `sitelog` entry shows that a task updated the message of the day (`motd`) file.

```
2016-01-17 12:15:27,671 TASK: [cle_motd | task motd, release]
*****************************************
2016-01-17 12:15:27,671 changed: [localhost] => {"changed": true,
"cmd": "grep RELEASE /etc/opt/cray/release/cle-release | awk -F\\='{print $2}'",
"delta": "0:00:00.002536", "end": "2016-01-17 12:15:27.471384", "rc": 0,
"start": "2016-01-17 12:15:27.468848", "stderr": "", "stdout": "6.0.UP01",
"warnings": []}
```

The location of failing task can be found in plays:

```
boot# grep -Rn "task motd, release" /etc/ansible \
/etc/opt/cray/config/current/ansible
/etc/ansible/roles/cle_motd/tasks/motd.yaml:15:- name: task motd, release
```

The `file-changelog` files show the Ansible phase, each changed file, and the play that changed the file. This an entry from a `file-changelog-init` changelog:

```
Apr 05 2016 21:07:47 (init) template: file '/etc/nologin' changed by Ansible
task file '/etc/ansible/roles/early/tasks/nologin.yaml' with owner=root,
group=root, mode=0775
```

This an entry from a `file-changelog-booted` changelog:

```
May 16 2016 22:26:39 (booted) lineinfile: file '/etc/hosts' changed by Ansible
task file '/etc/ansible/roles/hosts/tasks/main.yaml' with owner=None,
group=None, mode=None
```

The same entry for the `/etc/hosts` edit in the `file-changelog-booted.yaml` changelog:

```
- backup_file_path: ''
  file_path: /etc/hosts
  group: null
  mode: null
  module: lineinfile
  owner: null
  phase: booted
  play: populate local hostfile
  state: null
  task_file: /etc/ansible/roles/hosts/tasks/main.yaml
  task_name: Add additional hosts to master file
  time: May 16 2016 22:26:39
```

The changelog entry fields are:

Field Name	Description
backup_file_path	Location of backup copy of file modified or deleted, if available.
file_path	Full path to the file which was modified.
group	Group given to the file if created or modified, or null if not specified.
mode	Permissions changed on the file if created or modified, or null if permissions were not changed.
module	Ansible module executed.
owner	Owner given to the file if created or modified, or null if not specified.
phase	Values are "booted" or "init".
play	Name of play making change.
state	Whether a line should be "present" or "absent".
task_file	Name of the task file which made the change.
task_name	Name of the task which made this change.
time	Format is "Month Day Year HH:mm:ss".

Debug Ansible Failures in `init`

About this task

Check the console log on the SMW to find out which nodes failed. Ansible failures in `init` drop a node into debug shell. The boot process is not allowed to continue until `cray-ansible` during `init` is successful on a node.

Procedure

1. Look for `cray-ansible` failures in the SMW console log.

```
crayadm@smw~> /var/opt/cray/log/p0-current> cat console-20160523 | grep
'completed in init - FAILED'
```

```
<158>1 2016-05-23T12:01:22.576591-05:00 c0-0c0s0n1 xtconsole 31798
p0-20160523t115109 [console@34] cray-ansible: /etc/ansible/site.yaml completed
in init - FAILED.
<158>1 2016-05-23T12:01:22.576634-05:00 c0-0c0s0n1 xtconsole 31798
p0-20160523t115109 [console@34] cray-ansible: /etc/ansible/site.yaml completed
in init - FAILED.
<158>1 2016-05-23T12:01:34.411653-05:00 c0-0c0s1n2 xtconsole 31798
p0-20160523t115109 [console@34] cray-ansible: /etc/ansible/site.yaml completed
in init - FAILED.
<158>1 2016-05-23T12:01:34.411699-05:00 c1-0c2s1n2 xtconsole 31798
p0-20160523t115109 [console@34] cray-ansible: /etc/ansible/site.yaml completed
in init - FAILED.
```

2. Access the debug shell with `xtcon` from the SMW.

```
smw# xtcon c1-0c2s1n2
```

```
nid00035#
```

3. Inspect Ansible logs on the node in `/var/opt/cray/log/ansible`, make a configuration change in the config set, or do some other corrective action. Exiting from the debug shell causes `cray-ansible` to run again in `init`.

Examine System Dumps

The `xtdumpsys` command collects and analyzes information from a Cray XC system that is failing or has failed, has crashed, or is hung. The dump file includes:

- Event log data, active heartbeat probing, voltages, temperatures, health faults, in-memory console buffers, and high-speed interconnection network errors.
- Config sets from the SMW.
- Ansible logs from nodes.
- Ansible changed files log from nodes can be collected.
- NIMS logs from SMW can be collected.

Include the files that Ansible changed by using the `ansible_changed_files` `xtdumpsys` plugin.

```
xtdumpsys --plugins-include=ansible_changed_files --reason="add changed files" -
add c0-0c0s3n2
```

Include the NIMS logs from the SMW by using the `nims_logs` `xtdumpsys` plugin. The NIMS logs are written to the `nims` directory in the dump.

```
xtdumpsys --plugins-include=nims_logs --reason="include NIMS logs"
```

Log on to the Boot Node

About this task

The standard Cray configuration has a gigabit Ethernet connection between the SMW and boot node. All other nodes on the Cray system are accessible from the boot node.

Procedure

1. Log on to the SMW as `crayadm`.
2. There are two methods to log on to the boot node: `ssh` to the boot node.
   - Use `ssh`:

     ```
     crayadm@smw:~>  ssh boot
     crayadm@boot:~>
     ```

   - Open an administrator window on the SMW:

     ```
     crayadm@smw:~> xterm -ls -vb -sb -sl 2049 6&
     ```

     After the window opens, use it to `ssh` to the boot node.

Display Boot Configuration Information

Use the `xtcli` command to display the configuration information for the primary and backup boot nodes, the primary and backup SDB nodes, and the `cpio` path.

Display boot configuration information for the entire system

```
crayadm@smw:~> xtcli boot_cfg show
Network topology: class 2
=== xtcli_boot_cfg ===
[boot]: c0-0c0s0n1:ready,c0-0c0s0n1:ready
[sdb]: c1-0c0s1n1:ready
[cpio_path]: /tmp/boot/kernel.cpio_5.2.14-wGPFS
```

Display boot configuration information for one partition in a system

```
crayadm@smw:~> xtcli part_cfg show pN
```

Where pN is the partition number. p0 is always the whole system.

Update the Boot Configuration

The HSS xtcli boot_cfg command allows the administrator to specify the primary and backup boot nodes and the primary and backup SDB nodes for s0 or p0 (the entire system).

For a partitioned system, use xtcli part_cfg to manage boot configurations for partitions.

For more information, see the xtcli_boot(8) and xtcli_part(8) man pages.

For this example, update the boot configuration using the boot image /bootimagedir/bootimage, primary boot node (for example, c0-0c0s0n1), backup boot node, primary SDB node, and the backup SDB node:

```
crayadm@smw:~> xtcli boot_cfg update -b primaryboot_id,backupboot_id \
-d primarySDB_id,backupSDB_id -i /bootimagedir/bootimage
```

Display the Format of the SDB attributes Table

When the SDB boots, it reads the /etc/opt/cray/sdb/attributes file and loads it into the SDB attributes table.

To display the format of the attributes SDB table, use the mysql command:

```
crayadm@login:~> mysql -e "desc attributes;" -h sdb XTAdmin
+----------+------------------+------+-----+---------+-------+
| Field    | Type             | Null | Key | Default | Extra |
+----------+------------------+------+-----+---------+-------+
| nodeid   | int(32) unsigned | NO   | PRI | 0       |       |
| archtype | int(4) unsigned  | NO   |     | 2       |       |
| osclass  | int(4) unsigned  | NO   |     | 2       |       |
| coremask | int(4) unsigned  | NO   |     | 1       |       |
| availmem | int(32) unsigned | NO   |     | 0       |       |
| pageszl2 | int(32) unsigned | NO   |     | 12      |       |
| clockmhz | int(32) unsigned | YES  |     | NULL    |       |
| label0   | varchar(32)      | YES  |     | NULL    |       |
| label1   | varchar(32)      | YES  |     | NULL    |       |
| label2   | varchar(32)      | YES  |     | NULL    |       |
| label3   | varchar(32)      | YES  |     | NULL    |       |
| numcores | int(4) unsigned  | NO   |     | 1       |       |
| sockets  | int(4) unsigned  | NO   |     | 1       |       |
| dies     | int(4) unsigned  | NO   |     | 1       |       |
+----------+------------------+------+-----+---------+-------+
```

The service database command pair xtdb2attr and xtattr2db enables the system administrator to update the attributes table in the SDB. For additional information about updating SDB tables using command pairs, see *Update SDB Tables* on page 51.

Update SDB Tables

The CLE command pairs shown enable the system administrator to update tables in the SDB. One command converts the data into an ASCII text file to edit; the other writes the data back into the database file.

Table 2. Service Database Update Commands

Get Command	Put Command	Table Accessed	Reason to Use	Default File
xtdb2proc	xtproc2db	processor	Updates the database when a node is taken out of service	./processor
xtdb2attr	xtattr2db	attributes	Updates the database when node attributes change	./attribute
xtdb2segment	xtsegment2db	segment	For nodes with multiple NUMA nodes, updates the database when attribute information about node changes	./segment
xtdb2servcmd	xtservcmd2db	service_cmd	Updates the database when characteristics of a service change	./serv_cmd
xtdb2servconfig	xtservconfig2db	service_config	Updates the database when services change	./serv_config
xtdb2etchosts	none	processor	Manages IP mapping for service nodes	none
xtdb2lustrefailover	xtlustrefailover2db	lustre_failover	Updates the database when a node's Lustre failover state changes	./lustre_failover
xtdb2lustreserv	xtlustreserv2db	lustre_service	Updates the database when a file system's failover process is changed	./lustre_serv
xtdb2filesys	xtfilesys2db	filesystem	Updates the database when a file system's status changes	./filesys
xtdb2gpus	xtgpus2db	gpus	Updates the database when	./gpus

Get Command	Put Command	Table Accessed	Reason to Use	Default File
			attributes about the accelerators change	
`xtprocadmin`	none	`processor`	Displays or sets the current value of processor flags and node attributes in the service database (SDB). The batch scheduler and ALPS are impacted by changes to these flags and attributes.	none
`xtservconfig`	none	`service_config`	Adds, removes, or modifies service configuration in the SDB `service_config` table	none

Free Up Disk Space in the btrfs File System After Removing SMW Snapshots

The btrfs file system requires some maintenance, particularly to free up disk space consumed by unneeded snapshots. When administrators encounter a `No space left on device` message, snapshots could be causing the problem. For information about this issue, see the troubleshooting file systems section of the *SUSE Storage Administration Guide* (*https://www.suse.com/documentation/sles-12/stor_admin/data/stor_admin.html*). Also refer to the snapshot management section of the *SUSE Administration Guide* (*https://www.suse.com/documentation/sles-12/book_sle_admin/data/book_sle_admin.html*).

Boot a Node or Set of Nodes Using the xtcli boot Command

To boot a specific image or load file on a given node or set of nodes, execute the HSS `xtcli boot` *`boot_type`* command, as shown in the following examples. When using a file for the boot image, the same file must be on both the SMW and the `bootroot` at the same path.

WARNING: Each system boot must be started with an `xtbootsys` session to establish a *`sessionid`*. Perform direct boot commands using the `xtcli boot` command only **after** a session has been established through `xtbootsys`.

Boot all service nodes with a specific image

For this example, the specific image is located at `/raw0`:

```
crayadm@smw:~> xtcli boot all_serv_img -i /raw0
```

Boot all compute nodes with a specific image

For this example, the specific image is located at */bootimagedir/bootimage*:

```
crayadm@smw:~> xtcli boot all_comp_img -i /bootimagedir/bootimage
```

Boot compute nodes using a load file

The following example boots all compute nodes in the system with using a load file name `CNL0`:

```
crayadm@smw:~> xtcli boot CNL0 -o compute s0
```

Increase the Boot Manager Timeout Value

On systems of 4,000 nodes or larger, the time that elapses until the boot manager receives all responses to the boot requests can be greater than the default 60-second time-out value. This is due, in large part, to the amount of other event traffic that occurs as each compute node generates its console output.

To avoid this problem, change the `boot_timeout` value in the `/opt/cray/hss/default/etc/bm.ini` file on the SMW to increase the default 60-second time-out value by 60 seconds for every 5,000 nodes; for example:

Increase the `boot_timeout` value

For systems of 5,000 to 10,000 nodes, change the `boot_timeout` line to:

```
boot_timeout 120
```

For systems of 10,000 to 15,000 nodes, change the `boot_timeout` line to:

```
boot_timeout 180
```

Reboot Controllers of a Cabinet or Blade

The `xtccreboot` command provides a means to reboot controllers. Options allow for rebooting all controllers of a specified type (cabinet or blade) or providing a list of controllers of a specified type to be rebooted.

For additional information, see the `xtccreboot(8)` man page.

Reboot cabinet controller c0-0, with verbose output

```
smw:~> xtccreboot -v -c c0-0
xtccreboot: /opt/cray-xt-pdsh/default/bin/pdsh -w "c0-0" /sbin/reboot
xtccreboot: reboot sent to specified CCs
```

Bounce Blades Repeatedly Until All Blades Succeed

About this task

IMPORTANT: This iterative `xtbounce` should typically be done in concert with an `xtbootsys` automation file where bounce and routing are turned off.

Procedure

1. Bounce the system.

```
smw:~> xtbounce s0
```

2. Bounce any blades that failed the first bounce. Repeat as necessary.

3. Execute the following command, which copies route configuration files, based on the *idlist* (such as `s0`), to the blade controllers. This avoids having old, partial route configuration files left on the blades that were bounced earlier and ensures that the links are initialized correctly.

```
smw:~> xtbounce --linkinit s0
```

4. Route and boot the system without executing `xtbounce` again. If using a `xtbootsys` automation file, specify `set data(config,xtbounce) 0`, or use the `xtbootsys --config xtbounce=0` command.

Request and Display System Routing

Use the HSS `rtr` command to request routing for the HSN, to verify current route configuration, or to display route information between nodes. Upon startup, `rtr` determines whether it is making a routing request or an information request.

For more information, see the `rtr(8)` man page.

Display routing information

The `--system-map` option to `rtr` writes the current routing information to `stdout` or to a specified file. This command can also be helpful for translating node IDs (NIDs) to physical ID names.

```
crayadm@smw:~> rtr --system-map
```

Route the entire system

The `rtr -R | --route-system` command sends a request to perform system routing. If no components are specified, the entire configuration is routed as a single routing domain based on the configuration information provided by the state manager. If a component list (*idlist*) is provided, routing is limited to the listed components. The state manager configuration further limits the routing domain to omit disabled blades, nodes, and links and empty blade slots.

```
crayadm@smw:~> rtr --route-system
```

Initiate a Network Discovery Process

Use the HSS `rtr --discover` command to initiate a network discovery process.

```
crayadm@smw:~> rtr --discover
```

The discovery process must be done on the system as a whole—it cannot be applied to individual partitions. Therefore, discovery will immediately fail if the system does not have partition `p0` enabled.

The `rtr --discover` process should be used under the following circumstances:

- During an initial install, after successful execution of `xtdiscover`
- During the installation of additional cabinets in an existing installation, after the successful execution of `xtdiscover`
- During an upgrade of optical cabling in a system, after all recabling is complete

The `rtr --discover` process is NOT required under the following circumstances:

- On any single group system at any time, even those listed above
- During a warmswap operation

See the `rtr(8) man` page for additional information.

Configure IP Routes

Prerequisites

Configuring IP routes for compute nodes is not required on a CLE system.

About this task

An `/etc/routes` file can provide route entries for compute nodes. This provides a mechanism for administrators to configure routing access from compute nodes to login and network nodes, using external IP destinations without having to traverse RSIP tunnels. Careful consideration should be given before using this capability for general purpose routing.

The `/etc/routes` file will provide a route from the compute nodes to a gateway node (login or network). However, that gateway node must provide a connection to the network of interest (via IP forwarding, NAT, or something else). These instructions do not cover providing that connection.

Use the `simple_sync` functionality to make the `/etc/routes` file available on the compute nodes.

Procedure

Configure IP routes via `simple_sync`.

The new `/etc/routes` file is examined during startup. Non-comment, non-blank lines are passed to the route add command. The empty file contains comments describing the syntax.

To make the routes file available to the compute nodes, do the following on the SMW.

a. Edit a `routes` file with the desired compute node routes in a local directory.

```
smw# vi routes
```

b. Create the directory `etc` in the desired config set directory, `/var/opt/cray/imps/config/sets/<config set>/files/roles/simple_sync/classes/compute`. This will create an `/etc` directory on the compute nodes.

```
smw# mkdir -p /var/opt/cray/imps/config/sets/p0/files/roles/simple_sync/
classes/compute/etc
```

c. Copy the `routes` files from the local directory into the newly created `etc` directory. Then, this file will be available on all of the compute nodes when they boot.

```
smw# cp -p routes /var/opt/cray/imps/config/sets/p0/files/roles/simple_sync/
classes/compute/etc
```

Shut Down the System Using the auto.xtshutdown File

The preferred method to shut down the system is to use the `xtbootsys` command with the auto shutdown file as follows:

```
crayadm@smw:~> xtbootsys -s last -a auto.xtshutdown
```

Or, for a partitioned system with partition `pN`:

```
smw:~# xtbootsys --partition pN -s last -a auto.xtshutdown
```

This method shuts down the compute nodes (which are commonly also Lustre clients), then executes `xtshutdown` on service nodes, halting the nodes and then stopping processes on the SMW. A system administrator can shut down the system using both user-defined and built-in procedures in the `auto.xtshutdown` file, which is located on the SMW in the `/opt/cray/hss/default/etc` directory.

For related procedures, see *CLE Installation and Configuration Guide*. For more information about using automation files, see the `xtbootsys(8)` man page.

The xtshutdown Command

The `xtshutdown` command executes a series of commands locally on the boot node and service nodes to shut down the system in an orderly fashion. The sequence of shutdown steps and the nodes on which to execute them are defined by the system administrator in the `/etc/opt/cray/init-service/xtshutdown.conf` file or in the file specified by the environment variable `XTSHUTDOWN_CONF`.

Root user privileges are required to run `xtshutdown`. Passwordless `ssh` must be enabled for the root user from the boot node to all service nodes.

The `xtshutdown` command uses `pdsh` to invoke commands on the selected service nodes (i.e., boot node, SDB node, a class of nodes, or a single host). A system administrator can define functions to execute when the system is shut down. Place these functions in the `/etc/opt/cray/init-service/xt_shutdown_local` file or the file defined by the `XTSHUTDOWN_LOCAL` environment variable.

Shut Down the System or Part of the System Using the xtcli shutdown Command

The HSS `xtcli shutdown` command shuts down the system or a part of the system. To shut down compute nodes, execute the `xtcli shutdown` command. Under normal circumstances, for example to successfully disconnect from Lustre, invoking the `xtcli shutdown` command attempts to gracefully shut down the specified nodes.

For information, see the `xtcli(8)` man page.

Shut down all compute nodes

```
crayadm@smw:~> xtcli shutdown compute
```

Shut down specified compute nodes

For this example, shut down only compute nodes in cabinet `c13-2`:

```
crayadm@smw:~> xtcli shutdown c13-2
```

Shut down all nodes of a system

```
crayadm@smw:~> xtcli shutdown s0
```

Shut down a partition p*N* of a system

```
crayadm@smw:~> xtcli shutdown pN
```

Force nodes to shut down (immediate halt)

When all nodes of a system must be halted immediately, use the `-f` argument; nodes will not go through their normal shutdown process. Forced shutdown occurs even if the nodes have an alert status present.

```
crayadm@smw:~> xtcli shutdown -f s0
```

After the software on the nodes is shutdown, the system administrator can halt the hardware, reboot, or power down.

Shut Down Service Nodes

Prerequisites

`Root` user privileges are required to run `xtshutdown`. Passwordless `ssh` must be enabled for the `root` user from the boot node to all service nodes.

CAUTION: The `xtshutdown` command does not shut down compute nodes. To shut down the compute and service nodes, see *Shut Down the System or Part of the System Using the xtcli shutdown Command*.

About this task

For information about shutting down service nodes, see the `xtshutdown(8)` man page.

Procedure

1. Modify the `/etc/opt/cray/init-service/xtshutdown.conf` file or the file specified by the `XTSHUTDOWN_CONF` environment variable to define the sequence of shutdown steps and the nodes on which to execute them. The `/etc/opt/cray/init-service/xtshutdown.conf` file resides on the boot node.

2. If desired, define functions to execute when the system is shut down. Place these functions in the `/etc/opt/cray/init-service/xt_shutdown_local` file or the file defined by the `XTSHUTDOWN_LOCAL` environment variable.

3. Execute `xtshutdown`.

   ```
   boot:~ # xtshutdown
   ```

   After the software on the nodes is shutdown, the administrator can halt the hardware, reboot, or power down.

Stop System Components

When a system administrator removes, stops, or powers down components, any applications and compute processes that are running on those components are lost.

Reserve a Component

To allow applications and compute processes to complete before stopping components, use the HSS `xtcli set_reserve` *`idlist`* command to prevent the selected nodes from accepting new jobs.

A node running CNL and using ALPS is considered to be down by ALPS after it is reserved using the `xtcli set_reserve` command. The output from `apstat` will show the node as down (DN), even though there may be an application running on that node. This DN designation indicates that no other work will be placed on the node after the currently running application has terminated.

For more information, see the `xtcli_set(8)` man page.

Reserve a component

```
crayadm@smw:~> xtcli set_reserve idlist
```

Power Down Blades or Cabinets

WARNING: Power down the cabinets with software commands. Tripping the circuit breakers may result in damage to system components.

WARNING: Before powering down a blade or a cabinet, ensure the operating system is not running.

The `xtcli power down` command powers down the specified cabinet and/or blades within the specified partition, chassis or list of blades. Cabinets must be in the `READY` state to receive power commands.

When a request is made to power down a blade consisting of Intel® Xeon® processor Scalable Family nodes or a Cabinet containing processor blades of this type, the nodes are powered off into the G3 state (full power off) prior to the Cabinet controller removing power from the blade. See *System Component States* on page 60.

The `xtcli power down` command has the following form, where *`physIDlist`* is a comma-separated list of cabinets, blades, or nodes present on the system.

```
xtcli power down physIDlist
```

The `xtcli power force_down` and `xtcli power down_slot` commands are aliases for the `xtcli power down` command. For information about disabling and enabling components, see *Disable Hardware Components*, and *Enable Hardware Components*, respectively.

WARNING: Although a blade is powered off, the HSS in the cabinet is live and has power.

For information about powering down a component, see the `xtcli_power(8)` man page.

Power down a specified blade

For this example, power down a blade with the ID `c0-0c0s7`:

```
crayadm@smw:~> xtcli power down c0-0c0s7
```

Power Down a Specific Node

The `xtcli power down_node` command powers down the specified node and/or nodes within a specified partition, chassis, list of blades, or list of nodes. When specifying a specific node or list of nodes, all node types are powered down to the G3 state except for Intel® Xeon® processor Scalable Family nodes, which are powered down to the S5 state (soft off). These nodes can be powered down to the G3 state using one of the following methods:

- Issue the `xtcli power down_node` command with the `--with-si` flag.
- Power down the blade that the Intel® Xeon® processor Scalable Family nodes reside on. Blades must be in the READY state to receive power commands. See *System Component States* on page 60.

The `xtcli power down_node` command has the following form, where *`physIDlist`* is a comma-separated list of cabinets, blades, or nodes present on the system.

```
xtcli power down_node physIDlist
```

Power down specified nodes

In these example commands, `c0-0c0s7n0` is a Haswell node and `c0-1c1s8n2` is a Intel® Xeon® processor Scalable Family node. The following `down_node` power command does not include the `--with-si` flag.

```
crayadm@smw:~> xtcli power down_node c0-0c0s7n0,c0-1c1s8n2
```

HSS reports both nodes as being in the `off` state. The state of `c0-0c0s7n0` is G3, and the state of `c0-1c1s8n2` is S5.

The next example uses the `--with-si` flag to power down the same two nodes.

```
crayadm@smw:~> xtcli power down_node --with-si c0-0c0s7n0,c0-1c1s8n2
```

HSS reports both nodes as being in the `off` state. Both nodes are in the G3 state. See the `xtcli_power(8)` man page for more information.

Halt Selected Nodes

Use the HSS `xtcli halt` command to halt selected nodes. For more information, see the `xtcli(8)` man page.

Halt a node

For this example, halt node `157`:

```
crayadm@smw:~> xtcli halt 157
```

Restart a Blade or Cabinet

IMPORTANT: Change the state of the hardware only when the operating system is not running or is shut down.

The `xtcli power up` command powers up the specified cabinet and/or blades within the specified partition, chassis or list of blades. Cabinets must be in the `READY` state (see *System Component States* on page 60) to receive power commands. The `xtcli power up` command does not attempt to power up network mezzanine cards or nodes that are handled by the `xtbounce` command during system boot.

The `xtcli power up_slot` command is an alias for the `xtcli power up` command.

The `xtcli power up` command has the following form, where *`physIDlist`* is a comma-separated list of cabinets or blades present on the system.

```
xtcli power up physIDlist
```

For more information, see the `xtcli_power(8)` man page.

Power up blades in `c0-0c0s7`

```
crayadm@smw:~> xtcli power up c0-0c0s7
```

System Component States

Component state definitions are designated by uppercase letters. The state of `OFF` means that a component is present on the system. If the component is a blade controller, node, or ASIC, then this will also mean that the

component is powered off. If the administrator disables a component, the state shown becomes `disabled`. When the `xtcli enable` command is used to enable that component for use once again, its state switches from `disabled` to `off`. In the same manner, enabling an empty component means that its state switches from `empty` to `off`.

The state of `EMPTY` components does not change when using the `xtcli enable` or the `xtcli disable` command, unless the force option (`-f`) is used.

Disabling of a cabinet, chassis, or blade will fail if any nodes under the component are in the `ready` state, unless the force option (`-f`) is used. An error message will indicate the reason for the failure.

Table 3. State Definitions

State	Cabinet Controller	Blade Controller	Cray ASIC	CPU	Link
`OFF`	Powered off	Powered off	Powered off	Powered off	Link is down
`ON`	Powered on	Powered on	Powered on and operational	Powered on	Link is up
`HALT`	--	--	--	CPU halted	--
`STANDBY`	--	--	--	Booting was initiated	--
`READY`	Operational	Operational	Operational	Booted	Operational

Table 4. Additional State Definitions (Common to all components)

State	Description
`DISABLED`	Operator disabled this component.
`EMPTY`	Component does not exist.
`N/A`	Component cannot be accessed by the system.
`RESVD`	Reserved; new jobs are not allocated to this component.

There are two notification flags, which can occur with any state.

`WARNING` A condition of the component was detected that is outside the normal operating range but is not yet dangerous.

`ALERT` A dangerous condition or fatal error has been detected for the component.

Administrative states are hierarchal, so disabling or enabling a component has a cascading effect on that component's children. A component may not be enabled if its parent component is disabled, but a subcomponent may be disabled without affecting its parents.

Table 5. `xtcli` Commands and Valid States

`xtcli` Command	Subcommand	Cabinet Controller	Blade Controller	Node
`power`	`up`	`ON`	`OFF`	`OFF`
	`down`	`READY`	`ON`	`ON`, `HALT`, `DIAG`

xtcli Command	Subcommand	Cabinet Controller	Blade Controller	Node
	up_slot (an alias for up) down_slot (an alias for down) force_down (an alias for down)			
halt		N/A	N/A	STANDBY, READY
boot		N/A	N/A	ON, HALT

Abort Active Sessions on the HSS Boot Manager

About this task

Use the HSS `xtcli session abort` command to abort sessions in the boot manager. A session corresponds to executing a specific command such as `xtcli power up` or `xtcli boot`.

For more information about manager sessions, see the `xtcli(8)` man page.

Procedure

1. Display all running sessions in the boot manager. Only the boot manager supports multiple simultaneous sessions.

```
crayadm@smw:~> session show BM all
```

2. Abort the selected session, *session_id*.

```
crayadm@smw:~>  xtcli session abort BM session_id
```

Display and Change Software System Status

The user command `xtnodestat` provides a display of the status of nodes: how they are allocated and to what jobs. The `xtnodestat` command provides current job and node status summary information, and it provides an interface to ALPS and jobs running on CNL compute nodes. ALPS must be running in order for `xtnodestat` to report job information.

For more information, see the `xtnodestat(1)` man page.

Configure Current System Timezone

Prerequisites

Start with the XC system booted.

About this task

Changing the timezone of a system can be done with a few configuration changes and then rebooting components.

Procedure

Check current timezone

1. Check timezone on SMW.

```
smw# date
```

2. Check timezone on cabinet and blade controllers.

```
smw# xtrsh -l root -s date
```

3. Check timezone on boot node.

```
smw# ssh boot date
```

4. Check timezone on SDB node. This command works from the SMW if the SDB node is a tier1 node with an Ethernet connection to the SMW.

```
smw# ssh sdb date
```

5. Check timezone on all service nodes.

```
smw# ssh sdb pcmd -r -n ALL_SERVICE_NOT_ME "date"
```

6. Check timezone on all compute nodes.

```
smw# ssh sdb pcmd -r -n ALL_COMPUTE "date"
```

Change SMW local timezone

7. Execute this command to change the default timezone. The default timezone on the SMW is "America/Chicago".

```
smw# yast2 timezone
```

The change on the SMW will be immediate, but users will need to logout and then login again to get the new environment.

This does not change the timezone for the CLE nodes or the cabinet and blade controllers. See below to make those changes.

Change timezone in global config set

8. Set `cray_time.settings.service.data.timezone` to be the desired timezone. A list of possible timezones is available on the SMW in `/usr/share/zoneinfo/zone1970.tab`.

```
smw# cfgset update -s cray_time -m interactive global
```

9. Validate the config set.

```
smw# cfgset validate global
```

Change timezone in CLE config set

If the CLE config set has `cray_time.inherit` set to true, then the timezone and other time settings from the global config set will be inherited by the CLE config set.

If the CLE config set has `cray_time.inherit` set to false, then use the following command to change the setting.

10. Set `cray_time.settings.service.data.timezone` to be the desired timezone. A list of possible timezones is available on the SMW in `/usr/share/zoneinfo/zone1970.tab`.

```
smw# cfgset update -s cray_time -m interactive p0
```

11. Validate the config set.

```
smw# cfgset validate p0
```

Reboot for new timezone

Follow these steps to set a new timezone for all components in the SMW and CLE system after the global and CLE config sets and SMW yast2 have been updated with the new setting.

12. Reboot SMW.

a. Shutdown CLE and reboot the SMW.

```
crayadm@smw> xtbootsys -s last -a auto.xtshutdown
crayadm@adm> su - root
smw# reboot
```

b. Check that the SMW has the desired timezone setting once the SMW reboots.

```
smw# date
```

13. Power down the system.

```
smw# xtcli power down s0
```

14. Reboot the cabinet controllers

```
smw# xtccreboot -c all
xtccreboot: reboot sent to specified CCs
smw# sleep 120
smw# xtalive -l cc
```

15. Power up the system.

```
smw# xtcli power up s0
```

16. Boot CLE nodes for new timezone.

```
crayadm@smw> xtbootsys -a auto.rhine
```

17. Check current timezone.

a. Check timezone on SMW.

```
smw# date
```

b. Check timezone on cabinet and blade controllers.

```
smw# xtrsh -l root -s date
```

c. Check timezone on boot node.

```
smw# ssh boot date
```

d. Check timezone on SDB node. This command works from the SMW if the SDB node is a tier1 node with an Ethernet connection to the SMW.

```
smw# ssh sdb date
```

e. Check timezone on all service nodes.

```
smw# ssh sdb pcmd -r -n ALL_SERVICE_NOT_ME "date"
```

f. Check timezone on all compute nodes.

```
smw# ssh sdb pcmd -r -n ALL_COMPUTE "date"
```

View and Change the Status of Nodes

Use the `xtprocadmin` command on a service node to view the status of components of a booted system in the `processor` table of the SDB. The command enables the system administrator to retrieve or set the processing mode (`interactive` or `batch`) of specified nodes. The administrator can display the state (`up`, `down`, `admindown`, `route`, or `unavailable`) of the selected components, if needed. The administrator can also allocate processor slots or set nodes to become unavailable at a particular time. The node is scheduled only if the status is `up`.

When the `xtprocadmin -ks` option is used, then the option can either a normal argument (`up`, `down`, etc.), or it can have a colon in it to represent a conditional option; for example, the option of the form `up:down` means "if state was up, mark down".

For more information, see the `xtprocadmin(8)` man page.

View node characteristics

```
login:~> xtprocadmin
   NID    (HEX)    NODENAME     TYPE     STATUS        MODE
     1      0x1  c0-0c0s0n1  service         up       batch
     2      0x2  c0-0c0s0n2  service         up       batch
     5      0x5  c0-0c0s1n1  service         up       batch
     6      0x6  c0-0c0s1n2  service         up       batch
     8      0x8  c0-0c0s2n0  compute         up       batch
     9      0x9  c0-0c0s2n1  compute         up       batch
    10      0xa  c0-0c0s2n2  compute         up       batch
    11      0xb  c0-0c0s2n3  compute         up       batch
```

View all node attributes

```
login:~> xtprocadmin -A
   NID    (HEX)    NODENAME     TYPE ARCH        OS CPUS  CU AVAILMEM   PAGESZ CLOCKMHZ GPU SOCKETS DIES C/
CU                       LABEL0                          LABEL1
LABEL2                     LABEL3
     1      0x1  c0-0c0s0n1  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     2      0x2  c0-0c0s0n2  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     5      0x5  c0-0c0s1n1  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     6      0x6  c0-0c0s1n2  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     8      0x8  c0-0c0s2n0  compute   xt       CNL   32  16    65536     4096     2600   0       2    2
2
     9      0x9  c0-0c0s2n1  compute   xt       CNL   32  16    65536     4096     2600   0       2    2
2
    10      0xa  c0-0c0s2n2  compute   xt       CNL   32  16    65536     4096     2600   0       2    2
2
```

View selected attributes of selected nodes

For this example, the `-a` option lists the selected attributes to display:

```
login:~> xtprocadmin -n 8 -a arch,clockmhz,os,cores
   NID    (HEX)    NODENAME     TYPE ARCH CLOCKMHZ         OS CPUS
     8      0x8  c0-0c0s2n0  compute   xt     2600        CNL   32
```

Disable a node

For this example, the `admindown` option disables node `c0-0c0s3n1` such that it cannot be allocated:

```
crayadm@nid00004:~> xtprocadmin -n c0-0c0s3n1 -k s admindown
```

Disable all processors

```
crayadm@nid00004:~> xtprocadmin -k s admindown
```

Perform Parallel Operations on Compute Nodes

The parallel command tool (`pcmd`) facilitates execution of the same commands on groups of compute nodes in parallel, similar to `pdsh`. Although `pcmd` is launched from a service node, it acts on compute nodes. It allows administrators and/or, if the site deems it feasible, other users to securely execute programs in parallel on compute nodes. The user can specify on which nodes to execute the command. Alternatively, the user can specify an application ID (*`apid`*) to execute the command on all the nodes available under that *`apid`*.

An unprivileged user must execute the command targeting nodes where the user is currently running an `aprun`. A `root` user is allowed to target any compute node, regardless of whether there are jobs running there or not. In either case, if the `aprun` exits and the associated applications are killed, any commands launched by `pcmd` will also exit.

By default, `pcmd` is installed as a `root`-only tool. It must be installed as `setuid root` in order for unprivileged users to use it.

The `pcmd` command is located in the `nodehealth` module. If the `nodehealth` module is not part of the default profile, load it by specifying:

```
module load nodehealth
```

For additional information, see the `pcmd(1)` man page.

Perform Parallel Operations on Service Nodes

Use `pdsh`, the CLE parallel remote shell utility for service nodes, to issue commands to groups of nodes in parallel. The system administrator can select the nodes on which to use the command, exclude nodes from the command, and limit the time the command is allowed to execute. Only user `root` can execute the `pdsh` command. The command has the following form:

```
pdsh [options] command
```

For more information, see the `pdsh(1)` man page.

Restart the NTP service

```
boot:~ # pdsh -w 'login[1-9]' /etc/init.d/ntp restart
```

Mark a Compute Node as a Service Node

Use the `xtcli mark_node` command to mark a node in a compute blade to have a role of `service` or `compute`; `compute` is the default. It is not permitted to change the role of a node on a service blade, which always has the `service` role.

Marking a node on a compute blade as `service` or `compute` allows the administrator to load the desired boot image at boot time. Compute nodes marked as `service` can run software-based services. A request to change the role of a running node (that is, the node is in the `ready` state and the operating system is running) will be denied.

For more information, see the `xtcli(8)` man page and *Check the Status of System Components* on page 197.

Find Node Information

Translate Between Physical ID Names and Integer NIDs

To translate between physical ID names (cnames) and integer NIDs, generate a system map on the System Management Workstation (SMW) and filter the output, enter the following command:

```
crayadm@smw:~> rtr --system-map | grep cname | awk '{ print $1 }'
```

For more information, see the `rtr(8)` man page.

Find Node Information Using the `xtnid2str` Command

The `xtnid2str` command converts numeric node identification values to their physical names (cnames). This allows conversion of Node ID values, ASIC NIC address values, or ASIC ID values.

For additional information, see the `xtnid2str(8)` man page.

Find the physical ID for node 38

```
smw:~> xtnid2str 28
node id 0x26 = 'c0-0c0s1n2'
```

Find the physical ID for nodes 0, 1, 2, and 3

```
smw:~> xtnid2str 0 1 2 3
node id 0x0 = 'c0-0c0s0n0'
node id 0x1 = 'c0-0c0s0n1'
node id 0x2 = 'c0-0c0s1n0'
node id 0x3 = 'c0-0c0s1n1'
```

Find the physical IDs for Aries IDs 0-7

```
smw:~> xtnid2str -a 0-7
aries id 0x0 = 'c0-0c0s0a0'
aries id 0x1 = 'c0-0c0s1a0'
aries id 0x2 = 'c0-0c0s2a0'
aries id 0x3 = 'c0-0c0s3a0'
aries id 0x4 = 'c0-0c0s4a0'
aries id 0x5 = 'c0-0c0s5a0'
aries id 0x6 = 'c0-0c0s6a0'
aries id 0x7 = 'c0-0c0s7a0'
```

Find Node Information Using the `nid2nic` Command

The `nid2nic` command prints the *nid*-to-*nic* address mappings, *nic*-to-*nid* address mappings, and a specific *physical_location*-to-*nic* address and *nid* mappings.

For information about using the `nid2nic` command, see the `nid2nic(8)` man page.

Print the *nid*-to-*nic* address mappings for the node with NID 31

```
smw:~> nid2nic 31
NID:0x1f        NIC:0x21        c0-0c0s7n3
```

Print the *nid*-to-*nic* address mappings for the node with NID 31, but specify the NIC value in the command line

```
smw:~> nid2nic -n 0x21
NIC:0x21        NID:0x1f        c0-0c0s7n3
```

Display and Change Hardware System Status

A system administrator can execute commands that look at and change the status of the hardware.

CAUTION: Execute commands that change the status of hardware only when the operating system is shut down.

Recreate HSS Database File System After Corruption

About this task

Recreating the HSS database includes:

- Creating a new `btrfs` filesystem with a subvolume which matches the currently booted snapshot
- Updating `/etc/fstab`
- Mounting the new `btrfs` snapshot
- Starting the MySQL database
- Initializing the data in the HSS database
- Preparing snapshots of the `/var/lib/mysql` filesystem with similar names to other snapshots which the SMW might be switched to as part of reversion from a staged upgrade

Procedure

1. Stop `mysql` service.

```
smw# /etc/init.d/mysql stop
redirecting to systemctl stop mysql.service
```

2. Make a new `btrfs filesystem` on `/deva/mapper/smw_node_vg-db`.

3. Show `btrfs filesystem` on database volume.

```
smw# btrfs filesystem show /dev/mapper/smw_node_vg-db
```

4. Mount database filesystem.

```
smw# mount /dev/mapper/smw_node_vg-db /mnt
```

5. Verify the btrfs filesystem is mounted.

```
smw# mount | grep mnt
/dev/mapper/smw_node_vg-db on /mnt type btrfs
(rw,relatime,space_cache,subvolid=5,subvol=/)
```

6. List btrfs subvolume.

```
smw# btrfs subvolume list /mnt
```

7. Show btrfs subvolume.

```
smw# btrfs subvolume show /mnt
/mnt is btrfs root
```

8. Confirm `mysql` entry in `/etc/fstab` file.

```
smw# cat /etc/fstab | grep msyql
/dev/mapper/smw_node_vg-db /var/lib/mysql btrfs x-cray.managed,noauto,x-
cray.snapshot,subvol=snapshots/SMW-8.0UP02_CLE-6.0UP02.20160317c,nofail 0 0
```

9. Confirm same device is mounted on `/mnt` as was in `/etc/fstab` file.

```
smw# mount | grep mnt
/dev/mapper/smw_node_vg-db on /mnt type btrfs
(rw,relatime,space_cache,subvolid=5,subvol=/)
```

10. Create subvolume.

```
smw# btrfs sub create /mnt/snapshots
Create subvolume '/mnt/snapshots'
```

11. Create snapshot.

```
smw# btrfs sub snap /mnt /mnt/snapshots/SMW-8.0UP02_CLE-6.0UP02.20160317c
Create a snapshot of '/mnt' in '/mnt/snapshots/
SMW-8.0UP02_CLE-6.0UP02.20160317c'
```

12. Unmount temporary mount point of `/mnt`.

```
smw# umount /mnt
```

13. Mount path from `/etc/fstab`.

```
smw# mount /var/lib/mysql
```

14. Confirm that the new database filesystem was mounted.

```
smw# df | grep mysql
/dev/mapper/smw_node_vg-db   10485760     16960   8359680   1% /var/lib/mysql
```

15. Start `mysql` service.

```
smw# /etc/init.d/mysql start
redirecting to systemctl start mysql.service
```

16. Create new password.

```
smw# mysqladmin -uroot password -p
```

    a. Enter password.

```
Enter password: oldpassword
```

    b. Enter new password.

```
New password: newpassword
```

    c. Confirm new password.

```
Confirm new password: newpassword
```

17. Initialize the HSS datastore.

```
smw# hssds_init
********** hssds_init started **********Command line: hssds_init

Cray HSS Datastore Setup Application [1.0]

Please enter your MySQL root password: newpassword
hssds_init: SUCCESSFULLY initialized HSS Data Store.
hssds_init: Restarting HSS Daemons...
hssds_init: Restarting service: rsms
hssds_init: Restart successful.
********** hssds_init finished **********
```

18. List directory contents to confirm data in `mysql` directory.

```
mars1-smw# ls /var/lib/mysql
aria_log.00000001  hssds        ib_logfile1  mars1-smw.err
mysql               performance_schema  test
aria_log_control   ib_logfile0  ibdata1      multi-master.info
mysql_upgrade_info  snapshots
```

19. Show `btrfs` subvolume.

```
mars1-smw# btrfs sub show /var/lib/mysql
/var/lib/mysql
        Name:                   SMW-8.0UP02_CLE-6.0UP02.20160317c
        uuid:                   de9c765c-f212-5e4b-82ba-346305ee274d
        Parent uuid:            -
        Creation time:          2016-03-24 10:25:03
        Object ID:              258
        Generation (Gen):       16
        Gen at creation:        11
        Parent:                 257
        Top Level:              257
        Flags:                  -
        Snapshot(s):
```

The above steps only make one snapshot. If `snaputil` is used to switch to other snapshots, there will be a failure at that point. To address this—once the filesytem has been recreated with valid contents for the currently booted snapshot—the following commands should be created to make snapshots of this state with names matching the other snapshots which might be booted.

20. List snapshots.

```
mars1-smw# snaputil list
Status    Name                                                  Size (MB
unshared)  Created
--------  ----------------------------------------------------
-------------------  -------------------
          @
14213.1   2015-12-04 10:31:24
          SLES12
92.25  2015-12-07 08:19:56
          SLES12_pristine
92.25  2015-12-07 08:20:11
          SMW-8.0UP02_CLE-6.0UP02.20160302
6.28  2016-03-02 06:36:19
```

```
          SMW-8.0UP02_CLE-6.0UP02.20160302.save1.postinstall
48.09  2016-03-02 08:22:56
          SMW-8.0UP02_CLE-6.0UP02.20160303
789.27  2016-03-03 06:36:31
          SMW-8.0UP02_CLE-6.0UP02.20160316
2744.32  2016-03-16 06:22:56
          SMW-8.0UP02_CLE-6.0UP02.20160316.save1.postinstall
49.54  2016-03-16 07:16:11
          SMW-8.0UP02_CLE-6.0UP02.20160317b
37.4   2016-03-17 08:58:14
cur,def   SMW-8.0UP02_CLE-6.0UP02.20160317c
559.79  2016-03-17 10:58:50
```

21. Prepare `OLDSNAPSHOT` variable for use in later commands.

```
smw# export OLDSNAPSHOT=SMW-8.0UP02_CLE-6.0UP02.20160302
```

22. Prepare `SNAPSHOT` variable for use in later commands.

```
smw# export SNAPSHOT=SMW-8.0UP02_CLE-6.0UP02.20160317c
```

23. Make temporary mount points.

```
smw# mkdir /tmp/tmp1.$SNAPSHOT /tmp/tmp2.$SNAPSHOT
```

24. Mount the root subvolume for the `/var/lib/mysql filesystem`.

```
smw# mount -o subvolid=0 /dev/mapper/smw_node_vg-db /tmp/tmp1.$SNAPSHOT
```

25. Show current `btrfs` subvolumes.

```
smw# btrfs subvolume show /tmp/tmp1.$SNAPSHOT
```

26. Mount subvolume so snapshot can be made.

```
smw# mount -o subvol=snapshots/$SNAPSHOT /dev/mapper/smw_node_vg-db /tmp/
tmp2.$SNAPSHOT
```

27. Create a snapshot of the subvolume.

```
smw# btrfs subvolume snapshot /tmp/tmp2.$SNAPSHOT /tmp/tmp1.$SNAPSHOT/snapshots/
$OLDSNAPSHOT
Create a snapshot of '/tmp/tmp2.SMW-8.0UP02_CLE-6.0UP02.20160317c' in '/tmp/
tmp1.SMW-8.0UP02_CLE-6.0UP02.20160317c/snapshots/
SMW-8.0UP02_CLE-6.0UP02.20160316.save2.devint
```

28. Show subvolume.

```
smw# btrfs subvolume show /tmp/tmp1.$SNAPSHOT/snapshots/$SNAPSHOT
/tmp/tmp1.SMW-8.0UP02_CLE-6.0UP02.20160317c/snapshots/
SMW-8.0UP02_CLE-6.0UP02.20160317c
        Name:                   SMW-8.0UP02_CLE-6.0UP02.20160317c
        uuid:                   de9c765c-f212-5e4b-82ba-346305ee274d
        Parent uuid:            -
        Creation time:          2016-03-24 10:25:03
        Object ID:              258
        Generation (Gen):       45
```

```
        Gen at creation:        11
        Parent:                 257
        Top Level:              257
        Flags:                  -
        Snapshot(s):
                                SMW-8.0UP02_CLE-6.0UP02.20160302
```

29. Unmount temporary mounts.

```
smw# umount /tmp/tmp1.$SNAPSHOT /tmp/tmp2.$SNAPSHOT
```

Dynamic Fan Speed Control

Effective with SMW version 8.0.UP04, the HSS cooling system for liquid-cooled XC and XC+ cabinets supports dynamic fan speed control by row or for the entire system.

When dynamic fan speed control is not enabled the HSS cooling software operates the cabinet fans at one of 3 fan speeds, defined as `fan_speed_idle` when the blades in the cabinet are not powered on, `fan_speed_high` when a CPU or GPU is within 8 degrees of the highest temperature that it can operate at without being throttled (TJMAX), and `fan_speed_normal` at all other times.

The speed setting of `fan_speed_normal` ensures that, under normal operation, the temperature of the CPU/GPU dies are maintained below the hot spot detection threshold. If the cooling water is at the required temperature and the temperature setpoint is set appropriately, no hot spot should be detected, as this setting is expected to cover the worst case. Typically, die temperatures on a production system fluctuate but are below the throttle threshold most of the time. Setting fan speed to a constant `fan_speed_normal` is unnecessary and can consume more energy than is needed to properly cool the system.

When the dynamic fan speed feature is enabled, the cabinets self-regulate their fan speed based upon observed CPU and/or GPU temperatures. Each cabinet in a row runs its fans at the same speed, based on the highest CPU or GPU temperature sensor reading from all of the blades in all cabinets within the row. The frequency with which fan speeds change in response to temperature sensor readings varies depending on the type of jobs running on the system, and is bounded by two pre-existing `ini` file variables:

- `fan_speed_step_up_delay` This variable controls how fast the system will switch to a higher speed in a fan speed table if die temperatures are increasing. The default is 20 seconds.
- `fan_speed_step_down_delay` This variable controls how fast the system will switch to a lower fan speed if die temperatures are decreasing.The default is 300 seconds.

  IMPORTANT: Cray recommends that these and other cooling variables related to dynamic fan speeds in the initialization files be kept at their default values. The exception is `fan_auto_speed_enable`, which enables dynamic fan speed control.

Enabling dynamic fan speed control does not supercede CPU hot spot detection and control. When a hot spot is detected, the cabinet fans in a row will still switch to the `fan_speed_high` setting and remain at that setting until the hot spot is cleared. Similarly, if the blades are powered down, the fans will run at the `fan_speed_idle` setting.

Enable Dynamic Fan Speed Control

Prerequisites

Dynamic fan speed has not enabled at the system level or on a specified row within the system.

Procedure

1. Edit the system-level (`hss.ini`) file or a row-level (`hss_rN.ini`) file in the `/opt/tftpboot/ccrd` directory to set the `fan_auto_speed_enable` variable to 1.

   Setting fan speeds dynamically on systems with mixed blower types within the same row is not supported. On systems with both STD and HP blowers in separate rows, fan speed settings must be done via row-specific `ini` files.

   ```
   fan_auto_speed_enable=1
   ```

2. If the system is running, reload the `ini` file or files.

   ```
   crayadm@smw>xtccr load_ini
   ```

3. The cooling software on each blade will automatically generate fan speed tables based on the CPUs and/or GPUs that are on the blade. To view the current fan speed table run the following command on the SMW:

   ```
   crayadm@smw>xtdaemonconfig --daemon ccrd|grep _table
   c0-0c0s7: fan_auto_speed_table_cpu=-:92:2750|91:87:2600|86:82:2450|81:77:2300
   |76:72:2150|71:-:*2000
   c0-0c0s7: fan_auto_speed_table_gpu=-:80:2750|79:75:2600|74:70:2450|69:65:2300
   |64:60:2150|59:-:*2000
   ```

   The above fan speed tables were generated for both the CPUs and the GPUs on blade `c0-0c0s7`. Each set of values between the | symbol gives the temperature range in degrees C and the corresponding fan speed in RPMs. For example, On the CPUs, a fan speed of 2750 RPMs is specified for component temperatures of 92 C and above, and a fan speed of 2600 RPMs is specified for component temperatures between 91 C and 87 C.

Configure and Validate Dynamic Cooling Control Variables

Under normal circumstances, administrators need only set the `fan_auto_speed_enable` to 1 to enable dynamic fan speed control. All other dynamic fan speed related variables should be left at their default settings.

In particular, adjusting the `fan_auto_speeds` variable is not recommended as the automatically generated fan speed tables will always be correct for the type of hardware on each blade.

The following settings are described here for use in special situations where the default values are not adequate.

⚠ **CAUTION:** It is recommended that these settings (other than `fan_auto_speed_enable`) be changed only in consultation with Cray service personnel.

`fan_auto_speed_enable`

Enables or disables dynamic fan speed control.

```
fan_auto_speed_enable=0 # disable
fan_auto_speed_enable=1 # enable
```

fan_auto_speed_temp_step

Specifies the change in temperature (in C) that would be required for the fan speed to change. The default value for `fan_auto_speed_temp_step` is 5C. The allowed range is 5-12C. This variable applies to both specified and auto-generated fan speed tables.

Example:

```
fan_auto_speed_temp_step=5
```

fan_auto_speed_rpm_step

Specifies the RPM step between fan speeds for auto-generated fan speeds. It is also used to compute the value of the highest allowed RPM in a auto-generated fan speed table (`fan_auto_speed_high - fan_auto_speed_rpm_step`). The default value for `fan_auto_speed_rpm_step` is 150. The allowed range is 150-300. The value of `fan_auto_speed_rpm_step` does not limit user-specified fan speeds.

Example:

```
fan_auto_speed_rpm_step=150
```

fan_auto_speed_high

Specifies the highest fan speed that can be used within a fan speed table, whether the table is user-specified or auto-generated. The default value of `fan_auto_speed_high` in auto-generated fan speed tables is the value of `fan_speed_normal`. The potential range of values for this variable are `>= fan_speed_normal` and `<= fan_speed_high`.

Example:

```
fan_auto_speed_high=3100
```

fan_auto_high_temp_offset

Specifies the offset from the highest temperature that a CPU or GPU can operate at without being throttled (TJMAX), that corresponds to the highest fan speed in a fan speed table. The default value of `fan_auto_high_temp_offset` is 10. The potential range of values for this variable are `>= 0` and `<= 20`. For example, if `fan_auto_speed_high` is not set and `fan_auto_high_temp_offset` is set if a component has a TJMAX of 100, then the highest fan speed in the fan speed table will be equal to `fan_speed_normal`, and the corresponding temperature for that fan speed will be at `>= 90`. Example:

```
fan_auto_high_temp_offset=10
```

fan_auto_speeds

Specifies a fan speed table choice. A minimum of 3 and a maximum of 15 fan speeds can be specified. If duplicate values are specified, or any specified fan speeds fail validation checks, then all specified fan speeds are ignored.

The minimum fan speed value is `fan_auto_speed_min` and the maximum value is `fan_auto_speed_high`. Each fan speed is separated by a vertical bar (|). Example:

```
fan_auto_speeds=3100|2800|2500|2200
```

In this case, if the default values for `fan_auto_temp_step` and `fan_auto_high_temp_offset` are used and TJMAX for a component were to be 100C, then the fan speed table lools like:

```
-:90:3100|89:85:2800|84:80:2500|79:-:2200
```

fan_speed_step_up_delay

Specifies the amount of time before the system switches to a higher speed in a fan speed table when die temperatures are increasing. The default is 20 seconds.

fan_speed_step_down_delay

Specifies the amount of time before the system switches to a lower fan speed when die temperatures are decreasing.The default is 300 seconds.

INI File Validation

If the dynamic fan speed variables have been changed from their default values, it's important to validate the `.ini` files, prior to loading them onto the controllers. Use the `xtccr --validate` command to do this.

```
crayadm@smw> xtccr --validate=filename
```

Some of the variables defined in the cooling `.ini` files may be fully validated in this fashion, whereas other variables may only be provisionally validated, as information specific to each cabinet is required to fully validate the value of a variable.

Setting fan speeds dynamically via `xtccr` on systems with mixed blower types within the same row is not supported. On systems with both STD and HP blowers in separate rows, fan speed settings must be done by means of row-specific `.ini` files.

For example, the value of `fan_speed_high` can only be validated provisionally because knowledge of the type of fans installed within a cabinet (STD or HP) is required to fully validate the value.

Disable Hardware Components

If links, nodes, or Cray ASICs have hardware problems, the system administrator can direct the system to ignore the components with the `xtcli disable` command.

By default, when disabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

The `xtcli disable` command has the following form, where *`idlist`* is a comma-separated list of components (in cname format) that the system is to ignore. The system disregards these links or nodes.

```
xtcli disable [{-t type [-a] } | -n] [-f] idlist
```

IMPORTANT: The `-n` option with the `xtcli disable` command must be used carefully because this may create invalid system state configurations.

Disabling of a cabinet, chassis, or blade will fail if any nodes under the component are in the `ready` state, unless the force option (`-f`) is used. An error message will indicate the reason for the failure.

Disabling of a node in the `ready` state will fail, unless the force option (`-f`) is used. An error message will indicate the reason for the failure.

The state of `empty` components will not change when using the `disable` command, unless the force option (`-f`) is used.

For detailed information about using the `xtcli disable` command, see the `xtcli(8)` man page.

Disable the Aries ASIC `c0-0c1s3a0`

1. Determine that the ASIC is in the `OFF` state.

```
crayadm@smw:~> xtcli status -t aries c0-0c1s3a0
```

2. If the ASIC is not in the `OFF` state, power down the blade that contains the ASIC.

```
crayadm@smw:~> xtcli power down c0-0c1s3
```

3. Disable the ASIC.

```
crayadm@smw:~> xtcli disable c0-0c1s3a0
```

4. Power up the blade that contains the ASIC.

```
crayadm@smw:~> xtcli power up c0-0c1s3
```

Enable Hardware Components

If links, nodes, or Cray ASICs that have been disabled are later fixed, the system administrator can add them back to the system with the `xtcli enable` command.

The `xtcli enable` command has the following form, where *`idlist`* is a comma-separated list of components (in cname format) for the system to recognize.

```
xtcli enable [{-t type [-a] } | -n] [-f] idlist
```

By default, when disabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

> **IMPORTANT:** The `-n` option with the `xtcli disable` command must be used carefully because this may create invalid system state configurations.

The state of `empty` components does not change when using the `xtcli enable` command, unless the force option (`-f`) is used.

The state of `off` means that a component is present on the system. If the component is a blade controller, node, or ASIC, then this will also mean that the component is powered off. If the administrator disables a component, the state shown becomes `disabled`. When the `xtcli enable` command is used to enable that component for use once again, its state switches from `disabled` to `off`. In the same manner, enabling an empty component means that its state switches from `empty` to `off`.

For more information, see the `xtcli(8)` man page.

Check Current State of Compute Node SSDs

Prerequisites

This procedure is intended only for XC systems that have compute nodes with SSDs, such as DataWarp SSDs or Intel® Xeon Phi™ "Knights Landing" processors.

About this task

Use this command after an initial installation, SSD hardware change, or system update. Cray also recommends running `xtcheckssd` periodically (daily/weekly).

Procedure

Run `xtcheckssd` to ensure that SMW databases have the current state of compute node SSDs.

```
root@login# pcmd -r -n ALL_COMPUTE "/opt/cray/ssd/bin/xtcheckssd"
```

Set Hardware Components to `EMPTY`

Use the `xtcli set_empty` command to set a selected component to the `EMPTY` state. HSS managers and the `xtcli` command ignore empty or disabled components.

Setting a selected component to the `EMPTY` state is typically done when a component, usually a blade, is physically removed. By setting it to `EMPTY`, the system ignores it and routes around it.

By default, when disabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

> **IMPORTANT:** The `-n` option with the `xtcli disable` command must be used carefully because this may create invalid system state configurations.

For more information, see the `xtcli(8)` man page.

Set a blade to the `EMPTY` state

```
crayadm@smw:~> xtcli set_empty -a c0-0c1s7
```

Lock Hardware Components

Components are automatically locked when a command that can change their state is running. As the command is started, the state manager locks these components so that nothing else can affect their state while the command executes. When the manager is finished with the command, it unlocks the components.

Use the HSS `xtcli lock` command to lock components. Locking a component prints out the state manager session ID.

For more information, see the `xtcli(8)` man page.

Lock cabinet `c0-0`

```
crayadm@smw:~> xtcli lock -l c0-0
```

Show all session (lock) data

```
crayadm@smw:~> xtcli lock show
```

Unlock Hardware Components

Use the HSS `xtcli lock` command to unlock components. This command is useful when an HSS manager fails to unlock some set of components.

The system administrator can manually check for locks with the `xtcli lock show` command and then unlock them. Unlocking a component does not print out the state manager session ID. The `-u` option must be used to unlock a component as follows:

```
crayadm@smw:~> xtcli lock -u lock_number
```

Where *`lock_number`* is the value given when initiating the lock; it is also indicated in the `xtcli lock show` query. Unlocking does nothing to the state of the component other than to release locks associated with it.

HSS daemons cannot affect components that are locked by a different session.

Over-provision an Intel P3608 SSD

Prerequisites

- A Cray XC series system with one or more Intel P3608 SSD cards installed
- Ability to log in as `root`

About this task

This procedure is only valid for Intel P3608 SSDs. The examples provided are based on the 4TB drives, but this procedure also works for the 1.6TB drives.

WARNING: This procedure destroys any existing data on the SSDs.

Over-provisioning determines the size of the device available to the Logical Volume Manager (LVM) commands and needs to occur prior to executing any LVM commands. Typically, over-provisioning is done when the SSD cards are first installed.

> **TIP:** Throughout these procedures, units of bytes are described using the binary prefixes defined by the International Electrotechnical Commission (IEC). For further information, see *Prefixes for Binary and Decimal Multiples*.

Procedure

1. Log on to an Intel P3608 SSD-endowed node as `root`, then determine the SSD model number.

   ```
   ssd# module load linux-nvme-ctl
   ssd# nvme id-ctrl /dev/nvme0 |grep mn
   mn      : INTEL SSDPECME040T4Y
   ```

2. Shut down the DataWarp manager daemon (`dwmd`).

   ```
   ssd# systemctl stop dwmd
   ```

3. Remove any existing configuration.

> **TIP:** Numerous methods exist for creating configurations on an SSD; these instructions may not capture all possible cleanup techniques.

a. Unmount file systems (if any).

```
nid00350# df
boot:/home                    20961280     11352064     9609216  55% /home
tmp                        61504671488    624927640 57802802440   2% /scratch
nid00350# umount -f /scratch
```

b. Remove logical volumes (if any).

```
nid00350# lvdisplay
  --- Logical volume ---
  LV Path                /dev/dwcache/s98i94f104o0
  LV Name                s98i94f104o0
  VG Name                dwcache
  LV UUID                910tio-RJXq-puYV-s3UL-yDM1-RoQl-HugeTM
  LV Write Access        read/write
  LV Creation host, time nid00350, 2017-02-22 13:29:11 -0500
  LV Status              available
  # open                 0
  LV Size                3.64 TiB
  Current LE             953864
  Segments               2
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     1024
  Block device           253:0

nid00350# lvremove /dev/dwcache
```

c. Remove volume groups (if any).

```
nid00350# vgs
  VG       #PV #LV #SN Attr   VSize VFree
  dwcache    4   0   0 wz--n- 7.28t 7.28t
nid00350# vgremove dwcache
  Volume group "dwcache" successfully removed
```

d. Remove physical volumes (if any).

```
nid00350# pvs
PV           VG        Fmt Attr PSize PFree
/dev/nvme0n1           lvm2 a--  1.82t 1.82t
/dev/nvme1n1           lvm2 a--  1.82t 1.82t
/dev/nvme2n1           lvm2 a--  1.82t 1.82t
/dev/nvme3n1           lvm2 a--  1.82t 1.82t

nid00350# pvremove /dev/nvme0n1 /dev/nvme1n1 /dev/nvme2n1 /dev/nvme3n1
  Labels on physical volume "/dev/nvme0n1" successfully wiped
  Labels on physical volume "/dev/nvme1n1" successfully wiped
  Labels on physical volume "/dev/nvme2n1" successfully wiped
  Labels on physical volume "/dev/nvme3n1" successfully wiped
```

e. Clear partitions for each device removed in the previous step (if any).

> **WARNING:** This operation destroys any existing data on an SSD. Back up any existing data before proceeding.

```
nid00350# dd if=/dev/zero of=phys_vol bs=512 count=1
```

```
nid00350# dd if=/dev/zero of=/dev/nvme0n1 bs=512 count=1
nid00350# dd if=/dev/zero of=/dev/nvme1n1 bs=512 count=1
nid00350# dd if=/dev/zero of=/dev/nvme2n1 bs=512 count=1
nid00350# dd if=/dev/zero of=/dev/nvme3n1 bs=512 count=1
```

4. Reconfigure the device based on the model number determined in step 1 on page 79 and the corresponding over-provision value from the following table.

