Hewlett Packard
Enterprise

HPE Cray EX System Installation and Configuration Guide

(1.4.2)

S-8000

Contents

1 About the HPE Cray EX Installation and Configuration Guide

Scope and Audience

The *HPE Cray EX Installation and Configuration Guide* contains procedures for installing the Cray Operating System (COS), Cray System Management (CSM), and other product streams that comprise the EX software. Following these installation procedures, the "Procedures that Support Installation" section includes troubleshooting procedures, and the "Slingshot Post-Installation Tasks" section includes procedures for common administrative tasks relevant to the Slingshot high speed network.

This publication is intended for system installers, system administrators, and network administrators of the system. It assumes some familiarity with standard Linux and open source tools, such as yum, Ansible, and YAML/JSON, etc.

Hostnames in Command Output

The output for commands in this document may contain incorrect hostnames. The output on a system will not include references to *sms*.

Record of Revision

New in Revision A

- Changes to the *Apply the CSM Patch* on page 32 procedure
- Procedural change in *CSM Deploy NCNs* on page 46 to avoid potential network degradation
- Added a statement that the first attempt to reboot the compute nodes is likely to fail in *Install and Configure the Cray Operating System (COS)* on page 87
- Correction of patch file name in *Apply the UAN Patch* on page 44

Major updates

- This release supports SLES 15 SP2 for management nodes and SLE 15 SP1 for compute and application nodes. Installation procedures have been updated accordingly.
- **Updates to procedures** - The heading '**NEW IN RELEASE v1.4**' is used to identify new procedures, as well as updates to existing procedures.
- New section **1.4.1 Patch Upgrade**.
- Minor bug fixes.

Table 1. Record of Revision

Publication Title	Date	Release
HPE Cray EX Installation and Configuration Guide S-8000 (v1.4 Rev B)	March 2021	v1.4 Rev B
HPE Cray EX Installation and Configuration Guide S-8000 (v1.4 Rev A)	March 2021	v1.4 Rev A
HPE Cray EX Installation and Configuration Guide S-8000 (v1.4)	March 2021	v1.4
HPE Cray EX Installation and Configuration Guide S-8000 (v1.4.1)	April 2021	V1.4.1
HPE Cray EX Installation and Configuration Guide S-8000 (v1.4.1)	May 2021	V1.4.1 Rev A

Typographic Conventions

`Monospace`	Indicates program code, reserved words, library functions, command-line prompts, screen output, file/path names, 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 **GUI Window**, **GUI element**, cascading menu (**Ctrl**→**Alt**→**Delete**), or key strokes (press **Enter**).
\ (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).

Trademarks

1.1 General Guidelines for Using the Shasta Installation and Configuration Guide

Copy-Paste Commands from this Guide

To ensure that commands are copied and pasted correctly while performing the procedures in this guide:

1. Copy a command from the PDF.
2. Paste it to a neutral editing form.
3. Copy the command from the neutral form and paste it into the console.

If entering commands manually, double-check them for correctness as some commands may not render correctly in the PDF.

Update Ansible Files

Some installation instructions require updating Ansible files. These files should be updated with great caution. The syntax of Ansible files does not support using tabs for editing. Only spaces are supported. These files should be updated with great caution. Refer to online documentation to learn more about Ansible syntax.

1.2 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`.

Node abbreviations

The following list contains abbreviations for nodes used below.

- CN - Compute Node
- NCN - Non Compute Node
- AN - Application Node (special type of NCN)
- UAN - User Access Node (special type of AN)
- PIT- Pre-install Toolkit (initial node used as the inception node during software installation booted from the LiveCD)

`ncn#`	Run the command as root on any NCN, except an NCN which is functioning as an Application Node (AN) such as a UAN.
`ncn-m#`	Run the command as `root` on any NCN-M (NCN which is a Kubernetes master node).
`ncn-m002#`	Run the command as `root` on the specific NCN-M (NCN which is a Kubernetes master node) which has this hostname (ncn-m002).
`ncn-w#`	Run the command as `root` on any NCN-W (NCN which is a Kubernetes worker node).
`ncn-w001#`	Run the command as `root` on the specific NCN-W (NCN which is a Kubernetes worker node) which has this hostname (ncn-w001).
`ncn-s#`	Run the command as `root` on any NCN-S (NCN which is a Utility Storage node).
`ncn-s003#`	Run the command as `root` on the specific NCN-S (NCN which is a Utility Storage node) which has this hostname (ncn-s003).
`pit#`	Run the command as `root` on the PIT node.

`uan#`	Run the command as `root` on any UAN.
`uan01#`	Run the command on a specific UAN.
`user@uan>`	Run the command as any non-`root` user on any UAN.
`cn#`	Run the command as `root` on any CN. Note that a CN will have a hostname of the form nid123456, that is “nid” and a six digit, zero padded number.
`hostname#`	Run the command as `root` on the specified hostname.
`user@hostname>`	Run the command as any non-`root` user on the specified hostname.

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.

```
hostname# chroot /path/to/chroot
chroot-hostname#
```

Command Prompt Inside Kubernetes Pod

If executing a shell inside a container of a Kubernetes pod where the pod name is $podName, the prompt changes to indicate that it is inside the pod. Not all shells are available within every pod, this is an example using a commonly available shell.

```
ncn# kubectl exec -it $podName /bin/sh
pod#
```

Command Prompt Inside Image Customization Session

If using ssh during an image customization session, the prompt changes to indicate that it is inside the image customization environment (pod). This example uses $PORT and $HOST as environment variables with specific settings. When using chroot in this context the prompt will be different than the above chroot example

```
hostname# ssh -p $PORT root@$HOST
root@POD# chroot /mnt/image/image-root
:/#
```

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.

Example of prompts as they appear on the system:

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

Examples of prompts as they appear in this publication:

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

Command Prompts for Network Switch Configuration

The prompts when doing network switch configuration can vary widely depending on which vendor switch is being configured and the context of the item being configured on that switch. There may be two levels of user privilege which have different commands available and a special command to enter configuration mode.

Example of prompts as they appear in this publication:

Enter "setup" mode for the switch make and model, for example:

```
remote# ssh sw-leaf-001
sw-leaf-001>  enable
sw-leaf-001#  configure terminal
sw-leaf-001(conf)#
```

Refer to the switch vender OEM documentation for more information about configuring a specific switch.

2 Related Documentation and Software Versions

Many of the components of the system are from the open-source community and are well documented by members of that community. The following table lists some of the more prominent third-party components, the version used in the system, and where to find relevant documentation. For links that point to code, scroll down to the README to find the referenced documentation.

Table 2. Third-Party Software Documentation

Third-Party Software	Version	Documentation Source
Apache® Spark™	3.0.0-SNAPSHOT	*https://spark.apache.org/*
Ansible	2.8.1	*https://docs.ansible.com/*
Apache Kafka	2.2.1 and 2.3.1	*https://kafka.apache.org/*
Ceph	14.2.11	*http://docs.ceph.com/docs/nautilus/*
CoreDNS	1.6.5	*https://coredns.io/*
DHCP	4.4.1	*https://kb.isc.org/docs/aa-00333*
Docker	19.03.5	*https://www.docker.com/*
Docker Registry	2.6.2	
etcd	3.4.3 is the etcd version running on the Kubernetes master nodes. 3.3.8 is the etcd version running on the pods for the etcd clusters.	General documentation. *https://github.com/etcd-io/etcd*
		Documentation about `etcdctl`, the `etcd` command line client, which is useful for troubleshooting. Scroll down to README. *https://github.com/etcd-io/etcd/tree/master/etcdctl*
		Documentation about the particular version used in the system. *https://github.com/etcd-io/etcd/releases/tag/v3.4.3*
Grafana	7.0.3	*https://grafana.com/*
Helm/Tiller	0.1.0-2-bd9d893	
iPXE	1.0	*https://ipxe.org/docs*
Istio	1.6.13	*https://istio.io/v1.6/docs/*
Kiali	1.18	
Kea	1.7.10	*https://www.isc.org/kea/*

Third-Party Software	Version	Documentation Source
Keepalived	0.11	
Keycloak	9.0.0	*https://www.keycloak.org/documentation.html*
Kibana	7.3.2	*https://www.elastic.co/products/kibana*
Kubernetes	1.17.2	*https://kubernetes.io/docs/concepts/overview/components/*
LDMS	4.2.0	*https://github.com/ovis-hpc/ovis-wiki/wiki*
Macvlan	0.3.0	*https://docs.docker.com/network/macvlan*
MariaDB	10.3.15	*https://mariadb.com/kb/en/*
MetalLB	0.8.1	*https://metallb.universe.tf/*
Multus	3.1	*https://kubernetes.io/docs*
PicOS (on Edgecore switch)	2.10.2.1	*https://docs.pica8.com/spacedirectory/view.action*
PostgreSQL	12.2	*https://www.postgresql.org/docs/*
Prometheus	2.18.1	
Redfish API	N/A	*https://www.dmtf.org/standards/redfish*
Singularity	3.2	*https://www.sylabs.io/*
Slurm	19.05.0	*https://slurm.schedmd.com/configurator.html*
Strimzi Operator	0.15.0	
SUSE® Linux Enterprise Server (SLES)	SLE 15 SP1 and SLE 15 SP2	*https://www.suse.com/documentation/*
TensorFlow	1.13.1 (CPU)	*https://www.tensorflow.org/*
TFTP	5.2	Docker based on Alpine.
TimescaleDB	0.10.0-pg10	*https://github.com/timescale/timescaledb*
Vault	1.1.5	
Vault Operator	1.8.0	
Weave	2.8.0	*https://github.com/weaveworks/weave*
ZooKeeper	3.4.10	*https://github.com/apache/zookeeper/*

3 Introduction to Software Installation on a HPE Cray EX System

Installing software on a HPE Cray EX system is quite different than on previous Cray systems (XC series and CS series) because the fundamental architecture of the Shasta System Management is different. Rather than a set of highly available management nodes, the Shasta System Management uses a resilient set of services running within a distributed system. This architecture exists on both the public and private cloud management systems and supports tooling interoperability with the cloud. To gain an in depth understanding an installation, it is important to note the following:

- Management and Service Nodes no longer have dedicated purposes like boot, SDB, etc. Instead, the installation prepares and deploys a distributed system across a set of converged commodity hardware. Services can "float" from one server and horizontally scale to support the Managed Plane.
- System services on NCNs are provided as containerized microservices within a Kubernetes cluster, and packaged for deployment as helm charts.
- Containers and Helm charts are installed through native package management tools instead of rpm and can be queried and audited through native tooling.
- RPM content for installation is minimal and limited to those utilities required for installation and certificate generation.

The installation procedure is managed via automation, with tooling to interact with the automation.

The declarative configuration is available for analysis and management after the installation is complete. It is intended to be safely re-applied to return a system to the original state for micro-services.

The installation process can also be configured for turnkey operation for consistent, reliable, predictable installations.

3.1 System Nodes and Networks

The HPE Cray EX system includes two types of nodes:

- **Compute Nodes**, where high performance computing applications are run, and node names in the form of `nidXXXXXX`
- **Non-Compute Nodes (NCNs)**, which carry out system functions and come in three versions:
  - **Master** nodes, with names in the form of `ncn-mXXX`
  - **Worker** nodes, with names in the form of `ncn-wXXX`
  - **Utility Storage** nodes, with names in the form of `ncn-sXXX`

The HPE Cray EX system includes the following nodes:

- Nine or more non-compute nodes (NCNs) that host system services:

  - `ncn-m001`, `ncn-m002`, and `ncn-m003` are configured as Kubernetes master nodes.
  - `ncn-w001`, `ncn-w002`, and `ncn-w003` are configured as Kubernetes worker nodes. Every system contains three or more worker nodes.
  - `ncn-s001`, `ncn-s002` and `ncn-s003` for storage. Every system contains three or more utility storage node.
- Compute nodes, of the form nidXXXXXX. Commonly starting at nid000001, but this is configurable.

The following system networks connect the devices listed:

- Networks external to the system:
  - Customer Network (Data Center)
    - `ncn-m001` BMC is connected by the customer network switch to the customer **management** network
    - All NCNs (worker, master, and storage) are connected
    - ClusterStor System Management Unit (SMU) interfaces
    - User Access Nodes (UANs)
- System networks:
  - Customer Network (Data Center)
    - `ncn-m001` BMC is connected by the customer network switch to the customer **management** network
    - ClusterStor SMU interfaces
    - User Access Nodes (UANs)
  - Hardware Management Network (HMN)
    - BMCs for Admin tasks
    - Power distribution units (PDU)
    - Keyboard/video/mouse (KVM)
  - Node Management Network (NMN)
    - All NCNs and compute nodes
    - User Access Nodes (UANs)
  - ClusterStor Management Network
    - ClusterStor controller management interfaces of all ClusterStor components (SMU, Metadata Management Unit (MMU), and Scalable Storage Unit (SSU))
  - High-Speed Network (HSN), which connects the following devices:
    - Kubernetes worker nodes
    - UANs
    - ClusterStor controller data interfaces of all ClusterStor components (SMU, MMU, and SSU)

During initial installation, several of those networks are created with default IP address ranges. See *Default IP Address Ranges* on page 15.

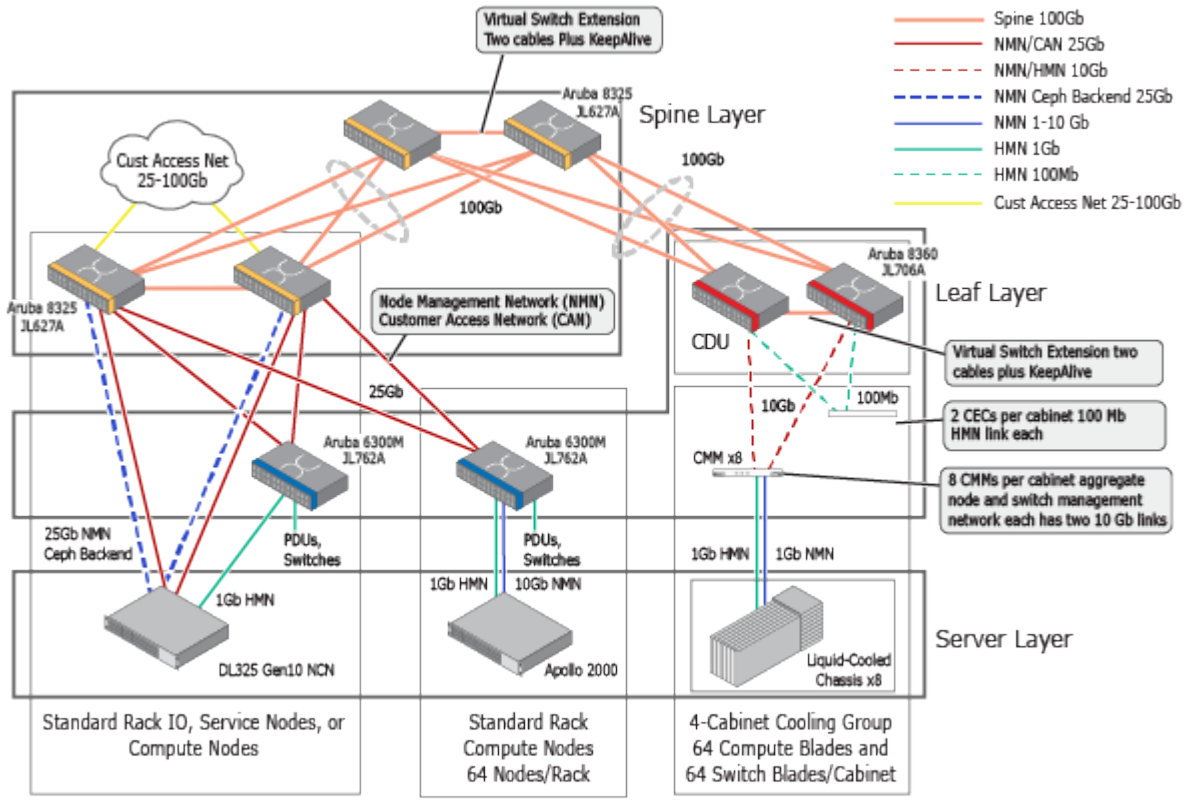
The network management system (NMS) data model and REST API enable customer sites to construct their own "networks" of nodes within the high-speed fabric, where a "network" is a collection of nodes that share a VLAN and an IP subnet.

The low-level network management components (switch, DHCP service, ARP service) of the NCNs and ClusterStor interfaces are configured to serve one particular network (the "supported network") on the high-speed

fabric. As part of the initial installation, the supported network is created to include all of the compute nodes, thereby enabling those compute nodes to access the gateway, user access services, and ClusterStor devices.

A site may create other networks as well, but it is only the supported network that is served by those devices.

Figure 1. Management Network Connections - Cray EX System



3.2 Default IP Address Ranges

The initial installation of the system creates default networks with default settings and with no external exposure. These IP address default ranges ensure that no nodes in the system attempt to use the same IP address as a Kubernetes service or pod, which would result in undefined behavior that is extremely difficult to reproduce or debug.

The following table shows the default IP address ranges:

Table 3. Default IP Address Ranges

Network	IP Address Range
Kubernetes service network	10.16.0.0/12
Kubernetes pod network	10.32.0.0/12
Install Network (MTL)	10.1.0.0/16

Network	IP Address Range
Node Management Network (NMN)	10.252.0.0/17
High Speed Network (HSN)	10.253.0.0/16
Hardware Management Network (HMN)	10.254.0.0/17
Mountain NMN Allocate a `/22` from this range per liquid cooled cabinet: ● cabinet 1 ● cabinet 2 ● cabinet 3 ● ...	10.100.0.0/17 Example IP address in the allocated ranges: ● 10.100.0.0/22 ● 10.100.4.0/22 ● 10.100.8.0/22 ● ...
Mountain HMN Allocate a `/22` from this range per liquid cooled cabinet: ● cabinet 1 ● cabinet 2 ● cabinet 3 ● ...	10.104.0.0/17 Example IP address in the allocated ranges: ● 10.104.0.0/22 ● 10.104.4.0/22 ● 10.104.8.0/22 ● ...
River NMN	10.106.0.0/17
River HMN	10.107.0.0/17
Load Balanced NMN	

The above values could be modified prior to install if there is a need to ensure that there are no conflicts with customer resources, such as LDAP or license servers. If a customer has more than one HPE Cray EX system, these values can be safely reused across them all.

Contact customer support for this site if it is required to change the IP address range for Kubernetes services or pods; for example, if the IP addresses within those ranges must be used for something else. The cluster must be fully reinstalled if either of those ranges are changed.

Customizable Network Values

There are several network values and other pieces of system information that must be unique to the customer system.

- IP values and the network for `ncn-m001` and the BMC on `ncn-m001`.
- The main Customer Access Network (CAN) subnet and the two address pools mentioned below need to be part of the main subnet.

  For more information on the CAN, see "Customer Access Network (CAN)" in the HPE Cray EX System Administration Guide S-8001.

  - Subnet for the MetalLB static address pool (`can-static-pool`), which is used for services that need to be pinned to the same IP address, such as the system DNS service.
  - Subnet for the MetalLB dynamic address pool (`can-dynamic-pool`), which is used for services such as User Access Instances (UAIs) that can be reached by DNS.
- HPE Cray EX Domain: The value of the subdomain that is used to access externally exposed services.

  For example, if the system is named TestSystem, and the site is `example.com`, the HPE Cray EX domain would be *testsystem.example.com*. Central DNS would need to be configured to delegate requests for addresses in this domain to the HPE Cray EX DNS IP for resolution.
- HPE Cray EX DNS IP: The IP address used for the HPE Cray EX DNS service. Central DNS delegates the resolution for addresses in the HPE Cray EX Domain to this server. The IP address will be in the `can-static-pool` subnet.
- CAN gateway IP address: The IP address assigned to a specific port on the spine switch, which will act as the gateway between the CAN and the rest of the customer's internal networks. This address would be the last-hop route to the CAN network.

3.3 Resilience of System Management Services

HPE Cray EX systems are designed so that system management services (SMS) are fully resilient and that there is no single point of failure. The design of the system allows for resiliency in the following ways:

- Three non-compute nodes (NCNs) are configured as Kubernetes master nodes. When one master goes down, operations (such as jobs running across compute nodes) are expected to continue.
- At least three utility storage nodes provide persistent storage for the services running on the Kubernetes management nodes. When one of the utility storage nodes goes down, operations (such as jobs running across compute nodes) are expected to continue.
- At least three NCNs are configured as Kubernetes worker nodes. If one of only three Kubernetes worker nodes were to go down, it would be much more difficult for the remaining two NCN worker nodes to handle the total balance of pods. It is less significant to lose one of the NCN worker nodes if the system has more than three NCN worker nodes because there are more worker nodes able to handle the pod load.
- The state and configuration of the Kubernetes cluster are stored in an etcd cluster distributed across the Kubernetes master nodes. This cluster is also backed up on an interval, and backups are pushed to Ceph Rados Gateway (S3).
- A microservice can run on any node that meets the requirements for that microservice, such as appropriate hardware attributes, which are indicated by labels and taints.
- All microservices have shared persistent storage so that they can be restarted on any NCN in the Kubernetes management cluster without losing state.

Kubernetes is designed to ensure that the wanted number of deployments of a microservice are always running on one or more worker nodes. In addition, it ensures that if one wotker node becomes unresponsive, the microservices that were running on it are migrated to another NCN that is up and meets the requirements of those microservices.

See "Restore System Functionality if a Kubernetes Worker Node is Down" in the "Resilience of System Management Services" section of the *HPE Cray EX System Administration Guide S-8001* for more information.

Resiliency Improvements in Release v1.4

To increase the overall resiliency of system management services and software within the system, the following improvements were made during the v1.4 release:

- Capsules services have implemented replicas for added resiliency.
- Added support for new storage class that supports Read-Write-Many and in doing so, eliminated some of the errors we encountered on pods which could not seamlessly start up on other worker NCNs upon termination (due to a PVC unmount error).
- Modified procedures for reloading DVS on NCNs to reduce DVS service interruptions.
- DVS now retries DNS queries in dvs_generate_map, which improves boot resiliency at scale.
- Additional retries implemented in the BOA, IMS, and CFS services for increased protection around outages of dependent services.
- Image based installs emphasizing "non-special" node types eliminated single points of failure previously encountered with DNS, administrative and installation tooling, and gathering of Cray System Management (CSM) logging for Ceph and Kubernetes.

Expected Resiliency Behavior in Release v1.4

In addition, the following general criteria describe the expected behavior of the system if a single Kubernetes node (master, worker, or storage) goes down temporarily:

- Once a job has been launched and is executing on the compute plane, it is expected that it will continue to run without interruption during planned or unplanned outages characterized by the loss of an NCN master, NCN worker, or NCN storage node. Applications launched through PALS may show error messages and lost output if a worker node goes down during application runtime.
- If an NCN worker node goes down, it will take between 4 and 5 minutes before most of the pods which had been running on the downed NCN will begin terminating. This is a predefined Kubernetes behavior, not something inherent to HPE Cray EX.
- Within around 20 minutes or less, it should be possible to launch a job using a UAI or UAN after planned or unplanned outages characterized by the loss of an NCN master, NCN worker, or NCN storage node.
    - In the case of a UAN, the recovery time is expected to be quicker. However, launching a UAI after an NCN outage means that some UAI pods may need to relocate to other NCN worker nodes... and the status of those new UAI pods will remain unready until all necessary content has been loaded on the new NCN that the UAI is starting up on. This process can take approximately 10 minutes.
- Within around 20 minutes or less, it should be possible to boot and configure compute nodes after planned or unplanned outages characterized by the loss of an NCN master, NCN worker, or NCN storage node.
- At least three utility storage nodes provide persistent storage for the services running on the Kubernetes management nodes. When one of the utility storage nodes goes down, critical operations such as job launch, app run, or compute boot are expected to continue to work.
- Not all pods running on a downed NCN worker node are expected to migrate to a remaining NCN worker node. There are some pods which are configured with anti-affinity such that if the pod exists on another NCN worker node, it will not start another of those pods on that same NCN worker node. At this time, this mostly only applies to etcd clusters running in the cluster. It is optimal to have those pods balanced across the NCN worker nodes (and not have multiple etcd pods, from the same etcd cluster, running on the same NCN worker node). Thus, when an NCN worker node goes down, the etcd pods running on it will remain in terminated state and will not attempt to relocate to another NCN worker node. This should be fine as there should be at

least two other etcd pods (from the cluster of 3) running on other NCN worker nodes. Additionally, any pods that are part of a stateful set will not migrate off a worker node when it goes down. Those are expected to stay on the node and also remain in the terminated state until the NCN worker nodes comes back up or unless deliberate action is taken to force that pod off the NCN worker node which is down.

    - The cps-cm-pm pods are part of a daemonset and they only run on designated nodes. When the node comes back up the containers will be restarted and service restored. See "Content Projection Service (CPS)" in the *HPE Cray EX System Administration Guide S-8001* for more information on changing node assignments.

- After an NCN worker, storage, or master node goes down, if there are issues with launching a UAI session or booting compute nodes, that does not necessarily mean that the problem is due to a worker node being down. It is certainly, possible, but it is advised to also check the relevant "Compute Node Boot Troubleshooting Information" and "User Access Service" (specifically with respect to "Troubleshooting UAS Issues") sections in the *HPE Cray EX System Administration Guide S-8001*. Those sections can give guidance around general known issues and how to troubleshoot them. For any customer support ticket opened on these issues, however, it would be an important piece of data to include in that ticket if the issue was encountered while one or more of the NCNs were down.

Known Issues and Workarounds for Release v1.4

Though an effort was made to increase the number of pod replicas for services that were critical to system operations such as booting computes, launching jobs, and running applications across the compute plane, there are still some services that remain with single copies of their pods. In general, this does not result in a critical issue if these singleton pods are on an NCN worker node that goes down. Most microservices should (after being terminated by Kubernetes), simply be rescheduled onto a remaining NCN worker node. That assumes that the remaining NCN worker nodes have sufficient resources available and meet the hardware/network requirements of the pods.

However, it is important to note that some pods, when running on a worker NCN that goes down, may require some manual intervention to be rescheduled. Note the workarounds in this section for such pods. Work is on-going to correct these issues in a future release.

- **Nexus pod**
    - The nexus pod is a single pod deployment and serves as our image repository. If it is on an NCN worker node that goes down, it will attempt to start up on another NCN worker node. However, it is likely that it can also encounter the "Multi-Attach error for volume" error that can be seen in the `kubectl describe` output for the pod that is trying to come up on the new node.
    - To determine if this is happening, run the following:

      ```
      # kubectl get pods -n nexus | grep nexus
      ```

    - Describe the pod obtained in the previous bullet:

      ```
      # kubectl describe pod -n nexus NEXUS_FULL_POD_NAME
      ```

    - If the event data at the bottom of the `describe` command output indicates that a Multi-Attach PVC error has occurred. See the "Troubleshooting Pods Failing to Restart on Other Worker Nodes" procedure in the *HPE Cray EX System Administration Guide S-8001* to unmount the PVC. This will allow the Nexus pod to begin successfully running on the new NCN worker node.
- **High-speed network resiliency after `ncn-w001` goes down**

- The slingshot-fabric-manager pod running on one of NCNs does not rely on `ncn-w001`. If `ncn-w001` goes down, the slingshot-fabric-manager pods should not be impacted as the pod is runs on other NCNs, such as `ncn-w002`.
- The slingshot-fabric-manager pod relies on Kubernetes to launch the new pod on another NCN if the slingshot-fabric-manager pod is running on `ncn-w001` when it is brought down.

  Use the following command and check the `NODE` column to check which NCN the pod is running on:

  ```
  # kubectl get pod -n services -o wide | awk 'NR == 1 || /slingshot-fabric-manager/'
  ```
- When the slingshot-fabric-manager pod goes down, the switches will continue to run. Even if the status of the switches changes, those changes will be picked up after slingshot-fabric-manager pod is brought back up and the sweeping process restarts.
- The slingshot-fabric-manager relies on data in persistent storage. The data is persistent across upgrades but when the pods are deleted, the data is also deleted.

Future Resiliency Improvements

In a future release, strides will be made to further improve the resiliency of the system. These improvements may include one or more of the following:

- Further emphasis on eliminating singleton system management pods.
- Reduce time delays for individual service responsiveness after its pods are terminated, due to running on a worker node that has gone down.
- Rebalancing of pods/workloads after an NCN worker node that was down, comes back up.
- Analysis/improvements wrt outages of the Node Management Network (NMN) and the impact to critical system management services.
- Expanded analysis/improvements of resiliency of noncritical services (those that are not directly related to job launch, application run, or compute boot).

3.4 Access to System Management Services

The standard configuration for System Management Services (SMS) is the containerized REST micro-service with a public API. All of the micro-services provide an HTTP interface and are collectively exposed through a single gateway URL. The API gateway for the system is available at a well known URL based on the domain name of the system. It acts as a single HTTPS endpoint for terminating Transport Layer Security (TLS) using the configured certificate authority. All services and the API gateway are not dependent on any single node. This resilient arrangement ensures that services remain available during possible underlying hardware and network failures.

Access to individual APIs through the gateway is controlled by a policy-driven access control system. Admins and users must retrieve a token for authentication before attempting to access APIs through the gateway and present a valid token with each API call. The authentication and authorization decisions are made at the gateway level which prevent unauthorized API calls from reaching the underlying micro-services. Refer to the *HPE Cray EX System Administration Guide S-8001* for more detail on the process of obtaining tokens and user management.

Review the API documentation in the supplied container before attempting to use the API services. This container is generated with the release using the most current API descriptions in OpenAPI 2.0 format. Since this file serves

as both an internal definition of the API contract and the external documentation of the API function, it is the most up-to-date reference available.

The API Gateway URL for accessing the APIs on a site-specific system is `https://api.SYSTEM-NAME_DOMAIN-NAME/apis/`.

The internal URL from a local console on any of the NCNs is `https://api-gw-service-nmn.local/apis`.

3.5 System Management Levels

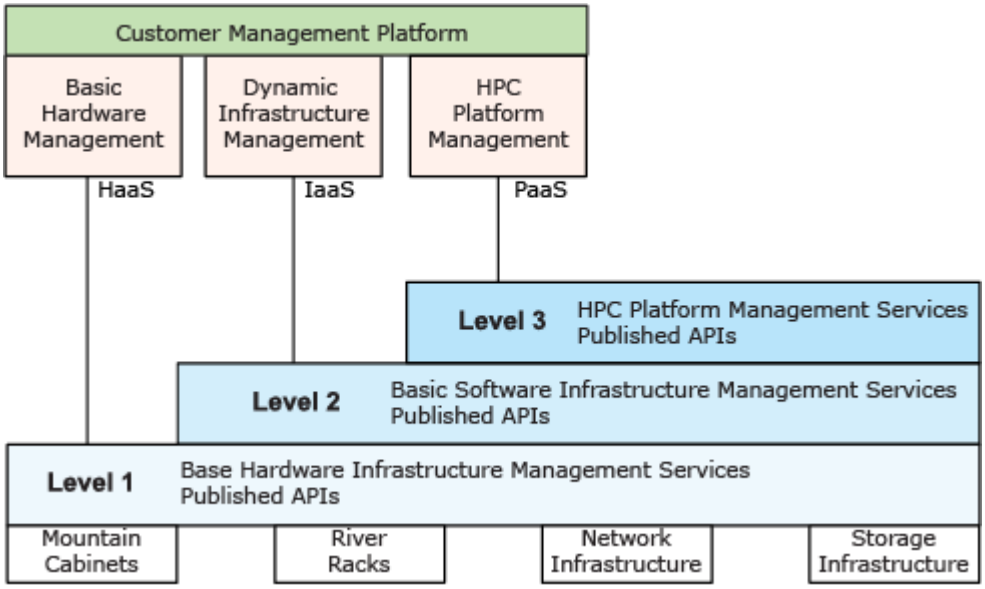
Cray customers are provided flexible options through the following system management levels:

Level 1 Hardware as a Service (HaaS) uses Redfish APIs provide access to services that monitor and control system hardware components, such as chassis, nodes, racks, and switches. Level 1 services are provided by Cray, Inc. only.

Level 2 Infrastructure as a Service (IaaS) APIs provide access to services that manage the infrastructure and interaction of compute, network, and basic storage resources. This level of system management includes the capability to deliver customer-provided image content onto these resources through an image repository and at-scale booting capabilities. Level 2 services are provided by Cray, Inc. and optionally, by Cray customers. Level 2 services consume level 1 services, but level 1 services do not consume or depend on level 2 services.

Level 3 Platform as a Service (PaaS) APIs build on the capabilities provided at levels 1 and 2 to provide platform support services for Cray's capability-class software stack: Cray Linux Enviroment (CLE) and Cray Analytics environment. In addition to supporting the CLE, the system management software allows customers to build and manage their own software stack. Level 3 services consume level 2 and level 1 services, but level 2 and level 1 services do not consume or depend on level 3 services.

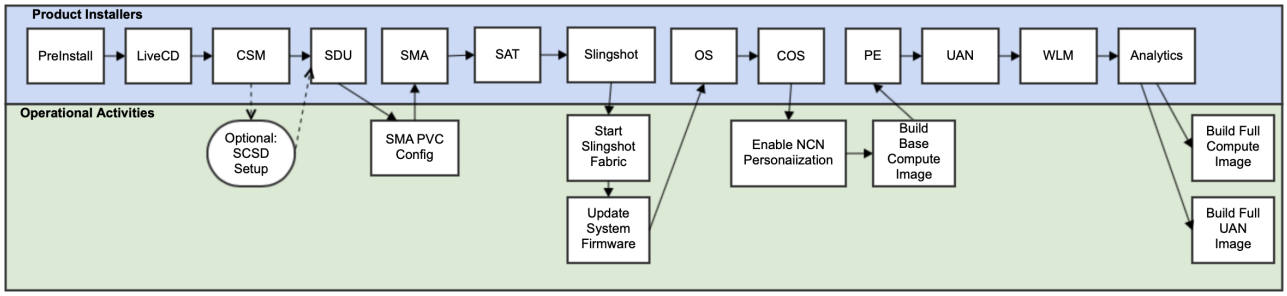
Figure 2. System Management Levels



3.6 HPE Cray EX Installation Overlay

In HPE Cray EX-1.4, the components of the release have been separated into individual products to allow for greater flexibility in the upgrades of products in the future. This is an outline of the order in which a typical HPE Cray EX-1.4 install might occur. While software products do not necessarily need to be installed in this order using this sequence for product installations ensures system dump (SDU), logging (SMA) and administrative tools are available early in the install.

Figure 3. Installation Overview



Product Installations

CSM

Generate the LiveCD for the system and boot from it using instructions provided in the CSM installation. Following the boot of the Pre-Install Toolkit (PIT), follow Product Installation per CSM install documentation provided with the release. Once the CSM install is complete, the PIT node will be rebooted to the on-disk with ncn-m001running as a kubernetes master node.

CSM documentation is changing and updating daily. The content reflected in the guide may not be the most current version. It is important to obtain the latest CSM documentation and workarounds. The latest CSM documentation and workaround RPMs can be installed by running the following:

```
rpm -Uhv https://storage.googleapis.com/csm-release-public/shasta-1.4/csm-install-workarounds/csm-install-workarounds-latest.noarch.rpm
rpm -Uhv https://storage.googleapis.com/csm-release-public/shasta-1.4/docs-csm-install/docs-csm-install-latest.noarch.rpm
```

If the RPMs will be installed onto a system with no internet access, they will need to be manually downloaded and synced to the system.

The RPMs need to be installed three times.

- On the prep machine used to prepare the LiveCD
- Once booted into the LiveCD
- After rebooting the LiveCD into its permanent role as m001

The documentation can be found under `/user/share/doc/metal` and workarounds under `/opt/cray/csm/workarounds`.

Prep for Additional Packages

After ncn-m001 has rebooted into the kubernetes cluster at the completion of the CSM installer, login to ncn-m001 and create a mount point for the USB drive that was used for the LiveCD and mount the PITDATA partition.

```
ncn-m001# mkdir -p /mnt/pitdata
ncn-m001# mount -L PITDATA /mnt/pitdata
```

After mounting the USB, transfer other product installers from the Shasta-1.4 install media to the system being installed.

```
administrator_system % cd V1.4
administrator_system % scp RC1/* customer-system-m001:/mnt/pitdat
```

Configure SCSD (Operational Task)

For systems with Liquid cooled hardware, it may be useful to use SCSD to enable public key authentication between NCNs and hardware management components (e.g. CMMs, NCs). See the "Enable Passwordless Connections to Liquid Cooled Node BMCs" section of the Admin Guide**.** This step simplifies debugging of many operational issues related to the use of Liquid cooled hardware

SDU

Extract the install media from the product tar ball. Run the SDU installer as directed by the install guide and complete the configuration of SDU.

SMA/PMDB

Extract the install media from the product tar ball. Update the customizations.yaml with SMA related PVC information specific to the system using instructions within the install guide.

Run the SMA installer using the updated customizations.yaml with system specific PVC values as directed in the INSTALL log within the product.

SAT

Extract the install media from the product tar ball. Run the SAT installer as directed in the install guide and configure as documented in the installation guide.

Slingshot

Extract the install media from the product tar ball. Run the slingshot installer as directed in installation guide.

Start Slingshot Fabric (Operational Task)

Initialize the slingshot fabric using the instructions in the install guide.

Update Slingshot Switch Firmware (Operational Task)

Update slingshot controller firmware to ensure correct operation of the HSN with the Shasta-1.4 fabric manager.

Update Other Hardware Firmware (Operational Task)

System firmware for compute nodes and other liquid cooled hardware components should be updated prior to continuing on to the next steps.

OS

Extract the install media from the product tar ball.

Run the OS installers as directed in the INSTALL log within the product. Note this product installation can be time consuming, but can generally be performed along with other product installations.

COS

Extract the install media from the product tar ball.

Run the COS installers as directed in the INSTALL log within the product.

Enable NCN Personalization (Operational Task)

Run the NCN Personalization instructions using development guidance

Build Initial Boot Image (Operational Task)

Run the Boot Image build commands per the COS CFS instructions. Optionally attempt to boot the COS customized BOS sessiontemplate.

PE

Install per INSTALL payload within the PE installation media.

UAN

Install per UAN documentation within the install media.

WLM

Install per Workload Manager payload within the installer package.

Analytics

Install per the instructions within the Analytics product installation in the install guide.

Build Full Compute OS Image (Operational Task)

Build the CFS configuration file with the components that should be included in the compute node image. This could include:

- COS
- LDMS
- Cray Programming Environment (CPE)
- WLM (SLURM or PBS Pro)
- Analytics

Build Full UAN OS Image (Operational Task)

Build the CFS configuration file with the components that should be included in the UAN / Application node image. This could include:

- UAN
- Cray Programming Environment (CPE)

- WLM (SLURM or PBS Pro)
- Analytics

Update NCN Personalization (Operational Task)

Tasks Requiring Downtime

- Firmware Updates.
  - Perform Mellanox Connect X4 FW updates to enable PXE booting over the NICs

    ```
    groot-ncn-m001:~/csm/csm-0.8.0 # find . -name mft\*
    ./rpm/embedded/cray/slingshot/sle-15sp2/x86_64/mft-4.15.1-9.x86_64.rpm
    ```
  - Update Mellanox Configuration to allow for PXE booting
  - Perform Gigabyte NCN FW updates (limited to workers, storage and master nodes)
  - Perform Dell Switch FW updates
  - Perform Mellanox Switch FW updates
- Move the site management cable for ncn-w001 to ncn-m001 for the site connection and BMC
- Update the "hmn_connections.json" file from 1.3 to show the connection going to wn01 rather than mn01

4 Prepare for a 1.4 Install

The following procedures detail prerequisites needed for 1.4 installs and reinstalls.

4.1 1.3 to 1.4 Install Prerequisites

About this task

The following prerequisites must be performed when installing 1.4 onto systems with existing 1.3 software.

Procedure

1. Collect Shasta-1.4 config payload. Refer to *Collect Data From Healthy Shasta 1.3 System for EX 1.4 Installation* on page 151 an*Service Guides* on page 165 .
2. Quiesce Shasta v1.3 system.
   a. Check for running slurm jobs.

      ```
      ncn-w001# ssh nid001000 squeue -l
      ```

   b. Check for running PBS jobs.

      ```
      ncn-w001# ssh nid001000 qstat -Q
      ncn-w001# ssh nid001000 qstat -a
      ```

   c. Obtain authorization key for SAT.

      See System Security and Authentication, Authenticate an Account with the Command Line, SAT Authentication in the Cray Shasta Administration Guide 1.3 S-8001 for more information.

   d. Check for running sessions from BOS, CFC, CRUS, FAS, and NMD.

      ```
      ncn-w001# sat bootsys shutdown --stage session-checks
      Checking for active BOS sessions.
      Found no active BOS sessions.
      Checking for active CFS sessions.
      Found no active CFS sessions.
      Checking for active CRUS upgrades.
      Found no active CRUS upgrades.
      Checking for active FAS actions.
      Found no active FAS actions.
      Checking for active NMD dumps.
      Found no active NMD dumps.
      No active sessions exist. It is safe to proceed with the shutdown procedure.
      ```

e. Shut down and power off all compute nodes and application nodes.

- Gigabyte nodes should use the Gigabyte Node Firmware Update Guide (1.3.2) S-8010 while booted with Shasta v1.3.2. However, since v1.3 will never be booted again on this system, there is no need to ensure that the etcd clusters are healthy and that BGP Peering has been ESTABLISHED as recommended.
- Nodes with Mellanox ConnectX-4 and ConnectX-5 PCIe NICs need to update their firmware. This should be done while Shasta v1.3.2 is booted. The Mellanox ConnectX-4 cards will be enabled for PXE booting later.
- v1.3.2 use the "sat bootsys shutdown" command to do compute nodes and application nodes at the same time. See the Cray Shasta Administration Guide 1.3 S-8001 RevF (or later) in the section "Shut Down and Power Off Compute and User Access Nodes"
- v1.3.0 use the "cray bos" command to create a shutdown session for the compute nodes and then for the application nodes. See the Cray Shasta Administration Guide 1.3 S-8001 RevC (or later) in the section "Shut Down and Power Off Compute and User Access Nodes"

3. Upgrade BIOS and Firmware.

   For minimum Network switch versions and NCN firmware versions, refer to *Firmware Updates* on page 243.

4. Confirm Site and PCIe connections, refer to *Update Site Connections* on page 219 and *Netboot an NCN from a Spine Switch* on page 251.

5. Shut down Ceph and the Kubernetes management cluster. This performs several actions to quiesce the management services and leaves each management NCN running Linux, but no other services.

   If v1.3.2 software or later is installed use the following command.

   Shutdown platform services.

   ```
   ncn-w001# sat bootsys shutdown --stage platform-services
   ```

   If Shasta v1.3.0 or v1.3.1 is installed, follow these three steps from Cray Shasta Administration Guide 1.3 S-8001 in section 6.6 Shut Down and Power Off the Management Kubernetes Cluster.