Table 6. Over-provision values for supported Intel P3608 models

Model Number	Size (TB)	Over-provision Value (bytes)	HEX
SSDPECME016T4Y	1.6	1250259487	0x4a85721f
SSDPECME040T4	4.0	3125623327	0xba4d3a1f
SSDPECME040T4Y	4.0	3125623327	0xba4d3a1f

```
nid00350# nvme set-feature device -n 1 -f 0XC1 -v op_value
set-feature:193(Unknown), value:00000000
```

For the remainder of this procedure, the examples assume 4TB SSDs; values will be different for 1.6TB SSDs.

```
nid00350# nvme set-feature /dev/nvme0 -n 1 -f 0XC1 -v 3125623327
set-feature:193(Unknown), value:00000000
nid00350# nvme set-feature /dev/nvme1 -n 1 -f 0XC1 -v 3125623327
set-feature:193(Unknown), value:00000000
nid00350# nvme set-feature /dev/nvme2 -n 1 -f 0XC1 -v 3125623327
set-feature:193(Unknown), value:00000000
nid00350# nvme set-feature /dev/nvme3 -n 1 -f 0XC1 -v 3125623327
set-feature:193(Unknown), value:00000000
```

5. Confirm the change based on the SSD model number and values in *Over-provision values for supported Intel P3608 models* on page 81. Note that 0xba4d3a1f = 3125623327.

```
nid00350# nvme get-feature device -n 1 -f 0XC1 --sel=0
get-feature:193(Unknown), value:0xba4d3a1f
```

```
nid00350# nvme get-feature /dev/nvme0 -n 1 -f 0XC1 --sel=0
get-feature:193(Unknown), value:0xba4d3a1f
nid00350# nvme get-feature /dev/nvme1 -n 1 -f 0XC1 --sel=0
get-feature:193(Unknown), value:0xba4d3a1f
nid00350# nvme get-feature /dev/nvme2 -n 1 -f 0XC1 --sel=0
get-feature:193(Unknown), value:0xba4d3a1f
nid00350# nvme get-feature /dev/nvme3 -n 1 -f 0XC1 --sel=0
get-feature:193(Unknown), value:0xba4d3a1f
```

6. Return to the SMW, and warm boot the DataWarp node.

```
crayadm@smw> xtnmi cname
crayadm@smw> sleep 60
crayadm@smw> xtbootsys --reboot -r "warmboot for Intel SSD node" cname
```

7. Log in to the Intel P3608 SSD-endowed node as `root`, and confirm that `SIZE` = `1600319143936` bytes for all volumes.

```
nid00350# lsblk -b
NAME    MAJ:MIN RM          SIZE RO TYPE MOUNTPOINT
loop0     7:0    0        196608 0  loop /var/opt/cray/imps-distribution/squash/
loop1     7:1    0         65536 0  loop /var/opt/cray/imps-distribution/squash/
nvme0n1 259:0    0 1600319143936 0  disk
nvme1n1 259:1    0 1600319143936 0  disk
nvme2n1 259:2    0 1600319143936 0  disk
nvme3n1 259:3    0 1600319143936 0  disk
```

   Contact Cray service personnel if `SIZE` is incorrect.

xtbounce Error Message Indicates Cabinet Controller and Its Blade Controllers Not in Sync

During the `gather_cab_pwr_states` phase of `xtbounce`, if the HSS software on a cabinet controller and any of its blade controllers is out of sync, error messages such as the following will be printed during the `xtbounce`.

```
***** gather_cab_pwr_states *****
18:28:42 - Beginning to wait for response(s)

ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
```

If this occurs, it indicates that the blade controller software is at a different revision than the cabinet controller software. `xtbounce` will print a list of cabinets for which this error has occurred. The message will be similar to the following:

```
ERROR: power state check error on 2 cabinet(s)
WARNING: unable to find c0-0 in err_cablist
WARNING: unable to find c0-2 in err_cablist
```

This error is an indication that when the HSS software was previously updated, the cabinet controllers and the blade controllers were not updated to the same version.

To correct this error, cancel out of `xtbounce` (with **Ctrl-C**), wait approximately five minutes for the `xtbounce` related activities on the blade controllers to finish, then reboot the cabinet controller(s) and their associated blade controllers to get the HSS software synchronized. Following this, the `xtbounce` may be executed once again.

Power-cycle a Component to Handle Bus Errors

About this task

Bus errors are caused by machine-check exceptions. If a bus error occurs, try power-cycling the component.

Procedure

1. Power down the components. The *`physIDlist`* is a comma-separated list of components present on the system.

```
crayadm@smw:~> xtcli power down physIDlist
```

2. Power up the components.

```
crayadm@smw:~> xtcli power up physIDlist
```

When a Component Fails

Components that fail are replaced as field replaceable units (FRUs). FRUs include compute blade components, service blade components, and power and cooling components.

When a field replaceable unit (FRU) problem arises, contact a Customer Service Representative to schedule a repair.

Dump and Reboot Nodes Automatically

The SMW daemon `dumpd` initiates automatic dump and reboot of nodes when requested by the Node Health Checker (NHC).

CAUTION: The `dumpd` daemon is invoked automatically by `xtbootsys` on system (or partition) boot. In most cases, system administrators do not need to use this daemon directly.

A system administrator can set global variables in the `/etc/opt/cray/nodehealth/nodehealth.conf` configuration file to control the interaction of NHC and `dumpd`. For more information about NHC and the `nodehealth.conf` configuration file, see *Configure the Node Health Checker (NHC)*.

Variables can also be set in the `/etc/opt/cray-xt-dumpd/dumpd.conf` configuration file on the SMW to control how `dumpd` behaves on the system.

Each CLE release package also includes an example `dumpd` configuration file, `/etc/opt/cray-xt-dumpd/dumpd.conf.example`. The `dumpd.conf.example` file is a copy of the `/etc/opt/cray-xt-dumpd/dumpd.conf` file provided for an initial installation.

> **IMPORTANT:** The `/etc/opt/cray-xt-dumpd/dumpd.conf` file is not overwritten during a CLE upgrade if the file already exists. This preserves the site-specific modifications previously made to the file. Cray recommends comparing the site's `/etc/opt/cray-xt-dumpd/dumpd.conf` file content with the `/etc/opt/cray-xt-dumpd/dumpd.conf.example` file provided with each release to identify any changes and then update the site's `/etc/opt/cray-xt-dumpd/dumpd.conf` file accordingly.
>
> If the `/etc/opt/cray-xt-dumpd/dumpd.conf` file does not exist, then the `/etc/opt/cray-xt-dumpd/dumpd.conf.example` file is copied to the `/etc/opt/cray-xt-dumpd/dumpd.conf` file.

The CLE installation and upgrade processes automatically install `dumpd` software, but it must be explicitly enabled.

Collect Debug Information From Hung Nodes Using the xtnmi Command

CAUTION: This is not a harmless tool to use to repeatedly get information from a node at various times; only use this command when debugging data from nodes that are in trouble is needed. The `xtnmi`

command output may be used to determine problems such as a core hang. `xtnmi` will stop a running node. It is best used when a node is not running correctly and debugging information is needed, or to stop a node that is running incorrectly.

The sole purpose of the `xtnmi` command is to collect debug information from unresponsive nodes. As soon as that debug information is displayed to the console, the node panics.

For additional information, see the `xtnmi(8)` man page.

Modify BIOS Parameters

There are a few, rare circumstances where it may be necessary to modify BIOS parameters, for example, in order to troubleshoot a problem, or if there is a need to test a new BIOS version on a small set of nodes before implementing the change across an entire system.

The `xtbiosconf` command allows administrators to specify BIOS parameters at the node, blade, chassis, or cabinet level. BIOS parameters can be associated with a BIOS revision, numeric parameter offset or parameter name, and target nodes. BIOS revision wildcards are supported. The BIOS parameter data is saved in a database on the SMW, and made available automatically to blade controllers via the ERFS file system. In most cases a cold reboot of the affected nodes is needed to apply the new settings.

CAUTION: *Do not* attempt to use this command except under guidance by Cray support personnel, who will provide all the steps for shutting down the nodes, changing the settings, and bringing the nodes back up. Improper use of this command can damage a system.

The following command displays the current BIOS Parameter settings for the entire system:

```
smw~> xtbiosconf --show s0
==============|======|==================================
              | BIOS | BIOS
Node          | REV  | Parameter
==============|======|==================================
c0-1c0s0n1    | 4030 | numlock=1
c0-1c0s0n1    | 4030 | acpiauto=0
==============|======|==================================
c0-1c0s0n2    | 4030 | numlock=1
c0-1c0s0n2    | 4030 | acpiauto=0
==============|======|==================================
```

For more information see the `xtbiosconf` man page.

Increase File System Size

About this task

When a `btrfs` or `xfs` file system on the boot RAID needs to be increased, both the `cray_bootraid_config.yaml` file needs to be changed for the new size and the commands to grow the file system need to be done.

Procedure

1. Edit the `cray_bootraid_config.yaml` file to increase the size for the filesystem which needs to grow.

   ```
   smw# vi /var/opt/cray/imps/config/sets/global/config/cray_bootraid_config.yaml
   ```

   For example, to increase the size of the `/var/opt/cray/imps` file system on the SMW, locate the "`smwdefault`" storage set, the `smw_node_vg` volume group, and the "`home`" volume within that storage set. Change the "`fs_size`" for imps from 600 to 800.

   Increase the size of the `/home` file system on the SMW, in the "`smwdefault`" storage set, the "`smw_node_vg`" volume group, and the "`home`" volume within that storage set. Change the "`fs_size`" for imps from 50 to 100.

   ```
            -   key: smwdefault
                volume_groups:
                -   key: smw_node_vg
                    owner: smw
                    devices:
                    - /dev/disk/by-id/wwn-0x600a0980006b47b7000000e5561260a7
                    volumes:
                    -   key: home
                        description: LVM volume for user home directories on the
   SMW.
                        type: lvm
                        fs_type: xfs
                        fs_size: 50
                        fs_mount_point: /home
                        snapshot: false
                        mount_options:
                    -   key: imps
                        description: LVM Volume for storage of IMPS
   configuration.
                        type: lvm
                        fs_type: btrfs
                        fs_size: 600
                        fs_mount_point: /var/opt/cray/imps
                        snapshot: false
                        mount_options:
   ```

2. Extend an LVM volume.

   a. Extend the "`home`" volume in the "`smw_node_vg`" LVM volume from the existing size to 100GB.

      ```
      smw# lvextend -L100G /dev/mapper/smw_node_vg-home
      ```

   b. Extend the "`imps`" volume in the "`smw_node_vg`" LVM volume from the existing size to 800GB.

      ```
      smw# lvextend -L800G /dev/mapper/smw_node_vg-imps
      ```

3. Grow a `btrfs` file system.

   ```
   smw# btrfs filesystem resize max /var/opt/crayimps
   ```

4. Grow an `xfs` file system.

   ```
   smw# xfs_growfs /home
   ```

Add New Hardware to a System

About this task

Whether adding a single compute blade or a single service blade or several components in a full cabinet or several cabinets, the process is similar.

Procedure

1. Add new components to system partition.

   a. If the system is partitioned, then add the new components to the specific partition. If the system is not partitioned, then this step can be skipped.

   ```
   crayadm@smw> xtcli part_cfg show p2
   crayadm@smw> xtcli part_cfg deactivate p2
   ```

   b. Update the members of the partition with the old components and the new components.

   ```
   crayadm@smw> xtcli part_cfg update p2 -m
   c2-0c0s0,c2-0c0s1,c2-0c0s7,c0-0c0s9,c2-0c0s11,c2-0c0s13,c2-0c0s15,c2-0c0s3
   crayadm@smw> xtcli part_cfg activate p2
   ```

2. Ensure new components are not disabled and are assigned to the desired partition. If they are disabled, they will not be discovered. If they are not assigned to a partition, they will not be bounced during the `xtdiscover` process, and therefore will not be properly discovered.

   Full system:

   ```
   crayadm@smw> xtcli status s0
   ```

   Partitioned system:

   ```
   crayadm@smw> xtcli status p1
   crayadm@smw> xtcli status p2
   ```

3. Discover the new hardware.

   ```
   crayadm@smw> su -
   smw# xtdiscover
   smw# exit
   ```

   a. Run `rtr --discover` if there is a significant change modifying the routing configuration.

   Full system:

   ```
   crayadm@smw> rtr --discover
   ```

   Partitioned system:

   ```
   crayadm@smw> xtcli part_cfg deactivate p1
   crayadm@smw> xtcli part_cfg deactivate p2
   crayadm@smw> xtcli part_cfg activate p0
   crayadm@smw> rtr --discover
   crayadm@smw> xtcli part_cfg deactivate p0
   ```

```
crayadm@smw> xtcli part_cfg activate p1
crayadm@smw> xtcli part_cfg activate p2
```

b. Confirm the new components are now seen.

```
crayadm@smw> xtcli status s0
```

If the new components do not show up properly in the status output, do not continue. Power cycle the whole system, try the `xtdiscover` again. If they still are not showing, there may be a problem with the new hardware components.

4. Update firmware on new components. Check whether any firmware needs to be updated on the various controllers.

```
crayadm@smw> xtzap -r -v s0
```

If any are out of date, output like the following from the `xtzap` command will be seen and the firmware needs to be updated.

```
Individual Revision Mismatches:

Type       ID                Expected   Installed
---------- ----------------- ---------- ----------------------------------------
cc_bios    c0-0              0013       0012
bc_bios    c0-0c0s0          0013       0012
bc_bios    c0-0c0s1          0013       0012
bc_bios    c0-0c0s2          0013       0012
bc_bios    c0-0c0s3          0013       0012
```

a. Update firmware, if not all current.

CAUTION: The `xtzap` command is normally intended for use by Cray Service personnel only. Improper use of this restricted command can cause serious damage to the computer system.

If the output of `xtzap` includes a "Revision Mismatches" section, then some firmware is out of date and needs to be reflashed. To update, run `xtzap` with one or more of the options described in the next paragraph.

While the `xtzap -a` command can be used to update all components with a single command, it may be faster to use the `xtzap -blade` command when only blade types need to be updated, or the `xtzap -t` command when only a single type needs to be updated. On larger systems, this can save significant time.

This is the list of all cabinet level components:

```
cc_mc (CC Microcontroller)
cc_bios (CC Tolapai BIOS)
cc_fpga (CC FPGA)
chia_fpga (CHIA FPGA)
```

This is a list of all blade level components:

```
cbb_mc (CBB BC Microcontroller)
ibb_mc (IBB BC Microcontroller)
anc_mc (ANC BC Microcontroller)
bc_bios (BC Tolapai BIOS)
lod_fpga (LOD FPGA)
node_bios (Node BIOS)
loc_fpga (LOC FPGA)
qloc_fpga (QLOC FPGA)
```

If the output of the `xtzap` command shows that only a specific type needs to be updated, then use the `-t` option with that type (this example uses the `node_bios` type).

```
crayadm@smw:~> xtzap -t node_bios s0
```

If the output of the `xtzap` command shows that only blade component types need to be updated, then use the `-b` option:

```
crayadm@smw:~> xtzap -b s0
```

If the output of the `xtzap` command shows that only cabinet component types need to be updated, then use the `-c` option:

```
crayadm@smw:~> xtzap -c s0
```

If the output of the `xtzap` command shows that both blade- and cabinet-level component types need to be updated, or if unsure of what needs to be updated, then use the `-a` option:

```
crayadm@smw:~> xtzap -a s0
```

b. Perform `xtbounce --linktune`, if not all current. Force `xtbounce` to do a linktune on the full system before checking firmware again.

```
crayadm@smw> xtbounce --linktune=all s0
```

c. Check firmware, after update and linktune. After updating them, confirm that they were all updated.

```
crayadm@smw> xtzap -r -v s0
```

5. Check routing configuration of the system.

The `rtr -R` command produces no output unless there is a routing problem.

Full system:

```
crayadm@smw> rtr -R s0
```

Partitioned system:

```
crayadm@smw> rtr -R p1
crayadm@smw> rtr -R p2
```

6. Update NIMS for new components. Now that the new components have been added, and the firmware is up to date, several NIMS commands are needed.

a. See what settings are for already existing similar nodes.

```
crayadm@smw> cnode list -p p0
```

b. If this blade was swapped out and replaced with a different type (that is, was compute, swapped for service), remove it from the old group.

```
crayadm@smw> cnode update --partition p1 -c p1 -G netroot_compute
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3
```

c. Assign the nodes to the correct config set, group (`compute`, `netroot_compute`, `service`, `login`, `dal`, etc.) and image.

```
crayadm@smw> cnode update --partition p1 -c p1 -g service -i /var/opt/cray/
imps/boot_images/service_XXX.cpio c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3
```

d. If this is a `netroot_compute` node, assign the key for `netroot` (can be combined with the config set, group and image assignment in above command).

```
crayadm@smw> cnode update --partition p1 -s netroot=compute-large_cle_XXX
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3
```

e. If this was a `netroot_compute` and is not anymore, remove the key.

```
crayadm@smw> cnode update --partition p1 -K netroot
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3
```

f. If this was a compute node, and is now a service, remove the rest of the extraneous keys.

```
crayadm@smw> cnode update --partition p1 -c p1 -K hsn_ipv4_mask
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3'
crayadm@smw> cnode update --partition p1 -c p1 -K hsn_ipv4_net
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3'
crayadm@smw> cnode update --partition p1 -c p1 -K sdbnodeip
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3'
crayadm@smw> cnode update --partition p1 -c p1 -K bootnodeip
c0-0c0s1n0 c0-0c0s1n1 c0-0c0s1n2 c0-0c0s1n3'
```

7. Update config sets with the new components. This will generate a new `/etc/hosts` file for the CLE nodes.

   Full system:

```
crayadm@smw> su -
smw# cfgset update p0
smw# exit
```

   Partitioned system:

```
crayadm@smw> su -
smw# cfgset update p1
smw# cfgset update p2
smw# exit
```

8. Update any workload manager configuration as specified in their documentation. For internal systems running native slurm, see *http://oskernel/wiki/Workload_Managers_Rhine_Redwood#Slurm*.

9. Boot the system using the standard boot procedure.

Add a New Disk to a Volume Group in a Storage Set

About this task

When more disk space is needed in an LVM volume group, add another physical volume to the `cray_bootraid_config.yaml` file and rerun `cray-ansible` for the node which owns the storage.

Procedure

1. Edit the `cray_bootraid_config.yaml` file to add another physical device to the list of devices in the volume group.

```
smw# vi /var/opt/cray/imps/config/sets/global/config/cray_bootraid_config.yaml
```

For example, to add a new disk device called "/dev/disk/by-id/wwn-0x600a0980006b47b7000000e756127f9d" to the "smw_node_vg" volume group, add the new disk device. Change this entry:

```
            -   key: smwdefault
                volume_groups:
                -   key: smw_node_vg
                    owner: smw
                    devices:
                    - /dev/disk/by-id/wwn-0x600a0980006b47b7000000e5561260a7
```

To be this:

```
            -   key: smwdefault
                volume_groups:
                -   key: smw_node_vg
                    owner: smw
                    devices:
                    - /dev/disk/by-id/wwn-0x600a0980006b47b7000000e5561260a7
                    - /dev/disk/by-id/wwn-0x600a0980006b47b7000000e756127f9d
```

2. Run `cray-ansible` on the node.

   If the storage was added to the SMW volume group.

   ```
   smw# /media/SMW/SMWinstall --mode=provision-storage
   ```

   If the storage was added to the boot node volume group.

   ```
   boot# /etc/init.d/cray-ansible start
   ```

   If the storage was added to the SDB node volume group.

   ```
   sdb# /etc/init.d/cray-ansible start
   ```

Reboot Controllers of a Cabinet or Blade

The `xtccreboot` command provides a means to reboot controllers. Options allow for rebooting all controllers of a specified type (cabinet or blade) or providing a list of controllers of a specified type to be rebooted.

For additional information, see the `xtccreboot(8)` man page.

Reboot cabinet controller c0-0, with verbose output

```
smw:~> xtccreboot -v -c c0-0
xtccreboot: /opt/cray-xt-pdsh/default/bin/pdsh -w "c0-0" /sbin/reboot
xtccreboot: reboot sent to specified CCs
```

Bounce Blades Repeatedly Until All Blades Succeed

About this task

IMPORTANT: This iterative `xtbounce` should typically be done in concert with an `xtbootsys` automation file where bounce and routing are turned off.

Procedure

1. Bounce the system.

```
smw:~> xtbounce s0
```

2. Bounce any blades that failed the first bounce. Repeat as necessary.

3. Execute the following command, which copies route configuration files, based on the *idlist* (such as `s0`), to the blade controllers. This avoids having old, partial route configuration files left on the blades that were bounced earlier and ensures that the links are initialized correctly.

```
smw:~> xtbounce --linkinit s0
```

4. Route and boot the system without executing `xtbounce` again. If using a `xtbootsys` automation file, specify `set data(config,xtbounce) 0`, or use the `xtbootsys --config xtbounce=0` command.

Shut Down the System Using the auto.xtshutdown File

The preferred method to shut down the system is to use the `xtbootsys` command with the auto shutdown file as follows:

```
crayadm@smw:~> xtbootsys -s last -a auto.xtshutdown
```

Or, for a partitioned system with partition `pN`:

```
smw:~# xtbootsys --partition pN -s last -a auto.xtshutdown
```

This method shuts down the compute nodes (which are commonly also Lustre clients), then executes `xtshutdown` on service nodes, halting the nodes and then stopping processes on the SMW. A system administrator can shut down the system using both user-defined and built-in procedures in the `auto.xtshutdown` file, which is located on the SMW in the `/opt/cray/hss/default/etc` directory.

For related procedures, see *CLE Installation and Configuration Guide*. For more information about using automation files, see the `xtbootsys(8)` man page.

The xtshutdown Command

The `xtshutdown` command executes a series of commands locally on the boot node and service nodes to shut down the system in an orderly fashion. The sequence of shutdown steps and the nodes on which to execute them are defined by the system administrator in the `/etc/opt/cray/init-service/xtshutdown.conf` file or in the file specified by the environment variable `XTSHUTDOWN_CONF`.

Root user privileges are required to run `xtshutdown`. Passwordless `ssh` must be enabled for the root user from the boot node to all service nodes.

The `xtshutdown` command uses `pdsh` to invoke commands on the selected service nodes (i.e., boot node, SDB node, a class of nodes, or a single host). A system administrator can define functions to execute when the system is shut down. Place these functions in the `/etc/opt/cray/init-service/xt_shutdown_local` file or the file defined by the `XTSHUTDOWN_LOCAL` environment variable.

Shut Down Service Nodes

Prerequisites

`Root` user privileges are required to run `xtshutdown`. Passwordless `ssh` must be enabled for the `root` user from the boot node to all service nodes.

CAUTION: The `xtshutdown` command does not shut down compute nodes. To shut down the compute and service nodes, see *Shut Down the System or Part of the System Using the xtcli shutdown Command*.

About this task

For information about shutting down service nodes, see the `xtshutdown(8)` man page.

Procedure

1. Modify the `/etc/opt/cray/init-service/xtshutdown.conf` file or the file specified by the `XTSHUTDOWN_CONF` environment variable to define the sequence of shutdown steps and the nodes on which to execute them. The `/etc/opt/cray/init-service/xtshutdown.conf` file resides on the boot node.

2. If desired, define functions to execute when the system is shut down. Place these functions in the `/etc/opt/cray/init-service/xt_shutdown_local` file or the file defined by the `XTSHUTDOWN_LOCAL` environment variable.

3. Execute `xtshutdown`.

   ```
   boot:~ # xtshutdown
   ```

   After the software on the nodes is shutdown, the administrator can halt the hardware, reboot, or power down.

Shut Down the System or Part of the System Using the xtcli shutdown Command

The HSS `xtcli shutdown` command shuts down the system or a part of the system. To shut down compute nodes, execute the `xtcli shutdown` command. Under normal circumstances, for example to successfully disconnect from Lustre, invoking the `xtcli shutdown` command attempts to gracefully shut down the specified nodes.

For information, see the `xtcli(8)` man page.

Shut down all compute nodes

```
crayadm@smw:~> xtcli shutdown compute
```

Shut down specified compute nodes

For this example, shut down only compute nodes in cabinet `c13-2`:

```
crayadm@smw:~> xtcli shutdown c13-2
```

Shut down all nodes of a system

```
crayadm@smw:~> xtcli shutdown s0
```

Shut down a partition *pN* of a system

```
crayadm@smw:~> xtcli shutdown pN
```

Force nodes to shut down (immediate halt)

When all nodes of a system must be halted immediately, use the `-f` argument; nodes will not go through their normal shutdown process. Forced shutdown occurs even if the nodes have an alert status present.

```
crayadm@smw:~> xtcli shutdown -f s0
```

After the software on the nodes is shutdown, the system administrator can halt the hardware, reboot, or power down.

Stop System Components

When a system administrator removes, stops, or powers down components, any applications and compute processes that are running on those components are lost.

Reserve a Component

To allow applications and compute processes to complete before stopping components, use the HSS `xtcli set_reserve` *`idlist`* command to prevent the selected nodes from accepting new jobs.

A node running CNL and using ALPS is considered to be down by ALPS after it is reserved using the `xtcli set_reserve` command. The output from `apstat` will show the node as down (DN), even though there may be an application running on that node. This DN designation indicates that no other work will be placed on the node after the currently running application has terminated.

For more information, see the `xtcli_set(8)` man page.

Reserve a component

```
crayadm@smw:~> xtcli set_reserve idlist
```

Power Down Blades or Cabinets

WARNING: Power down the cabinets with software commands. Tripping the circuit breakers may result in damage to system components.

WARNING: Before powering down a blade or a cabinet, ensure the operating system is not running.

The `xtcli power down` command powers down the specified cabinet and/or blades within the specified partition, chassis or list of blades. Cabinets must be in the `READY` state to receive power commands.

When a request is made to power down a blade consisting of Intel® Xeon® processor Scalable Family nodes or a Cabinet containing processor blades of this type, the nodes are powered off into the G3 state (full power off) prior to the Cabinet controller removing power from the blade. See *System Component States* on page 60.

The `xtcli power down` command has the following form, where *`physIDlist`* is a comma-separated list of cabinets, blades, or nodes present on the system.

```
xtcli power down physIDlist
```

The `xtcli power force_down` and `xtcli power down_slot` commands are aliases for the `xtcli power down` command. For information about disabling and enabling components, see *Disable Hardware Components*, and *Enable Hardware Components*, respectively.

WARNING: Although a blade is powered off, the HSS in the cabinet is live and has power.

For information about powering down a component, see the `xtcli_power(8)` man page.

Power down a specified blade

For this example, power down a blade with the ID `c0-0c0s7`:

```
crayadm@smw:~> xtcli power down c0-0c0s7
```

Power Down a Specific Node

The `xtcli power down_node` command powers down the specified node and/or nodes within a specified partition, chassis, list of blades, or list of nodes. When specifying a specific node or list of nodes, all node types are powered down to the G3 state except for Intel® Xeon® processor Scalable Family nodes, which are powered down to the S5 state (soft off). These nodes can be powered down to the G3 state using one of the following methods:

- Issue the `xtcli power down_node` command with the `--with-si` flag.
- Power down the blade that the Intel® Xeon® processor Scalable Family nodes reside on. Blades must be in the READY state to receive power commands. See *System Component States* on page 60.

The `xtcli power down_node` command has the following form, where *`physIDlist`* is a comma-separated list of cabinets, blades, or nodes present on the system.

```
xtcli power down_node physIDlist
```

Power down specified nodes

In these example commands, `c0-0c0s7n0` is a Haswell node and `c0-1c1s8n2` is a Intel® Xeon® processor Scalable Family node. The following `down_node` power command does not include the `--with-si` flag.

```
crayadm@smw:~> xtcli power down_node c0-0c0s7n0,c0-1c1s8n2
```

HSS reports both nodes as being in the `off` state. The state of `c0-0c0s7n0` is G3, and the state of `c0-1c1s8n2` is S5.

The next example uses the `--with-si` flag to power down the same two nodes.

```
crayadm@smw:~> xtcli power down_node --with-si c0-0c0s7n0,c0-1c1s8n2
```

HSS reports both nodes as being in the `off` state. Both nodes are in the G3 state. See the `xtcli_power(8)` man page for more information.

Halt Selected Nodes

Use the HSS `xtcli halt` command to halt selected nodes. For more information, see the `xtcli(8)` man page.

Halt a node

For this example, halt node `157`:

```
crayadm@smw:~> xtcli halt 157
```

Restart a Blade or Cabinet

IMPORTANT: Change the state of the hardware only when the operating system is not running or is shut down.

The `xtcli power up` command powers up the specified cabinet and/or blades within the specified partition, chassis or list of blades. Cabinets must be in the `READY` state (see *System Component States* on page 60) to receive power commands. The `xtcli power up` command does not attempt to power up network mezzanine cards or nodes that are handled by the `xtbounce` command during system boot.

The `xtcli power up_slot` command is an alias for the `xtcli power up` command.

The `xtcli power up` command has the following form, where *`physIDlist`* is a comma-separated list of cabinets or blades present on the system.

```
xtcli power up physIDlist
```

For more information, see the `xtcli_power(8)` man page.

Power up blades in `c0-0c0s7`

```
crayadm@smw:~> xtcli power up c0-0c0s7
```

Abort Active Sessions on the HSS Boot Manager

About this task

Use the HSS `xtcli session abort` command to abort sessions in the boot manager. A session corresponds to executing a specific command such as `xtcli power up` or `xtcli boot`.

For more information about manager sessions, see the `xtcli(8)` man page.

Procedure

1. Display all running sessions in the boot manager. Only the boot manager supports multiple simultaneous sessions.

```
crayadm@smw:~> session show BM all
```

2. Abort the selected session, *session_id*.

```
crayadm@smw:~>  xtcli session abort BM session_id
```

Display and Change Software System Status

The user command `xtnodestat` provides a display of the status of nodes: how they are allocated and to what jobs. The `xtnodestat` command provides current job and node status summary information, and it provides an interface to ALPS and jobs running on CNL compute nodes. ALPS must be running in order for `xtnodestat` to report job information.

For more information, see the `xtnodestat(1)` man page.

View and Change the Status of Nodes

Use the `xtprocadmin` command on a service node to view the status of components of a booted system in the `processor` table of the SDB. The command enables the system administrator to retrieve or set the processing mode (`interactive` or `batch`) of specified nodes. The administrator can display the state (`up`, `down`, `admindown`, `route`, or `unavailable`) of the selected components, if needed. The administrator can also allocate processor slots or set nodes to become unavailable at a particular time. The node is scheduled only if the status is `up`.

When the `xtprocadmin -ks` option is used, then the option can either a normal argument (`up`, `down`, etc.), or it can have a colon in it to represent a conditional option; for example, the option of the form `up:down` means "if state was up, mark down".

For more information, see the `xtprocadmin(8)` man page.

View node characteristics

```
login:~> xtprocadmin
   NID    (HEX)    NODENAME     TYPE    STATUS        MODE
     1      0x1  c0-0c0s0n1  service        up       batch
     2      0x2  c0-0c0s0n2  service        up       batch
     5      0x5  c0-0c0s1n1  service        up       batch
     6      0x6  c0-0c0s1n2  service        up       batch
```

```
      8       0x8   c0-0c0s2n0   compute          up        batch
      9       0x9   c0-0c0s2n1   compute          up        batch
     10       0xa   c0-0c0s2n2   compute          up        batch
     11       0xb   c0-0c0s2n3   compute          up        batch
```

View all node attributes

```
login:~> xtprocadmin -A
   NID    (HEX)    NODENAME     TYPE ARCH        OS CPUS  CU AVAILMEM   PAGESZ CLOCKMHZ GPU SOCKETS DIES C/
CU                      LABEL0                              LABEL1
LABEL2                     LABEL3
     1      0x1  c0-0c0s0n1  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     2      0x2  c0-0c0s0n2  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     5      0x5  c0-0c0s1n1  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     6      0x6  c0-0c0s1n2  service   xt (service)   16   8    32768     4096     2600   0       1    1
2
     8      0x8  c0-0c0s2n0  compute   xt       CNL   32  16    65536     4096     2600   0       2    2
2
     9      0x9  c0-0c0s2n1  compute   xt       CNL   32  16    65536     4096     2600   0       2    2
2
    10      0xa  c0-0c0s2n2  compute   xt       CNL   32  16    65536     4096     2600   0       2    2
2
```

View selected attributes of selected nodes

For this example, the `-a` option lists the selected attributes to display:

```
login:~> xtprocadmin -n 8 -a arch,clockmhz,os,cores
   NID    (HEX)    NODENAME     TYPE ARCH CLOCKMHZ        OS CPUS
     8      0x8  c0-0c0s2n0  compute   xt     2600       CNL   32
```

Disable a node

For this example, the `admindown` option disables node `c0-0c0s3n1` such that it cannot be allocated:

```
crayadm@nid00004:~> xtprocadmin -n c0-0c0s3n1 -k s admindown
```

Disable all processors

```
crayadm@nid00004:~> xtprocadmin -k s admindown
```

Mark a Compute Node as a Service Node

Use the `xtcli mark_node` command to mark a node in a compute blade to have a role of `service` or `compute`; `compute` is the default. It is not permitted to change the role of a node on a service blade, which always has the `service` role.

Marking a node on a compute blade as `service` or `compute` allows the administrator to load the desired boot image at boot time. Compute nodes marked as `service` can run software-based services. A request to change the role of a running node (that is, the node is in the `ready` state and the operating system is running) will be denied.

For more information, see the `xtcli(8)` man page and *Check the Status of System Components* on page 197.

Find Node Information

Translate Between Physical ID Names and Integer NIDs

To translate between physical ID names (cnames) and integer NIDs, generate a system map on the System Management Workstation (SMW) and filter the output, enter the following command:

```
crayadm@smw:~> rtr --system-map | grep cname | awk '{ print $1 }'
```

For more information, see the `rtr(8)` man page.

Find Node Information Using the `xtnid2str` Command

The `xtnid2str` command converts numeric node identification values to their physical names (cnames). This allows conversion of Node ID values, ASIC NIC address values, or ASIC ID values.

For additional information, see the `xtnid2str(8)` man page.

Find the physical ID for node 38

```
smw:~> xtnid2str 28
node id 0x26 = 'c0-0c0s1n2'
```

Find the physical ID for nodes 0, 1, 2, and 3

```
smw:~> xtnid2str 0 1 2 3
node id 0x0 = 'c0-0c0s0n0'
node id 0x1 = 'c0-0c0s0n1'
node id 0x2 = 'c0-0c0s1n0'
node id 0x3 = 'c0-0c0s1n1'
```

Find the physical IDs for Aries IDs 0-7

```
smw:~> xtnid2str -a 0-7
aries id 0x0 = 'c0-0c0s0a0'
aries id 0x1 = 'c0-0c0s1a0'
aries id 0x2 = 'c0-0c0s2a0'
aries id 0x3 = 'c0-0c0s3a0'
aries id 0x4 = 'c0-0c0s4a0'
aries id 0x5 = 'c0-0c0s5a0'
aries id 0x6 = 'c0-0c0s6a0'
aries id 0x7 = 'c0-0c0s7a0'
```

Find Node Information Using the `nid2nic` Command

The `nid2nic` command prints the *nid*-to-*nic* address mappings, *nic*-to-*nid* address mappings, and a specific *physical_location*-to-*nic* address and *nid* mappings.

For information about using the `nid2nic` command, see the `nid2nic(8)` man page.

Print the *nid*-to-*nic* address mappings for the node with NID 31

```
smw:~> nid2nic 31
NID:0x1f        NIC:0x21        c0-0c0s7n3
```

Print the *nid*-to-*nic* address mappings for the node with NID 31, but specify the NIC value in the command line

```
smw:~> nid2nic -n 0x21
NIC:0x21        NID:0x1f        c0-0c0s7n3
```

Display and Change Hardware System Status

A system administrator can execute commands that look at and change the status of the hardware.

CAUTION: Execute commands that change the status of hardware only when the operating system is shut down.

Generate HSS Physical IDs

The HSS `xtgenid` command generates HSS physical IDs, for example, to create a list of blade controller identifiers for input to the flash manager. Selection can be restricted to components of a particular type. Only user `root` can execute the `xtgenid` command.

For more information, see the `xtgenid(8)` man page.

Create a list of node identifiers that are not in the DISABLE, EMPTY, or OFF state

```
smw:~ # xtgenid -t node --strict
```

Disable Hardware Components

If links, nodes, or Cray ASICs have hardware problems, the system administrator can direct the system to ignore the components with the `xtcli disable` command.

By default, when disabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

The `xtcli disable` command has the following form, where *idlist* is a comma-separated list of components (in cname format) that the system is to ignore. The system disregards these links or nodes.

```
xtcli disable [{-t type [-a] } | -n] [-f] idlist
```

IMPORTANT: The `-n` option with the `xtcli disable` command must be used carefully because this may create invalid system state configurations.

Disabling of a cabinet, chassis, or blade will fail if any nodes under the component are in the `ready` state, unless the force option (`-f`) is used. An error message will indicate the reason for the failure.

Disabling of a node in the `ready` state will fail, unless the force option (`-f`) is used. An error message will indicate the reason for the failure.

The state of `empty` components will not change when using the `disable` command, unless the force option (`-f`) is used.

For detailed information about using the `xtcli disable` command, see the `xtcli(8)` man page.

Disable the Aries ASIC `c0-0c1s3a0`

1. Determine that the ASIC is in the `OFF` state.

   ```
   crayadm@smw:~> xtcli status -t aries c0-0c1s3a0
   ```

2. If the ASIC is not in the `OFF` state, power down the blade that contains the ASIC.

   ```
   crayadm@smw:~> xtcli power down c0-0c1s3
   ```

3. Disable the ASIC.

   ```
   crayadm@smw:~> xtcli disable c0-0c1s3a0
   ```

4. Power up the blade that contains the ASIC.

   ```
   crayadm@smw:~> xtcli power up c0-0c1s3
   ```

Enable Hardware Components

If links, nodes, or Cray ASICs that have been disabled are later fixed, the system administrator can add them back to the system with the `xtcli enable` command.

The `xtcli enable` command has the following form, where *`idlist`* is a comma-separated list of components (in cname format) for the system to recognize.

```
xtcli enable [{-t type [-a] } | -n] [-f] idlist
```

By default, when disabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

> **IMPORTANT:** The `-n` option with the `xtcli disable` command must be used carefully because this may create invalid system state configurations.

The state of `empty` components does not change when using the `xtcli enable` command, unless the force option (`-f`) is used.

The state of `off` means that a component is present on the system. If the component is a blade controller, node, or ASIC, then this will also mean that the component is powered off. If the administrator disables a component, the state shown becomes `disabled`. When the `xtcli enable` command is used to enable that component for use once again, its state switches from `disabled` to `off`. In the same manner, enabling an empty component means that its state switches from `empty` to `off`.

For more information, see the `xtcli(8)` man page.

Set Hardware Components to `EMPTY`

Use the `xtcli set_empty` command to set a selected component to the `EMPTY` state. HSS managers and the `xtcli` command ignore empty or disabled components.

Setting a selected component to the `EMPTY` state is typically done when a component, usually a blade, is physically removed. By setting it to `EMPTY`, the system ignores it and routes around it.

By default, when disabling a component, this command takes into consideration the hierarchy of components, performs the action upon the identified component(s) and cascades that action to any subcomponent of the identified component(s), unless the `-n` option is specified.

> **IMPORTANT:** The `-n` option with the `xtcli disable` command must be used carefully because this may create invalid system state configurations.

For more information, see the `xtcli(8)` man page.

Set a blade to the `EMPTY` state

```
crayadm@smw:~> xtcli set_empty -a c0-0c1s7
```

Lock Hardware Components

Components are automatically locked when a command that can change their state is running. As the command is started, the state manager locks these components so that nothing else can affect their state while the command executes. When the manager is finished with the command, it unlocks the components.

Use the HSS `xtcli lock` command to lock components. Locking a component prints out the state manager session ID.

For more information, see the `xtcli(8)` man page.

Lock cabinet `c0-0`

```
crayadm@smw:~> xtcli lock -l c0-0
```

Show all session (lock) data

```
crayadm@smw:~> xtcli lock show
```

Unlock Hardware Components

Use the HSS `xtcli lock` command to unlock components. This command is useful when an HSS manager fails to unlock some set of components.

The system administrator can manually check for locks with the `xtcli lock show` command and then unlock them. Unlocking a component does not print out the state manager session ID. The `-u` option must be used to unlock a component as follows:

```
crayadm@smw:~> xtcli lock -u lock_number
```

Where *`lock_number`* is the value given when initiating the lock; it is also indicated in the `xtcli lock show` query. Unlocking does nothing to the state of the component other than to release locks associated with it.

HSS daemons cannot affect components that are locked by a different session.

Set the Turbo Boost Limit

Turbo boost limiting is supported on the Intel® Xeon® processor Scalable family. Turbo boost limiting is NOT supported on Intel® Xeon Phi™ "Knights Landing" (KNL) or on Intel® Xeon® "Sandy Bridge" processors.

Because Intel processors have a high degree of variability in the amount of turbo boost each processor can supply, limiting the amount of turbo boost can reduce performance variability and reduce power consumption. Turbo boost can be limited by setting the `turbo_boost_limit` kernel parameter to one of these valid values:

Value	Result
0	Disable turbo boost.
100	Limits turbo boost to 100 MHz.
200	Limits turbo boost to 200 MHz.
300	Limits turbo boost to 300 MHz.
400	Limits turbo boost to 400 MHz.
999 (default)	No limit is applied.

The limit applies only when a high number of cores are active. On an N-core processor, the limit is in effect when the active core count is N, N-1, N-2, or N-3. For example, on a 12-core processor, the limit is in effect when 12, 11, 10, or 9 cores are active.

Set or Change the Turbo Boost Limit Parameter

Set or change the turbo boost limit parameter using one of the following methods:

- TEMPORARY. To make a one-time non-persistent change, warm boot the compute nodes using the `--compute-boot-params` option.

  ```
  smw> xtbootsys --reboot -L DEFAULT --compute-boot-params \
  turbo_boost_limit=value idlist
  ```

  where *`value`* is one of the values listed above and *`idlist`* is a comma-separated list of compute node cnames (or `all_comp` for all compute nodes) to be booted. **This configuration change is not persisted.**

- PERMANENT. To make a persistent change, use `cnode` (as root) to change the parameter, and then reboot the nodes. Note that the following commands target all nodes or all compute nodes. To specify individual nodes, add their cnames at the end of the command line.

  1. To list the current kernel parameters:

     ```
     smw# cnode list
     ```

  2. To change the `turbo_boost_limit` kernel parameter for all compute nodes, substitute one of the values listed above for *`value`* in this command:

     ```
     smw# cnode update --filter group=compute \
     --add-parameter turbo_boost_limit=value
     ```

  3. To remove the change, if needed, use one of these commands:

     ```
     smw# cnode update --filter group=compute \
     --remove-parameter turbo_boost_limit
     ```

Verify that the Turbo Boost Limit Parameter was Changed

If desired, the `turbo_boost_limit` change can be verified on the nodes after they have been rebooted. To verify, look at the contents of the following file on the target NIDs.

```
login> aprun -L nidlist cat /sys/module/cray_power_management/\
parameters/turbo_boost_limit
```

where *nidlist* is a comma-separated list of NIDs.

Perform Parallel Operations on Service Nodes

Use `pdsh`, the CLE parallel remote shell utility for service nodes, to issue commands to groups of nodes in parallel. The system administrator can select the nodes on which to use the command, exclude nodes from the command, and limit the time the command is allowed to execute. Only user `root` can execute the `pdsh` command. The command has the following form:

```
pdsh [options] command
```

For more information, see the `pdsh(1)` man page.

Restart the NTP service

```
boot:~ # pdsh -w 'login[1-9]' /etc/init.d/ntp restart
```

Perform Parallel Operations on Compute Nodes

The parallel command tool (`pcmd`) facilitates execution of the same commands on groups of compute nodes in parallel, similar to `pdsh`. Although `pcmd` is launched from a service node, it acts on compute nodes. It allows administrators and/or, if the site deems it feasible, other users to securely execute programs in parallel on compute nodes. The user can specify on which nodes to execute the command. Alternatively, the user can specify an application ID (*apid*) to execute the command on all the nodes available under that *apid*.

An unprivileged user must execute the command targeting nodes where the user is currently running an `aprun`. A `root` user is allowed to target any compute node, regardless of whether there are jobs running there or not. In either case, if the `aprun` exits and the associated applications are killed, any commands launched by `pcmd` will also exit.

By default, `pcmd` is installed as a `root`-only tool. It must be installed as `setuid root` in order for unprivileged users to use it.

The `pcmd` command is located in the `nodehealth` module. If the `nodehealth` module is not part of the default profile, load it by specifying:

```
module load nodehealth
```

For additional information, see the `pcmd(1)` man page.

xtbounce Error Message Indicates Cabinet Controller and Its Blade Controllers Not in Sync

During the `gather_cab_pwr_states` phase of `xtbounce`, if the HSS software on a cabinet controller and any of its blade controllers is out of sync, error messages such as the following will be printed during the `xtbounce`.

```
***** gather_cab_pwr_states *****
18:28:42 - Beginning to wait for response(s)

ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
ERROR: rs_phys_node2ascii() failed for node struct 0xb7e70150080700f8
```

If this occurs, it indicates that the blade controller software is at a different revision than the cabinet controller software. `xtbounce` will print a list of cabinets for which this error has occurred. The message will be similar to the following:

```
ERROR: power state check error on 2 cabinet(s)
WARNING: unable to find c0-0 in err_cablist
WARNING: unable to find c0-2 in err_cablist
```

This error is an indication that when the HSS software was previously updated, the cabinet controllers and the blade controllers were not updated to the same version.

To correct this error, cancel out of `xtbounce` (with **Ctrl-C**), wait approximately five minutes for the `xtbounce` related activities on the blade controllers to finish, then reboot the cabinet controller(s) and their associated blade controllers to get the HSS software synchronized. Following this, the `xtbounce` may be executed once again.

Reduce Impact of Btrfs Periodic Maintenance on SMW Performance

About this task

Btrfs (B-tree file system) runs periodic maintenance. The weekly and monthly maintenance scripts, which include balance, trim, and scrub actions, can consume large amounts of compute resource. This can impact a site's ability to use the SMW for normal operations, even using SSH to log into nodes. This procedure describes how to reduce the impact to SMW performance by controlling when these scripts are run.

Procedure

1. Create a file `/etc/cron.d/cray_btrfs.cron`.

   The new cron file needs to be in `/etc/cron.d` because the btrfs RPM installs links to maintenance scripts into the `/etc/cron.{weekly,monthly}` directories.

   ```
   smw# vi /etc/cron.d/cray_btrfs.cron
   ```

   Add these lines to the new file. Adjust as needed for this site.

   ```
   # Control when btrfs maintenance scripts run by deleting the corresponding
   # 'lastrun' files at a predetermined time.  Caveat, this affects all of the
   # scripts in the corresponding cron directories (/etc/cron.{weekly,monthy})
   ```

```
# Run weekly on Saturday at 2 AM as root
0 2 * * 6 root rm -f /var/spool/cron/lastrun/cron.weekly
# Run monthly on the first Sunday of the month at 2 AM as root
0 2 * * 0 root [ $(date +%d) -le 07 ] && rm -f /var/spool/cron/lastrun/cron.monthly
```

2. Set ownership of the new cron file to root,root with permissions 644.

```
smw# chown root:root /etc/cron.d/cray_btrfs.cron
smw# chmod 644 /etc/cron.d/cray_btrfs.cron
```

Power-cycle a Component to Handle Bus Errors

About this task

Bus errors are caused by machine-check exceptions. If a bus error occurs, try power-cycling the component.

Procedure

1. Power down the components. The *physIDlist* is a comma-separated list of components present on the system.

```
crayadm@smw:~> xtcli power down physIDlist
```

2. Power up the components.

```
crayadm@smw:~> xtcli power up physIDlist
```

When a Component Fails

Components that fail are replaced as field replaceable units (FRUs). FRUs include compute blade components, service blade components, and power and cooling components.

When a field replaceable unit (FRU) problem arises, contact a Customer Service Representative to schedule a repair.

Capture and Analyze System-level and Node-level Dumps

The `xtdumpsys` command collects and analyzes information from a Cray system that is failing or has failed, has crashed, or is hung. Analysis is performed on, for example, event log data, active heartbeat probing, voltages, temperatures, health faults, in-memory console buffers, and high-speed interconnection network errors. When failed components are found, detailed information is gathered from them.

To collect similar information for components that have not failed, invoke the `xtdumpsys` command with the `--add` option and name the components from which to collect data. The HSS `xtdumpsys` command saves dump information in `/var/opt/cray/dump/`*`timestamp`* by default.

> **NOTE:** When using the `--add` option to add multiple components, separate components with spaces, not commas.

Dump information about a working component

For this example, dump the entire system and collect detailed information from all blade controllers in chassis `0` of cabinet `0`:

```
crayadm@smw:~> xtdumpsys --add c0-0c0s0
```

The `xtdumpsys` command is written in Python and supports plug-ins written in Python. A number of plug-in scripts are included in the software release. Call `xtdumpsys --list` to view a list of included plug-ins and their respective directories. The `xtdumpsys` command also now supports the use of configuration files to specify `xtdumpsys` presets, rather than entering them via the command line.

For more information, see the `xtdumpsys(8)` man page.

Configure xtdumpsys for Systems Using passwordless ssh

The `xtdumpsys` command collects data from a system that is failing, crashed, or hung. By default, `xtdumpsys` collects information only from the SMW and HSS. To collect data from other system nodes automatically, specify plugins from the `/etc/opt/cray/dumpsys/config/plugin.conf` file that enable `xtdumpsys` to gather the needed data. `xtdumpsys` runs all the enabled plugins unless individual plugins are included or excluded on the command line.

Access to the additional nodes can be achieved using passwordless `ssh`. `xtdumpsys` is not aware of site-specific passwords. When `xtdumpsys` cannot access a node because of a site-specific password or an inability to used passwordless `ssh`, it prompts the user for a password for each time, which could be very often. `xtdumpsys` uses the following ssh connections (*`user`*@*`node`*), and passwordless `ssh` needs to be set up for these connections. To skip one or more of these `ssh` connections in `xtdumpsys` processing, exclude the plugin specified with the connection.

- root@boot → root@sdb
  - Skip using `--plugins-exclude slurm_status` command line option.
- root@boot → crayadm@sdb
  - Skip using the `--plugins-exclude slurm_status` command line option.
- For each compute node that is `down`, or `admindown`, or `unavail`:
  - root@boot → root@<*`compute_node`*>
    - Skip using the `--plugins-exclude alps_compute_logs` command line option.
- For each node with the `dwrest` service (`/boot/dwrest_gw.conf`):
  - root@boot → root@<*`node`*>
    - Skip using the `--plugins-exclude datawarp` command line option.
- For each suspect/dead node:
  - root@boot → root@<*`node`*>
    - Skip using the `--plugins-exclude knc_host_logs` command line option.
- For each service node:
  - crayadm@boot → root@<*`service_node`*>
    - Skip using the `--plugins-exclude systemd_status` command line option.

- For each node specified by the `xtdumpsys --add` (node) option:
  - root@<*boot*> → root@<*node*>
    - Skip using the `--plugins-exclude ansible_changed_files` command line option.

A more permanent way of excluding plugins is to edit the `/etc/opt/cray/dumpsys/config/default.conf` file. Add the plugin names, delimited with commas, to the `plugins_exclude_default` setting.

cdump and crash Utilities for Node Memory Dump and Analysis

The `cdump` and `crash` utilities may be used to analyze the memory on any Cray service node or CNL compute node. The `cdump` command is used to dump node memory to a file. After `cdump` completes, the `crash` utility can be used on the dump file generated by `cdump`.

Cray recommends executing the `cdump` utility only if a node has panicked or is hung, or if a dump is requested by Cray.

To select the desired access method for reading node memory, use the `cdump -r` *`access`* option. Valid access methods are:

`xt-bhs` The `xt-bhs` method uses a basic hardware system server that runs on the SMW to access and read node memory. `xt-bhs` is the default access method for these systems.

`xt-hsn` The `xt-hsn` method utilizes a proxy that reads node memory through the High-speed Network (HSN). The `xt-hsn` method is faster than the `xt-bhs` method, but there are situations where it will not work (for example, if the ASIC is not functional). However, the `xt-hsn` method is preferable because the dump completes in a short amount of time and the node can be returned to service sooner.

`xt-file` The `xt-file` method is used for memory dump file created by the `-z` option. The compressed memory dump file must be uncompressed prior to executing this command. Use the file name for *`node-id`*.

`xc-knc` The `xc-knc` method is used to dump Intel Xeon Phi nodes. Use this method when dumping only the Xeon Phi coprocessor without dumping the host node. When dumping the host node, do not use `xc-knc`. A host node dump automatically includes dumping the Xeon Phi coprocessors unless they are suppressed by specifying the `-n` option.

To dump Cray node memory, *`access`* takes the following form:

```
method[@host]
```

For additional information, see the `cdump(8)` and `crash(8)` man pages.

Dump and Reboot Nodes Automatically

The SMW daemon `dumpd` initiates automatic dump and reboot of nodes when requested by the Node Health Checker (NHC).

CAUTION: The `dumpd` daemon is invoked automatically by `xtbootsys` on system (or partition) boot. In most cases, system administrators do not need to use this daemon directly.

A system administrator can set global variables in the `/etc/opt/cray/nodehealth/nodehealth.conf` configuration file to control the interaction of NHC and `dumpd`. For more information about NHC and the `nodehealth.conf` configuration file, see *Configure the Node Health Checker (NHC)*.

Variables can also be set in the /etc/opt/cray-xt-dumpd/dumpd.conf configuration file on the SMW to control how dumpd behaves on the system.

Each CLE release package also includes an example dumpd configuration file, /etc/opt/cray-xt-dumpd/dumpd.conf.example. The dumpd.conf.example file is a copy of the /etc/opt/cray-xt-dumpd/dumpd.conf file provided for an initial installation.

> **IMPORTANT:** The /etc/opt/cray-xt-dumpd/dumpd.conf file is not overwritten during a CLE upgrade if the file already exists. This preserves the site-specific modifications previously made to the file. Cray recommends comparing the site's /etc/opt/cray-xt-dumpd/dumpd.conf file content with the /etc/opt/cray-xt-dumpd/dumpd.conf.example file provided with each release to identify any changes and then update the site's /etc/opt/cray-xt-dumpd/dumpd.conf file accordingly.
>
> If the /etc/opt/cray-xt-dumpd/dumpd.conf file does not exist, then the /etc/opt/cray-xt-dumpd/dumpd.conf.example file is copied to the /etc/opt/cray-xt-dumpd/dumpd.conf file.

The CLE installation and upgrade processes automatically install dumpd software, but it must be explicitly enabled.

The /etc/opt/cray-xt-dumpd/dumpd.conf Configuration File

The dumpd configuration file, /etc/opt/cray-xt-dumpd/dumpd.conf, is located on the SMW. There is no need to change any installation configuration parameters, but a system administrator can edit the /etc/opt/cray-xt-dumpd/dumpd.conf file to customize how dumpd behaves on the system using the following configuration variables.

enable: yes|no

Provides a quick on/off switch for all dumpd functionality.

Default is no.

partitions: *number*

Specifies whether or not dumpd acts on specific partitions or ranges of partitions. Placing ! in front of a partition or range disables it.

For example, specifying

```
partitions: 1-10,!2-4
```

enables partitions 1, 5, 6, 7, 8, 9, and 10 but not 2, 3, or 4. Partitions must be explicitly enabled. Leaving this option blank disables all partitions.

disabled_action: ignore|queue

Specifies what to do when requests come in for a disabled partition. If ignore is specified, requests are removed from the database and not acted upon. If queue is specified, requests continue to build while dumpd is disabled on a partition. When the partition is reenabled, the requests will be acted on. Specifying queue is not recommended if dumpd will be disabled for long periods of time, as it can cause SMW stress and database problems.

save_output: always|errors|never

Indicates when to save stdout and stderr from dumpd commands that are executed. If save_output is set to always, all output is saved. If errors is specified, output is saved only when the command exits with a nonzero exit code. If never is specified, output is never saved.

The default is to save output on errors.

command_output: *directory*

Specifies where to save output of `dumpd` commands, per the `save_output` variable. The command output is put in the file *`action.pid.timestamp`*`.out` in the directory specified by this option.

Default directory is `/var/opt/cray/dump`.

dump_dir: *directory*

Specifies the directory in which to save dumps.

Default directory is `/var/opt/cray/dump`.

max_disk: *nnnMB*| unlimited

Specifies the amount of disk space beyond which no new dumps will be created. This is not a hard limit; if `dumpd` sees that this directory has less than this amount of space, it starts a new dump, even if that dump subsequently uses enough space to exceed the `max_disk` limit.

The default value is `max_disk: unlimited`.

no_space_action: *action*

Specifies a command to be executed if the directory specified by the variable `dump_dir` does not have enough space free, as specified by `max_disk`. For example:

Deletes the oldest dump in the dump directory:

```
no_space_action: rm -rf $dump_dir/$(ls -rt $dump_dir | head
-1)
```

Moves the oldest dump somewhere useful:

```
no_space_action: mv $dump_dir/$(ls -t $dump_dir|head
-1) /some/dump/archive
```

Sends E-mail to an administrator at *admin@fictionalcraysite.com*:

```
no_space_action: echo "" | mail -s "Not Enough Space in
$dump_dir" \
admin@fictionalcraysite.com
```

The dumpd-dbadmin Tool

The `dumpd` daemon sits and waits for requests from NHC (or some other entity using the `dumpd-request` tool). When `dumpd` gets a request, it creates a database entry in the `mznhc` database for the request, and calls the script `/opt/cray-xt-dumpd/default/bin/executor` to read the `dumpd.conf` configuration file and perform the requested actions.

Use the `dumpd-dbadmin` tool to view or delete entries in the `mznhc` database in a convenient manner.

The dumpd-request Tool

Use the `dumpd-request` tool to send dump and reboot requests to `dumpd` from the SMW. A request includes a comma-separated list of actions to perform, and the node or nodes on which to perform the actions.

A typical request from NHC looks like this:

```
cname: c0-0c1s4n0 actions: halt,dump,reboot
```

A system administrator can define additional actions in the `dumpd.conf` configuration file. To use, execute the `dumpd-request` tool located on the SMW. A typical call would be:

```
dumpd-request -a halt,dump,reboot -c c0-0c1s4n0
```

Or

```
dumpd-request -a myaction1,myaction2 -c
c1-0c0s0n0,c1-0c0s0n1,c1-0c0s0n2,c1-0c0s0n3
```

For this example to work, `myaction1` and `myaction2` must be defined in the `dumpd.conf` file. See the examples in the configuration file for more detail.

Collect Debug Information From Hung Nodes Using the xtnmi Command

CAUTION: This is not a harmless tool to use to repeatedly get information from a node at various times; only use this command when debugging data from nodes that are in trouble is needed. The `xtnmi` command output may be used to determine problems such as a core hang. `xtnmi` will stop a running node. It is best used when a node is not running correctly and debugging information is needed, or to stop a node that is running incorrectly.

The sole purpose of the `xtnmi` command is to collect debug information from unresponsive nodes. As soon as that debug information is displayed to the console, the node panics.

For additional information, see the `xtnmi(8)` man page.

Modify BIOS Parameters

There are a few, rare circumstances where it may be necessary to modify BIOS parameters, for example, in order to troubleshoot a problem, or if there is a need to test a new BIOS version on a small set of nodes before implementing the change across an entire system.

The `xtbiosconf` command allows administrators to specify BIOS parameters at the node, blade, chassis, or cabinet level. BIOS parameters can be associated with a BIOS revision, numeric parameter offset or parameter name, and target nodes. BIOS revision wildcards are supported. The BIOS parameter data is saved in a database on the SMW, and made available automatically to blade controllers via the ERFS file system. In most cases a cold reboot of the affected nodes is needed to apply the new settings.

CAUTION: *Do not* attempt to use this command except under guidance by Cray support personnel, who will provide all the steps for shutting down the nodes, changing the settings, and bringing the nodes back up. Improper use of this command can damage a system.

The following command displays the current BIOS Parameter settings for the entire system:

```
smw~> xtbiosconf --show s0
==============|======|==================================
              | BIOS | BIOS
Node          | REV  | Parameter
==============|======|==================================
c0-1c0s0n1    | 4030 | numlock=1
```

```
c0-1c0s0n1     | 4030 | acpiauto=0
===============|======|===================================
c0-1c0s0n2     | 4030 | numlock=1
c0-1c0s0n2     | 4030 | acpiauto=0
===============|======|===================================
```

For more information see the `xtbiosconf` man page.

Set or Change the HSS Data Store (MariaDB) Root Password

About this task

The method for setting or changing the HSS data store (database) root password has changed with the release of CLE 6.0.

Old The HSS database was implemented with MySQL. After initial installation, its root password was changed from the initial default empty string to a user-defined value by the `SMWconfig` script, which was run after `SMWinstall` and the initial discovery of the system.

New The HSS database is now implemented with MariaDB, a MySQL work-alike database with identically named commands. As before, the initial default root password is the empty string; however, the `SMWconfig` script is no longer used to set it after installation. The administrator must use the following procedure to set the root password to a user-defined value.

After the MariaDB root password has been set, it must be placed in `/root/.my.cnf`, a file readable only by root that has the format shown in step 2. This file is the mechanism by which the installer and the `snaputil` command obtain the root password when they access MariaDB as root. If the file does not exist or it has no `password=` line, the system will attempt to access MariaDB using the default empty-string password, which will fail once the password has been changed.

- Create `/root/.my.cnf` the first time the root password is set to a user-defined value.
- Update `/root/.my.cnf` to match the MariaDB root password whenever it is changed.

Procedure

1. Set or change the MariaDB root password.

   ```
   smw# mysqladmin -uroot password -p
   ```

   a. Enter existing password.

   At the "Enter password" prompt, do ONE of the following:

   - If **setting** the root password for the first time (fresh install, migration, database reinitialization), the existing password is an empty string (the default initial password), so just press **Enter**.

     ```
     Enter password: <cr>
     ```

   - If **changing** the root password, enter the existing password.

     ```
     Enter password: existing_password
     ```

   b. Enter and confirm the new password.

At these prompts, enter the new root password, and then enter it again.

```
New password:
Confirm new password:
```

2. Ensure that the root password in the `/root/.my.cnf` file matches the new root password.

   If this file does not yet exist, create it and add the lines shown in the example, substituting the new password for the placeholder *<MariaDB-password>*.

```
smw# vi /root/.my.cnf
[client]
user=root
password=<MariaDB-password>
```

3. Ensure that only root can see or write to the `/root/.my.cnf` file.

```
smw# chmod 600 /root/.my.cnf
```

Recover from a Corrupt or Missing SMW MariaDB Database

About this task

If the HSS MariaDB (formerly MySQL) database has been damaged or is missing, there are three possible courses of action:

- Repair.

  If the database has become corrupt, MariaDB automatically attempts to repair damaged tables. Look in the log file (default `/var/lib/mysql/machine.err`) for suggested manual recovery steps, if any, and try those first. Repairing the database is the best option when possible.

- Restore and regenerate.

  If there are no suggestions or the suggested steps fail to repair the database, use the procedure *Restore the HSS MariaDB Database from a Backup* on page 113. Restoring the database from the most recent backup (provided a more recent manual backup is not available) will restore the database to its state just prior to the last `xtdiscover` or `warmswap add` operation. An incremental discovery to the present system state will usually be faster than one made from a fresh database, and it will not require administrative state changes made prior to the backup (such as marking compute nodes as 'service') to be performed again.

  TIP: To minimize needed discovery, make more frequent backups:

```
/usr/bin/mysqldump --add-drop-database --routines -uhssds -phssds hssds
 > /home/crayadm/hss_db_backup/my-new-hssds-backup.sql
```

  The HSS MariaDB database could be backed up after every successful warmswap (**`xtdiscover --warmswap`**), regular **`xtdiscover`**, and any administrative state change (e.g., **`xtcli disable/enable/set_empty/mark_node`**). Because these actions are all logged in the commands log, they could be used to automatically trigger backups.

- Regenerate from scratch.

  If all else fails, use the procedure *Regenerate the HSS MariaDB Database from Scratch* on page 114. In this case, the database and the database root password are wiped out, and discovery is used to regenerate the database.

Restore the HSS MariaDB Database from a Backup

About this task

If the HSS MariaDB database becomes corrupt or is missing, and automated attempts to repair damaged tables have failed, use this procedure to do a partial restoration from backup.

Procedure

1. Stop the HSS daemons (by stopping RSMS) and the MariaDB service.

```
crayadm@smw> sudo /usr/bin/systemctl stop rsms.service
crayadm@smw> sudo /usr/bin/systemctl stop mysql.service
```

2. Move the damaged database files out of the database directory.

```
crayadm@smw> mkdir /tmp/backup12
crayadm@smw> cd /var/lib/mysql
crayadm@smw> sudo mv ibdata1 ib_logfile0 ib_logfile1 hssds /tmp/backup12
```

   This procedure assumes that the old database files cannot be repaired; however, this step retains those old database files (just in case) and clears out the database directory.