6. Determine if patch 1.3.1 has been installed.

7. (Only perform this step if 1.3.1 has been installed). Modify /opt/cray/crayctl/ansible_framework/main/roles/cri-ctl/tasks/main.yml file with the following lines.

8. Run the platform-shutdown.yml playbook.

   ```
   ncn-w001# ansible-playbook \
      /opt/cray/crayctl/ansible_framework/main/platform-shutdown.yml
   ```

9. Power off NCNs.

   NCNs hosting Kubernetes services need to be powered off to facilitate a 1.4 install. Wiping the node will avoid boot mistakes, making the PXE the only viable option.

   UANs and CNs do not need to power off.

   ```
   ncn-m001# ssh ncn-w001
   ncn-w001# ansible ncn -m shell -a 'wipefs --all --force /dev/sd*'
   ```

Disks that have no labels will show no output by the wipefs commands being run. If one or more disks have labels, output similar to the following example is expected.

```
/dev/sda: 8 bytes were erased at offset 0x00000200 (gpt): 45 46 49 20 50 41 52
54
/dev/sda: 8 bytes were erased at offset 0x6fc86d5e00 (gpt): 45 46 49 20 50 41
52 54
/dev/sda: 2 bytes were erased at offset 0x000001fe (PMBR): 55 aa
/dev/sdb: 8 bytes were erased at offset 0x00000200 (gpt): 45 46 49 20 50 41 52
54
/dev/sdb: 8 bytes were erased at offset 0x6fc86d5e00 (gpt): 45 46 49 20 50 41
52 54
/dev/sdb: 2 bytes were erased at offset 0x000001fe (PMBR): 55 aa
/dev/sdc: 6 bytes were erased at offset 0x00000000 (crypto_LUKS): 4c 55 4b 53
ba be
/dev/sdc: 6 bytes were erased at offset 0x00004000 (crypto_LUKS): 53 4b 55 4c
ba be
```

Power off all nodes except for ncn-m001 and ncn-w001.

```
ncn-w001# ansible ncn -m shell --limit='!ncn-w001:!ncn-m001' -a 'ipmitool power
off'
```

Power off ncn-w001.

```
ncn-w001# ipmitool power off
```

At this time ncn-m001 is the only node powered on. The final ipmitool power off command should disconnect from ncn-w001, returning to ncn-m001.

If the connection fails to disconnect, an admin can escape and disconnect the IPMI without exiting the SSH session by pressing `~.` until `ipmitool` disconnects.

10. Exit typescript from prep.install.

11. Save collected typescript and health check information off the system so it can be referenced later.

    The system is now ready to start an installation.

4.2 Bare Metal Install Prerequisites

About this task

The following procedure details prerequisites that must be performed before continuing with a bare metal installation.

Procedure

1. Setup LiveCD.

   A 1TB USB3.0 USB stick will be required in order to create a bootable LiveCD.

2. Collect Shasta-1.4 config payload. Refer to *Collect Data From Healthy Shasta 1.3 System for EX 1.4 Installation* on page 151 and *Service Guides* on page 165.

3. Apply network configuration and firmware. Refer to *Install the Management Network* on page 267.

4. Upgrade BIOS and Firmware.

   For minimum Network switch versions and NCN firmware versions, refer to *Firmware Updates* on page 243.

4.3 1.4 Reinstall Prerequisites

About this task

The following procedure must be performed before starting a 1.4 reinstall. UANs and CNs do not need to power off during this procedure.

Procedure

1. Scale down DHCP.

   Scale the deployment from either the LiveCD or any Kubernetes node.

   ```
   ncn-m001# kubectl scale -n services --replicas=0 deployment cray-dhcp-kea
   ```

2. Perform a wipe of the system.

   Wipe disks from LiveCD (pit).

   ```
   pit# ncns=$(grep Bond0 /etc/dnsmasq.d/statics.conf | grep -v m001 | awk -F',' 
   '{print $6}')
   pit# for h in $ncns; do
       read -r -p "Are you sure you want to wipe the disks on $h? [y/N] " response
       response=${response,,}
       if [[ "$response" =~ ^(yes|y)$ ]]; then
            ssh $h 'wipefs --all --force /dev/sd* /dev/disk/by-label/*'
       fi
   done
   ```

   Wipe NCN disks from ncn-m001.

   ```
   ncn-m001# ncns=$(grep ncn /etc/hosts | grep nmn | grep -v m001 | awk '{print 
   $3}')
   ncn-m001# for h in $ncns; do
       read -r -p "Are you sure you want to wipe the disks on $h? [y/N] " response
       response=${response,,}
       if [[ "$response" =~ ^(yes|y)$ ]]; then
            ssh $h 'wipefs --all --force /dev/sd* /dev/disk/by-label/*'
       fi
   done
   ```

   Disks that have no labels will have show no output. If one or more disks have labels, output similar to the following is expected:

   ```
   ...
   Are you sure you want to wipe the disks on ncn-m003? [y/N] y
   /dev/sda: 8 bytes were erased at offset 0x00000200 (gpt): 45 46 49 20 50 41 52 
   54
   ```

```
/dev/sda: 8 bytes were erased at offset 0x6fc86d5e00 (gpt): 45 46 49 20 50 41
52 54
/dev/sda: 2 bytes were erased at offset 0x000001fe (PMBR): 55 aa
/dev/sdb: 8 bytes were erased at offset 0x00000200 (gpt): 45 46 49 20 50 41 52
54
/dev/sdb: 8 bytes were erased at offset 0x6fc86d5e00 (gpt): 45 46 49 20 50 41
52 54
/dev/sdb: 2 bytes were erased at offset 0x000001fe (PMBR): 55 aa
/dev/sdc: 6 bytes were erased at offset 0x00000000 (crypto_LUKS): 4c 55 4b 53
ba be
/dev/sdc: 6 bytes were erased at offset 0x00004000 (crypto_LUKS): 53 4b 55 4c
ba be
...
```

3. Power down each NCN to minimize network activity.

   Power each NCN off using `ipmitool` from ncn-m001 or the booted LiveCD if reinstalling an incomplete install.

   - Shutdown from LiveCD.

     ```
      pit# export username=root
     pit# export IPMI_PASSWORD=changeme
     pit# conman -q | grep mgmt | xargs -t -i  ipmitool -I lanplus -U $username -
     E -H {} power off
     ```

   - Shutdown from ncn-m001.

     ```
     ncn-m001# export username=root
     ncn-m001# export IPMI_PASSWORD=changeme
     ncn-m001# grep ncn /etc/hosts | grep mgmt | grep -v m001 | sort -u | awk
     '{print $2}' | xargs -t -i ipmitool -I lanplus -U $username -E -H {} power
     off
     ```

4. Set the BMCs on the system back to DHCP.

   - This step uses the old statics.conf on the system in case CSI changes IPs:

     ```
     pit# export username=root
     pit# export IPMI_PASSWORD=
     pit# for h in $( grep mgmt /etc/dnsmasq.d/statics.conf | grep -v m001 | awk -
     F ',' '{print $2}' )
     do
     ipmitool -U $username -I lanplus -H $h -E lan set 1 ipsrc dhcp
     done

     The timing of this change can vary based on the hardware, so if the IP can
     no longer be reached after running the above command, run these commands.

     pit# for h in $( grep mgmt /etc/dnsmasq.d/statics.conf | grep -v m001 \
     | awk -F ',' '{print $2}' )
     do
     ipmitool -U $username -I lanplus -H $h -E lan print 1 | grep Source
     done

     pit# for h in $( grep mgmt /etc/dnsmasq.d/statics.conf | grep -v m001 \
     | awk -F ',' '{print $2}' )
     do
     ipmitool -U $username -I lanplus -H $h -E mc reset cold
     done
     ```

- This step uses to the `/etc/hosts` file on ncn-m001 to determine the IP addresses of the BMCs. If on ncn-m001:

```
ncn-m001# export username=root
ncn-m001# export IPMI_PASSWORD=
ncn-m001# for h in $( grep ncn /etc/hosts | grep mgmt | grep -v m001 \
| awk '{print $2}' )
do
ipmitool -U $username -I lanplus -H $h -E lan set 1 ipsrc dhcp
done

The timing of this change can vary based on the hardware, so if the IP can
no longer be reached after running the above command, run these commands.

ncn-m001# for h in $( grep ncn /etc/hosts | grep mgmt | grep -v m001 \
| awk '{print $2}' )
do
ipmitool -U $username -I lanplus -H $h -E lan print 1 | grep Source
done

ncn-m001# for h in $( grep ncn /etc/hosts | grep mgmt | grep -v m001 \
| awk '{print $2}' )
do
ipmitool -U $username -I lanplus -H $h -E mc reset cold
done
```

5. Power off LiveCD or the ncn-m001.

This step can be skipped if the node is going to be used as a staging area to create the LiveCD.

```
ncn-m001:~ # poweroff
```

5 Apply the CSM Patch

About this task

ROLE	System installer
OBJECTIVE	Apply a CSM update patch to the release tarball. This ensures that the latest CSM product artifacts are installed on the HPE Cray EX supercomputer.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in this release.

Procedure

1. Verify that the Git version is at least 2.16.5.

   The patch process is known to work with Git >= 2.16.5. Older versions of Git may not correctly apply the binary patch.

   ```
   ncn-m001# git version
   git version 2.26.2
   ```

   If the Git version is less than 2.16.15, update Git to at least that version.

2. Download the source release version (*0.8.22*) of the CSM software package tarball and the compressed patch *csm-0.8.22-0.9.0.patch.gz* to the HPE Cray EX system.

   The following steps should be run from the node in which the release was downloaded onto. The following examples use ncn-m001.

3. Extract the source release distribution.

   ```
   ncn-m001# tar -xzf csm-0.8.22.tar.gz
   ```

4. Uncompress the patch.

   ```
   ncn-m001# gunzip csm-0.8.22-0.9.0.patch.gz
   ```

5. Apply the patch.

   ```
   ncn-m001# git apply -p2 --whitespace=nowarn --directory=csm-0.8.22 \
   csm-0.8.22-0.9.0.patch
   ```

6. Set CSM_RELEASE to reflect the new version.

   ```
   ncn-m001# export CSM_RELEASE="$(./csm-0.8.22/lib/version.sh)"
   ```

7. Update the name of the CSM release distribution directory.

```
ncn-m001# mv csm-0.8.22 $CSM_RELEASE
```

8. Create a tarball from the patched release distribution.

```
ncn-m001# tar -cvzf ${CSM_RELEASE}.tar.gz "${CSM_RELEASE}/"
```

6 Install LiveCD

The three distinct scenarios continue the same way at this point, with preparation and then booting from the LiveCD. This version of the documentation supports booting from a USB LiveCD.

6.1 Configure USB LiveCD

Prerequisites

- If installing on a previous 1.3 system, move external connections from ncn-w001 to ncn-m001.
- A USB stick or other Block Device (1TB), local to ncn-m001.
- The drive letter of the USB or Block Device (I.e. `/dev/sdd`).
- The CCD/SHCD `.xlsx` file for the system.
- The number of Mountain, River, and Hill cabinets in the system.
- A set of configuration information sufficient to fill out the listed flags for the `csi config init` command.

About this task

ROLE	• System installer/system administrator
OBJECTIVE	The following procedure must be logged into an operating system that is running on the disk of ncn-m001.
NEW IN THIS RELEASE	This entire procedure is new with this release.

Procedure

1. Start a typescript to capture the commands and output from the installation.

```
linux# script -af csm-usb-livecd.$(date +%Y-%m-%d).txt
linux# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Download the CSM software release to the Linux host which will be preparing the LiveCD.

```
linux# cd ~
linux# export ENDPOINT=URL_SERVER_Hosting_tarball
linux# export CSM_RELEASE=csm-x.y.z
linux# wget ${ENDPOINT}/${CSM_RELEASE}.tar.gz
```

3. Expand the CSM release.

```
linux# tar -zxvf ${CSM_RELEASE}.tar.gz
linux# ls -l ${CSM_RELEASE}
```

4. Install CSI.

```
linux# rpm -Uvh ./${CSM_RELEASE}/rpm/cray/csm/sle-15sp2/x86_64/cray-site-init-*.x86_64.rpm
```

5. Download and install/upgrade the workaround and documentation RPMs. If this machine does not have direct internet access these RPMs will need to be externally downloaded and then copied to be installed.

```
linux# rpm -Uvh https://storage.googleapis.com/csm-release-public/shasta-1.4/docs-csm-install/docs-csm-install-latest.noarch.rpm
linux# rpm -Uvh https://storage.googleapis.com/csm-release-public/shasta-1.4/csm-install-workarounds/csm-install-workarounds-latest.noarch.rpm
```

6. Show version of CSI installed.

```
linux# csi version
CRAY-Site-Init build signature...
Build Commit   : b3ed3046a460d804eb545d21a362b3a5c7d517a3-release-shasta-1.4
Build Time     : 2021-02-04T21:05:32Z
Go Version     : go1.14.9
Git Version    : b3ed3046a460d804eb545d21a362b3a5c7d517a3
Platform       : linux/amd64
App. Version   : 1.5.18
```

7. Install podman to support container tools required to generate sealed secrets.

   Podman RPMs are included in the `embedded` repository in the CSM release and can be installed in the pre-LiveCD enviroment using `zypper`.

   Add the `embedded` repository.

```
linux# zypper ar -fG "./${CSM_RELEASE}/rpm/embedded" "${CSM_RELEASE}-embedded"
```

   Install `podman` and `podman-cni-config` packages.

```
linux# zypper in -y podman podman-cni-config
```

8. Install lssci to view attached stoarge devices.

   lsscsi RPMs are included in the embedded repository in the CSM release and may be installed in your pre-LiveCD environment using `zypper` as shown in the following examples.

   Add the *embedded* repository.

```
linux# zypper ar -fG "./${CSM_RELEASE}/rpm/embedded" "${CSM_RELEASE}-embedded"
```

   Install `lsscsi` package.

```
linux# zypper in -y lsscsi
```

9. Install Git RPMs.

   Git RPMs are included in the embedded repository in the CSM release and may be installed in your pre-LiveCD environment using `zypper` as shown in the following examples.

Add the embedded repository.

```
linux# zypper ar -fG "./${CSM_RELEASE}/rpm/embedded" "${CSM_RELEASE}-embedded"
```

Install `lsscsi` package.

```
linux# zypper in -y git
```

10. Create bootable media.

a. Identify the USB device.

The following example shows the USB device is /dev/sdd on the host.

```
linux# lsscsi
[6:0:0:0]    disk    ATA      SAMSUNG MZ7LH480 404Q  /dev/sda
[7:0:0:0]    disk    ATA      SAMSUNG MZ7LH480 404Q  /dev/sdb
[8:0:0:0]    disk    ATA      SAMSUNG MZ7LH480 404Q  /dev/sdc
[14:0:0:0]   disk    SanDisk  Extreme SSD      1012  /dev/sdd
[14:0:0:1]   enclosu SanDisk  SES Device       1012  -
```

In the above example, the internal disks can be seen as the ATA devices and the USB as the disk or enclsou device. Since the SanDisk fits the profile, /dev/sdd is used as the disk.

b. Set a variable with the disk.

```
linux# export USB=/dev/sd<disk_letter>
```

c. Format the USB device.

On Linux using the CSI application:

```
linux# csi pit format $USB ~/${CSM_RELEASE}/cray-pre-install-toolkit-*.iso \
50000
```

d. Mount the configuration and persistent data partition.

```
linux# mkdir -pv /mnt/{cow,pitdata}
linux# mount -L cow /mnt/cow && mount -L PITDATA /mnt/pitdata
```

e. Copy and extract the tarball into the USB.

```
linux# cp -r ~/${CSM_RELEASE}.tar.gz /mnt/pitdata/
linux# tar -zxvf ~/${CSM_RELEASE}.tar.gz -C /mnt/pitdata/
```

11. Generate installation files.

Some files are needed for generating the configuration payload. New systems will need to create these files before continuing. Systems upgrading from Shasta v1.3 should prepare by gathering data from the exiting system.

Pull the following files into the current working directory:

- application-node-config.yaml (Optional)
- cabinets.yaml (Optional)
- hmn_connections.json
- ncn_metadata.csv
- switch_metadata.csv

- system_config.yaml

The optional `application-node-config.yaml` file may be provided for further defining of settings relating to how application nodes will appear in HSM. See the CSI usage for more information.

The optional `cabinets.yaml` file allows cabinet naming and numbering as well as some networking overrides (e.g. VLAN) which will allow systems on Shasta v1.3 to minimize changes to the existing system while migrating to Shasta v1.4. Refer to *Construct cabinets.yaml File* on page 168 for more information.

12. Change into the preparation directory.

```
linux# mkdir -pv /mnt/pitdata/prep
linux# cd /mnt/pitdata/prep
```

13. Generate the system configuration, reusing a parameter file.

If moving from a Shasta v1.3 system, the system_config.yaml file will not be available and this step should be skipped. Go to *15* on page 37.

The following files should be in the current directory.

```
linux# ls -1
application_node_config.yaml
cabinets.yaml
hmn_connections.json
ncn_metadata.csv
shasta_system_configs
switch_metadata.csv
system_config.yaml
```

14. Generate the system configuration.

```
linux# csi config init
```

A new directory matching `--system-name` argument will now exist in the working directory.

Set an environment variable so this system name can be used in later commands.

```
linux# export SYSTEM_NAME=eniac
```

15. (Required if moving from a Shasta V1.3 system) Generate the system configuration when a pre-existing parameter file is unavailable.

a. The following files should be in the current directory. `application_node_config.yaml` file is optional.

```
linux# ls -1
application_node_config.yaml
hmn_connections.json
ncn_metadata.csv
shasta_system_configs
switch_metadata.csv
```

Set an environment variable so this system name can be used in later commands.

```
linux# export SYSTEM_NAME=eniac
```

b. Generate the system configuration.

```
linux# csi config init \
    --bootstrap-ncn-bmc-user root \
```

```
    --bootstrap-ncn-bmc-pass changeme \
    --system-name ${SYSTEM_NAME}  \
    --mountain-cabinets 4 \
    --starting-mountain-cabinet 1000 \
    --hill-cabinets 0 \
    --river-cabinets 1 \
    --can-cidr 10.103.11.0/24 \
    --can-external-dns 10.103.11.113 \
    --can-gateway 10.103.11.1 \
    --can-static-pool 10.103.11.112/28 \
    --can-dynamic-pool 10.103.11.128/25 \
    --nmn-cidr 10.252.0.0/17 \
    --hmn-cidr 10.254.0.0/17 \
    --ntp-pool time.nist.gov \
    --site-domain dev.cray.com \
    --site-ip 172.30.53.79/20 \
    --site-gw 172.30.48.1 \
    --site-nic p1p2 \
    --site-dns 172.30.84.40 \
    --install-ncn-bond-members p1p1,p10p1 \
    --application-node-config-yaml application_node_config.yaml \
    --cabinets-yaml cabinets.yaml \
    --hmn-mtn-cidr 10.104.0.0/17 \
    --nmn-mtn-cidr 10.100.0.0/17
```

A new directory matching `--system-name` argument will now exist in the working directory.

After generating a configuration, particularly when upgrading from Shasta v1.3 a visual audit of the generated files for network data should be performed. Specifically, `<systemname>/networks/HMN_MTN.yaml` and `<systemname>/networks/NMN_MTN.yaml` files should be viewed to ensure that cabinet names, subnets and VLANs have been preserved for an upgrade to Shasta v1.4. Failure of these parameters to match will likely mean a re-installation or reprogramming of CDU switches and CMM VLANs.

16. Run the command "csi config init --help" to get more information about the parameters mentioned in the example command above and others which are available.

The application_node_config.yaml file is optional, but if you have one describing the mapping between prefixes in hmn_connections.csv that should be mapped to HSM subroles, you need to include a command line option to have it used.

The bootstrap-ncn-bmc-user and bootstrap-ncn-bmc-pass must match what is used for the BMC account and its password for the management NCNs.

Set site parameters (site-domain, site-ip, site-gw, site-nic, site-dns) for the information which connects the ncn-m001 (PIT) node to the site. The site-nic is the interface on this node connected to the site. If coming from Shasta v1.3, the information for all of these site parameters was collected.

There are other interfaces possible, but the install-ncn-bond-members are typically: p1p1,p10p1 for HPE nodes; p1p1,p1p2 for Gigabyte nodes; and p801p1,p801p2 for Intel nodes. If coming from Shasta v1.3, this information was collected for ncn-m001.

Set the three cabinet parameters (mountain-cabinets, hill-cabinets, and river-cabinets) to the number of each cabinet which are part of this system.

The starting cabinet number for each type of cabinet (for example, starting-mountain-cabinet) has a default that can be overridden. See the "csi config init --help".

For systems that use non-sequential cabinet id numbers, use cabinets-yaml to include the cabinets.yaml file. This file can include information about the starting ID for each cabinet type and number of cabinets which

have separate command line options, but is a way to explicitly specify the id of every cabinet in the system. This process is described [here](310-CABINETS.md).

An override to default cabinet IPv4 subnets can be made with the hmn-mtn-cidr and nmn-mtn-cidr parameters. These are also used to maintain existing configuration in a Shasta v1.3 system

Several parameters (can-gateway, can-cidr, can-static-pool, can-dynamic-pool) describe the CAN (Customer Access network). The can-gateway is the common gateway IP used for both spine switches and commonly referred to as the Virtual IP for the CAN. The can-cidr is the IP subnet for the CAN assigned to this system. The can-static-pool and can-dynamic-pool are the MetalLB address static and dynamic pools for the CAN. The can-external-dns is the static IP assigned to the DNS instance running in the cluster to which requests the cluster subdomain will be forwarded. The can-external-dns IP must be within the can-static-pool range.

Set ntp-pool to a reachable NTP server.

These warnings from "csi config init" for issues in hmn_connections.json can be ignored.

The node with the external connection (ncn-m001) will have a warning similar to this because its BMC is connected to the site and not the HMN like the other management NCNs. It can be ignored.

"Couldn't find switch port for NCN: x3000c0s1b0"

An unexpected component may have this message. If this component is an application node with an unusual prefix, it should be added to the application_node_config.yaml file and then rerun "csi config init". See the procedure to [create the application_node_config.yaml](308-APPLICATION-NODE-CONFIG.md) in the CSM repository.

```
{"level":"warn","ts":1610405168.8705149,"msg":"Found unknown source prefix! If
this is expected to be an Application node, please update
application_node_config.yaml","row":

{"Source":"gateway01","SourceRack":"x3000","SourceLocation":"u33","DestinationRa
ck":"x3002","DestinationLocation":"u48","DestinationPort":"j29"}}
```

If a cooling door is found in hmn_connections.json, there may be a message like the following. It can be safely ignored.

```
{"level":"warn","ts":1612552159.2962296,"msg":"Cooling door found, but xname
does not yet exist for cooling doors!","row":
      {"Source":"x3000door-Motiv","SourceRack":"x3000","SourceLocation":"
","DestinationRack":"x3000","DestinationLocation":"u36","DestinationPort":"j27"}
}
```

17. Check for workarounds in the `/opt/cray/csm/workarounds/csi-config` directory. If there are any workarounds in that directory, run those now. Each has its own instructions in their respective `README.md` files.

    If there are any workarounds in the directory, run those now.

    ```
    # Example
    linux# ls /opt/cray/csm/workarounds/csi-config
    casminst-999
    ```

18. Prepare a `SHASTA-CFG` repository on the system. Refer to *Update the SHASTA-CFG Repository* on page 213.

19. Pre-populate LiveCD Daemons Configuration and NCN Artifacts.

    Now that the configuration is generated, populating the LiveCD with generated files can now begin. This will enable SSH and other services with LiveCD starts.

a. Set system name and enter prep directory.

```
linux# export SYSTEM_NAME=eniac
```

b. Use CSI to populate the LiveCD, provide both the mount point and the CSI generated config directory.

```
linux# csi pit populate cow /mnt/cow/ ${SYSTEM_NAME}/
config-----------------------> /mnt/cow/rw/etc/sysconfig/network/config...OK
ifcfg-bond0------------------> /mnt/cow/rw/etc/sysconfig/network/ifcfg-
bond0...OK
ifcfg-lan0-------------------> /mnt/cow/rw/etc/sysconfig/network/ifcfg-
lan0...OK
ifcfg-vlan002----------------> /mnt/cow/rw/etc/sysconfig/network/ifcfg-
vlan002...OK
ifcfg-vlan004----------------> /mnt/cow/rw/etc/sysconfig/network/ifcfg-
vlan004...OK
ifcfg-vlan007----------------> /mnt/cow/rw/etc/sysconfig/network/ifcfg-
vlan007...OK
ifroute-lan0-----------------> /mnt/cow/rw/etc/sysconfig/network/ifroute-
lan0...OK
ifroute-vlan002--------------> /mnt/cow/rw/etc/sysconfig/network/ifroute-
vlan002...OK
CAN.conf---------------------> /mnt/cow/rw/etc/dnsmasq.d/CAN.conf...OK
HMN.conf---------------------> /mnt/cow/rw/etc/dnsmasq.d/HMN.conf...OK
NMN.conf---------------------> /mnt/cow/rw/etc/dnsmasq.d/NMN.conf...OK
mtl.conf---------------------> /mnt/cow/rw/etc/dnsmasq.d/mtl.conf...OK
statics.conf-----------------> /mnt/cow/rw/etc/dnsmasq.d/statics.conf...OK
conman.conf------------------> /mnt/cow/rw/etc/conman.conf...OK
```

c. (Optional) Set the hostname. Print it into the hostname file.

```
linux# echo "${SYSTEM_NAME}-ncn-m001-pit" >/mnt/cow/rw/etc/hostname
```

d. Unmount the overlay.

```
linux# umount /mnt/cow
```

e. Create directories needed for basecamp and the squashFS images.

```
linux# mkdir -p /mnt/pitdata/configs/
linux# mkdir -p /mnt/pitdata/data/{k8s,ceph}/
```

f. Copy basecamp data.

```
linux# csi pit populate pitdata ${SYSTEM_NAME} /mnt/pitdata/configs -b
data.json--------------------> /mnt/pitdata/configs/data.json...OK
```

g. Update CA Cert on the copied data.json file. Provide the path to the data.json, the path to customizations.yaml, and the sealed_secrets.key

```
linux# csi patch ca \
--cloud-init-seed-file /mnt/pitdata/configs/data.json \
--customizations-file /mnt/pitdata/prep/site-init/customizations.yaml \
--sealed-secret-key-file /mnt/pitdata/prep/site-init/certs/sealed_secrets.key
```

h. Copy k8s artifacts.

```
linux# csi pit populate pitdata ~/${CSM_RELEASE}/images/kubernetes/ /mnt/
pitdata/data/k8s/ -kiK
5.3.18-24.37-default-0.0.6.kernel----------------> /mnt/pitdata/data/
```

```
k8s/...OK
initrd.img-0.0.6.xz-------------------------------> /mnt/pitdata/data/
k8s/...OK
kubernetes-0.0.6.squashfs-------------------------> /mnt/pitdata/data/
k8s/...OK
```

i. Copy ceph/storage artifacts.

```
linux# csi pit populate pitdata ~/${CSM_RELEASE}/images/storage-ceph/ /mnt/
pitdata/data/ceph/ -kiC
5.3.18-24.37-default-0.0.5.kernel-----------------> /mnt/pitdata/data/
ceph/...OK
initrd.img-0.0.5.xz-------------------------------> /mnt/pitdata/data/
ceph/...OK
storage-ceph-0.0.5.squashfs-----------------------> /mnt/pitdata/data/
ceph/...OK
```

j. Unmount the data partition.

```
linux# cd; umount /mnt/pitdata
```

k. Quick the typescript session and copy the file to a location on another server for reference later.

```
linux# exit
```

20. Boot LiveCD.

a. Start a typescript on an external system.

```
external# script -a boot.livecd.$(date +%Y-%m-%d).txt
external# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

b. Confirm the IPMI credentials work for BMC by checking power status.

```
external# export SYSTEM_NAME=eniac
external# export username=root
external# export password=changeme
external# ipmitool -I lanplus -U $username -P $password -H ${SYSTEM_NAME}-
ncn-m001-mgmt chassis power status
```

c. Connect to the IPMI console.

Press return and login on the console as root.

```
external# ipmitool -I lanplus -U $username -P $password -H ${SYSTEM_NAME}-
ncn-m001-mgmt sol activate
eniac-ncn-m001 login: root
ncn-m001#
```

d. Reboot the system.

```
ncn-m001# reboot
```

21. Set the user name and password upon the first log in.

Set the user name of root and press return twice.

```
pit login: root
Password:           <-------just press Enter here for a blank password
You are required to change your password immediately (administrator enforced)
```

```
Changing password for root.
Current password:   <------- press Enter here, again, for a blank password
New password:       <------- type new password
Retype new password:<------- retype new password
Welcome to the CRAY Prenstall Toolkit (LiveOS)

Offline CSM documentation can be found at /usr/share/doc/metal (version: rpm -q
docs-csm-install)
```

22. Disconnect from IPMI console.

23. Login via SSH to the node.

```
external# ssh ${SYSTEM_NAME}-ncn-m001
ncn-m001 login: root
pit#
```

24. Start a typescript.

```
pit# script -af booted-csm-lived.$(date +%Y-%m-%d).txt
pit# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

25. Confirm the pit-release version.

```
pit# cat /etc/pit-release
VERSION=1.2.2
TIMESTAMP=20210121044136
HASH=75e6c4a
```

26. Mount the data partition.

```
pit# mount -L PITDATA
```

27. Start services.

```
pit# systemctl start nexus
pit# systemctl start basecamp
pit# systemctl start conman
```

28. Verify the system and check for failures.

```
pit:~ # csi pit validate --network
pit:~ # csi pit validate --services
```

- If dnsmasq is dead, restart it with `systemctl restart dnsmasq`.

The final output from validating the services should have information about the nexus and basecamp containers/images similar to the following example.

```
CONTAINER ID  IMAGE
COMMAND               CREATED        STATUS            PORTS   NAMES
ff7c22c6c6cb  dtr.dev.cray.com/sonatype/nexus3:3.25.0             sh -c $
{SONATYPE_...  3 minutes ago  Up 3 minutes ago          nexus
c7638b573b93  dtr.dev.cray.com/cray/metal-basecamp:
1.1.0-1de4aa6                        5 minutes ago  Up 5 minutes ago
basecamp
```

29. Follow the outputs directions for failed validations before moving on.

If this is a Shasta v1.3.x migration scenario, then now the Dell and Mellanox switches can be reconfigured with their new names, new IP addresses, and new configuration for v1.4. Refer to 23.21.10 Upgrades for Dell and Mellanox Switches.

7 Apply the UAN Patch

About this task

ROLE	System installer
OBJECTIVE	Apply a UAN update patch to the release tarball. This ensures that the latest UAN product artifacts are installed on the HPE Cray EX supercomputer.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in this release.

Procedure

1. Download the source release version (`uan-2.0.0.tar.gz`) of the UAN software package tarball and the wanted compressed patch `uan-2.0.0-uan-2.0.0.patch.gz` to the HPE Cray EX system.

2. Extract the source release distribution.

   ```
   ncn-w# tar -xzf uan-2.0.0.tar.gz
   ```

3. Uncompress the patch.

   ```
   ncn-w# gunzip uan-2.0.0-uan-2.0.0.patch.gz
   ```

4. Verify that the Git version is at least 2.16.5.

   The patch process is known to work with Git >= 2.16.5. Older versions of Git may not correctly apply the binary patch.

   ```
   ncn-w# git version
   git version 2.26.2
   ```

5. Apply the patch.

   ```
   ncn-w# git apply -p2 --whitespace=nowarn --directory=uan-2.0.0 \
   uan-2.0.0-uan-2.0.0.patch
   ```

6. Set UAN_RELEASE to reflect the new version.

   A different UAN release name can be chosen, if wanted.

   ```
   ncn-w# UAN_RELEASE="uan-2.0.0-day0-patch"
   ```

7. Update the name of the UAN release distribution directory.

   ```
   ncn-w# mv uan-2.0.0 $UAN_RELEASE
   ```

8. Create a tarball from the patched release distribution.

```
ncn-w# tar -cvzf ${UAN_RELEASE}.tar.gz "${UAN_RELEASE}/"
```

8 Install CSM

The following procedures detail how to install and configure CSM on a HPE Cray EX system.

8.1 CSM Deploy NCNs

About this task

ROLE

- System installer/system administrator

OBJECTIVE

The following procedure deploys Linux and Kubernetes software to the management NCNs. Deployment of the nodes starts with booting the storage nodes followed by the master nodes and worker nodes together.

After the operating system boots on each node, there are some configuration actions which take place. Watching the console or the console log for certain nodes can help to understand what happens and when. When the process completes for all nodes, the Ceph storage is initialized and the Kubernetes cluster is created and ready for a workload.

NEW IN THIS RELEASE

This entire procedure is new with this release.

WORKFLOW

The configuration workflow described here is intended to help understand the expected path for booting and configuring. See the actual steps below for the commands to deploy these management NCNs.

- Start watching the consoles for ncn-s001 and at least one other storage node.
- Boot all storage nodes at the same time.
- The first storage node ncn-s001 boots then starts a loop as ceph-ansible configuration waits for all other storage nodes to boot
- The other storage nodes boot and become passive. They are fully configured when ceph-ansible runs to completion on ncn-s001.
- After ncn-s001 notices that all other storage nodes are booted, ceph-ansible begins ceph configuration. This takes several minutes.
- After ceph-ansible finishes on ncn-s001, ncn-s001 waits for ncn-m002 to create `/etc/kubernetes/admin.conf`.
- Start watching the consoles for ncn-m002, ncn-m003 and at least one worker node.

- Boot master nodes (ncn-m002 and ncn-m003) and all worker nodes at the same time.
- The worker nodes boot and wait for ncn-m002 to create the `/etc/cray/kubernetes/join-command-control-plane` so they can join Kubernetes.
- The third master node ncn-m003 boots and waits for ncn-m002 to create the `/etc/cray/kubernetes/join-command-control-plane` so it can join Kubernetes.
- The second master node ncn-m002 boots, runs the kubernetes-cloudinit.sh, which creates `/etc/kubernetes/admin.conf` and `/etc/cray/kubernetes/join-command-control-plan`, and then waits for the storage node to create `etcd-backup-s3-credentials`.
- After ncn-s001 notices that ncn-m002 has created `/etc/kubernetes/admin.conf`, then ncn-s001 waits for any worker node to become available.
- After each worker node notices that ncn-m002 has created `/etc/cray/kubernetes/join-command-control-plan`, then it will join the Kubernetes cluster.
- When ncn-s001 notices this from any one of the worker nodes, it moves forward with creation of config maps and running the post-ceph playbooks (s3, OSD pools, quotas, etc.) .
- After ncn-s001 creates `etcd-backup-s3-credentials` during the benji-backups role, one of the last roles after ceph is set up, then ncn-m001 notices this and moves forward.

TIP: The timing of each set of boots varies based on hardware. Some manufacturers POST faster than others or vary based on BIOS setting. After powering a set of nodes on, an administrator can expect a healthy boot-session to take about 60 minutes depending on the number of storage and worker nodes.

Procedure

1. Create tokens.

   a. Copy the following tokens into the shell environment.

      Replace "changeme" with the real root password.

      ```
      pit# export mtoken='ncn-m(?!001)\w+-mgmt'
      pit# export stoken='ncn-s\w+-mgmt'
      pit# export wtoken='ncn-w\w+-mgmt'
      pit# export username=root
      pit# export IPMI_PASSWORD=changeme
      ```

2. Check for NCN pre-boot workarounds within the CSM tar file. If there are any workarounds in that directory, run them now. Each has its own instructions in their respective `README.md` files.

   ```
   pit# ls /opt/cray/csm/workarounds/before-ncn-boot
   CASMINST-980
   ```

3. Ensure that the PIT node has the current and correct time.

   ```
   pit# date "+%Y-%m-%d %H:%M:%S.%6N%z"
   ```

If the time is inaccurate, set the time manually.

```
pit# timedatectl set-time "2019-11-15 00:00:00"
```

Run the NTP script.

```
pit# /root/bin/configure-ntp.sh
```

4. Ensure the current time is set in BIOS for all management nodes.

   If each NCN is booted to the BIOS menu, check and set the current UTC time. Repeat this process for each NCN.

   a. Start an IPMI console session to the NCN.

      ```
      pit# bmc=ncn-w001-mgmt   <==Change this to be each node in turn.
      pit# conman -j $bmc
      ```

   b. Boot the node to BIOS in another terminal.

      ```
      pit# bmc=ncn-w001-mgmt  <== Change this to be each node in turn.
      pit# ipmitool -I lanplus -U $username -E -H $bmc chassis bootdev bios
      pit# ipmitool -I lanplus -U $username -E -H $bmc chassis power off
      pit# sleep 10
      pit# ipmitool -I lanplus -U $username -E -H $bmc chassis power on
      ```

      For HPE NCNs the above process boots the nodes to their BIOS, but the menu is unavailable through conman, as the node is booted into a graphical BIOS menu. To access the serial version of the BIOS setup, perform the ipmitool steps above to boot the node, and then in conman press ESC+9 key combination until the following menu is displayed in the console.

      ```
      For access via BIOS Serial Console:
      Press 'ESC+9' for System Utilities
      Press 'ESC+0' for Intelligent Provisioning
      Press 'ESC+!' for One-Time Boot Menu
      Press 'ESC+@' for Network Boot
      ```

      For HPE NCNs the date configuration menu can be found at the following path:

      ```
      System Configuration -> BIOS/Platform Configuration (RBSU) -> Date and Time
      ```

      Alternatively for HPE NCNs you can login to the BMC's web interface and access the HTML5 console for the node to interact with the graphical BIOS. From the administrators own machine create a SSH tunnel (-L creates the tunnel, and -N prevents a shell and stubs the connection).

      ```
      Change this to be each node in turn.
      linux# bmc=ncn-w001-mgmt
      linux# ssh -L 9443:$bmc:443 -N root@eniac-ncn-m001
      ```

      Opening a web browser to `https://localhost:9443` will give access to the BMC's web interface.

      When the node boots, the conman session can be used to see the BIOS menu to check and set the time to the current UTC time. The process varies on the vendor of the NCN.

      Repeat this process for each NCN.

5. Change the default root password and ssh keys. Refer to *Set Default Passwords During NCN Image Customization* on page 203.

6. Create boot directories for any NCN in DNS.

```
pit# /root/bin/set-sqfs-links.sh
```

7. Set each node to always UEFI network boot and ensure they are powered off.

```
 pit# grep -oP "($mtoken|$stoken|$wtoken)" /etc/dnsmasq.d/statics.conf \
| xargs -t -i ipmitool -I lanplus -U $username -E -H \
{} chassis bootdev pxe options=efiboot,persistent
pit# grep -oP "($mtoken|$stoken|$wtoken)" /etc/dnsmasq.d/statics.conf \
| xargs -t -i ipmitool -I lanplus -U $username -E -H {} power off
```

8. Validate that LiveCD is ready for installing NCNs.

```
pit# csi pit validate --livecd-preflight
```

9. Print consoles available.

```
pit# conman -q
ncn-m001-mgmt
ncn-m002-mgmt
ncn-m003-mgmt
ncn-s001-mgmt
ncn-s002-mgmt
ncn-s003-mgmt
ncn-w001-mgmt
ncn-w002-mgmt
ncn-w003-mgmt
```

10. Boot storage nodes.

```
pit# grep -oP $stoken /etc/dnsmasq.d/statics.conf | \
xargs -t -i ipmitool -I lanplus -U $username -E -H {} power on
```

Observe the installation through the ncn-s001-mgmt console. Print the console name and join the console.

```
pit# conman -q | grep s001
ncn-s001-mgmt
pit# conman -j ncn-s001-mgmt
```

Watch the storage node consoles carefully for error messages. If any are seen, refer to *Verify CEPH CSI* on page 205.

If the nodes have PXE boot issues, refer to *Troubleshoot PXE Boot* on page 177.

11. Once all storage nodes are up and ncn-s001 is running ceph-ansible, boot Kubernetes manager and worker nodes.

```
pit# \
grep -oP "($mtoken|$wtoken)" /etc/dnsmasq.d/statics.conf | \
xargs -t -i ipmitool -I lanplus -U $username -E -H {} power on
```

The installation can be observed through the ncn-m002-mgmt console. Print the console name and join the console.

```
pit# conman -q | grep m002
ncn-m002-mgmt
pit# conman -j ncn-m002-mgmt
```

If the nodes have pxe boot issues (e.g. getting pxe errors, not pulling the ipxe.efi binary) see [PXE boot troubleshooting](420-MGMT-NET-PXE-TSHOOT.md).

If one of the manager nodes seems hung waiting for the storage nodes to create a secret, check the storage node consoles for error messages. If any are found, consult [066-CEPH-CSI](066-CEPH-CSI.md.

If other issues arise, such as cloud-init (e.g. NCNs come up to linux with no hostname) see the CSM workarounds for fixes around mutual symptoms.

```
pit# ls /opt/cray/csm/workarounds/after-ncn-boot
CASMINST-1093
```

12. It should not take more than 60 minutes for the `kubectl get nodes` command to return output indicating that all the managers and workers aside from the LiveCD's node are `Ready`. Eventually, `kubectl get nodes` will return all the managers and workers aside from the LiveCD's node.

```
pit# ssh ncn-m002
ncn-m002# kubectl get nodes -o wide
NAME       STATUS   ROLES    AGE     VERSION   INTERNAL-IP   EXTERNAL-IP   OS-
IMAGE                                                KERNEL-VERSION
CONTAINER-RUNTIME
ncn-m002   Ready    master   14m     v1.18.6   10.252.1.5    <none>        SUSE
Linux Enterprise High Performance Computing 15 SP2   5.3.18-24.43-default
containerd://1.3.4
ncn-m003   Ready    master   13m     v1.18.6   10.252.1.6    <none>        SUSE
Linux Enterprise High Performance Computing 15 SP2   5.3.18-24.43-default
containerd://1.3.4
ncn-w001   Ready    <none>   6m30s   v1.18.6   10.252.1.7    <none>        SUSE
Linux Enterprise High Performance Computing 15 SP2   5.3.18-24.43-default
containerd://1.3.4
ncn-w002   Ready    <none>   6m16s   v1.18.6   10.252.1.8    <none>        SUSE
Linux Enterprise High Performance Computing 15 SP2   5.3.18-24.43-default
containerd://1.3.4
ncn-w003   Ready    <none>   5m58s   v1.18.6   10.252.1.12   <none>        SUSE
Linux Enterprise High Performance Computing 15 SP2   5.3.18-24.43-default
containerd://1.3.4
```

13. Apply post NCN boot workarounds.

a. Check for workarounds (WAR) in the `/var/www/ephemeral/prep/csm-x.x.x/fix/after-ncn-boot` directory. If there are any workarounds present in the directory, run them. Instructions can be found in the `README` files.

   The following shows an example of a workaround that may be present.

   ```
   pit# ls /opt/cray/csm/workarounds/after-ncn-boot
   casminst-12345
   ```

Patch the script.