3. Restart MariaDB.

```
crayadm@smw> sudo /usr/bin/systemctl start mysql.service
```

4. Ensure the database is gone.

```
crayadm@smw> mysql -uhssds -phssds -e "drop database hssds"
```

   If the database is gone, the following error message appears:

```
ERROR 1008 (HY000) at line 1: Can't drop database 'hssds'; database doesn't
exist
```

5. Load the most recent MariaDB backup (from `/home/crayadm/hss_db_backup/`).

```
crayadm@smw> mysql -uhssds -phssds < db_backup.11-17-2014.1120.sql
```

   The backups in `/home/crayadm/hss_db_backup/` are from past runs of `xtdiscover` and `xtwarmswap --add` and were taken *before* the state of the database was updated.

6. Restart the HSS daemons (important!)

```
crayadm@smw> sudo /usr/bin/systemctl start rsms.service
```

7. Run `xtdiscover` to pick up any changes to the system since the backup was taken (or all of the database, if a backup was not loaded in the previous step).

```
crayadm@smw> sudo xtdiscover
```

Regenerate the HSS MariaDB Database from Scratch

About this task

If the HSS MariaDB database becomes corrupt or is missing, and all attempts to repair or restore it have failed, use this procedure to regenerate the database from scratch. Deleting the contents of `/var/lib/mysql` removes everything that stores MariaDB state, including the password (hence the need to re-create it). When MariaDB is restarted and its directory is empty, `/var/lib/mysql` will be re-initialized.

Procedure

1. Stop the HSS daemons (by stopping RSMS) and the MariaDB service.

```
crayadm@smw> sudo /usr/bin/systemctl stop rsms.service
crayadm@smw> sudo /usr/bin/systemctl stop mysql.service
```

2. Remove the damaged database.

```
crayadm@smw> sudo mkdir /var/lib/mysql.bad
crayadm@smw> sudo mv /var/lib/mysql/* /var/lib/mysql.bad
crayadm@smw> sudo mv /var/lib/mysql/.??* /var/lib/mysql.bad
```

   The `/var/lib/mysql` directory is the mount point for a file system from the boot RAID, so it cannot simply be removed. However, its contents can be removed (moved). The `/var/lib/mysql` directory will be newly initialized when the MariaDB service is restarted.

3. Restart MariaDB.

```
crayadm@smw> sudo /usr/bin/systemctl start mysql.service
```

   The database directory is reinitialized, and the default password is set to the empty string.

4. Reset the MariaDB root password and update the `/root/.my.cnf` file.

   a. Reset the MariaDB root password to its former value.

```
smw# mysqladmin -uroot password -p
```

   b. Ensure that the root password in the `/root/.my.cnf` file matches the new root password.

```
smw# vi /root/.my.cnf
[client]
user=root
password=<MariaDB-password>
```

      If this file does not yet exist, create it and add the lines shown in the example, substituting the new password for the placeholder *<MariaDB-password>*.

   c. Ensure that only root can see or write to the `/root/.my.cnf` file.

```
smw# chmod 600 /root/.my.cnf
```

5. Initialize the HSS database tables and restore user permission tables.

```
crayadm@smw> hssds_init
crayadm@smw> dbgrant
```

The system will prompt for a password after each of the above two commands. Give the newly reset MariaDB root password each time.

6. Restart the HSS daemons (important!).

```
crayadm@smw> sudo /usr/bin/systemctl start rsms.service
```

7. Run `xtdiscover` twice (first with the `--bootstrap` option) to regenerate the database.

```
crayadm@smw> sudo xtdiscover --bootstrap
crayadm@smw> sudo xtdiscover
```

Troubleshoot Temperature Warnings Reported in an End Cabinet

About this task

If the consumer log or `xtcheckhss` reports temperature warnings in an end-of-row cabinet of a liquid-cooled system, the current `hss.ini` file may not have the necessary temperature set point defined, or the set point value may not be appropriate for the site. Use this procedure to ensure that this temperature set point is defined and is set to an appropriate value.

Details

In a liquid-cooled cabinet with chassis (cages) that are unevenly populated, the exit temperatures in each cage will be very different. In a normal cabinet, the water valve is controlled by the average temperature of the hottest temperature strip. By contrast, the water valve in an end-of-row cabinet is controlled by the average temperature of all temperature strips. This may lead to inadequate cooling of a populated cage if the other two cages are not populated or have minimal heat load.

To avoid problems arising from inadequate cooling, the exit air temperatures of the end-of-row cabinet can be independently controlled. This is achieved through an entry in the `hss.ini` file that sets the end-of-row cabinet exit temperature lower than that of other cabinets. The default value is 22°C; however this should be adjusted to meet site-specific requirements. If the end cabinet exit air temperature is not defined in the `hss.ini` file, the air temperature will default to the setting defined for the other cabinets in the cooling row.

What to look for

The consumer log may show entries similar to the example below:

```
Mon Jul 28 05:59:47 2014 - rs_event_t at 0x7f5bc0000920
ev_id = 0x080040ed (ec_l1_failed)
ev_src = ::c1-0
ev_gen = ::c0-0c0s0n0
ev_flag = 0x00000002 ev_priority = 0 ev_len = 158 ev_seqnum = 0x00000000
ev_stp = 53d5e6d3.0000176d [Mon Jul 28 05:59:47 2014]
svcid 0: ::c1-0 = svid_inst=0x0/svid_type=0x0/svid_node=c1-0[rsn_node=0x0/
rsn_type=0x3/rsn_state=0x6], err code 65914
- Cabinet Controller Temperature Fault
ev_data...
00000000: 01 00 00 00 00 00 00 00 00 00 00 00 0c 06 00 00 *................*
00000010: 04 00 00 00 00 00 00 00 01 00 00 00 7a 01 01 00 *............z...*
00000020: 7a 00 00 00 30 39 34 7c 57 41 52 4e 7c 54 45 4d *z...094|WARN|TEM*
00000030: 50 7c 2f 64 61 74 61 2f 63 6f 6d 70 75 74 65 5f *P|/data/compute_*
00000040: 63 61 62 69 6e 65 74 2f 61 69 72 5f 73 65 6e 73 *cabinet/air_sens*
00000050: 6f 72 73 2f 63 68 32 2f 61 69 72 5f 74 65 6d 70 *ors/ch2/air_temp*
00000060: 32 3a 64 65 67 63 2a 31 30 30 7c 4d 61 78 69 6d *2:degc*100|Maxim*
00000070: 75 6d 20 73 6f 66 74 20 6c 69 6d 69 74 20 65 78 *um soft limit ex*
```

```
00000080: 63 65 65 64 65 64 21 7c 44 61 74 61 3d 33 30 30 *ceeded!|Data=300*
00000090: 32 7c 4c 69 6d 69 74 3d 33 30 30 30 2e 00       *2|Limit=3000....*
```

With `xtcheckhss`, the problem may look like this:

```
No Version Mismatches Found!
============================================
========== Sensor Warnings =================
============================================
Component Module Sensor HMIN SMIN DATA UNIT SMAX HMAX
--------- ------------ ------------ ---- ---- ---- -------- ---- ----
c2-0 compute_cabinet ambient_temp0 30 50 324 degc*10 300 350
c2-0 compute_cabinet ambient_temp1 30 50 306 degc*10 300 350
c2-0 compute_cabinet ch0_air_temp0 0 1000 3486 degc*100 3000 3500
c2-0 compute_cabinet ch0_air_temp1 0 1000 3355 degc*100 3000 3500
c2-0 compute_cabinet ch0_air_temp2 0 1000 3338 degc*100 3000 3500
c2-0 compute_cabinet ch0_air_temp3 0 1000 3486 degc*100 3000 3500

No SEEP Errors Found!
No ITP Errors Found!
No NTP Time Sync Errors Found!
No Control Errors Found!
```

Procedure

1. Edit `hss.ini`.

   Open the `/opt/tftpboot/ccrd/hss.ini` and look for the following entry.

   ```
   crayadm@smw> vi /opt/tftpboot/ccrd/hss.ini
   ```

   ```
   ## ----------------- END CABINET -----------------
   # This group is used to define the attributes that are only applied to the end
   cabinet
   # of a row. The attributes defined here will override the same attributes in
   group [ccrd]
   # above. If no attributes are defined in this group the end cabinet will be
   configured
   # using the attributes of group [ccrd].
   [endcabinet]
   #define the temperature setpoint for the last cabinet in a row
   temp_setpoint=22
   ```

2. Adjust the value of `temp_setpoint` as appropriate for the installation site.

   To determine an appropriate value, consider the following:

   - The inlet water temperature, which should be below the exit air temperature setting.
   - The facility room environment.

Recover from SMW R630 Boot Disk Hardware RAID5 Failure

If one of the disks in the SMW R630, which is part of the hardware RAID5, fails, the hot spare will take over and the data will be rebuilt using the remaining drives. The bad drive should be removed. When a new disk is inserted into the SMW, the hardware RAID will begin the process of adding it back into the RAID5 set of drives.

This procedure does not apply to the SMW R815 which has software RAID1 for the boot disk.

Recover from SMW R815 Boot Disk Software RAID1 Failure

About this task

If one of the disks in the SMW R815, part of the software RAID1 mirror, fails, corrective action should be taken.

This procedure does not apply to the SMW R630 which has hardware RAID5 for the boot disk.

Procedure

1. Check status of RAID1 filesystems.

   a. Confirm that all RAID1 filesystems are fully synced.

   ```
   smw# cat /proc/mdstat
   ```

   b. Get detailed information on RAID1 devices. `swap` is on `/dev/md125`, `/boot` is on `/dev/md126`, and `/` is on `/dev/md127`.

   ```
   smw# mdadm -D /dev/md125
   smw# mdadm -D /dev/md126
   smw# mdadm -D /dev/md127
   ```

2. Replace the failed disk drive in slot 0 on the SMW.

   a. Shutdown CLE if still booted before the next step of shutting down and booting the SMW itself.

   ```
   crayadm@smw> xtbootsys -s last -a auto.xtshutdown
   ```

   b. Shutdown SMW.

   ```
   smw# shutdown -h now
   ```

   c. Remove the failed disk drive in slot 0 of the SMW so that drive 1 will become the bootable disk.

   d. Boot SMW from drive 1. System boots from drive 1, but calls it `/dev/sda` since it is the first drive found and there is no drive in slot 0.

   e. Remove failed drive from RAID1 configuration.

   ```
   smw# mdadm --manage /dev/md127 --fail /dev/sda1
   smw# mdadm --manage /dev/md127 --remove /dev/sda1
   smw# mdadm --manage /dev/md126 --fail /dev/sda3
   smw# mdadm --manage /dev/md126 --remove /dev/sda3
   smw# mdadm --manage /dev/md125 --fail /dev/sda2
   smw# mdadm --manage /dev/md125 --remove /dev/sda2
   ```

   f. Replace drive 0. The system still runs.

   g. Reboot the SMW.

   ```
   smw# reboot
   ```

   h. Check RAID1 status.

   System boots and immediately will use `/dev/md125` (`swap`) as shown by this command with [`UU`], however, `md126` and `md127` show [`_U`] indicating a degraded state.

```
smw# cat /proc/mdstat
```

`mdadm` shows active sync for both drives in `/dev/md125` (`/dev/sda2` and `/dev/sdb2`).

```
smw# mdadm -D /dev/md125
```

`mdadm` shows removed for drive 0 but active sync for `/dev/sdb1` in `/dev/md127` and `/dev/sdb3` in `/dev/md126`.

```
smw# mdadm -D /dev/md126
smw# mdadm -D /dev/md127
```

i. Partition new drive correctly using `sfdisk` or `fdisk` so it matches drive 1.

```
smw# sfdisk -d /dev/sdb | sfdisk --force /dev/sda
```

j. Add drive 0 back to RAID1 configuration to reconstruct degraded RAID1.

```
smw# mdadm -v --manage /dev/md126 --add /dev/sda3
smw# mdadm -v --manage /dev/md127 --add /dev/sda1
```

k. Check status of RAID1 rebuild with these commands.

```
smw# mdadm -D /dev/md126
smw# mdadm -D /dev/md127
```

Checking `mdstat` will display the percentage of recovery and an estimate of when it will complete for each device being reconstructed.

```
smw# cat /proc/mdstat
```

When all reconstruction is complete, `mdstat` will display the percentage of recovery and an estimate of when it will complete for each device being reconstructed.

```
smw# cat /proc/mdstat
```

3. Replace the failed disk drive in slot 1 of the SMW. If drive 1 is removed, then the process is similar to drive 0 above, but there are differences.

a. Confirm that all RAID1 filesystems are fully synced.

```
smw# cat /proc/mdstat
```

b. Get detailed information on RAID1 devices.

```
smw# mdadm -D /dev/md125
smw# mdadm -D /dev/md126
smw# mdadm -D /dev/md127
```

c. Shutdown CLE, if CLE is still booted, before the next step of shutting down and booting the SMW itself.

d. Shutdown SMW.

```
smw# shutdown -h now
```

e. Remove the failed disk drive in slot 1 of the SMW so that drive 0 will become the bootable disk.

```
smw# mdadm --manage /dev/md127 --fail /dev/sdb1
smw# mdadm --manage /dev/md127 --remove /dev/sdb1
```

```
smw# mdadm --manage /dev/md126 --fail /dev/sdb3
smw# mdadm --manage /dev/md126 --remove /dev/sdb3
smw# mdadm --manage /dev/md125 --fail /dev/sdb2
smw# mdadm --manage /dev/md125 --remove /dev/sdb2
```

f. Boot SMW from drive 0.

g. Replace drive 1. The SMW still runs, but in degraded mode for RAID1 devices. One of the other disks (local to SMW or in boot RAID) will be called `/dev/sdb`.

h. Reboot SMW so that drive 1 will appear as `/dev/sdb`.

```
smw# reboot
```

i. Check RAID 1 status. System boots and, unlike with disk 0 above, will not immediately use `/dev/md125` (`swap`) as shown by this command with [`U_`], also, `md126` and `md127` show [`U_`] indicating a degraded state.

```
smw# cat /proc/mdstat
```

`mdadm` shows removed for drive 1 but active sync for `/dev/sda1` in `/dev/md127` and `/dev/sda3` in `/dev/md/126` and `/dev/sda2` in `/dev/md125`.

```
smw# mdadm -D /dev/md125
smw# mdadm -D /dev/md126
smw# mdadm -D /dev/md127
```

j. Partition new drive correctly using `sfdisk` or `fdisk` so it matches drive 1.

```
smw# sfdisk -d /dev/sda | sfdisk --force /dev/sdb
```

k. Add Drive 1 back to RAID1 configuration.

```
smw# mdadm -v --manage /dev/md125 --add /dev/sdb2
smw# mdadm -v --manage /dev/md126 --add /dev/sdb3
smw# mdadm -v --manage /dev/md127 --add /dev/sdb1
```

l. Check status of RAID1 rebuild with these commands.

```
smw# mdadm -v --manage /dev/md125 --add /dev/sdb2
smw# mdadm -v --manage /dev/md126 --add /dev/sdb3
smw# mdadm -v --manage /dev/md127 --add /dev/sdb1
```

Checking `mdstat` will display the percentage of recovery and an estimate of when it will complete for each device being reconstructed.

```
smw# cat /proc/mdstat
```

When all reconstruction is complete, `mdstat` should show all drives as [`UU`].

```
smw# cat /proc/mdstat
```

About X.509 Certificates and How to Redistribute Them

Some features of Cray XC systems, such as Cray Advanced Platform Monitoring and Control (CAPMC), use X.509 certificate authority files (certificates) for access authorization. These certificates are generated and managed using the `xtmake_ca` tool. The certificate authority (CA) resides on the SMW and is typically generated during

the SMW software installation process; however, there may be occasion to rebuild the CA from scratch. The `xtmake_ca` man page describes how to do this, but it does not provide details about what certificates are used, where they are used, and how to redistribute them after rebuilding a CA from scratch. This topic fills that gap.

Here is a summary; details follow.

What uses certs	Certs used	Where	How redistributed

CAPMC API service	`certificate_authority.crt`
	`certificate_authority.crl`
	`hosts/host.crt`
	`hosts/host.key`
	`client/xtremoted.crt`
	`client/xtremoted.key`
		SMW	reconfigure and restart CAPMC API service

CAPMC SDB node service	`certificate_authority.crt`
	`host/sdb-p0.crt`
	`host/sdb-p0.key`
		SDB node	update and apply config set

DataWarp service	`certificate_authority.crt`
	`/etc/opt/cray/dws/`*`$dw_node_name`*`.crt`
	`/etc/opt/cray/dws/`*`$dw_node_name`*`.key`
		DataWarp service nodes	update and apply config set

`capmc`	`certificate_authority.crt`
	`client/client.crt`
	`client/client.key`
		SMW	move aside existing `capmc` configuration directory and rerun `xtremoted_setup`

In the default set of certificates that follows, file paths are specified relative to the certificate authority directory: `/var/opt/cray/certificate_authority`.

Certificate Authority

Certificates used to maintain the CA include:

`certificate_authority.crt` This is the root certificate in which the SMW CA is based. It is used to validate the authenticity of all other certificates created by the SMW private CA. It must be distributed to all clients and services that use certificates generated by the SMW CA.

`certificate_authority.key` This is the CA private key file, which must be kept private at all times. It must never be distributed to any system.

`certificate_authority.crl` This is an optional certificate revocation list. It is a PEM-encoded certificate containing a list of serial numbers that identify any client access or host certificates that have been revoked. `certificate_authority.crl` is rebuilt each time `xtmake_ca buildcrl` is invoked.

CAPMC API Service

The CAPMC API service runs on the SMW. It is implemented by `nginx`, a standard HTTP server that provides encrypted communications and client authorization, and `xtremoted`, which handles client requests that have been authorized by `nginx`.

Certificates used

The following certificates are used by the HTTP server (`nginx`) on the SMW.

`certificate_authority.crt`
`nginx` uses this certificate to validate that the client access certificate, presented when a client first connects, was issued by the SMW CA. If the certificate was not issued by the local SMW CA, the client is denied access.

`certificate_authority.crl`
If this file exists, the HTTP server checks it for client access certificates that have been revoked. Any client with a revoked certificate is denied access.

`hosts/host.crt`
This is the host certificate used by the HTTP server to enable encrypted communications. It is generated auotmatically at the time of SMW installation, or when a system administrator takes an explicit action to regenerate them using `xtmake_ca`. The Common Name (CN) field of the certificate subject line should match the DNS hostname associated with the SMW. This certificate implements the X509v3 Subject Alternative Name extension, which uses a list of DNS attribute values to specify additional host names that a client should consider valid. The default list of DNS attribute values includes these two elements:

- the fully qualified domain name (FQDN) of the SMW
- the text string literal "`smw`"

`hosts/host.key`
This is the private key associated with the SMW host certificate.

`client/xtremoted.crt`
This is the client access certificate used by `xtremoted` to identify itself to remote procedure call handlers. This is needed because some API calls require `xtremoted` to forward a client's request to another server running on the target partition's system database (SDB) node (see *CAPMC SDB Node Service* below).

`client/xtremoted.key`
This is the private key associated with the client access certificate.

How to redistribute

If the CA has been rebuilt from scratch, `certificate_authority.crl` has been rebuilt, or `hosts/host.crt` has been modified, reconfigure and restart the CAPMC API service (as root):

```
smw# xtremoted_setup
```

This command restarts the CAPMC API service and copies relevant files, with appropriate permissions, into a directory owned by that `xtremoted` userid (`/opt/cray/hss/default/etc/xtremoted`). This copy is necessary because the userid that

the `xtremoted` process is running under does not have read access to files located within the `certificate_authority` directory.

CAPMC SDB Node Service

The CAPMC SDB node service handles remote procedure call requests issued from the CAPMC API service running on the SMW. It is implemented by `nginx`, a front-end HTTP server that performs encryption and client access authorization, and `xtremoted-agent`, a remote procedure call handler that handles the specific request.

Certificates used

The following certificates are used by the HTTP server (`nginx`) on the SDB node.

certificate_authority.crt	`nginx` running on the SDB node uses this certificate to validate that the client access certificate, presented when `xtremoted` issues a remote procedure call request, was issued by the SMW CA. If the certificate was not issued by the local SMW CA, the request is denied. In addition, the CN field of the client access certificate subject line must match the string "`xtremoted`" for the request to be accepted.
hosts/sdb-p0.crt	This is the host certificate for the SDB node and config set `p0`.
hosts/sdb-p0.key	This is the private key associated with the SDB node host certificate and config set `p0`.

How to redistribute

If the CA has been rebuilt from scratch, update the config set and apply it.

1. Update the current configuration set (as root):

   ```
   smw# cfgset update -m auto p0
   ```

   When the config set is updated, the config set gets the new certificates by means of the `xremoted_agent` post-configuration callback script, which updates the certificates from the `/var/opt/cray/certificate_authority` location to the config set being updated. The `xremoted_agent` script is located in this directory:

   ```
   /opt/cray/imps_config/system-config/default/configurator/callbacks/post/xtremoted_agent.py
   ```

2. Reboot the system. When the node boots, the config set certificate files are copied from the config set to the node using an Ansible play.
3. After the Ansible play has run, verify that the certificates have been distributed.

   ```
   smw> ls -la /var/opt/cray/imps/config/sets/p0/files/roles/common/etc/opt/
   cray/xtremoted-agent
   total 12
   drwxr-xr-x 1 root root   90 Dec  7 15:39 .
   drwxr-xr-x 1 root root   42 Dec  7 15:39 ..
   -rw------- 1 root root  956 Dec  9 11:18 certificate_authority.crt
   -rw------- 1 root root 3002 Dec  9 11:18 sdb-p0.crt
   -rw------- 1 root root  916 Dec  9 11:18 sdb-p0.key
   ```

DataWarp Service Nodes

DataWarp service nodes (and elogin and compute nodes as well) use the SSL certificates only to connect to the HTTP API. The client certificates are not essential because they can be regenerated. What is essential is that the CA on the SMW is trusted on the remote nodes.

Certificates used

The following certificates are used primarily at the login node and any elogin node. Copies of the cert chain are made so that client compute nodes and service nodes are able to run tools that interact with the DataWarp API with no problems.

certificate_authority.crt	This file is synced with the certificate on the DataWarp service.
hosts/$dw_node_name.crt	This file is synced with the certificate on the DataWarp service.
hosts/$dw_node_name.key	This file is synced with the certificate on the DataWarp service.
/etc/opt/cray/dws/$dw_node_name.crt	This is the certificate on the DataWarp service.
/etc/opt/cray/dws/$dw_node_name.key	This is the private key on the DataWarp service.

How to redistribute

Certificates are deployed initially by means of the configurator and Ansible plays when the DataWarp service is set up. The Ansible plays generate the certificates using `xtmake_ca` and synchronize the certificate authority to the remote nodes as needed. If the CA has been rebuilt from scratch, update the config set and apply it.

1. Update the current configuration set (as root):

   ```
   smw# cfgset update -m auto p0
   ```

   When the config set is updated, the config set gets the new certificates by means of a post-configuration callback script, which updates the certificates from the `/var/opt/cray/certificate_authority` location to the config set being updated.

2. Reboot the system. When the node boots, the config set certificate files are copied from the config set to the node using an Ansible play.

Troubleshooting

Problem:

- `capmc` outputs a hostname mismatch error.

```
smw:/etc/opt/cray/capmc # capmc node_rules
    Error - Certificate identity does not match the target hostname
```

Possible Causes:

- The `capmc` client configuration, (`/etc/opt/cray/capmc/capmc.json`) `os_service_url`, setting is invalid.

  When capmc is being executed from the SMW on an internal Cray service node running the Cray Linux Environment, the `os_service_url` setting should configured as follows:

```
https://smw:8443
```

When `capmc` is being executed from an external system, the `os_service_url` setting should include the fully qualified domain name of the SMW as follows:

```
https://my-smw.my-domain.com:8443
```

- Resolution:
  - Reconfigure the `os_service_url` parameter.

- The SMW capmc API server host certificate contains an incorrect list of acceptable DNS names.

  Verify the "Subject Alternative Name" DNS name list contains the SMW FQDN and short hosname `smw`:

```
smw:/etc/opt/cray/capmc # openssl x509 -text -noout \
                -in /var/opt/cray/certificate_authority/hosts/host.crt | \
                grep -A 1 "Subject Alternative Name"

X509v3 Subject Alternative Name:
                DNS:my-smw.my-domain.com, DNS:smw
```

  - Resolution:
    - Regenerate the SMW host server certificate.

Problem:

- `capmc` outputs a certificate verification error.

```
smw:/etc/opt/cray/capmc # capmc node_rules
    Error - url(https://smw:8443/capmc/get_node_rules) \
    [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:581)
```

Possible Causes:

- The client's copy of the CA certificate is not from the actual certificate authority that generated the SMW CAPMC API server certificate.
  - Resolution:
    - Redistribute the certificate_authority.crt file from the SMW to the client system.
- The SMW CAPMC API server was not restarted after regenerating the certificate authority from scratch.
  - Resolution:
    - Reconfigure the capmc API server by invoking `xtremoted_setup`.

Problem

- Capmc client connection times out. IP connectivity is non-functional between the `capmc` client system and the SMW.

```
smw:/etc/opt/cray/capmc # capmc node_rules
    Error - url(https://smw:8443/capmc/get_node_rules) \
    [Errno 113] No route to host
```

Possible Causes:

- `capmc` client `os_service_url` is configured incorrectly.
  - Resolution:

  - For use on internal Cray service nodes, reconfigure the `os_service_url` to `https://smw:8443`.
  - For use on external nodes, reconfigure the `os_service_url` to be the SMW's fully qualified domain name and verify that a valid IP connectivity path is established.

- When using `capmc` from an internal Cray service node, the IP path taken is over the high speed network, to the boot node, and on the SMW. IP Routing tables may be misconfigured on the SMW, boot node, or internal service node.
  - Resolution:
    - Verify the boot node has IP forwarding enabled.

      ```
      boot-p0:~ # sysctl net.ipv4.ip_forward
                    net.ipv4.ip_forward = 1
      ```

    - Verify the boot node firewall has TCP port 8443 open.

      ```
      boot-p0:~ # iptables -L
                    ...
      ```

    - Verify the SMW has a return route on an internal interface to the high speed network via the boot node.

      For example:

      ```
      smw:/etc/opt/cray/capmc # netstat -rn
                    Kernel IP routing table
                    Destination Gateway    Genmask              Iface
                    ...
                    10.128.0.0  10.3.1.254 255.255.0.0 UG 0 0 0 eth3
                    ...
      ```

    - Verify the internal Cray service node has a route to the SMW's internal interface via the boot node.

      For example:

      ```
      svc-node:~ # netstat -rn
                    Kernel IP routing table
                    Destination Gateway         Genmask ...
      Iface
                    ...
                    10.3.1.1    10.128.255.254  255.255.255.255 UGH 0 0 0
      ipogif0
                    ...
      ```

Update X.509 Host Certificate After SMW Hostname Change

About this task

Whenever the SMW hostname changes, the previously generated x509 SMW host certificate `host.crt` file will need to be updated. Failure to perform this step will prevent the `capmc` client from connecting to the SMW, due to a host name certificate validation error.

Procedure

1. Create a backup copy of the `certificate_authority` directory.

```
smw:~# cd /var/opt/cray
smw:~# cp -a certificate_authority certificate_authority.backup
```

2. Run the host validation.

```
smw:~# xtmake_ca validate
```

You will be notified if the `host.crt` file has a common name that does not match the current hostname.

```
..
 - CN in SMW host file matches current hostname (my-smw.example.com)
Alternate names: my-smw.example.com, smw  - SMW host certificate file
validation succeeded.
..
```

3. Generate the new certificate.

- If the SMW was only renamed, rebuild the host certificate using the new hostname.

```
smw:~# xtmake_ca update
```

- If a specific SMW hostname or list of alternate names must be specified, manually revoke the SMW host server certificate and recreate it with a list of appropriate hostnames.

```
smw:~# xtmake_ca revoke \
/var/opt/cray/certificate_authority/hosts/host.crt
smw:~# xtmake_ca CN=my-smw.example.com,my-smw.local,my-smw
```

> **NOTE:** This does not require remaking or redistributing existing certificates. `xtmake_ca` will recreate only missing certificates. In this case, the only missing certificate should be SMW host certificate which was intentionally revoked. Any services, such as `nginx`, running on the SMW which are using the rebuilt host certificate should be restarted.

This step generates a new host certificate with the currently assigned hostname is listed in the **CN** field, as well as a list of additional DNS names which `capmc` should consider valid.

4. Run the host validation again.

```
smw:~# xtmake_ca validate
```

5. Reconfigure and restart `nginx` on the SMW.

```
smw:~# xtremoted_setup
```

6. View the contents of the newly generated SMW host server certificate.

```
smw:~# openssl x509 -noout -text -in
/var/opt/cray/certificate_authority/hosts/host.crt
```

Manage System Access

Change Account Passwords on the SMW

About this task

The SMW contains its own `/etc/passwd` and `/etc/shadow` files that are separate from the files for the rest of the CLE system.

Procedure

Execute the following commands to change the passwords on the SMW for the following Linux accounts.

```
smw# passwd root
smw# passwd crayadm
smw# passwd mysql
```

Change Account Passwords on CLE Nodes

About this task

Use this procedure to change a password for an account that is local to the CLE nodes, such as `root` and `crayadm`.

For LDAP or other authentication services, passwords are changed through those services.

Procedure

1. Update passwords in `cray_local_users`.

   a. Update the CLE config set to change passwords for `root` (`cray_local_users.settings.users.data.root.crypt`) and `crayadm` (`cray_local_users.settings.users.data.crayadm.crypt`).

      Full system:

      ```
      smw# cfgset update -s cray_local_users -l advanced -m interactive p0
      ```

      Partitioned system (update a config set for each partition):

      ```
      smw# cfgset update -s cray_local_users -l advanced -m interactive p1
      smw# cfgset update -s cray_local_users -l advanced -m interactive p2
      ```

2. Validate config set.

   Full system:

```
smw# cfgset validate p0
```

Partitioned system:

```
smw# cfgset validate p1
smw# cfgset validate p2
```

3. Activate new passwords for local accounts. The password changes can be made immediately on the CLE nodes or can take effect at the next boot of the nodes.

   a. Activate new passwords immediately on nodes. Doing so immediately does not require a reboot of the node, merely running `cray-ansible` again.

      On the boot node:

      ```
      boot# /etc/init.d/cray-ansible start
      ```

      On the SDB node:

      ```
      sdb# /etc/init.d/cray-ansible start
      ```

      On all service nodes:

      ```
      sdb# pcmd -r -n ALL_SERVICE_NOT_ME "/etc/init.d/cray-ansible start"
      ```

      On all compute nodes:

      ```
      sdb# pcmd -r -n ALL_COMPUTE "/etc/init.d/cray-ansible start"
      ```

4. Activate new passwords by rebooting nodes. Either a full system reboot or warm booting individual nodes will cause `cray-ansible` to activate these new passwords on the CLE nodes.

Configure the System

Cray XC System Configuration

To configure Cray XC systems and manage configuration content, system administrators use the Cray configuration management framework (CMF). The CMF comprises configuration data, the tools to manage and distribute that data, and software to apply the configuration data to the running image at boot time. Its major components include configuration service packages, config sets, the IMPS distribution service (IDS), the configurator, cray-ansible, and Ansible.

Configuration Starts with Configuration Service Packages

Configuration content (data and software) is installed as configuration service packages on the management node of Cray XC systems (in `/opt/cray/imps_config/<service package>/default/configurator` by default). Each service package delivers configuration content for one or more system services. The contents of each service package reside in the following subdirectories:

`ansible`	Drop zone for Cray-provided Ansible play content.
`callbacks`	Pre- and post-configuration scripts.
`dist`	Drop zone for other Cray-provided content, such as static files required for the configuration of a service.
`template`	Configuration templates that define the configuration settings to be set and provide some default values. These templates are never modified by administrators or other users.

Configuration service packages are installed for system upgrades and updates as well as for initial installation.

Configuration Information is Stored in Config Sets

Administrators use the `cfgset` command to manage configuration information. It takes configuration content delivered in service packages and invokes the *configurator* tool to combine that content with site-specific configuration content gathered from administrators either interactively or through bulk import. The results are used by `cfgset` to create a configuration set or *config set*. A config set is a central repository that stores all configuration information necessary to operate the system. Config sets reside on the management node (e.g., the SMW) in `/var/opt/cray/imps/config/sets` by default. The contents of each config set reside in the following subdirectories:

`ansible`	Drop zone for local site-provided Ansible play content to be distributed with the config set. When the config set is created, `cfgset` copies Ansible content from service packages to this location. Whenever the config set is updated, `cfgset` copies Ansible content from service packages again, overwriting the previous service-package Ansible content and leaving the site-provided content unchanged.
`changelog`	YAML change logs from previous sessions with the configurator.

config	Configuration templates containing configuration information. When the config set is created, the configurator copies service package templates to this location. Administrators can modify the content of these templates using `cfgset` and the configurator. Whenever the config set is updated, the configurator merges service package templates with the templates in this location.
dist	Drop zone for other site-provided content, such as static files required for the configuration of a service. When the config set is created, `cfgset` copies dist content from service packages to this location. Whenever the config set is updated, `cfgset` copies dist content from service packages again, overwriting the previous service-package dist content and leaving the site-provided content unchanged.
files	Files necessary for system configuration that are generated by configuration callback scripts or manually and distributed with the config set (e.g., `/etc/hosts`).
worksheets	Configuration worksheets generated by the configurator using data stored in the configuration templates in the `config` subdirectory of the config set. Administrators copy these worksheets to a location outside the config set, edit them with site-specific configuration data, and then import them to create a new config set or update an existing one.

An administrator may create multiple config sets to support partitions or alternate configurations. Typically a config set of type `cle` is created for each partition to store partition- and CLE-specific content, and another config set of type `global` is created to store management node and global configuration data.

IDS Distributes Config Sets to Nodes

IDS, a read-only network share of content from the management node to the rest of the system, distributes config sets to every node in the system. All config sets are shared throughout the system, but only one `cle` config set is active on a given node at a time (in addition to an active `global` config set, which is applied to the entire system). Currently, IDS leverages the 9P network file system and the Linux automounter facility as its distribution mechanism; however, the content and use of the config sets is independent of the distribution mechanism.

Ansible Plays Apply Configuration during System Boot

Prior to booting the system, each node will have an image, the `global` config set, and the `cle` config set. When the system boots, each node boots an unconfigured software image. Then Ansible plays, which can be located in both the image and the config set (config set is the preferred location for site-supplied Ansible plays), apply configuration to that image, bringing up the services pertinent to each node.

Administrators Configure/Reconfigure the System on an Ongoing Basis

Configuration happens at times other than initial installation. New configuration service packages can be installed during system upgrades and updates, sites can decide to enable a new service or change the configuration of an existing service, and so forth. In all of these scenarios, an administrator uses the `cfgset` command to manage config sets and the `cray-ansible` script to apply any configuration changes. The `cfgset` command and its associated subcommands and options enable administrators to perform a variety of operations on config sets in addition to create and update, such as search, diff, list, show, validate, push, and remove. See the `cfgset` man page for a description of its subcommands and options and some examples of each.

About the Configurator

The configurator plays a major role in Cray XC system configuration. The configurator gathers configuration data from several sources (including the user, with helpful prompts and default values), merges and validates it, and

stores it in a central location on the management node, where it is used during boot to configure the entire system. The configurator is invoked by the `cfgset` command to:

- handle all configuration template and worksheet operations
- perform steps 4, 5, and 6 of the *Config Set Create/Update Process*, including providing a user interface to gather and modify configuration data interactively or through the import of configuration worksheets

The configurator is invoked with the `cfgset` subcommands `create` (except when the `--clone` option used) and `update`. It is invoked also with the `search` subcommand, because that involves searching data stored in the configuration templates, but no changes are made to the config set using `search`. The options selected for the `create` and `update` subcommands determine the mode in which the configurator is run (with or without user interaction), which settings can be viewed and set by a user, and whether callback scripts are run before and after the configurator session. The configurator is not involved when the remaining `cfgset` subcommands are used: `diff`, `list`, `push`, `remove`, `show`, and `validate`. See the `cfgset` man page for a description of its subcommands and options and some examples of each, or use `cfgset` *`SUBCOMMAND`* `-h` to see information about just one of the subcommands.

Choose How to Interact with the Configurator: Modes

The `mode` option of the `cfgset` command determines how the configurator interacts with a user. Mode can be specified only with subcommands `create` and `update`.

`--mode` | `-m` Possible values: `auto` (default), `interactive`, `prepare`

`auto` The configurator searches through all available configuration templates in the config set and automatically presents all configuration settings that meet state and level filtering criteria. It presents the configuration settings in a certain order (taking into account dependencies among services) one at a time until all have been presented to the user, and then it automatically ends the session and saves the config set.

`interactive` The configurator searches through templates as with auto mode, but in interactive mode, it presents a menu of all available services (or a menu of all available settings, when a service has been selected) that meet state and level filtering criteria. This mode enables the user to navigate through the services and settings to view and modify the settings as needed. The configuration session ends when the user exits the session. The user chooses whether to save any changes to the config set upon exit.

`prepare` The configurator prepares configuration worksheets, one for each service. Each worksheet contains all configuration settings (unfiltered) for that service, and the worksheet can be edited offline and then imported later to create or update a config set. In this mode, the configurator does not open an interactive session with the user.

Choose What to See with the Configurator: Filters

Two `cfgset` command options act as filters to determine which settings are available to view and set or update. These options can be specified only with subcommands `create`, `update`, and `search`.

`--state` | `-S` Possible values: `unset` (default), `set`, `all`

`--level` | `-l` Possible values: `required`, `basic` (default), `advanced`

required Settings that must be set or the system will not function. The config set will not validate if any required settings are skipped (i.e., left unset). Specify level `required` in a `cfgset` command to filter for required settings only.

basic Settings that are likely to be used by most sites. If a `basic` setting is left unset, the template-provided default is used. Specify level `basic` in a `cfgset` command to filter for both basic and required settings.

advanced Settings that are likely to be used only by advanced users to tune a service. If an `advanced` setting is left unset, the template-provided default is used. Specify level `advanced` in a `cfgset` command to filter for all settings: advanced, basic, and required.

Create a Config Set

Choosing the best strategy for creating a config set depends on the circumstances ("when to use"):

Strategy	When to use	Rationale
Create a Config Set from Configuration Worksheets	when performing fresh installs, major upgrades, or any time there is a large amount of configuration data to set up	Worksheets can be generated, filled out offline with site-specific data by the appropriate staff, and then imported when needed.
Create a Config Set by Cloning	when there is already a config set with site-specific data and additional config sets are needed with minor variations (for partitions, alternate configurations, etc.), or when manually backing up a config set	Cloning is quick, and it is easy to interactively update the clone with needed variations.
Create a Config Set without Callbacks	when no hardware is attached to the XC system, as in some testing scenarios	Pre- and post-configuration callback scripts may invoke utilities that query hardware in order to provide additional config set content.
Create a Config Set Interactively	when configuring a smaller system with little configuration data to change	Setting all configuration values one at a time in response to a series of prompts or when selected from a menu can be very time-consuming.

These strategies all use the `cfgset` command. Use `cfgset create -h` for information about the create subcommand. See *Config Set Create/Update Process* on page 142 for an outline of the process followed by `cfgset` each time the `create` or `update` subcommand is used.

Note that when the `create` subcommand is used in any of these strategies (except cloning), it is necessary to specify the config set type for any type other than the default `cle`. Most of the following `create` procedures omit `--type` because they are for config sets of type `cle`.

REMEMBER: Run `cfgset` as root.

CAUTION: Boot failure possible if using `cfgset` under certain conditions.

The `cfgset create` and `cfgset update` commands always call pre- and post-configuration scripts. Some of these scripts require HSS daemons and other CLE services to be running. This can cause problems under these conditions:

- If `xtdiscover` is running, `cfgset` may hang or produce incorrect data that can result in system boot failure.
- If `xtbounce` is in progress or if the SMW is not connected to XC hardware, `cfgset` will fail.

In these circumstances, use the `--no-scripts` option with `cfgset create` or `cfgset update` to avoid running the scripts. Because using that option results in an invalid config set, remember to run `cfgset update` without the `--no-scripts` option afterwards, when circumstances permit, to ensure that all pre- and post-configuration scripts are run.

For more information on creating a config set using `--no-scripts`, see *Create a Config Set without Callbacks* on page 135

Create Backup Config Sets Automatically

If the `auto_clone` option in the IMPS configuration file (`/etc/opt/cray/imps/imps.json`) is enabled, the `cfgset create` and `cfgset update` commands will automatically clone a config set as a backup upon successful creation/update of the original config set. A failed operation will not create a backup.

The `autosave_limit` parameter in the IMPS configuration file determines how many clones will be retained. Config set backups are rotated with the oldest backup removed as a new backup is generated. Config set backups are saved with names of the form *`CONFIGSET`*`-autosave-`*`YYYY-MM-DDTHH:mm:SS`*, where *`CONFIGSET`* is the name of the original config set.

Create a Config Set from Configuration Worksheets

Prerequisites

This procedure has no prerequisites.

About this task

Use this procedure when performing fresh installs, major upgrades, or any time there is a large amount of configuration data to set up. To create a config set from configuration worksheets, use this process:

1. Generate the worksheets.
2. Copy the worksheets to a new location on the management node.
3. Edit the worksheets.
4. Import the worksheets.

The detailed steps of this procedure show an example of how to create config set `p0` of type `cle` (default) from configuration worksheets.

Note that the `cfgset` command is run as root.

Procedure

1. Generate new worksheets from configuration service packages installed on the system.

```
smw# cfgset create --mode prepare p0
```

2. Locate the newly generated worksheets and copy them to a new location.

```
smw# cfgset show --fields path p0
p0:
  path: /var/opt/cray/imps/config/sets/p0

smw# cp /var/opt/cray/imps/config/sets/p0/worksheets/* /some/edit/location
```

3. Edit the worksheets to customize them for this site.

   The system administrator typically distributes them to site staff members with knowledge about the services being configured so that they can edit the worksheets and enter appropriate values. Each worksheet is a YAML file that contains instructions on how to edit it; the basic idea is to locate the settings of interest, uncomment them, and either retain or change the default setting (if provided).

4. Import the completed worksheets using `cfgset update` or `cfgset create`.

   Import the completed worksheets by updating the config set created when the worksheets were generated originally or by creating an entirely new config set. The argument to the `--worksheet-path` option is a file glob to allow multiple worksheets to be imported in a single create/update operation. Full paths to single worksheets can also be used.

   - Import to the config set created with `--mode prepare` in step 1.

     ```
     smw# cfgset update --worksheet-path '/some/edit/location/*_worksheet.yaml' p0
     ```

   - Import to a new config set.

     ```
     smw# cfgset create --worksheet-path '/some/edit/location/*_worksheet.yaml' \
      p0-new
     ```

     REMEMBER: When importing worksheets using `cfgset` with the `--worksheet-path` option,

     - Always add single quote marks around the worksheet path if a wildcard is used (e.g.,`*_worksheet.yaml`).
     - Do not add mode, state, level, or service options; the configurator ignores them for worksheet import.
     - The *type* of the config set must match the *type* of the worksheets being imported.

Create a Config Set by Cloning

Prerequisites

This procedure assumes that the config set to be cloned (the original) already exists.

About this task

Use this procedure when there is already a config set with site-specific data and additional config sets are needed with minor variations (for partitions, alternate configurations, etc.), or when manually backing up a config set. This procedure shows an example of creating config set `p0-new` by cloning it from existing config set `p0`. No callback scripts or configurator sessions occur when cloning a config set. The clone will have the same config set type as the original.

Note that the `cfgset` command is run as root.

Procedure

Create a clone using the `--clone` option.

```
smw# cfgset create --clone p0 p0-new
```

The configurator is not invoked when the `--clone` option is used, so no configurator session occurs, and no changes are made to the configuration data in the original config set.

Create a Config Set without Callbacks

Prerequisites

This procedure has no prerequisites.

About this task

Pre- and post-configuration callback scripts may invoke utilities that query hardware in order to provide additional config set content. Use this procedure when no hardware is attached to the XC system, as in some testing scenarios. This procedure shows an example of creating config set `global0` of type `global` from worksheets while skipping all callback scripts. The `--no-scripts` option can also be used when creating a config set interactively.

Note that the `cfgset` command is run as root.

Procedure

Create a config set without callbacks.

```
smw# cfgset create --no-scripts --worksheet-path \
'/some/edit/location/*_worksheet.yaml' --type global global0
```

CAUTION: Skipping callback script processing invalidates a config set. A config set cannot be considered validated unless it is updated successfully without the `--no-scripts` option. Update all config sets to run the callback scripts before using the config set with the system.

Create a Config Set Interactively

Prerequisites

This procedure has no prerequisites.

About this task

This procedure shows examples of creating config set `p0` of type `cle` interactively. For additional examples, use `cfgset create -h`.

Note that the `cfgset` command is run as root.

Procedure

Invoke the configurator in auto mode (default) or interactive mode.

- **Auto mode.**

  To be presented with all settings with state `unset` (default) and level `basic` (default) in all services in config set `p0`:

  ```
  smw# cfgset create p0
  ```

  To be presented with all settings (any state and any level) in all services in config set `p0`:

```
smw# cfgset create --state all --level advanced p0
```

- **Interactive mode.**

  To display a menu of services in config set `p0` that have configuration settings with state `unset` (default) and level `basic` (default):

  ```
  smw# cfgset create --mode interactive p0
  ```

  To display a menu of all services (with settings of any state and any level):

  ```
  smw# cfgset create --mode interactive --state all --level advanced p0
  ```

Update a Config Set

Choosing the best strategy for updating a config set depends on the circumstances ("when to use"):

Strategy	When to use	Rationale
Update a Config Set Interactively	when one or more config sets require a few changes (e.g., cloned config sets that need to be adjusted for a particular purpose), when a software update introduces just a few new fields to configure, or to confirm that all required and basic settings have been set (very useful!)	Setting just a few configuration values one at a time in response to a series of prompts or when selected from a menu works well when there are just a few settings that need to be configured or updated.
Update a Config Set from Configuration Worksheets	when performing system upgrades and updates, or any time there is a large amount of configuration data to change	Worksheets can be generated, filled out offline with site-specific data by the appropriate staff, and then imported when needed.
Update a Config Set without Callbacks	when no hardware is attached to the XC system, as in some testing scenarios	Pre- and post-configuration callback scripts may invoke utilities that query hardware in order to provide additional config set content.
Rename a Config Set	when a config set needs to be renamed as well as updated, or just renamed	This could become necessary for a variety of reasons.
Update a Single Service in a Config Set	when setting up a new service, or when just one service requires modification	This can be done either interactively or with worksheets, so refer to those circumstances and rationales for the right strategy.

These strategies all use the `cfgset` command. Use `cfgset update -h` for information about the update subcommand. See *Config Set Create/Update Process* on page 142 for an outline of the process followed by `cfgset` each time the `create` or `update` subcommand is used.

CAUTION: Boot failure possible if using `cfgset` under certain conditions.

The `cfgset create` and `cfgset update` commands always call pre- and post-configuration scripts. Some of these scripts require HSS daemons and other CLE services to be running. This can cause problems under these conditions:

- If `xtdiscover` is running, `cfgset` may hang or produce incorrect data that can result in system boot failure.

- If `xtbounce` is in progress or if the SMW is not connected to XC hardware, `cfgset` will fail.

  In these circumstances, use the `--no-scripts` option with `cfgset create` or `cfgset update` to avoid running the scripts. Because using that option results in an invalid config set, remember to run `cfgset update` without the `--no-scripts` option afterwards, when circumstances permit, to ensure that all pre- and post-configuration scripts are run.

For information on updating a config set using `--no-scripts`, see *Update a Config Set without Callbacks* on page 139

Update a Config Set Interactively

Prerequisites

This procedure assumes an existing config set needs to be updated.

About this task

Use this procedure when one or more config sets require a few changes (e.g., cloned config sets that need to be adjusted for a particular purpose), or to confirm that all required and basic settings have been set (very useful!). To update just one service in a config set, see *Update a Single Service in a Config Set* on page 140.

`cfgset` has two modes that initiate an interactive configurator session: `auto` (default) and `interactive`. This procedure shows examples of updating config set `p0` of type `cle` interactively in either mode. For additional examples, use `cfgset update -h`.

Note that the `cfgset` command is run as root.

Procedure

Invoke the configurator in auto mode (default) or interactive mode.

- **Interactive mode.**

  To display a menu of services in config set `p0` that have configuration settings with state `unset` (default) and level `basic` (default):

  ```
  smw# cfgset update --mode interactive p0
  ```

  To display a menu of services in config set `p0` that have configuration settings with level `required` and state `unset`:

  ```
  smw# cfgset update --mode interactive --level required p0
  ```

  To display a menu of all services in config set `p0`, use the broadest state and level filters:

  ```
  smw# cfgset update --mode interactive --state all --level advanced p0
  ```

- **Auto mode.**

  To confirm that all required and basic settings have been set (in which case, the configurator will not initiate an interactive session) or to be presented with all settings with state `unset` (default) and level `basic` (default) in all services in config set `p0`:

  ```
  smw# cfgset update p0
  ```

  For a discussion of common outcomes of this command, see *cfgset Troubleshooting Tips* on page 151.

To be presented with all settings in config set `p0`, use the broadest state and level filters:

```
smw# cfgset update --state all --level advanced p0
```

Update a Config Set from Configuration Worksheets

Prerequisites

This procedure assumes an existing config set needs to be updated.

About this task

Use this procedure when performing system upgrades and updates, or any time there is a large amount of configuration data to change. The configurator overwrites all data in a service with the contents of the worksheets specified on the command line. If a worksheet with stale data is used to update the config set, data loss may occur. To ensure that the worksheets used to update the config set are as up-to-date as possible, use this process:

1. Generate worksheets from the current config set.
2. Copy the worksheets to a new location on the management node.
3. Edit the worksheets.
4. Import the worksheets to the current config set.

The detailed steps of this procedure show an example of how to update config set `p0` of type `cle` (default) from configuration worksheets. To update just one service in a config set, see *Update a Single Service in a Config Set* on page 140.

Note that the `cfgset` command is run as root.

Procedure

1. Generate new worksheets from configuration service packages installed on the system and config set `p0`.

   ```
   smw# cfgset update --mode prepare p0
   ```

2. Locate the newly generated worksheets and copy them to a new location on the management node.

   ```
   smw# cfgset show --fields path p0
   p0:
     path: /var/opt/cray/imps/config/sets/p0

   smw# cp /var/opt/cray/imps/config/sets/p0/worksheets/* /some/edit/location
   ```

3. Edit one or more worksheets to make the needed changes.

   To edit the worksheets, open those with settings that need to be changed and make changes, as needed. Each worksheet is a YAML file that contains instructions on how to edit it.

4. Import the completed worksheets to `p0` using `cfgset update`.

   ```
   smw# cfgset update --worksheet-path '/some/edit/location/*_worksheet.yaml' p0
   ```

   The argument to the `--worksheet-path` option is a file glob to allow multiple worksheets to be imported in a single create/update operation. Full paths to single worksheets can also be used. The configurator will

replace config set data with imported worksheet data only for services that have matching worksheets provided on the command line.

REMEMBER: When importing worksheets using `cfgset` with the `--worksheet-path` option,

- Always add single quote marks around the worksheet path if a wildcard is used (e.g.,`*_worksheet.yaml`).
- Do not add mode, state, level, or service options; the configurator ignores them for worksheet import.
- The *type* of the config set must match the *type* of the worksheets being imported.

Update a Config Set without Callbacks

Prerequisites

This procedure assumes an existing config set needs to be updated.

About this task

Pre- and post-configuration callback scripts may invoke utilities that query hardware in order to provide additional config set content. Use this procedure when no hardware is attached to the XC system, as in some testing scenarios. This procedure shows an example of updating config set `p0` of type `cle` interactively while skipping all callback scripts. The `--no-scripts` option can also be used when updating a config set from worksheets.

Note that the `cfgset` command is run as root.

Procedure

Update a config set without callbacks.

```
smw# cfgset update --no-scripts p0
```

CAUTION: Skipping callback script processing invalidates a config set. A config set cannot be considered validated unless it is updated successfully without the `--no-scripts` option. Update all config sets to run the callback scripts before using the config set with the system.

Rename a Config Set

Prerequisites

This procedure assumes an existing config set.

About this task

Use this procedure when a config set needs to be renamed or updated as well as renamed. The renaming operation follows the same basic configurator flow as a regular update but renames the config set prior to other processing. If auto-cloning is enabled, config set backups of the original config set will not be renamed. This procedure shows an example of renaming config set `p0`.

Note that the `cfgset` command is run as root.

Procedure

Rename a config set using the `update` subcommand with the `--rename` option.

```
smw# cfgset update p0 --rename p0.new
```

Note that the config set being operated on (`p0` in this example), does not have to be the last argument on the command line.

Update a Single Service in a Config Set

Prerequisites

This procedure assumes an existing config set.

About this task

Use this procedure when setting up a new service, or when just one service requires modification. This procedure provides examples of updating a single service at a time instead of the entire config set, and it can be done either interactively or using a configuration worksheet.

Procedure

Update a single service in config set `p0`.

- **Update interactively: use the `--service` option.**

  IMPORTANT: For a service with configuration template file `cray_example_config.yaml`, use only the `cray_example` portion on the command-line when specifying a single service.

  To display a menu of settings in the `cray_example` service in config set `p0` that are level `required` and any state (default for interactive mode when only one service is specified):

  ```
  smw# cfgset update --service cray_example --mode interactive \
  --level required p0
  ```

  To display a menu of all settings (with settings of any state and any level):

  ```
  smw# cfgset update --service cray_example --mode interactive \
  --level advanced p0
  ```

  To be presented with all settings (with settings of any state and any level):

  ```
  smw# cfgset update --service cray_example --state all --level advanced p0
  ```

- **Update with a worksheet: use the `--worksheet-path` option.**

  To update the service using a worksheet, use the `--worksheet-path` option instead of `--service`. Unlike the `--service` option, with the `--worksheet-path` option it is necessary to provide the full path to the worksheet for that service, which includes the `_worksheet.yaml` portion.. The configurator will replace only the config set data that corresponds to the data in the worksheet being imported.

  ```
  smw# cfgset update --worksheet-path \
  /path/to/worksheets/cray_example_worksheet.yaml p0
  ```

Validate a Config Set and List Validation Rules

It is important to validate any config set that has been modified, because there is currently no mechanism to prevent the system from trying to use an invalid config set. Validation is useful for determining if the config set is minimally viable for use with the system it is intended to configure.

IMPORTANT: Validation ensures that a config set passes all rules stored on the system. A validated config set does not necessarily equate to a config set with configuration data that will result in a properly configured system.

When validating a config set, the configurator checks the following:

- Config set has the proper directory structure and permissions.
- All configuration templates have correct YAML syntax.
- All configuration templates adhere to the configurator schema.
- All fields of type `lookup` reference values and settings that exist in the available configuration services.
- All level `required` fields in enabled services are configured (i.e., their state is `set`).
- Pre-configuration and post-configuration callback scripts ran successfully during the latest config set update.
- `cfgset validate` has run all validation rules installed on the system.

Validate a Config Set with the `validate` Command

To validate a config set, use the `cfgset validate` command:

```
smw# cfgset validate p0
```

The `cfgset validate` command runs all rules installed on the system. Users may specify which rules to include or exclude by using the rules file in `/etc/opt/cray/imps/rules.yaml`.

The `--no-rules` subcommand can be used to prevent the `cfgset` from executing any validation rules against the config set. All other validation checks will be done.

```
smw# cfgset validate --no-rules p0
```

NOTE: Using the `--no-rules` option will not invalidate a config set, unlike `cfgset create/update --no-scripts` command behavior.

The `--include-rule` subcommand specifies a rule name to execute to validate the config set. Multiple `--include-rule` declarations can be made. Rules included via this parameter supersede rules specified in the rules file (`/etc/opt/cray/imps/rules.yaml`). Included rules supersede all excluded rules as well.

```
smw# cfgset validate --include-rule INCLUDE_RULE p0
```

The `--exclude-rule` subcommand specifies a rule name to skip when validating the config set. Multiple `--exclude-rule` declarations can be made. Rules excluded via this parameter supersede rules specified in the rules file (`/etc/opt/cray/imps/rules.yaml`).

To validate the resulting configuration services after a merge of the service packages with the config set content, add the `--merge` option.

```
smw# cfgset validate --merge SERVICE_PACKAGE
```

List Validation Rules with the `list-rules` Command

Use the `cfgset list-rules` command to list the validation rules for a given config set:

```
smw# cfgset list-rules p0
Listing the rules for the config set.
```

```
Rules:

- name: sdb.cray_sdb.CraySDBEnabled
  description: The cray_sdb service must be enabled.
  location: /opt/cray/imps_config/sdb/default/configurator/rules/cray_sdb.py

- name: sdb.cray_sdb.SDBGroupsNodeCheck
  description: The cray_sdb service must only configure tier1 and/or tier2 nodes
as SDB nodes.
  location: /opt/cray/imps_config/sdb/default/configurator/rules/cray_sdb.py
```

The `--service` *SERVICE* subcommand can be used to list the rules that apply to a specified service. The `--service` subcommand should not be used with the `--name` subcommand.

```
smw# cfgset list-rules --service cray_boot p0
Listing rules for the cray_boot service.

Rules:

- name: system-config.cray_boot.BootGroupsNodeCheck
  description: The cray_boot service must only configure tier1 and/or tier2 nodes
as boot nodes.
  location: /opt/cray/imps_config/system-config/default/configurator/rules/
cray_boot.py

- name: system-config.cray_boot.BootNodeGroupsNotEmpty
  description: The cray_boot service must set at least one node as the boot node.
  location: /opt/cray/imps_config/system-config/default/configurator/rules/
cray_boot.py
```

The `--name` *NAME* subcommand can be used to limit the output of the rule listing to a specified service for the given config set. The `--name` subcommand should not be used with the `--service` subcommand.

```
smw# cfgset list-rules --name system-config.cray_storage.CrayStorageEnabled p0

- name: system-config.cray_storage.CrayStorageEnabled
  description: The cray_storage service must be enabled.
  location: /opt/cray/imps_config/system-config/default/configurator/rules/
cray_storage.py
```

Config Set Create/Update Process

Config sets are created and updated using the `cfgset` command with the `create` and `update` subcommands, respectively. Invoking `cfgset` with one of those subcommands initiates the following process, which defines how configuration content is discovered from service packages installed on the management node and used, along with site-supplied content, to create or update a config set.

1. `cfgset` searches for service packages in `/opt/cray/imps_config`.
2. `cfgset` copies to the config set (for `create`) or overwrites in the config set (for `update`) ansible and dist content from each service package. Note that it is only content from service packages that is overwritten; content placed in those directories manually is unchanged.

   NOTE: Manual changes to service package content in this directory will be overwritten!
3. `cfgset` runs pre-configuration callback scripts from each service package. Scripts act on the config set to create content necessary for system configuration, which they place into the `files` subdirectory of the config set.
4. `cfgset` invokes the configurator to do steps 4 through 6.

Configurator finds configuration templates from each service package that match the config set type, and then copies them into the config set (for `create`) or merges them with the templates already in the config set (for `update`).

5. Configurator takes *one* of these actions to further modify config set template data, depending on the command-line options used:

interacts with user	Initiates an interactive session with the user and modifies config set template data based on the values supplied by the user. Occurs when `--mode interactive` option used or no mode option used, which defaults to `auto` mode.
does not interact with user	Does not initiate an interactive session and does no further modification to config set template data beyond the copy/merge of service package data already done in step 4. Occurs when `--mode prepare` option used. Note that although this action is associated with preparing worksheets, all three actions result in worksheets being written in step 6.
imports worksheets	Imports configuration worksheets and modifies config set template data based on the values in each service worksheet. Occurs when `--worksheet-path` *`FILEPATH`* option used.

6. Configurator writes configuration template data, configuration worksheets, and a changelog to the config set. Note that the configurator never modifies the configuration templates in service packages, which are found in `/opt/cray/imps_config/`*`SERVICE PACKAGE`* for each service package.
7. `cfgset` runs post-configuration callback scripts from each service package.
8. `cfgset` autosaves the config set to a time-stamped clone.

The following three figures illustrate how this eight-step process is used to create a CLE config set. They differ in how configuration data in a config set is further modified in step 5, corresponding to the three different actions: interacting with the user (modification through user interaction), not interacting with the user (no further modification), and importing worksheets (modification through bulk import of configuration worksheets). Black lines indicate `cfgset` actions, and red lines indicate actions taken by the configurator when invoked by `cfgset`.

This first figure shows how the configurator creates config set templates (in the `config` subdirectory) from service package templates in step 4, enables the user to enter new or modify existing configuration data in step 5, and then saves the new/modified data to the config set templates and worksheets in step 6.

Figure 19. Process to Create a Config Set Interactively

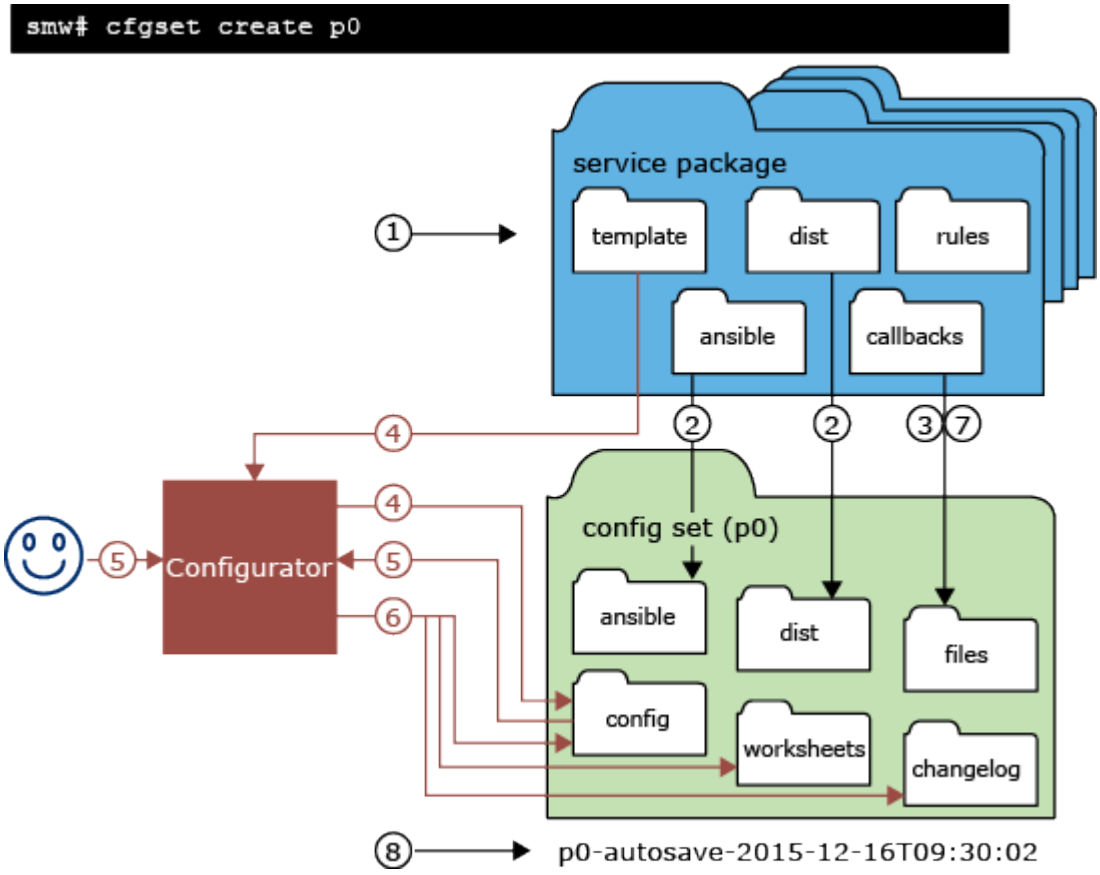
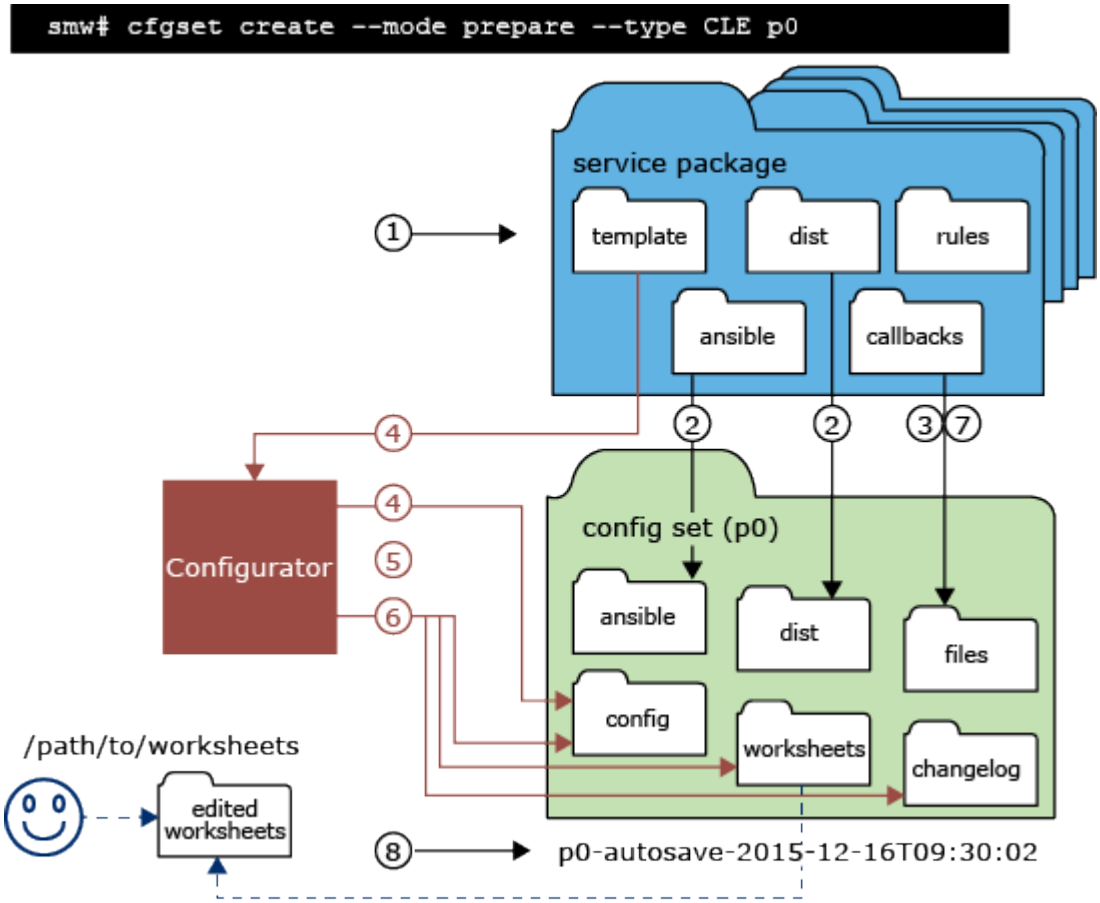
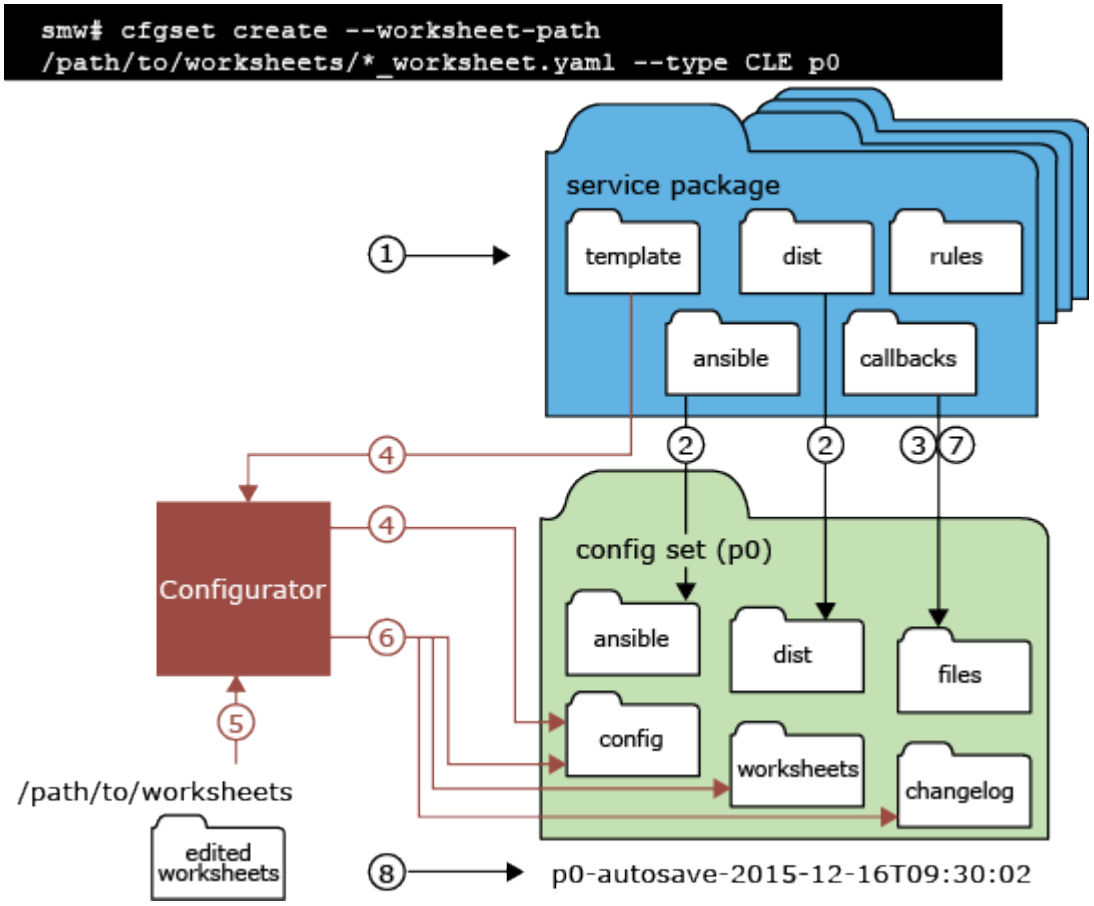


Figure 20. Process to Create a Config Set and Prepare Worksheets



The prepare-mode figure shows how the configurator creates config set templates from service package templates in step 4, does nothing to that configuration data in step 5, and then saves the data from step 4 to config set templates and worksheets in step 6. The blue dashed line indicates an action taken by the user after `cfgset` has completed the create/update process to prepare worksheets. The user (usually an installer or system administrator) copies the worksheets prepared by the configurator to a location outside the config set and edits them (or has other site staff edit them) with site-specific configuration values. It is these edited worksheets that are used when creating (or updating) a config set from worksheets (shown in worksheets figure).

Figure 21. Process to Create a Config Set from Worksheets



The worksheets figure shows how the configurator creates config set templates from service package templates in step 4, imports new or modified configuration data from worksheets in step 5, and then saves the new/modified data to the config set templates and worksheets in step 6.

Tips for Configurator Interactive Sessions

When a user invokes `cfgset` in `auto` or `interactive` mode to create or update a config set, `cfgset` invokes the configurator to initiate an interactive session with the user. The configurator provides command help to aid users in navigating the tool and adding/updating configuration data. These tips supplement that help.

Know the difference between the two "interactive" modes

Interactive mode and auto mode can both result in a configurator interactive session, but their uses and behaviors are quite different.

auto mode	Helpful for verifying that all desired settings have been set. Auto mode initiates an interactive session when there are one or more settings in the config set that meet state and level filtering criteria. Those settings are presented one at a time, and when all have been presented, the configurator exits the session.
interactive mode	Helpful for seeing the "big picture" and having more control over which services/settings are presented for configuration.

Interactive mode always initiates an interactive session. It provides two tiers of menus from which users can select one or more services/settings to drill down and configure just what is needed. The configurator presents the selected settings one at a time, as in auto mode, but when all selected settings have been presented, it returns the user to the menu from which the selection was made.

- *Service Configuration List Menu* (or Service List Menu) lists the services in the config set
- *Service Configuration Menu* (or service menu) lists the settings in a particular service

Filter wisely

Level and state filters determine what the configurator displays to users: what is included in the menu of services/settings for selection in interactive mode, and what setting fields are presented automatically for configuration in auto mode. The filters can be specified on the command line when invoking `cfgset`, and they can be changed in interactive mode. If not specified, they default to level `basic` and state `unset` (exception: for interactive mode, if a single service is specified, the default state is `all`).

In interactive mode, the configurator populates the Service List Menu with only those services that meet state and level filtering criteria; both filters can be switched to different values on this menu screen. In the case of a service menu, the configurator populates it with only those setting fields that meet level filtering criteria (shows all states); level can be switched on this menu screen, but state cannot. Just for fun, cycle through all levels/states, noting how level affects which services appear in the list, while state affects the status displayed for each service.

> **TIP:** If the desired service/setting is not visible in an interactive-mode menu, simply switch level.

In auto mode, the configurator presents only those setting fields that meet state and level filtering criteria. There is no opportunity to switch filter values in auto mode, except by first switching to interactive mode.

> **TIP:** A good way to confirm that all basic settings have been set is to run `cfgset update p0` (where `p0` is the config set name), which defaults to auto mode, level `basic`, and state `unset`. If the configurator does not present any settings, it means that no `basic` or `required` settings are unset.

How to switch states and levels (interactive mode only):

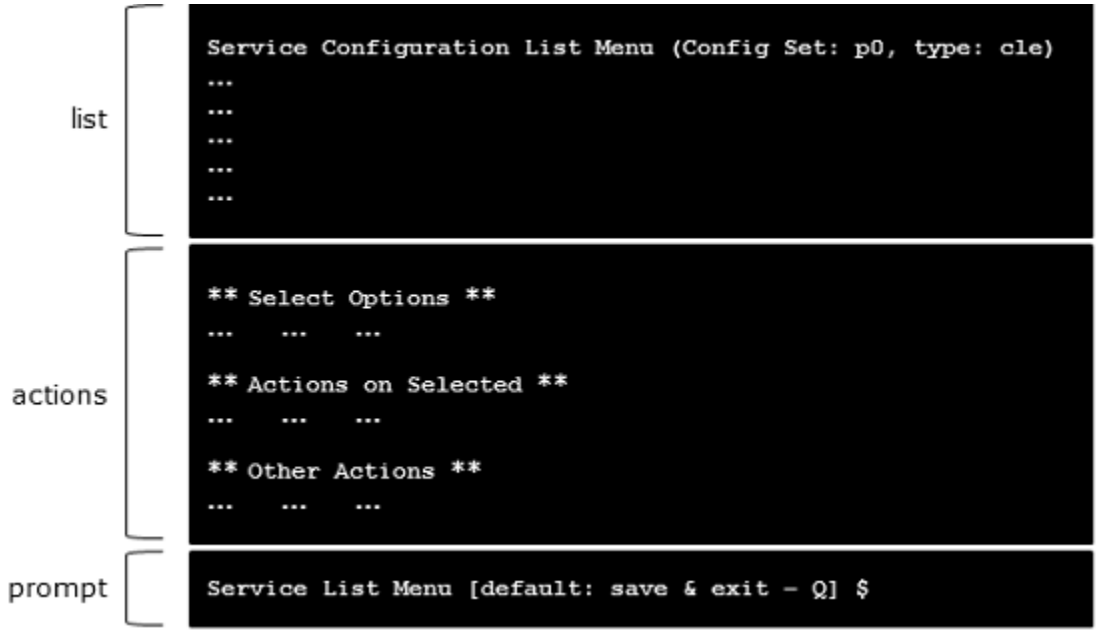
switch states	Enter **`s`** at the configurator prompt to switch from the current state to the next one: `unset` → `set` → `all`. To see all services/settings with the specified level, enter **`s`** until `state=all` displays in the menu header.
switch levels	Enter **`l`** (lowercase L) at the configurator prompt to switch from the current level to the next one: `basic` → `advanced` → `required`. To view all services/settings with the specified state, enter **`l`** until `level=advanced` displays in the menu header.

To see all possible services/settings, switch to `state=all` and `level=advanced`.

Get familiar with menus in interactive mode

The Service List Menu and all service menus have the same three-section layout: a list of services/settings, actions the user can take, and a prompt.

Figure 22. Sections of Interactive-Mode Menus



list The menu name, config set name, and config set type are shown at the top of the list section. This section is helpful for seeing which services still have unconfigured settings (status column—see what changes when state is switched) and for selecting which service(s) to configure or reconfigure.

In a service menu, the list items are configuration settings for that particular service, filtered by level only (state is set to `all` and cannot be switched). This list is helpful for seeing the current state and value of the settings and for selecting which setting(s) to set or change.

actions These three submenus show all commands currently available. Always use an action from the **Select Options** submenu before using any from the **Actions on Selected** submenu. Items in the **Other Actions** submenu can be used at any time (with the obvious exceptions of the exit commands **Q** and **x**, because when one of those is used, the configurator exits the interactive session).

Select Options	Actions that select one or more services/settings from the list. The selected services/settings are the only ones that can be acted upon. Once selected, an asterisk appears in the **Selected** column next to the item and its font color changes.
Actions on Selected	Actions that can be used on the selected service(s) or setting(s); a selection must be made first. Shows in parentheses how many items have been selected. A few of these actions, like toggle whether a service is enabled and toggle whether it inherits setting values from the global version of its template (applies to only a few services) move to the **Other Actions** submenu on service menu screens.
Other Actions	Actions that can be used on all services/settings or on the current configurator session. The most commonly used are the filter switches and help (`?`).

prompt The prompt shows which menu is active and what the default action is. Before a selection is made, the default action is to save and exit (as shown in previous figure). When a selection is made, the default action is to configure the selected service(s) or setting(s), and the prompt changes to

```
MENU_NAME [default: configure - C] $
```

Note that accepting this default action (or entering **c**) displays the configuration setting screen for the first selected setting.

Get familiar with configuration setting screens

A configuration setting screen shows users information about the setting field to be configured (default/current values, data type, level, current state, etc.) and enables the user to navigate among setting fields, enter/change field values, and switch to interactive mode. The configuration setting screen is displayed when a user makes a selection and enters **c** in interactive mode, or when a setting matches state and level filters in auto mode. Configuration setting screens have a prompt that is packed with useful information. Consider this example of a prompt:

```
cray_lmt.settings.lmt_database.data.database_fstype
[<cr>=set 'ext3', <new value>, ?=help, @=less] $
```

The first line is the full name of the setting field being presented (this is the same as the corresponding entry in the configuration worksheet for this service). The part that precedes `.settings.` is the service name (`cray_lmt`, the Lustre Monitoring Tool service, in the example), and the part that follows is the setting field being presented. In the example, the setting is `lmt_database` and the field to be set (one of several for that setting) is `database_fstype`.

The second line lists available commands. In the example, the default command (selected by pressing **Enter** or <cr>) sets the value to `ext3`, which is the default value provided in the configuration template for that service. If this setting field had already been configured with the value `ext3`, the default command would be `<cr>=keep 'ext3',` (`set` becomes `keep`). This list of available commands is not exhaustive: to see all possible options, enter **?** after the prompt, which will insert a context-sensitive menu of commands between the information section and the prompt.

Switch to interactive mode, as needed

When in a configuration setting screen, whether the user has arrived there by invoking `cfgset` in auto mode or by making a selection and entering **c** in interactive mode, it is possible to switch to interactive mode and display either the service menu (lists settings for a single service) or the Service List Menu (lists services in the config set).

switch from setting screen to a service menu

To switch to interactive mode and display the service menu, enter **^** at the configurator prompt. Example:

```
cray_node_health.enabled
[<cr>=keep 'true', <new value>, ?=help, @=less] $ ^
```

switch from setting screen to Service List Menu

To switch to interactive mode and display the Service List Menu, enter **^^** at the configurator prompt. This action can be taken only if `cfgset` was invoked for all services (as this is the default, this is true unless the `--service` or `-s` option was used). Example:

```
cray_node_health.enabled
[<cr>=keep 'true', <new value>, ?=help, @=less] $ ^^
```

Switch between menus in interactive mode, as needed

switch from Service List Menu to service menu	When a service has been selected from the Service List Menu in interactive mode, enter **v** (view settings) to switch to the selected service's menu instead of taking the default action of Configure (**c**). The **v** action is available if only a single service is selected. If multiple services are selected, **c** is the only action available. Example: `Service List Menu [default: configure - C] $ v`
switch from service menu to Service List Menu	To switch from a service menu to the Service List Menu, enter **^^** at the configurator prompt. This action can be taken only if `cfgset` was invoked for all services (as this is the default, this is true unless the `--service` or `-s` option was used). Example: `Node Health Service Menu [default: save & exit - Q] $ ^^`

When in doubt, jump out

It is better to leave a setting field unconfigured than set it to an incorrect value or 'none.' If unsure what the value should be or whether that setting field is needed, jump out using one of these methods:

- *Switch to interactive mode, as needed*.
- Skip to the next setting field: enter **>** at the configurator prompt.

Get help early and often

Enter **?** at the configurator prompt at any time to see a list of available commands. In interactive mode, this simply displays a verbose list of the same commands listed in the menu's three action submenus. However, in a configuration setting screen, entering **?** displays a context-sensitive menu of available commands not displayed elsewhere. Here is an example of the commands available in the context of configuring a multival setting in a service (multival settings are configured by adding/changing entries). Use the **?** command in configuration setting screens early and often to learn the available commands.

```
|--- Command Help
|   *    ++     - double view limit (currently 2)
|   *    --     - decrease view limit by half (currently 2)
|   *    *      - view all entries (no limit)
|   *    +      - add entries
|   *    <#>*   - change the <#> entry. Example: '2b*' selects sub-item b in entry 2
to change
|   *    <#>-   - delete the <#> entry. Example: '4-' deletes entry 4
|   *    d      - delete all entries in the list
|   *    <cr>   - accept the current value(s)
|   *    #      - set the value to its default
|   *    <      - go back to the previous setting
|   *    >      - skip and go to the next setting
|   *    ^      - Go to the 'cray_dvs' service menu (interactive mode)
|   *    ^^     - Go to the service list menu (interactive mode)
|   *    Q      - write out changes and exit the configurator
|   *    x      - revert all changes and exit the configurator
|   *    r      - refresh the screen
|   *    @      - toggle more/less info
|   *    ?      - show this help
```

cfgset Troubleshooting Tips

Unable to Update a Service in a Config Set

The following command to update *SERVICE* in config set `p0` can result in a variety of outcomes, depending on the level and state of the settings in that service.

```
smw# cfgset update --service SERVICE p0
```

Note that for a service with configuration template file `cray_example_config.yaml`, use only the `cray_example` portion on the command-line when specifying a single service.

- **Outcome 1: No configuration settings presented.**

  ```
  INFO - Running pre-configuration scripts
  ...
  INFO - Merging configuration templates and validating schema.
  INFO - Configuration worksheets will be saved to /var/opt/cray/imps/config/sets/
  p0/worksheets
  INFO - Changelog will be written to
          - /var/opt/cray/imps/config/sets/p0/changelog/
  changelog_2015-12-02T16:39:25.yaml
  INFO - Running post-configuration scripts
  ...
  INFO - ConfigSet 'p0' has been updated.
  ```

  The command does not specify mode, level, or state, so defaults are used: `auto` mode, level `basic`, and state `unset`. Therefore, the configurator looks only for required and basic settings that are unset. If it finds none, no interaction with the user is necessary, so it proceeds directly to saving worksheets and logs, and then `cfgset` runs post-configuration activities and exits automatically. **If the intention was to confirm that all required and basic settings have been set, then this is the desired outcome.** However, if the intention was to view all settings and perhaps change a few, use this command instead:

  ```
  smw# cfgset update --service SERVICE --level advanced --mode interactive p0
  ```

- **Outcome 2: Some configuration settings presented, but not the ones that need to be changed.**

  The settings that need to be set/changed are not presented because either they are already set or they are level `advanced`. Try this:

  1. Enter **^** at the configurator prompt to switch to `interactive` mode. Now settings of all states are displayed in the service menu and can be selected and set/changed. If the desired settings are still not found in the service menu, continue to the next step.
  2. Enter **l** (lowercase L) at the configurator prompt to switch to the next level (cycles through all three levels) until `level=advanced` displays in the service menu header. Now settings of all levels and states are displayed in the service menu and can be selected and set/changed.

- **Outcome 3: Some new and unfamiliar configuration settings presented.**

  If the service package that contains the service being updated has been reinstalled, the associated service configuration template may have new or revised settings and values. The configurator will find that template in `/opt/cray/imps_config/`*SERVICE_PACKAGE*`/default/configurator/template` and merge its contents with configuration data already in the config set. When the configurator presents those new settings to the user, they may appear unfamiliar. If settings other than the ones presented need to be set/changed, see Outcome 2.

Validation Rule Failure

When `cfgset validate` encounters a rule failure, a non-zero value is returned and the rule failure is printed:

```
smw# cfgset validate p0
...
Validating ConfigSet 'p0'

Lookup/Reference Errors (1):
    Template: /var/opt/cray/imps/config/sets/p0.alison/config/cray_dvs_config.yaml
        Error: The configured value 'dvs_servers' is not located in the reference
field 'cray_node_groups.settings.groups'
        Location: cray_dvs.settings.client_mount.data.test-ro.server_groups
```

Rule failure can be remedied by adjusting config set data to conform with the failed rule. Alternatively, the rule can be temporarily bypassed using either the `--no-rules` or `--exclude-rule` option. See *Validate a Config Set and List Validation Rules* on page 140 for more details on bypassing validation rules.

Remove Shallow Checksum after Pushing a Config Set from One SMW to Another

About this task

Whenever a config set is pushed from one SMW to another SMW, a shallow checksum line is added to the `.imps_ConfigSet_metadata` file in the top level directory for the CLE config set. After the push is complete, that shallow checksum line must be removed from that file to prevent config set validation failure.

Here is an example of a validation error due to the presence of the shallow checksum line.

```
smw# cfgset validate p0
INFO - Checking directory access
INFO - Checking configuration services
INFO - Checking services for valid YAML syntax
INFO - Checking services for schema compliance
INFO - Merging services and validating schema
INFO - Checking services for valid lookup resolution
INFO - Checking services for required fields
INFO - Checking the global configuration services
INFO - Checking services for valid YAML syntax
INFO - Checking services for schema compliance
INFO - Checking services for valid lookup resolution
INFO - Checking global services for required fields

Validating ConfigSet 'p0'

File Errors (1):
    Error: ConfigSet 'p0' shallow cached checksum identity failure.
Total errors: 1

ConfigSet 'p0' is not valid. Please review the configuration errors above.
Error: 1 of 1 config sets failed to validate.
```

This procedure shows how to remove the shallow checksum line after pushing a config set so that the config set will validate.

Procedure

1. Push a config set p0 from 'oldsmw' to 'newsmw.'

```
oldsmw# cfgset push -d newsmw p0
```

2. Edit the config set metadata file on newsmw.

```
newsmw# vi /var/opt/cray/imps/config/sets/p0/.imps_ConfigSet_metadata
```

3. Remove from the file any line with "shallow checksum."

   For example:

```
shallow checksum: 9d247dc0f0a95e0a50d932103cdff56a
```

4. Validate the config set to ensure that the shallow checksum has been correctly removed and that there are no other validation issues.

```
smw# cfgset validate p0
```

Update cray_sysenv Worksheet

Prerequisites

This procedure assumes that a work area has been set up for editing CLE configuration worksheets and that the current directory has been set to that work area.

```
smw# cd /var/adm/cray/release/p0_worksheet_workarea
```

About this task

The Cray System Environment service enables sites to make any sysctl or limit changes needed within the CLE system environment. This procedure enables the cray_sysenv configuration service.

> **ATTENTION:** Changes to sysctl settings take effect as soon as cray-ansible is run. However, changes to limits settings made after a system has booted take effect only at the next boot.

Procedure

1. Edit `cray_sysenv_worksheet.yaml`.

```
smw# vi cray_sysenv_worksheet.yaml
```

2. Uncomment `cray_sysenv.enabled` and set it to `true`.

Prepare and Update the Global Config Set

Prerequisites

This procedure assumes that the SMW and CLE software has been installed so that the global config set is present.

About this task

The global config set must be updated with site-specific information about several services. This procedure describes how to add site configuration data to the configuration worksheets for each service in the global config set, update the config set with the edited configuration worksheets, and then run Ansible plays on the SMW to effect the changes.

Notes on editing a configuration worksheet:

- Uncomment all settings that are marked level=basic and modify values as needed. All settings that remain commented are considered unconfigured.
- Settings that are already uncommented in the original worksheet are preconfigured to ensure proper configuration of the system; Cray recommends not modifying those preconfigured settings.
- Leave commented all settings that are marked level=advanced unless a default value needs to be modified. Leaving them commented (unconfigured) allows the configurator to safely update defaults that may change in later releases.
- To enter a value for a string that currently is set to `''` (empty string), replace the quotes with the new value. For example, `ipv4_network: ''` becomes `ipv4_network: 10.1.0.0`. In cases where the string value might be interpreted as a number, retain the single quotes. For example, a string setting with value '512' needs quotes.
- To enter one or more values for a list that is currently set to `[]` (empty list), remove the brackets and add each entry on a separate line, preceded by a hyphen and a space (`- `). For example, a list with multiple entries would look like this:

  ```
  cray_global_net.settings.networks.data.management.dns_servers:
  - 172.31.84.40
  - 172.30.84.40
  ```

- Do NOT change or remove the null value in lines like this that appear at the beginning of each set of network, host, or host interface definitions. This line sets the key, or identifier, for that definition. In this example, "hsn" is the identifier for the HSN network definition.

  ```
  cray_net.settings.networks.data.name.hsn: null
  ```

For more information about editing configuration worksheets and updating config sets, see *XC™ Series Configurator User Guide* (S-2560).

> **NOTE:** (SMW HA only) For SMW HA systems, the following procedures are done only on the first SMW because the config sets are shared between both SMWs in the HA cluster. In contrast, Ansible plays must be run on each SMW.

Procedure

1. Save a copy of original global worksheets.

   Copy the original configuration worksheets into a new directory to preserve them in case they are needed later for comparison.

   ```
   smw# ls -l /var/opt/cray/imps/config/sets/global/worksheets

   smw# cp -a /var/opt/cray/imps/config/sets/global/worksheets \
   /var/opt/cray/imps/config/sets/global/worksheets.orig
   ```

2. Make a work area for global worksheets.

   a. Copy the global configuration worksheets to a new work area for editing.

      The worksheets should not be edited in their original location for two reasons: (1) the configurator will not permit updating a config set from worksheets within that config set, and (2) edits would be overwritten when the config set is updated.

      ```
      smw# cp -a /var/opt/cray/imps/config/sets/global/worksheets \
      /var/adm/cray/release/global_worksheet_workarea
      ```

   b. Change to the work area directory to simplify the editing commands in the following steps.

      ```
      smw# cd /var/adm/cray/release/global_worksheet_workarea
      ```

   UPDATE WORKSHEETS FOR GLOBAL SERVICES

3. Update cray_firewall.

   a. Edit `cray_firewall_worksheet.yaml`.

      ```
      smw# vi cray_firewall_worksheet.yaml
      ```

   b. Uncomment `cray_firewall.enabled` and set it to `true`.

4. Update cray_global_net.

   a. Edit `cray_global_net_worksheet.yaml`.

      ```
      smw# vi cray_global_net_worksheet.yaml
      ```

   b. Uncomment `cray_global_net.enabled` and ensure that it is set to `true`.

   c. Search in the file for `'networks' DATA`, then uncomment all of the lines below it that begin with `cray_global_net.settings.networks` so that those settings will be applied and marked as configured. They define four networks: "admin," "SMW failover," "HSS," and "management."

      NOTE: Do NOT uncomment the similar lines under this heading, because they are examples only and are not configured for these four networks.

      ```
      # ** EXAMPLE 'networks' VALUE (with current defaults) **
      ```

   d. Enter SMW-specific or site-specific values for these management network fields.

      ```
      cray_global_net.settings.networks.data.management.ipv4_network:
      cray_global_net.settings.networks.data.management.ipv4_netmask:
      cray_global_net.settings.networks.data.management.ipv4_gateway:
      cray_global_net.settings.networks.data.management.dns_servers:
      cray_global_net.settings.networks.data.management.dns_search:
      cray_global_net.settings.networks.data.management.ntp_servers:
      ```

      Add values for the `dns_servers` and `dns_search` fields for the management network only, not to any other network. The DNS information to use for these fields was entered during the SLES12 installation, so those values can be found in `/etc/resolv.conf`.

      NOTE: If this site does not use DNS search but does use DNS domain in `/etc/resolv.conf`, then adding a single entry to the `dns_search` setting is functionally equivalent to setting the DNS domain.

e. Set the management network external firewall to true.

```
cray_global_net.settings.networks.data.management.fw_external: true
```

f. Search in the file for `'hosts' DATA`, then uncomment all of the lines that begin with `cray_global_net.settings.hosts` so that those settings will be applied and marked as configured. They define a host called "primary_smw" and two interfaces for it: one that connects to the customer management network ("customer_ethernet") and one that connects to admin nodes ("admin_interface"), such as the boot and SDB nodes.

g. Enter SMW-specific or site-specific values for these items.

There are many more fields defining the "primary_smw" host and its interfaces than are included in this example. These four fields are shown because they are the most likely to need site customization. Sites may wish to change the values of other fields as well.

See the notes on editing worksheets at the beginning of this procedure for information about changing empty string and empty list values.

```
cray_global_net.settings.hosts.data.primary_smw.aliases:
cray_global_net.settings.hosts.data.primary_smw.hostid:
cray_global_net.settings.hosts.data.primary_smw.hostname:
cray_global_net.settings.hosts.data.primary_smw.interfaces.customer_ethernet.ipv4_address:
```

Note that if the customer Ethernet IP address changes, the output from the `hostid` command will be different. After changing the following Ethernet field

```
cray_global_net.settings.hosts.data.primary_smw.interfaces.customer_ethernet.ipv4_addr
ess
```

ensure that this field (the SMW host ID) is set to the output of the `hostid` command.

```
cray_global_net.settings.hosts.data.primary_smw.hostid
```

h. Set the `unmanaged_interface` field of the `customer_ethernet` and `admin_interface` interface settings to `true`.

This applies to both stand-alone SMWs and SMW HA systems. In the case of an SMW that is or will be configured for an SMW HA system, this prevents Ansible from managing eth0 and eth3 before the SMW HA cluster has been configured.

```
cray_global_net.settings.hosts.data.primary_smw.interfaces.customer_ethernet.unmanaged_interface:
 true
...
cray_global_net.settings.hosts.data.primary_smw.interfaces.admin_interface.unmanaged_interface:
true
```

i. (Optional) Configure a virtual LAN (VLAN) interface, as needed.

This example shows the configuration fields needed to configure a VLAN interface with common name set to **`vlan0`**. With the vlan_id set to **`'42'`** (important to keep the single quotes to ensure that this is interpreted as a string) and the etherdevice set to **`eth0`**, the interface name will be set to **`eth0.42`** (*`vlan_etherdevice`*.*`vlan_id`*) automatically if the name field is left empty (recommended). If this site chooses to leave vlan_id empty (NOT recommended), the name field must be set to a non-empty string.

```
cray_net.settings.hosts.data.primary_smw.interfaces.common_name.vlan0: null
cray_net.settings.hosts.data.primary_smw.interfaces.vlan0.name: ''
cray_net.settings.hosts.data.primary_smw.interfaces.vlan0.vlan_id: '42'
cray_net.settings.hosts.data.primary_smw.interfaces.vlan0.vlan_etherdevice: eth0
cray_net.settings.hosts.data.primary_smw.interfaces.vlan0.ipv4_address: some_IP_address
cray_net.settings.hosts.data.primary_smw.interfaces.vlan0.startmode: auto
```

j. (Optional) Configure a bonded interface, as needed.

This example shows the configuration fields needed to configure a bonded interface with common name set to **bond0** and interface name set also to **bond0**. There is no field for bonding master because it is set automatically when the bonding_slaves list has at least one member.

```
cray_net.settings.hosts.data.some_host.interfaces.common_name.bond0: null
cray_net.settings.hosts.data.some_host.interfaces.bond0.name: bond0
cray_net.settings.hosts.data.some_host.interfaces.bond0.bonding_slaves:
- eth0
- eth2
cray_net.settings.hosts.data.some_host.interfaces.bond0.bonding_module_opts: mode=active-backup
    miimon=100
cray_net.settings.hosts.data.some_host.interfaces.bond0.ipv4_address: some_IP_address
cray_net.settings.hosts.data.some_host.interfaces.bond0.startmode: onboot
cray_net.settings.hosts.data.some_host.interfaces.bond0.bootproto: static
```

5. Update cray_global_sysenv.

   The cray_global_sysenv config service, new in CLE 6.0.UP04, enables sites to make any sysctl, systemd, or limit changes needed on the SMW. It provides the same functionality and works the same way as its counterpart in the CLE config set, cray_sysenv. The only difference between them is that cray_sysenv is used for CLE nodes and uses node groups to specify the scope of any change, while cray_global_sysenv is used for the SMW and uses the 'scope' field (always set to 'smw') instead of node groups.

   > **ATTENTION:** Changes to sysctl settings take effect as soon as cray-ansible is run. However, changes to systemd or limits settings made after a system has booted take effect only at the next boot.

   "DefaultTasksMax" and "UserTasksMax" limits on the CLE system and the SMW have been increased in CLE 6.0.UP04. These limit increases will happen automatically, with no need for action by the system administrator.

   a. Edit `cray_global_sysenv_worksheet.yaml`.

      ```
      smw# vi cray_global_sysenv_worksheet.yaml
      ```

   b. Uncomment `cray_global_sysenv.enabled`, if it is commented out, and ensure that it is set to `true`.

6. Update cray_ipforward.

   a. Edit `cray_ipforward_worksheet.yaml`.

      ```
      smw# vi cray_ipforward_worksheet.yaml
      ```

   b. Uncomment `cray_ipforward.enabled`, if it is commented out, and ensure that it is set to `true`.

7. Update cray_liveupdates.

   a. Edit `cray_liveupdates_worksheet.yaml`.

      ```
      smw# vi cray_liveupdates_worksheet.yaml
      ```

   b. Uncomment `cray_liveupdates.enabled` and ensure that it is set to `true`.

8. Update cray_logging.

   a. Edit `cray_logging_worksheet.yaml`.

      ```
      smw# vi cray_logging_worksheet.yaml
      ```

   b. Uncomment `cray_logging.enabled` and ensure that it is set to `true`.

c. Uncomment `cray_logging.settings.global_options.data.raid`. If the boot RAID has a non-standard IP address, change the value of this setting.

d. Uncomment `cray_logging.settings.site_loghost.data.name`. If this site has a site_loghost, change the value of this setting.

9. Update cray_multipath.

Multipath does NOT need to be fully cabled to be used. The multipath driver can handle using one path or many.

a. Edit `cray_multipath_worksheet.yaml`.

```
smw# vi cray_multipath_worksheet.yaml
```

b. Choose one of the following options, depending on whether this site intends to use multipath.

> **NOTE:** (SMW HA only) Cray recommends configuring multipath before configuring and enabling HA. If HA is configured and enabled first, then additional precautions must be taken when enabling multipath, as documented in *XC™ Series SMW HA Installation Guide*.

Will multipath be used?

If no, then uncomment `cray_multipath.enabled` and ensure that it is set to `false`. There is nothing else to configure in this step; proceed to step *10* on page 159.

If yes, then uncomment `cray_multipath.enabled` and set it to `true`. Continue with the following substeps.

c. Enter the list of multipath nodes.

Uncomment `cray_multipath.settings.multipath.data.node_list`, remove the `[]` (denotes empty list), and add a list of nodes (by cname or host ID) in this system that have multipath devices and need to have multipath configured. For sites with boot node failover and/or SDB node failover, Cray recommends adding both the active and passive (failover) nodes to this list.

This example shows a list of three nodes: an SMW with host ID 1eac4e0c, a boot node with cname c0-0c0s4n1, and an SDB node with cname c0-0c0s3n1.

```
cray_multipath.settings.multipath.data.node_list:
- 1eac4e0c
- c0-0c0s4n1
- c0-0c0s3n1
```

d. Configure enabled devices.

Cray has provided a number of enabled devices with pre-populated data under `# ** 'enabled_devices' DATA **`. These storage devices are the devices that will be whitelisted, which means they will be listed as exceptions to the blacklist. The settings for these devices have default values provided by the device vendors and do not need to be changed. If this site intends to configure a multipath device that does not appear in this group of enabled devices, contact a Cray representative for help.

e. (Optional) Configure aliases for the multipath devices.

This is the equivalent of adding aliases to the multipaths section of the `multipath.conf` file. If no aliases are specified, this setting will show as unconfigured when the config set is updated, but this is not a problem. It can remain unconfigured and will not cause the config set to be invalid.

In the worksheet, copy the two lines below `# ** EXAMPLE 'aliases' VALUE (with current defaults) **` and paste them below `# NOTE: Place additional 'aliases' setting entries here, if desired.`

```
# ** EXAMPLE 'aliases' VALUE  (with current defaults) **
#   cray_multipath.settings.aliases.data.wwid.sample_key_a: null   <-- setting a multival key
#   cray_multipath.settings.aliases.data.sample_key_a.alias: ''
#
```

Uncomment the lines, replace `sample_key_a` with the World Wide Identifier (WWID) of the device to be aliased (60080e50002e203c00002a085551b2c8 in this example) in all lines, and remove the `<-- setting a multival key` text at the end of the first line (note that the null value is required; do not remove or change it). Finally, add the alias for this device (smw_node_pv1 in this example). Repeat this substep for each device, as needed.

```
# NOTE: Place additional 'aliases' setting entries here, if desired.
cray_multipath.settings.aliases.data.wwid.60080e50002e203c00002a085551b2c8: null
cray_multipath.settings.aliases.data.60080e50002e203c00002a085551b2c8.alias: smw_node_pv1
#***************** END Service Setting: aliases ***************
```

10. Skip cray_network_boot_packages_worksheet.yaml.

The cray_network_boot_packages configuration service is enabled by default and has no variables that need to be changed.

11. Update cray_time.

a. Edit `cray_time_worksheet.yaml`.

```
smw# vi cray_time_worksheet.yaml
```

b. Uncomment `cray_time.enabled`, if it is commented out, and ensure that it is set to `true`.

c. Uncomment `cray_time.settings.service.data.timezone` and change its value, as needed.

There are many possible values for time zone, such as I.E., US/Central, US/Eastern, and EMEA/BST.

UPLOAD WORKSHEETS AND UPDATE/VALIDATE GLOBAL CONFIG SET

CAUTION: Boot failure possible if using `cfgset` under certain conditions.

The `cfgset create` and `cfgset update` commands always call pre- and post-configuration scripts. Some of these scripts require HSS daemons and other CLE services to be running. This can cause problems under these conditions:

- If `xtdiscover` is running, `cfgset` may hang or produce incorrect data that can result in system boot failure.
- If `xtbounce` is in progress or if the SMW is not connected to XC hardware, `cfgset` will fail.

In these circumstances, use the `--no-scripts` option with `cfgset create` or `cfgset update` to avoid running the scripts. Because using that option results in an invalid config set, remember to run `cfgset update` without the `--no-scripts` option afterwards, when circumstances permit, to ensure that all pre- and post-configuration scripts are run.

12. Upload modified worksheets into global config set.

Note that the full filepath must be specified in this `cfgset` command, and it must be enclosed in single quotes (to prevent the shell trying to expand the file glob).

```
smw# cfgset update -w \
'/var/adm/cray/release/global_worksheet_workarea/*_worksheet.yaml' global
```

13. Update the global config set.

Using the configurator in interactive mode to update the global config set is a good way to check whether all required settings and basic settings have been configured for services that are enabled. If they have, then all enabled services will show OK status in the Service Configuration List Menu. If configuration of a basic setting was missed, then the menu will show how many unconfigured settings there are for each service. Set or change any settings from this menu, as needed.

Note that some basic settings can be left unconfigured, such as aliases for multipath devices, because configuring them is optional.

```
smw# cfgset update -m interactive global
```

When the configurator session completes, it displays a message indicating the file name of the changelog file for this configuration session. The changelog is written to a file in the `/var/opt/cray/imps/config/sets/global/changelog` directory.

14. Validate the global config set.

```
smw# cfgset validate global
```

APPLY CONFIGURATION CHANGES ON THE SMW

15. Run Ansible plays on the SMW.

After the global config set has been updated, reapply any Ansible plays that consume global config set data.

> **NOTE:** (SMW HA only) Both SMWs require this command. The procedure to install and configure the second SMW includes this command.

```
smw# /etc/init.d/cray-ansible start
```

Logs from running Ansible plays, such as cray-ansible, are stored on the SMW in `/var/opt/cray/log/ansible`.

CHECK TIME SETTINGS

16. Check for external NTP servers.

Check that external NTP servers have been set as desired in the global config set.

> **NOTE:** (SMW HA only) Both SMWs require this command. The procedure to install and configure the second SMW includes this command.

```
smw# grep server /etc/ntp.conf
server ntpserver1 minpoll 4 iburst
server ntpserver2 minpoll 4 iburst
```

17. Put the SMW time zone setting where the cabinet and blade controllers can access it.

This SMW time zone setting will be applied to the cabinet and blade controllers when they are rebooted later in the process.

> **NOTE:** (SMW HA only) Both SMWs require this command. The procedure to install and configure the second SMW includes this command.

```
smw# cp -p /etc/localtime /opt/tftpboot/localtime
```

About Simple Sync

The Cray Simple Sync service (cray_simple_sync) provides a simple, easy-to-use, generic mechanism for administrators to make configuration changes to their system without resorting to writing a custom Ansible play. When enabled, the service automatically copies files found in source directories in the config set on the SMW to one or more target nodes. Simple Sync is a simple tool and not intended as the sole solution for making configuration changes to the system. Writing custom Ansible plays might provide better maintainability, flexibility and scalability in the long term.

The Simple Sync service is enabled by default and has no additional configuration options. It can be enabled or disabled during the initial installation using worksheets or with the `cfgset` command at any time.

```
smw# cfgset update --service cray_simple_sync --mode interactive <config_set_name>
```

For more information, see man cfgset(8).

How Simple Sync Works

When enabled, Simple Sync is executed on all CLE nodes at boot time and whenever the site administrator executes `/etc/init.d/cray-ansible start` on a CLE node. When Simple Sync is executed, files placed in the following directory structure are copied onto nodes that match these criteria:

`smw:/var/opt/cray/imps/config/sets/<config_set>/files/simple_sync/`

./common/files/	Matches all nodes.
./hardwareid/*<hardwareid>*/files/	Matches a specific node with that hardware ID, which is the cname of a CLE node or the output of the `hostid` command (e.g., 1eac0b0c) on other nodes. An admin must create both the `<hardwareid>` directory and the `files` directory.
./hostname/*<hostname>*/files/	Matches a node with the specified host name. An admin must create both the `<hostname>` directory and the `files` directory. Use for eLogin nodes ONLY.
./nodegroups/*<node_group_name>*/files/	Matches all nodes in the specified node group. The directories for this `nodegroups` directory are automatically stubbed out when the config set is updated after node groups are defined and configured in the cray_node_groups service.
./platform/[compute, service]/files/	Matches all compute nodes or all service nodes, depending on whether they are placed in `platform/compute/files` or `platform/service/files`. Each time the config set is updated, the HSS data store is queried to update which nodes are service and which are compute.
./README	Provides brief guidance on using Simple Sync and a list of existing node groups in the order in which files will be copied. This ordering enables an administrator to predict behavior in cases where a file may be duplicated within the Simple Sync directory structure.

Simple Sync copies content into place prior to the standard Linux startup (`systemd`) and before `cray-ansible` runs any other services. As a result, Cray services that make small changes to files will operate on the administrator-provided file. Afterwards, the file will contain both non-conflicting administrator-provided content as well as the changes made by the Cray service. Because these changes happen prior to Linux startup, the changes will be in place when the services start up.

Note that there are some config files that are entirely managed by Cray services. Where possible, such files have a comment at the top indicating that the file is completely under the management of the Cray service. Files that have been changed by Cray services can be identified by checking the change logs on the running node in `/var/opt/cray/log/ansible`. Simple Sync does not provide a mechanism to override changes made by Cray services. To override changes made by Cray services, refer to the documentation for the specific service.

The ownership and permissions of copied directories and files are preserved when they are copied to root (`/`) on the matching target nodes. An administrator can run cray_ansible multiple times, as needed, and only the files that have changed will be copied to the target nodes.

Because of the way it works, Simple Sync can be used to configure services that have configuration parameters not currently supported by configuration templates and worksheets. An administrator can create a configuration file with the necessary settings and values, place it in the Simple Sync directory structure, and it will be distributed and applied to the specified node(s).

Characteristics of Simple Sync

Simple Sync is:	**Simple Sync is NOT:**
for simple and straightforward use cases	a comprehensive system management solution
for copying a moderate number of moderately sized files*	intended to transfer large objects or a large volume of files
	an interface to configure Cray "turnkey" services such as ALPS, Node Health or Lightweight Log Manager (LLM)

* Bear in mind that anything in the Simple Sync directory structure is part of a config set, and a SquashFS copy of the current config set is distributed to all nodes in the system. Even though it is a reduced-size config set that is distributed, it is good practice to not add very large files to a config set, hence the use of "moderate" here.

Introduced with the CLE 6.0.UP00 / SMW 8.0.UP00 release, Simple Sync has been enhanced to:

- run as early in the Ansible execution sequence as possible (it runs BEFORE other cray-ansible plays, so it can be used to make changes to files that Cray updates, like `sshd_config`)
- run during the netroot setup sequence, so it can be used to change LNet and DVS settings, if needed
- support Node Groups for targeting which system nodes to copy files to (see *About Node Groups* on page 166)

Simple Sync does not support:

- removing files
- appending to files
- changing file ownership and permissions (the permissions of the file in the config set are mirrored on-node)
- backing up files

- overriding Cray-set values (it cannot be used to change files that Cray completely overwrites, such as `alps.conf`, or change values in files that Cray modifies such as PermitRootLogin in `/etc/ssh/sshd_config`)

Cautions about the Use of Simple Sync

- Simple Sync copies files from the config set, which in the case of nodes without a persistent root file-system is cached in a compressed form, locally, in memory. As a result, each file stored in the config set uses some memory on the node. Therefore, using Simple Sync to copy binary files or large numbers of files is inadvisable.
- Be aware of differences in node environments when using Simple Sync. For example, systems configured with direct-attached Lustre (DAL) have nodes running CentOS instead of SLES. Administrators would have to be very careful to avoid putting an inappropriate configuration file into place when using the Simple Sync platform/service target in such a situation.
- Storage and distribution of verbatim config files through Simple Sync creates the potential for unintentional impact to the system when config files evolve due to software changes. Making minimal necessary changes through a site-local Ansible playbook provides more flexibility and minimizes the potential for unintended consequences.

Use Cases

Copy a non-conflicting file to all nodes

1. Place `etc/myfile` under `./common/files/` in the Simple Sync directory structure.
2. Simple Sync copies it to `/etc/myfile` on all nodes.

Copy a non-conflicting file to a service node

1. Place `etc/servicefile` under `./platform/service/files/` in the Simple Sync directory structure.
2. Simple Sync copies it to `/etc/servicefile` on all service nodes.

Copy a non-conflicting file to a compute node

1. Place `etc/computefile` under `./platform/compute/files/` in the Simple Sync directory structure.
2. Simple Sync copies it to `/etc/computefile` on all compute nodes.

Copy a non-conflicting file to a specific node

1. Place `etc/mynode` under `./hardwareid/c0-0c0s0n0/files/` in the Simple Sync directory structure.
2. Simple Sync copies it to `/etc/mynode` on c0-0c0s0n0.

Copy a non-conflicting file to a user-defined collection of nodes

1. Create a node group called "my_nodes" containing a list of nodes.
2. Update the config set.

```
smw# cfgset update p0
```

3. Place `etc/mynodes` under `./nodegroups/my_nodes/files/` in the Simple Sync directory structure.
4. Simple Sync copies it to `/etc/mynodes` on all nodes listed in node group my_nodes.

Copy to a node a file that has Cray-maintained content

To reduce the number of authentication tries from the default of six,

1. Place a version of `sshd_config` with the value “MaxAuthTries 3” under `./nodegroups/login_nodes/files/etc/ssh/` in the Simple Sync directory structure.
2. The booted system will contain both:
   - “MaxAuthTries 3” (from the file copied by Simple Sync)
   - “PasswordAuthentication yes” (from modification of file by Cray)

Copy to a node a file that is exclusively maintained by Cray

Files exclusively maintained by Cray such as `alps.conf` cannot be updated using Simple Sync. Please refer to the owning service (such as ALPS) for information on how to update the contents.

Copy to a node a file that resides on a file system that will be mounted during Linux boot

No special operational changes are necessary. However, Simple Sync will put the file in place early in the boot sequence, and then it will be over-mounted by the file system. Because Simple Sync runs again later, it will copy the file into the mounted file system. Due to the ordering of operations, the file will not be present between the time the file system was mounted until the late execution of Ansible.

On netroot login nodes, modify an LNet modprobe parameter

1. Generate a file `zz_lnet.conf` containing `options lnet router_ping_timeout=100`.
2. Place `zz_lnet.conf` under `./nodegroups/login/files/etc/modprobe.d/` in the Simple Sync directory structure.
3. The `lnet router_ping_timeout` value will be 100.

Note that normally Simple Sync does not allow the user to override Cray values, but this procedure takes advantage of the standard Linux mechanism to override Kernel module options.

Copy a file with an incompatible content to a node file that has Cray-maintained content

While Simple Sync allows an administrator to make changes to the same configuration files as modified by Cray, be very careful to avoid introducing syntax errors or incompatible values that may cause the system to fail to operate correctly.

Configure Files for Cray Simple Sync Service

About this task

Cray Simple Sync provides a generic mechanism to automatically distribute files to targeted locations on the system. This mechanism can be used to override or change default system behavior through the contents of the distributed files. When enabled, the Simple Sync service is executed on all CLE nodes at boot time and whenever the administrator executes `/etc/init.d/cray-ansible start` on a CLE node. When Simple Sync is executed, files placed in the following directory structure are copied to the root file system (`/`) on the target nodes.

About the Simple Sync Directory Structure

The Simple Sync directory structure has this root:
`smw:/var/opt/cray/imps/config/sets/<config_set>/files/simple_sync/`

Below that root are the directories listed on the left:

Files placed here	are copied to
`./common/files/`	all nodes
`./platform/[compute, service]/files/`	all compute or service nodes
`./hardwareid/<hardwareid>/files/`	nodes with matching hardware ID, which is the cname of a CLE node or the output of the `hostid` command (e.g., 1eac0b0c) on other nodes
`./hostname/<hostname>/files/`	nodes with matching host name (use this for eLogin nodes ONLY)
`./nodegroups/<node_group_name>/files/`	nodes in the matching node group

> **NOTE:** The directory structure for a particular hardware ID or host name (everything below `./hardwareid/` and `./hostname/`) must be created manually as needed. This is unnecessary for node groups because their associated directories are created automatically by post-configuration callback scripts when the config set is created or updated using `cfgset`.

Anything (directory structure and files) placed below `./files/` in the Simple Sync directory structure on the SMW is replicated on the target node starting at root (`/`). For example, this path on the SMW

```
/var/opt/cray/imps/config/sets/p0/files/simple_sync/common/files/etc/myapplication.conf
```

will place the `myapplication.conf` file on all nodes in this directory:

```
/etc/myapplication.conf
```

Note that the ownership and permissions of files in the config set are preserved in the copies made to nodes.

For more information and use cases, see *About Simple Sync* on page 161.

About the Node Image Mapping Service (NIMS)

The Node Image Mapping Service (NIMS) maps a node to *boot attributes*, which are used when the node is booted.

The primary NIMS component is the daemon, `nimsd`. Interact with `nimsd` either by sending a Hardware Supervisory System (HSS) event or by using the NIMS command line interface (CLI). The HSS Boot Manager daemon communicates with `nimsd` via HSS events. All other interactions with `nimsd` take place through the CLI.

The `nimsd` daemon provides these boot attributes to Boot Manager upon request. Boot Manager uses the boot attributes when it boots or reboots nodes. Boot Manager also provides the boot attributes to the `xtcli` command.

Two conceptual components, nodes and maps, are affected by `nimsd`. A node represents a physical, bootable node on the mainframe. A map is a collection of nodes, typically all the nodes in a partition, or for a non-partitioned system, all the nodes in the entire mainframe.

There can be multiple NIMS maps. However, only one map can be active at a time. The reason to have multiple maps is to differentiate the boot attributes. For example, one map may be a test map to allow booting nodes with a test boot image or a test Config Set.

About Node Groups

The Cray Node Groups service (cray_node_groups) enables administrators to define and manage logical groupings of system nodes. Nodes can be grouped arbitrarily, though typically they are grouped by software functionality or hardware characteristics, such as login, compute, service, DVS servers, and RSIP servers.

Node groups that have been defined in a config set can be referenced by name within all CLE services in that config set, thereby eliminating the need to specify groups of nodes (often the same ones) for each service individually and greatly streamlining service configuration. Node groups are used in many Cray-provided Ansible configuration playbooks and roles and can be also used in site-local Ansible plays. Node groups are similar to but more powerful than the class specialization feature of releases prior to CLE 6.0. For example, a node can be a member of more than one node group but could belong to only one class.

Sites are encouraged to define their own node groups and specify their members. Administrators can define and manage node groups using any of these methods:

- Edit and upload the node groups configuration worksheet (`cray_node_groups_worksheet.yaml`).
- Use the `cfgset` command to view and modify node groups interactively with the configurator.
- Edit the node groups configuration template (`cray_node_groups_config.yaml`) directly. Use `cfgset` to update the config set afterwards so that pre- and post-configuration scripts are run.

After using any of these methods, remember to validate the config set.

Characteristics of Node Groups

- Node group membership is not exclusive, that is, a node may be a member of more than one node group.
- Node group membership is specified as a list of cnames. However, if the SMW is part of a node group, it is specified with the output of the `hostid` command. Also, host names are used for eLogin nodes that are to be included in node groups.
- All compute nodes and/or all service nodes can be added as node group members by including the keywords “platform:compute” and/or “platform:service” in a node group.
- Any CLE configuration service is able to reference any defined node group by name.

- The Configuration Management Framework (CMF) exposes node group membership of the current node through the local system "facts" provided by the Ansible runtime environment. This means that each node knows what node groups it belongs to, and that knowledge can be used in Cray and site-local Ansible playbooks.

Default Node Groups

Default node groups are groups of nodes that

- are likely to be customized and used by many sites
- support useful default values for many of the migrated services

Several of the default node groups require customization by a site to provide the appropriate node membership information. This table lists the Cray default groups and indicates which ones require site customization.

Table 7. cray_node_groups

Default Node Group	Requires Customization?	Notes
compute_nodes	No	Defines all compute nodes for the given partition. The list of nodes is determined at runtime.
service_nodes	No	Defines all service nodes for the given partition. The list of nodes is determined at runtime.
smw_nodes	Yes	Add the output of the `hostid` command for the SMW. For an SMW HA system, add the host ID of the second SMW also.
boot_nodes	Yes	Add the cname of the boot node. If there is a failover boot node, add its cname also.
sdb_nodes	Yes	Add the cname of the SDB node. If there is a failover SDB node, add its cname also.
login_nodes	Yes	Add the cnames of internal login nodes on the system.
elogin_nodes	Yes	Add the host names of external login nodes on the system. Leave empty (set to `[]`) if there are no eLogin nodes.
all_nodes	Maybe	Defines all compute nodes and service nodes on the system. Add external nodes (e.g., eLogin nodes), if needed.
tier2_nodes	Yes	Add the cnames of nodes that will be used as tier2 servers in the cray_scalable_services configuration.

Why is there no "tier1_nodes" default node group? Cray provides a default tier2_nodes node group to support defaults in the cray_simple_shares service. Cray does not provide a tier1_nodes node group because no default data in any service requires it. Because it is likely that tier1 nodes will consist of only the boot node and the SDB node, for which node groups already exist, Cray recommends using those groups to populate the cray_scalable_services tier1_groups setting rather than defining a tier1_nodes group.

About eLogin nodes. To add eLogin nodes to a node group, use their host names instead of cnames, because unlike CLE nodes, eLogin nodes do not have cname identifiers. If eLogin nodes are intended to receive

configuration settings associated with the all_nodes group, add them to that group, or change the relevant settings in other configuration services to include both all_nodes and elogin_nodes.

Additional Platform Keywords

Cray uses these two platform keywords to create default node groups that contain all compute or all service nodes.

```
platform:compute
platform:service
```

Sites that need finer-grained groupings can use these additional platform keywords to create custom node groups that contain all compute or service nodes with a particular core type.

```
platform:compute-XXNN
platform:service-XXNN
```

For *XXNN*, substitute a four-character processor/core suffix, such as KL64 or KL68, which designate two Intel® Xeon Phi™ "Knights Landing" (KNL) processors with different core counts. These suffixes are found in the "Core" column of the output from the following command:

```
smw# xtcli status p0
Network topology: class 0
Network type: Aries
Nodeid: Service Core Arch| Comp state [Flags]
----------------------------------------------------
c0-0c0s0n0: service BW18 X86| ready [noflags|]
c0-0c0s0n1: service BW18 X86| ready [noflags|]
c0-0c0s0n2: service BW18 X86| ready [noflags|]
c0-0c0s0n3: service BW18 X86| ready [noflags|]
c0-0c0s1n0: service BW18 X86| ready [noflags|]
c0-0c0s1n1: service BW18 X86| ready [noflags|]
c0-0c0s1n2: service BW18 X86| ready [noflags|]
c0-0c0s1n3: service BW18 X86| ready [noflags|]
c0-0c0s2n0: - HW12 X86| ready [noflags|]
c0-0c0s2n1: - HW12 X86| ready [noflags|]
c0-0c0s2n2: - HW12 X86| ready [noflags|]
c0-0c0s2n3: - HW12 X86| ready [noflags|]
```

The following table lists some of the common suffixes supported by Cray.

Table 8. Cray Supported Intel Processor/Core (XXNN) Designations

Processor (XX)	Core (NN)	Intel Code Name
BW	12, 14, 16, 18, 20, 22, 24, 28, 32, 36, 40, 44	"Broadwell"
HW	04, 06, 08, 10, 12, 14, 16, 18, 20, 24, 28, 32, 36	"Haswell"
IV	02, 04, 06, 08, 10, 12, 16, 20, 24	"Ivy Bridge"
KL	60, 64, 66, 68, 72	"Knights Landing"
SB	04, 06, 08, 12, 16	"Sandy Bridge"

About the Image Management and Provisioning System (IMPS)

The Image Management and Provisioning System (IMPS) allows the system administrator to manage software content in images. IMPS leverages and extends industry-standard tools such as `zypper` and `yum`.

IMPS uses an *image recipe* to install collections of software (RPMs) into an *image root*. The image root is used create a *boot image*. Image recipes tie together collections of software defined in the package collections and the repositories that contain the software. IMPS creates an image root from an image recipe and resolves all RPM dependencies. When building an image root from a image recipe, IMPS builds any subrecipes and then gathers all specified packages and package collections and software repositories in the image recipe before generating the call to the package manager (`rpm`). After the package manager has created the image root, it may be further modified by non-RPM-based content if there are post-build directives in the recipe.

The Node Image Mapping Service (NIMS) is responsible for keeping track of which images get booted on which nodes, what additional kernel parameters to pass to nodes at boot time, and which load file to use within a boot image. The NIMS map is created during installation and changed when other images are created or when nodes are added, removed, or change function. The administrator can use NIMS to assign a boot image to any node or group of nodes. For more information, see *About the Node Image Mapping Service (NIMS)* on page 166.

IMPS objects include:

Image recipe

Defines the image name and image contents (software). The recipe can include one or more packages (RPMs), package collections (logical groupings of RPMs), repositories, and other recipes (called *sub-recipes* or *nested recipes*). A recipe can also specify post-boot actions such as copying files or executing commands using the `postbuild_copy` and `postbuild_chroot` directives.

Custom image recipes can reference remote repositories that are hosted on an external repository server. For more information, see *Install Third-Party Software with a Custom Image Recipe* on page 173.

Location on the SMW: `/etc/opt/cray/imps/image_recipes.d/`

Image root

Directory on the SMW that contains the installed software. IMPS creates an image root from an image recipe. System administrators can chroot into the image root directory to examine its contents and the packages (RPMs) that were included to resolve build dependencies.

Location on the SMW: `/var/opt/cray/imps/image_roots/`

Boot image

IMPS creates a boot image (a CPIO file) from an image root by packaging the image root into a format suitable for booting on a Cray node or eLogin node. Note that a boot image is essentially unconfigured; the node configuration comes from the config set.

A boot image is the root file system for a node or group of nodes.The Cray XC™ Series root file system for nodes can either reside in RAM (tmpfs) or be mounted from a network source (netroot), depending on the type of node. The boot and SDB nodes, all other service nodes (except login nodes), and all DAL (direct-attached Lustre) nodes must use tmpfs. Compute nodes and login nodes may use either tmpfs or netroot. For more information, see *Where to Place the Root File System—tmpfs versus netroot* on page 172.

Location on the SMW: `/var/opt/cray/imps/boot_images/`

Package collection

Logical grouping of RPM packages. A package collection can contain versioned and unversioned package names, and can include other package collections. (Note that the package collections installed for CLE are read-only.) Only the top-level packages should be included in a package collection. The IMPS image creation process takes care of determining package dependencies and installing them from the defined repositories.

Cray recommends using a package collection because the RPMs can be used in multiple image types (such as compute and service node images).

Location on the SMW: `/etc/opt/cray/imps/package_collections.d/`

Repository Logical grouping of RPMs based on repository type and architecture. (The content of SLES repositories and CentOS repositories should never be mixed.) The installation process creates and populates the required repositories. System administrators can create their own repositories for third-party software.

Most operating system and Cray repositories come in pairs (base and updates), such as `sles_12_x86-64` and `sles_12_x86-64_updates`. The updates repository is for future patches and security updates.

Location on the SMW: `/var/opt/cray/repos/`

IMPS Commands for Working with Images

These IMPS commands are available for working with recipes, repositories, package collections, and images:

`recipe`	Creates and manages image recipes.
`repo`	Creates and manages repositories.
`pkgcoll`	Creates and manages package collections.
`image`	Creates and manages image roots and boot images.
`imgbuilder`	Calls several IMPS commands so that multiple images can be built as a set with a single command. This command can also call the NIMS command `cnode` to assign boot images to nodes and adjust the netroot kernel parameter for nodes. The configuration file for `imgbuilder` is `cray_image_groups.yaml`. The `imgbuilder` command uses *image group* configuration information to build boot images. Image groups are defined in the global config set in the `cray_image_groups` configuration file (`/var/opt/cray/imps/config/sets/global/config/cray_image_groups.yaml`).

For each command, use the `list` subcommand to display the existing items. Use the `-h` option to display the available subcommands and arguments. For more information, see the man pages on the SMW.

Cray-provided Image Recipes

Cray provides read-only image recipes that system administrators can build into bootable images for these node types: service, compute, admin (for boot and SDB nodes), login, Direct Attached Lustre (DAL), and eLogin. In addition, administrators can create custom recipes that are based on the read-only recipes.

IMPORTANT: Do not directly modify a Cray-provided image recipe.

The Cray-provided image recipes have names like `compute_cle_6.0.up04_sles_12sp2_x86-64_ari`. The name includes the node function (such as `compute`), Cray product and version (`cle_6.0.UP04`), OS type and version (`sles_12sp2`), architecture (`x86-64`), and network type (`ari` for the XC system Aries network).

Custom Image Recipes

System administrators can use IMPS commands to create custom image recipes that are based on the Cray-provided recipes.There are two ways to customize an image recipe:

- Create a new recipe (with `recipe create`) and add the existing recipe as a sub-recipe. Cray recommends this approach for most images because the recipe will receive updates from patches.
- Clone the Cray-provided recipe (with `recipe clone`) and change the contents. However, the recipe will not receive modifications from patches. For that reason, Cray does not recommend cloning an image recipe except for testing purposes.

Local image recipes are stored in the image recipe directory (`/etc/opt/cray/imps/image_recipes.d/`) in the file `image_recipes.local.json`.

For the procedure to create a custom image, see *Install Third-Party Software with a Custom Image Recipe* on page 173 and *Use a Custom Image Recipe to Change a File on a Compute Node* on page 184.

Format of an Image Recipe

An image recipe is in a JSON file. Note that a single JSON file can contain multiple image recipes. Each image recipe starts with a name and description (a comment describing the intended use for the image). The remaining elements in a recipe specify the package collections, packages (RPMs), repositories, and other recipes (sub-recipes). Each item has a *rationale* (a comment explaining why the item is included in the image). A recipe can also include post-build actions to copy files and execute commands or scripts in a chroot environment.