```
pit# pdsh -w $(kubectl get nodes | grep -oP 'ncn\w\d+' | sort -u | tr -t '\n' ',') \
"sed -i -e 's:^for dev in $(ls /sys/class/net) ; do$:for dev in $(ls /sys/class/net | grep hsn) ; \
do:g' /opt/cray/cray-network-config/default/bin/shasta-network-lldp.sh"
```

Run the patched LLDP configuration script on NCNs.

```
pit# pdsh b -w $(kubectl get nodes | grep -oP 'ncn\w\d+' | sort -u | \
tr -t '\n' ',') '/bin/bash /opt/cray/cray-network-config/default/bin/shasta-network-lldp.sh'
```

14. Add cluster credentials to the LiveCD.

After 5-10 minutes, the first master node should be provisioning the other nodes in the cluster. At this time, credentials can be obtained.

Copy the Kubernetes config to the LiveCD.

This will always be the node that the `first-master-hostname` in the `/var/www/ephemeral/configs/data.json | jq` file. If provisioning from ncn-m001 then it is expected to obtain these from ncn-m002.

```
pit# mkdir ~/.kube
pit# scp ncn-m002.nmn:/etc/kubernetes/admin.conf ~/.kube/config
```

15. After the management NCNs are booted, the BGP peers will need to be checked and updated if the neighbor IPs are incorrect on the switches. Refer to *Configure BGP Neighbors on Management Switches* on page 146.

    BGP sessions should be cleared.

    - Aruba: `clear bgp *`
    - Mellanox: `clear ip bgp all`

    At this point all but one of the peering sessions with the BGP neighbors should be in IDLE or CONNECT state and not ESTABLISHED state. If the switch is an Aruba, one peering session will be established with the other switch. Check that all of the neighbor IPs are correct.

    The following helper scripts are available for the various switch types.

    ```
    pit# ls -1 /usr/bin/*peer*py
    /usr/bin/aruba_set_bgp_peers.py
    /usr/bin/mellanox_set_bgp_peers.py
    ```

16. Validate Ceph and Kubernetes configurations.

    The following commands will run a series of remote tests on the other nodes to validate they are healthy and configured correctly.

    a. Check Ceph.

    ```
    pit:~ # csi pit validate --ceph
    ```

    Refer to the Utility Storage section of the HPE Cray EX System Administration Guide S-8001, to help resolve any failed tests.

    b. Check Kubernetes.

    ```
    pit:~ # csi pit validate --k8s
    ```

    If test failures for "/dev/sdc" are observed they should be discarded for a manual test:

    ```
    # masters:
    ncn# blkid -L ETCDLVM
     # workers:
    ncn# blkid -L CONLIB
    ncn# blkid -L CONRUN
    ncn# blkid -L K8SLET
    ```

    The test should be looking for the ephemeral disk, that disk is sometimes `/dev/sdc`. The name of the disk is a more accurate test, and isn't prone to the random path change.

    If shell terminal is not echoing input after running this, type "reset" and press enter to recover.

17. Ensure that Kubernetes weave hasn't split-brained.

    Run the following command on each member of the Kubernetes cluster (master nodes and worker nodes) to ensure that weave is operating as a single cluster.

```
ncn# weave --local status connections  | grep failed
```

If IP allocation was seeded by different peers' is observed, the weave has been split-brained. At this point it is necessary to wipe the NCNs and start the PXE boot again.

18. Configure and trim UEFI entries.

a. Create the `boot-bios-selector` file(s) based on the manufacturer.

Gigabyte Technology

Masters

```
ncn-m# efibootmgr | grep -iP '(pxe ipv?4.*adapter)' | tee /tmp/bbs1
Boot0007* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:BE:8F:2E
Boot0009* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:BE:8F:
2F
ncn-m# efibootmgr | grep cray | tee /tmp/bbs2
Boot0000* cray (sda1)
Boot0002* cray (sdb1)
```

Storage

```
ncn-s# efibootmgr | grep -iP '(pxe ipv?4.*adapter)' | tee /tmp/bbs1
Boot0007* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:C7:11:FA
Boot0009* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:C7:11:FB
ncn-s# efibootmgr | grep cray | tee /tmp/bbs2
Boot0000* cray (sda1)
Boot0002* cray (sdb1)
```

Workers

```
ncn-w# efibootmgr | grep -iP '(pxe ipv?4.*adapter)' | tee /tmp/bbs1
Boot0007* UEFI: PXE IP4 Mellanox Network Adapter - 98:03:9B:AA:88:30
Boot0009* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:34:89:2A
Boot000B* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:34:89:2B
ncn-w# efibootmgr | grep cray | tee /tmp/bbs2
Boot0000* cray (sda1)
Boot0002* cray (sdb1)
```

Hewlett Packard Enterprise

Masters

```
ncn-m# efibootmgr | grep -i 'port 1' | grep -i 'pxe ipv4' | tee /tmp/bbs1
Boot0014* OCP Slot 10 Port 1 : Marvell FastLinQ 41000 Series - 2P 25GbE
SFP28 QL41232HQCU-HC OCP3 Adapter - NIC - Marvell FastLinQ 41000 Series - 2P
25GbE SFP28 QL41232HQCU-HC OCP3 Adapter - PXE (PXE IPv4)
Boot0018* Slot 1 Port 1 : Marvell FastLinQ 41000 Series - 2P 25GbE SFP28
QL41232HLCU-HC MD2 Adapter - NIC - Marvell FastLinQ 41000 Series - 2P 25GbE
SFP28 QL41232HLCU-HC MD2 Adapter - PXE (PXE IPv4)
ncn-m# efibootmgr | grep cray | tee /tmp/bbs2
Boot0021* cray (sdb1)
Boot0022* cray (sdc1)
```

Storage

```
ncn-s# efibootmgr | grep -i 'port 1' | grep -i 'pxe ipv4' | tee /tmp/bbs1
Boot001C* OCP Slot 10 Port 1 : Marvell FastLinQ 41000 Series - 2P 25GbE
SFP28 QL41232HQCU-HC OCP3 Adapter - NIC - Marvell FastLinQ 41000 Series - 2P
25GbE SFP28 QL41232HQCU-HC OCP3 Adapter - PXE (PXE IPv4)
```

```
Boot001D* Slot 1 Port 1 : Marvell FastLinQ 41000 Series - 2P 25GbE SFP28
QL41232HLCU-HC MD2 Adapter - NIC - Marvell FastLinQ 41000 Series - 2P 25GbE
SFP28 QL41232HLCU-HC MD2 Adapter - PXE (PXE IPv4)
ncn-s# efibootmgr | grep cray | tee /tmp/bbs2
Boot0002* cray (sdg1)
Boot0020* cray (sdh1)
```

Workers

```
ncn-w# efibootmgr | grep -i 'port 1' | grep -i 'pxe ipv4' | tee /tmp/bbs1
Boot0012* OCP Slot 10 Port 1 : Marvell FastLinQ 41000 Series - 2P 25GbE
SFP28 QL41232HQCU-HC OCP3 Adapter - NIC - Marvell FastLinQ 41000 Series - 2P
25GbE SFP28 QL41232HQCU-HC OCP3 Adapter - PXE (PXE IPv4)
ncn-w# efibootmgr | grep cray | tee /tmp/bbs2\
ncn-w# efibootmgr | grep cray | tee /tmp/bbs2
Boot0017* cray (sdb1)
Boot0018* cray (sdc1)
```

Intel Corporation

Masters

```
ncn-m# efibootmgr | grep -i 'ipv4' | grep -iv 'baseboard' | tee /tmp/bbs1
Boot000E* UEFI IPv4: Network 00 at Riser 02 Slot 01
Boot0014* UEFI IPv4: Network 01 at Riser 02 Slot 01
ncn-m# efibootmgr | grep -i 'cray' | tee /tmp/bbs2
Boot0011* cray (sda1)
Boot0012* cray (sdb1)
```

Storage

```
ncn-s# efibootmgr | grep -i 'ipv4' | grep -iv 'baseboard' | tee /tmp/bbs1
Boot000E* UEFI IPv4: Network 00 at Riser 02 Slot 01
Boot0012* UEFI IPv4: Network 01 at Riser 02 Slot 01
ncn-s# efibootmgr | grep -i 'cray' | tee /tmp/bbs2
Boot0014* cray (sda1)
Boot0015* cray (sdb1)
```

Workers

```
ncn-w# efibootmgr | grep -i 'ipv4' | grep -iv 'baseboard' | tee /tmp/bbs1
Boot0008* UEFI IPv4: Network 00 at Riser 02 Slot 01
Boot000C* UEFI IPv4: Network 01 at Riser 02 Slot 01
ncn-w# efibootmgr | grep -i 'cray' | tee /tmp/bbs2
Boot0010* cray (sda1)
Boot0011* cray (sdb1)
```

b. Set Order.

```
ncn-m# efibootmgr -o $(cat /tmp/bbs* | sed 's/^Boot//g' \
| awk '{print $1}' | tr -t '*' ',' | tr -d '\n' | sed 's/,$//') | grep -i
bootorder
BootOrder: 000E,0014,0011,0012
```

19. Trim down boot menu to only contain relevant entries.

a. Find the other PXE entries.

Gigabyte Technology

```
ncn# efibootmgr | grep -ivP '(pxe ipv?4.*)' | grep -iP '(adapter|
connection)' | tee /tmp/rbbs1
```

Hewlett Packard Enterprise

```
ncn# efibootmgr | grep -i 'port 1' | grep -vi 'pxe ipv4' | tee /tmp/rbbs1
```

Intel Corporation

```
ncn# efibootmgr | grep -vi 'ipv4' | grep -i 'baseboard' | tee /tmp/rbbs1
```

b. Remove the other PXE entries.

```
ncn# cat /tmp/rbbs* | sed 's/^Boot//g' | awk '{print $1}' | tr -d '*' |
xargs -t -i efibootmgr -b {} -B
```

8.2 Install CSM Platform

About this task

ROLE	• System installer/system administrator
OBJECTIVE	The following procedure for installing CSM applications and services.
NEW IN THIS RELEASE	This entire procedure is new with this release.

Procedure

1. Verify that Nexus is running.

```
pit# systemctl status nexus
```

2. Verify that Nexus is ready.

   Any HTTP response other than `200 OK` indicates Nexus is not ready.

```
pit# curl -sSif http://localhost:8081/service/rest/v1/status/writable
HTTP/1.1 200 OK
Date: Thu, 04 Feb 2021 05:27:44 GMT
Server: Nexus/3.25.0-03 (OSS)
X-Content-Type-Options: nosniff
Content-Length: 0
```

3. Load skopeo image installed by the cray-nexus RPM.

```
pit# podman load \
-i /var/lib/cray/container-images/cray-nexus/skopeo-stable.tar \
quay.io/skopeo/stable
```

4. Use `skopeo sync` to upload container images from the CSM release.

```
pit# podman run --rm --network host \
-v /var/www/ephemeral/${CSM_RELEASE}/docker/dtr.dev.cray.com:/images:ro quay.io/skopeo/stable sync \
```

```
--scoped --src dir --dest docker --dest-tls-verify=false --dest-creds admin:admin123 /images \
localhost:5000
```

5. Create `site-init` secret to contain `/var/www/ephemeral/prep/site-init/customizations.yaml`.

   The `site-init` secret in loftsman namespace makes `/var/www/ephemeral/prep/site-init/customizations.yaml` available to product installers.

   The `site-init` secret should only be updated when the corresponding `customizations.yaml` data is changed, such as during system installation or upgrade.

   Create the `site-init` secret to contain `/var/www/ephemeral/prep/site-init/customizations.yaml`.

   ```
   pit# kubectl create secret -n loftsman generic site-init --from-file=/var/www/
   ephemeral/prep/site-init/customizations.yaml
   secret/site-init created
   ```

6. Deploy the sealed secret decryption key.

   ```
   pit# /var/www/ephemeral/prep/site-init/deploy/deploydecryptionkey.sh
   ```

   An error similar to the following example may occur when deploying the key. This is expected and can be safely be ignored.

   ```
   pit# /var/www/ephemeral/prep/site-init/deploy/deploydecryptionkey.sh
   Error from server (NotFound): secrets "sealed-secrets-key" not found

   W0304 17:21:42.749101   29066 helpers.go:535] --dry-run is deprecated and can
   be replaced with --dry-run=client.
   secret/sealed-secrets-key created
   Restarting sealed-secrets to pick up new keys
   No resources found
   ```

7. Install CSM.

   ```
   pit# export SYSTEM_NAME=eniac
   ```

   ```
   pit# cd /var/www/ephemeral/$CSM_RELEASE
   pit# ./install.sh
   ```

   If the above commands were ran successfully, `install.sh` will output `OK` to stderr and exit with status code `0`.

   ```
   + CSM applications and services deployed
   install.sh: OK
   ```

8. Setup Nexus.

   a. Configure Nexus and upload CSM RPM repositories, container images, and HELM charts.

      ```
      pit# ./lib/setup-nexus.sh
      + Nexus setup complete
      setup-nexus.sh: OK
      ```

   In the event of an error, consult the known issues below to resolve potential problems and then try running `setup-nexus.sh` again. Note that subsequent runs of `setup-nexus.sh` may report `FAIL` when uploading

duplicate assets. This is ok as long as `setup-nexus.sh` outputs `setup-nexus.sh: OK` and exits with status code `0`.

Known Issues:

- [error: timed out waiting for the condition on jobs/cray-sls-init-load](#error-timed-out-sls-init-load-job)
- [Error: not ready: https://packages.local](#error-not-ready)
- [Error initiating layer upload ... in registry.local: received unexpected HTTP status: 200 OK] (#error-initiating-layer-upload)
- [Error lookup registry.local: no such host](#error-registry-local-no-such-host)

9. Set NCNs to use Unbound.

```
pit# ./lib/list-ncns.sh
+ Getting admin-client-auth secret
+ Obtaining access token
+ Querying SLS
ncn-m001
ncn-m002
ncn-m003
ncn-s001
ncn-s002
ncn-s003
ncn-w001
ncn-w002
ncn-w003
```

If any NCNs are missing from the output, take corrective action before proceeding.

10. SSH to each NCn and update `/etc/resolc.conf` to use Unbound as the nameserver.

If password-less SSH is not configured, the administrator will have to enter the corresponding password as the script attempts to connect to each NCN.

```
pit# ./lib/set-ncns-to-unbound.sh
+ Getting admin-client-auth secret
+ Obtaining access token
+ Querying SLS
+ Updating ncn-m001
Password:
ncn-m001: nameserver 127.0.0.1
ncn-m001: nameserver 10.92.100.225
+ Updating ncn-m002
Password:
ncn-m002: nameserver 10.92.100.225
+ Updating ncn-m003
Password:
ncn-m003: nameserver 10.92.100.225
+ Updating ncn-s001
Password:
ncn-s001: nameserver 10.92.100.225
+ Updating ncn-s002
Password:
ncn-s002: nameserver 10.92.100.225
+ Updating ncn-s003
Password:
ncn-s003: nameserver 10.92.100.225
+ Updating ncn-w001
```

```
Password:
ncn-w001: nameserver 10.92.100.225
+ Updating ncn-w002
Password:
ncn-w002: nameserver 10.92.100.225
+ Updating ncn-w003
Password:
ncn-w003: nameserver 10.92.100.225
```

11. Validate CSM Install.

The administrator should wait at least 15 minutes to let the various Kubernetes resources get initialized and started. After waiting, the administrator can start the CSM validation process. Refer to *CSM Health Checks and Install Validation* on page 184.

12. Add compute cabinet routing to NCNs.

```
pit# /opt/cray/csm/workarounds/livecd-post-reboot/CASMINST-1570/CASMINST-1570.sh
```

8.3 CSM Install Reboot

Prerequisites

These services must be healthy in Kubernetes before the reboot of the LiveCD can take place.

- cray-dhcp-kea
- cray-dns-unbound
- cray-bss
- cray-sls
- cray-s3
- cray-ipxe
- cray-tftp

About this task

The following procedure contains information for rebooting and deploying the non-compute node that is currently hosting the LiveCD. The following steps detail how an administrator through loading hand-off data and rebooting the node. This assists with remote-console setup to aide in observing the reboot. At the end of this procedure, the LiveCD will no longer be active. The node it was using will join the Kubernetes cluster as the final of three master nodes forming a quorum.

Important: While the node is rebooting, it will be available only through Serial-over-LAN and local terminals. This procedure entails deactivating the LiveCD, meaning the LiveCD and all of it's resources will be unavailable.

Procedure

1. Check for workarounds in the `/var/www/ephemeral/${CSM_RELEASE}/fix/livecd-pre-reboot` directory.

If workarounds are present, run them now.

```
pit# ls /opt/cray/csm/workarounds/livecd-pre-reboot
casminst-435
```

2. Upload SLS file.

```
pit# csi upload-sls-file \
--sls-file /var/www/ephemeral/prep/${SYSTEM_NAME}/sls_input_file.json
2021/02/02 14:05:15 Retrieving S3 credentails ( sls-s3-credentials ) for SLS
2021/02/02 14:05:15 Uploading SLS file: /var/www/ephemeral/prep/surtur/
sls_input_file.json
2021/02/02 14:05:15 Successfully uploaded SLS Input File.
```

3. Obtain token for authenticated communication with the gateway.

```
pit# export TOKEN=$(curl -k -s -S -d grant_type=client_credentials \
-d client_id=admin-client \
-d client_secret=`kubectl get secrets admin-client-auth \
-o jsonpath='{.data.client-secret}' | base64 -d` \
https://api-gw-service-nmn.local/keycloak/realms/shasta/protocol/openid-connect/
token \
| jq -r '.access_token')
```

4. Upload `data.json` file used to BSS.

```
pit# csi handoff bss-metadata --data-file /var/www/ephemeral/configs/data.json
```

5. Upload NCN artifacts.

   Fill CSM_RELEASE with release tarball.

```
pit# export CSM_RELEASE=csm-x.y.z
pit# export artdir=/var/www/ephemeral/${CSM_RELEASE}/images
pit# csi handoff ncn-images \
     --k8s-kernel-path $artdir/kubernetes/*.kernel \
     --k8s-initrd-path $artdir/kubernetes/initrd.img*.xz \
     --k8s-squashfs-path $artdir/kubernetes/kubernetes*.squashfs \
     --ceph-kernel-path $artdir/storage-ceph/*.kernel \
     --ceph-initrd-path $artdir/storage-ceph/initrd.img*.xz \
     --ceph-squashfs-path $artdir/storage-ceph/storage-ceph*.squashfs
```

6. List ipv4 boot options using `efibootmgr`.

```
pit# efibootmgr | grep -Ei "ip(v4|4)"
```

7. Set `efibootmgr` for booting next from Port-1 of Riser-1.

```
pit# efibootmgr | grep -i ipv4
Boot0005* UEFI IPv4: Network 00 at Riser 02 Slot 01
Boot0007* UEFI IPv4: Network 01 at Riser 02 Slot 01
Boot000A* UEFI IPv4: Intel Network 00 at Baseboard
Boot000C* UEFI IPv4: Intel Network 01 at Baseboard
```

8. Use `efibootmgr` to set next boot device to port selected in the previous step.

```
pit# efibootmgr -n $PXEPORT 2>&1 | grep -i BootNext
```

9. (Optional) Set up conman or serial console.

```
external# script -a boot.livecd.$(date +%Y-%m-%d).txt
external# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
external# SYSTEM_NAME=eniac
external# username=root
external# IPMI_PASSWORD=changeme
external# ipmitool -I lanplus -U $username -E -H ${SYSTEM_NAME}-ncn-m001-mgmt \
chassis power status
external# ipmitool -I lanplus -U $username -E -H ${SYSTEM_NAME}-ncn-m001-mgmt \
sol activate
```

10. Collect CAN IPs for logging into other NCNs.

```
pit# ssh ncn-m002
ncn-m002# ip a show vlan007 | grep inet
inet 10.102.11.13/24 brd 10.102.11.255 scope global vlan007
inet6 fe80::1602:ecff:fed9:7820/64 scope link
```

11. Login from another machine to verify that IP is usable.

```
external# ssh root@10.102.11.13
ncn-m002#
```

12. Wipe the node beneath the LiveCD.

    Erasing the RAIDs labels will trigger a fresh partition table to deploy.

    a. Select disks to wipe.

    ```
    pit# md_disks="$(lsblk -l -o SIZE,NAME,TYPE,TRAN | grep -E '(sata|nvme|sas)'
    | sort -h | awk '{print "/dev/" $2}')"
    ```

    b. Print disks into typescript or console.

    ```
    pit# echo $md_disks
    ```

    c. Wipe the disks. **THIS IS IRREVERSIBLE**.

    ```
    pit# wipefs --all --force $md_disks
    /dev/sda: 8 bytes were erased at offset 0x00000200 (gpt): 45 46 49 20 50 41
    52 54
    /dev/sda: 8 bytes were erased at offset 0x6fc86d5e00 (gpt): 45 46 49 20 50
    41 52 54
    /dev/sda: 2 bytes were erased at offset 0x000001fe (PMBR): 55 aa
    /dev/sdb: 6 bytes were erased at offset 0x00000000 (crypto_LUKS): 4c 55 4b
    53 ba be
    /dev/sdb: 6 bytes were erased at offset 0x00004000 (crypto_LUKS): 53 4b 55
    4c ba be
    /dev/sdc: 8 bytes were erased at offset 0x00000200 (gpt): 45 46 49 20 50 41
    52 54
    /dev/sdc: 8 bytes were erased at offset 0x6fc86d5e00 (gpt): 45 46 49 20 50
    41 52 54
    /dev/sdc: 2 bytes were erased at offset 0x000001fe (PMBR): 55 aa
    ```

13. Save the typescript.

    Quit the typescript with the `exit` command and copy the file (booted-csm-lived.<date>.txt) to a location on another server for reference later.

```
pit# exit
```

14. Reboot the LiveCD.

```
pit# reboot
```

15. Obtain the hostname of the booted node.

a. (Optional) Create a directory to copy network config files to.

Only perform this step if m001 booted without a hostname or didn't run all the cloud-init scripts.

```
ncn-m001# mkdir /mnt/cow
```

b. Mount the USB.

```
ncn-m001# mount -L cow /mnt/cow
```

c. Copy network config files.

```
ncn-m001# cp -pv /mnt/cow/rw/etc/sysconfig/network/ifroute* /etc/sysconfig/
network/
# cp -pv /mnt/cow/rw/etc/sysconfig/network/ifcfg-lan0 /etc/sysconfig/network/
```

d. Run the dhcp to static script.

```
ncn-m001# /srv/cray/scripts/metal/set-dhcp-to-static.sh
```

e. Pull all required cloud-init data for the NCN to join the cluster.

```
ncn-m001# cloud-init clean
ncn-m001# cloud-init init
ncn-m001# cloud-init modules -m init
ncn-m001# cloud-init modules -m config
ncn-m001# cloud-init modules -m final
```

16. Login and start a typescript to capture the output of running 008.

```
external# ssh root@10.102.11.13
ncn-m002# ssh ncn-m001
ncn-m001# script -a verify.csm.$(date +%Y-%m-%d).txt
ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

17. (Optional) Change the root password on ncn-m001 to match the other management NCNs.

This step is optional and is only needed when the other management NCNs passwords were customized during the CSM Deploy NCNs procedure. If the management NCNs still have the default password this can be skipped.

```
ncn-m001# passwd
```

18. Verify each NCN is part of the Kubernetes cluster and reports as ready.

```
ncn-m001:~ # kubectl get nodes
NAME       STATUS   ROLES    AGE     VERSION
ncn-m001   Ready    master   7s      v1.18.6
ncn-m002   Ready    master   4h40m   v1.18.6
ncn-m003   Ready    master   4h38m   v1.18.6
ncn-w001   Ready    <none>   4h39m   v1.18.6
```

```
ncn-w002   Ready    <none>   4h39m   v1.18.6
ncn-w003   Ready    <none>   4h39m   v1.18.6
```

19. Set CSM Release.

    ```
    ncn# export CSM_RELEASE=csm-x.y.z
    ```

20. Create directories.

    ```
    ncn# mkdir -pv /mnt/livecd /mnt/rootfs /mnt/sqfs /mnt/pitdata
    ```

21. Mount the rootfs.

    ```
    ncn# mount -L PITDATA /mnt/pitdata
    ncn# mount /mnt/pitdata/${CSM_RELEASE}/cray-pre-install-toolkit-*.iso /mnt/
    livecd/
    ncn# mount /mnt/livecd/LiveOS/squashfs.img /mnt/sqfs/
    ncn# mount /mnt/sqfs/LiveOS/rootfs.img /mnt/rootfs/
    ```

22. Invoke CSI usage to validate that it runs and is ready for use.

    ```
    ncn# /mnt/rootfs/usr/bin/csi --help
    ```

23. Copy the CSI binary and CSM workaround documentation off to `tmp/`.

    ```
    ncn# cp -pv /mnt/rootfs/usr/bin/csi /tmp/csi
    ```

24. Restore and verify the site link.

    It is necessary to restore the `ifcfg-lan0` file from either the manual backup or by remounting the USB and coping it from the prep directory to `/etc/sysconfig/network/`.

    The following example assumes that the USB stick has been remounted.

    ```
    ncn-m001# export SYSTEM_NAME=eniac
    ncn-m001# cp /mnt/pitdata/prep/${SYSTEM_NAME}/pit-files/ifcfg-lan0 /etc/
    sysconfig/network/
    ncn-m001# cp /mnt/pitdata/prep/${SYSTEM_NAME}/pit-files/ifroute-lan0 /etc/
    sysconfig/network/
    ncn-m001# cp /mnt/pitdata/prep/${SYSTEM_NAME}/pit-files/ifroute-vlan002 /etc/
    sysconfig/network/
    ncn-m001# wicked ifreload lan0
    ```

25. Verify site link.

    ```
    ncn-m001# ip a show lan0
    ```

26. Verify all VLANs have IPs.

    ```
    ncn-m001# ip a show vlan002
    ncn-m001# ip a show vlan004
    ncn-m001# ip a show vlan007
    ```

27. Verify that there is no metal bootstrap IP.

    ```
    ncn-m001# ip a show bond0
    ```

28. Enable NCN disk wiping safeguard.

```
pit# export TOKEN=$(curl -k -s -S -d grant_type=client_credentials \
  -d client_id=admin-client \
  -d client_secret=`kubectl get secrets admin-client-auth -o
jsonpath='{.data.client-secret}' | base64 -d` \
  https://api-gw-service-nmn.local/keycloak/realms/shasta/protocol/openid-
connect/token | jq -r '.access_token')
```

29. Update BSS.

```
pit# /tmp/csi handoff bss-update-param --set metal.no-wipe=1
```

30. Download and install/upgrade the workaround and documentation RPMs.

```
ncn-m001# rpm -Uvh https://storage.googleapis.com/csm-release-public/shasta-1.4/
docs-csm-install/docs-csm-install-latest.noarch.rpm
ncn-m001# rpm -Uvh https://storage.googleapis.com/csm-release-public/shasta-1.4/
csm-install-workarounds/csm-install-workarounds-latest.noarch.rpm
```

31. Check for workarounds in the `/opt/cray/csm/workarounds/livecd-post-reboot` directory.

```
ncn-m001# ls /opt/cray/csm/workarounds/livecd-post-reboot
```

32. The administrator can now refer to *NCN and Management Node Locking* on page 241 and *Manage Firmware Updates with FAS* on page 206.

33. After deploying the LiveCD's NCN, the LiveCD USB is unharmed and available to an administrator.

    Mount and view the USB stick.

```
ncn-m001# mkdir -pv /mnt/{cow,pitdata}
ncn-m001# mount -L cow /mnt/cow
ncn-m001# mount -L PITDATA /mnt/pitdata
ncn-m001# ls -ld /mnt/cow/rw/*
drwxr-xr-x  2 root root 4096 Jan 28 15:47 /mnt/cow/rw/boot
drwxr-xr-x  8 root root 4096 Jan 29 07:25 /mnt/cow/rw/etc
drwxr-xr-x  3 root root 4096 Feb  5 04:02 /mnt/cow/rw/mnt
drwxr-xr-x  3 root root 4096 Jan 28 15:49 /mnt/cow/rw/opt
drwx------ 10 root root 4096 Feb  5 03:59 /mnt/cow/rw/root
drwxrwxrwt 13 root root 4096 Feb  5 04:03 /mnt/cow/rw/tmp
drwxr-xr-x  7 root root 4096 Jan 28 15:40 /mnt/cow/rw/usr
drwxr-xr-x  7 root root 4096 Jan 28 15:47 /mnt/cow/rw/var
ncn-m001# ls -ld /mnt/pitdata/*
drwxr-xr-x  2 root root        4096 Feb  3 04:32 /mnt/pitdata/configs
drwxr-xr-x 14 root root        4096 Feb  3 07:26 /mnt/pitdata/csm-0.7.29
-rw-r--r--  1 root root 22159328586 Feb  2 22:18 /mnt/pitdata/csm-0.7.29.tar.gz
drwxr-xr-x  4 root root        4096 Feb  3 04:25 /mnt/pitdata/data
drwx------  2 root root       16384 Jan 28 15:41 /mnt/pitdata/lost+found
drwxr-xr-x  5 root root        4096 Feb  3 04:20 /mnt/pitdata/prep
drwxr-xr-x  2 root root        4096 Jan 28 16:07 /mnt/pitdata/static
```

    Unmount the USB before ejecting it.

```
ncn-m001# umount /mnt/cow /mnt/pitdata
```

(Optional) If a USB drive was used for the install of the CSM product, the USB drive can be used to create space for installation of other product streams. This allows for the installation to occur without having to monitor disk usage on ncn-m001.

```
ncn-m001:~ # mkdir -p /var/www/ephemeral
ncn-m001:~ # mount -L PITDATA /var/www/ephemeral/

linux # cd 1.4
linux # rsync -av * root@hostname-m001:/var/www/ephemeral
```

If this step is not used, it is important to monitor disk usage during the product installs to ensure the disk on ncn-m001 doesn’t fill up.

9 Install System Dump Utility (SDU) Product Stream

Prerequisites

- CSM is installed and verified
- Cray-product-catalog is running

About this task

ROLE

- System installer/system administrator

LIMITATIONS

Piping SDU command output to the less or more commands causes formatting problems. This is a known issue when using podman exec with the `--tty` option. Refreshing the less or more will workaround this issue. Less and more have the following commands to refresh the screen using `CTRL-r`, `CTRL-l`, or `r`.

OBJECTIVE

The following procedure details how to install the SDU product stream, which has its own independent release distribution and installer. The SDU release distribution is a gzipped tar file that includes the SDU, the installation script, and libraries required to install it.

System Dump Utility (SDU) - Is responsible for gathering all necessary information needed to debug a significant majority of problems. "Gathering" can be extended to imply:

- Collection of multiple data sources across either a stand-alone or distributed system in a scalable manner, with the ability to tailor collection targets and time bounds
- Step-wide analysis of collected data to make assertions about a running product or platform, such as indicating that specific hardware is suspect.
- Presentation of system collection data for operator or programmatic analysis, such as triage tools and customer support summary tools.
- Capable of both exporting a portable collection to a POSIX file system and uploading a collection from a customer system into the Metis call-home data lake via RDA

Remote Device Access (RDA) - HPE RDA (Remote Device Access) provides integrated remote connectivity for support automation, device telemetry and remote service delivery. RDA is integrated with the SDU product stream.

- HPE RDA Documentation: *https://midway.ext.hpe.com/home*

NEW IN THIS RELEASE

This entire procedure is new with this release.

Procedure

1. Start a typescript to capture the commands and output from this installation.

SDU is intended to be started on only one non-compute management node (ncn-m00x). However, it is staged and ready to be started on all management nodes for fail over support. Typically, ncn-m001 is chosen as the node to run SDU and will be used in the following examples.

```
ncn-m001# script -af product-sdu.$(date +%Y-%m-%d).txt
ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Copy the release distribution gzipped tar file (e.g cray-sdu-rda-1.0.11.tar.gz) to ncn-m001.

3. Unzip and extract the release distribution.

```
ncn-m001 # tar xzvf cray-sdu-rda-1.0.11.tar.gz
```

4. Change to the extracted release distribution directory.

```
ncn-m001 # cd cray-sdu-rda-1.0.11
```

5. Run the installer.

```
ncn-m001 # ./install.sh
```

6. Add the `cray-sdu-rda` nexus repository to zypper.

```
ncn-m001# zypper addrepo -Gf https://packages.local/repository/cray-sdu-rda
cray-sdu-rda
```

   The following error message can be ignored, " Repository named '`cray-sdu-rda`' already exists. Please use another alias".

7. Refresh the local metadata cache.

```
ncn-m001# zypper refresh -f cray-sdu-rda
```

8. Install the `cray-sdu-rda` RPM.

```
ncn-m001# zypper install -y cray-sdu-rda
```

9. Enable the `cray-sdu-rda` systemd service.

```
ncn-m001 # systemctl enable cray-sdu-rda
```

10. Restart the `cray-sdu-rda` systemd service.

```
ncn-m001 # systemctl start cray-sdu-rda
```

11. Verify that SDU is ready for use.

    - Poll for ready.

      ```
      ncn-m001 # sdu is_service_ready
      ```

    - Wait (block) for ready.

      ```
      ncn-m001 # sdu wait_for_service 30
      ```

Starting the `cray-sdu-rda` service is asynchronous (i.e. the command returns before the service is fully started). Downloading the podman image is part of starting the container and may take several minutes to come back as ready. The above commands help to handle this situation.

Alternatively, issue the following to block until the service/container is ready or fails. An optional timeout in seconds argument may be provided. The default timeout is 30 seconds.

```
ncn-m001 # sdu wait_for_service 30
```

The following error may occur:

```
Error: error inspecting object: no such container cray-sdu-rda
```

If the above error is encountered, re-run the command to wait for the container to complete building.

```
ncn-m001 # sdu wait_for_service 30
```

If the error message still occurs after 5 minutes, run the following command to gather additional information.

```
ncn-m001# journalctl -f -u cray-sdu-rda
```

12. Configure SDU with system specific information.

The sdu setup command will ask for information, such as system name, serial number, system type, system description, product number, company name, company site, and country code.

See "sdu setup --help" for all of the information which will be requested.

- Start an interactive installation session.

  ```
  ncn-m001 # sdu setup
  ```

- Alternatively, non-blocking setup is provided by command line arguments. In particular, the --batch option. Run the following for more information about options that can be passed with the --batch option. Refer to `sdu setup --help` at the command line for more info on what will need to be input.

  ```
  ncn-m001 # sdu bash
  ncn-m001-sdu: # sdu setup --help
  ```

13. Verify configuration file is good.

```
ncn-m001 # sdu --check_conf
[stdout] INFO Configuration file
"/etc/opt/cray/sdu/sdu.conf" and CLI Options Valid.
```

14. Verify that a bash session within the SDU container can be started.

Verify ncn-m001-sdu or equivalent command prompt prior to exiting the container.

```
ncn-m001 # sdu bash
```

Verify "ncn-m001-sdu" or equivalent comment prompt prior to exiting the container.

```
ncn-m001-sdu:# exit
```

15. Verify bash commands can be executed in the context of the SDU container.

```
ncn-m001 # sdu bash ls
```

16. Verify the SDU man page and setup help.

a. Verify that the SDU man-page can be viewed from the NCN.

```
ncn-m001 # man sdu
```

b. Verify that the main SDU man-page and setup can be viewed within the container.

```
ncn-m001 # sdu bash man sdu
ncn-m001# sdu bash
ncn-m001-sdu: # man sdu
ncn-m001-sdu: # sdu setup --help
ncn-m001-sdu: # exit
```

17. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

10 Install the System Monitoring Application Product Stream

Prerequisites

- CSM is installed and verified.
- gitea, cray-cfs-api, cray-cfs-operator, and cray-ims are running.

About this task

ROLE	• System administrator
OBJECTIVE	Install the System Monitoring Application (SMA) product stream which now has its own independent release distribution and installer.
LIMITATIONS	None
NEW IN THIS RELEASE	This entire procedure is new with this release.

Procedure

1. Start a typescript to capture the commands and output from this installation.

```
ncn-m001# script -af product-sma.$(date +%Y-%m-%d).txt
ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Copy the release distribution gzipped tar file, e.g. `sma-1.4.14.tar.gz`, to **`ncn-m001`**.

3. Extract the SMA release distribution.

```
ncn-m001# tar xvfz sma-1.4.14.tar.gz
```

4. Change directories to the extracted directory.

```
ncn-m001# cd sma-1.4.14
```

5. Run the **`./configure_sma_pvc.sh script`** as shown in the following example to configure PVCs that are appropriately sized for the available storage. The script will generate and update **`customizations.yaml`** with the appropriate PVC sizes for Kafka, Postgres, and ElasticSearch.

```
ncn-m001# ./configure_sma_pvc.sh
report 4050606990
Fri Feb 19 22:06:20 UTC 2021 - Raw storage = 46089092726784 bytes
Fri Feb 19 22:06:20 UTC 2021 - SMF raw storage = 39492022960128 bytes
Fri Feb 19 22:06:20 UTC 2021 - SMF quota bytes = 19746011480064 bytes
```

```
Fri Feb 19 22:06:20 UTC 2021 - SMF quota GiB = 18389GiB
Fri Feb 19 22:06:20 UTC 2021 - Configure ceph SMF pool quota to 18389GiB
Fri Feb 19 22:06:20 UTC 2021 - Total available for kafka, postgres, and
elasticsearch = 18279GiB
Fri Feb 19 22:06:20 UTC 2021 - Backup ./build/customizations.yaml to ./build/
customizations.yaml.bak
Fri Feb 19 22:06:20 UTC 2021 - Update ./build/customizations.yaml with new
values
Fri Feb 19 22:06:20 UTC 2021 - Changing Kafka PVC size from 65Gi to 1218GiB
Fri Feb 19 22:06:20 UTC 2021 - Changing Postgres PVC size from 1.6Ti to 3655GiB
Fri Feb 19 22:06:20 UTC 2021 - Changing ElasticSearch PVC size from 188Gi to
2611GiB
```

6. Run the **`./install.sh`** script.

```
ncn-m001# ./install.sh
```

> **NOTE:** The SMA product stream installation may possibly fail deploying the **`sma-monasca`** Helm chart. Check the output and if the **`sma-monasca`** Helm chart failed to deploy, perform the following steps. After the following steps are successfully performed the SMA product stream installation will be complete.

7. Uninstall the **`sma-monasca`** helm chart.

```
ncn-m001# helm -n sma uninstall sma-monasca
release "sma-monasca" uninstalled
```

8. On another terminal, delete the **`sma-monasca`** jobs as they are started to allow the Helm chart uninstall to complete.

```
ncn-m001# kubectl -n sma get jobs
NAME                          COMPLETIONS   DURATION   AGE
mysql-init-job1               1/1           4s         9h
sma-ldms-config               1/1           21s        9h
sma-monasca-mysql-init-job    0/1           5s         5s
sma-pgdb-cron-1615522200      1/1           4s         3h42m
sma-pgdb-init-job1            1/1           3s         9h

ncn-m001# kubectl -n sma delete job \
sma-monasca-mysql-init-job
job.batch "sma-monasca-mysql-init-job" deleted

ncn-m001# kubectl -n sma get jobs
NAME                           COMPLETIONS   DURATION   AGE
mysql-init-job1                1/1           4s         9h
sma-ldms-config                1/1           21s        9h
sma-monasca-alarms-init-job    0/1           2s         2s
sma-pgdb-cron-1615522200       1/1           4s         3h42m
sma-pgdb-init-job1             1/1           3s         9h

ncn-m001# kubectl -n sma delete job \
sma-monasca-alarms-init-job
job.batch "sma-monasca-alarms-init-job" deleted

ncn-m001# kubectl -n sma get jobs
NAME                            COMPLETIONS   DURATION   AGE
mysql-init-job1                 1/1           4s         9h
sma-ldms-config                 1/1           21s        9h
sma-monasca-cleanup-job-qlvc1   0/1                      3s
```

```
sma-pgdb-cron-1615522200        1/1           4s          3h42m
sma-pgdb-init-job1              1/1           3s          9h

ncn-m001# kubectl -n sma delete job \
sma-monasca-cleanup-job-qlvc1
job.batch "sma-monasca-cleanup-job-qlvc1" deleted

ncn-m001# kubectl -n sma get jobs
NAME                        COMPLETIONS   DURATION   AGE
mysql-init-job1             1/1           4s         9h
sma-ldms-config             1/1           21s        9h
sma-pgdb-cron-1615522200    1/1           4s         3h46m
sma-pgdb-init-job1          1/1           3s         9h
```

9. Create a Loftsman manifest file that only contains the **sma-monasca** Helm chart.

```
ncn-m001# cat sma-monasca-only.yaml
apiVersion: manifests/v1beta1
metadata:
  name: sma-manifest
spec:
  charts:
  - name: sma-monasca
    namespace: sma
    values:
      mysql:
        imagesHost: dtr.dev.cray.com
        persistence:
          size: 5Gi
          storageClass: sma-block-replicated
    version: 1.6.1
```

10. Redeploy the sma-monasca Helm chart using Loftsman.

```
ncn-m001# loftsman ship --manifest-path sma-monasca-only.yaml \
--charts-path <PATH_TO_SMA_PRODUCT_STREAM>/helm
2021-03-12T07:58:56Z INF Initializing the connection to the Kubernetes cluster
using KUBECONFIG (system default), and context (current-context) command=ship
2021-03-12T07:58:56Z INF Initializing helm client object command=ship
         |\
         | \
         |  \
         |___\      Shipping your Helm workloads with Loftsman
       \--||___/
  ~~~~~~\_____/~~~~~~~

2021-03-12T07:58:56Z INF Ensuring that the loftsman namespace exists
command=ship
2021-03-12T07:58:56Z INF Loftsman will use the packaged charts at helm as the
Helm install source command=ship
2021-03-12T07:58:56Z INF Running a release for the provided manifest at build/
manifests/sma-monasca-only.yaml command=ship

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Releasing sma-monasca v1.6.1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

2021-03-12T07:58:56Z INF Found value overrides for chart, applying:
mysql:
  imagesHost: dtr.dev.cray.com
```

```
  persistence:
    size: 5Gi
    storageClass: sma-block-replicated
 chart=sma-monasca command=ship namespace=sma version=1.6.1
2021-03-12T07:58:56Z INF Running helm install/upgrade with arguments: upgrade --
install sma-monasca helm/sma-monasca-1.6.1.tgz --namespace sma --create-
namespace --set global.chart.name=sma-monasca --set global.chart.version=1.6.1 -
f /tmp/loftsman-1615535936/sma-monasca-values.yaml chart=sma-monasca
command=ship namespace=sma version=1.6.1
2021-03-12T07:59:55Z INF Release "sma-monasca" does not exist. Installing it
now.
NAME: sma-monasca
LAST DEPLOYED: Fri Mar 12 07:58:56 2021
NAMESPACE: sma
STATUS: deployed
REVISION: 1
TEST SUITE: None
 chart=sma-monasca command=ship namespace=sma version=1.6.1
2021-03-12T07:59:55Z INF Ship status: success. Recording status, manifest, and
log data to configmap loftsman-sma-manifest in namespace loftsman command=ship
```

11. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

11 Install the System Admin Toolkit Product Stream

Prerequisites

- CSM is installed and verified.
- cray-product-catalog is running.

About this task

ROLE	• System Administrator
OBJECTIVE	Install the SAT (System Admin Toolkit) product stream which now has its own independent release distribution and installer.
LIMITATIONS	None
NEW IN THIS RELEASE	This entire procedure is new with this release. The SAT release distribution, or release artifact, is a gzipped tar file that includes the SAT content and the installation script and libraries required to install it.

Procedure

1. Start a typescript to capture the commands and output from this installation.

   ```
   ncn-m001# script -af product-sat.$(date +%Y-%m-%d).txt
   ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
   ```

2. Copy the release distribution gzipped tar file, e.g. `sat-2.0.x.tar.gz`, to **`ncn-m001`**.

3. Unzip and extract the release distribution.

   ```
   ncn-m001# tar -xvzf sat-2.0.x.tar.gz
   ```

4. Change directory to the extracted release distribution directory.

   ```
   ncn-m001# cd sat-2.0.x
   ```

5. Run the installer, **install.sh**.

   ```
   ncn-m001# ./install.sh
   ```

6. Verify that SAT is successfully installed by running the following command to confirm the expected version.

   ```
   ncn-m001# sat --version
   sat 3.4.0
   ```

7. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

12 Install Slingshot

This procedure installs Slingshot software on an HPE Cray EX system.

About Slingshot Installation

On HPE Cray EX systems the Fabric Manager software is installed on a Kubernetes cluster. This document provides installation steps to set up the Fabric Manager in a container.

The Slingshot High Speed Network (HSN) hardware has standard network components like switches and network interface cards (NICs) that are separate from the compute blade hardware. All HSN switches are connected to a common fabric with a three-hop Dragonfly topology.

Prerequisites

1. Ensure that the switches are up and reachable on the management network.

Installing Fabric Manager software in a container involves the following process:

Table 4. Fabric Manager Installation Sequence

Step #	Description
1.	Install Slingshot software
2.	Access the Fabric Manager pod
3.	Generate credential files for Redfish APIs
4.	Configure Fabric topology via a point-to-point file or SHCD
5.	Set up DNS for HSN IP address
6.	Provision TLS certificates
7.	Bring up the fabric

1. Install the Slingshot software

The Slingshot release distribution, or release artifact, is a gzipped tar file that includes the Slingshot content and the installation script and libraries required to install it. For more reference, refer to the documentation that comes with Slingshot.

1. Start a typescript to capture the commands and output from this installation.