An image recipe has the following basic format:

```
{
"image_recipe_name": {
        "description": "Example recipe",
        "package_collections": { ... },
        "packages": { ... },
        "postbuild_chroot": [ ... ],
        "postbuild_copy": [ ... ],
        ″recipes": { ... },
        "repositories": { ... }
    },
}
```

The following example shows the format of a custom image recipe for service nodes that includes a workload manager (WLM). It includes the Cray-provided recipe as a sub-recipe, then specifies post-build actions to copy WLM files and run the necessary scripts.

```
"wlm_service": {
      "description": "WLM service node image",
      "package_collections": {},
      "packages": {},
      "postbuild_chroot": [
          "rpm -ivh ${IMPS_POSTBUILD_FILES}/wlm.rpm",
          "${IMPS_POSTBUILD_FILES}/wlm.installer ${IMPS_POSTBUILD_FILES}/
wlm.config"
          ],
      "postbuild_copy": [
          "/home/crayadm/wlm_install/wlm.rpm",
          "/home/crayadm/wlm_install/wlm.installer",
          "/home/crayadm/wlm_install/wlm.config"
          ],
      "recipes": {
          "service_cle_6.0up03_sles_12_x86-64_ari": {
              "rationale": "Start from standard service node recipe"
          },
```

```
        "repositories": {}
    },
```

When using post-build actions, use `postbuild_copy` to copy files and directories from a location on the SMW. Use `postbuild_chroot` to execute post-build commands or scripts, which run in the `chroot` environment of the image root (on the SMW). Use the following environment variables for post-build scripts:

- `IMPS_IMAGE_NAME`
- `IMPS_VERSION`
- `IMPS_IMAGE_RECIPE_NAME`
- `IMPS_POSTBUILD_FILES`

Where to Place the Root File System—tmpfs versus netroot

The Cray XC™ Series root file system for nodes can either reside in RAM (tmpfs) or be mounted from a network source (netroot), depending on the type of node. The boot and SDB nodes, all other service nodes (except login nodes), and all DAL (direct-attached Lustre) nodes must use tmpfs. Compute nodes and login nodes may use either tmpfs or netroot. Use the information provided here to decide whether to use netroot for some or all compute and login nodes at this site.

About netroot and Dynamic Shared Objects and Libraries (DSL)

In releases prior to CLE 6.0 / SMW 8.0, the dynamic shared objects and libraries (DSL) feature was optional. It was necessary for many sites because it enabled both dynamic shared libraries and large network-based images, which were needed for systems with NVIDIA GPUs and for most production workloads.

In the current release, DSL is supported by default. Note, however, that the DSL feature no longer includes provision for large network-based images. That capability is now provided by netroot.

- Sites that require large network-based images and additional storage should use netroot.
- Sites using NVIDIA GPUs must use netroot.

Comparison of tmpfs and netroot

tmpfs By default, the root file system on Cray XC™ Series systems resides in the memory resident file system, tmpfs.

tmpfs has these characteristics and limitations:

- always used for service nodes (except login nodes) and DAL (direct-attached Lustre) nodes
- efficient and fast root file system access
- large memory footprint
- file system content needs to be restricted to reduce memory footprint
- typically used when minimal commands and libraries required
- works well for compute nodes with well defined workloads and for service nodes that are used primarily for internal services

netroot netroot is an alternative approach that mounts the root file system from a network source. It is used only for compute and login nodes. It uses overlayfs to layer tmpfs on top of a read-only network file system.

Due to the reliance on overlayfs, the decision to use netroot should include consideration of the characteristics and limitations of overlayfs in addition to those of netroot listed here.

netroot has these characteristics and limitations:

- used only for compute and login nodes, never for service nodes (except login nodes)
- slower root file system access
- increased node boot time
- minimized memory footprint (mounted from network, so requires less disk space)
- no restriction on file system content
- typically used when a robust set of commands and libraries required (netroot enables large network-based images, formerly enabled through the DSL feature)
- works well for compute nodes with diverse workloads and for compute nodes with a high memory footprint
- always used for GPUs
- supports a SquashFS compressed image format for better boot performance (recommended)

This comparison of tmpfs and netroot memory footprints is based on a fresh install with nothing extra added. These numbers could be larger or smaller for a site depending on whether the Cray image recipes for tmpfs and netroot have been extended (by adding necessary RPMs) or reduced (by removing unnecessary RPMs).

Table 9. Comparison of tmpfs and netroot Memory Footprints

Image Type	Memory Consumption	Number of RPMs
Admin image root - tmpfs	1400 MB	600
Service image root – tmpfs	1700 MB	670
Login image root – tmpfs	3600 MB	1100
Compute image root – tmpfs	1500 MB	745
Login image root – netroot	125 MB	2500
Compute image root – netroot	150 MB	2380

Install Third-Party Software with a Custom Image Recipe

About this task

Any software that is created independent from Cray *and* that is not delivered with a Cray system is third-party software that administrators install as add-ons to the Cray system. (The information in this section does not pertain to software installed on an external file system that is connected to a Cray system.) There are several ways to install third-party software:

- (Recommended) Create a custom image recipe for the third-party software and add a Cray-provided recipe as a subrecipe (also called *extending a recipe*). This method is preferred because the update to the image is persisted in the recipe.

- Clone an existing recipe, then modify the clone to add the third-party software. This method is not recommended because cloned recipes do not receive updates from patches.
- Use the `chroot` command to install the software to an existing image. Software installed with this method is lost when a node image is rebuilt from a recipe. However, this approach can be useful when persistence is not important, such as when testing third-party software.
- Use the `zypper` command to install software on a node. Software installed with this method is lost the next time the node is booted. Like the `chroot` method, this approach can be used when testing software that does not need to persist in the image.

  IMPORTANT: Do not directly modify a Cray-provided recipe.

This procedure describes the recommended method of creating a new image recipe for third-party software that will run on Cray nodes (except eLogin nodes). The procedure explains how to add a Cray-provided image recipe as a subrecipe, then add the third-party repositories, package collections, and RPMs, as well as optional non-RPM content. It then shows how to build an image root, export the image root into a boot image, push the boot image to the boot node (netroot only), test it on a single node, and assign the tested image to all applicable nodes.

For more information on image-related concepts and commands, see *About the Image Management and Provisioning System (IMPS)* on page 169.

NOTE: This procedure does not apply to eLogin images. To create, build, and transfer custom eLogin images, see the *XC Series eLogin Installation Guide*.

Procedure

CREATE REPOSITORY

1. Create a new repository and add the third-party packages (RPMs). Skip this step if the repository already exists on the SMW or is hosted on a remote repository server.

   a. Use the `repo create` command to create the new repository (for example, `my_sles12_repo`). This command requires the architecture (such as `x86-64`) and operating system type (either `SLES12` or `CentOS`).

   ```
   smw# repo create --arch x86-64 --type SLES12 my_sles12_repo
   ```

   b. Verify that the new repository was created.

   ```
   smw# repo list my*
   my_sles12_repo
   ```

   c. Add the third-party RPMs to the repository. This example takes all RPMs starting with `myrpm` in the example repository path */path/to/repos/* and copies them to the example repo `my_sles12_repo`.

   ```
   smw# repo update -a "/path/to/repos/myrpm*.rpm" my_sles12_repo
   smw# ls -l /var/opt/cray/repos/my_sles12_repo
   -rw-r--r-- 1 crayadm crayadm 485137 Nov 23 08:56 myrpm-11.13.1.1-4.x86_64.rpm
   ```

   d. (Optional.) Check the contents of the repository. This command displays the packages but not the full RPM names.

   ```
   smw# repo show --fields contents
   ```

Add the `--detailed` option to display the version and architecture for each package in the repository.

e. Validate the repository.

```
smw# repo validate my_sles12_repo
```

CREATE PACKAGE COLLECTION

2. Create a package collection and add the RPM package names.

A package collection represents a logical grouping of RPMs. Cray recommends using a package collection because the RPMs can be used in multiple image types (such as compute and service node images). Package collections are stored on the SMW in `/etc/opt/cray/imps/package_collections.d/`.

a. Create an empty package collection (for example, `my_collection`).

```
smw# pkgcoll create --description "Example package collection" my_collection
```

b. Verify that the package collection was created.

```
smw# pkgcoll list my*
my_collection
```

c. Add the RPM package name or names (for example, `myrpm`) to the package collection.

```
smw# pkgcoll update -p myrpm \
--description "My package collection" my_collection
```

d. Display information about the package collection.

```
smw# pkgcoll show my_collection
my_collection:
     name: my_collection
     description: My package collection
     package_collections:
          myrpm
```

CREATE RECIPE

3. Create a new recipe and customize it by adding a subrecipe (the Cray-provided image) and the content for the third-party software.

a. List the existing recipes to determine which image recipe to include.

```
smw# recipe list
compute-large_cle_6.0up01_sles_12_x86-64_ari
compute-large_cle_6.0up02_sles_12_x86-64_ari
compute-large_cle_6.0up03_sles_12_x86-64_ari
compute-large_cle_6.0up04_sles_12sp2_x86-64_ari
compute_cle_6.0up01_sles_12_x86-64_ari
compute_cle_6.0up02_sles_12_x86-64_ari
compute_cle_6.0up03_sles_12_x86-64_ari
compute_cle_6.0up04_sles_12sp2_x86-64_ari
dal_cle_6.0up01_centos_6.5_x86-64_ari
dal_cle_6.0up02_centos_6.5_x86-64_ari
dal_cle_6.0up03_centos_6.5_x86-64_ari
dal_cle_6.0up04_centos_6.5_x86-64_ari
elogin_cle_6.0up01_sles_12_x86-64_ari
...
```

b. Create a new image recipe. This example uses the recipe name `site_compute`.

```
smw# recipe create --description \
"Example recipe for 3rd-party software on compute nodes" site_compute
```

c. Add the existing image recipe as a subrecipe. This example uses the Cray-provided recipe `compute_cle_6.0up04_sles_12sp2_x86-64_ari`.

```
smw# recipe update -i compute_cle_6.0up04_sles_12sp2_x86-64_ari site_compute
```

d. Add the package collection that contains the third-party RPMs (in this example, `my_collection`).

```
smw# recipe update -c my_collection \
--rationale "Include my package collection" site_compute
```

e. Add the repository that contains the third-party RPMs (for example, `my_sles12_repo`).

```
smw# recipe update -r my_sles12_repo \
--rationale "Include third-party RPMs" site_compute
```

To add a remote repository that is hosted on an external repository server, specify the repository's Uniform Resource Identifier (URI) starting with `http://` or `https://`.

f. Add the objects mentioned in the subrecipe that are also needed for the parent recipe.

IMPORTANT: The objects mentioned in a subrecipe are used to build that subrecipe but are not available to the parent recipe. If a package (RPM) or package collection is specified in the parent recipe, the custom recipe must explicitly contain the set of repositories where the packages can be found.

1. Determine which repository contains the necessary RPM or RPMs. This example `find` command identifies the Cray repository that contains the RPM `otherrpm`.

```
smw# find /var/opt/cray/repos -name otherrpm\* -ls
```

2. Select the correct repository:
   - Choose the repository for the image's operating system type — use a SLES repository for a SLES image recipe; use a CentOS repository for a CentOS recipe.
   - Most operating system and Cray repositories come in pairs (base and updates), such as `sles_12_x86-64` and `sles_12_x86-64_updates`. Be sure to select both the *base* and *base*`_updates` repositories if they exist.
3. Add the required repository or repositories (in this example, `otherrepo`).

```
smw# recipe update -r otherrepo \
--rationale "Additional repo for third-party software" site_compute
```

Repeat the `-r` option to add multiple repositories, such as a *base* and *base*`_updates` repository pair.

```
smw# recipe update -r sles_12_x86-64 \
-r sle-server_12sp2_x86-64 -r sle-server_12sp2_x86-64_updates \
--rationale "SLES12 update repo" site_compute
```

g. (Optional.) Add post-build actions by manually editing the image recipe in `/etc/opt/cray/imps/image_recipes.d/image_recipes.local.json`. In this file, locate the image recipe definition for the custom image (for example, `site_compute`).

Post-build actions can add non-RPM content (files or directories) to the image or specify commands to run in the chroot environment of the image root (on the SMW). For example, the post-build actions could include copying a tar file into the image, then using `chroot` to run the commands to untar it and run an install script.

- In the `postbuild_chroot` section, add the commands to run in a `chroot` environment for this image root.
- In the `postbuild_copy` section, add the files to copy into the image.

```
smw# vi /etc/opt/cray/imps/image_recipes.d/image_recipes.local.json
"site_compute": {
   "description": "Example recipe for 3rd-party software on compute nodes",
   "package_collections": { ... }
   "packages": { ... },
   "postbuild_chroot": [
            "command1",
            ...
            "commandN"
      ],
   "postbuild_copy": [
            "/file/1",
            ...
            "/dir/2/content"
      ],
   ”recipes": [ ... ]
   "repositories": { ... },
   },
```

TIP: Post-build scripts can use the following environmental variables:

- `IMPS_IMAGE_NAME`
- `IMPS_VERSION`
- `IMPS_IMAGE_RECIPE_NAME`
- `IMPS_POSTBUILD_FILES`

h. Validate the image recipe.

```
smw# recipe validate site_compute
INFO - Validating Image 'site_compute-validate-timestamp'
...
INFO - Calling package manager to validate Recipe 'site_compute'; this can
take a few minutes
Removed Image 'site_compute-validate-timestamp'
INFO - Recipe validates.
```

This command checks that the JSON syntax of the image recipe is correct. It also validates all repositories and package collections referenced by the image recipe, checks that all required packages are included in the recipe, and ensures that it can access any files in the `postbuild_copy` section.

BUILD AND PACKAGE IMAGE

4. Build the image recipe to create the image root. Choose a unique name for the image root. Cray recommends using the image recipe name plus the current date/time. This example shows the image root name `site_compute_timestamp`.

IMPORTANT: If the image root name is not unique, it will overwrite an existing image root. A unique name is especially important for images that are pushed to the boot node. Do not overwrite the image root that is currently used by running nodes.

The `image create` command builds the image recipe starting with the package manager installation and then proceeds to step through the postbuild copy and `chroot` commands (in that order).

```
smw# image create -r site_compute site_compute_timestamp
INFO - Repository 'my_sles12_repo' validates.
INFO - Recipe 'site_compute' is valid for building.
INFO - Calling Package manager to build new image root; this will take a few
minutes.
INFO - Rebuilding RPM database for Image 'site_compute_timestamp'.
INFO - RPM database does not need to be rebuilt.
INFO - Running post-build scripts for Image 'site_compute_timestamp'.
INFO - Copying postbuild files to /tmp/tmpmAyzGl in Image
'site_compute_timestamp'
INFO - * Executing post-build chroot script: 'chroot_command1'
INFO - post-build chroot script output will be located in /tmp/
site_compute_postbuild_out_20150713-15:55:11g4WA6p
INFO - Build of Recipe 'site_compute' has completed successfully.
```

5. (Optional.) Display the build history of the image root.

```
smw# image show site_compute_timestamp
 site_compute_timestamp:
         name: site_compute_timestamp
         created: 2016-07-13T15:54:06
         history:
              2016-07-13T15:55:16:        Successful build of Recipe
                 'site_compute into Image 'site_compute_timestamp'.
              2016-07-13T15:55:17:        Successful rebuild of RPM database.
          path: /var/opt/cray/imps/image_roots/site_compute_timestamp
```

6. Package the image root into a boot image.

```
smw# image export site_compute_timestamp

INFO - Copying kernel /var/opt/cray/imps/image_roots/site_compute_timestamp/boot/
bzImage-3.12.28-4.6_1.0000.8685-cray_ari_c into /tmp/temp_tempfs_50LJ93/DEFAULT
INFO - Copying parameters file /var/opt/cray/imps/image_roots/site_compute_timestamp/
boot/parameters-ari_c into /tmp/temp_tempfs_50LJ93/DEFAULT
 .
 .
 .
INFO - Image 'site_compute_timestamp' has been packaged into /var/opt/cray/imps/
boot_images/site_compute_timestamp.cpio.
```

The `image export` command displays the boot image file name at the end of the output. This `cpio` file is used in the next step.

7. If this is a netroot image, push the image to the boot node.

IMPORTANT: Before pushing the image, make sure that there is sufficient space on the boot node in `/var/opt/cray/imps/image_roots`.

```
smw# image sqpush site_compute_timestamp --destination boot
```

The `image sqpush` command puts a SquashFS compressed boot image on the boot node. Cray recommends using this command instead of `image push` for better boot performance.

TEST IMAGE

8. Test the new boot image on a single node.

a. Assign the boot image to a node with the NIMS `cnode` command. This example assigns the boot image file `site_compute_`*`timestamp`*`.cpio` (in the directory `/var/opt/cray/imps/boot_images/`) to the compute node with the cname `c0-0c0s15n3`.

- For a tmpfs image:

```
smw# cnode update -i \
/var/opt/cray/imps/boot_images/site_compute_timestamp.cpio c0-0c0s15n3
```

- For a netroot image:

```
smw# cnode update c0-0c0s15n3 \
--set-parameter netroot=site_compute_timestamp
```

b. Warm-boot the node to test the boot image.

```
smw#  xtcli shutdown c0-0c0s15n3
.
.
.
crayadm@smw> xtbootsys --reboot \
-r "testing new boot image site_compute_timestamp" c0-0c0s15n3
```

ASSIGN IMAGE TO NODES

9. Change the NIMS map to assign the new image to the applicable nodes.

a. Back up the current map before changing to the new image. First, identify the active map.

```
smw# cmap list | grep -i 'true'
```

The following steps use the active map name "*current-map*".

b. Next, clone the current map.

```
smw# cmap create -clone current-map new-map
```

c. Mark the new map as the active map.

```
smw# cmap setactive new-map
```

d. Assign the new boot image to all applicable nodes. This example uses "`--group compute`" to assign the image to all compute nodes.

- For a tmpfs image:

```
smw# cnode update --group compute \
-i /var/opt/cray/imps/boot_images/site_compute_timestamp.cpio
```

- For a netroot image:

```
smw# cnode update --group compute \
--set-parameter netroot=site_compute_timestamp
```

Trouble? If problems occur, use this command to revert to the previous map (*current-map*):

```
smw# cmap setactive current-map
```

10. Choose when the nodes should switch to the new image.

- To immediately use the new image, warm-boot all applicable nodes with the new image. This example specifies the compute nodes as a comma-separated list of cnames; see the `xtcli(8)` man page for other ways of specifying multiple nodes.

```
smw# xtcli shutdown cname, cname, ... cname
.
.
.

smw# xtbootsys --reboot -r "Booting custom image on all compute nodes" \
cname, cname, ... cname
```

- To have the workload manager (WLM) reboot the node once the current user's job finishes, see *Apply Rolling Patches to Compute Nodes with cnat* on page 276.
- Otherwise, wait until the next full system reboot. The nodes will boot with the new image.

After a recipe has been defined and tested, the `imgbuilder` command can be used to rebuild and package boot images.

About Config Set Caching

Config sets are defined and reside on the Server of Authority, which on XC systems is the SMW. Config set content is made available to all nodes in the system by means of Cray Scalable Services.

To make the sharing of config set content both quick and reliable, the cray-cfgset-cache service was created. It caches config sets locally on nodes (compressed for a smaller footprint). On the SMW, it does the following:

- notices changes to config sets on the SMW
- refreshes the local caches dynamically
- detects failures and retries automatically

The cray-cfgset-cache service ensures that config set content gets refreshed on all nodes whenever config sets are created or updated on the SMW. It is triggered when `cray-ansible` is run on a node with the `start`, `restart`, or `link` commands.

> **ATTENTION:** If the cray-cfgset-cache service is stopped, config set content in node-local memory will not get refreshed when `cray-ansible` is run. If that happens, nodes will continue to use the most recent compressed copy of the config set data created before the service was stopped.

What Gets Cached

The cray-cfgset-cache service does not copy an entire config set to node-local memory. Instead, it uses the config set on the SMW to create these two files in the root of the config set:

- a compressed copy of the config set using SquashFS tools, (typically < 3 MB)
- a checksum of the compressed copy of the config set

The compressed copy is made available (effectively copied) to node-local RAM, and the checksum is used to know when the config set in node-local memory no longer matches the config set on the SMW. Even though Scalable Services makes the entire config set directory structure on the SMW available to the rest of the system, only the compressed copy and its associated checksum are used by nodes. They are the key to the performance, scalability, and reliability improvements provided by config set caching.

When `cray-ansible` is run on a node, the node will do the following:

1. Check to see if the cached node-local version of the compressed config set is out of date.
2. If it is stale, replace it with a newer version available on the SMW and start using that newer version.

Add Kernel Watch Descriptors to Improve Config Set Caching Performance

About this task

Config set caching is a mechanism for exporting config set data to nodes quickly and reliably. The cray-cfgset-caching service operates on kernel watch descriptors to automatically generate the config set compressed copy and checksum files. The performance of this service depends on the number of config sets created and the number of directories within each config set. Service startup performance is affected by large numbers of config sets and the availability of kernel watch descriptors. Additional watch descriptors may be required to provide coverage for large numbers of config sets.

For more information about config set caching, see *About Config Set Caching* on page 180.

Procedure

1. Determine how many watch descriptors are in use.

```
smw# cd /var/opt/cray/imps/config/sets
smw# find . -type d | wc
```

2. Increase the total number of available watch descriptors, if desired.

```
smw# sysctl fs.inotify.max_user_watches=524288
```

Change a File on a Compute Node

About this task

System administrators sometimes need to change files such as `modprobe.conf`, `fstab`, and `nodehealth.conf` on compute nodes. For example, to tune DataWarp or Lustre, the `modprobe.conf` file might need to be changed. Cray provides configuration templates and Ansible plays for most Cray services (such as `cray_net`, `cray_rsip`, `cray_node_health`, and `cray_dvs`), which generate or change such files automatically as part of the boot process or after reconfiguring a service. If no Cray-provided play exists to make the needed changes or an existing play does not cover a needed use case, administrators can change these files directly.

There are three general methods of changing a file on a compute node:

- Option 1: Use `chroot`, either in a custom image recipe (recommended) or after building an image.
- Option 2: Use the Cray-provided Simple Sync service.
- Option 3: Write an Ansible play that either changes the file directly or runs a script to change the file.

It is important to understand the pros and cons of each method.

Option 1: `chroot`

Use `chroot` to change the file in an image root using one of these methods:

Option 1a: (Recommended) Create a custom image recipe that includes `postbuild_chroot`. This method is preferred because the changed file persists in image roots and boot images. Every time the recipe is built into an image root, the changed file will be there. When the image root is packaged into a boot image, the boot image will still have this content.

Option 1b: After building an image, use `chroot` to navigate into the image root and put the file there (or merge it with an existing file). With this method, the changed file does not persist when rebuilding an image root and then packaging it into a boot image.

Pros:

- Works well for static files
- Done on the SMW

Cons:

- Option 1b (manual `chroot`): Must repeat change each time an image recipe is rebuilt or an image root is packed into a boot image.

For more information: *Use a Custom Image Recipe to Change a File on a Compute Node* on page 184

Option 2: Simple Sync

Use the Cray-provided Simple Sync service.

Pros:

- Easy to do: just put a file in a directory and turn on the Simple Sync service
- Done on the SMW
- Can specialize targets to a limited set of targets: by class, cname, node groups, or hostname (hostnames used for non-Cray platforms that do not have cnames)
- Works best for providing access during run time to small admin tools (e.g., a widget or script) and third-party software

Cons:

- Simple Sync writes the file to the desired place without regard for what may already be there, so must know what else touches the file (such as other Ansible plays)

For more information: *About Simple Sync* on page 161

Option 3: Ansible play

Create an Ansible play using one of these methods:

Option 3a: (Recommended) Use the Ansible module `lineinfile` (a directive) to change a file directly.

Option 3b: Use the Ansible module `shell` to run a script to change the file.

Pros:

- Done on SMW in config set
- Can specialize the target nodes further than possible with Simple Sync; can specify any grouping of nodes

- Can choose when the Ansible play is run during the boot sequence
- Can edit or replace a file programmatically, but be careful to not delete something that needs to be there
- Once a play is set up and tested, it is easy to maintain
- Easily scales to large systems
- If play is written at a high enough level of abstraction, can reuse for different systems by using node groups or by changing the target node list in the play

Cons:

- Requires some knowledge of the boot process (ordering, timing)
- More work up front to set up a play
- Plays/scripts must be tested

For more information: *Use an Ansible Play to Change a File on a Compute Node* on page 183.

Use an Ansible Play to Change a File on a Compute Node

About this task

This procedure describes how to change a file on a compute node with an Ansible play, which can use several methods. For other ways to accomplish this task, see *Change a File on a Compute Node* on page 181.

For information on using Ansible on a Cray system, see the *XC Series Ansible Play Writing Guide*.

Procedure

1. Choose one of these methods for the play content.
   - Option 1: Use the Ansible module `lineinfile` (a directive) to change a file directly.
   - Option 2: Use the Ansible module `shell` to run a script to change the file.
2. Write the Ansible play.
   - Option 1: Use the Ansible module `lineinfile` to change a file directly. This example changes a specific line in `/etc/fstab`.

```
# change_file.yaml
# Use Ansible modules to do individual steps
# (e.g., add a line to a file)

- hosts: localhost

  vars: # Cray-provided node "facts"
      in_init: ansible_local.cray_system.in_init
      is_compute: ansible_local.cray_system.platform == "compute"

  tasks:
  - name: add mount to fstab
    lineinfile:
      dest=/etc/fstab
      regexp="^172.30.79.66:/home/users"
```

```
      line="172.30.79.66:/home/users /home/users nfs nfsvers=3,noacl 0 0"
      backup=yes
    when: in_init and is_compute
```

In the task, `lineinfile` specifies the the file to be changed (`/etc/fstab`) and how to change it. More information about the Ansible `lineinfile` module is at: *http://docs.ansible.com/ansible/service_module.html*.

- Option 2: Use the Ansible module `shell`. The following example runs a script named `site_script.sh` to change the file.

```
# change_file.yaml
# Use the shell directive to do everything in a script

- hosts: localhost
  vars: # Cray-provided node "facts"
      in_init: ansible_local.cray_system.in_init
      is_compute: ansible_local.cray_system.platform == "compute"

  tasks:
  - name: run my script on all service nodes
    shell: /etc/opt/cray/config/current/dist/site_script.sh
    when: in_init and is_compute
```

Note that the config set path on the SMW, such as `/var/opt/cray/imps/config/sets/p0/dist` for the `p0` config set, will appear on the node as `/etc/opt/cray/config/current/dist`.

3. Put the Ansible play and any supporting content into the config set on the SMW.

/var/opt/cray/imps/config/sets/p0/ansible/	Location in config set p0 for site Ansible plays, like this new `change_file.yaml`.
/var/opt/cray/imps/config/sets/p0/dist/	Location in config set `p0` for content that supports or is used by site Ansible plays. If using the Ansible `script` directive, as in option 2, put the site script (`site_script.sh`) here.

4. Test the new Ansible play by running it manually on a compute node.

```
node# /etc/init.d/cray-ansible change_file.yaml
```

This Ansible play will be available on all nodes any time that `cray-ansible` is run on the node to pull new config set data to the node, order Ansible plays, and then run the Ansible plays. When the system boots, this play will run on all nodes, and the conditional (`when`) clauses will determine whether a particular task will execute on any given node.

Use a Custom Image Recipe to Change a File on a Compute Node

About this task

This procedure describes how to change a file on a compute node by using a custom recipe with post-build actions (`postbuild_copy` and `postbuild_chroot`). For other ways to accomplish this task, see *Change a File on a Compute Node* on page 181.

IMPORTANT: Do not directly modify a Cray-provided recipe.

Procedure

1. List the existing recipes to determine which image recipe to include.

```
smw# recipe list
compute-large_cle_6.0up01_sles_12_x86-64_ari
compute-large_cle_6.0up02_sles_12_x86-64_ari
compute-large_cle_6.0up03_sles_12_x86-64_ari
compute-large_cle_6.0up04_sles_12sp2_x86-64_ari
compute_cle_6.0up01_sles_12_x86-64_ari
compute_cle_6.0up02_sles_12_x86-64_ari
compute_cle_6.0up03_sles_12_x86-64_ari
compute_cle_6.0up04_sles_12sp2_x86-64_ari
dal_cle_6.0up01_centos_6.5_x86-64_ari
dal_cle_6.0up02_centos_6.5_x86-64_ari
dal_cle_6.0up03_centos_6.5_x86-64_ari
dal_cle_6.0up04_centos_6.5_x86-64_ari
elogin_cle_6.0up01_sles_12_x86-64_ari
...
```

2. Create a new image recipe. This example uses the recipe name `site_compute`.

```
smw# recipe create --description \
"Example recipe for 3rd-party software on compute nodes" site_compute
```

3. Add the existing image recipe as a subrecipe. This example uses the Cray-provided recipe `compute_cle_6.0up04_sles_12sp2_x86-64_ari`.

```
smw# recipe update -i compute_cle_6.0up04_sles_12sp2_x86-64_ari site_compute
```

4. Add post-build actions by manually editing the image recipe in `/etc/opt/cray/imps/image_recipes.d/image_recipes.local.json`. In this file, locate the image recipe definition for the custom image (for example, `site_compute`).

   Post-build actions can add non-RPM content (files or directories) to the image or specify commands to run in the chroot environment of the image root (on the SMW). For example, the post-build actions could include copying a tar file into the image, then using `chroot` to run the commands to untar it and run an install script.

   - In the `postbuild_chroot` section, add the commands to run in a `chroot` environment for this image root.
   - In the `postbuild_copy` section, add the files to copy into the image.

```
smw# vi /etc/opt/cray/imps/image_recipes.d/image_recipes.local.json
"site_compute": {
        ...
        "postbuild_chroot": [
            "chroot_command1",
            ...
            "chroot_commandN"
        ],
        "postbuild_copy": [
            "/file/1",
            ...
            "/dir/2/content"
        ],
        ”recipes": { ... },
```

```
        ...
    },
```

TIP: Post-build scripts can use the following environmental variables:

- IMPS_IMAGE_NAME
- IMPS_VERSION
- IMPS_IMAGE_RECIPE_NAME
- IMPS_POSTBUILD_FILES

5. Validate the image recipe.

```
smw# recipe validate site_compute
INFO - Validating Image 'site_compute-validate-timestamp'
...
INFO - Calling package manager to validate Recipe 'site_compute'; this can take
a few minutes
Removed Image 'site_compute-validate-timestamp'
INFO - Recipe validates.
```

This command checks that the JSON syntax of the image recipe is correct. It also validates all repositories and package collections referenced by the image recipe, checks that all required packages are included in the recipe, and ensures that it can access any files in the `postbuild_copy` section.

6. Build the image recipe to create the image root. Choose a unique name for the image root. Cray recommends using the image recipe name plus the current date/time. This example shows the image root name `site_compute_timestamp`.

IMPORTANT: If the image root name is not unique, it will overwrite an existing image root. A unique name is especially important for images that are pushed to the boot node. Do not overwrite the image root that is currently used by running nodes.

The `image create` command builds the image recipe starting with the package manager installation and then proceeds to step through the postbuild copy and `chroot` commands (in that order).

```
smw# image create -r site_compute site_compute_timestamp
INFO - Repository 'my_sles12_repo' validates.
INFO - Recipe 'site_compute' is valid for building.
INFO - Calling Package manager to build new image root; this will take a few
minutes.
INFO - Rebuilding RPM database for Image 'site_compute_timestamp'.
INFO - RPM database does not need to be rebuilt.
INFO - Running post-build scripts for Image 'site_compute_timestamp'.
INFO - Copying postbuild files to /tmp/tmpmAyzGl in Image
'site_compute_timestamp'
INFO - * Executing post-build chroot script: 'chroot_command1'
INFO - post-build chroot script output will be located in /tmp/
site_compute_postbuild_out_20150713-15:55:11g4WA6p
INFO - Build of Recipe 'site_compute' has completed successfully.
```

7. (Optional.) Display the build history of the image root.

```
smw# image show site_compute_timestamp
 site_compute_timestamp:
         name: site_compute_timestamp
         created: 2016-07-13T15:54:06
         history:
              2016-07-13T15:55:16:        Successful build of Recipe
```

```
                'site_compute into Image 'site_compute_timestamp'.
            2016-07-13T15:55:17:        Successful rebuild of RPM database.
         path: /var/opt/cray/imps/image_roots/site_compute_timestamp
```

8. Package the image root into a boot image.

```
smw# image export site_compute_timestamp

INFO - Copying kernel /var/opt/cray/imps/image_roots/site_compute_timestamp/boot/
bzImage-3.12.28-4.6_1.0000.8685-cray_ari_c into /tmp/temp_tempfs_50LJ93/DEFAULT
INFO - Copying parameters file /var/opt/cray/imps/image_roots/site_compute_timestamp/
boot/parameters-ari_c into /tmp/temp_tempfs_50LJ93/DEFAULT
 .
 .
 .
INFO - Image 'site_compute_timestamp' has been packaged into /var/opt/cray/imps/
boot_images/site_compute_timestamp.cpio.
```

The `image export` command displays the boot image file name at the end of the output. This `cpio` file is used in the next step.

9. If this is a netroot image, push the image to the boot node.

> **IMPORTANT:** Before pushing the image, make sure that there is sufficient space on the boot node in `/var/opt/cray/imps/image_roots`.

```
smw# image sqpush site_compute_timestamp --destination boot
```

The `image sqpush` command puts a SquashFS compressed boot image on the boot node. Cray recommends using this command instead of `image push` for better boot performance.

10. Test the new boot image on a single node.

a. Assign the boot image to a node with the NIMS `cnode` command. This example assigns the boot image file `site_compute_timestamp.cpio` (in the directory `/var/opt/cray/imps/boot_images/`) to the compute node with the cname `c0-0c0s15n3`.

- For a tmpfs image:

```
smw# cnode update -i \
/var/opt/cray/imps/boot_images/site_compute_timestamp.cpio c0-0c0s15n3
```

- For a netroot image:

```
smw# cnode update c0-0c0s15n3 \
--set-parameter netroot=site_compute_timestamp
```

b. Warm-boot the node to test the boot image.

```
smw#  xtcli shutdown c0-0c0s15n3
.
.
.
crayadm@smw> xtbootsys --reboot \
-r "testing new boot image site_compute_timestamp" c0-0c0s15n3
```

11. Change the NIMS map to assign the new image to the applicable nodes.

a. Back up the current map before changing to the new image. First, identify the active map.

```
smw# cmap list | grep -i 'true'
```

The following steps use the active map name "*current-map*".

b. Next, clone the current map.

```
smw# cmap create -clone current-map new-map
```

c. Mark the new map as the active map.

```
smw# cmap setactive new-map
```

d. Assign the new boot image to all applicable nodes. This example uses "`--group compute`" to assign the image to all compute nodes.

- For a tmpfs image:

```
smw# cnode update --group compute \
-i /var/opt/cray/imps/boot_images/site_compute_timestamp.cpio
```

- For a netroot image:

```
smw# cnode update --group compute \
--set-parameter netroot=site_compute_timestamp
```

Trouble? If problems occur, use this command to revert to the previous map (*current-map*):

```
smw# cmap setactive current-map
```

12. Choose when the nodes should switch to the new image.

- To immediately use the new image, warm-boot all applicable nodes with the new image. This example specifies the compute nodes as a comma-separated list of cnames; see the `xtcli(8)` man page for other ways of specifying multiple nodes.

```
smw# xtcli shutdown cname, cname, ... cname
.
.
.

smw# xtbootsys --reboot -r "Booting custom image on all compute nodes" \
cname, cname, ... cname
```

- To have the workload manager (WLM) reboot the node once the current user's job finishes, see *Apply Rolling Patches to Compute Nodes with cnat* on page 276.
- Otherwise, wait until the next full system reboot. The nodes will boot with the new image.

About Custom Ansible Plays

The following procedures provide examples of tasks that can be done with Ansible plays within the Cray Management System (CMS). In most cases, there are several ways to accomplish the same task. For example, a site could choose to keep using a favorite script within the Ansible framework, convert it to an Ansible play, or use it outside of the framework. Using an Ansible play (or a script within the Ansible framework) is useful for sites with a large number of Cray nodes.

For XC systems with release CLE 6.0 and later, Cray uses Ansible to orchestrate the boot sequence and configuration. Configuration for all applications installed in an image is applied at boot time using `cray-ansible`, a wrapper that finds all Ansible plays installed on the system and executes them with Ansible. Configuration content is centralized in a config set located on the SMW. All content within the config set is accessible by every

CLE node on the system, which is how configuration information is distributed throughout the system. For more information, see *Cray XC System Configuration* on page 129.

For information on using Ansible on a Cray system, see the *XC Series Ansible Play Writing Guide*.

Built-in and Cray-supplied Facts

Ansible provides access to facts about the system—network interfaces, disks, operating system version, and so forth—for use within each Ansible play ("built-in" facts). In addition, Cray provides access to facts that are Cray-specific, such as nid name, cname, node type (smw, sdb, boot, etc.), for use within each Ansible play.

To view the available built-in facts, run this command on the node where the Ansible plays will run:

```
# ansible hostname -m setup
```

To view the Cray-supplied facts, run this Python script:

```
# /etc/ansible/facts.d/cray_system.fact
```

Elements in a Typical Ansible Play

The example Ansible plays in this publication contain the following elements:

hosts Specifies where the play will run. In Cray systems, this is typically set to `localhost`, because unlike many configuration management tools that push information out to nodes from a centralized location, Ansible (as used by Cray in this release) pulls information to the local node and runs all plays there.

vars Defines variables scoped to the play and all other plays that come after it. Ansible built-in and Cray-supplied facts are accessible without having to define these variables; however, it is good practice to define variables using the provided facts because they can be assigned shorter names and can be set to useful boolean values.

tasks Each Ansible play `task` is like a line in a Bash script. Each task must have a name, a directive or module, and a conditional (a `when` clause), which indicates the conditions under which the play should execute that task on the node.

For a description of all Ansible modules and their arguments, see the Module Index in the *Ansible Documentation* website. Look for modules for Ansible 1.9.2 and earlier.

Ansible is run twice during the boot of a CLE node: first from the `/init` script (referred to as “in init”) before Linux `systemd` starts, then again after Linux `systemd` starts (referred to as “booted” and also as “not in init”). When running plays that control processes, it is usually best to avoid running plays in init. To accomplish that, use the Cray-provided fact `ansible_local.cray_system.in_init` which is true if “in init” and false if in “booted”. To ensure a play is not run in init, but only in the booted phase, use this in a conditional statement:

```
not ansible_local.cray_system.in_init
```

Play Ordering

The `cray-ansible` command gathers all Ansible plays and determines the order in which they are executed. The following directives allow control over the play order:

- `run_before` and `run_after`
- `run_early` and `run_late`

The `run_before` and `run_after` directives are Cray-provided ways to specify dependencies between plays. A variable can be set to a single play or list of Ansible plays that the play should precede or follow.

The `run_early` and `run_late` directives specify a play-ordering group. Plays that specify `run_early: true` are run first, then plays that don't specify a `run_` directive, and finally plays that specify `run_late: true`. However, plays that call out dependencies with `run_before` and `run_after` directives take precedence over all other play-ordering mechanisms.

Drop Zone for Custom Ansible Plays

Within a config set, Cray provides a drop zone for customers to place Ansible plays and other content referenced by those plays. By default, the Ansible drop zone is on the SMW in these directories:

/var/opt/cray/imps/config/sets/*<config_set>*/ansible/	Location in config set for site Ansible plays.
/var/opt/cray/imps/config/sets/*<config_set>*/dist/	Location in config set for content that supports or is used by site Ansible plays, such as a script used in an Ansible play.

In the following procedures, the example config set `p0` is for the entire system. For a partitioned system, substitute the partition number (`p`*N*) in the example path. For an Ansible play that must run on multiple partitions, put the play and associated content in the config set for each applicable partition.

Ansible Logs

Ansible logs under `/var/opt/cray/log/ansible` are collected via `cdump` and `xtdumpsys`. For more information, see *Debug Ansible Failures During System Boot* on page 44.

Control a Service on Specific Nodes at Boot Time

Using a custom Ansible play is the best way to start or stop a service on specified nodes. This example ensures that `cron` runs on login nodes only. Because `cron` starts automatically on all nodes at boot time, this example play stops `cron` on nodes that are not login nodes.

```
# manage_cron_service.yaml
- hosts: localhost
  vars:
      in_init: ansible_local.cray_system.in_init
      is_login: ansible_local.cray_system.hostid |
ismember(cray_login.settings.login_nodes.data.member_groups)

  tasks:
  - name: stop cron on non-login nodes
    service:
      name: cron
      state: stopped
    when: not is_login and not in_init
```

The `cron` service starts on all nodes, so this example play causes `cron` to boot in a stopped state on all nodes except the login nodes.

In the task, `service` specifies the service to be started (`cron`) and what to do to the service. More information about the Ansible `service` module is at: *http://docs.ansible.com/ansible/service_module.html*.

The conditional statement (`when`) specifies the nodes where the action is to be taken and when to run the play.

- The Cray-provided fact `ansible_local.cray_system.in_init` indicates when to run the play. Ansible is run twice during the boot of a CLE node: first from the `/init` script (referred to as “in init”) before Linux `systemd` starts, then again after Linux `systemd` starts (referred to as “booted” and also as “not in init”). When running plays that control processes, it is usually best to avoid running plays in init. To accomplish that, use the Cray-provided fact `ansible_local.cray_system.in_init` which is true if “in init” and false if in “booted”.
- The `ansible_local.cray_system.hostid` string is a Cray-supplied Ansible fact that corresponds to the cname on CLE nodes.
- To determine if the node running this play is a login node, the Cray-supplied Ansible filter `ismember` is used to determine if the current node is a member of the node groups that are configured as login nodes.

When this play runs on a node, `cron` is stopped if the node is not a login node.

Manage Node Configuration, Services, and Settings at Boot Time (boot.last Script)

About this task

In previous releases, many Cray customer sites used a `boot.last` script, or something like it, to start up and manage additional services, configurations, and settings, such as tuning Lustre, starting a secondary `sshd` for a customer network, starting a workload manager, or setting up service nodes to talk to other service nodes using `ssh`. This script would run last on each service node as it booted. Its value lay partly in enabling sites to specialize nodes and/or classes in a scalable way.

This procedure shows two ways to accomplish the same thing using Ansible.

Procedure

1. Choose how to accomplish the purpose of the original `boot.last` script.
   - Option 1: Write an Ansible play that uses Ansible modules (directives, a bit like function calls) to do individual steps equivalent to the lines in the `boot.last` script.
   - Option 2: Write an Ansible play that simply uses the `shell` directive (an Ansible module) to run the original `boot.last` script.
   - Option 3: Run the `boot.last` script outside the Ansible framework, after the system nodes have finished booting.

   Options 1 and 2 are executed when `cray-ansible`, which is a wrapper around Ansible, gathers all Ansible plays into a master playbook and then runs that playbook. Option 3 would occur after the system has booted. Because Option 3 does not use the Ansible framework, it is not described further in this procedure.

2. Write the Ansible play.
   - Option 1: Use Ansible modules to perform individual steps. This example play starts a service named `myserviced`. There could be many other tasks that a site would want to include in a `boot.last` play.

```
# boot.last.yaml
# Option 1: Use Ansible modules to do individual steps (e.g., start a
service)

    - hosts: localhost
      vars:       # Cray-provided node "facts" + config set data
        nid:      ansible_local.cray_system.nid
```

```
        is_nid7:  ansible_local.cray_system.nid == "7"
        is_login: ansible_local.cray_system.hostid |
ismember(cray_login.settings.login_nodes.data.member_groups)
        is_sdb:   ansible_local.cray_system.hostid |
ismember(cray_sdb.settings.node_groups.data.sdb_groups)
        in_init:  ansible_local.cray_system.in_init
        run_late: true

      tasks:
      - name: start myserviced service on nid0007, sdb, login nodes
        service: name=myserviced state=started args="-f /path/to/
myservice_config.conf"
        when:
          (is_nid7 or is_login or is_sdb) and not in_init
```

In this example, `run_late: true` is a Cray-provided directive for play ordering that places the current play in the last of three groups of Ansible plays that are executed by `cray-ansible`. Plays that specify `run_early: true` are run first, then plays that don't specify `run_early` or `run_late: true`, and finally plays that specify `run_late: true` are executed. However, plays that call out dependencies with the Cray-provided `run_after` and `run_before` directives take precedence over all other play ordering mechanisms.

- Option 2: Use Ansible to run a script on specified nodes: This example play uses `script` to run a site-specific script (`site_script.sh`) on all service nodes.

```
# boot.last.yaml
# Option 2: Just do everything in site_script.sh

    - hosts: localhost
      vars:       # Cray-provided node "facts" + config set data
        nid:      ansible_local.cray_system.nid
        is_nid7:  ansible_local.cray_system.nid == "7"
        is_login: ansible_local.cray_system.hostid |
ismember(cray_login.settings.login_nodes.data.member_groups)
        is_sdb:   ansible_local.cray_system.hostid |
ismember(cray_sdb.settings.node_groups.data.sdb_groups)
        in_init:  ansible_local.cray_system.in_init
        run_late: true

      tasks:
      - name: run site script on all service nodes
        shell: /etc/opt/cray/config/current/dist/site_script.sh
        when:
          (is_nid7 or is_login or is_sdb) and not in_init
```

3. Put the Ansible play and any supporting content into the config set.

/var/opt/cray/imps/config/sets/p0/ansible/	Location in config set `p0` for site Ansible plays, like this new `boot.last.yaml`.
/var/opt/cray/imps/config/sets/p0/dist/	Location in config set `p0` for content that supports or is used by site Ansible plays. If using the Ansible `script` directive (as in option 2), put the site script here.

4. Test the new Ansible play by running it manually on two nodes: one where the task should be executed (a service node), and another where the task should NOT be executed (a compute node).

```
node# /etc/init.d/cray-ansible start
```

This Ansible play will be distributed to all nodes. When the system boots, this play will run on all nodes, and the conditional (`when`) clauses will determine whether a particular task will execute on any given node.

About Secure Shell Configuration

The Cray secure shell (SSH) configuration service, which generates and manages SSH keys, provides a turnkey environment that establishes SSH functionality quickly and easily and supports basic customer needs. SSH functionality can now be established in a variety of ways that support more complex SSH configurations for both CLE and eLogin nodes. The primary changes are summarized here:

- Automatic SSH key generation can be disabled to prevent interference with site-provided configuration.

  The cray_ssh configuration service has a new flag: simple_ssh_keys. It is set to 'true' by default, which enables automatic SSH key generation/management. If this flag is set to 'false,' that functionality is disabled, and the site assumes responsibility for providing a working SSH key configuration.

- eLogin nodes can have different SSH keys.

  The cray_login configuration service has a new setting that must be set on all systems: elogin_groups. It specifies which nodes will be used as external login nodes, and it is set to the pre-populated 'elogin_nodes' node group by default.

  > **IMPORTANT: Action required.** Sites that DO NOT have eLogin nodes MUST set elogin_groups to an empty list (`[]`). Sites that DO have eLogin nodes must ensure that the node group(s) specified for elogin_groups contain ALL elogin nodes in the system. Instructions are included in the appropriate fresh install and software update procedures.

- Simple Sync and node groups are used to synchronize SSH keys.

  The location for all SSH keys is now in the Simple Sync directory structure. The new location for common keys is in the common directory, and keys for specific node groups can be placed in the associated node group directories.

  keys for CLE nodes
  - Old common key location: `./files/roles/common`
  - New common key location: `./files/simple_sync/common/files`
  - New additional key locations: `./files/simple_sync/nodegroups/`*`my_node_group`*`/files`

  keys for eLogin nodes
  - Old common key location: `./files/roles/common/elogin`
  - New common key location: `./files/simple_sync/nodegroups/`*`elogin_node_group`*`/files`

  No action required. To migrate keys to new common location, no administrative action is required. If simple_ssh_keys is 'true' (default), then when the config set is updated, keys that are in the old common location will be automatically copied to the new common location, but only if there are no keys there already. Any keys in the new common location will not be overwritten.

Basic Components

These three basic components of SSH configuration can be combined in several ways to create a wide range of SSH functionality.

SSH key generation	• [default] generated automatically by Cray • generated entirely by the site • a mixture of Cray-generated and site-generated
SSH key synchronization	• [default] synchonized automatically by Cray using Simple Sync or the Cray SSH play (only if Simple Sync disabled) • synchonized automatically using Simple Sync only • synchronized entirely by the site
sshd_config	• [default] minimally modified by the Cray SSH play • never modified by the Cray SSH play

The following use cases illustrate common combinations of these elements.

Use Case 1: [Default] Automatic SSH Key Management

By default, the Cray SSH play and automatic key management are enabled. This means:

- **Generation.** System and root user SSH keys will be automatically generated (if none are present in the common key location) when the config set is updated.
- **Synchronization.** Keys will be copied automatically from the config set onto the nodes.
- **sshd_config.** The Cray SSH play will make minimal changes to `sshd_config` to ensure that basic logins are enabled.

The behavior is identical to previous CLE 6.0 releases, except that the location in the config set of the SSH files is now in the Simple Sync directory.

Use Case 2: Site Modifies SSH Content in Simple Sync Directories

The Cray SSH play and automatic key management are enabled, as in Use Case 1, but after installation or configuration, the site administrator adds new or different content in Simple Sync directories for SSH, such as different keys for login nodes. This use case illustrates that sites can leave automatic key generation in place but still customize SSH keys in Simple Sync.

- **Generation.** Automatic key generation is enabled, as in Use Case 1, but after the admin adds site-specific content to the common key SSH key location in the Simple Sync directory, no new keys will be generated.
- **Synchronization.** Same as Use Case 1.
- **sshd_config.** Same as Use Case 1.

Use Case 3: Automatic SSH Key Management Disabled

Disabling automatic key generation and synchronization (set simple_ssh_keys to 'false' in cray_ssh config service) enables sites to have complete control over key management. A site may wish to use a configuration that has no common SSH keys, and because the absence of keys in the common location triggers the generation of new keys, the site would need to disable automatic SSH key management.

> **ATTENTION:** A site that disables automatic SSH key management assumes responsibility for providing a working SSH key configuration.

- **Generation.** No SSH keys will be automatically generated when the config set is updated, even if none are present in the common key location.
- **Synchronization.** No special synchronization will be performed for SSH keys beyond generic Simple Sync functionality.
- **sshd_config.** Same as Use Case 1.

Use Case 4: SSH Play Disabled

Disabling the Cray SSH play (set `cray_ssh.enabled: false` in cray_ssh config service) enables sites to completely replace Cray SSH configuration. The site must provide `sshd_config` as well as SSH keys. Keys may be synchronized using Simple Sync or a site-local Ansible play.

- **Generation.** Same as Use Case 3.
- **Synchronization.** Site will synchronize keys using Simple Sync or a site-local Ansible play.
- **sshd_config.** No configuration of `sshd_config` will take place.

Use Case 4-EZ: SSH Play Disabled after System Boot

Customers who wish total control over SSH and SSH keys can still leverage the Cray SSH infrastructure:

1. Boot the system with Cray SSH play and automatic key management are enabled (Use Case 1).
2. Copy `sshd_config` from the booted system into the Simple Sync directory.
3. Disable the Cray SSH play (Use Case 4).

Monitor the System

Manage Log Files Using CLE and HSS Commands

Boot, diagnostic, and other Hardware Supervisory System (HSS) events are logged on the SMW in the `/var/opt/cray/log` directory, which is created during the installation process. The time-stamped `bootinfo`, `console`, `consumer`, and `netwatch` log files are located in the `/var/opt/cray/log/`*`sessionid`* directory by default.

For example, the HSS `xtbootsys` command starts the `xtconsole` command, which redirects the output to a time-stamped log file, such as `/var/opt/cray/log/p0-20120716t104708/console-20120716`.

The `SMWinstall`, `SMWconfig`, and `SMWinstallCLE` commands create several detailed log files in the `/var/adm/cray/logs` directory. The log files are named using the PID of the `SMWinstall` or the `SMWinstallCLE` command; the exact names are displayed when the command is invoked.

CLE logs are saved on the SMW in `/var/opt/cray/log/`*`sessionid`*.

Controller logs are saved on the SMW in `/var/opt/cray/log/controller/`*`cabinet`*`/`*`controller`*`/messages-`*`yyyymmdd`*, where *`cabinet`* is of the form `c0-0`, `c1-0`, etc.; and *`controller`* is either of the form `c0-0`, `c1-0` for cabinet controllers (CC) or `c0-0c0s0` for blade controllers (BC) .

For more information, see the `intro_llm_logfiles(5)` man page.

Filter the Event Log

The `xtlogfilter` command enables the system administrator to filter the event log for information such as the time a particular event occurred or messages from a particular cabinet.

For more information, see the `xtlogfilter(8)` man page.

Finding information in the event log

For this example, search for all console messages from node `c9-2c0s3n2`:

```
crayadm@smw:~> xtlogfilter -f /var/opt/cray/log/event-yyyymmdd
c9-2c0s3n2
```

Add Entries to Log Files

The system administrator can add entries (e.g., the start or finish of system activities) to the `syslog` with the `logger` command. The entry is then available to anyone who reads the log.

For more information, see the `logger(1)` man page.

Add entries to syslog file

For this example, mark the start of a new system test:

```
login# logger -is "Start of test 4A $(date) "
Start of test 4A Thu Jul 14 16:20:43 CDT 2011
```

The system log shows:

```
Jul 14 16:20:43 nid00003 xx[21332]:
Start of test 4A Thu Jul 13 16:20:43 CDT 2012
```

Examine Log Files

Time-stamped log files of boot, diagnostic and other HSS events are located on the SMW in the `/var/opt/cray/log` directory. The time-stamped `bootinfo`, `console`, `consumer`, and `netwatch` log files are located in the `/var/opt/cray/log/`*`sessionid`* directory by default.

For example, the HSS `xtbootsys` command starts the `xtconsole` command, which redirects the output to a time-stamped log file, such as `/var/opt/cray/log/p0-20120716t104708/console-20120716`.

The `SMWinstall`, `SMWconfig`, and `SMWinstallCLE` commands create several detailed log files in the `/var/adm/cray/logs` directory. The log files are named using the PID of the `SMWinstall` or the `SMWinstallCLE` command; the exact names are displayed when the command is invoked.

Remove Old Log Files

The `xttrim` utility provides a simple and configurable method to automate the compression and deletion of old log files. The `xttrim` utility is intended to be run on the SMW from `cron` and is automatically configured to do this as part of the SMW software installation process. Review the `xttrim.conf` configuration file and ensure that `xttrim` will manage the desired directories and that the compression and deletion times are appropriate.

The `xttrim` utility does not perform any action unless the `--confirm` flag is used (to avoid unintended actions), nor will `xttrim` perform any action on open files. All actions are based on file-modified time.

For additional information, see the `xttrim(8)` and `xttrim.conf(5)` man pages.

Check the Status of System Components

Check the status of the system or a component with the `xtcli status` command on the System Management Workstation (SMW). By default, the `xtcli status` command returns the status of nodes.

The `xtcli status` command has the following form:

```
xtcli status [-n] [-m] [{-t type -a}] node_list
```

Where *`type`* may be: `cc`, `bc`, `cage`, `node`, `aries`, `aries_lcb`, `pdc`, or `qpdc`. The list must have component IDs only and contain no wild cards.

Use the `-m` option to display all nodes that were repurposed by using the `xtcli mark_node` command. (The `xtcli mark_node` command can be used to repurpose a service node to a compute role or to repurpose a compute node to a service role.)

For more information, see the `xtcli(8)` man page.

Show the status of a component

For this example, display all nodes that were repurposed using the `xtcli mark_node` command:

```
crayadm@smw:~> xtcli status -m c0-0c0
Network topology: class 2
Network type: Aries
             Nodeid: Service  Core Arch|  Comp state   [Flags]
   ------------------------------------------------------------------
          c0-0c0s2n0:       -  SB16  X86|         off   [noflags|]
          c0-0c0s3n0: service  SB16  X86|         off   [noflags|]
   ------------------------------------------------------------------
```

This shows that `c0-0c0s2n0` is a service node repurposed as a compute node, and that `c0-0c0s3n0` is a compute node repurposed as a service node.

Check the Status of Compute Processors

Use the `xtprocadmin` command on a service node to check that compute nodes are available after the system is booted.

Use the `xtprocadmin` command on a node to check that compute nodes are available after the system is booted.

Identify nodes in down or admindown state

```
nid00007:~> xtprocadmin | grep down
```

Use the user `xtnodestat` command to display the current allocation and status of each compute processing element and the application that it is running. A simplified text display shows each processing element on the Cray system interconnection network.

Display current allocation and status of each compute processing element and the application that it is running

```
nid00007:~> xtnodestat
Current Allocation Status at Wed Jul 06 13:53:26 2011

     C0-0
  n3 AAaaaaaa
  n2 AAaaaaaa
  n1 Aeeaaaa-
c2n0 Aeeaaaaa
  n3 Acaaaaa-
  n2 cb-aaaa-
  n1 AA-aaaa-
c1n0 Aadaaaa-
  n3 SASaSa--
  n2 SbSaSa--
  n1 SaSaSa--
c0n0 SASaSa--
```

```
    s01234567

Legend:
   nonexistent node                 S  service node
;  free interactive compute node    -  free batch compute node
A  allocated interactive or ccm node ?  suspect compute node
W  waiting or non-running job       X  down compute node
Y  down or admindown service node   Z  admindown compute node

Available compute nodes:          0 interactive,         15 batch

Job ID      User      Size   Age        State        command line
--- ------- --------  -----  ---------  --------  ---------------
a   3772974    user1     48      0h06m       run              app1
b   3773088    user2      2      0h01m       run              app2
c   3749113    user3      2     28h26m       run              app3
d   3773114    user4      1      0h00m       run              app4
e   3773112    user5      4      0h00m       run              app5
```

For more information, see the `xtprocadmin(8)` and `xtnodestat(1)` man pages.

Monitor the System with the System Environmental Data Collector (SEDC)

The System Environment Data Collections (SEDC) manager, `sedc_manager`, monitors the system's health and records the environmental data and status of hardware components such as power supplies, processors, temperature, and fans. SEDC can be set to run at all times or only when a client is listening. The SEDC configuration file provided by Cray has automatic data collection set as the default action.

The SEDC configuration file (`/opt/cray/hss/default/etc/sedc_srv.ini` by default) configures the SEDC server. In this file, the administrator can create sets of different configurations as groups so that the blade and cabinet controller daemons can scan components at different frequencies. The `sedc_manager` sends out the scanning configuration for specific groups to the cabinet and blade controllers and records the incoming data by group.

For information about configuring the SEDC manager, see *XC™ Series System Environment Data Collections (SEDC) Guide*.

Monitor the Health of PCIe Channels

Processors are connected to the high-speed interconnect network (HSN) ASIC through PCIe channels.

The `xtpcimon` command is executed from the System Management Workstation (SMW) and is started and run during the boot process.

Any PCIe-related errors are reported to `stdout`, unless directed to a log file.

`xtpcimon` also displays CLE-originated GHAL-based Advanced Error Reporting (AER) errors for PCIe.

If the optional `/opt/cray/hss/default/etc/xtpcimon.ini` initialization file is present, the `xtpcimon` command uses the settings provided in the file.

For more information, see the `xtpcimon(8)` man page.

Report PCIe-related errors to stdout

```
crayadm@smw:~> xtpcimon
starting
----> connection to event router made
121017 04:57:01  #############  #################   ##################
121017 04:57:01  Node           Category            Description
121017 04:57:01  #############  #################   ##################
Received all responses to request to start monitoring
121017 04:58:01  c0-0c0s7a0n1   CorrectableMemErr   0:0:0 AER Correctable: Non-fatal \
                                                      error (mask bit: 1)
121008 05:42:00  c0-0c1s6a0n2   CorrectableMemErr   Link CRC error (cnt: 3)
121008 05:43:30  c0-0c1s6a0n2   Info                Correctable/CRC error
```

Examine Activity on the HSS Boot Manager

Use the HSS `xtcli session show` command to examine sessions in the boot manager. A session corresponds to running a specific command such as `xtcli power up` or `xtcli boot`. This command reports on sessions, not daemons.

For more information, see the `xtcli(8)` man page.

View a session running on the boot manager

```
crayadm@smw:~> xtcli session show BM
```

Poll a Response from an HSS Daemon, Manager, or the Event Router

Use the HSS `xtalive` command to verify that an HSS daemon, manager, or the event router is responsive.

For more information, see the `xtalive(8)` man page.

Check the boot manager

```
crayadm@smw:~> xtalive -l smw -a bm s0
```

Validate the Health of the HSS

The `xtcheckhss` command initiates a series of tests that validate the health of the HSS by gathering and displaying information supplied by scripts located on blade controllers (BCs) and cabinet controllers (CCs). `xtcheckhss` includes the following tests:

- **Version Checker**: Reads the current version running on the L0C, QLOC, L0Ds, BC micro, CC micro, CC FPGA, CHIA FPGAs, Tolapai BIOSes, and Node BIOS. The version that is read from each device is compared to the currently installed versions on the SMW.
- **Sensor Checker**: Reads environment sensors including temperatures, voltages, currents, and other data.
- **SEEP Checker:** Reads serial electrically erasable PROMs (SEEPs) in the system. This test can report any un-initialized, zeroed, or unreadable SEEPs.

- **AOC Checker**: Reads all active optical cable (AOC) data. This test displays any outliers relative to the average data calculated by previous runs.
- **ITP Checker**: Validates the embedded ITP path
- **NTP Checker**: Reads system time on all controllers and compares them with the SMW time; displays any mismatches.
- **Control Checker**: Examines and modifies system controls.
- **Configuration Information Checker**: Reads the system hardware configuration and reports the system setup, including the blade type, daughter card type, CPU type and count, and the CPU and PDC mask.
- **PCI checker**: Checks for missing or degraded PCIe connectivity on add-in cards on an IBB. This test requires that the nodes be powered up and bounced. Any cards that do not train to the PCIe Gen or Width specified in the Link Capability register are flagged. Any cards that are reported as physically present but not seen by the node are flagged.

For complete information, see the `xtcheckhss(8)` man page.

Monitor Event Router Daemon (erd) Events

The HSS `xtconsumer` command enables the system administrator to monitor events mediated by the event router daemon `erd`, which runs passively.

Monitor for specific events

For this example, watch two events: `ec_heartbeat_stop`, which will be sent if either the node stops sending heartbeats or if the system interconnection network ASIC stops sending heartbeats, and `ec_l0_health`, which will be sent if any of the subcomponents of a blade controller report a bad health indication:

```
crayadm@smw:~> xtconsumer -b ec_heartbeat_stop ec_l0_health
```

Use the `xthb` command to confirm the stopped heartbeat. Use the `xthb` command only when actively looking into a known problem because it is intrusive and degrades system performance.

Check events except heartbeat

```
crayadm@smw:~> xtconsumer -x ec_l1_heartbeat
```

For more information, see the `xtconsumer(8)` and `xthb(8)` man pages.

Monitor Node Console Messages

The `xtbootsys` command automatically initiates an `xtconsole` session, which displays the console text of a specified node(s) or accelerator(s). The `xtconsole` command operates in a shell window and monitors the event router daemon (`erd`) for console messages. The node or accelerator ID appears at the beginning of each line. The messages are written into `/var/opt/cray/log/`*`sessionid`*`/console-`*`yyyymmdd`* where the administrator may monitor them.

The `xtconsole` utility may only have one concurrent instance.

For more information, see the `xtconsole(8)` man page.

View Component Alert, Warning, and Location History

Use the `xtcli comp_hist` command to display component alert, warning, and location history. Either an error history, which displays alerts or warnings found on designated components, or a location history may be displayed.

Display the location history for component `c0-0c0s0n1`

```
crayadm@smw:~> xtcli comp_hist -o loc c0-0c0s0n1
```

For more information, see the `xtcli(8)` man page.

Display Component Information

Use the HSS `xtshow` command to identify compute and service components. Commands are typed as `xtshow --option_name`. Combine the `--service` or `--compute` option with other `xtshow` options to limit the selection to the specified type of node.

For a list of all `xtshow --option_name` options, see the `xtshow(8)` man page.