```
ncn-m001# script -af product-slingshot.$(date +%Y-%m-%d).txt
ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Copy the release distribution gzipped tar file for example, `slingshot-VERSION.tar.gz`, to a master node like ncn-m001. *`VERSION`* refers to the Slingshot version.

3. Untar and extract the release distribution.

```
ncn-m001# tar -xvzf slingshot-VERSION.tar.gz
```

4. Change directory to the extracted release distribution directory.

```
ncn-m001# cd slingshot-shasta-VERSION
```

5. Run the installer, **`install.sh`**.

```
ncn-m001# ./install.sh
```

6. Check software versions installed:

```
ncn-m001# kubectl get cm -n services cray-product-catalog -o json | jq -r .data
```

7. Verify that the deployment is up and running.

```
ncn-m001# kubectl get deployment -n services | grep fabric
    slingshot-fabric-manager                       1/1     1            1           18m
```

8. Verify that the Fabric Manager pod is up and running.

```
ncn-m001# kubectl get pods -A |grep slingshot
services         slingshot-fabric-manager-5dc448779c-d8t72            2/2     Running
0          56m
```

9. Verify that the Slingshot repositories are available in Nexus by viewing them in a Web browser or by using a cURL command:

   https://nexus.<shasta domain>/#browse/browse:slingshot-*VERSION*-sle-15sp2

   https://nexus.<shasta domain>/#browse/browse:slingshot-*VERSION*-sle-15sp2

```
ncn-m001# curl -s -k https://packages.local/service/rest/v1/repositories |jq -r '.[] |
select(.name) .name' |grep -E 'slingshot-0.8-sle-15sp2|slingshot-0.8.0-sle-15sp2'
slingshot-VERSION-sle-15sp2-ncn
slingshot-VERSION-sle-15sp2-ncn
```

2. Access the Fabric Manager pod

TIP: The `slingshot-fabric-manager:#` prompt shows when commands must be entered from within the fabric manager pod.

1. Determine the name of the Fabric Manager pod. You can run this command from any master or worker node.

```
ncn-m001# kubectl get pods -n services | grep fabric
slingshot-fabric-manager-65d99d4c97-5bd2t   2/2     Running   1     2d20h
```

2. Open a shell to access the Fabric Manager pod:

```
ncn-m001# kubectl exec -it slingshot-fabric-manager-65d99d4c97-5bd2t -n services -- /bin/bash
```

3. Generate credential files for Redfish APIs

From the Fabric Manager pod, generate credential files for Slingshot switch Redfish APIs. A password must be entered. Replace '*`Enter your password`*' with your password.

```
slingshot-fabric-manager# fmn_pw root:Enter your password
```

4. Configure Fabric topology via a point-to-point file or SHCD

Configure the Fabric topology by either using a point-to-point file or SHCD.

If you have an existing `Shasta_system_hsn_pt_pt.csv` file, you can configure Fabric Manager with the point-to-point file.

4(a) Configure Fabric topology using a point-to-point file

The following procedure shows how you can configure the topology using a point-to-point file.

1. Log on to a Kubernetes worker or master node.
2. Find the name of the Fabric Manager pod:

```
ncn-m001# kubectl get pods -A |grep fabric |awk '{print $2}'
slingshot-fabric-manager-5dc448779c-d8t72
```

3. Copy the `Shasta_system_hsn_pt_pt.csv` file to a directory container:

```
ncn-m001# kubectl cp Shasta_system_hsn_pt_pt.csv services/slingshot-fabric-manager-5dc448779c-d8t72:/
tmp
```

4. Log on to the Fabric Manager pod.

```
ncn-m001# kubectl exec -it -n services slingshot-fabric-manager-5dc448779c-d8t72 -- /bin/bash
Defaulting container name to slingshot-fabric-manager.
Use 'kubectl describe pod/slingshot-fabric-manager-5dc448779c-d8t72 -n services' to see all of the
containers in this pod.
```

5. Determine the HSN IP address range. You may not see host IPs and xnames if the Fabric Manager is started for the first time.

```
slingshot-fabric-manager# fmn_shasta_dns -p
HSN
HERE IS THE SUBNET YOU NEED cabinet_3000 10.253.0.0/22
10.253.0.6 nc
10.253.0.27 x3000c0s9b0n0h1
10.253.0.26 x3000c0s9b0n0h0
10.253.0.13 x3000c0s7b0n0h1
10.253.0.12 x3000c0s7b0n0h0
10.253.0.8 x3000c0s11b0n0h1
10.253.0.9 x3000c0s11b0n0h0
10.253.0.3 x3000c0s19b2n0h0
10.253.0.2 x3000c0s19b1n0h0
10.253.0.20 x3000c0s19b4n0h0
10.253.0.21 x3000c0s19b3n0h0
10.253.0.7 x3000c0s27b0n0h0
```

   Note the <IP address range>. The <IP address range> in this example is 10.253.0.0/22.

6. Run STT to update fabric template with HSN NIC configuration.

```
slingshot-fabric-manager:# slingshot-topology-tool
STT diags log directory -  /root/stt_diags_logs
STT diags log directory -  /root/stt_diags_logs/default
Loading  point2point file /opt/cray/etc/sct/Shasta_system_hsn_pt_pt.csv to default topology
Loading fabric template file /opt/cray/fabric_template.json to default topology
Welcome to the Slingshot Topology Tool.
   General Usage is <command> <arguments>
   Type help or ? to list commands.
(STT)
```

   a. Create a new topology.

```
(STT) new topology setup
STT diags log directory -  /root/stt_diags_logs/setup
```

   b. Set the active topology.

```
(STT) set_active topology setup
```

   c. Load the point-to-point file.

```
(STT) load p2p /tmp/Shasta_system_hsn_pt_pt.csv
Working with 'setup' topology.
```

d. Add NICs and modify the IP address range in this example for the system's high-speed network (HSN) address range from step 4.

```
(STT) add nics <IP address range>
Working with 'setup' topology.
```

e. Save the fabric template file.

```
(STT) save topology fabric_template_setup.json
Working with 'setup' topology.
Wrote 'setup' topology to fabric_template_setup.json
```

f. Exit STT.

```
(STT) exit
Exiting STT
```

7. Save the updated fabric topology files to the correct directories:

```
slingshot-fabric-manager# cp fabric_template_setup.json /opt/cray/fabric_template.json
slingshot-fabric-manager:# mkdir /opt/cray/etc
slingshot-fabric-manager:# mkdir /opt/cray/etc/sct/
slingshot-fabric-manager:# cp /tmp/Shasta_system_hsn_pt_pt.csv /opt/cray/etc/sct/.
```

4(b) Configure Fabric topology using SHCD

1. Log on to a Kubernetes worker or master node.

2. Find the name of the Fabric manager pod:

```
ncn-m001# kubectl get pods -A |grep fabric |awk '{print $2}'
slingshot-fabric-manager-5dc448779c-d8t72
```

3. Upload the `SHCD.csv` file.

The `SHCD.csv` file is sourced from the point-to-point (pt_pt) tab of the SHCD Excel spreadsheet.

```
ncn-m001# kubectl cp shcd.csv services/slingshot-fabric-manager-5dc448779c-d8t72:/opt/slingshot/
```

4. Log on to the Fabric Manager pod.

```
ncn-m001# kubectl exec -it -n services slingshot-fabric-manager-5dc448779c-d8t72 -- /bin/bash
Defaulting container name to slingshot-fabric-manager.
Use 'kubectl describe pod/slingshot-fabric-manager-5dc448779c-d8t72 -n services' to see all of the
containers in this pod.
```

5. Determine the HSN IP address range.

```
slingshot-fabric-manager# fmn_shasta_dns -p
HSN
HERE IS THE SUBNET YOU NEED cabinet_3000 10.253.0.0/22
10.253.0.6 nc
10.253.0.27 x3000c0s9b0n0h1
10.253.0.26 x3000c0s9b0n0h0
10.253.0.13 x3000c0s7b0n0h1
10.253.0.12 x3000c0s7b0n0h0
10.253.0.8 x3000c0s11b0n0h1
10.253.0.9 x3000c0s11b0n0h0
10.253.0.3 x3000c0s19b2n0h0
10.253.0.2 x3000c0s19b1n0h0
10.253.0.20 x3000c0s19b4n0h0
10.253.0.21 x3000c0s19b3n0h0
10.253.0.7 x3000c0s27b0n0h0
```

Note the <IP address range>. The <IP address range> in this example is 10.253.0.0/22.

6. Start the Slingshot Topology Tool (STT):

```
slingshot-fabric-manager# slingshot-topology-tool
```

7. Run the following STT commands to create the fabric configuration files:

   a. Create a new topology.

   ```
   (STT) new topology setup
   STT diags log directory -  /root/stt_diags_logs/setup
   ```

   b. Set the active topology.

   ```
   (STT) set_active topology setup
   ```

   c. Load the `SHCD.csv` file.

   ```
   (STT) load shcd shcd.csv
   ```

   d. Add NICS and modify the IP address range in this example for the system's high-speed network (HSN) address range from step 4.

   ```
   (STT)add nics <IP address range>
   ```

   e. Save the `fabric_template.json` file.

   ```
   (STT)save topology fabric_template.json
   ```

   f. Save the point-to-point file.

   ```
   (STT)save p2p Shasta_system_hsn_pt_pt.csv
   ```

   g. Exit STT.

   ```
   (STT) exit
   ```

8. Create a new directory and move the configuration files to the correct folders:

   ```
   slingshot-fabric-manager# mkdir -p /opt/cray/etc/sct/
   slingshot-fabric-manager# mv /tmp/Shasta_system_hsn_pt_pt.csv /opt/cray/etc/sct/.
   slingshot-fabric-manager# mv fabric_template.json /opt/cray/.
   ```

5. Set up DNS for HSN IP Addresses

The `fabric_template.json` can be parsed using the `fmn_shasta_dns` command for a mapping of the HSN IP addresses to their component names (xname). Run the following command from the Fabric Manager pod and add the HSN IP addresses and their corresponding DNS names to the DNS server.

```
slingshot-fabric-manager# fmn_shasta_dns -i /opt/cray/fabric_template.json -f && fmn_shasta_dns -p
Warning:  Skipping record without proper xname: {'id': 'x3000c0r21a0l7', 'meta': {'cable_id':
'3000.3000.00.0003', 'calculated_distance': '000.8795086', 'conn_port': 'x3000c0r21j8p0',
'dst_egress_a': 'none', 'dst_egress_b': 'none', 'dst_port': 'nc', 'hsnIp': '10.253.0.7/22',
'link_type': 'edge', 'part_length': '001.00', 'part_number': '102234600', 'route': '[]',
'src_egress_a': 'none', 'src_egress_b': 'none', 'stage': '1'}}
New record.
  New     : 10.253.0.26 x3000c0s9b0n0h1
New record.
  New     : 10.253.0.27 x3000c0s9b0n0h0
New record.
  New     : 10.253.0.12 x3000c0s7b0n0h1
New record.
  New     : 10.253.0.13 x3000c0s7b0n0h0
New record.
  New     : 10.253.0.6 x3000c0s27b0n0h0
New record.
  New     : 10.253.0.9 x3000c0s11b0n0h1
New record.
  New     : 10.253.0.8 x3000c0s11b0n0h0
New record.
  New     : 10.253.0.2 x3000c0s19b2n0h0
New record.
  New     : 10.253.0.3 x3000c0s19b1n0h0
New record.
  New     : 10.253.0.21 x3000c0s19b4n0h0
```

```
New record.
  New     : 10.253.0.20 x3000c0s19b3n0h0

Added or updated existing reservation record in SLS

HSN
    HERE IS THE SUBNET YOU NEED cabinet_3000 10.253.0.0/22
      10.253.0.26 x3000c0s9b0n0h1
      10.253.0.27 x3000c0s9b0n0h0
      10.253.0.12 x3000c0s7b0n0h1
      10.253.0.13 x3000c0s7b0n0h0
      10.253.0.6 x3000c0s27b0n0h0
      10.253.0.9 x3000c0s11b0n0h1
      10.253.0.8 x3000c0s11b0n0h0
      10.253.0.2 x3000c0s19b2n0h0
      10.253.0.3 x3000c0s19b1n0h0
      10.253.0.21 x3000c0s19b4n0h0
      10.253.0.20 x3000c0s19b3n0h0
```

Use the `fmn_shasta_dns -p` command to print the current reservations in the System Layout Service (SLS).

```
slingshot-fabric-manager# fmn_shasta_dns -p

HSN
    HERE IS THE SUBNET YOU NEED cabinet_3000 10.253.0.0/22

# This output shows that there are no reservations yet
# To add reservations from /opt/cray/fabric_template.json
# pass this file with the -I flag
# The -f flag specifies to replace any existing reservation found for this IP.
```

The `fmn_shasta_dns` script has many options. For example, the `-f -x` flags can be used to remove existing DNS reservations. Note that you can run this command once. This information is only for reference and can be used if you need to remove existing DNS reservations.

```
slingshot-fabric-manager:# fmn_shasta_dns -i /opt/cray/fabric_template.json -f -x
Warning:  Skipping record without proper xname: {'id': 'x3000c0r21a0l7', 'meta': {'cable_id':
'3000.3000.00.0003', 'calculated_distance': '000.8795086', 'conn_port': 'x3000c0r21j8p0',
'dst_egress_a': 'none', 'dst_egress_b': 'none', 'dst_port': 'nc', 'hsnIp': '10.253.0.7/22',
'link_type': 'edge', 'part_length': '001.00', 'part_number': '102234600', 'route': '[]',
'src_egress_a': 'none', 'src_egress_b': 'none', 'stage': '1'}}
Existing record match.
  Existing: 10.253.0.26 x3000c0s9b0n0h1
  New     : 10.253.0.26 x3000c0s9b0n0h1
Deleting record.
Existing record match.
  Existing: 10.253.0.27 x3000c0s9b0n0h0
  New     : 10.253.0.27 x3000c0s9b0n0h0
Deleting record.
Existing record match.
  Existing: 10.253.0.12 x3000c0s7b0n0h1
  New     : 10.253.0.12 x3000c0s7b0n0h1
Deleting record.
Existing record match.
  Existing: 10.253.0.13 x3000c0s7b0n0h0
  New     : 10.253.0.13 x3000c0s7b0n0h0
Deleting record.
Existing record match.
  Existing: 10.253.0.6 x3000c0s27b0n0h0
  New     : 10.253.0.6 x3000c0s27b0n0h0
...
Deleted required reservations record in SLS
slingshot-fabric-manager# fmnshasta_dns -p

HSN
    HERE IS THE SUBNET YOU NEED cabinet_3000 10.253.0.0/22
```

6. Provision TLS certificates

TLS certificates must be provisioned every time a new switch is added to the topology.

1. From the Fabric Manager pod, use the `fmn_cert_provision` command and fabric template to provision the certificates.

```
slingshot-fabric-manager# fmn_cert_provision -t /opt/cray/fabric_template.json
```

7. Bring up the Fabric Manager

1. From the Fabric Manager pod, run the following command to initiate the fabric.

```
slingshot-fabric-manager# fmn_fabric_bringup -c
```

2. Set up the IP/MAC configuration:

```
slingshot-fabric-manager# fmn_fabric_bringup -n
```

3. Exit the pod to go back to the master node. Finish the typescript file.

```
slingshot-fabric-manager# exit
ncn-m001# exit
```

Slingshot Post-Installation Tasks

Note that Slingshot post-installation tasks like verifying the fabric, configuring NTP, Syslog, and telemetry, and configuring LAG must be done after the compute nodes are booted. See *Slingshot Post-Installation Tasks* for reference.

12.1 Update Slingshot Switch Firmware

Prerequisites

- All the switches must be configured and cabled on the management network.
- v1.4

About this task

This procedure uses the Firmware Action Service (FAS) to update switch firmware. Do this only if the switch firmware needs to be updated.

LEVEL	**Level 2 IaaS**
ROLE	System administrator, system installer
OBJECTIVE	Use the Firmware Action Service (FAS) to update switch firmware.
LIMITATIONS	Release 1.4.
NEW IN THIS RELEASE	This procedure uses Firmware Action Service (FAS) available with v1.4.

Procedure

When the Fabric Manager is running inside a Kubernetes-orchestrated container, use FAS to update switch firmware. You can perform the firmware update from any Kubernetes worker or master node.

UPDATE FIRMWARE WITH FAS

1. List the available firmware packages.

```
ncn-m001# cray fas images list
[[images]]
imageID = "b67bb2a2-a5eb-44da-a05e-406a5cca9937"
createTime = "2021-01-27T00:04:43Z"
deviceType = "RouterBMC"
manufacturer = "cray"
models = [ "ColoradoSwitchBoard_REV_A", "ColoradoSwitchBoard_REV_B", "ColoradoSwitchBoard_REV_C",
"ColoradoSwtBrd_revA", "ColoradoSwtBrd_revB", "ColoradoSwtBrd_revC", "ColoradoSWB_revA",
"ColoradoSWB_revB", "ColoradoSWB_revC", "101878104_", "ColumbiaSwitchBoard_REV_A",
"ColumbiaSwitchBoard_REV_B", "ColumbiaSwitchBoard_REV_D", "ColumbiaSwtBrd_revA",
"ColumbiaSwtBrd_revB", "ColumbiaSwtBrd_revD", "ColumbiaSWB_revA", "ColumbiaSWB_revB",
"ColumbiaSWB_revD",]
target = "BMC"
tags = [ "default",]
firmwareVersion = "sc.1.4.399-shasta-release.arm64.2021-01-22T11:50:10+00:00.24ee2c7"
semanticFirmwareVersion = "1.4.399"
pollingSpeedSeconds = 30
s3URL = "s3:/fw-update/40d35f0c603311eb9a3282a8e219a16d/controllers-1.4.399.itb"
```

2. Determine the switch component names (xnames).

```
ncn-m001# cray hsm inventory redfishEndpoints list --type RouterBMC --format json | jq -r 'flatten'
| jq -c '[ .[] | .ID ]'
["x3000c0r15b0","x9000c1r1b0","x9000c1r3b0","x9000c1r5b0","x9000c1r7b0","x9000c3r1b0","x9000c3r3b0","
x9000c3r5b0"]
```

3. To update firmware on many or all switches using FAS, create an update input file, for example, `fas_slingshot.json`.

```
ncn-m001# vi fas_slingshot.json
```

Use the `fas_slingshot.json` file in the call to `cray fas`. To do a dry run before performing an actual update, set the `overrideDryrun` flag to `false`. To perform an actual update, set the `overrideDryrun` flag to `true`. The switch component names listed here are examples and need to be provided from your environment.

```
ncn-m001# more fas_slingshot.json
{
  "stateComponentFilter": {
    "xnames":[
      "x3000c0r42b0",
      "x9000c1r7b0",
      "x9000c1r7b0",
      "x9000c1r5b0",
      "x9000c1r7b0",
      "x9000c3r1b0",
      "x9000c3r3b0",
      "x9000c3r5b0"
      ]
  },
  "targetFilter": {
    "targets":[
       "BMC"
    ]
  },
  "command": {
  "overrideDryrun":true,
  "restoreNotPossibleOverride":true
  }
}
```

> **NOTE:** If a switch fails to update, use the alternative `fas_slingshot_workaround.json` JSON file settings that follow. This workaround should force the switch to update.

```
fas_slingshot_workaround.json | jq
{
  "stateComponentFilter": {
    "xnames": [
      "x3122c0r47b0"
     ]
  },
  "imageFilter": {
    "imageID": "c4a2891f-75ca-483b-a8b3-67e49fdab41f",
    "overrideImage": true
  },
  "targetFilter": {
    "targets": [
      "BMC"
    ]
  },
  "command": {
    "overrideDryrun": true,
    "restoreNotPossibleOverride": true,
    "overwriteSameImage": false
  }
}
```

4. Update the Slingshot switch firmware with FAS.

```
ncn-m001# cray fas actions create fas_slingshot.json
actionID = "13729f52-4bab-4134-95a6-7cb8fa85993d" overrideDryrun = true
```

5. Check the status of the FAS update.

```
ncn-m001# cray fas actions describe "13729f52-4bab-4134-95a6-7cb8fa85993d"
actionID = "13729f52-4bab-4134-95a6-7cb8fa85993d"
snapshotID = "00000000-0000-0000-0000-000000000000"
startTime = "2021-01-29 19:53:30.238123956 +0000 UTC"
endTime = "2021-01-29 19:57:31.661732442 +0000 UTC"
state = "completed"
blockedBy = []

[command]
overrideDryrun = true
restoreNotPossibleOverride = true
overwriteSameImage = false
version = "latest"
tag = "default"
description = ""

[parameters.stateComponentFilter]
xnames = [ "x3000c0r15b0", "x9000c1r1b0", "x9000c1r3b0", "x9000c1r5b0", "x9000c1r7b0",
"x9000c3r1b0", "x9000c3r3b0", "x9000c3r5b0",]

[parameters.inventoryHardwareFilter]

[parameters.imageFilter]
imageID = "00000000-0000-0000-0000-000000000000"
overrideImage = false

[parameters.targetFilter]
targets = [ "BMC",]

[parameters.command]
overrideDryrun = true
restoreNotPossibleOverride = true
overwriteSameImage = false
version = "latest"
tag = "default"
description = ""

[operationSummary.initial]
operationKeys = []

[operationSummary.configured]
operationKeys = []
```

```
[operationSummary.blocked]
operationKeys = []

[operationSummary.inProgress]
operationKeys = []

[operationSummary.needsVerified]
operationKeys = []

[operationSummary.verifying]
operationKeys = []

[operationSummary.failed]
operationKeys = []

[operationSummary.succeeded]
[[operationSummary.succeeded.operationKeys]]
operationID = "07cf3ce1-fc2e-480c-98ea-d76b4dd8b815"
xname = "x9000c1r7b0"
target = "BMC"
targetName = "BMC"
fromFirmwareVersion = "sc.1.4.112-prod-master.arm64.2020-07-20T17:30:19+00:00.df27e04"
stateHelper = "Update Successful to version: sc.1.4.399-shasta-
release.arm64.2021-01-22T11:50:10+00:00.24ee2c7"

[[operationSummary.succeeded.operationKeys]]
operationID = "279532b9-e414-41c5-a7df-d051d3d0a15e"
xname = "x9000c3r1b0"
target = "BMC"
targetName = "BMC"
fromFirmwareVersion = "sc.1.2.161-shasta-release.arm64.2020-03-25T15:37:26+00:00.bb6e556"
stateHelper = "Update Successful to version: sc.1.4.399-shasta-
release.arm64.2021-01-22T11:50:10+00:00.24ee2c7"

[[operationSummary.succeeded.operationKeys]]
operationID = "30db6060-6f6d-4d7d-bd43-99707687bcaa"
xname = "x9000c1r5b0"
target = "BMC"
targetName = "BMC"
fromFirmwareVersion = "sc.1.4.112-prod-master.arm64.2020-07-20T17:30:19+00:00.df27e04"
stateHelper = "Update Successful to version: sc.1.4.399-shasta-
release.arm64.2021-01-22T11:50:10+00:00.24ee2c7"

[[operationSummary.succeeded.operationKeys]]
operationID = "7988ee39-0fba-49a6-aceb-201c5fc2c90e"
xname = "x9000c1r1b0"
target = "BMC"
targetName = "BMC"
fromFirmwareVersion = "sc.1.4.112-prod-master.arm64.2020-07-20T17:30:19+00:00.df27e04"
stateHelper = "Update Successful to version: sc.1.4.399-shasta-
release.arm64.2021-01-22T11:50:10+00:00.24ee2c7"

[[operationSummary.succeeded.operationKeys]]
operationID = "a09765d1-0d5b-4213-856d-1b61fffef326"
xname = "x9000c3r5b0"
target = "BMC"
targetName = "BMC"
fromFirmwareVersion = "sc.1.2.178-shasta-release.arm64.2020-04-15T14:34:49+00:00.3d0134c"
stateHelper = "Update Successful to version: sc.1.4.399-shasta-
release.arm64.2021-01-22T11:50:10+00:00.24ee2c7"

[[operationSummary.succeeded.operationKeys]]
operationID = "ace645d7-6c2b-463d-b2be-4c4f8a74c960"
xname = "x9000c3r3b0"
target = "BMC"
targetName = "BMC"
fromFirmwareVersion = "sc.1.2.178-shasta-release.arm64.2020-04-15T14:34:49+00:00.3d0134c"
stateHelper = "Update Successful to version: sc.1.4.399-shasta-
release.arm64.2021-01-22T11:50:10+00:00.24ee2c7"

[[operationSummary.succeeded.operationKeys]]
operationID = "d67cbbc7-4a2f-41e5-90ba-c9829e180abe"
xname = "x9000c1r3b0"
target = "BMC"
targetName = "BMC"
fromFirmwareVersion = "sc.1.4.112-prod-master.arm64.2020-07-20T17:30:19+00:00.df27e04"
```

```
stateHelper = "Update Successful to version: sc.1.4.399-shasta-
release.arm64.2021-01-22T11:50:10+00:00.24ee2c7"

[operationSummary.noOperation]
[[operationSummary.noOperation.operationKeys]]
operationID = "1c49e3af-32a6-4dce-ae35-24effe6ab54d"
xname = "x3000c0r15b0"
target = "BMC"
targetName = "BMC"
fromFirmwareVersion = "sc.1.4.399-shasta-release.arm64.2021-01-22T11:50:10+00:00.24ee2c7"
stateHelper = "Firmware at requested version"

[operationSummary.noSolution]
operationKeys = []

[operationSummary.aborted]
operationKeys = []

[operationSummary.unknown]
operationKeys = []
```

13 Install OS

Prerequisites

- Nexus must be installed and running

About this task

The following procedure details how to install SUSE operating system rpm repositories to Nexus on v1.4 systems. The following examples use ncn-m001 as a generic placeholder for the node that the following steps will be run on. The SUSE tarballs will be untarred and the /install.sh scripts run to get the contents uploaded to Nexus. TIP: Ensure that there is adequate disk space where tarballs are located. Remove the ...gz files after their extraction.

Procedure

1. Start a typescript to capture the commands and output from this installation.

```
ncn-m001# script -af product-os.$(date +%Y-%m-%d).txt
ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Change directory.

```
ncn-m001# cd [path to location of tarballs]/os
```

3. List all files.

```
ncn-m001# ls -lh
total 58G
    -rw-r--r--  1 root   root         22M Feb  6 17:25 SUSE-Backports-
x86_64-21.5.2.tar.gz
    -rw-r--r--  1 root   root         16G Feb  6 18:20 SUSE-Products-
x86_64-21.5.1.tar.gz
    -rw-r--r--  1 root   root         42G Feb  6 18:55 SUSE-Updates-
x86_64-21.5.1.tar.gz
```

4. Unarchive and uncompress the desired file (`SUSE-Backports-x86_64`).

```
ncn-m001# tar -zxf SUSE-Backports-x86_64-21.5.2.tar.gz
```

5. Change directory.

```
ncn-m001# cd SUSE-Backports-x86_64-21.5.2
```

6. Run the install script.

```
ncn-m001# ./install.sh
Install SUSE-Backports-x86_64-21.5.2
```

Repeat the above steps for `SUSE-Products-x86_64`.

7. Unarchive and uncompress the desired file (`SUSE-Products-x86_64`).

   ```
   ncn-m001# tar -zxf SUSE-Products-x86_64-21.5.2.tar.gz
   ```

8. Change directory.

   ```
   ncn-m001# cd SUSE-Products-x86_64-21.5.2
   ```

9. Run the install script.

   ```
   ncn-m001# ./install.sh
   Install SUSE-Products-x86_64-21.5.2
   ```

   Repeat the above steps for `SUSE-Updates-x86_64`.

10. Unarchive and uncompress the desired file (`SUSE-Updates-x86_64`).

    ```
    ncn-m001# tar -zxf SUSE-Updates-x86_64-21.5.2.tar.gz
    ```

11. Change directory.

    ```
    ncn-m001# cd SUSE-Updates-x86_64-21.5.2
    ```

12. Run the install script.

    ```
    ncn-m001# ./install.sh
    Install SUSE-Updates-x86_64-21.5.2
    ```

13. There may also be `SUSE-isos-x86_64` or `SUSE-PTF-x86_64` tarballs included in this directory but do not need to be extracted and installed unless directed by HPE. For v1.4, it is recommended to install the PTF tarball, but not the ISOs tarball.

    Products and update tarballs are LARGE and may take up to 25 minutes to run `install.sh`.

14. Finish the typescript file started at the beginning of this procedure.

    ```
    ncn-m001# exit
    ```

14 Install and Configure the Cray Operating System (COS)

Prerequisites

- The Cray command line interface (CLI) tool is initialized and configured on the system. See "Configure the Cray Command Line Interface (CLI)" in the *HPE Cray EX System Administration Guide S-8001* for more information.
- Cray system management has been installed and verified, including the following Helm Charts:
  - gitea
  - cray-product-catalog
  - cray-cfs-api
  - cray-cfs-operator
  - cray-ims

```
ncn-m001# helm ls -n services | grep -E 'gitea|cray-cfs-operator|cray-cfs-api|cray-ims|cray-product-catalog'
cray-cfs-api            services    1    2020-10-19 15:00:12.189790185 -0500 CDT deployed    cray-cfs-
api-1.5.0-20201006133025+58367e7            0.14.1-20201006133025_58367e7
cray-cfs-operator       services    1    2020-10-19 15:00:10.411523529 -0500 CDT deployed    cray-cfs-
operator-1.8.0-20201006133513+dab1dab       1.8.0-20201006133513_dab1dab
cray-product-catalog    services    1    2020-10-22 14:10:11.321431248 -0500 CDT deployed    cray-product-
catalog-0.0.1-20201028184555+30b896c    0.0.1-20201028184555_30b896c
cray-ims                services    2    2020-10-30 16:21:25.560614562 -0500 CDT deployed    cray-
ims-2.4.1-20201030143835+0cb54eb                2.3.12-20201030143835_0cb54eb
gitea                   services    1    2020-10-23 14:25:55.544840236 -0500 CDT deployed
gitea-1.8.0-20201023142353+f21d3f0
```

About this task

LEVEL	**Level 2 IaaS.**
ROLE	System administrator
OBJECTIVE	Install, configure, prepare, and boot COS on a HPE Cray EX system.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in this release.

Procedure

COS INSTALLATION

1. Start a typescript to capture the commands and output from this installation.

```
ncn-m001# script -af product-cos.$(date +%Y-%m-%d).txt
ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Copy the release distribution gzipped tar file to ncn-m001.

3. Unzip and extract the release distribution.

```
ncn-m001# tar xzvf cos-2.0.10.tar.gz
```

4. Change to the extracted release distribution directory.

```
ncn-m001# cd cos-2.0.10
```

5. Run the `install.sh` script to begin installing COS.

```
ncn-m001# ./install.sh
+ trap notify ERR ++ dirname ./install.sh ...
```

6. Query Nexus repo and verify repos were created for COS.

```
ncn-m001# curl -s -k https://packages.local/service/rest/v1/repositories \
| jq -r '.[] | select(.name | startswith("cos")) | .name'
...
cos-2.0.10-sle-15sp1-compute
cos-2.0.10-sle-15sp2
cos-2.0-sle-15sp1-compute
cos-2.0-sle-15sp2
...
```

7. Verify COS config and image recipes components.

   Check that COS pods have completed without error before checking the cray-product-catalog for the presence of the COS recipe.

```
ncn-m001# kubectl get jobs -A | egrep "NAME|cos"
NAMESPACE     NAME                                                  COMPLETIONS
DURATION   AGE
services      cos-config-import-2.0.10                               1/1
94s        17m
services      cos-image-sp1-recipe-import-2.0.10                     1/1
2m35s      12m
ncn-m001# kubectl get pods -A | egrep "NAME|cos"
NAMESPACE        NAME
READY   STATUS             RESTARTS   AGE
services         cos-config-import-2.0.10-dqx2j
0/3     Completed          0          17m
services         cos-image-sp1-recipe-import-2.0.10-pdq7t
0/3     Completed          0          12m
ncn-m001# kubectl get cm cray-product-catalog -n services -o json | jq -r .data.cos

2.0.10:
  configuration:
    clone_url: https://vcs.sif.dev.cray.com/vcs/cray/cos-config-management.git
    commit: 0c70558938105b7a2841ae458c9184fe05ca7d0a
    import_branch: cray/cos/2.0.10
    import_date: 2021-01-29 23:38:10.971626
    ssh_url: git@vcs.sif.dev.cray.com:cray/cos-config-management.git
  images:
    cray-shasta-compute-sles15sp1.x86_64-1.4.56:
      id: 88c5c6ed-a873-4328-b37d-9983b2e20e27
  recipes:
```

```
    cray-shasta-compute-sles15sp1.x86_64-1.4.56:
      id: f5e0ea59-5687-476c-95c5-85d41c93749c

ncn-m001:~ # export IMS_RECIPE_ID=f5e0ea59-5687-476c-95c5-85d41c93749c

ncn-m001:~ # cray ims recipes describe $IMS_RECIPE_ID --format json
{
  "created": "2021-01-29T23:38:56.014283+00:00",
  "id": "f5e0ea59-5687-476c-95c5-85d41c93749c",
  "link": {
    "etag": "fba10f7df4591d9a4f4ad92e7b5ceb40",
    "path": "s3://ims/recipes/f5e0ea59-5687-476c-95c5-85d41c93749c/recipe.tar.gz",
    "type": "s3"
  },
  "linux_distribution": "sles15",
  "name": "cray-shasta-compute-sles15sp1.x86_64-1.4.56",
  "recipe_type": "kiwi-ng"
}
```

8. Verify the COS microservices CPS and NMD have deployed and are in a Running state

```
ncn-m001# kubectl get pods -A | egrep "NAME|nmd|cps"

NAMESPACE        NAME
READY   STATUS      RESTARTS   AGE
services         cray-cps-5f57496484-627zk
2/2     Running     0          17m
services         cray-cps-5f57496484-gv6f6
2/2     Running     0          17m
services         cray-cps-cm-pm-bnlz2
5/5     Running     0          17m
services         cray-cps-cm-pm-fn57v
5/5     Running     0          17m
services         cray-cps-cray-jobs-cray-cps-job-1-rwglr
0/2     Completed   0          17m
services         cray-cps-etcd-4zk4qxrv7x
1/1     Running     0          17m
services         cray-cps-etcd-wkf7889jqh
1/1     Running     0          17m
services         cray-cps-etcd-xf4jnxmrps
1/1     Running     0          17m
services         cray-cps-wait-for-etcd-1-g74dg
0/1     Completed   0          17m
services         nmdv2-service-b8c4b598b-6rq6d
2/2     Running     0          15m
```

Refer to *Enable NCN Personalization* on page 100 before proceeding.

DVS SERVER CONFIGURATION

9. Check that the DVS servers have loaded the DVS and LNet kernel modules.

   This quick check signals that the DVS server install and config procedures were successfully completed.

```
ncn-m001# pdsh -w ncn-w[001-NNN] lsmod | grep -P '\bdvs\b' | dshbak -c
----------------
ncn-w[001-NNN]
----------------
dvs 425984 0
dvsipc 188416 2 dvs
dvsproc 151552 3 dvsipc,dvsipc_lnet,dvs
craytrace 20480 5 dvsipc,dvsproc,dvsipc_lnet,dvs
dvskatlas 24576 4 dvsipc,dvsproc,dvsipc_lnet,dvs
```

COS CONFIGURATION

10. Create a branch using the imported branch from the installation to customize COS.

The imported branch will be reported in the cray-product-catalog and can be used as a base branch. The imported branch from the installation cannot be modified. In this procedure, `integration` will be used to add customizations to the configuration. Replace the *`clone_url`* hostname with `api-gw-service-nmn.local`.

```
ncn-m001# kubectl get cm cray-product-catalog -n services -o yaml \
| yq r - 'data.cos' | yq r - '"2.0.30"'
2.0.30:
  configuration:
    clone_url: https://vcs.rocket.dev.cray.com/vcs/cray/cos-config-management.git
    commit: 215eab2c316fb75662ace6aaade8b8c2ab2d08ee
    import_branch: cray/cos/2.0.30    <==  IMPORT_BRANCH
    import_date: 2021-02-21 23:01:16.100251
    ssh_url: git@vcs.rocket.dev.cray.com:cray/cos-config-management.git
  images:
    cray-shasta-compute-sles15sp1.x86_64-1.4.64:
      id: c8013386-6fe4-49f3-92ca-96d4f3be29a7
  recipes:
    cray-shasta-compute-sles15sp1.x86_64-1.4.64:
      id: 5149788d-4e5d-493d-b259-f56156a58b0d
```

```
ncn-m001# export IMPORT_BRANCH=cray/cos/2.0.20
```

```
ncn-m001# kubectl get secret -n services vcs-user-credentials \
--template={{.data.vcs_password}} | base64 --decode
<== password output ==>
```

```
ncn-m001# git clone https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git
```

```
ncn-m001# cd cos-config-management/
```

```
ncn-m001# git checkout $IMPORT_BRANCH && git pull
Branch 'cray/cos/2.0.20' set up to track remote branch 'cray/cos/2.0.20' from
'origin'.
Switched to a new branch 'cray/cos/2.0.20'
Already up to date
```

```
ncn-m001# git checkout -b integration && git merge $IMPORT_BRANCH
Switched to a new branch 'integration'$
Already up to date.
```

11. Apply any customizations and modifications to the Ansible configuration, if required. COS configuration can be done later. Refer to 'Change Root Passwords for Compute Nodes in the *HPE Cray EX System Administration Guide S-8001* for more information on different EX configurations.

12. Push the changes to the repository using the proper credentials.

a. Upload the updated local repository to VCS.

```
ncn-m001# git push --set-upstream origin integration
Username for 'https://api-gw-service-nmn.local': crayvcs
Password for 'https://crayvcs@api-gw-service-nmn.local':
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 64 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 428 bytes | 428.00 KiB/s, done.
```

```
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a new pull request for 'integration':
remote:   https://vcs.SYSTEM_REPO.dev.cray.com/vcs/cray/cos-config-management/compare/
master...integration
remote:
remote: . Processing 1 references
remote: Processed 1 references in total
To https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git
 * [new branch]      integration -> integration
Branch 'integration' set up to track remote branch 'integration' from 'origin'.
```

b. Capture the most recent commit.

```
ncn-m001# git rev-parse --verify HEAD
253fbc73b1b8b4c936b8c6fdfc6e2d239d474a7e
```

c. Store the commit hash for later use.

```
ncn-m001# export COS_CONFIG_COMMIT_HASH=5cd59022d5e4d953a4124cff73064a810cb7ed4f
```

13. Create a Configuration Framework Service (CFS) session configuration for COS.

a. Create a JSON file as input to the CFS configurations CLI command.

Do not attempt to cut and paste the following example. This will lead to an incorrectly formatted json syntax.

```
ncn-m001# cat cos-config-2.0.10.json
{
  "layers": [
    {
      "name": "cos-integration-2.0.10",
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
      "playbook": "site.yml",
      "commit": "<COS_CONFIG_COMMIT_HASH>"
    }
  ]
}
```

If other products have been installed on the system, add additional configuration layers to be applied during the CFS session. This configuration can be used for pre-boot image customization, as well as post-boot node personalization. The clone URL, commit, and top-level play should be run for all configuration layers.

b. Update the configuration using the new JSON file.

```
ncn-m001# cray cfs configurations update cos-config-2.0.10 --file ./cos-config-2.0.10.json \
--format json
{
  "lastUpdated": "2020-11-05T17:05:58Z",
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
      "commit": "5cd59022d5e4d953a4124cff73064a810cb7ed4f",
      "name": "cos-integration-2.0.10",
      "playbook": "site.yml"
    }
  ],
  "name": "cos-config-2.0.10"
}
```

c. Build COS compute image from image recipe.