Identify all service nodes

```
crayadm@smw:~> xtshow --service
L1s ...
Cages ...
L0s ...
         c0-0c0s0: service        X86|       ready      [noflags|]
         c0-0c0s1: service        X86|       ready      [noflags|]
         c1-0c0s0: service        X86|       ready      [noflags|]
         c1-0c0s1: service        X86|       ready      [noflags|]
         c2-0c0s1: service        X86|       ready      [noflags|]
         c2-0c1s1: service        X86|       ready      [noflags|]
Nodes ...
       c0-0c0s0n0: service        X86|       empty      [noflags|]
       c0-0c0s0n1: service  SB08  X86|       ready      [noflags|]
       c0-0c0s0n2: service  SB08  X86|       ready      [noflags|]
       c0-0c0s0n3: service        X86|       empty      [noflags|]
       c0-0c0s1n0: service        X86|       empty      [noflags|]
       c0-0c0s1n1: service  SB08  X86|       ready      [noflags|]
            .
            .
            .
Aries ...
       c0-0c0s0a0: service        X86|          on      [noflags|]
       c0-0c0s1a0: service        X86|          on      [noflags|]
       c1-0c0s0a0: service        X86|          on      [noflags|]
       c1-0c0s1a0: service        X86|          on      [noflags|]
       c2-0c0s1a0: service        X86|          on      [noflags|]
       c2-0c1s1a0: service        X86|          on      [noflags|]
AriesLcbs ...
    c0-0c0s0a0l00: service        X86|          on      [noflags|]
    c0-0c0s0a0l01: service        X86|          on      [noflags|]
```

```
    c0-0c0s0a0l02: service        X86|          on      [noflags|]
    c0-0c0s0a0l03: service        X86|          on      [noflags|]
    c0-0c0s0a0l04: service        X86|          on      [noflags|]
    c0-0c0s0a0l05: service        X86|          on      [noflags|]
    c0-0c0s0a0l06: service        X86|          on      [noflags|]
      .
      .
      .
```

Identify compute nodes in the `disabled` state

```
crayadm@smw:~> xtshow --compute --disabled
L1s ...
Cages ...
L0s ...
Nodes ...
       c0-0c2s0n3:       -        X86|    disabled      [noflags|]
      c0-0c2s11n0:       -        X86|    disabled      [noflags|]
      c0-0c2s11n3:       -        X86|    disabled      [noflags|]
      c1-0c0s11n2:       -        X86|    disabled      [noflags|]
Aries ...
AriesLcbs ...
```

Identify components with a status of not empty

```
crayadm@smw:~> xtshow --not_empty c0-0c0s0
L1s ...
             c0-0:       -           |          on      [warn|alert|]
Cages ...
L0s ...
         c0-0c0s0: service        X86|       ready      [noflags|]
Nodes ...
       c0-0c0s0n1: service  SB08  X86|       ready      [noflags|]
       c0-0c0s0n2: service  SB08  X86|       ready      [noflags|]
Aries ...
       c0-0c0s0a0: service        X86|          on      [noflags|]
AriesLcbs ...
    c0-0c0s0a0l00: service        X86|          on      [noflags|]
    c0-0c0s0a0l01: service        X86|          on      [noflags|]
    c0-0c0s0a0l02: service        X86|          on      [noflags|]
    c0-0c0s0a0l03: service        X86|          on      [noflags|]
    c0-0c0s0a0l04: service        X86|          on      [noflags|]
    c0-0c0s0a0l05: service        X86|          on      [noflags|]
    c0-0c0s0a0l06: service        X86|          on      [noflags|]
      .
      .
      .
```

Display Alerts and Warnings

Use the `xtshow` command to display alerts and warnings. Type commands as **`xtshow`** `--option_name`, where *`option_name`* is `alert`, `warn`, or `noflags`.

Alerts are not propagated through the system hierarchy, only information for the component being examined is displayed. For example, invoking the `xtshow --alert` command for a cabinet does not display an alert for a node. Similarly, checking the status of a node does not detect an alert on a cabinet.

Show all alerts on the system

```
crayadm@smw:~> xtshow --alert
```

Alerts and warnings typically occur while the HSS `xtcli` command operates; these alerts and warnings are listed in the command output with an error message. After they are generated, alerts and warnings become part of the state for the component and remain set until manually cleared.

For example, the temporary loss of a heartbeat by the blade controller may set a warning state on a chip.

For additional information, see the `xtshow(8)` man page.

Display System Network Congestion Protection Information

Two utilities help to identify the time and duration of system network congestion events, either by parsing through logs (`xtcpreport`) or in real time (`xtcptop`):

xtcpreport This command uses information contained in the given `xtnlrd` file to extract and display information related to system network congestion protection. See the `xtcpreport(8)` man page for additional information.

xtcptop This command monitors an `xtnlrd` file that is currently being updated and displays real-time system network congestion protection information, including start time, duration, and apid. See the `xtcptop(8)` man page for additional information.

To use these utilities, load the `congestion-tools` module if it is not already loaded.

```
crayadm@smw:~> module load congestion-tools
```

Clear Component Flags

Use the `xtclear` command to clear system information for selected components. Type commands as `xtclear --option_name`, where *option_name* is `alert`, `reserve`, or `warn`.

Clear all warnings in specified cabinet

For this example, clear all warnings in cabinet `c13-2`:

```
smw:~> xtclear --warn c13-2
```

Alerts, reserves, and warnings must be cleared before a component can operate. Clearing an alert on a component frees its state.

For more information, see the `xtclear(8)` man page.

Display Error Codes

When an HSS event error occurs, the related message is displayed on the SMW. The `xterrorcode` command on the SMW displays a single error code or the entire list of error codes.

Display HSS error codes

```
crayadm@smw:~>  xterrorcode errorcode
```

A system error code entered in a log file is a bit mask; invoking the `xterrorcode` *`bitmask_code_number`* command on the SMW displays the associated error code.

Display an HSS error code using its bit mask number

```
crayadm@smw:~> xterrorcode 131279
Maximum error code (RS_NUM_ERR_CODE) is 447
code = 207, string = 'Node Voltage Fault'
```

Cray Lightweight Log Management (LLM) System

The Cray Lightweight Log Management (LLM) system is the log infrastructure for Cray systems and must be enabled for systems to successfully log events. At a high level, a library is used to deliver messages to `rsyslog` utilizing the RFC 5424 protocol; `rsyslog` transports those messages to the SMW and places the messages into log files.

The LLM system relies on the *`sessionid`* that is generated by `xtbootsys`. Therefore, systems must always be booted using `xtbootsys`. If the site has multi-part boot procedures or uses manual procedures, have the process started by an `xtbootsys` session. That session can be effectively empty -- it is only needed to initiate a boot *`sessionid`*. Subsequent `xtbootsys` calls can then use `--session last` or manual processes.

By default, LLM has a log trimming mechanism enabled called `xttrim`.

> **IMPORTANT:** Do not use the `xtgetsyslog` command because it is not compatible with LLM.For additional information, see *Manage Log Files Using CLE and HSS Commands* on page 196.

For further information, see the `intro_LLM(8)` and `intro_LLM_logfiles(5)` man pages.

Debug the CLE System Debugger Using debugraw and debugmax

The `debugraw` and `debugmax` options listed in the `cray_logging_config.yaml` configurator template are for debugging the operation of the logging system itself. These settings put messages exactly as received into a log; normally, these messages are parsed, and the parsed elements are placed in a file based on a file template. These options are typically disabled and are only intended to be enabled to diagnose a logging issue.

The `debugraw` option shows the user what was sent to the SMW before being parsed. The `debugmax` option puts everything `rsyslog` knows about the message, including which host sent the message, into a file. The default value for `debugraw` and `debugmax` is `false`.

cdump and crash Utilities for Node Memory Dump and Analysis

The `cdump` and `crash` utilities may be used to analyze the memory on any Cray service node or CNL compute node. The `cdump` command is used to dump node memory to a file. After `cdump` completes, the `crash` utility can be used on the dump file generated by `cdump`.

Cray recommends executing the `cdump` utility only if a node has panicked or is hung, or if a dump is requested by Cray.

To select the desired access method for reading node memory, use the `cdump -r` *`access`* option. Valid access methods are:

`xt-bhs` The `xt-bhs` method uses a basic hardware system server that runs on the SMW to access and read node memory. `xt-bhs` is the default access method for these systems.

`xt-hsn` The `xt-hsn` method utilizes a proxy that reads node memory through the High-speed Network (HSN). The `xt-hsn` method is faster than the `xt-bhs` method, but there are situations where it will not work (for example, if the ASIC is not functional). However, the `xt-hsn` method is preferable because the dump completes in a short amount of time and the node can be returned to service sooner.

`xt-file` The `xt-file` method is used for memory dump file created by the `-z` option. The compressed memory dump file must be uncompressed prior to executing this command. Use the file name for *`node-id`*.

To dump Cray node memory, *`access`* takes the following form:

```
method[@host]
```

For additional information, see the `cdump(8)` and `crash(8)` man pages.

Resource Utilization Reporting

Resource Utilization Reporting (RUR) is an administrator tool for gathering statistics on how system resources are being used by applications or jobs. RUR is a scalable infrastructure that collects compute node statistics before an application or job runs and again after it completes. The extensible RUR infrastructure allows plugins to be easily written to collect data uniquely interesting to each site. Cray supplied plugins collect a variety of data, including process accounting, energy usage, memory usage, and GPU accounting.

When RUR is enabled on a Cray system running CLE, resource utilization statistics are gathered from compute nodes running all applications or jobs. RUR is configured to run per application, per job, or both. RUR runs primarily before an application/job has started and after it ends, ensuring minimal impact on performance.

Prior to application/job runtime, the ALPS or WLM prologue script calls an RUR prologue script that, based on enabled plugins, initiates pre-application/pre-job data staging on all compute nodes used by the application/job. This staging may involve resetting counters to zero or collecting initial values of counters. Following application/job completion, the ALPS or WLM epilogue script calls an RUR epilogue script that gathers these counters, compares them to the initial values, where applicable, stages the data on the compute nodes, and then transfers data from the compute nodes to the login/MOM node. RUR post-processes the data to create a summary report that is written out to a log file or other backing store.

Plugin Architecture

RUR supports a plugin architecture, allowing many types of usage data to be collected while using the same software infrastructure. Two basic types of RUR plugins are supported: *data plugins*, which collect particular statistics about system resources, and *output plugins*, which send the output of the RUR software stack to a backing store.

Cray supplies several plugins as part of the RUR distribution, including data collection plugins, output plugins, and an example plugin. Sites choose which plugins to enable or disable by modifying the RUR configuration file. See *Enable/Disable Plugins* on page 217 for more information. Sites can also create custom plugins, specific to their needs, as described in *Create Custom RUR Data Plugins* on page 233 and *Create Custom RUR Output Plugins* on page 235.

Overview of RUR Configuration

RUR is one of many services that store service configuration content in CLE configuration sets (*config sets*) on Cray systems. RUR can be configured when config sets are created during a fresh install or major update, or it can be configured/reconfigured later by updating existing config sets during normal system operation. Whether sites enter values in an interactive configurator session or enter values in a configuration worksheet for bulk import, the configurator takes the supplied values and ensures that they become part of the config set being created or updated.

What does RUR Need?

For RUR to function properly, the following tasks are required:

1. Enable and configure the `cray_rur` service.
2. Update the `cray_alps` service (or Slurm) to call RUR's prologue and epilogue scripts. This enables per-application RUR.
3. Modify the WLM prologue and epilogue scripts to call RUR's prologue and epilogue scripts. This enables per-job RUR.
4. Refresh CLE nodes with updated config set.

Enable and Configure RUR

Prerequisites

This procedure assumes that the user has generated configuration worksheets and is editing the RUR configuration worksheet (`cray_rur_worksheet.yaml`). If new worksheets need to be generated, use this procedure:

1. Generate up-to-date worksheets for config set *p0* (merges any new service packages installed on the system with data already in config set *p0*).

   ```
   smw# cfgset update --mode prepare --no-scripts p0
   ```

2. Locate the newly generated worksheets and copy them to a new location.

   ```
   smw# cfgset show --fields path p0
   p0:
     path: /var/opt/cray/imps/config/sets/p0
   smw# cp /var/opt/cray/imps/config/sets/p0/worksheets/* /some/edit/location
   ```

3. Edit the RUR worksheet.

```
smw# vi /some/edit/location/cray_rur_worksheet.yaml
```

About this task

This procedure identifies both necessary and optional settings for RUR to function properly. The following steps correspond to the configuration settings available in the RUR worksheet, and step numbering reflects the order in which those settings appear.

TIP: The default values assigned for settings are sufficient for an initial install.

Procedure

1. Edit `cray_rur_worksheet.yaml`.

2. Uncomment `cray_rur.enabled` and set it to `true`.

```
# Enable 'cray_rur' Service? (boolean, level=basic)
cray_rur.enabled: true
#
#********************* END Service Enable/Disable *******************
```

3. Uncomment the lines corresponding to the `base` settings. Review the guidance information and default value for each setting to determine whether or not to modify it.

```
#
cray_rur.settings.base.data.debug_level: ERROR
#
```

```
#
cray_rur.settings.base.data.keep_temp_files: false
#
```

```
#
cray_rur.settings.base.data.use_json: false
#
```

4. Uncomment the lines corresponding to the `rur_stage` settings. Review the guidance information and default value for each setting to determine whether or not to modify it.

```
#
cray_rur.settings.rur_stage.data.stage_timeout: 90
#
```

```
#
cray_rur.settings.rur_stage.data.stage_dir: /var/spool/RUR
#
```

5. Uncomment the lines corresponding to the `rur_gather` settings. Review the guidance information and default value for each setting to determine whether or not to modify it.

```
#
cray_rur.settings.rur_gather.data.gather_timeout: 90
#
```

```
#
cray_rur.settings.rur_gather.data.gather_dir: /tmp/rur
#
```

6. Uncomment the lines corresponding to the `rur_post` settings. Review the guidance information and default value for each setting to determine whether or not to modify it.

```
#
cray_rur.settings.rur_post.data.post_timeout: 90
#
```

```
#
cray_rur.settings.rur_post.data.post_dir: /tmp/rur
#
```

7. (Optional) Enable the `gpustat` data plugin.

   The `gpustat` plugin collects utilization statistics for NVIDIA GPUs, if present (see *The gpustat Data Plugin* on page 227).

   a. Uncomment `cray_rur.settings.gpustat.data.enable` and set it to `true`.

   ```
   #
   cray_rur.settings.gpustat.data.enable: true
   #
   ```

   b. Uncomment the remaining `gpustat` settings.

   ```
   #
   cray_rur.settings.gpustat.data.stage: /opt/cray/rur/default/bin/gpustat_stage.py
   #
   ```

   ```
   #
   cray_rur.settings.gpustat.data.post: /opt/cray/rur/default/bin/gpustat_post.py
   #
   ```

8. (Optional) Enable the `taskstats` data plugin.

   The `taskstats` plugin collects process accounting data (see *The taskstats Data Plugin* on page 229).

   a. Uncomment `cray_rur.settings.taskstats.data.enable` and set it to `true`.

   ```
   #
   cray_rur.settings.taskstats.data.enable: true
   #
   ```

b. Uncomment the remaining `taskstats` settings.

```
#
cray_rur.settings.taskstats.data.stage: /opt/cray/rur/default/bin/taskstats_stage.py
#
```

```
#
cray_rur.settings.taskstats.data.post: /opt/cray/rur/default/bin/taskstats_post.py
#
```

```
#
cray_rur.settings.taskstats.data.arg: json-dict
#
```

c. Review the guidance information for `cray_rur.settings.taskstats.data.arg` and modify its value if desired.

TIP: The amount of data reported by the `taskstats` plugin and the format in which it is written is determined by the value of `arg`. Examples are included in *The taskstats Data Plugin* on page 229.

9. (Optional) Enable the `energy` data plugin.

The `energy` plugin collects compute node energy usage data (see *The energy Data Plugin* on page 225).

a. Uncomment `cray_rur.settings.energy.data.enable` and set it to `true`.

```
#
cray_rur.settings.energy.data.enable: true
#
```

b. Uncomment the remaining `energy` settings.

```
#
cray_rur.settings.energy.data.stage: /opt/cray/rur/default/bin/energy_stage.py
#
```

```
#
cray_rur.settings.energy.data.post: /opt/cray/rur/default/bin/energy_post.py
#
```

```
#
cray_rur.settings.energy.data.arg: json-dict
#
```

c. Review the guidance information for `cray_rur.settings.energy.data.arg` and modify its value if desired.

TIP: The amount of data reported by the `energy` plugin and the format in which it is written is determined by the value of `arg`. Examples are included in *The energy Data Plugin* on page 225.

10. (Optional) Enable the `timestamp` data plugin.

The `timestamp` plugin collects the start and end times of an application or job (see *The timestamp Data Plugin* on page 232).

a. Uncomment `cray_rur.settings.timestamp.data.enable` and set it to `true`.

```
#
cray_rur.settings.timestamp.data.enable: true
#
```

b. Uncomment the remaining `timestamp` settings.

```
#
cray_rur.settings.timestamp.data.stage: /opt/cray/rur/default/bin/timestamp_stage.py
#
```

```
#
cray_rur.settings.timestamp.data.post: /opt/cray/rur/default/bin/timestamp_post.py
#
```

11. (Optional) Enable the `memory` data plugin.

The `memory` plugin collects information from `/proc` and `/sys` that is useful when assessing the memory performance of an application or job (see *The memory Data Plugin* on page 227).

a. Uncomment `cray_rur.settings.memory.data.enable` and set it to `true`.

```
#
cray_rur.settings.memory.data.enable: true
#
```

b. Uncomment the remaining `memory` settings.

```
#
cray_rur.settings.memory.data.stage: /opt/cray/rur/default/bin/memory_stage.py
#
```

```
#
cray_rur.settings.memory.data.post: /opt/cray/rur/default/bin/memory_post.py
#
```

```
#
cray_rur.settings.memory.data.arg: json-dict
#
```

c. Review the guidance information for `cray_rur.settings.memory.data.arg` and modify if desired.

TIP: The amount of data reported by the `memory` plugin is determined by the value of `arg`. Examples are included in *The memory Data Plugin* on page 227.

12. (Optional) Enable the `nodeuse` data plugin.

The `nodeuse` plugin collects compute node usage data within the scope of an application (see *The nodeuse Data Plugin* on page 229).

a. Uncomment `cray_rur.settings.nodeuse.data.enable` and set it to `true`.

```
#
cray_rur.settings.nodeuse.data.enable: true
#
```

b. Uncomment the remaining `nodeuse` settings.

```
#
cray_rur.settings.nodeuse.data.stage: /opt/cray/rur/default/bin/nodeuse_stage.py
#
```

```
#
cray_rur.settings.nodeuse.data.post: /opt/cray/rur/default/bin/nodeuse_post.py
#
```

13. (Optional) Enable the `dws` data plugin.

The `dws` plugin collects DataWarp utilization statistics (within the scope of an application) from compute nodes, if present (see *The dws Data Plugin* on page 219).

a. Uncomment `cray_rur.settings.dws.data.enable` and set it to `true`.

```
#
cray_rur.settings.dws.data.enable: true
#
```

b. Uncomment the remaining `dws` settings.

```
#
cray_rur.settings.dws.data.stage: /opt/cray/rur/default/bin/dws_stage.py
#
```

```
#
cray_rur.settings.dws.data.post: /opt/cray/rur/default/bin/dws_post.py
#
```

14. (Optional) Enable the `dws_server` data plugin.

The `dws_server` plugin collects utilization statistics (within the scope of an application) from DataWarp servers, if present (see *The dws_server Data Plugin* on page 222).

a. Uncomment `cray_rur.settings.dws_server.data.enable` and set it to `true`.

```
#
cray_rur.settings.dws_server.data.enable: true
#
```

b. Uncomment the remaining `dws_server` settings.

```
#
cray_rur.settings.dws_server.data.stage: /opt/cray/rur/default/bin/
```

```
dws_server_stage.py
#
```

```
#
cray_rur.settings.dws_server.data.post: /opt/cray/rur/default/bin/dws_server_post.py
#
```

15. (Optional) Enable the `dws_job_server` data plugin.

The `dws_job_server` plugin collects utilization statistics (within the scope of a job) from DataWarp servers, if present (*The dws_job_server Data Plugin* on page 219).

a. Uncomment `cray_rur.settings.dws_job_server.data.enable` and set it to `true`.

```
#
cray_rur.settings.dws_job_server.data.enable: true
#
```

b. Uncomment the remaining `dws_job_server` settings.

Note that the post script is the same as `dws_server`.

```
#
cray_rur.settings.dws_job_server.data.stage: /opt/cray/rur/default/bin/dws_job_server_stage.py
#
```

```
#
cray_rur.settings.dws_server.data.post: /opt/cray/rur/default/bin/dws_server_post.py
#
```

16. (Optional) Enable the `llm` output plugin.

The `llm` plugin aggregates log messages from various Cray nodes and places them on the SMW (*The llm Output Plugin* on page 232).

a. Uncomment `cray_llm.settings.llm.data.enable` and set it to `true`.

```
#
cray_rur.settings.llm.data.enable: true
#
```

b. Uncomment the other `llm` setting.

```
#
cray_rur.settings.llm.data.output: /opt/cray/rur/default/bin/llm_output.py
#
```

17. (Optional) Enable the `user` output plugin.

The `user` plugin writes RUR output for a user's application to the user's home directory (default) or a user-defined location, only if the user has indicated that this behavior is desired (*The user Output Plugin* on page 232).

a. Uncomment `cray_rur.settings.user.data.enable` and set it to `true`.

```
#
cray_rur.settings.user.data.enable: true
#
```

b. Uncomment the remaining `user` settings.

```
#
cray_rur.settings.user.data.output: /opt/cray/rur/default/bin/user_output.py
#
```

```
#
cray_rur.settings.user.data.arg: single, opt_in
#
```

c. Review the guidance information for `cray_rur.settings.user.data.arg` and modify if desired.

> **TIP:** The number of output files created by the `user` plugin and its opt-in flag are determined by the value of `arg`. Further details are included in *The user Output Plugin* on page 232.

Next, configure the `cray_alps` to call the RUR prologue and epilogue scripts. Sites running Slurm must modify the Slurm configuration file to call the RUR prologue and epilogue scripts.

Configure the cray_alps Service for Per-application RUR

Prerequisites

This procedure assumes that `cray_alps` has already been enabled.

About this task

Although Resource Utilization Reporting (RUR) is not a part of ALPS, it is initiated by the ALPS prologue and epilogue scripts. This enables per-application RUR.

During CLE installation, `cray_alps` might have been configured for RUR. If this is known to be true, then this procedure may be skipped; however, Cray recommends that sites verify the settings are accurate.

Procedure

1. Edit `cray_alps_worksheet.yaml`.
2. Verify that `cray_alps.enabled` is uncommented and set to true.

   This should have occurred during the initial CLE installation. If not, exit this procedure and refer to XC™ Series Software Installation and Configuration Guide.
3. Uncomment and define `cray_alps.settings.apsys.data.prologPath` and `cray_alps.settings.apsys.data.epilogPath`.

   ALPS supports only one prologue script and one epilogue script; therefore, enabling RUR is dependent on whether or not these parameters are already defined for ALPS.

a. If `prologPath` and `epilogPath` are not set, define them as follows.

```
cray_alps.settings.apsys.data.prologPath: /opt/cray/rur/default/bin/rur_prologue.py
cray_alps.settings.apsys.data.epilogPath: /opt/cray/rur/default/bin/rur_epilogue.py
```

b. For either parameter that is defined, a wrapper script must be written that will run both the ALPS script and the RUR script. Cray recommends adjusting the `prologTimeout` and `epilogTimeout` parameters to be the sum of the timeouts expected for the constituent scripts. Because RUR supports its own timeout, it is recommended to run RUR first, with a timeout, allowing the second plugin to run even if RUR times out.

4. Uncomment `cray_alps.settings.apsys.data.prologTimeout` and `cray_alps.settings.apsys.data.epilogTimeout`, review the guidance information and modify if desired.

```
cray_alps.settings.apsys.data.prologTimeout: 300
...
cray_alps.settings.apsys.data.epilogTimeout: 300
```

Next, configure the workload manager to enable per-job RUR.

Configure a WLM to Enable Per-job RUR

Prerequisites

Task prerequisites.

About this task

Although Resource Utilization Reporting (RUR) is not a part of a workload manager (WLM), it is initiated through the WLM prologue and epilogue scripts to enable per-job RUR. Job level RUR data is identified by records with `apid: 0` and `jobid: ID_of_WLM_job`.

Procedure

1. Edit the WLM prologue and epilogue scripts, according to the specific WLM system guidelines, to call the `rur_prologue` and `rur_epilogue` scripts, respectively.

   Prologue:

   ```
   /opt/cray/rur/default/bin/rur_prologue.py -C /etc/opt/cray/rur/rur2.conf -a 0
   -j $JOBID -A jobtoken=$JOBID -A jobfile=$JOBFILE -N /tmp/
   ```

   Epilogue:

   ```
   /opt/cray/rur/default/bin/rur_epilogue.py -C /etc/opt/cray/rur/rur2.conf -a 0
   -j $JOBID -A jobtoken=$JOBID -A jobfile=$JOBFILE -N /tmp/
   ```

   Where:

   `-C /etc/opt/cray/rur/rur2.conf`

   Is required to fully implement job scope RUR

   `$JOBFILE`

User's job script file that may or may not contain DataWarp directives (`#DW`)

`$JOBID`

Job ID selected by the WLM. It is available to the WLM's prologue and epilogue scripts, but implementation may vary between the various WLMs.

`-N nidfile`

Points to a file containing a list of the DataWarp node IDs (one node per line). Currently RUR only addresses the DataWarp nodes within this list.

TIP: To run other job scope RUR plugins, it is necessary to add a second call to `rur_prologue` and `rur_epilogue` scripts (with a different configuration file than used above) within the WLM prologue and epilogue scripts, respectively.

2. Consult the specific WLM documentation to restart the WLM.

Next, refresh nodes to apply configuration changes.

Refresh Nodes with Updated Config Sets

Prerequisites

This procedure assumes that configuration data has been changed, either by updating the config set using the configurator (interactive or worksheet upload) or by editing a configuration template (`cray_SERVICE_config.yaml`) directly.

About this task

Whenever a cray service (e.g., `cray_persistent_data`) is modified, it is necessary to update and validate the config set and run `cray-ansible` on any affected CLE nodes in order to apply the configuration changes. If the system will be rebooted, the steps to run `cray-ansible` are not needed because `cray-ansible` is run automatically when the system boots.

Using `cfgset update` ensures that all pre- and post-configuration scripts get run. Running `cray-ansible` on a CLE node triggers a refresh of the CLE config set cache on that node and applies configuration changes on the node.

Procedure

1. Update the config set, if not already done.

```
smw# cfgset update p0
```

2. Validate the config set.

```
smw# cfgset validate p0
```

3. Run Ansible plays on the CLE nodes, if the system will not be rebooted.

   After the CLE config set has been updated, refresh the local config set cache to pull any config set changes to the node and run `cray-ansible` to apply them on the node.

```
hostname# /opt/cray/imps-distribution/default/bin/refresh.py
hostname# /etc/init.d/cray-ansible start
```

Enable/Disable Plugins

About this task

RUR (`cray_rur`) configuration changes are done within the Cray configuration management framework. Changes are made **either** during an interactive configurator session or by modifying the `cray_rur` worksheet. The worksheet method is described in the procedure to initially enable and configure RUR. This procedure invokes an interactive configurator session, which would likely be the method used when only enabling or disabling plugins.

Procedure

1. Invoke an interactive configurator session.

   This example shows that `gpustat` is enabled, `taskstat` is disabled, and `dws` is not defined, which renders it disabled.

```
smw# cfgset update -m interactive -s cray_rur -l advanced p0
Service Configuration Menu (Config Set: p0, type: cle)

  cray_rur        [ status: enabled ]  [ validation: valid ]

------------------------------------------------------------------
  Selected     #      Settings     Value/Status (level=basic)
------------------------------------------------------------------
...
                     gpustat
                10)     enable    True
                11)     stage     /opt/cray/rur/default/bin/gpustat_stage.py
                12)     post      default=/opt/cray/rur/default/bin/
                                  gpustat_post.py

                      taskstats
                13)     enable    False
                14)     stage     /opt/cray/rur/default/bin/taskstats_stage.py
                15)     post      /opt/cray/rur/default/bin/taskstats_post.py
                16)     arg       json-dict
...
                     dws
                31)     enable    [ unconfigured, default=False ]
                32)     stage     [ unconfigured, default=/opt/cray/rur/default
                                  /bin/dws_stage.py ]
                33)     post      [ unconfigured, default=/opt/cray/rur/default
                                  /bin/dws_post.py ]
...
```

2. Disable a plugin:

   a. Select the number corresponding to its `enable` setting.

```
RUR service Menu [default: save & exit - Q] $ 10
```

   The setting is highlighted:

```
                   gpustat
   *     10)       enable              True
```

   b. Set `enable` to `false`.

```
RUR service Menu [default: configure - C] $ C
...
ray_rur.settings.gpustat.data.enable
[<cr>=keep 'true', <new value>, ?=help, @=less] $ false
```

The `enable` status changes.

```
                gpustat
        10)       enable              False
```

3. To enable a configured plugin:

   a. Select the number corresponding to its `enable` setting.

   ```
   RUR service Menu [default: save & exit - Q] $ 13
   ```

   The setting is highlighted:

   ```
                taskstat
   *    13)       enable              False
   ```

   b. Set `enable` to `true`.

   ```
   RUR service Menu [default: configure - C] $ C
   ...
   ray_rur.settings.taskstat.data.enable
   [<cr>=keep 'false', <new value>, ?=help, @=less] $ true
   ```

   The `enable` status changes.

   ```
                taskstat
   *    13)       enable               True
   ```

4. To enable an unconfigured plugin:

   a. Select the number corresponding to its `enable` setting.

   ```
   RUR service Menu [default: save & exit - Q] $ 31
   ```

   The setting is highlighted:

   ```
                dws
   *    31)       enable              [ unconfigured, default=False ]
   ```

   b. Set `enable` to `true`.

   ```
   RUR service Menu [default: configure - C] $ C
   ...
   ray_rur.settings.dws.data.enable
   [<cr>=keep 'false', <new value>, ?=help, @=less] $ true
   ```

   The `enable` status changes.

   ```
                dws
   *    31)       enable               True
   ```

   c. Configure **all** of the plugin's other settings.

```
RUR service Menu [default: save & exit - Q] $ 32
RUR service Menu [default: configure - C] $ C
cray_rur.settings.dws.data.stage
[<cr>=set '/opt/cray/rur/default/bin/dws_stage.py', <new value>, ?=help,
@=less] $ <cr>
RUR service Menu [default: save & exit - Q] $ 33
RUR service Menu [default: configure - C] $ C
cray_rur.settings.dws.data.post
[<cr>=set '/opt/cray/rur/default/bin/dws_post.py', <new value>, ?=help,
@=less] $ <cr>
```

5. Save and exit the configurator.

```
RUR service Menu [default: save & exit - Q] $ Q
```

To apply these configuration changes, refresh the appropriate nodes with the updated config set.

The dws Data Plugin

The `dws` plugin collects the following DataWarp utilization statistics from compute nodes, if present. This usage data is available within the scope of an application, not the scope of a job. The data is written in JSON dictionary format. Additional DataWarp usage data is available through the `dws_server` and `dws_job_server` plugins.

bytes_read	Number of bytes read
bytes_written	Number of bytes written
files_created	Number of files created
inodes_created	Number of inodes created, including files, directories and links
max_read_offset	Maximum byte offset read
max_write_offset	Maximum byte offset written

RUR `dws` output

This example shows `dws` data as written to `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW:

```
2017-02-03T13:27:32.662733-05:00 c0-1c0s8n0 RUR 16521
p0-20161006t064726 [RUR@34] uid: 12345, apid: 1140449, jobid:
1127.sdb, cmdname: /bin/hostname, plugin:dws {"token": "1127.sdb",
"mountpoint": "/var/opt/cray/dws/mounts/batch/1127.sdb/ss",
"inodes_created": 407, "files_created": 405, "bytes_read":
11207405004, "bytes_written": 6712208222, "max_read_offset":
4096024126, "max_write_offset": 21772241122}
```

The dws_job_server Data Plugin

The `dws_job_server` plugin collects utilization statistics from DataWarp servers, if present. This usage information is within the scope of a job, not the scope of an application. The data is written in JSON dictionary format. DataWarp server usage information within the scope of an application is available through the `dws_server` plugin. Compute node usage of DataWarp is available through the `dws` plugin.

Type `scratch` File Systems

The following data is collected for `type=scratch` file systems.

Per realm:

`dwtype`	Type of DataWarp file system (scratch)
`namespace_count`	Number of namespaces within the realm
`realm_id`	Realm ID
`server_count`	Number of servers in the realm
`server_hostname`	Server hostname

Per fragment:

`bytes_read`	Number of bytes read from this fragment
`bytes_written`	Number of bytes written to this fragment
`capacity_used`	Amount of file system capacity used
`capacity_max`	Maximum capacity of fragment
`files_created`	Number of files created in this realm
`fs_capacity`	Capacity of file system to which this fragment belongs
`max_window_write`	Maximum amount of data written during a write window period of time
`write_high_water`	The largest amount of data written
`write_limit`	Maximum bytes allowed to be written per fragment
`write_moving_avg`	The average amount of data written during a write window period of time

Per namespace:

`namespace_id`	Namespace ID
`bytes_read`	Number of bytes read from this namespace
`bytes_written`	Number of bytes written to this namespace
`files_create_threshold`	Maximum number of files allowed to be created within this namespace
`file_size_limit`	Maximum size (bytes) of one file
`files_created`	Number of files created within this namespace
`max_offset_read`	Maximum byte offset read
`max_offset_written`	Maximum byte offset written
`num_data_created`	Total number of data files created on all DataWarp servers
`stage_bytes_read`	Number of staged bytes read
`stage_bytes_written`	Number of staged bytes written
`stripe_size`	Size of each stripe (bytes)
`stripe_width`	Number of stripes in this namespace
`substripe_size`	Size of each substripe (bytes)

	substripe_width	Number of substripes in per stripe

RUR dws_server / dws_job_server output for type=scratch file systems

This example shows data as written to `/var/opt/cray/log/partition-current/messages-date` on the SMW:

```
uid: 12345, apid: 416746, jobid: 21268, cmdname: xio_p,
plugin:dws_server {{"realm": {"server_count": 1, "fragments": [{
{"capacity_used": 128648781824, "fs_capacity": 3296920076288,
"capacity_max": 128648781824, "max_window_write": 86400,
"files_created": 258, "write_high_water": 3407329284614,
"write_moving_avg": 3407329284614, "bytes_read": 3298534883328,
"write_limit": 32985348833280, "bytes_written": 3407329284614,
"server_hostname": "c0-0c1s1n1"], "namespaces": [{ "namespace_id":
9324, "stripe_width": 1, "stripe_size": 8388608, "bytes_read":
3298534883328, "substripe_width": 12, "stage_bytes_read": 0,
"substripe_size": 8388608, "max_offset_read": 1099511627776,
"files_created": 258, "bytes_written": 3407329284614,
"files_create_threshold": 0, "file_size_limit": 0,
"num_data_created": 258, "stage_bytes_written": 0,
"max_offset_written": 1099511627776 }], "realm_id": 3704}}
```

Type cache File Systems

The following data is collected for `type=cache` file systems.

Per realm:	**dwtype**	Type of DataWarp file system (cache)
	pfs_path	Backing path
	realm_id	Realm ID
	server_count	Number of servers in the realm
Per fragment:	**capacity_highwater**	Maximum number of bytes used in underlying file system
	fs_capacity	Capacity of file system to which this fragment belongs
	max_offset_read	Maximum byte offset read
	max_offset_threshold	Maximum byte offset allowed to be read/written
	max_offset_written	Maximum byte offset written
	pfs_read	Number of bytes read from the PFS
	pfs_written	Number of bytes written to the PFS
	window_write_bytes	Number of bytes written during a write window period of time
	window_write_seconds	Number of seconds in a write window period of time
	cache_read	Number of DataWarp storage bytes read
	cache_written	Number of DataWarp storage bytes written

RUR `dws_server` / `dws_job_server` output for `type=cache` file systems

This example shows data as written to `/var/opt/cray/log/partition-current/messages-date` on the SMW:

```
uid: 12345, apid: 416742, jobid: 21266, cmdname: fdfa1,
{plugin:dws_server "realm": {"fragments": [{ "server_hostname":
"c0-0c1s1n2", "window_write_bytes": 82165273, "fs_capacity": 0,
"capacity_highwater": 0, "window_write_seconds": 21600000,
"cache_written": 82165273, "max_offset_read": 183609,
"max_offset_written": 8388608, "pfs_read": 82165273,
"max_offset_threshold": 0, "pfs_written": 0, "cache_read":
6741137 }, { "server_hostname": "c0-0c1s1n1", "window_write_bytes":
101616737, "fs_capacity": 3856795508736, "capacity_highwater":
2657973817344, "window_write_seconds": 21600000, "cache_written":
101616737, "max_offset_read": 95786, "max_offset_written": 8388608,
"pfs_read": 101616737, "max_offset_threshold": 0, "pfs_written": 0,
"cache_read": 6586671 }], "server_count": 2, "realm_id": 1902,
"pfs_path": "/lus/peel" }}}
```

The dws_server Data Plugin

The `dws_server` plugin collects utilization statistics from DataWarp servers, if present. This usage information is within the scope of an application, not the scope of a job. The data is written in JSON dictionary format. DataWarp server usage information within the scope of a job is available through the `dws_job_server` plugin. Compute node usage of DataWarp is available through the `dws` plugin.

Type `scratch` File Systems

The following data is collected for `type=scratch` file systems.

Per realm:	**`dwtype`**	Type of DataWarp file system (scratch)
	`namespace_count`	Number of namespaces within the realm
	`realm_id`	Realm ID
	`server_count`	Number of servers in the realm
	`server_hostname`	Server hostname

Per fragment:	**`bytes_read`**	Number of bytes read from this fragment
	`bytes_written`	Number of bytes written to this fragment
	`capacity_used`	Amount of file system capacity used
	`capacity_max`	Maximum capacity of fragment
	`files_created`	Number of files created in this realm
	`fs_capacity`	Capacity of file system to which this fragment belongs
	`max_window_write`	Maximum amount of data written during a write window period of time
	`write_high_water`	The largest amount of data written

	write_limit	Maximum bytes allowed to be written per fragment
	write_moving_avg	The average amount of data written during a write window period of time
Per namespace:	**namespace_id**	Namespace ID
	bytes_read	Number of bytes read from this namespace
	bytes_written	Number of bytes written to this namespace
	files_create_threshold	Maximum number of files allowed to be created within this namespace
	file_size_limit	Maximum size (bytes) of one file
	files_created	Number of files created within this namespace
	max_offset_read	Maximum byte offset read
	max_offset_written	Maximum byte offset written
	num_data_created	Total number of data files created on all DataWarp servers
	stage_bytes_read	Number of staged bytes read
	stage_bytes_written	Number of staged bytes written
	stripe_size	Size of each stripe (bytes)
	stripe_width	Number of stripes in this namespace
	substripe_size	Size of each substripe (bytes)
	substripe_width	Number of substripes in per stripe

RUR `dws_server` / `dws_job_server` output for `type=scratch` file systems

This example shows data as written to `/var/opt/cray/log/partition-current/messages-date` on the SMW:

```
uid: 12345, apid: 416746, jobid: 21268, cmdname: xio_p,
plugin:dws_server {{"realm": {"server_count": 1, "fragments": [{
{"capacity_used": 128648781824, "fs_capacity": 3296920076288,
"capacity_max": 128648781824, "max_window_write": 86400,
"files_created": 258, "write_high_water": 3407329284614,
"write_moving_avg": 3407329284614, "bytes_read": 3298534883328,
"write_limit": 32985348833280, "bytes_written": 3407329284614,
"server_hostname": "c0-0c1s1n1"], "namespaces": [{ "namespace_id":
9324, "stripe_width": 1, "stripe_size": 8388608, "bytes_read":
3298534883328, "substripe_width": 12, "stage_bytes_read": 0,
"substripe_size": 8388608, "max_offset_read": 1099511627776,
"files_created": 258, "bytes_written": 3407329284614,
"files_create_threshold": 0, "file_size_limit": 0,
"num_data_created": 258, "stage_bytes_written": 0,
"max_offset_written": 1099511627776 }], "realm_id": 3704}}
```

Type `cache` File Systems

The following data is collected for `type=cache` file systems.

Per realm:	**`dwtype`**	Type of DataWarp file system (cache)
	`pfs_path`	Backing path
	`realm_id`	Realm ID
	`server_count`	Number of servers in the realm
Per fragment:	**`capacity_highwater`**	Maximum number of bytes used in underlying file system
	`fs_capacity`	Capacity of file system to which this fragment belongs
	`max_offset_read`	Maximum byte offset read
	`max_offset_threshold`	Maximum byte offset allowed to be read/written
	`max_offset_written`	Maximum byte offset written
	`pfs_read`	Number of bytes read from the PFS
	`pfs_written`	Number of bytes written to the PFS
	`window_write_bytes`	Number of bytes written during a write window period of time
	`window_write_seconds`	Number of seconds in a write window period of time
	`cache_read`	Number of DataWarp storage bytes read
	`cache_written`	Number of DataWarp storage bytes written

RUR `dws_server` / `dws_job_server` output for `type=cache` file systems

This example shows data as written to `/var/opt/cray/log/partition-current/messages-date` on the SMW:

```
uid: 12345, apid: 416742, jobid: 21266, cmdname: fdfa1,
{plugin:dws_server "realm": {"fragments": [{ "server_hostname":
"c0-0c1s1n2", "window_write_bytes": 82165273, "fs_capacity": 0,
"capacity_highwater": 0, "window_write_seconds": 21600000,
"cache_written": 82165273, "max_offset_read": 183609,
"max_offset_written": 8388608, "pfs_read": 82165273,
"max_offset_threshold": 0, "pfs_written": 0, "cache_read":
6741137 }, { "server_hostname": "c0-0c1s1n1", "window_write_bytes":
101616737, "fs_capacity": 3856795508736, "capacity_highwater":
2657973817344, "window_write_seconds": 21600000, "cache_written":
101616737, "max_offset_read": 95786, "max_offset_written": 8388608,
"pfs_read": 101616737, "max_offset_threshold": 0, "pfs_written": 0,
"cache_read": 6586671 }], "server_count": 2, "realm_id": 1902,
"pfs_path": "/lus/peel" }}}
```

The energy Data Plugin

The `energy` plugin collects compute node energy usage data. The amount of data reported and the format in which it is written is determined by the value of `arg` set for the `energy` plugin within the `cray_rur` service settings.

If `arg` is not set or set to `json-dict` (default), the plugin reports the following extended energy data, written in JSON dictionary format:

cpu_energy_used	The total energy (joules) used by each node's CPU energy domain. This statistic is zero for non-KNL nodes.
error	If a Python exception occurs during the post or staging scripts, the following data is reported: **traceback** Stack frame list **type** Python exception type **value** Python exception parameter **nid** NID on which exception occurred **cname** cname on which exception occurred
memory_energy_used	The total energy (joules) used by each node's memory energy domain. This statistic is zero for non-KNL nodes.
nodes	Number of nodes in job
nodes_cpu_throttled	Number of nodes experiencing CPU power/thermal throttling
nodes_memory_throttled	Number of nodes experiencing memory power/thermal throttling
nodes_power_capped	Number of nodes with nonzero power cap
nodes_throttled	Number of nodes experiencing any of the following types of throttling: • CPU power/thermal throttling • Memory power/thermal throttling
nodes_with_changed_power_cap	Number of nodes with power caps that changed during execution. On nodes with accelerators, this value includes the number of accelerators with power caps that changed.
max_power_cap	Maximum nonzero power cap
energy_used	The total energy (joules) used across all nodes. On nodes with accelerators, this value includes `accel_energy_used`, the total energy used by the accelerators.
max_power_cap_count	Number of nodes with the maximum nonzero power cap
min_power_cap	Minimum nonzero power cap
min_power_cap_count	Number of nodes with the minimum nonzero power cap

On nodes with accelerators, the extended data also include the following data:

accel_energy_used	Total accelerator energy (joules) used
nodes_accel_power_capped	Number of accelerators with nonzero power cap
max_accel_power_cap	Maximum nonzero accelerator power cap

max_accel_power_cap_count	Number of accelerators with the maximum nonzero power cap
min_accel_power_cap	Minimum nonzero accelerator power cap
min_accel_power_cap_count	Number of accelerators with the minimum nonzero power cap

If `arg` contains the `verbose` option, a log per node is generated in addition to the standard summary log. The verbose logs include the following data:

cname	The cname of the node
nid	The NID of the node
energy_used	The total energy (joules) on the node. On nodes with an accelerator, this value includes `accel_energy_used`. On a KNL node, this value includes `cpu_energy_used` and `cpu_memory_used`.
cpu_energy_used	The total energy (joules) used in the node's CPU energy domain. This statistic is zero on non-KNL nodes.
memory_energy_used	The total energy (joules) used in the node's memory energy domain. This statistic will be zero on non-KNL nodes
cpu_throttled	Non-zero if the node experienced CPU power/thermal throttling
memory_throttled	Non-zero if the node experienced memory power/thermal throttling
start_power_cap	Power cap at start of execution, if set
stop_power_cap	Power cap at end of execution, if set
accel_energy_used	Total accelerator energy (joules) used
start_accel_power_cap	Accelerator power cap at start of execution, if set
stop_accel_power_cap	Accelerator power cap at end of execution, if set
changed_power_cap	A power cap changed (includes changed accelerator power cap)

RUR extended `energy` output

This example shows extended `energy` data as written to `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW:

```
2017-02-03T15:44:23.583598-05:00 c0-0c0s7n1 RUR 6048
p1-20160906t093257 [RUR@34] uid: 12345, apid: 18554, jobid: 0,
cmdname: /bin/cat, plugin: energy {"nodes_throttled": 0,
"memory_energy_used": 120,"min_accel_power_cap_count": 0,
"nodes_with_changed_power_cap": 0,"max_power_cap_count": 0,
"energy_used": 1214, "max_power_cap": 0,"nodes_memory_throttled": 0,
"accel_energy_used": 0,"max_accel_power_cap_count": 0,
"nodes_accel_power_capped": 0,"min_power_cap": 0,
"max_accel_power_cap": 0, "min_power_cap_count":
0,"min_accel_power_cap": 0, "nodes_power_capped": 0, "nodes": 4,
"cpu_energy_used": 752, "nodes_cpu_throttled": 0}
```

If `arg` is set to `json-list` (deprecated), the plugin reports the following, written in JavaScript Object Notation (JSON) list format:

energy_used	The total energy (joules) used across all nodes. On nodes with accelerators, this value includes `accel_energy_used`, the total energy used by the accelerators. On KNL nodes, this value

includes `cpu_energy_used` and `cpu_memory_used`, the total energy used by the CPU and memory energy domains.

RUR `energy` output using `json-list` (deprecated)

This example shows default `energy` data as written to `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW:

```
2017-01-30T11:19:06.545114-05:00 c0-0c0s2n2 RUR 18657
p2-20130829t090349 [RUR@34] uid: 12345, apid: 10963, jobid: 0,
cmdname: /opt/intel/vtune_xe_2013/bin64/amplxe-cl plugin: energy
['energy_used', 318]
```

The gpustat Data Plugin

The `gpustat` plugin collects the following utilization statistics for NVIDIA GPUs, if present. The data is written in JSON list format.

`maxmem`	Maximum memory used across all nodes
`summem`	Total memory used across all nodes
`gpusecs`	Time spent processing on GPUs

RUR `gpustat` output

This example shows `gpustat` data as written in `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW.

```
2017-02-03T15:50:42.761257-05:00 c0-0c0s2n2 RUR 11329
p2-20130709t145714 [RUR@34] uid: 12345, apid: 8410, jobid: 0,
cmdname: /tmp/dostuff plugin: gpustats ['maxmem', 108000, 'summem',
108000, 'gpusecs', 44]
```

The memory Data Plugin

The `memory` plugin collects information from `/proc` and `/sys` that is useful when assessing the memory performance of an application or job. The data is written in JSON dictionary format. The type of data reported is determined by the value of `arg` set for the `memory` within the `cray_rur` service settings.

IMPORTANT: The `memory` plugin does not provide consolidated information for all nodes within an application; instead it reports memory statistics for each node within the application. This can result in a large amount of RUR output data for systems of even modest size. When the `memory` plugin is enabled, it produces a significant amount of output.

If `arg` is not set (default), the plugin reports the following data:

`%_of_boot_mem`	The % of boot memory for each order chunk in `/proc/buddyinfo` summed across all memory zones
`Active(anon)`	Total amount of memory in active use by the application
`Active(file)`	Total amount of memory in active use by cache and buffers
`boot_freemem`	Contents of `/proc/boot_freemem`

current_freemem	Contents of `/proc/current_freemem`
free	Number of hugepages that are not yet allocated
hugepages-*size*kB	The hugepage size for the select entries from `/sys/kernel/mm/hugepages/hugepages-*kB/*`
Inactive(anon)	Total amount of memory that is candidate to be swapped out
Inactive(file)	Total amount of memory that is candidate to be dropped from cache
nr	Number of hugepages that exist at this point
resv	Number of hugepages committed for allocation, but no allocation has occurred
Slab	Total amount of memory used by the kernel
surplus	Number of hugepages above `nr`

RUR default `memory` output

This example shows the default `memory` data as written to `/var/opt/cray/log/`*partition*`-current/messages-`*date* on the SMW.

```
2017-02-03T11:37:24.480982-05:00 c0-0c0s0n2 RUR 23710
p0-20140321t091957 [RUR@34] uid: 12345, apid: 33079, jobid: 0,
cmdname: /bin/hostname, plugin: memory {"current_freemem": 21858372,
"meminfo": {"Active(anon)": 35952, "Slab": 105824, "Inactive(anon)":
1104}, "hugepages-2048kB": {"nr": 5120, "surplus": 5120},
"%_of_boot_mem": ["67.23", "67.23", "67.23", "67.22", "67.21",
"67.18", "67.11", "67.04", "66.94", "66.83", "66.77", "66.66",
"66.53", "66.38", "65.87", "65.07", "63.05", "61.43"], "nid": "8",
"cname": "c0-0c0s2n0", "boot_freemem": 32432628}
```

If `arg` is set to `extended_buddy`, the output relating to `/proc/buddyinfo` includes NUMA node granularity information in addition to the existing node granularity information. This information is useful when troubleshooting certain fragmentation related issues.

RUR extended `memory` output

This example shows extended `memory` data as written to `/var/opt/cray/log/`*partition*`-current/messages-`*date* on the SMW:

```
2017-02-03T11:37:24.480982-05:00 c0-0c0s0n2 RUR 23710
p0-20140321t091957 [RUR@34] uid: 12345, apid: 33079, jobid: 0,
cmdname: /bin/hostname, plugin: memory {"current_freemem": 21858372,
"meminfo": {"Active(anon)": 35952, "Slab": 105824, "Inactive(anon)":
1104}, "hugepages-2048kB": {"nr": 5120, "surplus": 5120},
"Node_0_zone_DMA": ["0.05", "0.05", "0.05", "0.05", "0.05", "0.05",
"0.05", "0.05", "0.05", "0.04", "0.04", "0.03", "0.00", "0.00",
"0.00", "0.00", "0.00", "0.00"],"%_of_boot_mem": ["67.23", "67.23",
"67.23", "67.22", "67.21", "67.18", "67.11", "67.04", "66.94",
"66.83", "66.77", "66.66", "66.53", "66.38", "65.87", "65.07",
"63.05", "61.43"], "nid": "8", "cname": "c0-0c0s2n0", "boot_freemem":
32432628, "Node_0_zone_DMA32": ["6.07", "6.07", "6.07", "6.07",
"6.07", "6.07", "6.07", "6.06", "6.05", "6.04", "6.01", "5.94",
"5.86", "5.76", "5.46", "4.85", "3.23", "3.23"], "Node_0_zone_Normal":
["61.11", "61.11", "61.11", "61.11", "61.09", "61.07", "60.99",
```

```
"60.93", "60.84", "60.75", "60.72", "60.70", "60.67", "60.62",
"60.42", "60.22", "59.81", "58.20"]}
```

The nodeuse Data Plugin

The `nodeuse` plugin collects the following compute node usage data within the scope of an application. The data is written in JSON dictionary format.

nodes	Number of nodes reserved
nidlist	NIDs of the reserved nodes

RUR `nodeuse` output

This example shows `nodeuse` data as written to `/var/opt/cray/log/`*partition*`-current/messages-`*date* on the SMW:

```
2017-01-08T15:44:23.583598-05:00 uid: 12345, apid: 35489, jobid: 0,
cmdname: /usr/bin/df, plugin: nodeuse {"nodes": 6, "nidlist":
"36-38, 40-41, 43"}
```

The taskstats Data Plugin

The `taskstats` plugin collects process accounting data. The amount of data reported and the format in which it is written is determined by the value of `arg` set for the `taskstats` plugin within the `cray_rur` service settings.

If `arg` is not set or set to `json-dict` (default), the plugin reports the following basic process accounting data similar to that provided by UNIX process accounting or `getrusage`. This data is written in JSON dictionary format. If `arg` is set to `json-list` (deprecated), the data is written in JSON list format. These values are sums across all nodes, except for the memory used, which is the maximum value across all nodes.

core	Set to `1` if core dump occurred
exitcode	Lists all unique exit codes
max_rss	Maximum memory used
rchar	Characters read by process
stime	System time
utime	User time
wchar	Characters written by process

RUR `taskstats` output

This example shows `taskstats` output as written to `/var/opt/cray/log/`*partition*`-current/messages-`*date* on the SMW.

For a job that exits normally:

```
2017-02-02T11:09:49.457770-05:00 c0-0c1s1n2 RUR 2417
p0-20161101t153028 [RUR@34] uid: 12345, apid: 86989, jobid: 0,
cmdname: /lus/tmp/rur01.2338/./CPU01-2338 plugin: taskstats {"utime":
```

```
10000000, "stime": 0, "max_rss": 940, "rchar": 107480, "wchar": 90,
"exitcode:signal": ["0:0"], "core": 0}
```

For a job that core dumps:

```
2017-02-02T11:12:45.020716-05:00 c0-0c1s1n2 RUR 3731
p0-20131101t153028 [RUR@34] uid: 12345, apid: 86996, jobid: 0,
cmdname: /lus/tmp/rur01.3657/./exit04-3657 plugin: taskstats {"utime":
4000, "stime": 144000, "max_rss": 7336, "rchar": 252289, "wchar": 741,
"exitcode:signal": ["0:9", "139:0", "0:11", "0:0"], "core": 1}
```

If `arg` is set to `xpacct`, the plugin also provides the following extended process accounting data similar to that which was collected by the deprecated Cray System Accounting (CSA).

abortinfo	If abnormal termination occurs, a list of `abort_info` fields is reported
apid	Application ID as defined by application launcher
bkiowait	Total delay time (ns) waiting for synchronous block I/O to complete
btime	UNIX time when process started
comm	String containing process name. May be different than the header, which is the process run by the launcher.
coremem	Integral of RSS used by process in MB-usec
ecode	Process exit code
etime	Total elapsed time in microseconds
gid	Group ID
jid	Job ID - the PAGG job container used on the compute node
majfault	Number of major page faults
minfault	Number of minor page faults
nice	POSIX `nice value` of process
nid	String containing node ID
pgswapcnt	Number of pages swapped; should be `0` on Cray compute nodes
pid	Process ID
pjid	Parent job ID - the PAGG job container on the MOM node
ppid	Parent process ID
prid	Job project ID
rcalls	Number of read system calls
rchar	Characters read by process
rss	RSS highwater mark

[1] The current memory usage is added to these counters (i.e., `coremem`, `vm`) every time. A tick is charged to a task's system time. Therefore, at the end we will have memory usage multiplied by system time and an average usage per system time unit can be calculated.

sched	Scheduling discipline used on node
uid	User ID
vm	Integral of virtual memory used by process in MB-usecs[2]
wcalls	Number of write system calls
wchar	Characters written by process

RUR extended taskstats output

This example shows RUR extended `taskstats` output:

```
2017-02-03T10:29:38.285378-05:00 c0-0c0s1n1 RUR 24393
p1-20131018t081133 [RUR@34] uid: 12345, apid: 370583, jobid: 0,
cmdname: /bin/cat, plugin: taskstats {"btime": 1386061749, "etime":
8000, "utime": 0, "stime": 4000, "coremem": 442, "max_rss": 564,
"max_vm": 564, "pgswapcnt": 63, "minfault": 15, "majfault": 48,
"rchar": 2608, "wchar": 686, "rcalls": 19, "wcalls": 7, "bkiowait":
1000, "exitcode:signal": [0], "core": 0]
```

If `arg` is set to `xpacct, per-process`, the plugin reports extended accounting data for every compute node process rather than a summary of all processes for an application. `per-process` must be set in combination with `xpacct`.

CAUTION: If `per-process` is set and many processes are run on each node, the volume of data generated and stored on disk can become an issue.

RUR per-process taskstats output

This example shows RUR per-process `taskstats` output.

```
2017-02-03T13:25:34.446167-06:00 c0-0c2s0n2 RUR 7623
p3-20131202t090205 [RUR@34] uid: 12345, apid: 1560, jobid: 0,
cmdname: ./it.sh, plugin: taskstats {"uid": 12345, "wcalls": 37,
"pid": 2997, "vm": 16348, "jid": 395136991233, "bkiowait": 1201616,
"majfault": 1, "etime": 0, "btime": 1386098731, "gid": 0, "ppid":
2992, "utime": 0, "nice": 0, "sched": 0, "nid": "92", "prid": 0,
"comm": "mount", "stime": 4000, "wchar": 3465, "rss": 1028,
"minfault": 352, "coremem": 1109, "ecode": 0, "rcalls": 22, "pjid":
7045, "pgswapcnt": 0, "rchar": 12208}

2017-02-03T13:25:34.949138-06:00 c0-0c2s0n2 RUR 7623
p3-20131202t090205 [RUR@34] uid: 12345, apid: 1560, jobid: 0,
cmdname: ./it.sh, plugin: taskstats {"uid": 12345, "wcalls": 0, "pid":
2998, "vm": 20268, "jid": 395136991233, "bkiowait": 0, "majfault": 0,
"etime": 0, "btime": 1386098731, "gid": 0, "ppid": 2992, "utime": 0,
"nice": 0, "sched": 0, "nid": "92", "prid": 0, "apid": 1560, "comm":
"ls", "stime": 4000, "wchar": 0, "rss": 1040, "minfault": 360,
```

[2] The current memory usage is added to these counters (i.e., `coremem`, `vm`) every time. A tick is charged to a task's system time. Therefore, at the end we will have memory usage multiplied by system time and an average usage per system time unit can be calculated.

```
"coremem": 3140, "ecode": 0, "rcalls": 19, "pjid": 7045, "pgswapcnt":
0, "rchar": 10629}
```

The timestamp Data Plugin

The `timestamp` plugin collects the start and end times of an application or job.

RUR `timestamp` output

This example shows `timestamp` data, as written in `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW, for an application that slept 20 seconds:

```
2017-01-30T14:32:07.593469-05:00 c0-0c0s5n2 RUR 12882
p3-20130830t074847 [RUR@34] uid: 12345, apid: 6640, jobid: 0,
cmdname: /bin/sleep plugin: timestamp APP_START 2013-08-30T14:31:46CDT
APP_STOP 2013-08-30T14:32:06CDT
```

The file Output Plugin

The `file` plugin allows RUR data to be stored to a flat text file on any file system to which the login node can write. This plugin is also intended as a very simple guide for anyone interested in writing an output plugin.

This example shows sample output from `file` to a location defined in the RUR configuration file:

```
uid: 1000, apid: 8410, jobid: 0, cmdname: /tmp/dostuff plugin:
taskstats ['utime', 32000, 'stime', 132000, 'max_rss', 1736, 'rchar',
44524, 'wchar', 289] uid: 1000, apid: 8410, jobid: 0, cmdname: /tmp/
dostuff plugin: energy ['energy_used', 24551] uid: 1000, apid: 8410,
jobid: 0, cmdname: /tmp/dostuff plugin: gpustats ['maxmem', 108000,
'summem', 108000]
```

The llm Output Plugin

The `llm` plugin aggregates log messages from various Cray nodes and places them on the SMW. `llm` has its own configuration options, but typically it will place RUR messages into the messages log file `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* on the SMW. The messages shown in the previous sections are in LLM log format.

The user Output Plugin

The `user` plugin writes RUR output for a user's application to the user's home directory (default) or a user-defined location, only if the user has indicated that this behavior is desired (as described below).

The naming of the default output file(s), `rur.`*`suffix`*, is dependent on the value of the argument `arg`, which defines a report type and is set in the `user` section of the RUR configuration file. If `arg` is set to:

`apid` An output file is created for each application executed and *`suffix`* is the `apid`.

`jobid` An output file is created for each job submitted and *`suffix`* is the `jobid`

`single` All output is placed in a single file and no suffix is appended to the output file name.

User Options

Users have the option to opt-in or out for the `user` plugin, redirect plugin output to a specific file or directory, or override the default report type.

- By default, RUR data is written to a user's directory. A user must either create the file `~/.rur/user_output_optin` to indicate that data should be written, or create a file that initiates one of the following two options.

  1. Users may redirect the output of RUR by specifying a redirect location in `~/.rur/user_output_redirect`. The contents of this file must be a single line that specifies the absolute or relative (from the user's home directory) path of the directory or file to which the RUR output data is to be written. If the redirect file either does not exist, points to a path that does not exist, or points to a path to which the user does not have write permission, then the output is written to the user's home directory.
  2. A user with an existing `~/.rur/user_output_redirect` file can temporarily stop RUR data from being written by setting the redirect path to `/dev/null`.

- Additionally, the user may override the default report type by specifying a valid report type in `~/.rur/user_output_report_type`. Valid report types are `apid`, `jobid`, or `single`, resulting in the user's RUR data being written to one file per application, one file per job, or a single file, respectively. If the file `~/.rur/user_output_report_type` is empty or contains an invalid type, then the default report type, as defined in the configuration file, is created.

The database Example Output Plugin

The `database` plugin is provided as a guide for sites wanting to output RUR data to a site-supplied database. Sites will need to configure their own systems, provide an external database, create their own tables, and modify `database_output.py` to collect the desired data.

MySQL is the database supported by the example plugin. The following arguments are defined for connecting to a database:

- `DB_NAME='rur'`
- `DB_USER='rur_user'`
- `DB_PASS='rur_pass'`
- `DB_HOST='rur_host'`

  The `database` plugin collects the values: `energy_used`, `apid`, `jobid`, and `uid`, and saves this data to a table, `energy`. It does this by performing the following:

- Digests RUR data into a dictionary and saves it to class `DbData`
- Creates rules for saving data collected in `DbData` to particular tables
- Uses the rules to scan the `DbData` dictionary and `INSERT` that data into a database

  Cray recommends that the database is not hosted on SDB or login nodes. It should also be noted that, depending on job load, interacting with an external database may cause system latency.

Create Custom RUR Data Plugins

A data plugin is comprised of a *staging component* and a *post processing component*. The data plugin staging component is called by `rur-stage.py` on the compute node prior to the application/job running and again after the application/job has completed. The staging component may reset counters before application/job execution

and collect them after application/job completion, or it may collect initial and final values prior to and after application/job execution, respectively, and then calculate the delta values. Python functions have been defined to simplify writing plugins, although it is not necessary for the plugin to be written in Python. The interface for the data plugin staging component is through command line arguments.

Data Plugin Staging Component

All data plugin staging components must support the following arguments:

--apid=*apid*	Defines the application ID of the running application.
--timeout=*time*	Defines a timeout period in seconds during which the plugin must finish running. Set to `0` for unlimited; default is unlimited.
--pre	Indicates the plugin is being called prior to the application/job.
--post	Indicates the plugin is being called after the application/job.
--outputfile=*output_file*	Defines where the output data is written. Each plugin should define a default output file in `/var/spool/RUR/` if this argument is not provided.
--arg=*arg*	A plugin-specific argument, set in the RUR config file. RUR treats this as an opaque string.

The output of an RUR data plugin staging component is a temporary file located in `/var/spool/RUR` on the compute node. The file name must include both the name of the plugin, as defined in the RUR config file, and *`.apid`*. The RUR gather phase will automatically gather the staged files from all compute nodes after the application/job has completed and place it in `gather_dir` as defined in the configuration file.

Data plugin staging component

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# Sample data plugin staging component
#
#!/usr/bin/env python
import sys, os, getopt
from rur_plugins import rur_plugin_args
def main():
   apid, inputfile, outputfile, timeout, pre, post, \
                parg = rur_plugin_args(sys.argv[1:])
   if outputfile is "":
       outputfile = "/var/spool/RUR/pluginname."+str(apid)
   if (pre==1):
       zero_counters()
   else:
       write_postapp_stateto(outputfile)

if __name__ == "__main__":
   main()
```

Data Plugin Post Processing Component

A data plugin also requires a post processing component that processes the data staged by the staging component and collected during the RUR gather phase. The post processing component is called by

`rur-post.py`. The input file contains records, one node per line, of all of the statistics created by the staging component. The output of the post processing component is a file containing the summary of data from all compute nodes.

All data plugin post processing components must support the following arguments:

**`--apid=`*`apid`*	Defines the application ID of the running application.
**`--timeout=`*`time`*	Defines a timeout period in seconds during which the plugin must finish running. Set to `0` for unlimited; default is unlimited.
**`--inputfile=`*`input_file`*	Specifies the file from which the plugin gets its input data.
**`--outputfile=`*`output_file`*	Specifies the file to which the plugin writes its output data.

Data plugin post processing component

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# Sample data plugin post processing component
#
#!/usr/bin/env python
import sys, os
from rur_plugins import rur_args

def main():
   apid, inputfile, outputfile, timeout = rur_args(sys.argv[1:])
   if outputfile is "":
       outputfile = inputfile + ".out"

   pc = PostCompute()
   pc.process_file(inputfile)
   formated = pc.present_entries([('plugin_foo_data','sum')])
   fout=open(outputfile, 'w+')
   fout.write("energy %s" % formated)

if __name__ == "__main__":
   main()
```

Create Custom RUR Output Plugins

Output plugins allow RUR data to be outputted to an arbitrary backing store. This can be a storage device or another piece of software that then consumes the RUR data. The output plugin is passed a number of command line arguments that describe the application/job run and provide a list of input working files (the output of data plugin post processing components). The plugin takes the data in the working files and exports it to the destination specified in the RUR configuration file for the specific output plugin.