Refer to *Troubleshoot COS Image Building Failure* on page 182 before building the COS compute image. The following step will fail unless the workaround is performed.

```
Create and export an SSH Key for IMS
====================================
```

```
ncn-m001 # cray ims public-keys create --name "my public key" --public-key \
~/.ssh/id_rsa.pub

[[results]]
created = "2020-11-18T17:53:28.163021+00:00"
id = "4a629135-9ab6-4164-9fa2-86bd018a4c61"
name = "my public key"
...

NOTE: If you already have an IMS public key, use that for the build.
      Check by running `cray ims public-keys list`

ncn-m001 # export IMS_PUBLIC_KEY_ID=4a629135-9ab6-4164-9fa2-86bd018a4c61
```

Verify IMS_RECIPE_ID has expected value.

```
ncn-m001 # echo $IMS_RECIPE_ID (set in step 3)
```

Create an IMS image from the recipe

```
ncn-m001 # cray ims jobs create --job-type create \
--image-root-archive-name sles15sp1-recipe-example-image \
--artifact-id $IMS_RECIPE_ID \
--public-key-id $IMS_PUBLIC_KEY_ID \
--enable-debug False


Note results of command, should have action "creating"


        - initrd_file_name = "initrd"
          status = "creating"
          artifact_id = "a89a1013-f1dd-4ef9-b0f8-ed2b1311c98a"
          enable_debug = false
          kubernetes_configmap = "cray-ims-52e2d6f6-977f-4b91-
b0fb-174dc7e195e2-configmap"
          kubernetes_service = "cray-ims-52e2d6f6-977f-4b91-
b0fb-174dc7e195e2-service"
          kubernetes_job = "cray-ims-52e2d6f6-977f-4b91-b0fb-174dc7e195e2-
create"
          job_type = "create"
          id = "52e2d6f6-977f-4b91-b0fb-174dc7e195e2"
          created = "2020-11-06T20:40:51.823126+00:00"
          kubernetes_namespace = "ims"
          public_key_id = "4a629135-9ab6-4164-9fa2-86bd018a4c61"
          kernel_file_name = "vmlinuz"
          build_env_size = 10
          image_root_archive_name = "skern-sles15sp1-image"


      - Store the job id and kubernetes_job values from the output the IMS
job create


        ...
        kubernetes_job = "cray-ims-52e2d6f6-977f-4b91-b0fb-174dc7e195e2-
create"
        job_type = "create"
        id = "52e2d6f6-977f-4b91-b0fb-174dc7e195e2"
```

```
        ...

        - export IMS_JOB_ID=52e2d6f6-977f-4b91-b0fb-174dc7e195e2
        - export IMS_KUBERNETES_JOB=cray-ims-52e2d6f6-977f-4b91-
b0fb-174dc7e195e2-create

      - Use kubectl to describe the image create job. We want to identify
the POD doing the IMS image customization build.

        -  kubectl -n ims describe job $IMS_KUBERNETES_JOB
           ...
           Events:
             Type     Reason                Age    From            Message
             ----     ------                ----   ----            -------
             Normal   SuccessfulCreate      9m41s  job-controller  Created
pod: cray-ims-52e2d6f6-977f-4b91-b0fb-174dc7e195e2-create-tnk92

      - export POD=cray-ims-52e2d6f6-977f-4b91-b0fb-174dc7e195e2-create-tnk92

      - Check the kubernetes job containers for progress. Each kubernetes
log check is a tail that will not end until the container has completed. We
want to pend on each container in order. Once they have all completed, we
will be able to obtain the customized image id we then use for CFS image
customizations.

        - kubectl -n ims logs -f $POD -c fetch-recipe
        - kubectl -n ims logs -f $POD -c wait-for-repos
        - kubectl -n ims logs -f $POD -c build-ca-rpm
        - kubectl -n ims logs -f $POD -c build-image
        - kubectl -n ims logs -f $POD -c buildenv-sidecar

Store the resultant ID listed in the buildenv-sidecar container's log. You
can also use "cray ims jobs describe $IMS_JOB_ID" to obtain the
resultant_image_id.
```

```
ncn-m001 # export RESULTANT_IMAGE_ID=e3ba09d7-e3c2-4b80-9d86-0ee2c48c2214
```

d. If not already done, untar the 1.4.0 Day Zero Patch tarball.

```
ncn-m001 # tar -xvf shasta-1.4.0-p2.tar
1.4.0-p2/
1.4.0-p2/csm/
1.4.0-p2/csm/csm-0.8.22-0.9.0.patch.gz
1.4.0-p2/csm/csm-0.8.22-0.9.0.patch.gz.md5sum
1.4.0-p2/uan/
1.4.0-p2/uan/uan-2.0.0-uan-2.0.0.patch.gz
1.4.0-p2/uan/uan-2.0.0-uan-2.0.0.patch.gz.md5sum
1.4.0-p2/rpms/
1.4.0-p2/rpms/cray-dvs-compute-2.12_4.0.102-7.0.1.0_8.1__g30d29e7a.x86_64.rpm
1.4.0-p2/rpms/cray-dvs-devel-2.12_4.0.102-7.0.1.0_8.1__g30d29e7a.x86_64.rpm
1.4.0-p2/rpms/cray-dvs-kmp-
cray_shasta_c-2.12_4.0.102_k4.12.14_197.78_9.1.58-7.0.1.0_8.1__g30d29e7a.x86_
64.rpm
1.4.0-p2/rpms/cray-network-config-1.1.7-20210318094806_b409053-
sles15sp1.x86_64.rpm
1.4.0-p2/rpms/slingshot-network-config-1.1.7-20210318093253_83fab52-
sles15sp1.x86_64.rpm
1.4.0-p2/rpms/slingshot-network-config-full-1.1.7-20210318093253_83fab52-
sles15sp1.x86_64.rpm
1.4.0-p2/rpms/cray-dvs-
```

```
compute-2.12_4.0.102-7.0.1.0_8.1__g30d29e7a.x86_64.rpm.md5sum
1.4.0-p2/rpms/cray-dvs-
devel-2.12_4.0.102-7.0.1.0_8.1__g30d29e7a.x86_64.rpm.md5sum
1.4.0-p2/rpms/cray-dvs-kmp-
cray_shasta_c-2.12_4.0.102_k4.12.14_197.78_9.1.58-7.0.1.0_8.1__g30d29e7a.x86_
64.rpm.md5sum
1.4.0-p2/rpms/cray-network-config-1.1.7-20210318094806_b409053-
sles15sp1.x86_64.rpm.md5sum
1.4.0-p2/rpms/slingshot-network-config-1.1.7-20210318093253_83fab52-
sles15sp1.x86_64.rpm.md5sum
1.4.0-p2/rpms/slingshot-network-config-full-1.1.7-20210318093253_83fab52-
sles15sp1.x86_64.rpm.md5sum
```

e. Download the rootfs image which was created by the IMS job.

```
ncn-m001 # cray artifacts get boot-images ${RESULTANT_IMAGE_ID}/rootfs \
${RESULTANT_IMAGE_ID}.squashfs
ncn-m001 # la ${RESULTANT_IMAGE_ID}.squashfs
-rw-r--r-- 1 root root 1.5G Mar 17 19:35 e3ba09d7-
e3c2-4b80-9d86-0ee2c48c2214.squashfs
```

f. Mount the squashfs file and copy its contents to a different directory.

```
ncn-m001 # mkdir mnt
ncn-m001 # mount -t squashfs e3ba09d7-e3c2-4b80-9d86-0ee2c48c2214.squashfs \
mnt -o ro,loop
ncn-m001 # cp -a mnt compute-1.4.0-day-zero-CN
ncn-m001 # umount mnt
ncn-m001 # rmdir mnt
```

g. Copy the new RPMs into the new image directory. Chroot to the image.

`RPM -u` will update the RPMs.

When running the following commands, error messages may occur, as long as the packages are installed when running the commands a second time, they can safely be ignored.

```
ncn-m001# cp 1.4.0-p2/rpms/* compute-1.4.0-day-zero-CN/
ncn-m001# cd compute-1.4.0-day-zero-CN/
ncn-m001# chroot . bash
ncn-m001# rpm -Uv cray-dvs-*.rpm
ncn-m001# rpm -e cray-network-config
ncn-m001# rpm -e slingshot-network-config-full
ncn-m001# rpm -e slingshot-network-config
ncn-m001# rpm -iv slingshot-network-config-full-1.1.7-20210318093253_83fab52-
sles15sp1.x86_64.rpm \
slingshot-network-config-1.1.7-20210318093253_83fab52-sles15sp1.x86_64.rpm \
cray-network-config-1.1.7-20210318094806_b409053-sles15sp1.x86_64.rpm
ncn-m001# exit
exit
ncn-m001# rm *.rpm
ncn-m001# cd ..
```

h. Verify that there is only one sub-directory in the *lib/modules* directory for the image.

If there is more than one sub-directory, that is an indication that the image kernel does not match the new DVS RPMs being installed.

```
ncn-m001 # la compute-1.4.0-day-zero-CN/lib/modules/
total 8.0K
drwxr-xr-x 3 root root   49 Feb 25 17:50 ./
```

```
drwxr-xr-x 8 root root 4.0K Feb 25 17:52 ../
drwxr-xr-x 6 root root 4.0K Mar 17 19:49 4.12.14-197.78_9.1.58-cray_shasta_c/
```

i. Resquash the new image directory.

```
ncn-m001 # mksquashfs compute-1.4.0-day-zero-CN \
compute-1.4.0-day-zero-CN.squashfs
Parallel mksquashfs: Using 64 processors
Creating 4.0 filesystem on compute-1.4.0-day-zero-CN.squashfs, block size
131072.
```

j. Create a new IMS image to populate.

Note the "id" field.

```
ncn-m001 # cray ims images create --name compute-1.4.0-day-zero-CN
name = "compute-1.4.0-day-zero-CN"
created = "2021-03-17T20:23:05.576754+00:00"
id = "0c31e971-f990-4b5f-821d-c0c18daefb6e"
```

k. Upload the new image.

Use the UUID from the previous step.

```
ncn-m001 # cray artifacts create boot-images \
0c31e971-f990-4b5f-821d-c0c18daefb6e/compute-1.4.0-day-zero-CN.squashfs \
compute-1.4.0-day-zero-CN.squashfs
artifact = "0c31e971-f990-4b5f-821d-c0c18daefb6e/compute-1.4.0-day-zero-
CN.squashfs"
Key = "0c31e971-f990-4b5f-821d-c0c18daefb6e/compute-1.4.0-day-zero-
CN.squashfs"
```

l. Find the md5sum of the image.

```
ncn-m001 # md5sum compute-1.4.0-day-zero-CN.squashfs
db6a8934ad3c483e740c648238800e93  compute-1.4.0-day-zero-CN.squashfs
```

m. Create a manifest file like the following example.

Fill in the path from step k and the md5sum from step l.

```
ncn-m001 # cat manifest.json
{
   "artifacts" : [
      {
         "link" : {
            "path" : "s3://boot-images/0c31e971-f990-4b5f-821d-c0c18daefb6e/
compute-1.4.0-day-zero-CN.squashfs",
            "type" : "s3"
         },
         "md5" : "db6a8934ad3c483e740c648238800e93",
         "type" : "application/vnd.cray.image.rootfs.squashfs"
      }
   ],
   "created" : "20210317153136",
   "version" : "1.0"
}
```

n. Upload the manifest file.

```
ncn-m001 # cray artifacts create boot-images \
0c31e971-f990-4b5f-821d-c0c18daefb6e/manifest.json manifest.json
artifact = "0c31e971-f990-4b5f-821d-c0c18daefb6e/manifest.json"
Key = "0c31e971-f990-4b5f-821d-c0c18daefb6e/manifest.json"
```

o. Update the IMS image to use the new manifest.json.

```
ncn-m001 # cray ims images update \
0c31e971-f990-4b5f-821d-c0c18daefb6e --link-type s3 --link-path \
s3://boot-images/0c31e971-f990-4b5f-821d-c0c18daefb6e/manifest.json
created = "2021-03-17T20:23:05.576754+00:00"
id = "0c31e971-f990-4b5f-821d-c0c18daefb6e"
name = "compute-1.4.0-day-zero-CN"

[link]
etag = ""
path = "s3://boot-images/0c31e971-f990-4b5f-821d-c0c18daefb6e/manifest.json"
type = "s3"
```

p. Update the `RESULTANT_IMAGE_ID` with the new value.

```
ncn-m001 # export RESULTANT_IMAGE_ID=0c31e971-f990-4b5f-821d-c0c18daefb6e
```

q. Create a CFS session to do pre-boot image customization using the CFS session configuration that was just created.

Retrieve the image ID to be customized from the product catalog and supply the session configuration name from the previous step.

```
ncn-m001# cray cfs sessions create --name cos-config-2.0.10 \
--configuration-name cos-config-2.0.10 --target-definition image \
--target-group Compute ${RESULTANT_IMAGE_ID} --format json
{
  "ansible": {
    "config": "cfs-default-ansible-cfg",
    "verbosity": 0
  },
  "configuration": {
    "limit": "",
    "name": "cos-config-2.0.10"
  },
  "id": "41b9c2ae-575c-43f4-84ef-045b373c03eb",
  "links": [
    {
      "href": "/v2/sessions/cos-config-2.0.10",
      "rel": "self"
    },
    {
      "href": "/apis/cms.cray.com/v2/namespaces/services/cfsessions/cos-config-2.0.10",
      "rel": "k8s"
    }
  ],
  "name": "cos-config-2.0.10",
  "status": {
    "artifacts": [],
    "session": {
      "status": "pending",
      "succeeded": "none"
    }
  },
  "target": {
    "definition": "image",
    "groups": [
      {
        "members": [
          "e3ba09d7-e3c2-4b80-9d86-0ee2c48c2214"
        ],
        "name": "Compute"
      }
```

```
    ]
  }
}
```

14. Record the IMS image ID of the customized COS image given in the CFS session.

When the CFS session has completed, a new COS image will be available.

```
ncn-m001# cray cfs sessions describe cos-config-2.0.10 --format json \
| jq -r .status.artifacts[].result_id
5f3fb8a7-4189-4d04-b0c3-eec98fdc6440

ncn-m001# export IMAGE_ID=5f3fb8a7-4189-4d04-b0c3-eec98fdc6440

ncn-m001# cray artifacts describe boot-images ${IMAGE_ID}/manifest.json --format json
{
  "artifact": {
    "AcceptRanges": "bytes",
    "LastModified": "2021-01-30T01:20:33+00:00",
    "ContentLength": 1147,
    "ETag": "\"151cd5effbda573d8d05b3429c150e62\"",
    "ContentType": "binary/octet-stream",
    "Metadata": {
      "md5sum": "151cd5effbda573d8d05b3429c150e62"
    }
  }
}

ncn-m001# export ETAG_VALUE=151cd5effbda573d8d05b3429c150e62
```

BOOT PREPARATION

15. Verify cray-conman is connected to the node(s) being booted.

The cray-conman service determines which nodes it should monitor by checking with the Hardware State Manager (HSM) service. It does this *once* when it starts. If HSM has not discovered some nodes when conman starts, then HSM is unaware of them and so is conman. Therefore, it is important to verify that conman is monitoring all nodes' console logs. If it is not, re-initialize the cray-conman service and recheck the nodes it is monitoring. `conman -q` can be used to list the existing connections.

Use kubectl to exec into the running cray-conman pod, then check the existing connections.

```
cray-conman-b69748645-qtfxj:/ # conman -q
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

If the compute nodes and UANs are not included in the list of nodes being monitored, the conman process can be re-initialized by killing the conmand process.

```
cray-conman-b69748645-qtfxj:/ # ps -ax | grep conmand
     13 ?        Sl     0:45 conmand -F -v -c /etc/conman.conf
  56704 pts/3    S+     0:00 grep conmand
cray-conman-b69748645-qtfxj:/ # kill 13
```

This will regenerate the conman configuration file and restart the conmand process, and now include all nodes that are included in the state manager.

```
cray-conman-b69748645-qtfxj:/ # conman -q
x9000c1s7b0n1
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

16. Construct a boot session template using the xnames of the compute nodes, the customized image ID, and the CFS session configuration name.

    a. Create a boot session template using the `ETAG_VALUE` and `IMAGE_ID` identified in the previous step.

       Manually insert `ETAG_VALUE` and `IMAGE_ID` into the file. This file needs to be created manually. **Do not attempt to cut and paste the following example. This will lead to an incorrectly formatted json syntax.**

       ```
       ncn-m001# vi cos-sessiontemplate-2.0.10.json
       {
         "boot_sets": {
           "compute": {
             "boot_ordinal": 2,
             "kernel_parameters": "console=ttyS0,115200 bad_page=panic crashkernel=340M hugepagelist=2m-2g
       intel_iommu=off intel_pstate=disable iommu=pt ip=dhcp numa_interleave_omit=headless numa_zonelist_order=node
       oops=panic pageblock_order=14 pcie_ports=native printk.synchronous=y rd.neednet=1 rd.retry=10 rd.shell
       turbo_boost_limit=999 spire_join_token=${SPIRE_JOIN_TOKEN}",
             "network": "nmn",
             "node_roles_groups": [
               "Compute"
             ],
             "path": "s3://boot-images/<IMAGE_ID>/manifest.json",,
             "rootfs_provider": "cpss3",
             "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:nmn0",
             "etag": "<ETAG_VALUE>",
             "type": "s3"
           }
         },
         "cfs": {
           "configuration": "cos-config-2.0.10"
         },
         "enable_cfs": true,
         "name": "cos-sessiontemplate-2.0.10"
       }
       ```

    b. Register the session template file with the Boot Orchestration Service (BOS).

       For more information about BOS templates, refer to *BOS Session Templates* on page 173.

       Any name or string may be used for the `sesstiontemplate --name` in the following example.

       ```
       ncn-m001# cray bos v1 sessiontemplate create --file cos-sessiontemplate-2.0.10.json \
       --name cos-sessiontemplate-2.0.10
       /sessionTemplate/cos-sessiontemplate-2.0.10
       ```

17. Create a BOS Session to boot the compute nodes.

    ```
    ncn-m001# cray bos v1 session create --template-uuid \
     cos-sessiontemplate-2.0.10 --operation reboot
    ```

    The first attempt to reboot the compute nodes will most likely fail. The compute node boot may hang and the compute node console will look similar to the following:

    ```
    2021-03-19 01:32:41 dracut-initqueue[420]: DVS: node map generated.
    2021-03-19 01:32:41 katlas: init_module: katlas loaded, currently disabled
    2021-03-19 01:32:41
    2021-03-19 01:32:41 DVS: Revision: kbuild Built: Mar 17 2021 @ 15:14:05 against
    LNet 2.12.4
    ```

```
2021-03-19 01:32:41 DVS debugfs: Revision: kbuild Built: Mar 17 2021 @ 15:14:05
against LNet 2.12.4
2021-03-19 01:32:41 dracut-initqueue[420]: DVS: loadDVS: successfully added 10
new nodes into map.
2021-03-19 01:32:41 ed dvsproc module.
2021-03-19 01:32:41 DVS: message size checks complete.
2021-03-19 01:32:41 dracut-initqueuedvs_thread_generator: Watching pool DVS-
IPC_msg (id 0)
2021-03-19 01:32:41 [420]: DVS: loaded dvs module.
2021-03-19 01:32:41 dracut-initqueue[420]: mount is: /opt/cray/cps-utils/bin/
cpsmount.sh -a api-gw-service-nmn.local -t dvs -T 300 -i nmn0 -e
3116cf653e84d265cf8da94956f34d9e-181 s3://boot-images/763213c7-3d5f-4f2f-9d8a-
ac6086583f43/rootfs /tmp/cps
2021-03-19 01:32:41 dracut-initqueue[420]: 2021/03/19 01:31:01 cpsmount_helper
Version: 1.0.0
2021-03-19 01:32:47 dracut-initqueue[420]: 2021/03/19 01:31:07 Adding content:
s3://boot-images/763213c7-3d5f-4f2f-9d8a-ac6086583f43/rootfs
3116cf653e84d265cf8da94956f34d9e-181 dvs
2021-03-19 01:33:02 dracut-initqueue[420]: 2021/03/19 01:31:22 WARN:
readyForMount=false type=dvs ready=0 total=2
2021-03-19 01:33:18 dracut-initqueue[420]: 2021/03/19 01:31:38 WARN:
readyForMount=false type=dvs ready=0 total=2
2021-03-19 01:33:28 dracut-initqueue[420]: 2021/03/19 01:31:48 2 dvs servers
[10.252.1.7 10.252.1.8]
```

If this occurs, repeat the BOS command.

For more information regarding the compute boot process, refer to *Compute Node Boot Sequence* on page 175 and *Boot Orchestration Service (BOS)* on page 175.

18. Obtain the SSH private key used for compute nodes. Refer to *Passwordless SSH* on page 263.

19. SSH to a newly booted compute node.

```
ncn-m001 # ssh nid001000-nmn
Last login: Wed Mar 17 19:10:12 2021 from 10.252.1.12
nid001000:~ #
```

20. Verify that the DVS RPM versions match what exists in the `1.4.0-p2/rpms` directory.

```
nid001000:~ # rpm -qa | grep 'cray-dvs.*2\.12' | sort
cray-dvs-compute-2.12_4.0.102-7.0.1.0_8.1__g30d29e7a.x86_64
cray-dvs-devel-2.12_4.0.102-7.0.1.0_8.1__g30d29e7a.x86_64
cray-dvs-kmp-
cray_shasta_c-2.12_4.0.102_k4.12.14_197.78_9.1.58-7.0.1.0_8.1__g30d29e7a.x86_64
```

21. Verify the the `network-config` RPM versions match what exists in the 1.4.0-p2/rpms directory.

```
nid001000:~ # rpm -qa | grep network-config | sort
cray-network-config-1.1.7-20210318094806_b409053-sles15sp1.x86_64
slingshot-network-config-1.1.7-20210318093253_83fab52-sles15sp1.x86_64
slingshot-network-config-full-1.1.7-20210318093253_83fab52-sles15sp1.x86_64
```

22. Log out of the compute node.

```
nid000001:~ # exit
```

23. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

If licensed to run Cray's Programming Environment, refer to *HPE Cray Programming Environment Installation Guide: CSM on HPE Cray EX Systems* S-8003 before continuing on to *Install the Analytics Product Stream Update* on page 329.

14.1 Enable NCN Personalization

Prerequisites

- Products which include configuration content to be applied to the management NCNs must be installed. These products may include those which configure management functionality, such as COS, SMA, and CSM, or optional functionality to enable user productivity, such as PE and Analytics.

About this task

ROLE
- System installer, system administrator

OBJECTIVE This procedure enables key components of the HPE Cray EX CSM to function.

NEW IN THIS RELEASE This entire procedure is new with this release.

NCN Personalization performs automated post-boot configuration tasks on the HPE Cray EX NCNs. Many HPE Cray EX management services (including DVS and SMA) require NCN personalization to function.

In this release, NCN Personalization must be set up manually. In general, this process requires two steps:

1. **Create and upload a CFS Configuration file.** This file lists the Ansible playbooks that configure each NCN.
2. **Update the CFS components.** This step tells CFS to apply the CFS configuration automatically to a specific node.

Hewlett Packard Enterprise recommends the configurations for both CNs and NCNs of nodes use the same Git commits. This requirement ensures that the configurations remain compatible. This compatibility is necessary for services like DVS which require low-level compatibility between CNs and NCNs.

Procedure

Create a CFS configuration JSON file

1. Obtain a list of the branches of the `cos-config-management` VCS repository on the HPE Cray EX system.

   The COS layer must always be present in the `ncn-personalization` CFS configuration. Otherwise, the CNs will not be able to boot using DVS. The COS layer configures DVS and LNet. It can also mount Lustre and other file systems on NCN worker nodes.

   ```
   ncn-m# git ls-remote https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git \
   refs/heads/cray/cos/*
   ccd964f8eeeb52ab8f895b480c5d1142c7bc0a8e    refs/heads/cray/cos/2.0.17
   ccd964f8eeeb52ab8f895b480c5d1142c7bc0a8e    refs/heads/cray/cos/2.0.18
   ```

2. Obtain the git commit ID for the branch that is to be configured.

In this procedure, the `2.0.18` branch will be modified. The git commit ID for this branch can be obtained in the output of the preceding command or by running the following:

```
ncn-w001# git ls-remote https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git \
refs/heads/cray/cos/2.0.18 | awk '{print $1}'
ccd964f8eeeb52ab8f895b480c5d1142c7bc0a8e
```

The commit value is needed for the CFS configuration JSON file.

3. Repeat the previous two steps for the `sma-config-management` VCS repository.

```
ncn-m# git ls-remote https://api-gw-service-nmn.local/vcs/cray/sma-config-management.git \
refs/heads/cray/sma/*
. . .
ncn-m# git ls-remote https://api-gw-service-nmn.local/vcs/cray/sma-config-management.git \
refs/heads/cray/sma/1.4.2 | awk '{print $1}'
eb724e7135dc60d3fdfff9fb01672538f241e588
```

This commit value is also needed for the CFS configuration JSON file.

4. On the NCN worker and manager nodes, disable residual zypper services that refer to external URIs.

   1. List zypper services. For example, sort the list by URI:

      ```
      ncn-m001# zypper ls -U
      ```

   2. Disable zypper services that refer to external URIs. For example:

      ```
      ncn-m001# NCNS=ncn-m00[1-3],ncn-w00[1-3]
      ```

      ```
      ncn-m001# pdsh -w $NCNS \
      "zypper ms -d Basesystem_Module_15_SP2_x86_64; \
      zypper ms -d Public_Cloud_Module_15_SP2_x86_64; \
      zypper ms -d SUSE_Linux_Enterprise_Server_15_SP2_x86_64; \
      zypper ms -d Server_Applications_Module_15_SP2_x86_64"
      ```

5. Create a CFS configuration JSON file for NCN personalization. Add layers as needed if installing the Cray Programming Environment or any other optional products.

   All products that must exist on the NCNs must have a corresponding layer in the CFS JSON file. Each node in the HPE Cray EX system can only have one configuration applied. The COS layer must always be the first layer in the file. This procedure assumes that the example CFS configuration file is saved as `ncn-personalization.json` on a master NCN. Replace the values for `commit` and `cloneUrl` with the values obtained in the previous three steps.

   The `sma-base-config` and `sma-ldms-ncn` layers are required for SMA.

```
ncn-m# vi ncn-personalization.json
ncn-m# cat ncn-personalization.json
{
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-
management.git",
      "commit": "ccd964f8eeeb52ab8f895b480c5d1142c7bc0a8e",
      "name": "cos-integration-2.0.18",
      "playbook": "ncn.yml"
    },
    {
      "name": "sma-base-config",
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/sma-config-
management.git",
```

```
      "playbook": "sma-base-config.yml",
      "commit": "eb724e7135dc60d3fdfff9fb01672538f241e588"
    },
    {
      "name": "sma-ldms-ncn",
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/sma-config-
management.git",
      "playbook": "sma-ldms-ncn.yml",
      "commit": "eb724e7135dc60d3fdfff9fb01672538f241e588"
    }
  ]
}
```

6. **Optional:** Add configuration layers for other installed products. Obtain the git commit ID and `cloneUrl` for the appropriate branch or branches using the same method used for COS and SMA.

   Add the following layer to make the HPE Cray Programming Environment (PE) available on worker NCNs. A PE license must be purchased and the PE tar files must be installed.

   ```
   {
     "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/pe-config-
   management.git",
     "commit": "0d246ca5bc695f681c9fef2b2c19d0a1dc9f1c9b",
     "name": "pe-21.02",
     "playbook": "pe_deploy.yml"
   }
   ```

   The following example layer enables the Analytics product content available on the NCN worker nodes.

   ```
   {
     "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/analytics-config-
   management.git",
     "commit": "3a039f149fd75351f155f1ea0e4e32f478fdc5f1",
     "name": "analytics-integration-1.0.0",
     "playbook": "site.yml"
   }
   ```

   Upload and apply the CFS configuration file

7. Upload the configuration file to CFS and give the configuration a name.

   ```
   ncn-m# cray cfs configurations update ncn-personalization --file \
   ncn-personalization.json --format json
   {
     "lastUpdated": "2020-12-16T16:34:19Z",
     "layers": [
       {
         "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-
   management.git",
         "commit": "ccd964f8eeeb52ab8f895b480c5d1142c7bc0a8e",
         "name": "cos-integration-2.0.18",
         "playbook": "ncn.yml"
       },
       {
         "name": "sma-base-config",
         "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/sma-config-
   management.git",
         "playbook": "sma-base-config.yml",
         "commit": "eb724e7135dc60d3fdfff9fb01672538f241e588"
       },
   ```

```
    {
      "name": "sma-ldms-ncn",
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/sma-config-
management.git",
      "playbook": "sma-ldms-ncn.yml",
      "commit": "eb724e7135dc60d3fdfff9fb01672538f241e588"
    }
  ],
  "name": "ncn-personalization"
}
```

When running the above step, the CFS session may fail. The `TrustedUserCAKeys` may not exist. Run `systemctl restart cfs-state-reporter` on all NCNs which will automatically add the `TrustedUserCAKeys` entry to `/etc/ssh/sshd_config`.

```
ncn-m001# systemctl restart cfs-state-reporter
```

8. Verify that the following command runs successfully on each Kubernetes worker node.

   This step does not load any LNet/DVS kernel modules, which won't happen until after Steps 10 and 11.

   ```
   ncn-w# /opt/cray/dvs/default/sbin/dvs_generate_map -D --dummy
   ```

   This step is to check for potential issues that DVS may encounter on the NCNs. These issues are easier to troubleshoot at this stage rather than later, when the new configuration is actually applied to the NCNs. At that point, Ansible log analysis will be necessary to troubleshoot.

   Some of the potential issues that if encountered by `dvs_generate_map` should be resolved by the system administrator before continuing to the next step are:

   - **Failure to fetch authentication token:** the `dvs_generate_map: fetching auth token failed` error will appear in this case. This is typically caused by a SPIRE configuration issue on the node.
   - **HSM errors:** for example, messages about node xname not being present, or no Management nodes.
   - **DNS error from the NCN looking up its own xname**

   The `dvs_generate_map` script will retry and wait for any of these cases to be resolved. Ignore any DNS errors relating to CN or UAN xname lookups.

9. Press the **Ctrl** and **C** keys to exit the `dvs_generate_map` script if it loops for more than 30 seconds for any of the three reasons listed in the previous step. Then resolve the underlying problem.

   Skip this step if the `dvs_generate_map` ran successfully on the node.

10. **Optional:** Obtain the xnames of all NCNs that will be configured by NCN Personalization by running this command on each node:

    Skip this step if the required xnames have already been obtained.

    ```
    ncn-m# ssh ncn-w001 cat /etc/cray/xname
    x3000c0s7b0n0
    ```

11. Update the CFS component for all NCNs. Replace *`NCN_XNAME`* in the following command with the xname of an NCN.

    ```
    ncn-m# cray cfs components update --desired-config ncn-personalization \
    --enabled true --format json NCN_XNAME
    ```

    After the previous command is issued, the `cfs-state-reporter` pod will dispatch a CFS job to configure the NCN. The same will happen every time the NCN boots.

12. **Optional:** Query the status of the NCN Personalization process.

    The following example command for node x3000c0s7b0n0 will return `"configurationStatus": "pending"` until that configuration completes. When it completes, the command will return `"configurationStatus": "configured"`.

```
ncn-m# cray cfs components describe x3000c0s7b0n0 --format json
{
  "configurationStatus": "pending",
  "desiredConfig": "ncn-personalization",
  "enabled": true,
  "errorCount": 0,
  "id": "x3000c0s7b0n0",
  "retryPolicy": 3,
  "state": []
}
ncn-m# cray cfs components describe x3000c0s7b0n0 --format json
{
  "configurationStatus": "configured",
  "desiredConfig": "ncn-personalization",
  "enabled": true,
  "errorCount": 0,
  "id": "x3000c0s7b0n0",
  "retryPolicy": 3,
  "state": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-
management.git",
      "commit": "4f8e919ede7143929e26ee8d97e04c12572a2f90_skipped",
      "lastUpdated": "2020-12-15T21:04:32Z",
      "playbook": "ncn.yml",
      "sessionName": "batcher-d4aa7928-0c59-4057-9bd6-2c819f3f9b7d"
    }
  ]
}
```

13. Repeat Steps 9, 10, and 11 for all master and worker NCNs.

    This will ensure that all NCNs will be automatically configured by NCN Personalization.

15 Prepare for UAN Product Installation

Prerequisites

Install and Configure the Cray Operating System (COS) on page 87

About this task

ROLE	System administrator, system installer
OBJECTIVE	Make the HPE Cray EX supercomputer ready for UAN product installation.
NEW IN THIS RELEASE	This procedure is new in this release

This section describes the prerequisites for installing the User Access Nodes (UAN) product on Cray EX systems.

Procedure

MANAGEMENT NETWORK SWITCH CONFIGURATION

1. Ensure that the management network switches are properly configured.

   Refer to *Install the Management Network* on page 267.

2. Ensure that the management network switches have the proper firmware.

   Refer to the procedure "Update the Management Network Firmware" in the publication *HPE Cray EX Hardware Management Administration Guide (S-8015)*.

3. Ensure that the host reservations for the UAN CAN network have been properly set.

   Refer to the procedure "Add UAN CAN IP Addresses to SLS" in the publication *HPE Cray EX Hardware Management Administration Guide (S-8015)*.

BMC CONFIGURATION

4. Configure the BMC of the UAN.
   - Perform *Configure the BMC for UANs with iLO* on page 172 if the UAN is a HPE server with an iLO.

BIOS CONFIGURATION

5. Configure the BIOS of the UAN.
   - Perform *Configure the BIOS of an HPE UAN* on page 170 if the UAN is a HPE server with an iLO.

- Perform *Configure the BIOS of a Gigabyte UAN* on page 169 if the UAN is a Gigabyte server.

VERIFY UAN BMC FIRMWARE VERSION

6. Ensure the firmware for each UAN BMC meets the specifications.

   Use the System Admin Toolkit `firmware` command to check the current firmware version on a UAN node. Refer to *Node Firmware* on page 244.

   ```
   ncn-m001# sat firmware -x BMC_XNAME
   ```

7. Repeat the previous six Steps for all UANs.

VERIFY REQUIRED SOFTWARE FOR UAN INSTALLATION

8. Perform *Apply the UAN Patch* on page 44 to apply any needed patch content for the UAN product.

   It is critical to perform this process to ensure that the correct UAN release artifacts are deployed.

9. Unpackage the file.

   ```
   ncn-m001# tar zxf uan-PRODUCT_VERSION.tar.gz
   ```

10. Navigate into the `uan-PRODUCT_VERSION/` directory.

    ```
    ncn-m001# cd uan-PRODUCT_VERSION/
    ```

11. Run the pre-install goss tests to determine if the system is ready for the UAN product installation.

    This requires that goss is installed on the node running the tests. Skip this step to if the automated tests can not be run.

    ```
    ncn# ./validate-pre-install.sh
    ..............

    Total Duration: 1.304s
    Count: 15, Failed: 0, Skipped: 0
    ```

12. Run the `./tests/goss/scripts/uan_preflight_same_in_sls_and_hsm.py` script if the previous step reports an error for the `uans_same_in_sls_and_hsm` Goss test. Address any errors that are reported.

    This script must be run rerun manually because this test produces erroneous failures otherwise.

13. Manually verify the UAN software prerequisites.

    a. Verify that the `cray` CLI tool, `manifestgen`, and `loftsman` are installed.

       ```
       ncn-m001# which cray
       /usr/bin/cray
       ncn-m001# which manifestgen
       /usr/bin/manifestgen
       ncn-m001# which loftsman
       /usr/bin/loftsman
       ```

    b. Verify that Helm is installed and is at least version 3 or greater.

```
ncn-m001# which helm
/usr/bin/helm
ncn-m001# helm version
version.BuildInfo{Version:"v3.2.4", GitCommit:"0ad800ef43d3b826f31a5ad8dfbb4fe05d143688",
GitTreeState:"clean", GoVersion:"go1.13.12"}
```

c. Verify that the Cray System Management (CSM) software has been successfully installed and is running on the system.

   The following Helm releases should be installed and verified:

   - gitea
   - cray-product-catalog
   - cray-cfs-api
   - cray-cfs-operator
   - cray-ims

   The following command checks that all these releases are present and have a status of `deployed`:

```
ncn-m001# helm ls -n services -f '^gitea$|cray-cfs-operator|cray-cfs-api|cray-ims|cray-product-
catalog'\
 -o json | jq -r '.[] | .status + " " + .name'
deployed cray-cfs-api
deployed cray-cfs-operator
deployed cray-ims
deployed cray-product-catalog
deployed gitea
```

d. Verify `cray-conman` is connected to compute nodes and UANs.

   Sometimes the compute nodes an UAN are not up yet when `cray-conman` is initialized and will not be monitored yet. Verify that all nodes are being monitored for console logging or re-initialize `cray-conman` if needed.

   Use `kubectl` to `exec` into the running `cray-conman` pod, then check the existing connections.

```
cray-conman-b69748645-qtfxj:/ # conman -q
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

   If the compute nodes and UANs are not included in the list of nodes being monitored, the `conman` process can be re-initialized by killing the conmand process.

```
cray-conman-b69748645-qtfxj:/ # ps -ax | grep conmand
     13 ?        Sl     0:45 conmand -F -v -c /etc/conman.conf
  56704 pts/3    S+     0:00 grep conmand
cray-conman-b69748645-qtfxj:/ # kill 13
```

   This will regenerate the conman configuration file and restart the conmand process, and now include all nodes that are included in the state manager.

```
cray-conman-b69748645-qtfxj:/ # conman -q
x9000c1s7b0n1
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
```

```
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

e. Verify that the HPE Cray OS (COS) has been installed on the system.

   COS is required to build UAN images from the recipe and to boot UAN nodes.

```
ncn-m001# kubectl get cm -n services cray-product-catalog -o json | jq '.data | has("cos")'
true
```

f. Verify that the Data Virtualization Service (DVS) and LNet are configured on the nodes that are running Content Projection Service (CPS) `cps-cm-pm` pods provided by COS.

   The UAN product can be installed prior to this configuration being complete, but the the DVS modules must be loaded prior to booting UAN nodes. The following commands determine which nodes are running `cps-cm-pm` and then verifies that those nodes have the DVS modules loaded.

```
ncn-m001# kubectl get nodes -l cps-pm-node=True -o custom-columns=":metadata.name" --no-headers
ncn-w001
ncn-w002
ncn-m001# for node in `kubectl get nodes -l cps-pm-node=True -o custom-columns=":metadata.name"
--no-headers`; do
ssh $node "lsmod | grep '^dvs '"
done
ncn-w001
ncn-w002
```

   More nodes or a different set of nodes may be displayed.

Next, install the UAN product by peforming the procedure *Install the UAN Product Stream* on page 109.

16 Install the UAN Product Stream

Prerequisites

- The Cray command line interface (CLI) tool is initialized and configured on the system. See "Configure the Cray Command Line Interface (CLI)" in the *HPE Cray EX System Administration Guide S-8001* for more information.
- *Prepare for UAN Product Installation* on page 105

About this task

ROLE	System administrator, system installer
OBJECTIVE	Install the User Access Nodes (UAN) product on a HPE Cray EX system so that UAN boot images can be created.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in this release.

Replace *PRODUCT_VERSION* in the example commands with the UAN product stream string (**2.0.0** for example). Replace *CRAY_EX_DOMAIN* in the example commands with the FQDN of the HPE Cray EX.

Procedure

DOWNLOAD AND PREPARE THE UAN SOFTWARE PACKAGE

1. Start a typescript to capture the commands and output from this installation.

```
ncn-m001# script -af product-uan.$(date +%Y-%m-%d).txt
ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Run the installation script using either one of the following commands:
   - If the HPE Cray EX is configured for online installations, run this command:

     ```
     ncn-m001# ./install.sh --online
     ```

   - If the HPE Cray EX is configured for offline (that is, air-gapped) installations, run this command:

     ```
     ncn-m001# ./install.sh
     ```

VERIFY THE INSTALLATION

3. Verify that the UAN configuration, images, and recipes have been imported and added to the `cray-product-catalog` ConfigMap in the Kubernetes `services` namespace.

   a. Run the following command and verify that the output contains an entry for the *`PRODUCT_VERSION`* that was installed in the previous steps:

      The following command may return more than one version of the UAN product if previous versions have been installed.

      ```
      ncn-m001# kubectl get cm cray-product-catalog -n services -o json | jq -
      r .data.uan
      PRODUCT_VERSION:
           configuration:
             clone_url: https://vcs.CRAY_EX_DOMAIN/vcs/cray/uan-config-
      management.git
             commit: 6658ea9e75f5f0f73f78941202664e9631a63726
             import_branch: cray/uan/PRODUCT_VERSION
             import_date: 2021-07-28 03:26:00.399670
             ssh_url: git@vcs.CRAY_EX_DOMAIN:cray/uan-config-management.git
           images:
             cray-shasta-uan-cos-sles15sp1.x86_64-0.1.17:
               id: c880251d-b275-463f-8279-e6033f61578b
           recipes:
             cray-shasta-uan-cos-sles15sp1.x86_64-0.1.17:
               id: cbd5cdf6-eac3-47e6-ace4-aa1aecb1359a
      ```

   b. Verify that the `configuration`, `images`, and `recipes` sections for the installed product version contain information similar to the example output in the previous command.

   c. Check the Kubernetes jobs responsible for importing the configuration, image, and recipe content if the command does not show content for the UAN product version that was installed.

      A `STATUS` of `Completed` indicates that the Kubernetes jobs completed successfully.

      ```
      ncn-m001# kubectl get pods -n services -l job-name=uan-config-import-
      @product_version@
      NAME                                READY   STATUS      RESTARTS   AGE
      uan-config-import-@product_version@-gsvrc   0/3     Completed   0          5m
      ncn-m001# kubectl get pods -n services -l job-name=uan-image-recipe-import-
      @product_version@
      NAME                                       READY   STATUS      RESTARTS   AGE
      uan-image-recipe-import-@product_version@-2fvr7   0/3     Completed
      0          6m
      ```

4. Verify that the UAN RPM repositories have been created in Nexus using either of the following methods:

   - Navigate to https://nexus.*`CRAY_EX_DOMAIN`*/#browse/browse in a web browser to view the list of repositories. Ensure that the following repositories are present:
     - uan-2.0.0-sle-15sp1
     - uan-2.0-sle-15sp1
   - Query Nexus through its REST API to display the repositories prefixed with the name `uan`:

     ```
     ncn-m001# curl -s -k https://packages.local/service/rest/v1/repositories |
     jq -r '.[] | select(.name | startswith("uan")) | .name'
     uan-2.0-sle-15sp1
     uan-2.0.0-sle-15sp1
     ```

5. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

Perform *Apply UAN Upgrade Patch* on page 321

17 Create UAN Boot Images

Prerequisites

Install the UAN Product Stream on page 109

The UAN product stream must be installed. Refer to the publication *HPE Cray EX System Installation and Configuration Guide (S-8000)*.

About this task

ROLE	System administrator, system installer
OBJECTIVE	This procedure creates images for booting UANs.
LIMITATIONS	This guide only details how to apply UAN-specific configuration to the UAN image and nodes. Consult the manuals for the individual HPE products (for example, workload managers and the HPE Cray Programming Environment) that must be configured on the UANs.
NEW IN THIS RELEASE	This procedure is new in this release.

This is the overall workflow for preparing UAN images for booting UANs:

1. Clone the UAN configuration git repository and create a branch based on the branch imported by the UAN installation.
2. Update the configuration content and push the changes to the newly created branch.
3. Create a Configuration Framework Service (CFS) configuration for the UANs, specifying the git configuration and the UAN image to apply the configuration to. More Cray products can also be added to the CFS configuration so that the UANs can install multiple Cray products into the UAN image at the same time.
4. Configure the UAN image using CFS and generate a newly configured version of the UAN image.
5. Create a Boot Orchestration Service (BOS) boot session template for the UANs. This template maps the configured image, the CFS configuration to be applied post-boot, and the nodes which will receive the image and configuration.

Once the UAN BOS session template is created, the UANs will be ready to be booted by a BOS session.

Replace *`PRODUCT_VERSION`* and *`CRAY_EX_HOSTNAME`* in the example commands in this procedure with the current UAN product version installed (See Step 1) and the hostname of the HPE Cray EX system, respectively.

Procedure

UAN IMAGE PRE-BOOT CONFIGURATION

1. Obtain the artifact IDs and other information from the `cray-product-catalog` Kubernetes ConfigMap. Record the information labeled in the following example.

   Upon successful installation of the UAN product, the UAN configuration, image recipes, and prebuilt boot images are cataloged in this ConfigMap. This information is required for this procedure.

   ```
   ncn-m001# kubectl get cm -n services cray-product-catalog -o json | jq -r .data.uan
   PRODUCT_VERSION:
     configuration:
       clone_url: https://vcs.CRAY_EX_HOSTNAME/vcs/cray/uan-config-management.git # <--- Gitea clone url
       commit: 6658ea9e75f5f0f73f78941202664e9631a63726                  # <--- Git commit id
       import_branch: cray/uan/PRODUCT_VERSION                          # <--- Git branch with
   configuration
       import_date: 2021-02-02 19:14:18.399670
       ssh_url: git@vcs.CRAY_EX_HOSTNAME:cray/uan-config-management.git
     images:
       cray-shasta-uan-cos-sles15sp1.x86_64-0.1.17:                      # <--- IMS image name
         id: c880251d-b275-463f-8279-e6033f61578b                        # <--- IMS image id
     recipes:
       cray-shasta-uan-cos-sles15sp1.x86_64-0.1.17:                      # <--- IMS recipe name
         id: cbd5cdf6-eac3-47e6-ace4-aa1aecb1359a                        # <--- IMS recipe id
   ```

2. Generate the password hash for the `root` user. Replace *`PASSWORD`* with the `root` password you wish to use.

   ```
   ncn-m001# openssl passwd -6 -salt $(< /dev/urandom tr -dc _A-Z-a-z-0-9 | head -
   c4) PASSWORD
   ```

3. Obtain the HashiCorp Vault `root` token.

   ```
   ncn-m001# kubectl get secrets -n vault cray-vault-unseal-keys -o
   jsonpath='{.data.vault-root}' \
   | base64 -d; echo
   ```

4. Write the password hash obtained in Step 2 to the HashiCorp Vault.

   The `vault login` command will request a token. That token value is the output of the previous step. The `vault read secret/uan` command verifies that the hash was stored correctly. This password hash will be written to the UAN for the `root` user by CFS.

   ```
   ncn-m001# kubectl exec -itn vault cray-vault-0 -- sh
   export VAULT_ADDR=http://cray-vault:8200
   vault login
   vault write secret/uan root_password='HASH'
   vault read secret/uan
   ```

5. Obtain the password for the `crayvcs` user from the Kubernetes secret for use in the next command.

   ```
   ncn-m001# kubectl get secret -n services vcs-user-credentials \
   --template={{.data.vcs_password}} | base64 --decode
   ```

6. Clone the UAN configuration management repository. Replace *`CRAY_EX_HOSTNAME`* in clone url with **api-gw-service-nmn.local** when cloning the repository.

   The repository is in the VCS/Gitea service and the location is reported in the `cray-product-catalog` Kubernetes ConfigMap in the `configuration.clone_url` key. The *`CRAY_EX_HOSTNAME`* from the `clone_url` is replaced with `api-gw-service-nmn.local` in the command that clones the repository.

   ```
   ncn-m001# git clone https://api-gw-service-nmn.local/vcs/cray/uan-config-
   management.git
   . . .
   ncn-m001# cd uan-config-management && git checkout cray/uan/PRODUCT_VERSION &&
   ```

```
git pull
Branch 'cray/uan/PRODUCT_VERSION' set up to track remote branch 'cray/uan/
PRODUCT_VERSION' from 'origin'.
Already up to date.
```

7. Create a branch using the imported branch from the installation to customize the UAN image.

   This imported branch will be reported in the `cray-product-catalog` Kubernetes ConfigMap in the `configuration.import_branch` key under the UAN section. The format is `cray/uan/`*`PRODUCT_VERSION`*. In this guide, an `integration` branch is used for examples, but the name can be any valid git branch name.

   Modifying the `cray/uan/`*`PRODUCT_VERSION`* branch that was created by the UAN product installation is not allowed by default.

   ```
   ncn-m001# git checkout -b integration && git merge cray/uan/PRODUCT_VERSION
   Switched to a new branch 'integration'
   Already up to date.
   ```

8. Configure a root user in the UAN image by adding the encrypted password of the root user from `/etc/shadow` on an NCN worker to the file `group_vars/Application/passwd.yml`. Skip this step if the root user is already configured in the image.

   Hewlett Packard Enterprise recommends configuring a root user in the UAN image for troubleshooting purposes. The entry for root user password will resemble the following example:

   ```
   root_passwd: $6$LmQ/PlWlKixK$VL4ueaZ8YoKOV6yYMA9iH0gCl8F4C/3yC.jMIGfOK6F61h6d.iZ6/
   QB0NLyex1J7AtOsYvqeycmLj2fQcLjfE1
   ```

9. Apply any site-specific customizations and modifications to the Ansible configuration for the UAN nodes and commit the changes.

   The default Ansible play to configure UAN nodes is `site.yml` in the base of the `uan-config-management` repository. The roles that are executed in this play allow for nondefault configuration as required for the system.

   Consult the individual Ansible role `README.md` files in the `uan-config-management` repository roles directory to configure individual role variables. Roles prefixed with `uan_` are specific to UAN configuration and include network interfaces, disk, LDAP, software packages, and message of the day roles.

   Variables should be defined and overridden in the Ansible inventory locations of the repository as shown in the following example and **not** in the Ansible plays and roles defaults. See https://docs.ansible.com/ansible/2.9/user_guide/playbooks_best_practices.html#content-organization for directory layouts for inventory.

   WARNING: Never place sensitive information such as passwords in the git repository.

   The following example shows how to add a `vars.yml` file containing site-specific configuration values to the `Application` group variable location.

   These and other Ansible files do not necessarily need to be modified for UAN image creation.

   ```
   ncn-m001# vim group_vars/Application/vars.yml
   ncn-m001# git add group_vars/Application/vars.yml
   ncn-m001# git commit -m "Add vars.yml customizations"
   [integration ecece54] Add vars.yml customizations
    1 file changed, 1 insertion(+)
    create mode 100644 group_vars/Application/vars.yml
   ```

10. Verify that the System Layout Service (SLS) and the `uan_interfaces` configuration role refer to the Mountain Node Management Network by the same name. Skip this step if there are no Mountain cabinets in the HPE Cray EX system.

a. Edit the `roles/uan_interfaces/tasks/main.yml` file and change the line that reads `url: http://cray-sls/v1/search/networks?name=MNMN` to read `url: http://cray-sls/v1/search/networks?name=NMN_MTN`.

The following excerpt of the relevant section of the file shows the result of the change.

```
- name: Get Mountain NMN Services Network info from SLS
  local_action:
    module: uri
      url: http://cray-sls/v1/search/networks?name=NMN_MTN
    method: GET
  register: sls_mnmn_svcs
  ignore_errors: yes
```

b. Stage and commit the network name change

```
ncn-m# git add roles/uan_interfaces/tasks/main.yml
ncn-m# git commit -m "Add Mountain cabinet support"
```

11. Push the changes to the repository using the proper credentials, including the password obtained previously.

```
ncn-m001# git push --set-upstream origin integration
Username for 'https://api-gw-service-nmn.local': crayvcs
 Password for 'https://crayvcs@api-gw-service-nmn.local':
 . . .
 remote: Processed 1 references in total
 To https://api-gw-service-nmn.local/vcs/cray/uan-config-management.git
  * [new branch]      integration -> integration
  Branch 'integration' set up to track remote branch 'integration' from
'origin'.
```

12. Capture the most recent commit for reference in setting up a CFS configuration and navigate to the parent directory.

```
ncn-m001# git rev-parse --verify HEAD

ecece54b1eb65d484444c4a5ca0b244b329f4667

ncn-m001# cd ..
```

The configuration parameters have been stored in a branch in the UAN git repository. The next phase of the process is initiating the Configuration Framework Service (CFS) to customize the image.

CONFIGURE UAN IMAGES

13. Create a JSON input file for generating a CFS configuration for the UAN.

Gather the git repository clone URL, commit, and top-level play for each configuration layer (that is, Cray product). Add them to the CFS configuration for the UAN, if wanted.

For the `commit` value for the UAN layer, use the Git commit value obtained in the previous step.

See the product manuals for further information on configuring other Cray products, as this procedure documents only the configuration of the UAN. More layers can be added to be configured in a single CFS session.

The following configuration example can be used for preboot image customization as well as post-boot node configuration.

```
{
  "layers": [
    {
      "name": "uan-integration-PRODUCT_VERSION",
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/uan-config-
management.git",
      "playbook": "site.yml",
      "commit": "ecece54b1eb65d484444c4a5ca0b244b329f4667"
    }
    # { ... add configuration layers for other products here, if desired ... }
  ]
}
```

14. Add the configuration to CFS using the JSON input file.

In the following example, the JSON file created in the previous step is named `uan-config-PRODUCT_VERSION.json` only the details for the UAN layer are shown.

```
ncn-m001# cray cfs configurations update uan-config-PRODUCT_VERSION \
                  --file ./uan-config-PRODUCT_VERSION.json \
                  --format json
{
  "lastUpdated": "2021-07-28T03:26:00:37Z",
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/uan-config-
management.git",
      "commit": "ecece54b1eb65d484444c4a5ca0b244b329f4667",
      "name": "uan-integration-PRODUCT_VERSION",
      "playbook": "site.yml"
    }  # <-- Additional layers not shown, but would be inserted here
  ],
  "name": "uan-config-PRODUCT_VERSION"
}
```

15. Modify the UAN image to include the 1.4.0 day zero rpms .

a. Untar the 1.4.0 Day Zero Patch tarball if it is not untarred already.

```
ncn-m001# tar -xvf shasta-1.4.0-p2.tar
1.4.0-p2/
1.4.0-p2/csm/
1.4.0-p2/csm/csm-0.8.22-0.9.0.patch.gz
1.4.0-p2/csm/csm-0.8.22-0.9.0.patch.gz.md5sum
1.4.0-p2/uan/
1.4.0-p2/uan/uan-2.0.0-uan-2.0.0.patch.gz
1.4.0-p2/uan/uan-2.0.0-uan-2.0.0.patch.gz.md5sum
1.4.0-p2/rpms/
1.4.0-p2/rpms/cray-dvs-compute-2.12_4.0.102-7.0.1.0_8.1__g30d29e7a.x86_64.rpm
1.4.0-p2/rpms/cray-dvs-devel-2.12_4.0.102-7.0.1.0_8.1__g30d29e7a.x86_64.rpm
1.4.0-p2/rpms/cray-dvs-kmp-
cray_shasta_c-2.12_4.0.102_k4.12.14_197.78_9.1.58-7.0.1.0_8.1__g30d29e7a.x86_
64.rpm
```

```
1.4.0-p2/rpms/cray-network-config-1.1.7-20210318094806_b409053-
sles15sp1.x86_64.rpm
1.4.0-p2/rpms/slingshot-network-config-1.1.7-20210318093253_83fab52-
sles15sp1.x86_64.rpm
1.4.0-p2/rpms/slingshot-network-config-full-1.1.7-20210318093253_83fab52-
sles15sp1.x86_64.rpm
1.4.0-p2/rpms/cray-dvs-
compute-2.12_4.0.102-7.0.1.0_8.1__g30d29e7a.x86_64.rpm.md5sum
1.4.0-p2/rpms/cray-dvs-
devel-2.12_4.0.102-7.0.1.0_8.1__g30d29e7a.x86_64.rpm.md5sum
1.4.0-p2/rpms/cray-dvs-kmp-
cray_shasta_c-2.12_4.0.102_k4.12.14_197.78_9.1.58-7.0.1.0_8.1__g30d29e7a.x86_
64.rpm.md5sum
1.4.0-p2/rpms/cray-network-config-1.1.7-20210318094806_b409053-
sles15sp1.x86_64.rpm.md5sum
1.4.0-p2/rpms/slingshot-network-config-1.1.7-20210318093253_83fab52-
sles15sp1.x86_64.rpm.md5sum
1.4.0-p2/rpms/slingshot-network-config-full-1.1.7-20210318093253_83fab52-
sles15sp1.x86_64.rpm.md5sum
```

b. Download the rootfs image specified in the UAN product catalog.

   Replace *IMAGE_ID* in the following `export` command with the IMS image id recorded in Step 1.

```
ncn-m001# export UAN_IMAGE_ID=IMAGE_ID
ncn-m001# cray artifacts get boot-images ${UAN_IMAGE_ID}/rootfs \
${UAN_IMAGE_ID}.squashfs
ncn-m001# la ${UAN_IMAGE_ID}.squashfs
-rw-r--r-- 1 root root 1.5G Mar 17 19:35 f3ba09d7-
e3c2-4b80-9d86-0ee2c48c2214.squashfs
```

c. Mount the squashfs file and copy its contents to a different directory.

```
ncn-m001# mkdir mnt
ncn-m001# mount -t squashfs ${UAN_IMAGE_ID}.squashfs mnt -o ro,loop
ncn-m001# cp -a mnt UAN-1.4.0-day-zero
ncn-m001# umount mnt
ncn-m001# rmdir mnt
```

d. Copy the new RPMs into the new image directory.

```
ncn-m001# cp 1.4.0-p2/rpms/* UAN-1.4.0-day-zero/
ncn-m001# cd UAN-1.4.0-day-zero/
```

e. Chroot into the new image directory.

```
ncn-m001# chroot . bash
```

f. Update, erase, and install RPMs in the new image directory.

```
chroot-ncn-m001# rpm -Uv cray-dvs-*.rpm
chroot-ncn-m001# rpm -e cray-network-config
chroot-ncn-m001# rpm -e slingshot-network-config-full
chroot-ncn-m001# rpm -e slingshot-network-config
chroot-ncn-m001# rpm -iv slingshot-network-config-
full-1.1.7-20210318093253_83fab52-sles15sp1.x86_64.rpm \
slingshot-network-config-1.1.7-20210318093253_83fab52-sles15sp1.x86_64.rpm \
cray-network-config-1.1.7-20210318094806_b409053-sles15sp1.x86_64.rpm
```

g. Generate a new initrd to match the updated image by running the `/tmp/images.sh` script. Then wait for this script to complete before continuing.

```
chroot-ncn-m001# /tmp/images.sh
```

The output of this script will contain error messages. These error messages can be ignored as long as the message `dracut: *** Creating initramfs image file` appears at the end.

h. Copy the `/boot/initrd` and `/boot/vmlinuz` files out of the chroot environment and into a temporary location on the file system of the node.

i. Exit the chroot environment and delete the packages.

```
chroot-ncn-m001# exit
exit
ncn-m001# rm *.rpm
ncn-m001# cd ..
```

j. Verify that there is only one subdirectory in the `lib/modules` directory of the image.

The existence of more than one subdirectory indicates a mismatch between the kernel of the image and the DVS RPMS that were installed in the previous step.

```
ncn-m001# la UAN-1.4.0-day-zero/lib/modules/
total 8.0K
drwxr-xr-x 3 root root   49 Feb 25 17:50 ./
drwxr-xr-x 8 root root 4.0K Feb 25 17:52 ../
drwxr-xr-x 6 root root 4.0K Mar 17 19:49 4.12.14-197.78_9.1.58-cray_shasta_c/
```

k. Resquash the new image directory.

```
ncn-m001# mksquashfs UAN-1.4.0-day-zero UAN-1.4.0-day-zero.squashfs
Parallel mksquashfs: Using 64 processors
Creating 4.0 filesystem on UAN-1.4.0-day-zero.squashfs, block size 131072.
...
```

l. Create a new IMS image registration and save the `id` field in an environment variable.

```
ncn-m001# cray ims images create --name UAN-1.4.0-day-zero
name = "UAN-1.4.0-day-zero"
created = "2021-03-17T20:23:05.576754+00:00"
id = "ac31e971-f990-4b5f-821d-c0c18daefb6e"
ncn-m001# export NEW_IMAGE_ID=ac31e971-f990-4b5f-821d-c0c18daefb6e
```

m. Upload the new image, initrd, and kernel to S3 using the `id` from the previous step.

```
ncn-m001# cray artifacts create boot-images ${NEW_IMAGE_ID}/rootfs \
UAN-1.4.0-day-zero.squashfs
artifact = "ac31e971-f990-4b5f-821d-c0c18daefb6e/UAN-1.4.0-day-zero.rootfs"
Key = "ac31e971-f990-4b5f-821d-c0c18daefb6e/UAN-1.4.0-day-zero.rootfs"
ncn-m001# cray artifacts create boot-images ${NEW_IMAGE_ID}/initrd \
initrd
artifact = "ac31e971-f990-4b5f-821d-c0c18daefb6e/UAN-1.4.0-day-zero.initrd"
Key = "ac31e971-f990-4b5f-821d-c0c18daefb6e/UAN-1.4.0-day-zero.initrd"
ncn-m001# cray artifacts create boot-images ${NEW_IMAGE_ID}/kernel \
vmlinuz
artifact = "ac31e971-f990-4b5f-821d-c0c18daefb6e/UAN-1.4.0-day-zero.kernel"
Key = "ac31e971-f990-4b5f-821d-c0c18daefb6e/UAN-1.4.0-day-zero.kernel"
```

n. Obtain the md5sum of the squashfs image, initrd, and kernel.

```
ncn-m001# md5sum UAN-1.4.0-day-zero.squashfs initrd vmlinuz
cb6a8934ad3c483e740c648238800e93  UAN-1.4.0-day-zero.squashfs
```

```
3fd8a72a49a409f70140fabe11bdac25  initrd
5edcf3fd42ab1eccfbf1e52008dac5b9  vmlinuz
```

o. Use the image `id` from Step 1 to print out all the IMS details about the current UAN image.

```
ncn-m001# cray ims images describe c880251d-b275-463f-8279-e6033f61578b
created = "2021-03-24T18:00:24.464755+00:00"
id = "c880251d-b275-463f-8279-e6033f61578b"
name = "cray-shasta-uan-cos-sles15sp1.x86_64-0.1.32"[link]
etag = "d4e09fb028d5d99e4a0d4d9b9d930e13"
path = "s3://boot-images/c880251d-b275-463f-8279-e6033f61578b/manifest.json"
type = "s3"
```

p. Use the `path` of the `manifest.json` file to download that JSON to a local file. Omit everything before the image id in the `cray artifacts get boot-images` command, as shown in the following example.

```
ncn-m001# cray artifacts get boot-images \
c880251d-b275-463f-8279-e6033f61578b/manifest.json uan-manifest.json
ncn-m001# cat uan-manifest.json
{
    "artifacts": [
        {
            "link": {
                "etag": "6d04c3a4546888ee740d7149eaecea68",
                "path": "s3://boot-images/c880251d-b275-463f-8279-
e6033f61578b/rootfs",
                "type": "s3"
            },
            "md5": "a159b94238fc5bfe80045889226b33a3",
            "type": "application/vnd.cray.image.rootfs.squashfs"
        },
        {
            "link": {
                "etag": "6d04c3a4546888ee740d7149eaecea68",
                "path": "s3://boot-images/c880251d-b275-463f-8279-
e6033f61578b/kernel",
                "type": "s3"
            },
            "md5": "175f0c1363c9e3a4840b08570a923bc5",
            "type": "application/vnd.cray.image.kernel"
        },
        {
            "link": {
                "etag": "6d04c3a4546888ee740d7149eaecea68",
                "path": "s3://boot-images/c880251d-b275-463f-8279-
e6033f61578b/initrd",
                "type": "s3"
            },
            "md5": "0094629e4da25226c75b113760eeabf7",
            "type": "application/vnd.cray.image.initrd"
        }
    ],
    "created" : "20210317153136",
    "version": "1.0"
}
```

Alternatively, a `mainfest.json` can be created from scratch. In that case, create a new hexadecimal value for the `etag` if the image referred to by the manifest does not already have one. The `etag` field can not be left blank.

q. Replace the `path` and `md5` values of the initrd, kernel, and rootfs with the values obtained in substeps m and n.

r. Update the value for the `"created"` line in the manifest with the output of the following command:

```
ncn-m001# date '+%Y%m%d%H%M%S'
```

s. Verify that the modified JSON file is still valid.

```
ncn-m001# cat manifest.json | jq
```

t. Save the changes to the file.

u. Upload the updated `manifest.json` file.

```
ncn-m001# cray artifacts create boot-images \
${NEW_IMAGE_ID}/manifest.json uan-manifest.json
artifact = "ac31e971-f990-4b5f-821d-c0c18daefb6e/manifest.json"
Key = "ac31e971-f990-4b5f-821d-c0c18daefb6e/manifest.json"
```

v. Update the IMS image to use the new `uan-manifest.json` file.

```
ncn-m001# cray ims images update ${NEW_IMAGE_ID} \
--link-type s3 --link-path s3://boot-images/${NEW_IMAGE_ID}/manifest.json\
--link-etag 6d04c3a4546888ee740d7149eaecea68
created = "2021-03-17T20:23:05.576754+00:00"
id = "ac31e971-f990-4b5f-821d-c0c18daefb6e"
name = "UAN-1.4.0-day-zero"

[link]
etag = "6d04c3a4546888ee740d7149eaecea68"
path = "s3://boot-images/ac31e971-f990-4b5f-821d-c0c18daefb6e/manifest.json"
type = "s3"
```

16. Create a CFS session to perform preboot image customization of the UAN image.

```
ncn-m001# cray cfs sessions create --name uan-config-PRODUCT_VERSION \
                  --configuration-name uan-config-PRODUCT_VERSION \
                  --target-definition image \
                  --target-group Application $NEW_IMAGE_ID \
                  --format json
```

17. Wait until the CFS configuration session for the image customization to complete. Then record the ID of the IMS image created by CFS.

The following command will produce output while the process is running. If the CFS session completes successfully, an IMS image ID will appear in the output.

```
ncn-m001# cray cfs sessions describe uan-config-PRODUCT_VERSION --format json | jq
```

PREPARE UAN BOOT SESSION TEMPLATES

18. Retrieve the xnames of the UAN nodes from the Hardware State Manager (HSM).

These xnames are needed for Step 20.

```
ncn-m001# cray hsm state components list --role Application --subrole UAN --format json | jq -
r .Components[].ID
x3000c0s19b0n0
x3000c0s24b0n0
```

```
x3000c0s20b0n0
x3000c0s22b0n0
```

19. Determine the correct value for the `ifmap` option in the `kernel_parameters` string for the type of UAN.

- Use `ifmap=net0:nmn0,lan0:hsn0,lan1:hsn1` if the UANs are:
  - Either HPE DL325 or DL385 server that have a single OCP PCIe card installed.
  - Gigabyte servers that do not have additional PCIe network cards installed other than the built-in LOM ports.
- Use `ifmap=net2:nmn0,lan0:hsn0,lan1:hsn1` if the UANs are:
  - Either HPE DL325 or DL385 servers which have a second OCP PCIe card installed, regardless if it is being used or not.
  - Gigabyte servers that have a PCIe network card installed in addition to the built-in LOM ports, regardless if it is being used or not.

20. Construct a JSON BOS boot session template for the UAN.

a. Populate the template with the following information:

- The value of the `ifmap` option for the `kernel_parameters` string that was determined in the previous step.
- The xnames of Application nodes from Step 18
- The customized image ID from Step 17 for
- The CFS configuration session name from Step 17

b. Verify that the session template matches the format and structure in the following example:

```
{
   "boot_sets": {
     "uan": {
       "boot_ordinal": 2,
       "kernel_parameters": "console=ttyS0,115200 bad_page=panic crashkernel=360M
hugepagelist=2m-2g intel_iommu=off intel_pstate=disable iommu=pt ip=nmn0:dhcp
numa_interleave_omit=headless numa_zonelist_order=node oops=panic pageblock_order=14
pcie_ports=native printk.synchronous=y quiet rd.neednet=1 rd.retry=10 rd.shell
turbo_boost_limit=999 ifmap=net2:nmn0,lan0:hsn0,lan1:hsn1 spire_join_token=${SPIRE_JOIN_TOKEN}",
       "network": "nmn",
       "node_list": [
         # [ ... List of Application Nodes from cray hsm state command ...]
       ],
       "path": "s3://boot-images/IMS_IMAGE_ID/manifest.json",  # <-- result_id from CFS image
customization session
       "rootfs_provider": "cpss3",
       "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:nmn0",
       "type": "s3"
     }
   },
   "cfs": {
       "configuration": "uan-config-PRODUCT_VERSION"
   },
   "enable_cfs": true,
   "name": "uan-sessiontemplate-PRODUCT_VERSION"
 }
```

c. Save the template with a descriptive name, such as `uan-sessiontemplate-PRODUCT_VERSION.json`.

21. Register the session template with BOS.

The following command uses the JSON session template file to save a session template in BOS. This step allows administrators to boot UANs by referring to the session template name.

```
ncn-m001# cray bos sessiontemplate create \
                  --name uan-sessiontemplate-PRODUCT_VERSION \
                  --file uan-sessiontemplate-PRODUCT_VERSION.json
/sessionTemplate/uan-sessiontemplate-PRODUCT_VERSION
```

Perform *Boot UANs* on page 123 to boot the UANs with the new image and BOS session template.

18 Boot UANs

Prerequisites

Create UAN Boot Images on page 112

About this task

ROLE	System administrator
OBJECTIVE	Boot UANs with an image so that they are ready for user logins.
LIMITATIONS	None
NEW IN THIS RELEASE	This procedure is new in this release.

Procedure

1. Create a BOS session to boot the UAN nodes.

```
ncn-m001# cray bos session create --template-uuid uan-sessiontemplate-PRODUCT_VERSION \
--operation reboot --format json | tee session.json
{
 "links": [
   {
     "href": "/v1/session/89680d0a-3a6b-4569-a1a1-e275b71fce7d",
     "jobId": "boa-89680d0a-3a6b-4569-a1a1-e275b71fce7d",
     "rel": "session",
     "type": "GET"
   },
   {
     "href": "/v1/session/89680d0a-3a6b-4569-a1a1-e275b71fce7d/status",
     "rel": "status",
     "type": "GET"
   }
 ],
 "operation": "reboot",
 "templateUuid": "uan-sessiontemplate-PRODUCT_VERSION"
}
```

   The first attempt to reboot the UANs will most likely fail. The UAN boot may hang and the UAN console will look similar to the following:

```
2021-03-19 01:32:41 dracut-initqueue[420]: DVS: node map generated.
2021-03-19 01:32:41 katlas: init_module: katlas loaded, currently disabled
2021-03-19 01:32:41
2021-03-19 01:32:41 DVS: Revision: kbuild Built: Mar 17 2021 @ 15:14:05 against
LNet 2.12.4
2021-03-19 01:32:41 DVS debugfs: Revision: kbuild Built: Mar 17 2021 @ 15:14:05
against LNet 2.12.4
2021-03-19 01:32:41 dracut-initqueue[420]: DVS: loadDVS: successfully added 10
new nodes into map.
2021-03-19 01:32:41 ed dvsproc module.
2021-03-19 01:32:41 DVS: message size checks complete.
```

```
2021-03-19 01:32:41 dracut-initqueuedvs_thread_generator: Watching pool DVS-
IPC_msg (id 0)
2021-03-19 01:32:41 [420]: DVS: loaded dvs module.
2021-03-19 01:32:41 dracut-initqueue[420]: mount is: /opt/cray/cps-utils/bin/
cpsmount.sh -a api-gw-service-nmn.local -t dvs -T 300 -i nmn0 -e
3116cf653e84d265cf8da94956f34d9e-181 s3://boot-images/763213c7-3d5f-4f2f-9d8a-
ac6086583f43/rootfs /tmp/cps
2021-03-19 01:32:41 dracut-initqueue[420]: 2021/03/19 01:31:01 cpsmount_helper
Version: 1.0.0
2021-03-19 01:32:47 dracut-initqueue[420]: 2021/03/19 01:31:07 Adding content:
s3://boot-images/763213c7-3d5f-4f2f-9d8a-ac6086583f43/rootfs
3116cf653e84d265cf8da94956f34d9e-181 dvs
2021-03-19 01:33:02 dracut-initqueue[420]: 2021/03/19 01:31:22 WARN:
readyForMount=false type=dvs ready=0 total=2
2021-03-19 01:33:18 dracut-initqueue[420]: 2021/03/19 01:31:38 WARN:
readyForMount=false type=dvs ready=0 total=2
2021-03-19 01:33:28 dracut-initqueue[420]: 2021/03/19 01:31:48 2 dvs servers
[10.252.1.7 10.252.1.8]
```

If this occurs, repeat the BOS command.

2. Verify that the Day Zero patch was applied correctly during *Create UAN Boot Images* on page 112. Skip this step if the patch has already been verified.

   a. SSH into a newly-booted UAN.

   ```
   ncn-m001# ssh uan01-nmn
   Last login: Wed Mar 17 19:10:12 2021 from 10.252.1.12
   uan01#
   ```

   b. Verify that the DVS RPM versions match what exists in the `1.4.0-p2/rpms` directory.

   ```
   uan01# rpm -qa | grep 'cray-dvs.*2\.12' | sort
   cray-dvs-compute-2.12_4.0.102-7.0.1.0_8.1__g30d29e7a.x86_64
   cray-dvs-devel-2.12_4.0.102-7.0.1.0_8.1__g30d29e7a.x86_64
   cray-dvs-kmp-
   cray_shasta_c-2.12_4.0.102_k4.12.14_197.78_9.1.58-7.0.1.0_8.1__g30d29e7a.x86_
   64
   ```

   c. Log out of the UAN.

   ```
   uan01# exit
   ```

3. Retrieve the BOS session ID from the output of the previous command.

   ```
   ncn-m001# export BOS_SESSION=$(jq -r '.links[] | select(.rel=="session") |\
    .href' session.json | cut -d '/' -f4)
   ncn-m001# echo $BOS_SESSION
   89680d0a-3a6b-4569-a1a1-e275b71fce7d
   ```

4. Retrieve the Boot Orchestration Agent (BOA) Kubernetes job name for the BOS session.

   ```
   ncn-m001# BOA_JOB_NAME=$(cray bos session describe $BOS_SESSION --format json | jq -r .boa_job_name)
   ```

5. Retrieve the Kuberenetes pod name for the BOA assigned to run this session.

   ```
   ncn-m001# BOA_POD=$(kubectl get pods -n services -l job-name=$BOA_JOB_NAME \
   --no-headers -o custom-columns=":metadata.name")
   ```

6. View the logs for the BOA to track session progress.

```
ncn-m001# kubectl logs -f -n services $BOA_POD -c boa
```

7. List the CFS sessions started by the BOA. Skip this step if CFS was not enabled in the boot session template used to boot the UANs.

   If CFS was enabled in the boot session template, the BOA will initiate a CFS session.

   In the following command, `pending` and `complete` are also valid statuses to filter on.

   ```
   ncn-m001# cray cfs sessions list --tags bos_session=$BOS_SESSION --status
   running --format json
   ```

19 Install Slurm

Prerequisites

- Loftsman
- Helm
- CSM
- COS
- UAN

About this task

ROLE	• System installer/system administrator
OBJECTIVE	The following procedure details how to install Slurm on a 1.4 system.
NEW IN THIS RELEASE	This entire procedure is new with this release.

Procedure

1. Start a typescript to capture the commands and output from this installation.

```
ncn-m001# script -af product-slurm.$(date +%Y-%m-%d).txt
ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Copy Slurm release tarball onto the system.

3. Run installation script.

```
ncn-m001# tar -xf wlm-slurm-0.1.0.tar.gz
ncn-m001cd wlm-slurm-0.1.0
ncn-m001./install.sh
```

4. Customize Slurm ansible content.

```
ncn-m001# git clone https://api-gw-service-nmn.local/vcs/cray/slurm-config-
management.git
ncn-m001# cd slurm-config-management
ncn-m001# git merge origin/cray/slurm/0.1.0
```

   If desired, make additional changes and commits.

   a. Get crayvcs password for pushing.

```
ncn-m001# kubectl get secret -n services vcs-user-credentials --
template={{.data.vcs_password}} | base64 --decode
```

b. Push changes to VCS (username crayvcs).

```
ncn-m001# git push origin master
```

c. Obtain the commit hash to use for the CFS configurations.

```
ncn-m001# git rev-parse --verify HEAD
```

5. Configure COS ansible content.

```
ncn-m001# git clone https://api-gw-service-nmn.local/vcs/cray/cos-config-
management.git
ncn-m001# cd cos-config-management
ncn-m001# git checkout integration
```

Create the file group_vars/Compute/keycloak.yaml with the following content:

```
keycloak_config_computes: True
```

Get craycvs for pushing.

```
ncn-m001# kubectl get secret -n services vcs-user-credentials \
--template={{.data.vcs_password}} | base64 --decode
```

Push changes to VCS (username crayvcs).

```
ncn-m001# git add group_vars/Compute/keycloak.yaml
ncn-m001# git commit -m "Configure keycloak for SLURM"
ncn-m001# git push origin integration
```

Get the commit hash to use for the CFS configurations.

```
ncn-m001# git rev-parse --verify HEAD
```

6. Update COS CFS configuration.

The commit value for the COS layer needs to updated to the value obtained in step 5.

```
ncn-m001# cray cfs configurations describe cos-config-1.4.0 --format=json \
&>cos-config-x.x.x.json
```

Edit cos-config-x.x.x.json, adding a layer for Slurm.

```
ncn-m001# vi cos-config-x.x.x.json
ncn-m001# cat cos-config-x.x.x.json
{
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-
management.git",
      "commit": "5fb065ffc32414f02543654211486216f058986b",
      "name": "cos-integration-1.4.0",
      "playbook": "site.yml"
    },
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/slurm-config-
management.git",
```

```
      "commit": "b0cb51391940bb71844bb32cdcda7d89c2a1e80a",
      "name": "slurm-integration-0.1.0",
      "playbook": "site.yml"
    }
  ]
}
ncn-m001# cray cfs configurations update cos-config-1.4.0 \
--file cos-config-x.x.x.json
```

7. Create a new compute image.

```
ncn-m001# cray cfs sessions create \
    --name slurm-cos-0 \
    --configuration-name cos-config-1.4.0 \
    --target-definition image \
    --target-group Compute <IMAGE ID>
```

Obtain the customized image ID for the next step.

```
ncn-m001# IMS_ID=$(cray cfs sessions describe slurm-cos-0 --format json | jq -
r .status.artifacts[].result_id)
```

Obtain the etag and path for the BOS template.

```
ncn-m001# cray ims images describe $IMS_ID
```

8. Update COS BOS template with the new compute node image.

```
ncn-m001# cray bos v1 sessiontemplate describe cos-sessiontemplate-1.4.0 \
--format=json &>cos-sessiontemplate-1.4.0.json
```

Edit /upload/cos-sessiontemplate-1.4.0.json, replacing etag and path.

```
ncn-m001# cray bos v1 sessiontemplate create --name cos-sessiontemplate-1.4.0 \
--file cos-sessiontemplate-1.4.0.json
```

9. Verify cray-conman is connected to compute and UAN nodes.

The cray-conman service determines which nodes it should monitor by checking with the Hardware State Manager (HSM) service. It does this *once* when it starts. If HSM has not discovered some nodes when conman starts, then HSM is unaware of them and so is conman. Therefore, it is important to verify that conman is monitoring all nodes' console logs. If it is not, re-initialize the cray-conman service and recheck the nodes it is monitoring. `conman -q` can be used to list the existing connections.

Use kubectl to exec into the running cray-conman pod, then check the existing connections.

```
cray-conman-b69748645-qtfxj:/ # conman -q
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

If the compute nodes and UANs are not included in the list of nodes being monitored, the conman process can be re-initialized by killing the conmand process.

```
cray-conman-b69748645-qtfxj:/ # ps -ax | grep conmand
  13 ?        Sl     0:45 conmand -F -v -c /etc/conman.conf
  56704 pts/3    S+     0:00 grep conmand
cray-conman-b69748645-qtfxj:/ # kill 13
```

This will regenerate the conman configuration file and restart the conmand process, and now include all nodes that are included in the state manager.

```
cray-conman-b69748645-qtfxj:/ #
x9000c1s7b0n1
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

10. Reboot compute nodes with updated session template.

```
ncn-m001# cray bos v1 session create --template-uuid cos-sessiontemplate-1.4.0
\
--operation reboot
```

11. Update UAN CFS configuration.

```
ncn-m001# cray cfs configurations describe uan-config-2.0.0 --format=json &>uan-
config-2.0.0.json
```

Edit /upload/uan-config-2.0.0.json, adding a layer for Slurm.

```
{
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/uan-config-
management.git",
      "commit": "aa5ce7d5975950ec02493d59efb89f6fc69d67f1",
      "name": "uan-integration-2.0.0",
      "playbook": "site.yml"
    },
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/slurm-config-
management.git",
      "commit": "b0cb51391940bb71844bb32cdcda7d89c2a1e80a",
      "name": "slurm-integration-0.1.0",
      "playbook": "site.yml"
    }
  ]
}
ncn-m001# cray cfs configurations update uan-config-2.0.0 \
--file uan-config-2.0.0.json
```

The name and lastUpdated fields should be removed from the cos.config-x.x.x.json file since it generates an error when left in.

```
Error: Bad Request: Property is read-only - 'name'
```

12. Create a new UAN image.

```
ncn-m001# cray cfs sessions create \
    --name slurm-uan-0 \
    --configuration-name uan-config-2.0.0 \
    --target-definition image \
    --target-group Application <IMAGE ID>
```

Get the customized image ID for the next step.

```
ncn-m001# cray cfs sessions describe slurm-uan-0 --format json | jq -
r .status.artifacts[].result_id
```

13. Update UAN BOS template.

```
ncn-m001# cray bos v1 sessiontemplate describe uan-sessiontemplate-2.0.0 --
format=json &> uan-sessiontemplate-2.0.0.json
```

Edit /upload/uan-sessiontemplate-2.0.0.json, replacing etag and path.

```
ncn-m001# cray bos v1 sessiontemplate create --name uan-sessiontemplate-2.0.0 --
file /upload/uan-sessiontemplate-2.0.0.json
```

14. Reboot UANs with updated session template.

```
ncn-m001# cray bos v1 session create --template-uuid uan-sessiontemplate-2.0.0
--operation reboot
```

15. Validate installation.

a. Validate Kubernetes deployments.

Check for a running pod in each of the following deployments.

```
ncn-m001# kubectl get deployment -n user slurmdb
ncn-m001# kubectl get deployment -n user slurmdbd
ncn-m001# kubectl get deployment -n user slurmctld
```

b. Validate compute node content.

A private ssh key for root can be obtained with "kubectl get secrets -n services csm-private-key -o jsonpath="{.data.value}" | base64 -d

Check that `slurmd` is running.

```
ncn-m001# ssh nid000001
```

```
nid000001# systemctl status slurmd
```

```
nid00001# exit
```

c. Validate UAN content.

Check compute node Slurm state.

```
ncn-m001# ssh uan01
```

```
uan01# sinfo
```

Run *`hostname`* on every available compute node.

```
uan01# srun -N <num computes> hostname
```

```
uan01# exit
```

16. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

20 Install PBS

Prerequisites

- Loftsman
- Helm
- CSM
- COS
- UAN
- User home directories must be mounted on compute nodes

About this task

ROLE	• System installer/system administrator
OBJECTIVE	The following procedure details how to install PBS on a 1.4 system.
NEW IN THIS RELEASE	This entire procedure is new with this release.

Procedure

1. Start a typescript to capture the commands and output from this installation.

```
ncn-m001# script -af product-pbs.$(date +%Y-%m-%d).txt
ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Configure PBS Helm customizations in the system's customizations.yaml file.

   a. Set `spec.network.static_ips.pbs` to match `dig +short pbs-service-nmn.local`.

   b. Set `spec.network.static_ips.pbs_comm` to match `dig +short pbs-comm-service-nmn.local`.

3. Copy PBS release tarball onto the system.

4. Run installation script.

```
ncn-m001# tar -xf wlm-pbs-0.1.0.tar.gz
ncn-m001cd wlm-pbs-0.1.0
ncn-m001./install.sh
```

5. Configure PBS with `qmgr`.

```
ncn-m001# PBS_POD=$(kubectl get pod -n user -l app=pbs -o \
jsonpath='{.items[0].metadata.name}')

ncn-m001# kubectl exec -it -n user $PBS_POD -- qmgr
set server flatuid=true
set server acl_roots+=root@*
set pbshook PBS_cray_atom enabled=t
set server pbs_license_info=6200@cflls05.us.cray.com:6200@cflls06.us.cray.com:
6200@cflls07.us.cray.com

ncn-m001# quit
```

6. Customize PBS ansible content.

```
ncn-m001# git clone https://api-gw-service-nmn.local/vcs/cray/pbs-config-
management.git
ncn-m001# cd pbs-config-management
ncn-m001# git merge origin/cray/pbs/0.1.0
```

If desired, make additional changes and commits.

a. Get crayvcs password for pushing.

```
ncn-m001# kubectl get secret -n services vcs-user-credentials \
--template={{.data.vcs_password}} | base64 --decode
```

b. Push changes to VCS (username crayvcs).

```
ncn-m001# git push origin master
```

c. Obtain the commit hash to use for the CFS configurations.

```
ncn-m001# GIT REV-PARSE --VERIFY HEAD
```

7. Configure COS ansible content.

```
ncn-m001# git clone https://api-gw-service-nmn.local/vcs/cray/cos-config-
management.git
ncn-m001# cd cos-config-management
ncn-m001# git checkout integration
```

Create the file group_vars/Compute/keycloak.yaml with the following content:

```
keycloak_config_computes: True
```

Get craycvs for pushing.

```
ncn-m001# kubectl get secret -n services vcs-user-credentials \
--template={{.data.vcs_password}} | base64 --decode
```

Push changes to VCS (username crayvcs).