Data passed to custom output plugins can be optionally configured to be JSON-formatted by adding the `use_json` argument to the `[global] section of the configuration file and setting it to` `True`, `yes`, `1`, or `enable`.

> **TIP:** If there is an error from an output plugin, the error message appears in the ALPS log `/var/opt/cray/alps/log/apsys` on the service node rather than the LLM logs on the SMW.

Output Plugin

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# Sample output plugin
#
#!/usr/bin/env python
import sys, os
from rur_plugins import rur_output_args

def main():
   apid, jobid, uid, cmdname, inputfilelist, timeout, \
   parg = rur_output_args(sys.argv[1:])

   outfile = open(parg, "a")
   for inputfile in inputfilelist:
       infile =  open(inputfile, "r")
       lines = infile.readlines()
       for line in lines:
           outfile.write(line)
       infile.close()
   outfile.close()
```

Implement a Site-Written RUR Plugin

About this task

For a site written plugin to run, it must be added to the `cray_rur` service settings and enabled.

Procedure

1. Ensure that the site written plugin is located on a file system that is readable by compute nodes, owned by `root`, and not writeable by non-`root` users.

2. Invoke an interactive configurator session.

```
smw# cfgset update -m interactive -s -l interactive cray_rur p0
Service Configuration Menu (Config Set: p0, type: cle)

  cray_rur        [ status: enabled ]  [ validation: valid ]

------------------------------------------------------------------
  Selected     #      Settings        Value/Status (level=basic)
------------------------------------------------------------------
...
             42)      data_plugins    [ 5 sub-settings unconfigured, select
                                      and enter C to add entries ]
             43)      output_plugins  [ 4 sub-settings unconfigured, select
                                      and enter C to add entries ]
```

3. Add a site-written data plugin.

   a. Select the number corresponding to the `data_plugins` setting.

```
RUR service Menu [default: save & exit - Q] $ 42
```

The setting is highlighted:

```
   *     42)       data_plugins               True
```

b. Add a data plugin name.

```
RUR service Menu [default: configure - C] $ C
...
cray_rur.settings.data_plugins
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $ +
cray_rur.settings.data_plugins.data.plugin_name
[<cr>=set '', <new value>, ?=help, @=less] $ sitedataplug
```

c. Add the complete path to the data plugin's staging script.

```
cray_rur.settings.data_plugins.data.sitedataplug.stage
[<cr>=set 'none', <new value>, ?=help, @=less] $ /opt/cray/rur/default/bin/
sitedataplug_stage.py
```

d. Add the complete path to the data plugin's post script.

```
cray_rur.settings.data_plugins.data.sitedataplug.post
[<cr>=set 'none', <new value>, ?=help, @=less] $ /opt/cray/rur/default/bin/
sitedataplug_post.py
```

e. (Optional) Add a data plugin argument `arg`.

```
cray_rur.settings.data_plugins.data.sitedataplug.arg
[<cr>=set 'none', <new value>, ?=help, @=less] $ <cr>
```

f. Enable the data plugin.

```
cray_rur.settings.data_plugins.data.sitedataplug.enable
[<cr>=set 'true', <new value>, ?=help, @=less] $ <cr>
```

The configured values are displayed:

```
        1) 'sitedataplug'
            a) stage: /opt/cray/rur/default/bin/sitedataplug_stage.py
            b) post: /opt/cray/rur/default/bin/sitedataplug_post.py
            c) arg: none
            d) enable: True
```

g. Set the completed data plugin entry.

```
cray_rur.settings.data_plugins
[<cr>=set 1 entries, +=add an entry, ?=help, @=less] $ <cr>
```

The `data_plugins` setting is updated.

```
               42)     data_plugins
                          plugin_name: sitedataplug     [ OK ]
```

4. Add a site-written output plugin.

a. Select the number corresponding to the `output_plugins` setting.

```
RUR service Menu [default: save & exit - Q] $ 43
```

The setting is highlighted:

```
   *     43)       output_plugins              True
```

b. Add an output plugin name.

```
RUR service Menu [default: configure - C] $ C
...
cray_rur.settings.output_plugins
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $ +
cray_rur.settings.output_plugins.data.plugin_name
[<cr>=set '', <new value>, ?=help, @=less] $ siteoutplug
```

c. Add the path to the output plugin script or binary.

```
cray_rur.settings.output_plugins.data.siteoutplug.output
[<cr>=set 'none', <new value>, ?=help, @=less] $ /opt/cray/rur/site/bin/
siteoutplug_output.py
```

d. (Optional) Add an output plugin argument `arg`.

```
cray_rur.settings.output_plugins.data.siteoutplug.arg
[<cr>=set 'none', <new value>, ?=help, @=less] $ <cr>
```

e. Enable the output plugin.

```
cray_rur.settings.output_plugins.data.siteoutplug.enable
[<cr>=set 'true', <new value>, ?=help, @=less] $ <cr>
```

The configured values are displayed:

```
        1) 'siteoutplug'
            a) output: /opt/cray/rur/site/bin/siteoutplug_output.py
            b) arg: none
            c) enable: True
```

f. Set the completed output plugin entry.

```
cray_rur.settings.output_plugins
[<cr>=set 1 entries, +=add an entry, ?=help, @=less] $ <cr>
```

The `output_plugins` setting is updated.

```
               42)     output_plugins
                          plugin_name: siteoutplug     [ OK ]
```

5. Save and exit the configurator.

```
RUR service Menu [default: save & exit - Q] $ Q
```

To apply these configuration changes, refresh the appropriate nodes with the updated config set.

Additional Plugin Examples

This is a set of RUR plugins that report information about the number of available huge pages on each node. The huge page counts are reported in `/proc/buddyinfo`. There are two versions of the staging component: one that reports what is available and the second that reports changes during the application run.

Huge pages data plugin staging component (version A)

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# This is an RUR plugin that reports information about the number of
available
# huge pages on each node.  This is reported in /proc/buddyinfo.
#
# Each node reports its nid and the number of available pages of
each size.
#
#!/usr/bin/env python
import sys, os, getopt
from rur_plugins import rur_plugin_args
def main():
    apid, inputfile, outputfile, timeout, pre, post, parg
=rur_plugin_args(sys.argv[1:])
    if outputfile == 0:
         outputfile = "/var/spool/RUR/buddyinfo."+str(apid)
    if (pre==1):
         zero_counters()
    else:
         nidf = open("/proc/cray_xt/nid", "r")
         n = nidf.readlines()
         nid = int(n[0])
         inf = open("/proc/buddyinfo", "r")
         b = inf.readlines()
         sizes = dict( [( '2M' , 0 ), ('4M', 0), ('8M', 0), ('16M',
0), ('32M', 0), ('64M', 0)])

         for line in b:
              l = line.split()
              sizes['2M'] += int(l[13])
              sizes['4M'] += int(l[14])
              sizes['8M'] += int(l[15])
              sizes['16M'] += int(l[16])
              sizes['32M'] += int(l[17])
              sizes['64M'] += int(l[18])

        o = open(outputfile, "w")
        o.write("{6} {0} {1} {2} {3} {4}
{5}".format(sizes['2M'],sizes['4M'], \
        sizes['8M'], sizes['16M'], sizes['32M'], sizes['64M'], nid))
        o.close()

if __name__ == "__main__":
    main()
```

Huge pages data plugin staging component (version B)

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# This is an RUR plugin that reports information about the number of
available
# huge pages on each node.  This is reported in /proc/buddyinfo.
#
```

```
# This plugin records the number of available pages before the job
is launched.
# At job completion time it reports the change
#
#!/usr/bin/env python
import sys, os, getopt
from rur_plugins import rur_plugin_args
def main():
    apid, inputfile, outputfile, timeout, pre, post, parg
=rur_plugin_args(sys.argv[1:])
    if outputfile == 0:
            outputfile = "/var/spool/RUR/buddyinfo."+str(apid)
    if (pre==1):
            inf = open("/proc/buddyinfo", "r")
            b = inf.readlines()
            sizes = dict( [( '2M' , 0 ), ('4M', 0), ('8M', 0),
('16M', 0), ('32M', 0), ('64M', 0)])
            for line in b:
                    l = line.split()
                    sizes['2M'] += int(l[13])
                    sizes['4M'] += int(l[14])
                    sizes['8M'] += int(l[15])
                    sizes['16M'] += int(l[16])
                    sizes['32M'] += int(l[17])
                    sizes['64M'] += int(l[18])

        o = open("/tmp/buddyinfo_save", "w")
        o.write("{0} {1} {2} {3} {4}
{5}".format(sizes['2M'],sizes['4M'], \
                sizes['8M'], sizes['16M'], sizes['32M'],
sizes['64M']))
        o.close()
    else:
            nidf = open("/proc/cray_xt/nid", "r")
            n = nidf.readlines()
            nid = int(n[0])
            inf = open("/proc/buddyinfo", "r")
            b = inf.readlines()
            sizes = dict( [( '2M' , 0 ), ('4M', 0), ('8M', 0),
('16M', 0), ('32M', 0), ('64M', 0)])

            for line in b:
                    l = line.split()
                    sizes['2M'] += int(l[13])
                    sizes['4M'] += int(l[14])
                    sizes['8M'] += int(l[15])
                    sizes['16M'] += int(l[16])
                    sizes['32M'] += int(l[17])
                    sizes['64M'] += int(l[18])

            obf = open("/tmp/buddyinfo_save", "r")
            ob = obf.readlines()
            n=0

            obd0 = ob[0]
            obd = obd0.split()

            diff = [
                    (int(obd[0]) - sizes['2M']),
                    (int(obd[1]) - sizes['4M']),
                    (int(obd[2]) - sizes['8M']),
```

```
                    (int(obd[3]) - sizes['16M']),
                    (int(obd[4]) - sizes['32M']),
                    (int(obd[5]) - sizes['64M'])
                    ]

        o = open(outputfile, "w")
    # uncomment the following line to get the actual sizes
        #o.write("sizes {6} {0} {1} {2} {3} {4}
{5}\n".format(sizes['2M'],sizes['4M'], \
                sizes['8M'], sizes['16M'], sizes['32M'],
sizes['64M'], nid))
        o.write("diff {6} {0} {1} {2} {3} {4} {5}".format(diff[0],
diff[1], diff[2], \
                diff[3], diff[4], diff[5], nid))
        o.close()
    os.unlink("/tmp/buddyinfo_save")

if __name__ == "__main__":
    main()
```

Huge pages data plugin post processing component

```
#
# Copyright (c) 2013 Cray Inc. All rights reserved.
#
# This is a RUR postprocessing pluging for the buddyinfo data
# collection. It copies the input files to output, adding a
# "buddyinfo" label.
#
#!/usr/bin/env python
import sys, os
from rur_plugins import rur_args

def main():
   apid, inputfile, outputfile, timeout = rur_args(sys.argv[1:])
   if outputfile == 0:
       outputfile = inputfile + ".out"

   fin=open(inputfile, "r")
   l = fin.readlines()

   fout=open(outputfile, 'w+')
   for line in l:
            fout.write("buddyinfo {0}".format(line))

if __name__ == "__main__":
   main()
```

Application Completion Reporting (ACR) to RUR Migration Tips

Cray supplied RUR data plugins collect the same data found in Mazama's Application Completion Reporting (ACR) feature (deprecated), but RUR does not include a reporting utility like `mzreport`. When using RUR's `llm` output plugin, the type of data reported by `mzreport` can be extracted from the output files as demonstrated in the following sections.

ACR Job Reporting

The information provided by `mzreport -j` and `mzreport --job` can easily be obtained in the RUR environment from the log files `/var/opt/cray/log/`*`partition`*`-current/messages-`*`date`* by invoking the following command:

```
smw:~ # grep -e "RUR" messages-* |grep -e "jobid: jobid"
```

ACR Timespan Reporting

In ACR, `mzreport -t` and `mzreport -T` control the span of time over which job completions are reported. The following example is a simple Python script, `timesearch.py`, that provides this functionality.

```
#cat timesearch.py
#!/usr/bin/env python
for rurline in [line for line in open(sys.argv[1], 'r') if 'RUR' in line]:
   if (rurline.split(' ')[1] > sys.argv[2]) and (rurline.split(' ')[1] <
sys.argv[3]):
       print rurline
```

The script is called with the log file of interest and the desired start/stop time stamps, where *`start_time`* and *`end_time`* are formatted as "*`yyyy`*-*`mm`*-*`dd`*T*`hh`*:*`mm`*:*`ss`*", as follows:

```
smw:~ # python ./timesearch.py messages-date "start_time" "end_time"
```

ACR Exit Code Reporting

The `get_exit.py` Python script listed here provides a list of the user IDs with the most non-zero exit codes.

```
# cat get_exit.py
#!/usr/bin/env python
import os,sys,re

statre = re.compile("'(\w*):(\w*)',\s*\[('(\w*):(\w*)'(, )?)+\]")
statsre = re.compile("(\w*):(\w*)")
uidre  = re.compile("uid:\s*(\w*)")
cnt = {}

for rurline in [line for line in open(sys.argv[1], 'r') if 'RUR' in line]:
   if 'taskstats' in rurline:
       sus = statre.search(rurline)
       status = sus.group()
       stats = statsre.findall(status)
       for stat in stats[1:]:
           if stat[0] != '0':
               uid = int(uidre.findall(rurline)[0])
               if cnt.get(str(uid)):
                   cnt[str(uid)] += 1
               else:
                   cnt[str(uid)] = 1

x = sorted(cnt, key = cnt.get, reverse=True)
print "uids with the most non-zero exit codes %s" % x[:sys.argv[2]]
```

The script is called with the log file of interest and the number of user IDs on which to report, as follows:

```
smw:~ # python ./get_exit.py messages-date num
```

Application Resource Utilization (ARU) to RUR Migration Tips

Sites that use ARU (deprecated) will have an easy transition to RUR as all of the data provided in ARU is available in RUR, but in a slightly different format.

This example shows that the following ARU output is available by enabling the `taskstats` plugin's default behavior:

ARU output:

```
2012-11-26T08:52:37.802113-06:00 c0-0c0s0n2 apsys 19864
p0-20121126t060549 -
apid=6240364, Finishing, user=8855, batch_id=114.sdb, exit_code=0,
exitcode_array=0,
exitsignal_array=0, utime=0 stime=0 maxrss=3168 inblocks=0 outblocks=0
cpus=8
start=Mon Nov 26 08:52:37 2012 stop=Wed Dec 31 18:00:00 1969
cmd=growfiles
```

RUR `taskstats` default output:

```
2013-11-02T11:09:49.457770-05:00 c0-0c1s1n2 RUR 2417
p0-20131101t153028 [RUR@34]
uid: 10973, apid: 86989, jobid: 0, cmdname: /lus/esfs/overby/
rur01.2338/./CPU01-2338
plugin: taskstats ['utime', 10000000, 'stime', 0, 'max_rss', 940,
'rchar', 107480,
'wchar', 90, 'exitcode:signal', ['0:0'], 'core', 0]
```

This example shows that the following ARU output is available by enabling the RUR `timestamp` plugin.

ARU output:

```
2012-11-26T08:53:15.618239-06:00 c0-0c0s0n2 apsys 20604
p0-20121126t060549 -
apid=6240378, Finishing,user=8855, batch_id=121.sdb, exit_code=0,
exitcode_array=0,
exitsignal_array=0, utime=0 stime=0 maxrss=3152 inblocks=0 outblocks=0
cpus=1
start=Mon Nov 26 08:52:51 2012 stop=Wed Dec 31 18:00:00 1969
cmd=close2_01
```

RUR `timestamp` plugin output:

```
2013-08-30T14:32:07.593469-05:00 c0-0c0s5n2 RUR 12882
p3-20130830t074847 [RUR@34] uid: 0,
apid: 6640, jobid: 0, cmdname: /bin/sleep plugin: timestamp APP_START
2013-08-30T14:31:46CDT APP_STOP 2013-08-30T14:32:06CDT
```

CSA to RUR Migration Tips

The Cray supplied RUR data plugin `taskstats`, when enabled and configured for extended accounting data, collects all of the data in the CSA process accounting record with the exception of `ac_sbu`, the system billing units.

RUR extended `taskstats` output

This example shows RUR extended `taskstats` output:

```
2017-02-03T10:29:38.285378-05:00 c0-0c0s1n1 RUR 24393
p1-20131018t081133 [RUR@34] uid: 12345, apid: 370583, jobid: 0,
cmdname: /bin/cat, plugin: taskstats {"btime": 1386061749, "etime":
8000, "utime": 0, "stime": 4000, "corememem": 442, "max_rss": 564,
"max_vm": 564, "pgswapcnt": 63, "minfault": 15, "majfault": 48,
"rchar": 2608, "wchar": 686, "rcalls": 19, "wcalls": 7, "bkiowait":
1000, "exitcode:signal": [0], "core": 0]
```

RUR does not include the report generation capabilities provided by CSA, however, the type of data reported by CSA can be extracted from the messages files on the SMW. The following is a short Python script for searching through these files. It allows filtering for group ID (`-g`), job ID (`-j`), user ID (`-u`), and system time exceeding a certain value (`-s`); similar to the `csacom` filters `-g`, `-j`, `-u`, `-O`, respectively.

```
#!/usr/bin/env python
# Usage: filter-messages [-g gid] [-j jid] [-u uid] [-s stime] -f messages-date
import os,sys,re,getopt,collections

def getcmdlineargs(args):
   arglist = collections.defaultdict(lambda: 0, {})
   options, remainder = getopt.getopt(args,
       'g:j:u:s:f:',
       ['gid=', 'jid=', 'uid=', 'Stimeexceeds=', 'filename='])

   for opt,arg in options:
       if opt in ('-g', '--gid'):
           arglist['gid'] = arg
       if opt in ('-j', '--jid'):
           arglist['jid'] = arg
       if opt in ('-u', '--uid'):
           arglist['uid'] = arg
       if opt in ('-s', '--Stimeexceeds'):
           arglist['stimeexceeds'] = arg
       if opt in ('-f', '--filename'):
           arglist['filename'] = arg
   return arglist

def reeqgt(tag, restr, rurline, eq):
   retre = re.compile("'" + str(restr) + "'," + "\s*(\w*)")
   field = retre.findall(rurline)
   if field == []:
       return False
   if eq and tag == field[0]:
       return True
   elif (not eq) and tag <= field[0]:
       return True
   return False

arglist = getcmdlineargs(sys.argv[1:])
if not arglist['filename']:
   exit(1)
for rurline in [line for line in open(arglist['filename'], 'r') if 'RUR' in
line]:
  if 'taskstats' in rurline:
       if arglist['jid'] and not (reeqgt(arglist['jid'], 'jid', rurline, 1)):
```

```
                continue
        if arglist['uid'] and not (reeqgt(arglist['uid'], 'uid', rurline, 1)):
                continue
        if arglist['gid'] and not (reeqgt(arglist['gid'], 'gid', rurline, 1)):
                continue
        if arglist['stimeexceeds'] and not (reeqgt(arglist['stimeexceeds'],
'stime',
           rurline, 0)):
                continue

        print "%s" % rurline,
```

Modify an Installed System

Configure a Boot Failover Node

Prerequisites

The system must be shut down before invoking the `xtcli halt` command, which is used in this procedure.

About this task

When a secondary (backup) boot node is configured, boot-node failover occurs automatically when the primary node fails. If boot node failover was configured during the CLE software installation or upgrade, this procedure is not needed.

A boot node must have a Fibre Channel or SAS connection to the boot RAID. If boot node failover is enabled, then each boot node should have such a connection to the boot RAID. Also, each boot node must have an Ethernet connection to the network shared with the SMW in order to PXE boot and transfer data as a tier1 node.

Procedure

1. Configure `cray_multipath` for the failover boot node, if `cray_multipath` is enabled.

   `cray_multipath` is in the global config set and may be inherited by the CLE config set. If the global `cray_multipath` is enabled and the CLE `cray_multipath` is set to inherit from the global config set, then make the changes in the global `cray_multipath` service. If the CLE `cray_multipath` service is enabled and not set to inherit from the global config set, then make the changes in the CLE `cray_multipath` service.

   Enter the list of multipath nodes.

   Change `cray_multipath.settings.multipath.data.node_list`, so that it includes both the primary (active) boot node and the secondary (passive) failover boot node.

   This example shows a list of three nodes: an SMW with host ID 1eac4e0c, a primary boot node with cname c0-0c0s4n1, a secondary boot node with cname c0-2c0s4n1, and an SDB node with cname c0-0c0s3n1.

   ```
   cray_multipath.settings.multipath.data.node_list:
   - 1eac4e0c
   - c0-0c0s4n1
   - c0-2c0s4n1
   - c0-0c0s3n1
   ```

2. Configure `cray_node_groups` to add a failover boot node.

   In the CLE config set, the `cray_node_groups` service should have these settings configured with the proper cnames for all boot and SDB nodes. The `boot_nodes` node group should list as members the primary boot node (c0-0c0s4n1) and the secondary boot node (c0-2c0s3n1).

```
cray_node_groups.settings.groups.data.group_name.boot_nodes: null
cray_node_groups.settings.groups.data.boot_nodes.description: Default node
    group which contains the primary and failover (if applicable) boot
    nodes associated with the current partition.
cray_node_groups.settings.groups.data.boot_nodes.members:
- c0-0c0s4n1
- c0-2c0s3n1
```

3. Configure `cray_persistent_data` to add the `boot_nodes` node group.

   Ensure that this setting includes the `boot_nodes` node group and the `sdb_nodes` node group.

```
cray_persistent_data.settings.mounts.data./var/lib/nfs.client_groups:
- boot_nodes
- sdb_nodes
```

4. Configure `cray_scalable_services` to add `boot_nodes` node group.

   Ensure that this setting includes the `boot_nodes` node group and the `sdb_nodes` node group.

```
cray_scalable_services.settings.scalable_service.data.tier1_groups:
- boot_nodes
- sdb_nodes
```

5. Configure `cray_net` to add secondary boot node.

   These settings define the secondary boot node (backup_bootnode) when using boot node failover. Configure a host as the second boot node for boot node failover. If using the boot node failover feature, then define a backup boot node host with the `standby_node` variable set to `true`.

   > **NOTE:** The host name for the primary and backup boot node should both be set to `boot`. The aliases can be different so that the `/etc/hosts` entry for the cname has the `host` name alias.

```
cray_net.settings.hosts.data.common_name.backup_bootnode: null
cray_net.settings.hosts.data.backup_bootnode.description: backup Boot node for the system
cray_net.settings.hosts.data.backup_bootnode.aliases:
- cray-boot2
cray_net.settings.hosts.data.backup_bootnode.hostid: c0-2c0s3n1
cray_net.settings.hosts.data.backup_bootnode.host_type: admin
cray_net.settings.hosts.data.backup_bootnode.hostname: boot
cray_net.settings.hosts.data.backup_bootnode.standby_node: true

cray_net.settings.hosts.data.backup_bootnode.interfaces.common_name.hsn_boot_alias: null
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.name: ipogif0:1
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.description: Well
known address used for boot node services.
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.vlan_id: ''
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.vlan_etherdevice:
''
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.bonding_slaves: []
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.bonding_module_opt
s: mode=active-backup
    miimon=100
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.aliases: []
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.network: hsn
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.ipv4_address:
10.131.255.254
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.ipv4_secondary_add
resses: []
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.mac: ''
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.startmode: ''
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.bootproto: static
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.mtu: ''
```

```
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.extra_attributes:
[]
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.module: ''
cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.params: ''
#cray_net.settings.hosts.data.backup_bootnode.interfaces.hsn_boot_alias.unmanaged_interfa
ce: false

cray_net.settings.hosts.data.backup_bootnode.interfaces.common_name.primary_ethernet:
null
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.name: eth0
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.description:
Ethernet connecting boot node to the SMW.
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.vlan_id: ''
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.vlan_etherdevice
: ''
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.bonding_slaves:
[]
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.bonding_module_o
pts: mode=active-backup
    miimon=100
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.aliases: []
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.network: admin
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.ipv4_address:
10.3.1.254
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.ipv4_secondary_a
ddresses: []
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.mac: ''
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.startmode: ''
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.bootproto:
static
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.mtu: ''
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.extra_attributes
: []
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.module: ''
cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.params: ''
#cray_net.settings.hosts.data.backup_bootnode.interfaces.primary_ethernet.unmanaged_inter
face: false
```

6. Update the config set to regenerate the CLE `/etc/hosts` file so that it contains the appropriate backup node settings.

```
smw# cfgset update p0
smw# cfgset validate p0
```

7. Halt the primary and backup boot nodes.

```
crayadm@smw> xtcli halt boot_primary_id,boot_backup_id
```

8. Set the primary and backup boot nodes using the `xtcli` command. Use the `-b` argument for a boot node.

```
crayadm@smw> xtcli part_cfg update p0 -b boot_primary_id,boot_backup_id
```

9. Add boot node failover to the boot automation file, `auto.hostname.start`.

   When boot node failover is used, then the boot automation file should have a setting to ensure that STONITH has been enabled on the blade that has the primary boot node. The STONITH setting does not survive a power cycle. To maintain the STONITH setting, add these lines to the boot automation file.

   Use the blade that contains the primary boot node. For example, if the primary boot node is c0-0c0s0n1, then the blade to use is c0-0c0s0. Add these lines before the line for booting the boot node.

```
# Set STONITH for primary boot node
lappend actions {crms_exec "xtdaemonconfig c0-0c0s0 stonith=true"}
```

10. Enable the `xtfailover_halt` command in the `auto.hostname.stop` file.

    Uncomment the second of these lines in `auto.hostname.stop`. This file in `/opt/cray/hss/default/etc` is normally copied from `auto.xtshutdown` to `auto.hostname.stop` during a fresh install. The `xtfailover_halt` command ensures that the `xtbootsys` shutdown process sends a STOP NMI to the failover nodes.

    ```
    # Enable the following line if boot or sdb failover is enabled:
    lappend actions { crms_exec \
    "/opt/cray/hss/default/bin/xtfailover_halt --partition $data(partition,given) --
    shutdown"
    ```

11. Assign the bootimage to the failover boot node.

    Check which NIMS group and boot image are being used for the primary boot node and the secondary boot node.

    ```
    smw# cnode list boot_primary_id
    smw# cnode list boot_backup_id
    ```

    If the secondary boot node does not have the same NIMS group and boot image assigned, update the secondary boot node.

    Remove any old NIMS group from the secondary boot node.

    ```
    smw# cnode update -G oldNIMSgroup boot_backup_id
    ```

    Assign the primary boot nodes NIMS group and boot image to the secondary boot. node.

    ```
    smw# cnode update -g primaryNIMSgroup \
    -i /path/to/primary/bootimage boot_backup_id
    ```

    Confirm the change.

    ```
    smw# cnode list boot_backup_id
    ```

12. Boot the system.

    ```
    crayadm@smw> xtbootsys -a auto.hostname.start
    ```

Disable Boot Node Failover

About this task

For the examples in this procedure, the primary boot node is `c0-0c0s0n1` and the backup boot node is `c2-0c1s7n1`.

Procedure

1. Halt the primary and backup boot nodes.

    ```
    crayadm@smw:~> xtcli halt c0-0c0s0n1,c2-0c1s7n1
    ```

2. Update the default boot configuration.

```
crayadm@smw:~> xtcli boot_cfg update -b c0-0c0s0n1,c0-0c0s0n1
```

3. Update the HSS daemon.

```
crayadm@smw:~> xtdaemonconfig c0-0c0s0 stonith=false
```

Configure an SDB Failover Node

Prerequisites

The system must be shut down before invoking the `xtcli halt` command, which is used in this procedure.

About this task

When a secondary (backup) services database (SDB) node is configured, boot-node failover occurs automatically when the primary node fails. If SDB node failover was configured during the CLE software installation or upgrade, this procedure is not needed.

A service database (SDB) node must have a Fibre Channel or SAS connection to the boot RAID. If SDB node failover is enabled, then each SDB node should have such a connection to the boot RAID. Also, each SDB node must have an Ethernet connection to the network shared with the SMW in order to PXE boot and transfer data as a tier1 node.

Procedure

1. Configure `cray_multipath` for the failover node, if cray_multipath is enabled.

   `cray_multipath` is in the global config set and may be inherited by the CLE config set. If the global `cray_multipath` is enabled and the CLE `cray_multipath` is set to inherit from the global config set, then make the changes in the global `cray_multipath` service. If the CLE `cray_multipath` service is enabled and not set to inherit from the global config set, then make the changes in the CLE `cray_multipath` service.

   Enter the list of multipath nodes.

   Change `cray_multipath.settings.multipath.data.node_list`, so that it includes both the primary (active) SDB node and the secondary (passive) failover SDB node.

   This example shows a list of four nodes: an SMW with host ID 1eac4e0c, a primary boot node with cname c0-0c0s4n1, a secondary boot node with cname c0-2c0s4n1, a primary SDB node with cname c0-0c0s3n1, and a secondary SDB ndoe with cname c0-4c0s3n1.

```
cray_multipath.settings.multipath.data.node_list:
- 1eac4e0c
- c0-0c0s4n1
- c0-2c0s4n1
- c0-0c0s3n1
- c0-4c0s3n1
```

2. Configure `cray_node_groups` to add failover SDB node.

   In the CLE config set, the `cray_node_groups` service should have these settings configured with the proper cnames for all boot and SDB nodes.

The `sdb_nodes` node group should list as members the primary SDB node (c0-2c0s4n1) and the secondary SDB node (c0-4c0s3n1).

```
cray_node_groups.settings.groups.data.group_name.sdb_nodes: null
cray_node_groups.settings.groups.data.sdb_nodes.description: Default node
    group which contains the primary and failover (if applicable) SDB
    nodes associated with the current partition.
cray_node_groups.settings.groups.data.sdb_nodes.members:
- c0-2c0s4n1
- c0-4c0s3n1
```

3. Configure `cray_persistent_data` to add the `sdb_nodes` node group.

   Ensure that this setting includes the `boot_nodes` node group and the `sdb_nodes` node group.

```
cray_persistent_data.settings.mounts.data./var/lib/nfs.client_groups:
- boot_nodes
- sdb_nodes
```

4. Configure `cray_scalable_services` to add the `sdb_nodes` node group.

   Ensure that this setting includes the `boot_nodes` node group and the `sdb_nodes` node group.

```
cray_scalable_services.settings.scalable_service.data.tier1_groups:
- boot_nodes
- sdb_nodes
```

5. Configure `cray_net` to add secondary SDB node.

   These settings define the secondary SDB node (`backup_sdbnode`) when using SDB node failover.Configure a host as the second SDB node for SDB node failover. When using the SDB node failover feature, then define a backup SDB node host with the `standby_node` variable set to `true`.

   > **NOTE:** The host name for the primary and backup SDB node should both be set to `sdb`. The aliases can be different so that the `/etc/hosts` entry for the cname has the host name alias.

   These settings define the secondary SDB node (`backup_sdbnode`) when using SDB node failover.

```
cray_net.settings.hosts.data.common_name.backup_sdbnode: null
cray_net.settings.hosts.data.backup_sdbnode.description: backup SDB node for the system
cray_net.settings.hosts.data.backup_sdbnode.aliases:
- cray-sdb2
cray_net.settings.hosts.data.backup_sdbnode.hostid: c0-4c0s3n1
cray_net.settings.hosts.data.backup_sdbnode.host_type: admin
cray_net.settings.hosts.data.backup_sdbnode.hostname: sdb
cray_net.settings.hosts.data.backup_sdbnode.standby_node: true

cray_net.settings.hosts.data.backup_sdbnode.interfaces.common_name.hsn_boot_alias: null
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.name: ipogif0:1
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.description: Well
known address used for SDB node services.
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.vlan_id: ''
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.vlan_etherdevice:
''
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.bonding_slaves: []
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.bonding_module_opts
: mode=active-backup
    miimon=100
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.aliases: []
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.network: hsn
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.ipv4_address:
10.131.255.253
```

```
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.ipv4_secondary_addr
esses: []
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.mac: ''
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.startmode: auto
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.bootproto: static
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.mtu: ''
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.extra_attributes:
[]
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.module: ''
cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.params: ''
#cray_net.settings.hosts.data.backup_sdbnode.interfaces.hsn_boot_alias.unmanaged_interfac
e: false

cray_net.settings.hosts.data.backup_sdbnode.interfaces.common_name.primary_ethernet: null
cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.name: eth0
cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.description:
Ethernet connecting SDB node to the SMW.
cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.aliases: []
cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.network: admin
cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.ipv4_address:
10.3.1.253
cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.ipv4_secondary_ad
dresses: []
cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.mac: ''
cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.startmode: auto
cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.bootproto: static
cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.mtu: ''
cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.extra_attributes:
 []
cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.module: ''
cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.params: ''
#cray_net.settings.hosts.data.backup_sdbnode.interfaces.primary_ethernet.unmanaged_interf
ace: false
```

6. Update the config set to regenerate the hosts file so that it contains the appropriate backup node settings.

```
smw# cfgset update p0
smw# cfgset validate p0
```

7. Halt the primary and backup SDB nodes using their cnames.

```
crayadm@smw> xtcli halt sdb_primary_id,sdb_backup_id
```

8. Set the primary and backup SDB nodes using the `xtcli` command. Use the `-d` argument for an SDB node.

```
crayadm@smw> xtcli part_cfg update p0 -d sdb_primary_id,sdb_backup_id
```

9. Add SDB node failover to the boot automation file, `auto.hostname.start`.

When SDB node failover is used, then the boot automation file should have a setting to ensure that STONITH has been enabled on the blade that has the primary SDB node. The STONITH setting does not survive a power cycle. To maintain the STONITH setting, add these lines to the boot automation file.

Use the blade that contains the primary SDB node. For example, if the primary SDB node is c0-0c1s0n1, then the blade to use is c0-0c1s0. Add these lines before the line for booting the SDB node.

```
# Set STONITH for primary SDB node
lappend actions {crms_exec "xtdaemonconfig c0-0c1s0 stonith=true"}
```

10. Enable the `xtfailover_halt` command in the `auto.hostname.stop` file.

Uncomment the second of these lines in `auto.hostname.stop`. This file in `/opt/cray/hss/default/etc` is normally copied from `auto.xtshutdown` to `auto.hostname.stop` during a fresh install. The `xtfailover_halt` command ensures that the `xtbootsys` shutdown process sends a STOP NMI to the failover nodes.

```
# Enable the following line if boot or sdb failover is enabled:
lappend actions { crms_exec \
"/opt/cray/hss/default/bin/xtfailover_halt --partition $data(partition,given) --
shutdown" }
```

If the above lines are not present in your `auto.hostname.stop` automation file for shutting down CLE, add them.

11. Assign the bootimage to the failover boot node.

Check which NIMS group and boot image are being used for the primary boot node and the secondary boot node.

```
smw# cnode list boot_backup_id
smw# cnode list boot_backup_id
```

If the secondary boot node does not have the same NIMS group and boot image assigned, update the secondary boot node.

Remove any old NIMS group from the secondary boot node.

```
smw# cnode update -G oldNIMSgroup boot_backup_id
```

Assign the primary boot nodes NIMS group and boot image to the secondary boot node.

```
smw# cnode update -g primaryNIMSgroup \
-i /path/to/primary/bootimage boot_backup_id
```

Confirm the change.

```
smw# cnode list boot_backup_id
```

12. Boot the system.

```
crayadm@smw> xtbootsys -a auto.hostname.start
```

Perform SDB Node Failback

About this task

When a primary SDB node fails and a secondary node takes over, bring the primary node back online without doing a full system boot.

Procedure

Use the xtbootsys command to restore the primary SDB node.

```
smw# xtbootsys --reboot primary_id
```

Perform Boot Node Failback

About this task

When a primary boot node fails and a secondary node takes over, bring the primary node back online without doing a full system boot. An `xtbootsys --reboot` command is prohibited on the boot node. Therefore, enter several commands to reintroduce a failed primary node to the system.

Procedure

1. Shut down the primary boot node.

   ```
   smw# xtcli shutdown primary_id
   ```

2. Bounce the primary boot node.

   ```
   smw# xtbounce -s primary_id
   ```

3. Boot the primary boot node.

   ```
   smw# xtcli boot DEFAULT primary_id
   ```

Configure Realm-Specific IP

About this task

RSIP (realm-specific IP) helps to maintain packet integrity by allowing an RSIP host to borrow one or more IP addresses from a set of configured RSIP gateways. This procedure configures some basic settings in the Cray RSIP configuration service worksheet to add site-specific data.

Procedure

1. Edit `cray_rsip_worksheet.yaml`.

   ```
   smw# vi cray_rsip_worksheet.yaml
   ```

2. Uncomment `cray_rsip.enabled` and set it as follows.

   Will this system use RSIP, that is, does it have any service nodes that will provide the RSIP service?

   - If yes, set it to `true`. Proceed to the next step.
   - If no, set it to `false`. Skip the remaining steps. There is nothing else to configure in this worksheet.

3. (Only for systems with RSIP) Enter the node group (or groups) of the nodes that will be RSIP servers on this system.

   To create one or more node groups that contain the RSIP server nodes (by cname) for this system (rsip_nodes in this example), edit `cray_node_groups_worksheet.yaml`.

Uncomment `cray_rsip.settings.service.data.server_groups`, remove the empty list (`[]`), and add the node group(s) on separate lines prefixed by a hyphen and space (`- `).

```
cray_rsip.settings.service.data.server_groups:
- rsip_nodes
```

4. (Only for systems with RSIP) Enter the node group (or groups) of the service nodes that will be RSIP clients on this system, such as a MOM node.

   To create one or more node groups that contain the RSIP client nodes (by cname) for this system (rsip_servicenode_clients in this example), edit `cray_node_groups_worksheet.yaml`.

   Uncomment `cray_rsip.settings.service.data.node_groups_as_client`, remove the empty list (`[]`), and add the node group(s) on separate lines prefixed by a hyphen and space (`- `).

   ```
   cray_rsip.settings.service.data.node_groups_as_client:
   - rsip_servicenode_clients
   ```

5. (Only for systems with RSIP) Set `cray_rsip.settings.service.data.use_xtrsipcfg` to `false`.

   If the system has a complex RSIP configuration, then set `cray_rsip.settings.service.data.use_xtrsipcfg` to `false` during the initial configuration of complex RSIP. Run `/opt/cray-xt-rsipd/default/bin/xtrsipcfg_v2` to generate the more complex RSIP configuration files to be placed into the config set, and then set `cray_rsip.settings.service.data.use_xtrsipcfg` to `true`. The cray_rsip service guidance describes how to invoke `xtrsipcfg_v2` to make a complex RSIP configuration.

Use the `xtrsipcfg_v2` Script for an Advanced RSIP Configuration

Prerequisites

The configuration in this example requires a dedicated RSIP node.

About this task

The `xtrsipcfg_v2` script generates RSIP client and server configuration files, including configuring RSIP methods RSA-IP and RSAP-IP. This example configuration sets up an RSIP server that utilizes two IP addresses.

Procedure

1. Clone a config set to create a workspace for the new RSIP settings.

   ```
   smw# cfgset create --clone p0 myconfigset-p0
   ```

2. Generate worksheets from configuration service packages installed on the system and config set `myconfigset-p0`.

   ```
   smw# cfgset update --mode prepare myconfigset-p0
   ```

3. Locate the newly generated worksheets and copy them to a new location on the management node.

```
smw# cfgset show --fields path myconfigset-p0
myconfigset-p0:
  path: /var/opt/cray/imps/config/sets/myconfigset-p0
smw# cp /var/opt/cray/imps/config/sets/myconfigset-p0/worksheets/* /some/edit/
location
```

4. From the SMW, run the `xtrsipcfg_v2` script using the `-b` flag, which specifies that the `rsipd.`*`NID`*`.conf` files for each server be created.

```
crayadm@smw> xtrsipcfg_v2 -b
```

5. When the script prompts for the name of the config set to update, enter the config set name.

```
config_set_name: myconfigset-p0
Gathering Node Information
```

6. The script prompts whether to add isolated service nodes as RSIP clients. For this example, enter `N`, the default. If the response to this question is `y`, a list of the isolated service nodes displays. Enter a space delimited list of c-names to configure them as RSA clients. All other isolated nodes are configured as RSAP clients.

```
Should the isolated service nodes be setup as RSIP clients? [y/N]:
```

   Note that the list of isolated service nodes could be missing an isolated node that is targeted as an RSIP client. The SDB node for instance, may not show up because it has an Ethernet interface. Do not attempt to use non-isolated nodes as RSIP clients. However, exceptions can be made if the node simply has an interface that is connected only internally. The next prompt asks if there were any missing nodes to be added as RSIP clients (RSAP or RSA). It is possible to add the previously described nodes as clients in response to these prompts, but be very careful to only add nodes that do not have external network connectivity.

7. The script creates a list of compute nodes in a default location unless an alternative location is specified.

```
By default a file containing all compute_names is created in /tmp/
rsip_compute_names.txt
Refer to this file for the next steps if necessary.
Enter Alternate filepath for compute_names file or hit return:
```

8. Specify compute nodes to be used as RSA clients. By default all compute nodes are configured as RSAP clients. For this example, do not enter any nodes for RSA client configuration.

```
Enter a space delimited list of COMPUTE Node cnames to be RSA CLIENTS.
** Unlisted nodes will be configured as RSAP CLIENTS **

Compute RSA Clients:
```

   The script displays connectivity information to use in response to subsequent questions from the script.

```
Service node network connectivity:
(login)               c1-0c0s0n2: eth0 : 192.0.2.88 255.255.240.0
                                  eth1 : DOWN -
                      c1-0c1s3n1: eth0 : DOWN -
                                  eth1 : DOWN -
                                  eth2 : DOWN -
                                  eth3 : DOWN -
                      c1-0c1s3n2: eth0 : 192.0.2.253 255.255.0.0
                                  eth1 : DOWN -
                                  eth2 : 192.0.2.17 255.255.255.0
```

```
                                eth3 : DOWN -
                    c1-0c1s4n1: eth0 : DOWN -
                                eth1 : DOWN -
                                eth2 : DOWN -
                                eth3 : DOWN -
                    c1-0c1s4n2: eth0 : 192.0.2.43 255.255.240.0
                                eth1 : DOWN -
                                eth2 : DOWN -
                                eth3 : DOWN -
```

9. The script prompts for the c-names of service nodes to use as RSIP servers. *Specify only a dedicated RSIP node*. Clients are automatically assigned to servers by the script. Specify one or more service nodes from the connectivity information just provided by the script.

```
Auto Config Servers:  c1-0c1s4n2
```

```
Provide an Address Pool as a combination of IPs and IP ranges in a space
delimited list.
ex: 192.0.2.0-192.0.2.24 192.0.2.30 192.0.2.34-192.0.2.40
Leave this field empty if using RSAP-IP with the server's primary external
interface
```

10. The c-names of the specified servers are displayed. For each server, provide a pool of IP addresses to use. The script accepts a space-delimited list or a range of IP addresses. Not specifying a pool of available IP addresses causes the server to instead utilize its primary external interface. In this example, use multiple IP addresses for the node `c1-0c1s4n2`. The specified addresses should be on the same subnet. Accept the default for IPs reserved for RSA because no RSA clients or servers are configured in this example.

```
c1-0c1s4n2: 192.0.2.43 192.0.2.43
How many IPs should be reserved for RSA? [Default 0]:
```

11. Specify RSIP servers to manually assign clients to them. This example does not include any manually assigned clients to any servers, so press `Enter`.

```

Enter a space delimited list of node cnames for nodes that will be RSIP SERVERS.
For example: c0-0c0s7n0 c0-0c0s7n3 c0-0c1s1n3
* RSIP Servers may be manually or automatically configured with clients
You will list below the servers which you will manually assign clients to,
followed by the servers which you want clients automatically assigned to.
* RSIP Servers MUST have external network connectivity.

Manual Config Servers:
```

12. The script prompts for an RSIP port range and a system port range for each server. For this example, accept the defaults by pressing `Enter`.

```
Provide RSIP port range and System port range for the following servers (Non
overlapping ranges).
These ranges will be used only for RSAP IP.
Defaults RSIP: 1-60000 Default System: 60001-65535. Hit enter to use defaults
c1-0c1s4n2
     RSIP:
     System:
Should all subsequent servers utilize these settings? (Y/n):
```

13. The script displays the locations of the configuration files that it has created.

```
Created krsip config file at /var/opt/cray/imps/config/sets/myconfigset-p0/files/roles/rsip/etc/krsip.yaml
Created RSIPD.<nid>.conf files in /var/opt/cray/imps/config/sets/myconfigset-p0/files/roles/rsip/etc/opt/cray/
rsipd/
Created rsipd.yaml at /var/opt/cray/imps/config/sets/myconfigset-p0/files/roles/rsip/etc/opt/cray/rsipd/rsipd.yaml
```

14. Verify the contents of the RSIP configuration file.

```
crayadm@smw> cd /var/opt/cray/imps/config/sets/myconfigset-p0/files/roles/rsip/
etc/opt/cray/rsipd
rsipd.c1-0c1s4n2.conf rsipd.yaml
crayadm@smw> grep "pool " rsipd.c1-0c1s4n2.conf
pool 192.0.2.43.90
pool 192.0.2.43.91
```

Go to the topic *Update cray_net Worksheet for an Advanced RSIP Configuration* on page 258 to continue this configuration example.

Update cray_net Worksheet for an Advanced RSIP Configuration

Prerequisites

This procedure assumes that:

- The `xtrsipcfg_v2` script has been run.
- A work area has been set up for editing CLE configuration worksheets.
- The current directory is set to that work area.

```
smw# cd /some/edit/location
```

About this task

Update the `cray_net` worksheet with advanced RSIP settings. Then administer the config set mapping and re-run `cray-ansible`.

Procedure

1. Edit `cray_net_worksheet.yaml`.

```
smw# vi cray_net_worksheet.yaml
```

2. Update the RSIP host definition with values (bolded) shown in the following listing. If a RSIP host is not defined in the config set, add a host definition stanza for the RSIP server like the following one, placing it under `NOTE: Place additional 'host' setting entries here, if desired.` A sample host definition that includes default host settings is included in the worksheet under `cray_net.settings.hosts.data.common_name.sample_key_a: null`.

```
cray_net.settings.hosts.data.common_name.rsip_node: null
cray_net.settings.hosts.data.rsip_node.description: RSIP node
cray_net.settings.hosts.data.rsip_node.aliases:
- rsip
cray_net.settings.hosts.data.rsip_node.hostid: c1-0c1s4n2
cray_net.settings.hosts.data.rsip_node.host_type: ''
cray_net.settings.hosts.data.rsip_node.hostname: rsip1
cray_net.settings.hosts.data.rsip_node.standby_node: false
cray_net.settings.hosts.data.rsip_node.interfaces.common_name.eth0: null
```

```
cray_net.settings.hosts.data.rsip_node.interfaces.eth0.name: eth0
cray_net.settings.hosts.data.rsip_node.interfaces.eth0.description: Ethernet
connecting the RSIP node to the customer network.
cray_net.settings.hosts.data.rsip_node.interfaces.eth0.aliases: []
cray_net.settings.hosts.data.rsip_node.interfaces.eth0.network: login
cray_net.settings.hosts.data.rsip_node.interfaces.eth0.ipv4_address: 192.0.2.43
cray_net.settings.hosts.data.rsip_node.interfaces.eth0.mac: ''
cray_net.settings.hosts.data.rsip_node.interfaces.eth0.startmode: auto
cray_net.settings.hosts.data.rsip_node.interfaces.eth0.bootproto: static
cray_net.settings.hosts.data.rsip_node.interfaces.eth0.mtu: ''
cray_net.settings.hosts.data.rsip_node.interfaces.eth0.extra_attributes:
- IPADDR1='192.0.2.43.90/20'
- IPADDR2='192.0.2.43.91/20'
#cray_net.settings.hosts.data.rsip_node.interfaces.eth0.module: ''
#cray_net.settings.hosts.data.rsip_node.interfaces.eth0.params: ''
#cray_net.settings.hosts.data.rsip_node.interfaces.eth0.unmanaged_interface:
false
```

3. Import the completed `cray_net` worksheet to `myconfigset-p0`.

```
smw# cfgset update --worksheet-path \
'/some/edit/location/cray_net_worksheet.yaml' myconfigset-p0
```

4. Link the nodes to the config set. The following example updates `my_map` to link all system nodes to the new config set.

```
smw# cmap update --config-set myconfigset-p0 my_map
```

5. Use the `cmap setactive` command if necessary to make `my_map` active.

```
smw# cmap setactive my_map
```

6. Log in to the RSIP server node and re-run cray-ansible.

```
smw# ssh boot
crayadm@boot> ssh c1-0c1s4n2
user@host> /etc/init.d/cray-ansible start
```

7. Verify that multiple inets are displayed when the following command is run on the RSIP server node.

```
user@host> ip addr show eth0
```

The Node ARP Management Daemon (rca_arpd)

The node ARP management daemon (`rca_arpd`) manages the system ARP cache. This daemon deletes the IP to hardware address (ARP) mappings for failed nodes and reads them when they become available. It only manages ARP mappings on the high speed interconnect network and not external network interfaces such as Ethernet. If failover is configured, `rca_arpd` also manages ARP mappings for the backup boot or SDB node. When a node failed event from the primary boot or SDB node is received, `rca_arpd` updates the ARP mapping for the boot or SDB node virtual IP address to point to the backup node.

This functionality is included in the `cray-rca-compute` and `cray-rca-service` RPMs and is installed by default.

Create Logical Machines for Cray XC Series Systems

Configure a logical machine (sometimes known as a *system partition*) with the `xtcli part_cfg` command. Partition IDs are predefined as `p0` to `p31`. The default partition `p0` is reserved for the complete system and is no longer a valid ID once a system has been partitioned.

Cray XC Series systems can have one or more cabinets. Systems with one or two compute cabinets scale at the blade level. For larger liquid-cooled systems, every cabinet is fully populated (with 3 chassis), with the possible exception of the last cabinet.

For Cray XC Series systems, groups are made up of two-cabinet pairs starting from the beginning. The last group may not be completely full, and it can consist of 1 to 6 fully-populated chassis.

Multiple Group Systems

When a Cray XC Series system contains multiple groups, the system administrator can partition the system at a per-group level of granularity. Groups do not need to be sequentially positioned in a multi-group partition.

If a Cray XC Series system has more than 2 cabinets, every partition can consist of any number of groups; the last group (or remainder of system chassis that is not part of a full 6-chassis group) in the system should be considered a group whether it is fully-populated or not in this partitioning context.

Single Group, Multiple-chassis Systems

When a Cray XC Series system contains between two and six fully-populated chassis, then the administrator can partition the system at a per-chassis level of granularity. Each partition must be at least one full chassis, and a chassis cannot be shared between partitions. Chassis do not need to be sequentially positioned in a multi-chassis partition.

Single Chassis Systems

When a Cray XC Series system is composed of a single fully-populated chassis, each slot must be in the same partition with its corresponding even/odd pair, because even/odd pair nodes (for example, slot 0 and slot 1, or slot 8 and slot 9) share optical connections and therefore must be in the same partition.

There are 16 slots (or blades) in a single chassis, making 8 even/odd slot pairs, and a maximum of 8 partitions. Single chassis systems can have any combination of even/odd slot pairs (e.g., 4-4, 6-2, 4-2-2, 2-2-1-1-1-1), and even/odd slot pairs do not need to be sequentially positioned in a multiple slot pair partition. In order for a partition to be bootable, it must have a boot node, an SDB node, an I/O node, and a login node.

Configure a Logical Machine

The logical machine can have one of three states:

- `EMPTY` - not configured
- `DISABLED` - configured but not activated
- `ENABLED` - configured and activated

  When a partition is defined, its state changes to `DISABLED`. Undefined partitions are `EMPTY` by default.

The `xtcli part_cfg` command

Use the `xtcli part_cfg` command with the *`part_cmd`* option (`add` in the following example) to identify the operation to be performed and the *`part_option`* (`-m`, `-b`, `-d` and `-i`) to specify the characteristics of the logical machine. The boot image may be a raw device, such as `/raw0`, or a file.

Create a logical machine with a boot node and SDB node specifying the boot image path

- When using a file for the boot image, the same file must be on both the SMW and the `bootroot` at the same path.
- For the logical machine to be bootable, both the boot node and SDB node IDs must be specified.

```
crayadm@smw:~> xtcli part_cfg add p2 -m c0-0,c0-1,c0-2,c0-3 \
-b c0-0c0s0n0 -d c0-0c0s2n1 -i /bootimagedir/bootimage
```

To watch HSS events on the specified partition, execute the `xtconsumer -p partition_name` command.

To display the console text of the specified partition, execute the `xtconsole -p partition_name` command.

For more information, see the `xtcli_part(8)`, `xtconsole(8)`, and `xtconsumer(8)` man pages.

Boot a Logical Machine

The `xtbootsys --partition pN` option enables the administrator to indicate which logical machine (partition) to boot. If a partition name is not specified, the default partition `p0` (component name for the entire system) is booted. Alternatively, if a partition name is not specified and the `CRMS_PARTITION` environment variable is defined, this variable is used as the default partition name. Valid values are in the form `pN`, where *`N`* ranges from `0` to `31`.

`xtbootsys` manages a link from `/var/opt/cray/log/`*`partition`*`-current` to the current *`sessionid`* directory for that partition, allowing changes to `/var/opt/cray/log/p1-current`, for example.

Boot the System Using Another Snapshot

Prerequisites

A user must be `root` to run `snaputil`.

About this task

Use the `snaputil` command to roll back the system to a previous snapshot if a system becomes unstable or broken. Rolling back or forward can require switching the active config set if settings differ between the snapshots. Snapshot and config set synchronization is particularly critical when rolling back or forward between CLE releases because of significant config set differences.

Procedure

1. List the snapshots available on the system.

```
smw: # snapshot list
```

```
Status    Name
Created
--------  ----------------------------------------------------
------------------
          @                                                    2016-08-10
01:35:35
          SMW-8.1DV00_CLE-6.1DV00.20160928                     2016-09-28
10:41:59
          SMW-8.1DV00_CLE-6.1DV00.20160928.save1.postinstall   2016-09-28
12:15:30
          SMW-8.1DV00_CLE-6.1DV00.20160928.save2.postboot      2016-09-28
13:29:30
          SMW-8.1DV00_CLE-6.1DV00.20161003.save0.preupdate     2016-10-03
08:10:00
cur,def   SMW-8.1DV00_CLE-6.1DV00.20161003                     2016-10-03
08:13:50
          SMW-8.1DV00_CLE-6.1DV00.20161003.save1.postinstall   2016-10-03
10:09:52
          SMW-8.1DV00_CLE-6.1DV00.20161003.save2.postboot      2016-10-03
11:24:28
```

2. Roll back to the snapshot dated `20160928`.

```
smw # snaputil default SMW-8.1DV00_CLE-6.1DV00.20160928
```

```
subvolume SMW-8.1DV00_CLE-6.1DV00.20160928 is now default.
```

3. When a snapshot switch requires config set synchronization, use the `cmap` command to specify the config set to use when booting CLE nodes with a new default snapshot.

```
smw # cmap update --config-set config_set_name
```

4. Reboot the SMW. When the system is booted, it will use the new default snapshot.

Configure the NFS client to Mount the Exported Lustre File System

About this task

Depending on the site client system, the configuration may be different. This procedure contains general information that will help configure the client system to properly mount the exported Lustre file system. Consult the client system documentation for specific configuration instructions.

Procedure

1. As `root`, verify that the `nfs` client service is started at boot.

2. Add a line to the `/etc/fstab` file to mount the exported file system. (For more information on NFS mount options, see the `mount(8)` and `nfs(5) man` pages.)

```
server@network:/filesystem /client/mount/point lustre file_system_options 0 0
```

Recommended file system mount options.

rsize=1048576,wsize=1048576	Set the read and write buffer sizes from the server at 1MiB. These options match the NFS read/write transaction to the Lustre filesystem block size, which reduces cache/buffer thrashing on the service node providing the NFS server functionality.
soft,intr	Use a soft interruptible mount request.
async	Use asynchronous NFS I/O. Once the NFS server has acknowledged receipt of an operation, let the NFS client move along even though the physical write to disk on the NFS server has not been confirmed. For sites that need end-to-end write-commit validation, set this option to `sync` instead.
proto=tcp	Force use of TCP transport—this makes the larger `rsize`/`wsize` operations more efficient. This option reduces the potential for UDP retransmit occurrences, which improves end-to-end performance.
relatime,timeo=600,local_lock=none	Lock and time stamp handling, transaction timeout at 10 minutes.
nfsvers=3	Use NFSv3 specifically. NFSv4 is not supported at this time.

3. Mount the file system manually or reboot the client to verify that it mounts correctly at boot.

Define Bind Mount Points Within a Configuration Set

About this task

When a site's directory needs to be bind-mounted much like the programming environment image root, create a new image root that populates the bind mount point.

Procedure

1. Create a new image root using a recipe that populates a site directory, for example, `/opt/site`. This example uses an image root called `site_image_root_name` as the result of building this recipe into an image root.

```
smw# recipe create site_image_recipe
```

2. Extend the new recipe with sub-recipes, repositories, package collections, RPMs, and `post_build_chroot` commands and `post_build_copy` files to get the site content into `/opt/site` of the image root.

3. Build the image root from the recipe.

```
smw# image create -r site_image_recipe site_image_root_name
```

4. Push the `site_image_root_name` image from the SMW to the boot node.

```
smw# image sqpush -d boot site_image_root_name
```

5. Update the `cray_image_binding` worksheet with a new profile named `site`. The profile settings define the image name and the bind directories. The listing that follows shows the necessary settings.

```
cray_image_binding.settings.profiles.data.profile_name.site: null
#cray_image_binding.settings.profiles.data.site.image: site_image_root_name
cray_image_binding.settings.profiles.data.site.bind_directories:
/opt/site
#cray_image_binding.settings.profiles.data.site.callbacks: []
cray_image_binding.settings.profiles.data.site.enabled: false
```

   To have more than one directory from this image root bind mounted, add them to the directories setting.

```
cray_image_binding.settings.profiles.data.site.bind_directories:
/opt/site
/opt/another_site
/opt/last_site
```

Enable Multipath on an Installed XC System

Prerequisites

This procedure assumes that the Cray XC system has already been installed and configured without multipath having been enabled. If performing a fresh install, this procedure is not necessary if multipath was already set up using *Prepare and Update the Global Config Set* on page 153 or *Update cray_multipath Worksheet*.

About this task

This procedure describes how to enable multipath on a Cray XC system that has already been installed and configured. Note that multipath does NOT need to be fully cabled to be used. The multipath driver can handle using one path or many.

> **IMPORTANT:** If this system has partitions, repeat any steps that modify 'p0' for each partition. Multipath must be enabled everywhere or nowhere; enabling it on only part of the system causes problems.

Procedure

1. Start the multipath daemon now.

```
smw# systemctl start multipathd
```

   Later in this procedure, the `cray-ansible` command will be used to enable the multipath daemon.

2. Obtain the host ID of the SMW and the cnames of any nodes in the system that are connected to the boot RAID with an HBA (host bus adapter).

   The system should be bounced or booted for `xtcheckhss` to return a proper list.

```
smw# hostid
{8 digit hostid}
smw# xtcheckhss --detail=f --pci
```

Look for cnames with HBAs like 'QLogic_ISP2532_8Gb_Fibre_Channel_HBA.'

———— UPDATE CRAY_MULTIPATH IN GLOBAL CONFIG SET ————

3. Use the configurator to update cray_multipath in the global config set.

```
smw# cfgset update -s cray_multipath -m interactive -l advanced global
```

a. Enable multipath.

Enter **E** at the configurator prompt to toggle the enable status of the multipath service, which is disabled by default.

```
Cray Multipath Configuration Service Menu [default: save & exit - Q] $ E
```

b. Add the host ID and cnames obtained in step *2*.

At the prompt, enter **1** to select the node_list setting, then enter **C** to configure it. At the prompt for that setting, enter values **+** to add node_list entries: add the host IDs and cnames obtained in step *2*, one per line. When finished, press **Ctrl-d** and then **<cr>** to set the entries.

```
Cray Multipath Configuration Service Menu [default: save & exit - Q] $ 1
...
Cray Multipath Configuration Service Menu [default: configure - C] $ C
...
cray_multipath.settings.multipath.data.node_list
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $ +
Add node_list (Ctrl-d to exit) $
```

4. Correct the values of three pre-populated multipath device settings.

Perform this step if this system was updated from CLE 6.0.UP03 or an earlier release AND these values were not corrected during the update.

a. View all enabled devices.

At the prompt, enter **33** to select the enabled_devices setting, then enter **C** to configure it.

```
Cray Multipath Configuration Service Menu [default: save & exit - Q] $ 33

Cray Multipath Configuration Service Menu [default: configure - C] $ C
```

At the prompt for this setting, enter ***** to view all of the pre-populated device settings.

```
cray_multipath.settings.enabled_devices
[<cr>=set 11 entries, +=add an entry, ?=help, @=less] $ *
```

b. Change the value of the path grouping policy field for the DDN_EF3015 device.

Find the DDN_EF3015 device in the list of enabled devices, and enter its number (5 in this example) followed by 'd' and '*' to select and edit the path_grouping_policy field.

```
cray_multipath.settings.enabled_devices
[<cr>=set 11 entries, +=add an entry, ?=help, @=less] $ 5d*
```

If this field is not already set to **group_by_prio**, set it to that value now.

```
cray_multipath.settings.enabled_devices.data.DDN_EF3015.path_grouping_policy
[<cr>=keep 'multibus', <new value>, ?=help, @=less] $ group_by_prio
```

c. Change the value of the product field for the DDN_SFA12K_20 device.

Find the DDN_SFA12K_20 device in the list of enabled devices, and enter its number (10 in this example) followed by 'b' and '*' to select and edit the product field.

```
cray_multipath.settings.enabled_devices
[<cr>=set 11 entries, +=add an entry, ?=help, @=less] $ 10b*
```

If this field is not already set to **SFA12K-20**, set it to that value now.

```
cray_multipath.settings.enabled_devices.data.DDN_SFA12K_20.product
[<cr>=keep 'SFA12K20', <new value>, ?=help, @=less] $ SFA12K-20
```

d. Change the value of the product field for the DDN_SFA12K_40 device.

Find the DDN_SFA12K_40 device in the list of enabled devices, and enter its number (11 in this example) followed by 'b' and '*' to select and edit the product field.

```
cray_multipath.settings.enabled_devices
[<cr>=set 11 entries, +=add an entry, ?=help, @=less] $ 11b*
```

If this field is not already set to **SFA12K-40|SFA12KX***, set it to that value now.

```
cray_multipath.settings.enabled_devices.data.DDN_SFA12K_40.product
[<cr>=keep 'SFA12K40', <new value>, ?=help, @=less] $ SFA12K-40|SFA12KX*
```

Set the enabled_devices entries, but DO NOT save changes and exit the configurator yet.

```
cray_multipath.settings.enabled_devices
[<cr>=set 11 entries, +=add an entry, ?=help, @=less] $ <cr>
```

5. Correct the syntax of the multipath blacklist devices setting.

Perform this step if this system was updated from CLE 6.0.UP03 or an earlier release AND these values were not corrected during the update.

The multipath configuration contains syntax that works under SLES 12 but not under SLES 12 SP2. That syntax must be corrected in three places (more if there is more than one CLE config set):

- the `/etc/multipath.conf` file in the new SP2 snapshot
- multipath configuration service template in the global config set
- multipath configuration service template in every CLE config set in use

The `/etc/multipath.conf` file must be corrected manually because the corrections are needed for the init boot phase, and any changes to the multipath configuration service (the preferred approach) would not be reflected in `/etc/multipath.conf` until cray-ansible runs, which on the SMW occurs only in the multi-user boot phase. However, correcting only `/etc/multipath.conf` is not sufficient, because when cray-ansible runs in multi-user phase, that file is replaced with one that reflects the settings in the multipath configuration service. Therefore, the corrections must be made in the global and CLE config sets as well. Note that the corrected syntax works under both SLES 12 and SLES 12 SP2.

a. Select the blacklist_devices setting.

At the configuration service menu prompt, enter **31** to select blacklist_devices, and then enter **c** to configure that setting. Both the vendor and product values will be changed from ***** to **.***.

```
Cray Multipath Configuration Service Menu [default: save & exit - Q] $ 31
Cray Multipath Configuration Service Menu [default: configure - C] $ C
*************************** cray_multipath.settings.blacklist_devices
***************************
      blacklist_devices
        Enter the devices which you would like to blacklist for  multipath.
By
        default, all devices are blacklisted. Remove the  'all' key in this
        setting to de-blacklist all devices.

    Configured Values:
        1) 'all'
            a) vendor: *
            b) product: *

    Inputs: menu commands (? for help)

|--- Information
|  *    Multiple 'blacklist_devices' entries can be added using this menu
|---

cray_multipath.settings.blacklist_devices
[<cr>=set 1 entries, +=add an entry, ?=help, @=less] $
```

b. Enter **1a*** to change the vendor value.

```
cray_multipath.settings.blacklist_devices
[<cr>=set 1 entries, +=add an entry, ?=help, @=less] $ 1a*
```

c. Enter **.*** to update the current value to the correct value.

```
cray_multipath.settings.blacklist_devices.data.all.vendor
[<cr>=keep '*', <new value>, ?=help, @=less] $ .*
```

d. Enter **1b*** to change the product value.

```
cray_multipath.settings.blacklist_devices
[<cr>=set 1 entries, +=add an entry, ?=help, @=less] $ 1b*
```

e. Enter **.*** to update the current value to the correct value.

```
cray_multipath.settings.blacklist_devices.data.all.product
[<cr>=keep '*', <new value>, ?=help, @=less] $ .*
```

f. Set the changed blacklist_devices entry.

```
cray_multipath.settings.blacklist_devices
[<cr>=set 1 entries, +=add an entry, ?=help, @=less] $ <cr>
```

g. Save changes and exit the configurator.

```
Cray Multipath Configuration Service Menu [default: save & exit - Q] $ Q
```

h. Edit the multipath configuration file.

```
smw# vi /etc/multipath.conf
```

The following section in `/etc/multipath.conf` shows the incorrect `vendor` and `product` values of `"*"` and `"*"`:

```
blacklist {
    devnode "^(ram|raw|loop|fd|md|dm-|sr|scd|st)[0-9]*"
    devnode "^hd[a-z]"
    devnode "^cciss!c[0-9]d[0-9]*"
    device {
        vendor  "*"
        product "*"
    }
}
```

The same section displayed with correct `vendor` and `product` values:

```
blacklist {
    devnode "^(ram|raw|loop|fd|md|dm-|sr|scd|st)[0-9]*"
    devnode "^hd[a-z]"
    devnode "^cciss!c[0-9]d[0-9]*"
    device {
        vendor  ".*"
        product ".*"
    }
}
```

6. If still in the configurator, save changes and exit the configurator now.

If the previous step was skipped because the values had already been corrected during an update or this system had a fresh install of CLE 6.0.UP04, then the configurator may still be running from an earlier step.

```
Cray Multipath Configuration Service Menu [default: save & exit - Q] $ Q
```

———— UPDATE CRAY_BOOTRAID IN GLOBAL CONFIG SET ————

7. Use the configurator to update cray_bootraid in the global config set.

```
smw# cfgset update -s cray_bootraid -m interactive global
```

a. Select the storage sets setting to configure it.

```
Boot RAID Configuration Service Menu [default: save & exit - Q] $ 1
...
Boot RAID Configuration Service Menu [default: configure - C] $ C
```

b. For each device in the cledefault and smwdefault storage sets, modify the path name from scsi to dm-uuid-mpath.

This example shows selecting the cledefault (1) volume group (a) boot_node_vg (1) devices (b) field. The * indicates that the selection is to be edited.

```
cray_bootraid.settings.storage_sets
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $ 1a1b*
```

Remove the scsi path name and replace it with the dm-uuid-mpath name.

```
cray_bootraid.settings.storage_sets.data.cledefault.volume_groups.boot_node_vg.devices
[<cr>=set 1 entries, +=add an entry, ?=help, @=less] $ 1-

cray_bootraid.settings.storage_sets.data.cledefault.volume_groups.boot_node_vg.devices
[<cr>=set 0 entries, +=add an entry, ?=help, @=less] $ +
Add devices (Ctrl-d to exit) $ /dev/disk/by-id/dm-uuid-mpath-3600a0980009ec0750000010a5762af70
```

```
Add devices (Ctrl-d to exit) $ <Ctrl-d>
```

Press **Enter** (<cr>) to set the entries for the boot_node_vg volume group.

```
cray_bootraid.settings.storage_sets.data.cledefault.volume_groups.boot_node_vg.devices
[<cr>=set 1 entries, +=add an entry, ?=help, @=less] $ <cr>
```

Repeat substep b for each device in the cledefault and smwdefault storage sets. Enter * at the prompt to see all storage set entries.

- To select the next cledefault volume group device (sdb_node_vg), enter **1a2b*** at the prompt. If there are more cledefault volume groups, increment the third character to select each one (**1a3b***, **1a4b***, and so forth).
- To select the first smwdefault volume group device (smw_node_vg), enter **2a1b*** at the prompt. If there are more smwdefault volume groups, increment the third character to select each one (**2a2b***, **2a3b***, and so forth).

c. Set the storage set entries, then save changes and exit the configurator.

```
cray_bootraid.settings.storage_sets
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $ <cr>
...
Boot RAID Configuration Service Menu [default: save & exit - Q] $ Q
```

———— UPDATE CRAY_MULTIPATH IN CLE CONFIG SET(S) ————

8. Use the configurator to set up inheritance for multipath in the CLE config set of the active SMW.

This example uses 'p0' as the name of the CLE config set. Substitute the actual name used for this system.

```
smw# cfgset update -s cray_multipath -m interactive p0
```

Enter **I** at the configurator prompt to toggle the inherit status of the multipath service, which is disabled by default. This means that multipath settings in the global config set will be used instead of multipath settings in the CLE config set.

```
Cray Multipath Configuration Service Menu [default: save & exit - Q] $ I
```

Repeat this step for each CLE config set.

———— VALIDATE CONFIG SETS AND APPLY CHANGES ————

9. Validate the config sets and run cray-ansible to apply the config set changes.

a. Validate the config sets.

```
smw# cfgset validate global

smw# cfgset validate p0
```

b. Run cray-ansible.

```
smw# /etc/init.d/cray-ansible start
```

———— FOR SYSTEMS USING DAL ————

10. For systems using direct-attached Lustre (DAL), update the `dal.fs_defs` file.

Repeat these steps for each partition.

a. Locate the current `fs_defs` files (typically stored in `/home/crayadm`).

```
smw# find /home/crayadm -name "*fs_defs*"
```

b. Find the `fs_defs` files that are currently installed and compare with the one found in `/home/crayadm`.

```
smw# cd /var/opt/cray/imps/config/sets
smw# find p0 -name "*fs_defs*"
```

```
smw# diff /home/crayadm/dal.fs_defs \
p0/lustre/.lctrl/dal.fs_defs.20160205.1454685527
```

c. Edit the `dal.fs_defs` file to ensure that it has the proper mpath paths in it.

```
smw# cd /home/crayadm

smw# sed -i.nompath \
's/\/dev\/disk\/by-id\/scsi/\/dev\/disk\/by-id\/dm-uuid-mpath/g' \
dal.fs_defs

smw# cp -p dal.fs_defs dal.fs_defs.mpath
```

d. Install the new `dal.fs_defs` file using `lustre_control`.

```
smw# lustre_control install -c p0 /home/crayadm/dal.fs_defs
```

———— SHUT DOWN AND REBOOT SYSTEM ————

11. Shut down all partitions of the Cray system.

12. Reboot the SMW.

13. Boot the Cray system.

Change Lustre Versions

Prerequisites

System must be installed.

About this task

This procedure describes how to build image root and boot images with a version of Lustre other than the default version in the standard image recipes for CLE 6.x systems.

Procedure

1. Identify recipes with non-default Lustre.

   With CLE 6.x, the default Lustre version is Lustre 2.7.2 and that version is included in the package collections for all recipes. However, a second set of recipes exist which use package collections with the non-default version of Lustre `rpms` (2.5.4). These additional recipes can be located by this command.

```
smw# recipe list | grep lustre
compute-large-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari
compute-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari
elogin-large-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari
elogin-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari
initrd-compute-large-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari
initrd-login-large-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari
initrd-login-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari
login-large-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari
login-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari
service-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari
```

   BUILD CLE IMAGES WITH NON-DEFAULT LUSTRE VERSION

2. Customize the `cray_image_groups` configuration file, as needed, by editing `/var/opt/cray/imps/config/sets/global/config/cray_image_groups.yaml` and by adding stanzas for standard and/or netroot images to be created.

   Choose only the set of tmpfs recipes or the set of netroot recipes which matches the existing recipes in use on this system. There is a single recipe for service since it is always tmpfs.

```
smw# vi /var/opt/cray/imps/config/sets/global/config/cray_image_groups.yaml
```

   For tmpfs.

```
cray_image_groups:
  lustre25:
     - recipe: "compute-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari"
       dest: "compute-lustre-2.5{note}_cle_{cle_release}-build{cle_build}{patch}_sles_12sp2-
created{date}.cpio"
       nims_group: "compute"
     - recipe: "login-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari"
       dest: "login-lustre-2.5{note}_cle_{cle_release}-build{cle_build}{patch}_sles_12sp2-
created{date}.cpio"
       nims_group: "login"
     - recipe: "service-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari"
       dest: "service-lustre-2.5{note}_cle_{cle_release}-build{cle_build}{patch}_sles_12sp2-
created{date}.cpio"
       nims_group: "service"
```

   For netroot.

```
cray_image_groups
  lustre25:
       - recipe: "initrd-compute-large-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari"
         dest: "initrd-compute-large-lustre-2.5{note}_cle_{cle_release}-build{cle_build}
{patch}_sles_12sp2-created{date}.cpio"
         nims_group: "compute"
       - recipe: "initrd-login-large-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari"
         dest: "login-large-lustre-2.5{note}_cle_{cle_release}-build{cle_build}{patch}_sles_12sp2-
created {date}.cpio"
         nims_group: "login"
       - recipe: "service-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari"
         dest: "service-lustre-2.5{note}_cle_{cle_release}-build{cle_build}{patch}_sles_12sp2-
created{date}.cpio"
         nims_group: "service"
```

3. Run `imgbuilder` to make Lustre images.

```
smw# imgbuilder -g lustre25
```

a. For netroot only, push image roots to login node.

```
smw# image sqpush -d boot login-large-lustre-2.5_cle_version_and_build.cpio
```

b. For netroot only, push image roots to compute node.

```
smw# image sqpush -d boot compute-large-lustre-2.5_cle_version_and_build.cpio
```

ASSIGN NEW LUSTRE IMAGES TO TEST NODES

4. Determine names of boot images built above.

```
smw# image list | grep lustre
```

5. Use `cnode` to assign the boot images to nodes for testing.

a. Assign service image.

```
smw# cnode update -i \
/var/opt/cray/imps/boot_images/service-lustre-2.5_cle_version_and_build.cpio c9-9c0s0n1
```

b. Assign DAL image.

```
smw# cnode update -i \
/var/opt/cray/imps/boot_images/dal-lustre-2.5_cle_version_and_build.cpio c8-8c0s0n2
```

c. Assign login image.

- For tmpfs:

```
smw# cnode update -i \
/var/opt/cray/imps/boot_images/login-lustre-2.5_cle_version_and_build.cpio c8-8c0s0n1
```

- For netroot:

```
smw# cnode update -i \
/var/opt/cray/imps/boot_images/initrd-login-large-lustre-2.5_cle_version_and_build.cpio \
-k netroot=login-large-lustre-2.5_cle_version_and_build c8-8c0s0n1
```

d. Assign compute image.

- For tmpfs:

```
smw# cnode update -i \
/var/opt/cray/imps/boot_images/compute-lustre-2.5_cle_version_and_build.cpio c7-7c0s0n1
```

- For netroot:

```
smw# cnode update -i \
/var/opt/cray/imps/boot_images/initrd-compute-large-lustre-2.5_cle_version_and_build.cpio \
-k netroot=compute-large-lustre-2.5_cle_version_and_build c7-7c0s0n1
```

WARMBOOT TEST NODES WITH NEW LUSTRE IMAGES

6. Warmboot test nodes with new Lustre images.

a. Login as `crayadm`.

```
smw# su - crayadm
```

b. Shutdown node.

```
crayadm@smw> xtnmi c8-8c0s0n1
```

c. Wait sixty seconds.

```
crayadm@smw> sleep 60
```

d. Reboot node with new image.

```
crayadm@smw> xtbootsys --reboot -r "warmboot to test Lustre-2.5" c8-8c0s0n1
```

e. Repeat for each type of node to be tested.

REBOOT ENTIRE CLE SYSTEM WITH NEW LUSTRE IMAGES

7. Assign Lustre 2.5 images to all nodes.

a. Assign to all service nodes.

```
smw# cnode update --filter group=service -i \
/var/opt/cray/imps/boot_images/service-lustre-2.5_cle_version_and_build.cpio
```

b. Assign to all login nodes.

- For tmpfs:

```
smw# cnode --filter group=login update -i \
/var/opt/cray/imps/boot_images/login-lustre-2.5_cle_version_and_build.cpio
```

- For netroot:

```
smw# cnode update \
-i /var/opt/cray/imps/boot_images/initrd-login-large-lustre-2.5_cle_version_and_build.cpio \
-k netroot=login-large-lustre-2.5_cle_version_and_build --filter group=login
```

c. Assign to all compute nodes.

- For tmpfs:

```
smw# cnode update --filter group=compute -i \
/var/opt/cray/imps/boot_images/compute-lustre-2.5_cle_version_and_build.cpio
```

- For netroot:

```
smw# cnode update -i \
/var/opt/cray/imps/boot_images/initrd-compute-large-lustre-2.5_cle_version_and_build.cpio \
-k netroot=compute-large-lustre-2.5_cle_version_and_build.cpio --filter group=compute
```

For netroot only, if the compute-large and login-large image roots were not pushed to the boot node when testing the new images, push them to the boot node before rebooting the entire system. The boot node must be booted for the `image sqpush` command to succeed.

8. Shutdown CLE.

```
crayadm@smw> xtbootsys -s last -a auto.xtshutdown
```

9. Boot CLE.

```
crayadm@smw> xtbootsys -a auto.sitestartup
```

10. Build recipes and deploy boot image for eLogin nodes using non-default Lustre.

To build eLogin images, export them to the CMC and reboot the eLogin node with the new boot image described in the "Configure and Manage an eLogin Image" section of the *XC Series eLogin Installation Guide*.

The important difference from that guide is choosing the appropriate image recipe which includes the non-default version of Lustre. This example shows the elogin recipes which match the netroot and tmpfs login recipes configured for the XC system. Select the image recipe that most closely resembles what the XC login node uses.

```
smw# recipe list | grep lustre | grep elogin
elogin-large-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari
elogin-lustre-2.5_cle_6.0up04_sles_12sp2_x86-64_ari
```

Repurpose Compute Nodes

When a compute node is configured for a non-compute role, that node is a *repurposed compute node*. Compute nodes can be repurposed to become service nodes for use as tier2 servers (recommended) or in other capacities. Compute nodes should not be repurposed as service nodes for services that require external connectivity.

Use the `xtcli mark_node` command to repurpose a node in a compute blade. In this example, two compute nodes are being repurposed as service nodes and marked accordingly in the HSS database.

```
crayadm@smw> xtcli mark_node service c0-0c0s2n0,c0-0c0s2n1
```

Note that service nodes can be repurposed as compute nodes as well. In that case, the command would be **`xtcli mark_node compute`**.

Node Attributes

Users can control the selection of the compute nodes on which to run their applications and can select nodes on the basis of desired characteristics (*node attributes*). This allows a placement scheduler to schedule jobs based on the node attributes.

A user invokes the `cnselect` command to specify node-selection criteria. The `cnselect` script uses these selection criteria to query the table of node attributes in the SDB and returns a node list to the user based on the results of the query. When launching the application, the user includes the node list using the aprun -L *`node_list`* option as described on the `aprun(1)` man page. The ALPS placement scheduler allocates nodes based on this list.

To meet specific user needs, the administrator can modify the `cnselect` script. For additional information about the `cnselect` script, see the `cnselect(1)` man page.

View and Temporarily Set Node Attributes

Use the `xtprocadmin` command to view current node attributes. The `xtprocadmin -A` option lists all attributes of selected nodes. The `xtprocadmin -a attr1,attr2` option lists selected attributes of selected nodes.

An administrator can use the `xtprocadmin -a attr=value` command to temporarily set certain site-specific attributes. Using the `xtprocadmin -a attr=value` command to set certain site-specific attributes is not persistent across reboots. Attribute settings that are intended to be persistent across reboots (such as labels) must be specified in the `attr.defaults` file.

> **NOTE:** For compute nodes, `xtprocadmin` changes to attributes require restarting the `apbridge` daemon on the boot node in order for ALPS to detect changes that the `xtprocadmin` command has made to the SDB. Restarting the other ALPS components (for example, on the SDB node or on the login node if they are separate nodes) is not necessary. To restart `apbridge`, log into the boot node as `root` and execute the following command:
>
> ```
> boot:~ # /etc/init.d/alps restart
> ```

For example, the following command creates a new `label1` attribute value for the compute node whose NID is 350. The `xtprocadmin` command must be executed by `root` from a service node and the SDB must be running:

```
boot:~ # xtprocadmin -n 350 -a label1=eedept
```

```
Connected
NID    (HEX)    NODENAME    TYPE      LABEL1
350    0x15e    c1-0c1s0n0  compute   eedept
```

Then restart the `apbridge` daemon on the boot node in order for ALPS to detect changes that the `xtprocadmin` command has made to the SDB.

```
boot:~ # /etc/init.d/alps restart
```

The XTAdmin Database segment Table

The XTAdmin database contains a `segment` table that supports the memory affinity optimization tools for applications and CPU affinity options for all Cray compute nodes. The CPU affinity options apply to all Cray multicore compute nodes.

The `segment` table is similar to the `attributes` table but differs in that a node may have multiple segments associated with it; the `attributes` table provides summary information for each node.

In order to address the application launch and placement requirements for compute nodes with two or more NUMA nodes, the Application Level Placement Scheduler (ALPS) requires additional information that characterizes the intranode topology of the system. This data is stored in the `segment` table of the XTAdmin database and acquired by `apbridge` when ALPS is started, in much the same way that node attribute data is acquired.

The `segment` table contains the following fields:

node_id	The node identifier that maps to the `nodeid` field of the `attributes` table and `processor_id` field of the `processor` table.
socket_id	Contains a unique ordinal for each processor socket.
die_id	Contains a unique ordinal for each processor die; with this release, `die_id` is `0` in the segment table and is otherwise unused (reserved for future use).
numcores	The number of integer cores per node; in systems with accelerators this only applies to the host processor (CPU).
coremask	The processor core mask. The coremask has a bit set for each core of a CPU. 24-core nodes will have a value of `16777215` (hex `0xFFFFFF`). `coremask` is deprecated and will be removed in a future release.
mempgs	Represents the amount of memory available, in Megabytes, to a single segment.

The `/etc/sysconfig/xt` file contains `SDBSEG` field, which specifies the location of the `segment` table file; by default, `SDBSEG=/etc/opt/cray/sdb/segment`.

To update the `segment` table, use the following service database commands:

- `xtdb2segment`, which converts the data into an ASCII text file that can be edited
- `xtsegment2db`, which writes the data back into the database file

For more information, see the `xtdb2segment(8)` and `xtsegment2db(8)` man pages.

After manually updating the `segment` table, log on to any login node or the SDB node as root and execute the `apmgr resync` command to request that ALPS reevaluate the configuration node segment information and update its information.

If ALPS or any portion of the feature fails in relation to segment scheduling, ALPS reverts to the standard scheduling procedure.

Apply Rolling Patches to Compute Nodes with `cnat`

Prerequisites

- The system requires access to a workload manager (WLM) with administrative privileges.
- Enable the rolling patches service in the config set by editing the `cray_cnat_worksheet.yaml` file. Uncomment `cray_cnat.enabled` and set it to `true`.

About this task

A rolling patch applies a patch to a set of compute nodes without rebooting the system. The patch is applied to compute nodes between jobs. Applying rolling patches using `cnat` (short for compute node administrative tool) is not supported for service nodes. Patches with dependencies requiring a full system reboot do not support rolling patches. Another qualification for rolling patches is that patches must be updated within a CLE update release. Upgrades between update releases (for example, from CLE 6.0 UP01 to CLE 6.0 UP02) require a system reboot.

The `cnat` command runs a batch script through a workload manager and ensures that it runs successfully on each specified node. This allows administrative tasks, such as a rolling patch, to run on compute nodes without interfering with user jobs. The `cnat (1)` man page on the login node provides syntax and other details.

For patches that require node reboots, the `cnat` command uses the provided `cnat-reboot` script to control the reboot of compute nodes specified for a rolling patch. The `cnat-status` command returns information about a `cnat` run. See the `cnat-reboot (1)` and `cnat-status (1)` man pages for information about these commands.

Procedure

1. Set up the WLM. These WLMs are supported: Cobalt, Moab/Torque, PBS, and Slurm. The following setup information is for PBS and Slurm. Moab/Torque do not require any special setpup.
   - For PBS:
     1. Enable manager access to the server for the user running `cnat`.

        ```
        crayadm@login> qmgr -c 'set server managers+="user@hostname"'
        ```

     2. Add `cnat` to the `PBS_HOME/server_priv/resourcedef` file by appending this line to the file:

        ```
        cnat type=boolean flag=h
        ```

     3. Restart the PBS sever.

        ```
        crayadm@login> /etc/init.d/pbs restart
        ```

- For Slurm, `cnat` must be configured to submit to a partition without `Shared=FORCE` set. `cnat` must be run as `SlurmUser` or `root`.

2. Make a directory on the SMW (if it does not already exist) to hold any patches that may be available on CrayPort.

```
smw# mkdir -p /var/opt/cray/patchsets
```

3. Download patches to the `patchsets` directory on the SMW, as described in the release notes. The default location is `/var/adm/cray/release/patchsets`.

4. Run the `LOAD` script that is included in the patch.

   A `LOAD` script usually does following setup tasks:

   - Creates a SMW file system snapshot (optionally)
   - Backs up the Node Image Mapping Service (NIMS) active map
   - Backs up the CLE and global config sets
   - Mounts the patch ISO image file
   - Copies the RPMs from the ISO to the appropriate repositories on the SMW
   - Refreshes the repository metadata
   - Copies patch instructions and support files to `/var/opt/cray/patchsets/`*`patchset_directory`*

5. Follow the instructions in the `/var/opt/cray/patchsets/`*`patchset_directory`*`/README` file.

   An `INSTALL` script in the patch directory performs the necessary tasks to get the patched code into the appropriate image roots and boot images. This script will print further instructions required to get the patch changes onto the affected compute nodes.

6. When the README indicates that rolling updates are supported for the patch, run the `cnat` command as instructed by the README. Run `cnat` on the login node as the `crayadm` user, though some WLMs may have different user execution requirements. The example that follows calls the `cnat-reboot` script to reboot the patched compute nodes, but not all patches require a reboot.

```
crayadm@login> module load cnat
crayadm@login> cnat -n <node_list> /opt/cray/cnat/default/bin/cnat-reboot
```

7. Use the `cnat-status` script for information about the `cnat` run. The example command that follows specifies the output directory created for the `cnat` run that initiated the update. The `cnat-status` script output is placed in this directory and also displayed on the console.

```
crayadm@login> cnat-status cnat-20160502101159
```

Apply Live Updates to Nodes

Prerequisites

Enable the live updates service in the config set by editing the `cray_liveupdate_worksheet.yaml` file. Uncomment `cray_liveupdate.enabled` and set it to `true`.

About this task

A live update is an update that can be applied to running nodes. Live updates use package managers, such as `zypper` and `yum`, to install updated content on booted nodes. Live updates can be applied to both service nodes and compute nodes.

The `INSTALL` script for a patch updates the package repositories and node images on the SMW. When a patch can be applied with live updates, the patch script and `README` file provide further instructions to the administrator to properly update the images on the relevant booted nodes.

Procedure

1. Make a directory on the SMW (if it does not already exist) to hold any patches that may be available on CrayPort.

   ```
   smw# mkdir -p /var/opt/cray/patchsets
   ```

2. Download patches to the `patchsets` directory on the SMW, as described in the release notes. The default location is `/var/adm/cray/release/patchsets`.

3. Run the `LOAD` script that is included in the patch.

   A `LOAD` script usually does following setup tasks:

   - Creates a SMW file system snapshot (optionally)
   - Backs up the Node Image Mapping Service (NIMS) active map
   - Backs up the CLE and global config sets
   - Mounts the patch ISO image file
   - Copies the RPMs from the ISO to the appropriate repositories on the SMW
   - Refreshes the repository metadata
   - Copies patch instructions and support files to `/var/opt/cray/patchsets/`*`patchset_directory`*

4. Follow the instructions in the `/var/opt/cray/patchsets/`*`patchset_directory`*`/README` file.

   An `INSTALL` script in the patch directory performs the necessary tasks to get the patched code into the appropriate system images. This script will print further instructions required to get the patch changes onto the affected system nodes.

Reuse One or More Previously-failed HSN Links

About this task

To integrate failed links back into the HSN configuration, the `xtwarmswap` command may be invoked with one of the following:

- `-s` *`LCB, ...`*, specifying the list of LCBs to bring back up
- `-s all`, to bring in all available LCBs
- `-s none`, to cause a reroute without changing the LCBs that are in use

Procedure

1. Execute an `xtwarmswap -s LCB_names -p partition_name` to tell the system to reroute the HSN using the specified set of LCBs in addition to those that are currently in use.

   Doing so will clear the alert flags on the specified LCBs automatically. If the warm swap fails, the alert flag will be restored to the specified LCBs.

2. Execute an `xtwarmswap -s all -p partition_name` command to tell the system to reroute the HSN using all available links.

The `xtwarmswap` command results in `xtnlrd` performing the same link recovery steps as for a failed link, but with two differences: no alert flags are set, and an `init_new_links` and a `reset_new_links` step are performed to initialize both ends of any links to be used, before new routes are asserted into the Aries™ routing tables.

The elapsed time for the warm swap synchronization operation is typically about `30` seconds.

Add or Remove from Service

To specify one or more high-speed network (HSN) cables to add or remove from service, use the `xtwarmswap --add-cable` command or the `xtwarmswap --remove-cable` command, respectively. These options provide the ability to replace one or more cables without removing blades or shutting down the system. The routing of the Cray HSN will be updated to route around the removed cable or cables.

To add or remove a single HSN cable, specify one *`cable`* argument as in this `--add-cable` example:

```
# xtwarmswap --add-cable cable
```

To add or remove multiple HSN cables, specify a comma-separated list of cables, as in this `--remove-cable` example:

```
# xtwarmswap --remove-cable cable1,cable2,...,cableN
```

The `--add-cable` and `--remove-cable` options are not supported if more than a single active partition exists in the system. Do not specify the `-p`|`--partition` option when using these options. In addition, do not use the `--linktune` option when using the `--remove-cable` option.

Remove a Compute Blade from Service While the System is Running

About this task

A compute blade can be physically removed for maintenance or replacement while the system is running; however, the applications using the nodes on the blade to be removed must be allowed to drain, or be killed beforehand.

Before starting this warm swap procedure, verify that the proposed system configuration is routable. Doing this in advance of idling the nodes on the blades to be removed provides assurance that a valid set of nodes is being taken out of service before affecting the system. Log on to the SMW as `crayadm` and execute the following command, where `p`*`N`* is the partition from which the blades are being removed:

```
smw:~> rtr -S --id test --remove=blade_ID pN
```

CAUTION: This procedure warm swaps a compute blade from service while the system is running. Do not warm swap service blades, unless the blade is an I/O base blade (IBB) that has InfiniBand cards and is an LNET blade. Before attempting to warm swap any service blade, it is advisable to consult with a Cray service representative.

Procedure

1. Log on to the login node as `root`.
2. Ensure that the batch system or Slurm marks the blade as unavailable for scheduling.
3. Execute the following command to mark the nodes on the compute blade as `admindown`. This tells ALPS not to launch new applications onto them. (This command may also be executed from the boot node as user `root`.)

   ```
   login:~ # xtprocadmin -k s admindown -n blade_ID
   ```

   The arguments to the `-n` option should be the NID values for the nodes on the blade being removed, as shown by executing `xtprocadmin | grep` *bladename*.

   For example, to find the NID values for the nodes on the blade `c0-0c0s2` being removed:

   ```
   login:~ # xtprocadmin | grep c0-0c0s2
        8      0x8  c0-0c0s2n0  compute      up       batch
        9      0x9  c0-0c0s2n1  compute      up       batch
       10      0xa  c0-0c0s2n2  compute      up       batch
       11      0xb  c0-0c0s2n3  compute      up       batch
   ```

4. From the login node, execute the `apstat -n` command or the appropriate Slurm command to determine if any applications are running on the node marked `admindown`. This example shows that `apid 675722` is running on all nodes of blade `c0-0c0s2`.

   ```
   login:~ # apstat -n | egrep -w 'NID|8|9|10|11
     8   XT UP  B 32 32  1    4K 16777216 8388608  262144   1 675722
     9   XT UP  B 32 32  1    4K 16777216 8388608  262144   1 675722
    10   XT UP  B 32 32  1    4K 16777216 8388608  262144   1 675722
    11   XT UP  B 32 32  1    4K 16777216 8388608  262144   1 675722
   ```

5. Wait until the applications using the nodes on the blade finish or use the `apkill` *apid* command to kill the application.
6. Log on to the SMW as `crayadm`.
7. Execute the `xtcli halt` *blade_ID* command to halt the blade.

   ```
   smw:~> xtcli halt blade_ID
   ```

8. Execute the `xtwarmswap --remove` *blade_ID* command to remove the compute blade from service. The routing of the Cray HSN will be updated to route around the removed blade.

   The `--remove` stage of the `xtwarmswap` process uses the Aries™ resiliency infrastructure and takes about 30 seconds to complete.

```
smw:~> xtwarmswap --remove blade_ID
```

9. Execute the `xtcli power down blade_ID` command, which helps to identify which blade to pull (all lights are off on the blade).

```
smw:~> xtcli power down blade_ID
```

10. Physically remove the blade, if desired. To complete this step, see the hardware maintenance and replacement procedures documentation for the Cray system, or contact a Cray Service representative.

CAUTION: If a blade cannot be reinstalled in the empty slot within 2 minutes, install a filler blade assembly in the empty slot; failure to do so can cause other blades in the system to overheat.

Return a Compute Blade into Service

About this task

After a blade has been repaired or when a replacement blade is available, use the following procedure to return the blade into service.

Procedure

1. Physically insert the blade into the slot. To complete this step, see the hardware maintenance and replacement procedures documentation for the Cray system, or contact a Cray Service representative.

2. On the SMW, execute the `xtcli power up blade_ID` command.

```
smw:~> xtcli power up blade_ID
```

3. Ensure that the blade is ready by entering the following command, and wait until the command returns the correct response:

```
smw:~> xtalive blade_ID
The expected response was received.
```

4. Verify the status of the blade controller to ensure that its "Comp state" is "up" and that there are no flags set.

```
smw:~> xtcli status -t bc blade_ID
```

5. Bounce the blade.

```
smw:~> xtbounce blade_ID
```

6. If the blade or PDC type is different, `su` to `root`, execute the `xtdiscover` command, and then exit `root`. Otherwise, skip this step.

```
smw:~> su - root
smw:~> xtdiscover
smw:~> exit
smw:~>
```

7. Execute the `xtzap --blade` command to update the BC BIOS, node BIOS, microcontroller, and FPGAs as required.

```
smw:~> xtzap --blade blade_cname
```

8. Execute the `xtbounce --linkdown` *`blade_ID`* command to prepare the blade for the warm swap (takes down all HSN links on the blade).

   ```
   smw:~> xtbounce --linkdown blade_ID
   ```

9. Add the blade(s) to the HSN by executing the `xtwarmswap --add` *`blade_ID`*`,...` command. This command activates routing on the newly installed blade and automatically executes a mini-`xtdiscover` command once the warm swap steps have completed successfully. No additional manual invocation of `xtdiscover`, which gets the new hardware attributes from the added blades, is necessary.

   ```
   smw:~> xtwarmswap --add blade_ID
   ```

   Because the `xtwarmswap --add` command initializes the added blades, the time to return the blades back to service is about 10 minutes, including the time to initialize the blades, run the BIOS on the nodes, and initialize the links to the blades.

10. Boot the nodes on the blade(s) by executing the `xtcli boot CNL0` *`blade_ID`*`,...` command on the SMW.

    ```
    smw:~> xtcli boot CNL0 blade_ID
    ```

11. As `root` on the login node, execute the following command to mark the nodes on the compute blade as `up`. This tells ALPS that new applications may be launched onto those nodes. (This command may also be executed from the boot node as user `root`.)

    ```
    login:~ # xtprocadmin -k s up -n blade_ID
    ```

12. Verify that the blade is up.

    ```
    login:~ # xtprocadmin | grep blade_ID
    ```

13. Ensure that the batch system or Slurm marks the blade as available for scheduling.

State Manager LLM Logging

The log data from the State Manager is written to `/var/opt/cray/log/sm-`*`yyyymmdd`*. The default setting for the State Manager is to enable LLM logging. If LLM or craylog failures occur, State Manager logging is not disrupted. Logging then reverts to behavior that is very similar to legacy State Manager logging, which is also used when State Manager LLM logging is turned off.

To disable LLM logging for the State Manager, add the `-L n`option to the `/opt/cray/hss/default/bin/rsms` script entry:

```
sm=(/opt/cray/hss/default/bin/state_manager sm "-L n")
```

Boot Manager LLM Logging

The log data from the Boot Manager is written to `/var/opt/cray/log/bm-`*`yyyymmdd`*. If the `-L` command line option is used with the `bootmanager` command or if LLM is not enabled, Boot Manager reverts to legacy logging,

which writes log data to `/var/opt/cray/log/bm.out`. This is a less satisfactory logging method because each Boot Manager restart creates a new log and moves the previous log to `bm.out.1`. A third restart can possibly cause recent log data to be lost.

Configure Node Health Checker Tests

NHC is automatically invoked by ALPS upon the termination of an application. ALPS passes a list of CNL compute nodes associated with the terminated application to NHC. NHC performs specified tests to determine if compute nodes allocated to the application are healthy enough to support running subsequent applications. If not, it removes any compute nodes incapable of running an application from the resource pool. The CLE installation and upgrade processes automatically install and enable NHC software; there is no need to change any installation configuration parameters or issue any commands.

Use the `cray_node_health_worksheet.yaml` file or configurator to configure the NHC tests, which test CNL compute node functionality. All tests that are enabled will run when NHC is in either Normal Mode or in Suspect Mode. Tests run in parallel, independently of each other, except for the `Free Memory Check` test, which requires that the `Application Exited Check` test passes before the `Free Memory Check` test begins.

The `xtcheckhealth` binary runs the NHC tests; for information about the `xtcheckhealth` binary, see the `intro_NHC(8)` and `xtcheckhealth(8)` man pages.

The NHC tests are listed below. In the default NHC configuration file, each test that is enabled starts with an action of `admindown`, except for `Free Memory Check`, which starts with an action of `log`.

Also read important test usage information in *Guidance for the Accelerator Test* on page 286, *Guidance for the Application Exited Check and Apinit Ping Tests* on page 286, *Guidance for the Filesystem Test* on page 287, *Guidance for the Hugepages Test* on page 287, and *Guidance for the NHC Lustre File System Test* on page 288.

Accelerator	Tests the health of any accelerators present on the node. It is an application set test and should not be run in the reservation set. The global accelerator test (`gat`) script detects the type of accelerator(s) present on the node and then launches a test specific to the accelerator type. The test fails if it is unable to run successfully on the accelerator, or if the amount of allocated memory on the accelerator exceeds the amount specified using the `gat -m` argument. Default: enabled
Application Exited Check	Verifies that any remaining processes from the most recent application have terminated. It is an application set test and should not be run in the reservation set because an application is not associated with a reservation cancellation. The `Application Exited Check` test checks locally on the compute node for processes running under the ID of the application (APID). If processes are running, NHC waits a period of time (defined in the configuration file) to determine if the application processes exit properly. If the process does not exit within that time, this test fails. Default: enabled
Apinit Log and Core File Recovery	A plugin script to copy `apinit` core dump and log files to a login/service node. It is an application set test.

	Default: not enabled. `Apinit Log and Core File Recovery` should not be enabled until a destination directory is determined and specified in the NHC configuration file.
`Apinit Ping`	Verifies that the ALPS daemon is running on the compute node and is responsive. It is an application set test. The `Apinit Ping` test queries the status of the `apinit` daemon locally on each compute node; if the `apinit` daemon does not respond to the query, then this test fails. Default: enabled
`DataWarp`	A plugin script to check that any reservation-affiliated DataWarp mount points have been removed. Note that the plugin can only detect a problem after the last reservation on a node completes. Default: disabled
`Free Memory Check`	Examines how much memory is consumed on a compute node while applications are not running. Use it only as a reservation test because an application within a reservation may leave data for another application in a reservation. If run in the application set, `Free Memory Check` could consider data that was intentionally left for the next application to be leaked memory and mark the node `admindown`. Run the `Free Memory Check` only after the `Reservation` test passes successfully. Default: enabled (action is `log` only)
`Filesystem`	Ensures that the compute node is able to perform simple I/O to the specified file system. It is configured as an application set test in the default configuration, but it can be run in the reservation set. For a file system that is mounted read-write, the test performs a series of operations on the file system to verify the I/O. A file is created, written, flushed, synced, and deleted. If a mount point is not explicitly specified, the mount point(s) from the compute node `/etc/fstabs` file will be used and a `Filesystem` test will be created for each mount point found in the file. If a mount point is explicitly specified, then only that file system will be checked. An administrator can specify multiple `FileSystem` tests by placing multiple `Filesystem` lines in the configuration file. For example, one line could specify the implicit `Filesystem` test, and the next line could specify a specific file system that does not appear in `/etc/fstab`. This could continue for any and all file systems. When enabling the `Filesystem` test, an administrator can exclude mount points that should not be tested using the `excluding` setting in the configuration to list mount points that should not be tested by the `Filesystem` test. This allows intentionally excluding specific mount points even though they appear in the `fstab` file. This action prevents NHC from setting nodes to `admindown` because of errors on relatively benign file systems. Explicitly specified mount points cannot be excluded in this fashion; if they should not be checked, then they should simply not be specified. The `Filesystem` test creates its temporary files in a subdirectory (`.nodehealth.fstest`) of the file system root. An error message is written to the console when the `unlink` of a file created by this test fails. Default: enabled
`Hugepages`	Calculates the amount of memory available in a specified page size with respect to a percentage of `/proc/boot_freemem`. It is a reservation set test.

	This test will continue to check until either the memory clears up or the time-out is reached. The default time-out is 300 seconds. Default: disabled
Sigcont Plugin	Sends a SIGCONT signal to the processes of the current APID. It is an application set test. Default: disabled
Plugin	Allows scripts and executables not built into NHC to be run, provided they are accessible on the compute node. . Default: disabled so that local configuration settings may be used
ugni_nhc_plugins	Tests the User level Gemini Network Interface (uGNI) on compute nodes. It is a reservation set test and an application set test. By extension, testing the uGNI interface also tests the proper operation of parts of the network interface card (NIC). The test sends a datagram packet out to the node's NIC and back again.
Reservation	checks for the existence of the `/proc/reservations/`*`rid`* directory, where *`rid`* is the reservation ID. It is a reservation set test, and should not be run in the application set. If this directory still exists, the test will attempt to end the reservation and then wait for the specified *`timeout`* value for the directory to disappear. If the test fails and Suspect Mode is enabled, NHC enters Suspect Mode. In Suspect Mode, `Reservation` continues running, repeatedly requesting that the kernel clean up the reservation, until the test passes or until Suspect Mode times out. If the directory does not disappear in that time, the test prints information to the console and exits with a failure. Default: enabled with a *`timeout`* value of 300 seconds
CCM plugin	validates the cleanup of a cluster compatibility mode (CCM) environment at the end of a reservation. It is a reservation set test, and it will not run if it is misconfigured as an application test. This test runs on a compute node only when `/var/crayccm` is detected. The test removes the `/var/lib/{empty,debus}` directories, unmounts CCM mount points if they still exist, and unmounts `/dsl/dev/random` and `/dsl/dev/pts`. If the unmounts are successful, the test removes the `/var/crayccm`, `/var/lib/rpcbind`, and `/var/spool/{PBS,torque}` directories. The `CCM plugin` is not included in a site's NHC configuration file. Administrators must add the test to their configuration in order to use it. See the `cray_node_health_worksheet.yaml` file for `CCM plugin` settings to copy into a site's NHC configuration file.

Individual tests may appear multiple times in the configuration, with different variable values. Every time a test is specified, NHC will run that test. This means if the same line is specified five times, NHC will try to run that same test five times. This functionality is mainly used in the case of the `Plugin` test, allowing the administrator to specify as many additional tests as have been written for the site, or the `Filesystem` test, allowing the adminstrator to specify as many additional file systems as wanted. However, any test can be specified to run any number of times. Different parameters and test actions can be set for each test. For example, this could be used to set up hard limits and soft limits for the `Free Memory Check` test. Two `Free Memory Check` tests could be specified in the configuration file; the first test configured to only warn about small amounts of non-free memory,

and the second test configured to `admindown` a node that has large amounts of non-free memory. See the `cray_node_health_worksheet.yaml` file for configuration information.

Guidance for the Accelerator Test

This test uses the global accelerator test (`gat`) script (`/opt/cray/nodehealth/default/bin/gat.sh`) to first detect the accelerator type and then launch the test specific to that type of accelerator.

The `gat` script supports two arguments for NVIDIA GPUs:

-m*maximum_memory_size*	Specify the *maximum_memory_size* as either a kilobyte value or a percentage of total memory. For example, `-m 100` specifies that no more than 100 kilobytes of memory can be allocated, while `-m 10%` specifies that no more than 10 percent of memory can be allocated. In the default NHC configuration file, the specified memory size is `10%`.
-r	Perform a soft restart on the GPU and then rerun the test. In the default NHC configuration file, the `-r` argument is specified.

The `gat` script has the following options for Intel Xeon Phi:

-M *kilobytes* or **-M *n*%**	This option works exactly as the `-m` option for the NVIDIA GPUs.
-c	Specifies the minimum number of cores that must be active on the Xeon Phi for the test to pass. If `-c` is omitted, the minimum number of active cores required to pass the test is the total number of cores on the Xeon Phi.

Guidance for the Application Exited Check and Apinit Ping Tests

These two tests must be enabled and both tests must have their action set as `admindown` or `die`; otherwise, NHC runs the risk of allowing ALPS to enter a live-lock. Only specify the `die` action when the `advanced_features` control variable is turned off.

ALPS must guarantee two conditions about the nodes in a reservation before releasing that reservation:

- that ALPS is functioning on the nodes
- that the previous application has exited from the nodes

Either those two conditions are guaranteed or the nodes must be set to some state other than `up`. When either ALPS has guaranteed these two conditions about the nodes or the nodes have been set to some state other than `up`, then ALPS can release the reservation.

These NHC tests guarantee two conditions:

- `Apinit_ping` guarantees that ALPS is functioning on the nodes
- `Application_Exited_Check` guarantees that the previous application has exited from the nodes

If either test fails, then NHC sets the nodes to `suspect` state if Suspect Mode is enabled; otherwise, NHC sets the nodes to `admindown` or `unavail`. Therefore, either the nodes pass these tests or the nodes are no longer in the `up` state. In either case, ALPS is free to release the reservation and the live-lock is avoided. Note that this only happens if the two tests are enabled and their action is set as `admindown` or `die`. The `log` action does not suffice because it does not change the state of the nodes. If either test is disabled or has an action of `log`, then ALPS may live-lock. In this live-lock, ALPS will call NHC endlessly.

Guidance for the Filesystem Test

The NHC `Filesystem` test can take an explicit argument (the mount point of the file system) or no argument. If an argument is provided, the `Filesystem` test is referred to as an explicit `Filesystem` test. If no argument is given, the `Filesystem` test is referred to as an implicit `Filesystem` test.

The explicit `Filesystem` test will test the file system located at the specified mount point.

The implicit `Filesystem` test will test each file system listed in the `/etc/fstab` file on each compute node. The implicit `Filesystem` test is enabled by default in the NHC configuration file.

The `Filesystem` test will determine whether a file system is mounted read-only or read-write. If the file system is mounted read-write, then NHC will attempt to write to it. If it is mounted read-only, then NHC will attempt to read the directory entities "." and ".." in the file system to guarantee, at a minimum, that the file system is readable.

Some file systems are mounted on the compute nodes as read-write file systems, while their underlying permissions are read-only. As an example, for an auto-mounted file system, the base mount-point may have read-only permissions; however, it could be mounted as read-write. It would be mounted as read-write, so that the auto-mounted sub-mount-points could be mounted as read-write. The read-only permissions prevent tampering with the base mount-point. In a case such as this, the `Filesystem` test would see that the base mount-point had been mounted as a read-write file system. The `Filesystem` test would try to write to this file system, but the write would fail due to the read-only permissions. Because the write fails, the `Filesystem` test would fail, and NHC would incorrectly decide that the compute node is unhealthy because it could not write to this file system. For this reason, file systems that are mounted on compute nodes as read-write file systems, but are in reality read-only file systems, should be excluded from the implicit `Filesystem` test.

The administrator can exclude tests by adding an "Excluding: *`file system mount point`*" entry in the NHC configuration file. See the NHC configuration file for further details and an example.

A file system is deemed a critical file system if it is needed to run applications. All systems will likely need at least one shared file system for reading and writing input and output data. Such a file system would be a critical file system. File systems that are not needed to run applications or read and write data would be deemed as noncritical file systems. The administrator must determine the criticality of each file system.

Cray recommends the following:

- Exclude noncritical file systems from the implicit `Filesystem` test. See the NHC configuration file for further details and an example.
- If there are critical file systems that do not appear in the `/etc/fstab` file on the compute nodes (such file systems would not be tested by the implicit `Filesystem` test), these critical file systems should be checked through explicit `Filesystem` tests. Add explicit `Filesystem` tests to the NHC configuration file by providing the mount point of the file system as the final argument to the `Filesystem` test. See the NHC configuration file for further details and an example.
- If a file system that is mounted as read-write but it has read-only permissions, exclude it from the implicit `Filesystem` test. NHC does not support such file systems.
- Client mounts may fail as a system is booting because not all routes have had sufficient time to be established. The retry ensures a mount attempt will be made after all routes are up.

Guidance for the Hugepages Test

The `Hugepages` test runs the `hugepages_check` command, which supports two arguments:

`-t` *`threshold`* Use this argument to specify the *`threshold`* as a percentage of `/proc/boot_freemem`. If this test is enabled and this argument is not supplied, the default of `-t 90` is used.

-s *size* Specify the hugepage size. The valid sizes are `2`, `4`, `8`, `16`, `32`, `64`, `128`, `256`, and `512`. If this test is enabled and this argument is not supplied, the default of `-s 2` is used.

Guidance for the NHC Lustre File System Test

The Lustre file system has its own hard time-out value that determines the maximum time that a Lustre recovery will last. This time-out value is called `RECOVERY_TIME_HARD`, and it is located in the file system's `fs_defs` file. The default value for the `RECOVERY_TIME_HARD` is 15 minutes.

> **IMPORTANT:** The time-out value for the NHC `Lustre` file system test should be twice the `RECOVERY_TIME_HARD` value.

The default in the NHC configuration file is 30 minutes, which is twice the default `RECOVERY_TIME_HARD`. If the value of `RECOVERY_TIME_HARD` is changed, the time-out value of the NHC `Lustre` file system test must also change correspondingly.

The NHC time-out value is specified on the following line in the NHC configuration file:

```
# Lustre: <warning time-out> <test time-out> <restart delay>
Lustre: 900 1800 60
```

Further, the overall time-out value of NHC's Suspect Mode is based on the maximum time-out value for all of the NHC tests. Invariably, the NHC `Lustre` file system test has the longest time-out value of all the NHC tests.

> **IMPORTANT:** If the NHC `Lustre` file system test time-out value is changed, then the time-out value for Suspect Mode must also be changed. The time-out value for Suspect Mode is set by the `suspectend` variable in the NHC configuration file. The guidance for setting the value of `suspectend` is that it should be the maximum time-out value, plus an additional buffer. In the default case, `suspectend` was set to 35 minutes -- 30 minutes for the Lustre test, plus an additional 5 minute buffer. For more information about the `suspectend` variable, see *NHC Suspect Mode*.

NHC Control Variables

The following variables in `/etc/opt/cray/nodehealth/nodehealth.conf` affect the fundamental behavior of NHC.

advanced_features If set to `on`, this variable allows multiple instances of NHC to run simultaneously. This variable must be `on` to use CNCU and reservation sets.

Default: `on`

dumpdon If set to `off`, NHC will not request any dumps or reboots from `dumpd`. This is a quick way to turn off dump and reboot requests from NHC. The `dump`, `reboot`, and `dumpreboot` actions do not function properly when this variable is `off`.

Default: `on`

anyapid Turning `anyapid` on specifies that NHC should look for any `apid` in `/dev/cpuset` while running the `Application Exited Check` and print stack traces for processes that are found.

Default: `off`

Global Configuration Variables that Affect all NHC Tests

The following global configuration variables may be set in the `/etc/opt/cray/nodehealth/nodehealth.conf` file to alter the behavior of all NHC tests. The global configuration variables are case-insensitive.

`Runtests: Frequency`
Determines how frequently NHC tests are run on the compute nodes. *`Frequency`* may be either `errors` or `always`. When the value `errors` is specified, the NHC tests are run only when an application terminates with a non-zero error code or terminates abnormally. When the value `always` is specified, the NHC tests are run after every application termination. If the `Runtests` global variable is not specified, the implicit default is `errors`.

This variable applies only to tests in the application set; reservations do not terminate abnormally.

`Connecttime: TimeoutSeconds`
Specifies the amount of time, in seconds, that NHC waits for a node to respond to requests for the TCP connection to be established. If Suspect Mode is disabled and a particular node does not respond after `connecttime` has elapsed, then the node is marked `admindown`. If Suspect Mode is enabled and a particular node does not respond after `connecttime` has elapsed, then the node is marked `suspect`. NHC will then attempt to contact the node with a frequency established by the `recheckfreq` variable.

If the `Connecttime` global variable is not specified, then the implicit default TCP time-out value is used. NHC will not enforce time-out on the connections if none is specified. The `Connecttime`: *`TimeoutSeconds`* value provided in the default NHC configuration file is `60` seconds.

The following global variables control the interaction of NHC and `dumpd`, the SMW daemon that initiates automatic dump and reboot of nodes.

`maxdumps: MaximumNodes`
Specifies the number of nodes that fail with the `dump` or `dumpreboot` action that will be dumped. For example, if NHC was checking on 10 nodes that all failed tests with the `dump` or `dumpreboot` actions, only the number of nodes specified by `maxdumps` would be dumped, instead of all of them. The default value is `1`.

To disable dumps of failed nodes with `dump` or `dumpreboot` actions, set `maxdumps: 0`.

`downaction: action`
Specifies the action NHC takes when it encounters a down node. Valid actions are `log`, `dump`, `reboot`, and `dumpreboot`. The default action is `log`.

`downdumps: number_dumps`
Specifies the maximum number of dumps that NHC will dump for a given APID, assuming that the `downaction` variable is either `dump` or `dumpreboot`. These dumps are in addition to any dumps that occur because of NHC test failures. The default value is `1`.

The following global variables control the interaction between NHC, ALPS, and the SDB.

`alps_recheck_max: number of seconds`
NHC will attempt to verify its view of the states of the nodes with the ALPS view. If NHC is unable to contact ALPS, this variable controls the maximum delay between rechecks.

Default value: 10 seconds

`alps_sync_timeout: number of seconds`
If NHC is unable to contact ALPS to verify the states of the nodes, this variable controls the length of time before NHC gives up and aborts.

Default value: 1200 seconds

alps_warn_time: ***number of seconds***
If NHC is unable to contact ALPS to verify the states of the nodes, this variable controls how often warnings are issued.

Default value: 120 seconds

sdb_recheck_max: ***number of seconds***
NHC will contact the SDB to query for the states of the nodes. If NHC is unable to contact the SDB, this variable controls the maximum delay between rechecks.

Default value: 10 seconds

sdb_warn_time: ***number of seconds***
If NHC is unable to contact the SDB, this variable controls how often warnings are issued.

Default value: 120 seconds

node_no_contact_warn_time: ***number of seconds***
If NHC is unable to contact a specific node, this variable controls how often warnings are issued.

Default value: 600 seconds

The following global variable controls NHC's use of node states.

unhealthy_state: ***swdown***
When a node is deemed unhealthy, it is normally set to `admindown`. This variable permits a different state to be chosen instead.

Default: not set

unhealthy_state: ***rebootq***
When a node is going to be rebooted, it is normally set to `Unavail`. This variable permits a different state to be chosen instead.

Default: not set

Standard Variables that Affect Individual NHC Tests

The following variables are used with each NHC test; set each variable for each test. All variables are case-insensitive. Each NHC test has values supplied for these variables in the default NHC configuration file.

Specific NHC tests require additional variables, which are defined in the `nodehealth` configuration file.

action Specifies the action to perform if the compute node fails the given NHC test. *`action`* may have one of the following values:

`log` Logs the failure to the system console log. The `log` action will not cause a compute node's state to be set to `admindown`.

IMPORTANT: Tests that have an action of `Log` do not run in Suspect Mode. If using plugin scripts with an action of `Log`, the script will only be run once, in Normal Mode. This makes log collecting and various other maintenance tasks easier to code.

`admindown` Sets the compute node's state to `admindown` (no more applications will be scheduled on that node) and logs the failure to the system console log.

If Suspect Mode is enabled, the node will first be set to `suspect` state, and if the test continues to fail, the node will be set to `admindown` at the end of Suspect Mode.

`die` Halts the compute node so that no processes can run on it, sets the compute node's state to `admindown`, and logs the failure to the system console log. (The `die` action is the equivalent of a kernel panic.) This action is good for catching bugs because the state of the processes is preserved and can be dumped at a later time.

If the `advanced_features` variable is enabled, `die` is not allowed.

Each subsequent action includes the actions that preceded it; for example, the `die` action encompasses the `admindown` and `log` actions.

If NHC is running in Normal Mode and cannot contact a compute node, and if Suspect Mode is not enabled, NHC will set the compute node's state to `admindown`.

The following actions control the NHC and `dumpd` interaction.

`dump` Sets the compute node's state to `admindown` and requests a dump from the SMW, in accordance with the `maxdumps` configuration variable.

`reboot` Sets the compute node's state to `unavail` and requests a reboot from the SMW. The `unavail` state is used rather than the `admindown` state when nodes are to be rebooted because a node that is set to `admindown` and subsequently rebooted stays in the `admindown` state. The `unavail` state does not have this limitation.

`dumpreboot` Sets the compute node's state to `unavail` and requests a dump and reboot from the SMW.

The following actions control the NHC and `dumpd` interaction.

`warntime` Specifies the amount of elapsed test time, in seconds, before `xtcheckhealth` logs a warning message to the console file. This allows an administrator to take corrective action, if necessary, before the *`timeout`* is reached.

`timeout` Specifies the total time, in seconds, that a test should run before an error is returned by `xtcheckhealth` and the specified *`action`* is taken.

`restartdelay` Valid only when NHC is running in Suspect Mode. Specifies how long NHC will wait, in seconds, to restart the test after the test fails. The minimum restart delay is one second.

`sets` Indicates when to run a test. The default NHC configuration specifies to run specific tests after application completion and to run an alternate group of tests at reservation end. When ALPS calls NHC at the end of the application, tests marked with `Sets: Application` are run. By default, these tests are: `Filesystem`, `Accelerator`, `ugni_nhc_plugins`, `Application Exited Check`, `Apinit Ping Test`, and `Apinit Log and Core File Recovery`. At the end of the reservation, ALPS calls tests marked `Sets: Reservation`. By default, these are: `Free Memory Check`, `ugni_nhc_plugins`, `Reservation`, and `Hugepages Check`.

If no set is specified for a test, it will default to `Application`, and run when ALPS calls NHC at the end of the application. If NHC is launched manually, using the `xtcheckhealth`

command, and the `-m sets` argument is not specified on the command line, then `xtcheckhealth` defaults to running the `Application` set.

If a test is marked `Sets: All`, it will always run, regardless of how NHC is invoked.

NHC Suspect Mode

Upon entry into Suspect Mode, NHC immediately allows healthy nodes to be returned to the resource pool. Suspect Mode allows the remaining nodes, which are all in `suspect` state, an opportunity to return to healthiness. If the nodes do not return to healthiness by the end of the Suspect Mode (determined by the `suspectend` global variable; see below), their states are set to `admindown`. For more information about how Suspect Mode functions, see the `intro_NHC(8)` man page.

IMPORTANT: Suspect Mode is enabled in the default configuration. Cray recommends that sites run NHC with Suspect Mode enabled.

If enabled, the default NHC configuration file uses the following Suspect Mode variables:

`suspectenable:` Enables Suspect Mode; valid values are `y` and `n`.

Default: `y`

`suspectbegin:` Sets the Suspect Mode timer. Suspect Mode starts after the number of seconds indicated by `suspectbegin` have expired.

Default: `180`

`suspectend:` Suspect Mode ends after the number of seconds indicated by `suspectend` have expired. This timer only starts after NHC has entered Suspect Mode.

Default: `2100`

Considerations when evaluating shortening the length of Suspect Mode:

- The length of Suspect Mode can be shortened if there are no external file systems, such as Lustre, for NHC to check.
- Cray recommends that the length of Suspect Mode be at least a few seconds longer than the longest time-out value for any of the NHC tests. For example, if the `Filesystem` test had the longest time-out value at 900 seconds, then the length of Suspect Mode should be at least 905 seconds.
- The longer the Suspect Mode, the longer nodes have to recover from any unhealthy situations. Setting the length of Suspect Mode too short reduces this recovery time and increases the likelihood of the nodes being marked `admindown` prematurely.

`recheckfreq:` Suspect Mode rechecks the health of the nodes in `suspect` state at a frequency specified by `recheckfreq`. This value is in seconds.

For a detailed description about NHC actions during the recheck process, see the `intro_NHC(8)` man page.

Default: `300`

NHC Messages

NHC messages may be found on the SMW in `/var/opt/cray/log/sessionid/nhc-YYYYMMDD` with `'<node_health:M.m>'` in the message, where *M* is the major and *m* is the minor NHC revision number. All NHC messages are visible in the console file.

NHC prints a summary message per node at the end of Normal Mode and Suspect Mode when at least one test has failed on a node. For example:

```
<node_health:3.1> APID:100 (xtnhc) FAILURES: The following tests have failed in
normal mode:
<node_health:3.1> APID:100 (xtnhc) FAILURES: (Admindown) Apinit_Ping
<node_health:3.1> APID:100 (xtnhc) FAILURES: (Admindown) Plugin /example/plugin
<node_health:3.1> APID:100 (xtnhc) FAILURES: (Log Only ) Filesystem_Test on /
mydir
<node_health:3.1> APID:100 (xtnhc) FAILURES: (Admindown) Free_Memory_Check
<node_health:3.1> APID:100 (xtnhc) FAILURES: End of list of 5 failed test(s)
```

The `xtcheckhealth` error and warning messages include node IDs and application IDs and are written to the console file on the SMW; for example:

```
[2010-04-05 23:07:09][c1-0c2s0n0]<node_health:3.0> APID:2773749
(check_apid) WARNING: Failure: File /dev/cpuset/2773749/tasks exists and is not
empty. \
The following processes are running under expired APID
2773749:
[2010-04-05 23:07:09][c1-0c2s0n1]<node_health:3.0> APID:2773749
(check_apid) WARNING: Pid:   300 Name:           (marys_program) State: D
```

The `xtcleanup_after` script writes its normal launch information to the `/var/log/xtcheckhealth_log` file, which resides on the login nodes. The `xtcleanup_after` launch information includes the time that `xtcleanup_after` was launched and the time `xtcleanup_after` called `xtcheckhealth`.

The `xtcleanup_after` script writes error output (launch failure information) to the `/var/log/xtcheckhealth_log` file, to the console file on the SMW, and to the syslog.

Example `xtcleanup_after` output follows:

```
Thu Apr 22 17:48:18 CDT 2010 <node_health> (xtcleanup_after)
/opt/cray/nodehealth/3.0-1.0000.20840.30.8.ss/bin/xtcheckhealth -a 10515
-e 1 /tmp/apsysLVNqO9 /etc/opt/cray/nodehealth/nodehealth.conf
```

Recover from a Login Node Crash when a Login Node will not be Rebooted

About this task

If a login node crashes while `xtcheckhealth` binaries on that login node are monitoring compute nodes that are in `suspect` state, those `xtcheckhealth` binaries will die when the login node crashes. When the login node that crashed is rebooted, a recovery action takes place. When the login node boots, the `node_health_recovery` binary starts up. This script checks for all compute nodes that are in `suspect` state and were last set to `suspect` state by this login node. The script then determines the APID of the application that was running on each of these compute nodes at the time of the crash. The script then launches an `xtcheckhealth` binary to monitor each of these compute nodes. One `xtcheckhealth` binary is launched per compute node monitored.

If the `Application_Exited_Check` test is enabled in the configuration file (default), `xtcheckhealth` is launched with this APID to test for processes that may have been left behind by that application. Otherwise, NHC does not run the `Application_Exited_Check` test and will not check for leftover processes, but will run any other NHC tests that are enabled in the configuration file.

Nodes are changed from `suspect` state to `up` or `admindown`, depending upon whether they fail any health checks. No system administrator intervention should be necessary.

NHC automatically recovers the nodes in `suspect` state when the crashed login node is rebooted because the recovery feature runs on the rebooted login node. If the crashed login node is not rebooted, then manual intervention is required to rescue the nodes from `suspect` state. This manual recovery can commence as soon as the login node has crashed. To recover from a login node crash during the case in which a login node will not be rebooted, the `nhc_recovery` binary is provided to help release the compute nodes owned by the crashed login node; see *Recover from a Login Node Crash when a Login Node will not be Rebooted*. Also, see the `nhc_recovery(8)` man page for a description of the `nhc_recovery` binary usage.

Procedure

1. Create a file, *`nodelistfile`*, that contains a list of the nodes in the system that are currently in Suspect Mode. The file must be a list of NIDs, one per line; do not include a blank line at the end of the file.

2. List all of the `suspect` nodes in the system and the login nodes to which they belong.

   ```
   smw:~# nhc_recovery -d nodelistfile
   ```

3. Parse the `nhc_recovery` output for the NID of the login node that crashed, creating a file, *`computenodes`*, that contains all of the compute nodes owned by the crashed login node.

4. Use the *`computenodes`* file to create *`nodelist`* files containing nodes that share the same APID (to determine the nodes from the crashed login node). For example, the files can be named `nodelistfile-APID1`, `nodelistfile-APID2`, `nodelistfile-APID3`, etc.

5. Release all of the `suspect` compute nodes owned by the crashed login node.

   ```
   smw:~# nhc_recovery -r computenodes
   ```

   All of these compute nodes are released in the database, but they are all still in `suspect` state.

6. Determine what to do with these `suspect` nodes from the following three options:

   - (Cray recommends this option) Rerun NHC on a non-crashed login node to recover the nodes listed in step *4* on page 294. Invoke NHC for each `nodelist-APID` file. Supply the APID that corresponds to the *`nodelistfile`*; an iteration count of `0` (zero), which is the value normally supplied to NHC by ALPS; and an application exit code of `1` as the APID argument. An exit code of `1` ensures that NHC will run regardless of the value of the `runtests` variable (`always` or `errors`) in the NHC configuration file. For example:

     ```
     smw:~# xtcleanup_after -s nodelist-APID1  APID1  0  1
     smw:~# xtcleanup_after -s nodelist-APID2  APID2  0  1
     smw:~# xtcleanup_after -s nodelist-APID3  APID3  0  1
     .
     .
     .
     ```

- Set the `suspect` nodes to `admindown` and determine their fate by further analysis.
- Set the `suspect` nodes back to `up`, keeping in mind that they were in Suspect Mode for a reason.