```
ncn-m001# git add group_vars/Compute/keycloak.yaml
ncn-m001# git commit -m "Configure keycloak for PBS"
ncn-m001# git push origin integration
```

Get the commit hash to use for the CFS configurations.

```
ncn-m001# git rev-parse --verify HEAD
```

8. Update COS CFS configuration.

```
ncn-m001# cray cfs configurations describe cos-config-1.4.0 --format=json \
&>cos-config-1.4.0.json
```

Edit cos-config-1.4.0.json, adding a layer for PBS.

```
ncn-m001# vi cos-config-1.4.0.json
ncn-m001# cat cos-config-1.4.0.json
{
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-
management.git",
      "commit": "5fb065ffc32414f025436542114862l6f058986b",
      "name": "cos-integration-1.4.0",
      "playbook": "site.yml"
    },
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/pbs-config-
management.git",
      "commit": "b0cb51391940bb71844bb32cdcda7d89c2a1e80a",
      "name": "pbs-integration-0.1.0",
      "playbook": "site.yml"
    }
  ]
}
ncn-m001# cray cfs configurations update cos-config-1.4.0 \
--file cos-config-1.4.0.json
```

9. Create a new compute image.

```
ncn-m001# cray cfs sessions create \
    --name pbd-cos-0 \
    --configuration-name cos-config-1.4.0 \
    --target-definition image \
    --target-group Compute <IMAGE ID>
```

Get the customized image ID for the next step.

```
ncn-m001# cray cfs sessions describe pbs-cos-0 --format json | jq -
r .status.artifacts[].result_id
```

10. Update COS BOS template with the new compute node image.

```
ncn-m001# cray bos v1 sessiontemplate describe cos-sessiontemplate-1.4.0 \
--format=json &>cos-sessiontemplate-1.4.0.json
```

Edit /upload/cos-sessiontemplate-1.4.0.json, replacing etag and path.

```
ncn-m001# cray bos v1 sessiontemplate create --name cos-sessiontemplate-1.4.0 \
--file cos-sessiontemplate-1.4.0.json
```

11. Verify cray-conman is connected to compute and UAN nodes.

The cray-conman service determines which nodes it should monitor by checking with the Hardware State Manager (HSM) service. It does this *once* when it starts. If HSM has not discovered some nodes when conman starts, then HSM is unaware of them and so is conman. Therefore, it is important to verify that

conman is monitoring all nodes' console logs. If it is not, re-initialize the cray-conman service and recheck the nodes it is monitoring. `conman -q` can be used to list the existing connections.

Use kubectl to exec into the running cray-conman pod, then check the existing connections.

```
cray-conman-b69748645-qtfxj:/ # conman -q
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

If the compute nodes and UANs are not included in the list of nodes being monitored, the conman process can be re-initialized by killing the conmand process.

```
cray-conman-b69748645-qtfxj:/ # ps -ax | grep conmand
  13 ?        Sl     0:45 conmand -F -v -c /etc/conman.conf
  56704 pts/3    S+     0:00 grep conmand
cray-conman-b69748645-qtfxj:/ # kill 13
```

This will regenerate the conman configuration file and restart the conmand process, and now include all nodes that are included in the state manager.

```
cray-conman-b69748645-qtfxj:/ #
x9000c1s7b0n1
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

12. Reboot compute nodes with updated session template.

```
# cray bos v1 session create --template-uuid cos-sessiontemplate-1.4.0 --
operation reboot
```

13. Update UAN CFS configuration.

```
ncn-m001# cray cfs configurations describe uan-config-2.0.0 --format=json &>uan-
config-2.0.0.json
```

Edit /upload/uan-config-2.0.0.json, adding a layer for PBS.

```
{
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/uan-config-
management.git",
      "commit": "aa5ce7d5975950ec02493d59efb89f6fc69d67f1",
      "name": "uan-integration-2.0.0",
      "playbook": "site.yml"
    },
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/pbs-config-
management.git",
```

```
      "commit": "b0cb51391940bb71844bb32cdcda7d89c2a1e80a",
      "name": "pbs-integration-0.1.0",
      "playbook": "site.yml"
    }
  ]
}
ncn-m001# cray cfs configurations update uan-config-2.0.0 \
--file uan-config-2.0.0.json
```

14. Create a new UAN image.

```
ncn-m001# cray cfs sessions create \
    --name pbs-uan-0 \
    --configuration-name uan-config-2.0.0 \
    --target-definition image \
    --target-group Application <IMAGE ID>
```

Get the customized image ID for the next step.

```
ncn-m001# cray cfs sessions describe pbs-uan-0 --format json | jq -
r .status.artifacts[].result_id
```

15. Update UAN BOS template.

```
ncn-m001# cray bos v1 sessiontemplate describe uan-sessiontemplate-2.0.0 --
format=json &>/upload/uan-sessiontemplate-2.0.0.json
```

Edit /upload/uan-sessiontemplate-2.0.0.json, replacing etag and path.

```
ncn-m001# cray bos v1 sessiontemplate create --name uan-sessiontemplate-2.0.0 --
file /upload/uan-sessiontemplate-2.0.0.json
```

16. Reboot UANs with updated session template.

```
ncn-m001# cray bos v1 session create --template-uuid uan-sessiontemplate-2.0.0
--operation reboot
```

17. Validate installation.

a. Validate Kubernetes deployments.

   Check for a running pod in the following deployments.

   ```
   ncn-m001# kubectl get deployment -n user pbs
   ncn-m001# kubectl get deployment -n user pbs-comm
   ```

b. Validate compute node content.

   Check that `pbs` is running.

   ```
   ncn-m001# ssh nid000001
   ```

   ```
   nid000001# systemctl status pbs
   ```

   ```
   nid00001# exit
   ```

c. Validate UAN content.

   Check compute node PBS state.

```
ncn-m001# ssh uan01
```

```
uan01# qtat -fb
uan01# pbsnodes -a
uan01# qsub -I
uan01# exit
```

18. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

21 Install and Configure LDMS on the Compute Node Image

Prerequisites

None

About this task

ROLE	• System Administrator
OBJECTIVE	Install and configure LDMS on the compute node image.
LIMITATIONS	None

Procedure

1. Obtain the password for the **crayvcs** user for the VCS repository on the HPE Cray EX system.

```
ncn-m001# kubectl get secret -n services vcs-user-credentials --template={{.data.vcs_password}} \
| base64 --decode
<PASSWORD>
```

2. List the remote branches of the **sma-config-management** VCS repository and note the git commit ID for the 1.4.3 branch.

```
ncn-m001# git ls-remote https://api-gw-service-nmn.local/vcs/cray/sma-config-management.git \
refs/heads/cray/sma/*
Username for 'https://api-gw-service-nmn.local': crayvcs
Password for 'https://crayvcs@api-gw-service-nmn.local': <PASSWORD>
eb724e7135dc60d3fdfff9fb01672538f241e588        refs/heads/cray/sma/1.4.2
eb724e7135dc60d3fdfff9fb01672538f241e588        refs/heads/cray/sma/1.4.3
```

3. Determine which compute node image to customize and note the image ID, as shown in the following example.

```
ncn-m001# cray ims images list
.
.
.
[results.link]
etag = "187e27fb0ef8a526b461c363ec6c093d"
path = "s3://boot-images/18e9de59-901c-4afa-8179-314c185401f4/manifest.json"
type = "s3"
[[results]]
created = "2020-11-19T00:35:50.468926+00:00"
id = "51c9e0af-21a8-4bd1-b446-6d38a6914f2d"
name = "cray-shasta-compute-sles15sp1.x86_64-1.4.23"
.
```

```
.
.
```

4. Update the COS CFS configuration that was previously updated in the *Install Slurm* on page 126 and *Install PBS* on page 132 sections. In the example below, the configuration is "cos-config-1.4.0".

   Save the appropriate COS CFS configuration file.

```
ncn-m001# cray cfs configurations describe cos-config-1.4.0 --format=json \
&> cos-config-1.4.0-original.json
ncn-m001# cat cos-config-1.4.0-original.json
{
  "lastUpdated": "2021-03-24T14:03:48Z",
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-
management.git",
      "commit": "ed43261ede502fb246ef143f2070a3e8d05ed5d5",
      "name": "cos-integration-1.4.0",
      "playbook": "site.yml"
    },
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cpe-config-
management.git",
      "commit": "eff74eecba24c3db6a8ba4580e09717265f6a374",
      "name": "cpe-integration-21.3.x",
      "playbook": "pe_deploy.yml"
    },
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/slurm-config-
management.git",
      "commit": "ca8e02cd37d788dd7ff08f36c69dc74f72a7f43d",
      "name": "slurm-integration-0.1.0",
      "playbook": "site.yml"
    }
  ],
  "name": "cos-config-1.4.0"
}
```

   Edit the CFS configuration file by:

   - Deleting the "lastUpdated" and "name" lines
   - Adding a layer for SMA

```
ncn-m001# cp cos-config-1.4.0-original.json cos-config-1.4.0-sma.json
ncn-m001# vi cos-config-1.4.0-sma.json
```

   For reference, below is the difference between the original configuration (cos-config-1.4.0-original.json) and the configuration with SMA added (cos-config-1.4.0-sma.json):

```
ncn-m001# diff cos-config-1.4.0-original.json cos-config-1.4.0-sma.json
2d1
<   "lastUpdated": "2021-03-24T14:03:48Z",
20a20,25
>     },
>     {
>       "name": "sma-ldms-compute",
>       "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/sma-config-
management.git",
>       "playbook": "sma-ldms-compute.yml",
```

```
>       "commit": "eb724e7135dc60d3fdfff9fb01672538f241e588"
22,23c27
<   ],
<   "name": "cos-config-1.4.0"
---
>   ]
```

For reference, below is an example of the complete file with SMA added.

```
ncn-m001# cat cos-config-1.4.0-sma.json
{
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-
management.git",
      "commit": "ed43261ede502fb246ef143f2070a3e8d05ed5d5",
      "name": "cos-integration-1.4.0",
      "playbook": "site.yml"
    },
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cpe-config-
management.git",
      "commit": "eff74eecba24c3db6a8ba4580e09717265f6a374",
      "name": "cpe-integration-21.3.x",
      "playbook": "pe_deploy.yml"
    },
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/slurm-config-
management.git",
      "commit": "ca8e02cd37d788dd7ff08f36c69dc74f72a7f43d",
      "name": "slurm-integration-0.1.0",
      "playbook": "site.yml"
    },
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/sma-config-
management.git",
      "commit": "eb724e7135dc60d3fdfff9fb01672538f241e588",
      "name": "sma-ldms-compute",
      "playbook": "sma-ldms-compute.yml"
    }
  ]
}
```

5. Create a new compute image.

```
ncn-m001# cray cfs sessions create \
    --name sma-cos-0 \
    --configuration-name cos-config-1.4.0 \
    --target-definition image \
    --target-group Compute <IMAGE ID>
```

Obtain the customized image ID for the following step.

```
ncn-m001# IMS_ID=$(cray cfs sessions describe sma-cos-0 --format json | jq -r \
.status.artifacts[].result_id)
```

Obtain the etag and path for the BOS template.

```
ncn-m001# cray ims images describe $IMS_ID
```

6. Update the COS BOS template previously used in the "Install Slurm" and "Install PBS" sections with the new compute node image.

```
ncn-m001# cray bos v1 sessiontemplate describe cos-sessiontemplate-1.4.0 \
--format=json &>cos-sessiontemplate-1.4.0.json
```

Edit `cos-sessiontemplate-1.4.0.json`, replacing etag and path.

```
ncn-m001# cray bos v1 sessiontemplate create --name cos-sessiontemplate-1.4.0 \
--file cos-sessiontemplate-1.4.0.json
```

7. Reboot the compute nodes with the updated session template.

```
ncn-m001# cray bos v1 session create --template-uuid cos-sessiontemplate-1.4.0 \
--operation reboot
```

22 Install the Analytics Product Stream

Prerequisites

- Cray System Management (CSM) is installed and verified.
- UAN
- WLM
- PE
- gitea, cray-product-catalog, cray-cfs-api, cray-cfs-operator, and cray-ims are running.

About this task

ROLE	• System administrator
OBJECTIVE	Install the Analytics product stream which now has its own independent release distribution and installer. TIP: Ensure adequate disk space where tarballs are located. Remove the ...**gz** files after their extraction.
LIMITATIONS	None
NEW IN THIS RELEASE	This entire procedure is new with this release.

Procedure

1. Start a typescript to capture the commands and output from this installation.

   ```
   ncn-m001# script -af product-analytics.$(date +%Y-%m-%d).txt
   ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
   ```

2. Copy the release distribution gzipped tar file, e.g. `analytics-1.0.10.tar.gz`, to ncn-m001.

3. Extract the Analytics release distribution.

   ```
   ncn-m001# tar -zxvf analytics-1.0.10.tar.gz
   ```

4. Change directories to the extracted directory.

   ```
   ncn-m001# cd analytics-1.0.10
   ```

5. Verify the prerequisites.

   Cray System Management (CSM) has been installed and verified including the following Helm Charts.

   - gitea

- cray-product-catalog
- cray-cfs-api
- cray-cfs-operator
- cray-ims
- cray-uan
- WLM (slurm or pbs)

Verify the installation of these components with the following.

```
ncn-m001# helm ls -n services | grep -E \
'gitea|cray-cfs-operator|cray-cfs-api|cray-ims|cray-product-\catalog|uan|slurm|pbs'
```

Cray PE

- If Cray PE will be installed on the system, ensure Cray PE is installed and configured before installing Analytics.

6. Run the installer, **`install.sh`**.

```
ncn-m001# ./install.sh
```

7. Verify the Analytics import pods have run to completion (approximately 5 to 10 minutes).

```
ncn-m001# kubectl get pods -A -n services |grep analytics.*import
services analytics-config-import-1.0.0-8jx5v 0/3 Completed
services analytics-image-recipe-import-1.0.0-xvpcq 0/3 Completed
```

8. Verify that Analytics configuration and images have been imported and added to the **cray-product-catalog ConfigMap**.

   Review the output for the '1.0.0' key and confirm that the Analytics configuration and pre-built image entries exist.

```
ncn-m001# kubectl get cm cray-product-catalog -n services -o json | jq -r .data.analytics
1.0.0:
configuration:
clone_url: https://vcs.groot.dev.cray.com/vcs/cray/analytics-config-management.git
commit: 004e09395dc9dd05b80043949cbed6d17a2c3287
import_branch: cray/analytics/1.0.0
import_date: 2020-12-01 13:19:58.423026
ssh_url: git@vcs.groot.dev.cray.com:cray/analytics-config-management.git
images:
Cray-Analytics.x86_64-base:
id: 9e8f16be-632c-4c51-bd97-6ea3ca8bdd38
recipes: {}
```

9. Create a branch from the imported branch from the installation to customize the Analytics image.

   This imported branch will be reported in the **`cray-product-catalog`** and can be used as a base branch.

   > **NOTE:** The imported branch from the installation cannot be modified. In this guide integration will be used to add customizations to the configuration. Replace the **`clone_url`** hostname with **`api-gwservice-nmn.local`**.

```
ncn-m001# git clone https://api-gw-service-nmn.local/vcs/cray/analytics-config-management.git
Cloning into 'analytics-config-management'...
remote: Enumerating objects: 38, done.
remote: Counting objects: 100% (38/38), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 38 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (38/38), 8.85 KiB | 4.43 MiB/s, done.
```

```
ncn-m001# cd analytics-config-management
ncn-m001# git checkout cray/analytics/1.0.0
Branch 'cray/analytics/1.0.0' set up to track remote branch 'cray/analytics/1.0.0' from 'origin'.
Switched to a new branch 'cray/analytics/1.0.0'
Already up to date.
ncn-m001# git pull
ncn-m001# git checkout -b integration
Switched to a new branch 'integration'
ncn-m001# git merge cray/analytics/1.0.0
Already up to date.
```

10. Update the IMS ID of the image found in the above step.

```
ncn-m001# sed -i 's/analytics_ims_id:.*/analytics_ims_id: \
"9e8f16be-632c-4c51-bd97-6ea3ca8bdd38"/' \
roles/analyticscommon/defaults/main.yml
```

- Always use CPS.

```
ncn-m001# sed -i 's/.*root=craycps.*/#/' roles/analyticsdeploy/tasks/main.yml
```

- Do not rename the `analytics_img` filename on disk.

```
ncn-m001# sed -i 's/command: mv/command: echo/' roles/analyticsdeploy/tasks/main.yml
```

- Set the correct `analytics_img` filename and use a comma for **`sed`** separator.

```
ncn-m001# sed -i 's,analytics_img:.*,analytics_img: \
"{{ analytics_img }}/rootfs",' roles/analyticsdeploy/tasks/main.yml
```

11. Commit the updated IMS change to the repository.

```
ncn-m001# git commit -a -m 'Update IMS id'
```

12. Push the changes to the repository using the proper credentials. For the default **`crayvcs`** user, the credentials are obtained via a Kubernetes secret. Capture the most recent commit for later use and navigate to the parent directory when done.

```
ncn-m001# kubectl get secret -n services vcs-user-credentials --template={{.data.vcs_password}} |
base64 --decode
<== password output ==>
ncn-m001# git push --set-upstream origin integration
Username for 'https://api-gw-service-nmn.local': crayvcs
Password for 'https://crayvcs@api-gw-service-nmn.local': <== enter password from previous command ==>
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 128 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 417 bytes | 417.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a new pull request for 'integration':
remote: https://vcs.groot.dev.cray.com/vcs/cray/analytics-config-management/compare/
master...integration
remote:
remote: . Processing 1 references
remote: Processed 1 references in total
To https://api-gw-service-nmn.local/vcs/cray/analytics-config-management.git
* [new branch] integration -> integration
Branch 'integration' set up to track remote branch 'integration' from 'origin'.

ncn-m001# git rev-parse --verify HEAD
ecece54b1eb65d484444c4a5ca0b244b329f4667

ncn-m001# cd ../
```

13. Create a CFS session configuration for Analytics deployment by creating a JSON file that can be input to the CFS configurations CLI command.

This configuration is used for post-boot node personalization. Note the cloneUrl, commit, and top-level play that should be run for all configuration layers.

```
ncn-m001# vi analytics-config-1.0.0.json
ncn-m001# cat analytics-config-1.0.0.json
{
  "layers": [
    {
      "name": "analytics-integration-1.0.0",
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/analytics-config-management.git",
      "playbook": "site.yml",
      "commit": "ecece54b1eb65d484444c4a5ca0b244b329f4667"
    }
  ]
}


ncn-m001# cray cfs configurations update analytics-config-1.0.0 \
--file ./analytics-config-1.0.0.json --format json
{
  "lastUpdated": "2021-01-05T17:05:58Z",
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/analytics-config-management.git",
      "commit": "ecece54b1eb65d484444c4a5ca0b244b329f4667",
      "name": "analytics-integration-1.0.0",
      "playbook": "site.yml"
    }
  ],
  "name": "analytics-config-1.0.0"
}
```

14. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

23 Procedures that Support Installation

23.1 Configure BGP Neighbors on Management Switches

Prerequisites

This procedure requires administrative privileges.

About this task

ROLE	System administrator
OBJECTIVE	Manually configure and verify Border Gateway Protocol (BGP) neighbors on the management switches. This procedure applies to both Mellanox and Aruba switches.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Log into the spine switches.

   ```
   ncn-m001# ssh admin@SWITCH_IP
   ```

2. Verify the BGP neighbors on the spine switches.

   The BGP neighbors will be the NCN worker node IP addresses on the Node Management Network (NMN). If the system is using Aruba, one of the neighbors will be the other spine switch. The following command should be run for Aruba switches:

   ```
   sw-spine-001# show bgp ipv4 unicast summary
   VRF : default
   BGP Summary
   -----------
    Local AS               : 65533        BGP Router Identifier  : 10.252.0.1
    Peers                  : 4            Log Neighbor Changes   : No
    Cfg. Hold Time         : 180          Cfg. Keep Alive        : 60

    Neighbor        Remote-AS MsgRcvd MsgSent   Up/Down Time State        AdminStatus
    10.252.0.3      65533       31457   31474   00m:02w:04d  Established   Up
    10.252.2.8      65533       54730   62906   00m:02w:04d  Established   Up
    10.252.2.9      65533       54732   62927   00m:02w:04d  Established   Up
    10.252.2.18     65533       54732   62911   00m:02w:04d  Established   Up
   ```

   Run the following command for Mellanox switches:

```
sw-spine-001 [standalone: master] # show ip bgp summary
VRF name                  : default
BGP router identifier     : 10.252.0.1
local AS number           : 65533
BGP table version         : 308
Main routing table version: 308
IPV4 Prefixes             : 261
IPV6 Prefixes             : 0
L2VPN EVPN Prefixes       : 0

------------------------------------------------------------------------------------------------------------
-------------
Neighbor          V    AS           MsgRcvd   MsgSent   TblVer    InQ    OutQ   Up/Down       State/
PfxRcd
------------------------------------------------------------------------------------------------------------
-------------
10.252.0.7        4    65533        37421     42948     308       0      0      12:23:16:07
ESTABLISHED/53
10.252.0.8        4    65533        37421     42920     308       0      0      12:23:16:07
ESTABLISHED/51
10.252.0.9        4    65533        37420     42962     308       0      0      12:23:16:07
ESTABLISHED/51
```

3. Correct any IP address issues discovered in the previous step.

If the BGP neighbors are not in the `ESTABLISHED` state, make sure the IP addresses are correct for the route-map and BGP configuration. If the IPs are incorrect, update the configuration to match the IPs. The configuration below will need to be edited.

a. Retrieve the NCN IPs located in */etc/dnsmasq.d* to correct any incorrect IPs.

```
ncn-m001# grep w00 /etc/dnsmasq.d/statics.conf | grep nmn
host-record=ncn-w003,ncn-w003.nmn,10.252.1.13
host-record=ncn-w002,ncn-w002.nmn,10.252.1.14
host-record=ncn-w001,ncn-w001.nmn,10.252.1.15
```

b. Retrieve the Hardware Management Network (HMN) and Customer Access Network (CAN) IPs to correct the route-map configuration.

```
ncn-m001# grep ncn-w /etc/dnsmasq.d/statics.conf | \
egrep "NMN|HMN|CAN" | grep -v mgmt
dhcp-host=50:6b:4b:08:d0:4a,10.252.1.13,ncn-w003,20m # NMN
dhcp-host=50:6b:4b:08:d0:4a,10.254.1.20,ncn-w003,20m # HMN
dhcp-host=50:6b:4b:08:d0:4a,10.102.4.12,ncn-w003,20m # CAN
dhcp-host=98:03:9b:0f:39:4a,10.252.1.14,ncn-w002,20m # NMN
dhcp-host=98:03:9b:0f:39:4a,10.254.1.22,ncn-w002,20m # HMN
dhcp-host=98:03:9b:0f:39:4a,10.102.4.13,ncn-w002,20m # CAN
dhcp-host=98:03:9b:bb:a9:94,10.252.1.15,ncn-w001,20m # NMN
dhcp-host=98:03:9b:bb:a9:94,10.254.1.24,ncn-w001,20m # HMN
dhcp-host=98:03:9b:bb:a9:94,10.102.4.14,ncn-w001,20m # CAN
```

4. Delete the previous route-map and BGP configuration on Mellanox and Aruba switches.

The Aruba configuration requires the other peering switch to be set as a BGP neighbor. The Mellanox configuration does not require this. Delete the previous route-map and BGP configuration on both switches.

a. Delete the previous Aruba configuration.

```
spine01# conf t

spine01(config)# no router  bgp 65533
This will delete all BGP configurations on this device.
Continue (y/n)? y

spine01(config)# no route-map ncn-w003
spine01(config)# no route-map ncn-w002
spine01(config)# no route-map ncn-w001
```

The following is an example of the Aruba configuration:

```
route-map rm-ncn-w001 permit seq 10
     match ip address prefix-list pl-nmn
     set ip next-hop 10.252.2.8
route-map rm-ncn-w001 permit seq 20
     match ip address prefix-list pl-hmn
     set ip next-hop 10.254.2.27
route-map rm-ncn-w001 permit seq 30
     match ip address prefix-list pl-can
     set ip next-hop 10.103.10.10
route-map rm-ncn-w002 permit seq 10
     match ip address prefix-list pl-nmn
     set ip next-hop 10.252.2.9
route-map rm-ncn-w002 permit seq 20
     match ip address prefix-list pl-hmn
     set ip next-hop 10.254.2.25
route-map rm-ncn-w002 permit seq 30
     match ip address prefix-list pl-can
     set ip next-hop 10.103.10.9
route-map rm-ncn-w003 permit seq 10
     match ip address prefix-list pl-nmn
     set ip next-hop 10.252.2.18
route-map rm-ncn-w003 permit seq 20
     match ip address prefix-list pl-hmn
     set ip next-hop 10.254.2.26
route-map rm-ncn-w003 permit seq 30
     match ip address prefix-list pl-can
     set ip next-hop 10.103.10.11
!
router bgp 65533
    bgp router-id 10.252.0.1
    maximum-paths 8
    neighbor 10.252.0.3 remote-as 65533
    neighbor 10.252.2.8 remote-as 65533
    neighbor 10.252.2.9 remote-as 65533
    neighbor 10.252.2.18 remote-as 65533
    address-family ipv4 unicast
        neighbor 10.252.0.3 activate
        neighbor 10.252.2.8 activate
        neighbor 10.252.2.8 route-map ncn-w001 in
        neighbor 10.252.2.9 activate
        neighbor 10.252.2.9 route-map ncn-w002 in
        neighbor 10.252.2.18 activate
        neighbor 10.252.2.18 route-map ncn-w003 in
    exit-address-family
```

b. Delete the previous Mellanox configuration.

```
sw-spine-001 [standalone: master] #
sw-spine-001 [standalone: master] # conf t
sw-spine-001 [standalone: master] (config) # no router bgp 65533
sw-spine-001 [standalone: master] (config) # no route-map ncn-w001
sw-spine-001 [standalone: master] (config) # no route-map ncn-w002
sw-spine-001 [standalone: master] (config) # no route-map ncn-w003
```

The following is an example of the Mellanox configuration:

```
## Route-maps configuration
##
   route-map rm-ncn-w001 permit 10 match ip address pl-nmn
   route-map rm-ncn-w001 permit 10 set ip next-hop 10.252.0.7
   route-map rm-ncn-w001 permit 20 match ip address pl-hmn
   route-map rm-ncn-w001 permit 20 set ip next-hop 10.254.0.7
   route-map rm-ncn-w001 permit 30 match ip address pl-can
   route-map rm-ncn-w001 permit 30 set ip next-hop 10.103.8.7
   route-map rm-ncn-w002 permit 10 match ip address pl-nmn
   route-map rm-ncn-w002 permit 10 set ip next-hop 10.252.0.8
   route-map rm-ncn-w002 permit 20 match ip address pl-hmn
   route-map rm-ncn-w002 permit 20 set ip next-hop 10.254.0.8
   route-map rm-ncn-w002 permit 30 match ip address pl-can
   route-map rm-ncn-w002 permit 30 set ip next-hop 10.103.8.8
   route-map rm-ncn-w003 permit 10 match ip address pl-nmn
   route-map rm-ncn-w003 permit 10 set ip next-hop 10.252.0.9
   route-map rm-ncn-w003 permit 20 match ip address pl-hmn
   route-map rm-ncn-w003 permit 20 set ip next-hop 10.254.0.9
   route-map rm-ncn-w003 permit 30 match ip address pl-can
   route-map rm-ncn-w003 permit 30 set ip next-hop 10.103.8.9

##
## BGP configuration
##
   protocol bgp
   router bgp 65533 vrf default
   router bgp 65533 vrf default router-id 10.252.0.1 force
   router bgp 65533 vrf default maximum-paths ibgp 32
```

```
   router bgp 65533 vrf default neighbor 10.252.0.7 remote-as 65533
   router bgp 65533 vrf default neighbor 10.252.0.7 route-map ncn-w001
   router bgp 65533 vrf default neighbor 10.252.0.8 remote-as 65533
   router bgp 65533 vrf default neighbor 10.252.0.8 route-map ncn-w002
   router bgp 65533 vrf default neighbor 10.252.0.9 remote-as 65533
   router bgp 65533 vrf default neighbor 10.252.0.9 route-map ncn-w003
   router bgp 65533 vrf default neighbor 10.252.0.10 remote-as 65533
```

5. Restart the BGP process on the switches.

Once the IPs are updated for the route-maps and BGP neighbors, restart the BGP process on the switches. This may need to be done multiple times for all of the peers to come up.

a. Restart the BGP process for Mellanox switches.

```
# clear ip bgp all
```

b. Restart the BGP process for Aruba switches.

```
# clear bgp *
```

Troubleshooting: If the BGP peers are still not coming up you, check the `Metallb.yaml` configuration file for errors. The `Metallb.yaml` file should point to the NMN IPs of the configure switches.

```
ncn-m001# vi Metallb.yaml
---
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.252.0.2
      peer-asn: 65533
      my-asn: 65533
    - peer-address: 10.252.0.3
      peer-asn: 65533
      my-asn: 65533
    address-pools:
    - name: customer-access
      protocol: bgp
      addresses:
      - 10.102.9.112/28
    - name: customer-access-dynamic
      protocol: bgp
      addresses:
      - 10.102.9.128/25
    - name: hardware-management
      protocol: bgp
      addresses:
      - 10.94.100.0/24
    - name: node-management
      protocol: bgp
      addresses:
      - 10.92.100.0/24
```

If the management network has aggregation switches setup and peering is set up with them, the `Metallb.yaml` configuration file will need to be updated to reflect these IPs. The peer-address should be the IP of the switch that is doing BGP peering.

23.2 Install API Documentation Release Artifacts

The following procedures can be run on a web server or as a docker container. Depending on the environment of the system, one, both, or neither of the procedures can be chosen to execute on the system.

23.3 Install API Documentation Release Artifacts on a Web Server

About this task

Use this procedure to expose the API release artifacts to the web server. Hosting the untarred API/CLI documentation on a web server is required because this web-based documentation makes extensive use of the runtime fetch browser API, which is restricted by same-origin policy in the case of local filesystem URLs. Therefore, browsing the untarred API/CLI documentation via `file:///local_file` is not supported.

Procedure

1. Log into a system hosting a webserver.
2. Download the `shastadocs-directory-*.tar.gz` file from CrayPort.
3. Extract the `shastadocs-directory-*.tar.gz` file onto the web server.

   ```
   # tar xvf shastadocs-directory-unique-identifier.tar.gz
   ```

4. Browse to *http://yourwebserver/cray-portal/public* to view the API documentation.

23.4 Install API Documentation Release Artifacts with Docker

Prerequisites

Task prerequisites

- Docker is installed.

About this task

Use this procedure to expose the API release artifacts on a local host by running the file as a Docker container.

Procedure

1. Log into a system that has docker running.
2. Download the `shastadocs-website-*-dockerimage.tar` file from CrayPort.
3. Load the image.

```
# docker load -i <shastadocs-website-unique-identifier-dockerimage.tar
```

4. Expose the website on the localhost..

```
# docker run -p=8080:80 loaded-image-name
```

5. Browse to `localhost:8080` to view the API documentation.

23.5 Collect Data From Healthy Shasta 1.3 System for EX 1.4 Installation

Prerequisites

Task prerequisites.

About this task

ROLE	• System Administrator
OBJECTIVE	Collect data needed to prepare Shasta v1.4 installation pre-config files, save operational information about which components are disabled and why they are disabled, and save site customizations to use as a guide for customizing a Shasta v1.4 system.
LIMITATIONS	None
NEW IN THIS RELEASE	This entire procedure is new with this release.

Procedure

1. Start a **`prep.install`** typescript with timestamps and run commands to collect information.

   Although Some of the commands in this procedure will output to a file, others will output to stdout and be captured by this typescript file.

```
user@host> ssh ncn-w001
ncn-w001# mkdir -p ~/prep.install.1.4
ncn-w001# cd ~/prep.install.1.4
ncn-w001# script -af prep.install.1.4.$(date +%Y-%m-%d).txt
ncn-w001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Confirm installed version of CLE software.

   Some of the steps below will be different for v1.3.0 versus v1.3.2. Any variation in commands will be marked.

```
ncn-w001# cat /opt/cray/etc/release/cle
PRODUCT="Cray's Linux Environment"
OS=SLES15SP1
ARCH=x86_64
VERSION=1.3.2
DATE=20201014142855
```

3. Obtain user ID and passwords for system components.

   1. Obtain user ID and passwords for all the system components. management network spine, leaf, CDU, and aggregation switches. **For example**: **`sw-spine01-mtl sw-spine02-mtl sw-leaf01-mtl sw-leaf02-mtl sw-cdu01-mtl sw-cdu02-mtl`**

      User id: admin **Password**: PASSWORD

   2. If necessary, obtain the user ID and password for the ClusterStor primary management node. **For example: `cls01053n00`**.

      User id: admin **Password**: PASSWORD

   3. If necessary, obtain the user ID and password for the Arista edge switches.

4. Confirm BMC username/password for management NCNs.

   User id: admin **Password**: PASSWORD

   > **NOTE:** The external connection for ncn-w001 (BMC and node) will be moved to ncn-m001 for installation of the v1.4 software in a later step. It is critical that you be able to connect to the ncn-m001 BMC using a valid administrative username and password.

5. Check that the switches are reachable by substituting the system switch names in the following command for the correct number of spine, leaf, CDU, and aggregation switches in this system.

   ```
   ncn-w001# for switch in sw-leaf0{1,2}-mtl sw-spine0{1,2}-mtl sw-cdu0{1,2}-mtl; \
   do while true; do ping -c 1 $switch > /dev/null; if [[ $? == 0 ]]; \
   then echo "switch $switch is up"; break; \
   else echo "switch $switch is not up"; fi; sleep 5; done; done
   switch sw-leaf01-mtl is up
   switch sw-leaf02-mtl is up
   switch sw-spine01-mtl is up
   switch sw-spine02-mtl is up
   switch sw-cdu01-mtl is up
   switch sw-cdu02-mtl is up
   ```

   > **NOTE:** The IP addresses for these switches will need to be changed in a later step to align with v1.4 software.

6. Collect the current IP addresses for the switches.

   ```
   ncn-w001# grep sw /etc/hosts | grep mtl
   10.1.0.1  sw-spine01-mtl.local sw-spine01-mtl    #-label-10.1.0.1
   10.1.0.2  sw-leaf01-mtl.local sw-leaf01-mtl      #-label-10.1.0.2
   10.1.0.3  sw-spine02-mtl.local sw-spine02-mtl    #-label-10.1.0.3
   10.1.0.4  sw-leaf02-mtl.local sw-leaf02-mtl      #-label-10.1.0.4
   10.1.0.5  sw-cdu01-mtl.local sw-cdu01-mtl        #-label-10.1.0.5
   10.1.0.6  sw-cdu02-mtl.local sw-cdu02-mtl        #-label-10.1.0.6
   ```

7. Check firmware on all leaf, spine, CDU, and aggregation switches meets expected level for v1.4. If they need to be updated, remember that for later in this procedure.

   For minimum Network switch versions, see *Network Firmware* on page 246.

   Check the version of each leaf switch.

   ```
   ncn-w001# ssh admin@sw-leaf01-mtl
   switch# show version
   Dell EMC Networking OS10 Enterprise
   Copyright (c) 1999-2019 by Dell Inc. All Rights Reserved.
   ```

```
OS Version: 10.5.0.2P1
Build Version: 10.5.0.2P1.482
Build Time: 2019-10-29T19:58:10+0000
System Type: S3048-ON
Architecture: x86_64
Up Time: 9 weeks 5 days 05:17:33
```

Check the version for each spine switch.

```
ncn-w001# ssh admin@spine01-mtl
Mellanox Switch
sw-spine01 [standalone: master] > show version
Product name:      Onyx
Product release:   3.9.0300
Build ID:          #1-dev
Build date:        2020-02-26 19:25:24
Target arch:       x86_64
Target hw:         x86_64
Built by:          jenkins@7c42130d8bd6
Version summary:   X86_64 3.9.0300 2020-02-26 19:25:24 x86_64
Product model:     x86onie
Host ID:           506B4BF4FCB0
System serial num: MT1845X03127
System UUID:       7ce27170-e2ba-11e8-8000-98039befd600
Uptime:            68d 5h 21m 36.256s
CPU load averages: 3.20 / 3.20 / 3.18
Number of CPUs:    2
System memory:     2763 MB used / 5026 MB free / 7789 MB total
Swap:              0 MB used / 0 MB free / 0 MB total
```

Check the version for each CDU switch.

```
ncn-w001# ssh admin@sw-cdu01-mtl
switch# show version
Dell EMC Networking OS10 Enterprise
Copyright (c) 1999-2019 by Dell Inc. All Rights Reserved.
OS Version: 10.5.0.2P1
Build Version: 10.5.0.2P1.482
Build Time: 2019-10-29T19:58:10+0000
System Type: S4148T-ON
Architecture: x86_64
Up Time: 4 weeks 6 days 04:01:07
```

Check the version for each aggregation switch.

```
ncn-w001# ssh admin@sw-agg01-mtl
switch# show version
Dell EMC Networking OS10 Enterprise
Copyright (c) 1999-2019 by Dell Inc. All Rights Reserved.
OS Version: 10.5.0.2P1
Build Version: 10.5.0.2P1.482
Build Time: 2019-10-29T19:58:10+0000
System Type: S4148T-ON
Architecture: x86_64
Up Time: 4 weeks 6 days 04:01:07
```

8. Check firmware on Gigabyte nodes. If they need to be updated, remember that for later in this procedure.

   For minimum NCN firmware versions see Node Firmware

   Gigabyte nodes should be at the 21.00.00 version released after Shasta v1.3.2 in December 2020.

```
BIOS version C17
BMC version 12.84.09
CMC version 62.84.02
```

Check which version of sh-svr rpms are installed. If they are less than 21.00.00, then the firmware update bundle has not been installed or applied to the Gigabyte nodes. You will need to have both the firmware release tarball and the *Gigabyte Node Firmware Update Guide* to upgrade the firmware on Gigabyte nodes when indicated later in this procedure.

```
ncn-w001# rpm -qa | grep sh-svr
sh-svr-1264up-bios-21.00.00-20201022025951_2289a89.x86_64
sh-svr-3264-bios-crayctldeploy-0.0.14-20201022025951_2289a89.x86_64
sh-svr-3264-bios-21.00.00-20201022025951_2289a89.x86_64
sh-svr-5264-gpu-bios-21.00.00-20201022025951_2289a89.x86_64
sh-svr-5264-gpu-bios-crayctldeploy-0.0.14-20201022025951_2289a89.x86_64
sh-svr-1264up-bios-crayctldeploy-0.0.14-20201022025951_2289a89.x86_64
```

If the 21.00.00 rpms have been installed, then run this script to check whether the firmware update has been applied to the nodes. Information about how to interpret the output of this command and the procedures for updating Gigabyte compute node firmware or Gigabyte non-compute node firmware is in the *Gigabyte Node Firmware Update Guide*.

Check the BIOS, BMC, and CMC firmware versions on nodes.

```
ncn-w001# bash /opt/cray/FW/bios/sh-svr-1264up-bios/sh-svr-scripts/
find_GBT_Summary_for_Shasta_v1.3_rack.sh \
2>&1 | tee /var/tmp/rvr-inv
```

9. Determine which Boot Orchestration Service (BOS) templates to use to shutdown compute nodes and UANs.

For example:

- Compute nodes: cle-1.3.2
- UANs: uan-1.3.2

10. Obtain the authorization key for SAT.

See **System Security and Authentication**, **Authenticate an Account with the Command Line**, and **SAT Authentication** in the *Cray Shasta Administration Guide 1.3 S-8001* for more information.

v1.3.0: Use Rev C of the guide

v1.3.2: Use Rev E or later

11. Save the list of nodes that are disabled.

```
ncn-w001# sat status --filter Enabled=false > sat.status.disabled
```

12. Save a list of nodes that are off.

```
ncn-w001# sat status --filter State=Off > sat.status.off
```

13. Save the Slurm status on nodes.

```
ncn-w001# ssh nid001000 sinfo > sinfo
```

14. Save the Slurm list of nodes down and the reason why they are down.

```
ncn-w001# ssh nid001000 sinfo --list-reasons > sinfo.reasons
```

15. Get for PBS status on nodes (which are offline or down).

```
ncn-w001# ssh nid001000 pbsnodes -l > pbsnodes
```

16. Check slingshot port status.

```
ncn-w001# /opt/cray/bringup-fabric/status.sh > fabric.status
```

17. Verify Rosetta switches are accessible and healthy.

```
ncn-w001# /opt/cray/bringup-fabric/ssmgmt_sc_check.sh > fabric.ssmgmt_sc_check
```

18. Check current firmware.

```
ncn-w001# sat firmware > sat.firmware
```

There will be a point after the v1.4 software has been installed when firmware needs to be updated on many components. Those components which need a firmware update while v1.3 is booted, will be addressed later.

19. Check Lustre server health.

```
ncn-w001# ssh admin@cls01234n00.system.com
admin@cls01234n00 ~]$ cscli show_nodes
```

20. Save information from HSM about any site-defined groups/labels applied to nodes.

HSM groups might be used in BOS session templates, but they may be used in conjunction with Ansible plays in VCS to configure nodes. Saving this information now will make it easier to reload it after v1.4 has been installed.

Save information about label, description, and members of each group in HSM.

```
ncn-w001# cray hsm groups list --format json > hsm.groups
```

A sample group from that output follows.

```
  {
    "description": "NCNs running Lustre",
    "members": {
      "ids": [
        "x3000c0s7b0n0",
        "x3000c0s9b0n0",
        "x3000c0s11b0n0",
        "x3000c0s25b0n0",
        "x3000c0s13b0n0"
      ]
    },
    "label": "lnet_ncn"
  }
```

21. Save a copy of the master branch in VCS with all of the site modifications from v1.3.

See *Git Tools - Bundling* for information about how to manipulate this bundle repo to extract information.

> **NOTE:** This data cannot be directly imported and used on v1.4, but can serve as a reference to site modifications done with v1.3 which might be similar to those needed with v1.4.

```
ncn-w001# git clone \
https://api-gw-service-nmn.local/vcs/cray/config-management.git
```

```
Cloning into 'config-management'...
remote: Enumerating objects: 3148, done.
remote: Counting objects: 100% (3148/3148), done.
remote: Compressing objects: 100% (846/846), done.
remote: Total 3148 (delta 1341), reused 2518 (delta 1049)
Receiving objects: 100% (3148/3148), 10.35 MiB | 27.59 MiB/s, done.
Resolving deltas: 100% (1341/1341), done.
```

Make a git bundle which is transportable as a single file "VCS.bundle".

```
ncn-w001# git bundle create VCS.bundle HEAD master
```

Check for other branches. There may be other branches, in addition to master, which could be saved.

```
ncn-w001# git branch -a
 remotes/origin/HEAD -> origin/master
  remotes/origin/cray/cme-premium/1.3.0
  remotes/origin/cray/cme-premium/1.3.2
  remotes/origin/cray/cray-pe/20.08
  remotes/origin/cray/cray-pe/20.09
  remotes/origin/cray/cray-pe/20.11
  remotes/origin/master
  remotes/origin/slurm
  remotes/origin/slurm2
```

22. Save a copy of any site-modified recipes in IMS.

There is no need to save any of the HPE-provided recipes, but any modified recipes can be used as a guide to how the v1.4 HPE-provided recipes may need to be modified to meet site needs.

List all recipes in IMS.

```
ncn-w001# cray ims recipes list --format json
```

```
[ { "recipe_type": "kiwi-ng", "linux_distribution": "sles15", "created": "2020-09-04T03:22:15.764123+00:00",
"link": { "path": "s3://ims/recipes/0bfcf98b-3b73-49ad-ab08-5b868ed3dda2/recipe.tar.gz", "etag":
"dec4e4b7dd734a0a24dcf4b67e69c2f5", "type": "s3" }, "id": "0bfcf98b-3b73-49ad-ab08-5b868ed3dda2", "name": "cray-
sles15sp1-barebones-0.1.4" }, { "recipe_type": "kiwi-ng", "linux_distribution": "sles15", "created":
"2020-09-04T03:38:07.259114+00:00", "link": { "path": "s3://ims/recipes/b463fb84-ffaf-4e00-81f1-1682acae2f25/
recipe.tar.gz", "etag": "42f7b0c58ef5db1828dd772405f376b7", "type": "s3" }, "id": "b463fb84-
ffaf-4e00-81f1-1682acae2f25", "name": "cray-sles15sp1-cle" }, { "recipe_type": "kiwi-ng", "linux_distribution":
"sles15", "created": "2020-10-29T23:31:51.340962+00:00", "link": { "path": "s3://ims/recipes/
49c703e9-3b95-4409-804f-b9c0e790487b/recipe.tar.gz", "etag": "", "type": "s3" }, "id": "49c703e9-3b95-4409-804f-
b9c0e790487b", "name": "cray-sles15sp1-cle-1.3.26" } ]
```

Review the list of IMS recipes to determine which you want to save.

For each IMS recipe that you want to save, use **`cray artifacts get ims <object> <filename>`** where **`<object>`** is the S3 key for the `recipe.tgz` (from the recipe list and filename is the filename where you want to save the object). Once downloaded, uncompress the `tgz` file to view the files and directories that comprise the recipe.

From the output above, IMS has this metadata about the cray-sles15sp1-barebones-0.1.4 recipe.

```
{ "recipe_type": "kiwi-ng", "linux_distribution": "sles15", "created": "2020-09-04T03:22:15.764123+00:00", "link":
{ "path": "s3://ims/recipes/0bfcf98b-3b73-49ad-ab08-5b868ed3dda2/recipe.tar.gz", "etag":
"dec4e4b7dd734a0a24dcf4b67e69c2f5", "type": "s3" }, "id": "0bfcf98b-3b73-49ad-ab08-5b868ed3dda2", "name": "cray-
sles15sp1-barebones-0.1.4" },
```

This command will extract the recipe from the IMS S3 bucket and place it into a filename which is based on the name from the IMS recipe.

```
ncn-w001# cray artifacts get ims \
recipes/0bfcf98b-3b73-49ad-ab08-5b868ed3dda2/recipe.tar.gz \
cray-sles15sp1-barebones-0.1.4.tgz
```

23. Save a copy of any BOS session templates of interest.

There might be organizational information, such as the names of BOS session templates or how they specify the nodes to be booted using a list of groups or list of nodes, but there might also be special kernel parameters added by the site.

```
ncn-w001# cray bos v1 sessiontemplate list --format \
json > bos.sessiontemplate.list.json
```

24. Save a copy of any IMS images which might have been directly customized.

If all images were built from an IMS recipe and then customized with CFS before booting, then this step can be skipped.

List all IMS images.

```
ncn-w001# cray ims images list --format json

[
  {
    "link": {
      "path": "s3://boot-images/9f32ed7b-9e1c-444d-8b63-40cefbf7e846/
manifest.json",
      "etag": "db1b16d771a5a88cb320519e066348b8",
      "type": "s3"
    },
    "id": "9f32ed7b-9e1c-444d-8b63-40cefbf7e846",
    "name": "cle_default_rootfs_cfs_2488e992-ee60-11ea-88f1-b42e993b710e",
    "created": "2020-09-04T03:42:51.474549+00:00"
  },
  {
    "link": {
      "path": "s3://boot-images/dc1a430f-82cb-4b9b-a1dd-7c8540721013/
manifest.json",
      "etag": "5f243b60b4211c47e79b837df2271692",
      "type": "s3"
    },
    "id": "dc1a430f-82cb-4b9b-a1dd-7c8540721013",
    "name": "cle_default_rootfs_cfs_test_20200914125841b",
    "created": "2020-09-14T20:46:51.367749+00:00"
  },
]
```

Review the list of IMS images to determine which you want to save.

This example uses the "cle_default_rootfs_cfs_test_20200914125841b" image from above.

For each image that you want to save perform the following steps, use **`cray artifacts get boot-images <object> <filename>`** where object is the S3 key for the IMS manifest file and filename is the filename you want to save the manifest file to.

For each artifact in the manifest file, use **`cray artifacts get boot-images <object> <filename>`** where object is the S3 key for the image artifact (from the manifest file) and filename is the filename you want to save the artifact to.

Get the manifest file using the path to the S3 boot-images bucket and save as a local filename `manifest.json`.

```
ncn-w001# cray artifacts get boot-images fe4b8429-4ea7-4696-8018-2c7950a75e4b/
manifest.json manifest.json
```

Example manifest.json file for an IMS image.

```
ncn-w001# cat manifest.json
{
    "artifacts": [
        {
            "link": {
                "etag": "1f9f88a65be7fdd8190008e06b1da2c0-175",
                "path": "s3://boot-images/dc1a430f-82cb-4b9b-a1dd-7c8540721013/
rootfs",
                "type": "s3"
                },
            "md5": "d402ea98531f995106918815e4d74cc4",
            "type": "application/vnd.cray.image.rootfs.squashfs"
        },
        {
            "link": {
                "etag": "d76adf4ee44b6229dad69103c38d49ca",
                "path": "s3://boot-images/dc1a430f-82cb-4b9b-a1dd-7c8540721013/
kernel",
                "type": "s3"
            },
            "md5": "d76adf4ee44b6229dad69103c38d49ca",
            "type": "application/vnd.cray.image.kernel"
        },
        {
            "link": {
                "etag": "43b0f96a951590cc478ae7b311e3e413-4",
                "path": "s3://boot-images/dc1a430f-82cb-4b9b-a1dd-7c8540721013/
initrd",
                "type": "s3"
            },
            "md5": "9112b42d5e3da6902cc62158d6482552",
            "type": "application/vnd.cray.image.initrd"
        }
    ],
    "created": "2020-09-14 20:47:01.211752",
    "version": "1.0"
}
```

Changes might have been made in the `initrd` or in the `rootfs` or both. This example shows how to extract one of those files.

Extract the `rootfs` (squashfs) file.

```
ncn-w001# cray artifacts get boot-images dc1a430f-82cb-4b9b-a1dd-7c8540721013/
rootfs rootfs
```

25. Dump the information from SLS.

Some information from SLS can be extracted for use in the pre-config files, `hmn_connections.json` and `cabinets.yaml` for the v1.4 installation. The *Cray Shasta Administration Guide 1.3 S-8001* has a section "Dump SLS Information".

If there is a fully current SHCD (Shasta Cabling Diagram, previously called CCD for Cray Cabling Diagram) spreadsheet file for this system, there is a way to extract information from it to create the hmn_connections.json file later in the v1.4 installation process. However, SLS data may be more current than the SHCD or there may not be a valid SHCD file for this system if cabling changes have not been recorded as updates to the SHCD file. Saving this SLS information while v1.3 is booted may provide a point of comparison with the data in the SHCD.

This procedure will create three files in the current directory (private_key.pem, public_key.pem, sls_dump.json). These files should be kept in a safe and secure place as the private key can decrypt the encrypted passwords stored in the SLS dump file.

Use the get_token function to retrieve a token to validate requests to the API gateway.

```
ncn-w001# function get_token () {
    ADMIN_SECRET=$(kubectl get secrets admin-client-auth -
ojsonpath='{.data.client-secret}' | base64 -d)
    curl -s -d grant_type=client_credentials -d client_id=admin-client -d
client_secret=$ADMIN_SECRET \
        https://api-gw-service-nmn.local/keycloak/realms/shasta/protocol/openid-
connect/token | python \
        -c 'import sys, json; print json.load(sys.stdin)["access_token"]'
}
```

Generate a private and public key pair.

Execute the following commands to generate a private and public key to use for the dump.

```
ncn-w001# openssl genpkey -out private_key.pem -algorithm RSA \
-pkeyopt rsa_keygen_bits:2048
ncn-w001# openssl rsa -in private_key.pem -outform PEM -pubout \
-out public_key.pem
```

The above commands will create two files the private key private_key.pem file and the public key public_key.pem file.

Make sure to use a new private and public key pair for each dump operation, and do not reuse an existing private and public key pair. The private key should be treated securely because it will be required to decrypt the SLS dump file when the dump is loaded back into SLS. Once the private key is used to load state back into SLS, it should be considered insecure.

Perform the SLS dump.

The SLS dump will be stored in the sls_dump.json file. The sls_dump.json and private_key.pem files are required to perform the SLS load state operation.

```
ncn-w001# curl -X POST \
https://api-gw-service-nmn.local/apis/sls/v1/dumpstate \
-H "Authorization: Bearer $(get_token)" \
-F public_key=@public_key.pem > sls_dump.json

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  591k    0  590k  100   663   122k    137  0:00:04  0:00:04 --:--:--  163k
```

26. Save the contents of the `/opt/cray/site-info directory`.

The `customizations.yaml` and `certs` subdirectory have information which is needed for the v1.4 install.

```
ncn-w001# tar cf site-info.tar /opt/cray/site-info
```

27. Collect MAC address information from the leaf switches for each management NCN and its BMC.

This will be used during the v1.4 install in the ncn_metadata.csv file.

```
ncn-w001# ssh admin@sw-leaf01-mtl
```

Show vlan 4 which is the NMN. This shows the MAC addresses which will be used to boot the nodes.

```
sw-leaf01# show mac address-table vlan 4
VlanId  Mac Address             Type            Interface
4       00:0a:5c:90:1b:bf       dynamic         port-channel100
4       00:0a:9c:62:20:2e       dynamic         ethernet1/1/41
4       00:40:a6:82:f7:73       dynamic         ethernet1/1/42
4       00:40:a6:83:07:ff       dynamic         ethernet1/1/43
4       00:40:a6:83:08:65       dynamic         ethernet1/1/44
4       00:40:a6:83:08:8d       dynamic         ethernet1/1/45
4       3c:2c:30:67:c0:b5       dynamic         port-channel100
4       50:6b:4b:9c:c6:48       dynamic         port-channel100
4       50:9a:4c:e0:32:d1       dynamic         port-channel100
4       50:9a:4c:e0:88:d1       dynamic         port-channel100
4       98:03:9b:ef:d6:48       dynamic         port-channel100
4       b4:2e:99:3b:70:28       dynamic         ethernet1/1/46
4       b4:2e:99:3b:70:30       dynamic         ethernet1/1/47
4       b4:2e:99:3b:70:50       dynamic         ethernet1/1/34
4       b4:2e:99:3b:70:58       dynamic         ethernet1/1/48
4       b4:2e:99:3b:70:c0       dynamic         ethernet1/1/38
4       b4:2e:99:3b:70:c8       dynamic         ethernet1/1/36
4       b4:2e:99:3b:70:d0       dynamic         ethernet1/1/37
4       b4:2e:99:3b:70:d4       dynamic         port-channel100
4       b4:2e:99:3b:70:d8       dynamic         port-channel100
4       b4:2e:99:3e:82:66       dynamic         ethernet1/1/33
4       b4:2e:99:a6:5d:df       dynamic         ethernet1/1/25
4       b8:59:9f:2b:2f:9e       dynamic         port-channel100
4       b8:59:9f:2b:30:fa       dynamic         port-channel100
4       b8:59:9f:34:88:be       dynamic         port-channel100
4       b8:59:9f:34:88:c6       dynamic         port-channel100
4       b8:59:9f:34:89:3a       dynamic         port-channel100
4       b8:59:9f:34:89:46       dynamic         port-channel100
4       b8:59:9f:34:89:4a       dynamic         port-channel100
4       b8:59:9f:f9:1b:e6       dynamic         port-channel100
```

Show vlan 1 which is the HMN which has the node's BMC MAC address. This is needed for DHCP of the node BMC.

```
sw-leaf01# show mac address-table vlan 1
VlanId  Mac Address             Type            Interface
1       3c:2c:30:67:c0:b5       dynamic         port-channel100
1       50:6b:4b:9c:c6:20       dynamic         port-channel100
1       50:6b:4b:9c:c6:48       dynamic         port-channel100
1       50:9a:4c:e0:32:d1       dynamic         port-channel100
1       50:9a:4c:e0:88:d1       dynamic         port-channel100
1       98:03:9b:ef:d6:20       dynamic         port-channel100
1       98:03:9b:ef:d6:48       dynamic         port-channel100
1       b8:59:9f:2b:2e:aa       dynamic         port-channel100
1       b8:59:9f:2b:2e:b6       dynamic         port-channel100
1       b8:59:9f:2b:2f:9e       dynamic         port-channel100
1       b8:59:9f:2b:30:82       dynamic         port-channel100
1       b8:59:9f:2b:30:fa       dynamic         port-channel100
1       b8:59:9f:2b:31:0a       dynamic         port-channel100
1       b8:59:9f:34:88:be       dynamic         port-channel100
1       b8:59:9f:34:88:c6       dynamic         port-channel100
1       b8:59:9f:34:89:3a       dynamic         port-channel100
1       b8:59:9f:34:89:46       dynamic         port-channel100
1       b8:59:9f:34:89:4a       dynamic         port-channel100
1       b8:59:9f:f9:1b:e6       dynamic         port-channel100
sw-leaf01# exit
```

Another way to collect information about the BMC MAC address is this loop. Set the correct number of storage and worker nodes.

```
ncn-w001# nodes=""
ncn-w001# for name in ncn-m00{1,2,3}  ncn-s00{1,2,3} ncn-w00{1,2,3,4,5}; \
do nodes="$nodes $name"; done
ncn-w001# echo $nodes
ncn-m001 ncn-m002 ncn-m003 ncn-s001 ncn-s002 ncn-s003 ncn-w001 ncn-w002 ncn-
w003 ncn-w004 ncn-w005
```

Use "ipmitool lan print" to determine the BMC MAC address.

- Gigabyte nodes should use "lan print 1".
- Intel nodes should use "lan print 3".

```
ncn-w001# for ncn in $nodes; do echo $ncn; ssh $ncn ipmitool lan print 1 \
| grep "MAC Address"; done

ncn-m001
MAC Address             : b4:2e:99:3b:70:c0
ncn-m002
MAC Address            : b4:2e:99:3b:70:d0
ncn-m003
MAC Address             : b4:2e:99:3b:70:c8
ncn-s001
MAC Address             : b4:2e:99:3b:70:d8
ncn-s002
MAC Address             : b4:2e:99:3b:70:d4
ncn-s003
MAC Address             : b4:2e:99:3b:70:58
ncn-w001
MAC Address            : b4:2e:99:3b:71:10
ncn-w002
MAC Address            : b4:2e:99:3b:70:50
ncn-w003
MAC Address             : b4:2e:99:3e:82:66
ncn-w004
MAC Address             : b4:2e:99:a6:5d:df
ncn-w005
MAC Address             : b4:2e:99:3b:70:28
```

28. Collect output from IP address for all management NCNs.

```
ncn-w001# for ncn in $nodes; do echo $ncn; ssh $ncn ip address ; done
```

29. Check full BMC information for ncn-w001.

The connection will be moved to ncn-m001.

The IP address, subnet mask, and default gateway IP will be needed.

```
ncn-w001# ipmitool lan print
Set in Progress         : Set Complete
Auth Type Support       : NONE MD2 MD5 PASSWORD OEM
Auth Type Enable        : Callback : MD5
                        : User     : MD5
                        : Operator : MD5
                        : Admin    : MD5
                        : OEM      : MD5
IP Address Source       : Static Address
```

```
IP Address              : 172.30.56.3
Subnet Mask             : 255.255.240.0
MAC Address             : b4:2e:99:3b:71:10
SNMP Community String   : AMI
IP Header               : TTL=0x40 Flags=0x40 Precedence=0x00 TOS=0x10
BMC ARP Control         : ARP Responses Enabled, Gratuitous ARP Disabled
Gratituous ARP Intrvl   : 1.0 seconds
Default Gateway IP      : 172.30.48.1
Default Gateway MAC     : 00:00:00:00:01:50
Backup Gateway IP       : 0.0.0.0
Backup Gateway MAC      : 00:00:00:00:00:00
802.1q VLAN ID          : Disabled
802.1q VLAN Priority    : 0
RMCP+ Cipher Suites     : 0,1,2,3,6,7,8,11,12,15,16,17
Cipher Suite Priv Max   : caaaaaaaaaaaXXX
                        :     X=Cipher Suite Unused
                        :     c=CALLBACK
                        :     u=USER
                        :     o=OPERATOR
                        :     a=ADMIN
                        :     O=OEM
Bad Password Threshold  : 0
Invalid password disable: no
Attempt Count Reset Int.: 0
User Lockout Interval   : 0
```

30. Check Mellanox firmware for the CX-4 and CX-5 on management NCNs.

For minimum NCN firmware versions, refer to *Node Firmware* on page 244.

Confirm the version of the mft rpm installed on the management NCNs

```
ncn-w001# for ncn in $nodes; do echo $ncn; ssh $ncn rpm -q mft; done
```

Check version of firmware installed.

```
ncn-w001# for ncn in $nodes; do echo $ncn; ssh $ncn mlxfwmanager \
| egrep "FW|Device" ; done
```

Output from each node will look similar to the following example. Compare this information with the versions in *Node Firmware* on page 244.

```
Device #1:
  Device Type:      ConnectX4
  PCI Device Name:  0000:42:00.0
     FW             12.26.4012     N/A
Device #2:
  Device Type:      ConnectX5
  PCI Device Name:  0000:41:00.0
     FW             16.28.4000     N/A
Device #3:
  Device Type:      ConnectX5
  PCI Device Name:  0000:81:00.0
     FW             16.28.4000     N/A
```

31. When reinstalling a v1.3 system with v1.4 software, preserving the SDU configuration, plug-ins, and locally stored dump files may be desired. Run the following command and store the resulting tar file.

```
ncn-w001# tar czvf sdu-shastav1.3.tar.gz /opt/cray/sdu \
/opt/cray/sdu-shasta-plugins/default \
/etc/opt/cray/sdu /var/opt/cray/sdu
```

32. SMA services are stateful and stateless. For stateful SMA services, the data is saved in PVs. The data comprises of configuration attributes, telemetry streams, logging info, etc. Upon an upgrade, the data in all SMA PVs must be saved. The enumeration of PVs for SMA services is displayed by running the following command.

    ```
    ncn-w001# kubectl -n sma get pvc
    ```

33. LDMS, an SMA service, saves configuration data on the base OS for CNs and NCNs. The base OS directory `/etc/sysconfig/ldms.d` and all sub-directories under this path must be preserved, both on CNs and NCNs. Additionally, NCN configuration must be preserved in the locations as follows.

    - Samplers: `/etc/sysconfig/ldms.d dir` and the entire tree must be preserved.
    - Aggregator: The following PVs along with the data, must be preserved:
        - sma/ldms-sms-aggr-pvc
        - sma/ldms-sms-smpl-pvc
        - sma/ldms-compute-aggr-pvc
        - sma/ldms-compute-smpl-pvc

34. Finish the typescript file.

    ```
    ncn-w001# exit
    ```

35. Save the typescript file and the output from all of the above commands somewhere off of the system.

23.5.1 Save Fabric and LAG configuration from 1.3.x

The Slingshot software changed significantly between HPE Cray EX 1.3.x and HPE Cray EX 1.4 to support Slingshot in non-Kubernetes environments. Before wiping a 1.3.x system, save all the configuration files that are needed to install a new Fabric Manager. This includes Fabric configuration files and LAG configuration files. If you have made any customized port configurations, save all those files before wiping the system.

Before saving the configuration files, start a `prep.install` typescript with timestamps and run commands to collect information.

```
ncn-w001# mkdir -p ~/prep.install.1.4
ncn-w001# cd ~/prep.install.1.4
ncn-w001# script -af prep.install.1.4.$(date +%Y-%m-%d).txt
ncn-w001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

The following fabric configuration files must be saved:

- `/opt/cray/etc/sct/Shasta_system_hsn_pt_pt.csv`

  Navigate to `/opt/cray/etc/sct`. View and save the `Shasta_system_hsn_pt_pt.csv` file.

  ```
  ncn-w001# cd /opt/cray/etc/sct
  ```

  ```
  ncn-w001# /opt/cray/etc/sct # more Shasta_system_hsn_pt_pt.csv
  cable_id,src_conn_a,src_conn_b,dst_conn_a,dst_conn_b,stage,src_egress_a,src_egress_b,dst_egress_a,d
  st_egress_b,link_type,src_group,dst_group,part_number,part_length,calculated_distance,route
  3000.3000.00.0001,x3000c0r24j12,none,x3000c0s11b1n1h0,x3000c0s19b4n4h0,1,none,none,none,none,edge,0
  ,n/a,102234601,1.5,1.1684676,[]
  3000.3000.00.0000,x3000c0r24j14,none,x3000c0s7b1n1h0,x3000c0s9b1n1h0,1,none,none,none,none,edge,0,n
  ```

```
/a,102234601,1.5,1.3723802,[]
3000.3000.00.0003,x3000c0r24j8,none,x3000c0s19b3n3h0,x3000c0s19b2n2h0,1,none,none,none,none,edge,0,
n/a,102234600,1,0.8796151,[]
3000.3000.00.0002,x3000c0r24j10,none,x3000c0s28b3n3h0,x3000c0s28b2n2h0,1,none,none,none,none,edge,0
,n/a,102234600,1,0.9138218,[]
3000.3000.00.0005,x3000c0r24j4,none,x3000c0s26b3n3h0,x3000c0s26b2n2h0,1,none,none,none,none,edge,0,
n/a,102234600,1,0.7517303,[]
3000.3000.00.0004,x3000c0r24j6,none,x3000c0s26b1n1h0,x3000c0s28b4n4h0,1,none,none,none,none,edge,0,
n/a,102234600,1,0.8585811,[]
3000.3000.00.0007,x3000c0r24j16,none,x3000c0s28b1n1h0,NC,1,none,none,none,none,edge,0,n/a,102234600
,1,0.8398439,[]
3000.3000.00.0006,x3000c0r24j2,none,x3000c0s19b1n1h0,x3000c0s26b4n4h0,1,none,none,none,none,edge,0,
n/a,102234600,1,0.7234268,[]
3000.3000.00.0008,x3000c0r24j18,none,x7000c0s19b1n1h0,x7000c0s26b4n4h0,1,none,none,none,none,edge,0
,n/a,102234600,1,0.7234268,[]
3000.3000.00.0009,x3000c0r24j20,none,x7000c0s20b1n1h0,x7000c0s27b4n4h0,1,none,none,none,none,edge,0
,n/a,102234600,1,0.7234268,[]
```

- `/opt/cray/fabric_template.json`: Saving this file is optional since it can be regenerated from the **Shasta_system_hsn_pt_pt.csv**.

In addition, LAG configuration files should be saved. To configure LAG, information like ports and DMAC value is required. Use the following procedure to save the LAG configuration files.

1. Log on to a worker node.
2. Save the following file that contains ports that form the LAG. Navigate to the directory where you have saved the file. Note that there is no default location for this file.

```
ncn-w001# cat arista_ports.json
    {
       "name": "arista-ports",
       "ports": [
         "x1000c1r5j17p0",
         "x1000c0r7j17p0",
         "x1000c4r7j17p0",
         "x1000c5r5j17p0",
         "x1001c0r7j17p0",
         "x1001c1r5j17p0",
         "x1001c4r7j17p0",
         "x1001c5r5j17p0",
         "x1003c0r7j17p0",
         "x1003c4r7j17p0",
         "x1003c5r5j17p0",
         "x1003c1r5j17p0",
         "x1002c4r7j17p0",
         "x1002c5r5j17p0",
         "x1002c0r7j17p0",
         "x1002c1r5j17p0"
       ]
    }
```

3. Save the DMAC key value pair.

```
ncn-w001# cat 1.json
    {
       "lagname": "AristaStorage1",
       "dmacs": ["98:5d:82:71:bf:d9"]
    }
```

4. Combine the two files (`arista_ports.json` and `1.json`) into a single file (`lagProperties.json`) that contains the ports and the MAC address metadata. Save this file so that it can be used to configure LAG in 1.4.

```
ncn-w001# more lagProperties.json
{
  "lagPropertyMap": {
    "1": {
      "portLinks": [
        "/fabric/ports/x1000c1r5j17p0",
        "/fabric/ports/x1000c0r7j17p0",
        "/fabric/ports/x1000c4r7j17p0",
        "/fabric/ports/x1000c5r5j17p0",
```

```
        "/fabric/ports/x1001c0r7j17p0",
        "/fabric/ports/x1001c1r5j17p0",
        "/fabric/ports/x1001c4r7j17p0",
        "/fabric/ports/x1001c5r5j17p0",
        "/fabric/ports/x1003c0r7j17p0",
        "/fabric/ports/x1003c4r7j17p0",
        "/fabric/ports/x1003c5r5j17p0",
        "/fabric/ports/x1003c1r5j17p0",
        "/fabric/ports/x1002c4r7j17p0",
        "/fabric/ports/x1002c5r5j17p0",
        "/fabric/ports/x1002c0r7j17p0",
        "/fabric/ports/x1002c1r5j17p0"
      ],
      "dmacs" : ["98:5d:82:71:bf:d9"]
    }
  }
}
```

If you do not have the two configuration files (`arista_ports.json` and `1.json`) that are used to configure LAG for a 1.3.x system, you can query the fabric controller to get the list of LAGs and their configurations.

1. Log on to a worker node. Get the autorization token.

   ```
   ncn-w001# curl -k -s -S -d grant_type=client_credentials -d client_id=admin-client -d
   client_secret=`kubectl get secrets admin-client-auth -o jsonpath='{.data.client-secret}' | base64 -
   d`
   https://api-gw-service-nmn.local/keycloak/realms/shasta/protocol/openid-connect/token > /opt/cray/
   bringup-fabric/token.json
   ```

2. Get a list of LAGs. You can do a GET operation on fabric controller url: **/apis/fc/v2/lags**

   ```
   ncn-w001# curl -s -X GET -H "Content-Type: application/json" -H
   "Authorization: Bearer $(cat /opt/cray/bringup-fabric/token.json | jq -r '.access_token')" https://
   api-gw-service-nmn.local/apis/fc/v2/lags
   {"lags":[]}
   ```

3. Get the LAG configuration identified by LAG ID.

   For each LAG, you can do a GET operation on fabric controller url **/apis/fc/v2/lags**

   ```
   ncn-w001# curl -s -X GET -H "Content-Type: application/json" -H
   "Authorization: Bearer $(cat /opt/cray/bringup-fabric/token.json | jq -r '.access_token')" https://
   api-gw-service-nmn.local/apis/fc/v2/lags/251
   {"code":-1,"message":"lag does not exist"}
   ```

4. Finish the typescript file.

   ```
   ncn-w001# exit
   ```

23.6 Service Guides

About this task

Background info for task.

Procedure

ENVIRONMENTS

1. When installing a freshly racked system or fresh installing an existing system, the system must have access to one of the following:

- LiveCD, refer to *Configure USB LiveCD* on page 34.
- Linux and a serial cable,refer to*NCN Metadata over USB-Serial* on page 251

There are two parts to the NCN metadata file.

- Collecting the MAC of the BMC
- Collecting the MAC(s) of the shasta-network interface(s)

2. For more information on NCN networking, refer to *NCN Networking* on page 233.

   NCNs may have 1 or more bond interfaces, which may be comprised from one or more physical interfaces. The preferred default configuration is two physical network interfaces per bond. The number of bonds themselves depends on the systems network topology.

   FILES

   - **`ncn_metadata.csv`**

3. Confirm site and PCIe connections, refer to *Update Site Connections* on page 219 and *Netboot an NCN from a Spine Switch* on page 251.

4. Collect BMC MAC and NCN MAC addresses in order to (re)create `ncn_metadata.csv`. Refer to *Collect BMC MAC Addresses* on page 246 and *Collect NCN MAC Addresses* on page 248.

   The following example uses a single PCIe card.

```
Xname,Role,Subrole,BMC MAC,Bootstrap MAC,Bond0 MAC0
x3000c0s9b0n0,Management,Storage,
94:40:c9:37:77:26,14:02:ec:d9:76:88,14:02:ec:d9:76:89
x3000c0s8b0n0,Management,Storage,94:40:c9:37:87:5a,
14:02:ec:d9:7b:c8,14:02:ec:d9:7b:c9
x3000c0s7b0n0,Management,Storage,94:40:c9:37:0a:2a,14:02:ec:d9:7c:
88,14:02:ec:d9:7c:89
x3000c0s6b0n0,Management,Worker,94:40:c9:37:77:b8,14:02:ec:da:bb:
00,14:02:ec:da:bb:01
x3000c0s5b0n0,Management,Worker,
94:40:c9:35:03:06,14:02:ec:d9:76:b8,14:02:ec:d9:76:b9
x3000c0s4b0n0,Management,Worker,94:40:c9:37:67:60,14:02:ec:d9:7c:
40,14:02:ec:d9:7c:41
x3000c0s3b0n0,Management,Master,
94:40:c9:37:04:84,14:02:ec:d9:79:e8,14:02:ec:d9:79:e9
x3000c0s2b0n0,Management,Master,
94:40:c9:37:f9:b4,14:02:ec:da:b8:18,14:02:ec:da:b8:19
x3000c0s1b0n0,Management,Master,
00:00:00:00:00:00,14:02:ec:da:b5:18,14:02:ec:da:b5:d9
```

   The following example uses dual PCIe cards.

```
Xname,Role,Subrole,BMC MAC,Bootstrap MAC,Bond0 MAC0,Bond0 MAC1
x3000c0s9b0n0,Management,Storage,
94:40:c9:37:77:26,14:02:ec:d9:76:88,98:40:c9:d9:76:88
x3000c0s8b0n0,Management,Storage,94:40:c9:37:87:5a,
14:02:ec:d9:7b:c8,98:40:c9:d9:7b:c8
x3000c0s7b0n0,Management,Storage,94:40:c9:37:0a:2a,14:02:ec:d9:7c:
88,98:40:c9:d9:7c:88
x3000c0s6b0n0,Management,Worker,94:40:c9:37:77:b8,14:02:ec:da:bb:
00,98:40:c8:da:bb:00
```

```
x3000c0s5b0n0,Management,Worker,
94:40:c9:35:03:06,14:02:ec:d9:76:b8,98:40:c9:d9:76:b8
x3000c0s4b0n0,Management,Worker,94:40:c9:37:67:60,14:02:ec:d9:7c:
40,98:40:c9:d9:7c:40
x3000c0s3b0n0,Management,Master,
94:40:c9:37:04:84,14:02:ec:d9:79:e8,98:40:c9:d9:79:e8
x3000c0s2b0n0,Management,Master,
94:40:c9:37:f9:b4,14:02:ec:da:b8:18,98:40:c9:da:b8:18
x3000c0s1b0n0,Management,Master,00:00:00:00:00:00,94:40:c9:5f:b5:de,
94:40:c9:5f:b5:de
```

- **`switch_metadata.csv`**

5. Refer to *Construct the Switch Metadata File* on page 253 for instructions on creating this file.

   - **`hmn_connections.json`**

6. Refer to *Construct HMN Connections File* on page 167.

23.7 Construct HMN Connections File

Prerequisites

- SHCD Excel file for the system
- Podman or Docker Running

About this task

The following procedure shows the process for generating the `hmn_connections.json` from the system's SHCD Excel document. This process is typically needed when generating the `hmn_connections.json`file for a new system, or regenerating it when system's SHCD is changed (specifically the HMN tab). The hms-shcd-parser tool can be used to generate the `hmn_connections.json` file.

Procedure

1. (Docker Only) Confirm that the Docker service is running.

   ```
   root@ncn-m001:~ # systemctl status docker
   ● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor
   preset: disabled)
   Active: inactive (dead)
       Docs: http://docs.docker.com
   ```

   If the service is not running, start it.

   ```
   root@ncn-m001 # systemctl start docker
   ```

2. Load the hms-shcd-parser docker image from the CSM release distribution.

   Change directories into the location where the CSM release distribution was extracted.

```
ncn-m001:~/ # cd /path/to/extracted/csm
```

Load the hms-shcd-parser docker image.

```
ncn-m001:~/csm-0.7.10 # ./hack/load-container-image.sh dtr.dev.cray.com/cray/
hms-shcd-parser:1.1.1
```

3. Set the environment to point to the system's SHCD Excel file.

```
ncn-m001:~/ # export SHCD_FILE="/path/to/systems/SHCD.xsls"
```

4. Generate the hmn_connections.json file from SHCD.

   - If using Podman:

     ```
     ncn-m001:~/ # podman run --rm -it --name hms-shcd-parser -v "$(realpath "$SHCD_FILE")":/input/shcd_file.xlsx -
     v "$(pwd)":/output dtr.dev.cray.com/cray/hms-shcd-parser:1.1.1
     ```

   - If using Docker:

     ```
     ncn-m001:~/ # docker run --rm -it --name hms-shcd-parser -v "$(realpath "$SHCD_FILE")":/input/shcd_file.xlsx -
     v "$(pwd)":/output dtr.dev.cray.com/cray/hms-shcd-parser:1.1.1
     ```

23.8 Construct cabinets.yaml File

This page provides information on how to construct the optional `cabinets.yaml` file. This file lists cabinet IDs for any systems with non-contiguous cabinet ID numbers and controls how the `csi config init` command treats cabinet ids.

The `cabinets.yaml` file is important for upgrades from v1.3 systems, as it allows the preservation of cabinet names and network VLANs. An audit of the existing system will be required to gather the data needed to populate `cabinets.yaml`.

In the example below, the VLANs for cabinets 1000 and 1001 are overridden. This example can be used to preserve existing cabinet VLANs and prevent reconfiguring switches and CMMs. Similar cabinet numbering and preservation of VLANs can be used for Hill and River cabinets

Cabinet names and VLANs for `comptype_cabinet` should be collected from SLS data and used to populate `cabinets.yaml` for the system. Failure to do this will result in needing to change switch and CMM configurations. Use the original cabinet data for V1.3 systems, refer to *Collect Data From Healthy Shasta 1.3 System for EX 1.4 Installation* on page 151, for information on how to collect the data needed.

The cabinets.yaml file is maually created and follows the format in the example below. Each "type" of cabinet can have several fields:

- `total_number` of cabinets of this type
- `starting_id` for this cabinet type
- List of IDs

```
---
cabinets:
- type: hill
  total_number: 2
  starting_id: 9000
- type: mountain
  total_number: 4
  starting_id: 1000
```

```
  cabinets:
    - id: 1000
      nmn-vlan: 2000
      hmn-vlan: 3000
    - id: 1001
      nmn-vlan: 2001
      hmn-vlan: 3001
    - id: 1002
    - id: 1003
- type: river
  total_number: 4
  starting_id: 3000
```

In the above example, there are two Hill cabinets that will be automatically numbered as 9000 and 9001. The Mountain cabinets appear in three groupings of four IDs. The River cabinets are non-contiguous in four separated IDs.

A system will Hill cabinets can have one to four cabinet IDs. There is no limit on the number of Mountain or River cabinets.

When the above `cabinets.yaml` file is used, the fields for each type will take precedence over any command line argument to `csi config init` for `starting-mountain-cabinet`, `starting-river-cabinet`, `starting-hill-cabinet`, `mountain-cabinets`, `river-cabinets`, or `hill-cabinets`. If these command line arguments provide information which is not in the cabinets.yaml file, then the information will be merged with the information provided in `cabinets.yaml`

23.9 Configure the BIOS of a Gigabyte UAN

Prerequisites

About this task

Before the UAN product can be installed on Gigabyte UANs, specific network interface and boot settings must be configured in the BIOS.

Procedure

1. Press the **Delete** key to enter the setup utility when prompted to do so in the console.
2. Navigate to the boot menu.
3. Set the **Boot Option #1** field to `Network:UEFI: PXE IP4 Intel(R) I350 Gigabit Network Connection`.
4. Set all other **Boot Option** fields to `Disabled`.
5. Ensure that the Boot mode select is set to `[UEFI]`.
6. Select **Save & Exit** to save the settings.
7. Select **Yes** to confirm and press the **Enter** key.

The UAN will reboot.

8. **Optional:** Run the following IPMI commands if the BIOS settings do not persist.

   In these example commands, the BMC of the UAN is x3000c0s27b0. Replace *USERNAME* and *PASSWORD* with username and password of the BMC of the UAN. These commands do the following:

   1. Power off the node
   2. Perform a reset.
   3. Set the PXE boot in the options.
   4. Power on the node

```
ncn-m001# ipmitool -I lanplus -U *** -P *** -H x3000c0s27b0 power off
ncn-m001# ipmitool -I lanplus -U *** -P *** -H x3000c0s27b0 mc reset cold
ncn-m001# ipmitool -I lanplus -U *** -P *** -H x3000c0s27b0 chassis bootdev pxe
options=efiboot,persistent
ncn-m001# ipmitool -I lanplus -U *** -P *** -H x3000c0s27b0 power on
```

23.10 Configure the BIOS of an HPE UAN

Prerequisites

Configure the BMC for UANs with iLO on page 172

About this task

Before the UAN product can be installed on HPE UANs, specific network interface and boot settings must be configured in the BIOS.

Procedure

1. Force a UAN to reboot into the BIOS.

   In the following command, *UAN_BMC_XNAME* is the xname of the BMC of the UAN to configure. Replace *USER* and *PASSWORD* with the BMC username and password, respectively.

```
ncn-m001# ipmitool -U USER -P PASSWORD -H UAN_BMC_XNAME -I lanplus \
chassis bootdev pxe options=efiboot,persistent
```

2. Monitor the console of the UAN using either ConMan or the following command:

```
ncn-m001# ipmitool -U USER -P PASSWORD -H UAN_BMC_XNAME -I \
lanplus sol activate
```

   Refer to the section "About the ConMan Containerized Service" in the publication *HPE Cray EX System Administration Guide (S-8001)* for more information about ConMan.

3. Press the **ESC** and **9** keys to access the BIOS System Utilities when the option appears.

4. Ensure that OCP Slot 1 Port 1 is the only port with **Boot Mode** set to `Network Boot` (PXE). All other ports must have **Boot Mode** set to `Disabled`.

The settings must match the following example.

```
--------------------
       System Configuration

       BIOS Platform Configuration (RBSU) > Network Options > Network Boot Options > PCIe Slot
Network Boot

       Slot 1 Port 1 : Marvell FastLinQ 41000 Series -   [Disabled]
       2P 25GbE SFP28 QL41232HLCU-HC MD2 Adapter - NIC

       Slot 1 Port 2 : Marvell FastLinQ 41000 Series -   [Disabled]
       2P 25GbE SFP28 QL41232HLCU-HC MD2 Adapter - NIC

       Slot 2 Port 1 : Network Controller                [Disabled]
       OCP Slot 10 Port 1 : Marvell FastLinQ 41000       [Network Boot]
       Series - 2P 25GbE SFP28 QL41232HQCU-HC OCP3
       Adapter - NIC

       OCP Slot 10 Port 2 : Marvell FastLinQ 41000       [Disabled]
       Series - 2P 25GbE SFP28 QL41232HQCU-HC OCP3
       Adapter - NIC
       --------------------
```

5. Set the **Link Speed** to `SmartAN` for all ports.

```
 --------------------
        System Utilities

        System Configuration > Main Configuration Page > Port Level
Configuration

        Link Speed                                        [SmartAN]
        FEC Mode                                          [None]
        Boot Mode                                         [PXE]
        DCBX Protocol                                     [Dynamic]
        RoCE Priority                                     [0]
        PXE VLAN Mode                                     [Disabled]
        Link Up Delay                                     [30]
        Wake On LAN Mode                                  [Enabled]
        RDMA Protocol Support                             [iWARP + RoCE]
        BAR-2 Size                                        [8M]
        VF BAR-2 Size                                     [256K]
        ----------------------
```

6. Set the boot options to match the following example.

```
 ----------------------
        System Utilities

        System Configuration > BIOS/Platform Configuration (RBSU) > Boot Options

        Boot Mode                                         [UEFI
Mode]
        UEFI Optimized Boot
[Enabled]
        Boot Order Policy                                 [Retry Boot Order
Indefinitely]

        UEFI Boot Settings
        Legacy BIOS Boot Order
        ------------------------
```

7. Set the **UEFI Boot Order** settings to match the following example.

The order must be:

1. USB
2. Local disks
3. OCP Slot 10 Port 1 IPv4
4. OCP Slot 10 Port 1 IPv6

```
-----------------------
        System Utilities

        System Configuration > BIOS/Platform Configuration (RBSU) > Boot Options > UEFI Boot
Settings > UEFI Boot Order

        Press the '+' key to move an entry higher in the boot list and the '-' key to move an entry
lower
        in the boot list. Use the arrow keys to navigate through the Boot Order list.

        Generic USB Boot
        SATA Drive  Box 1 Bay 1 : VK000480GWTHA
        SATA Drive  Box 1 Bay 2 : VK000480GWTHA
        SATA Drive  Box 1 Bay 3 : VK001920GWTTC
        SATA Drive  Box 1 Bay 4 : VK001920GWTTC
        OCP Slot 10 Port 1 : Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HQCU-HC OCP3
Adapter -
        NIC - Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HQCU-HC OCP3 Adapter - PXE (PXE
IPv4)
        OCP Slot 10 Port 1 : Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HQCU-HC OCP3
Adapter -
        NIC - Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HQCU-HC OCP3 Adapter - PXE (PXE
IPv6)
        -------------------------
```

23.11 Configure the BMC for UANs with iLO

Prerequisites

Perform the first three steps of *Prepare for UAN Product Installation* on page 105.

About this task

ROLE	System administrator, system installer
OBJECTIVE	Enable the IPMI/DCMI settings on an HPE UAN that are necessary to continue UAN product installation on an HPE Cray EX supercomputer.

Procedure

1. Create the SSH tunnel necessary to access the BMC web GUI interface.

   a. Find the IP or hostname for a UAN.

   b. Create an SSH tunnel to the UAN BMC.

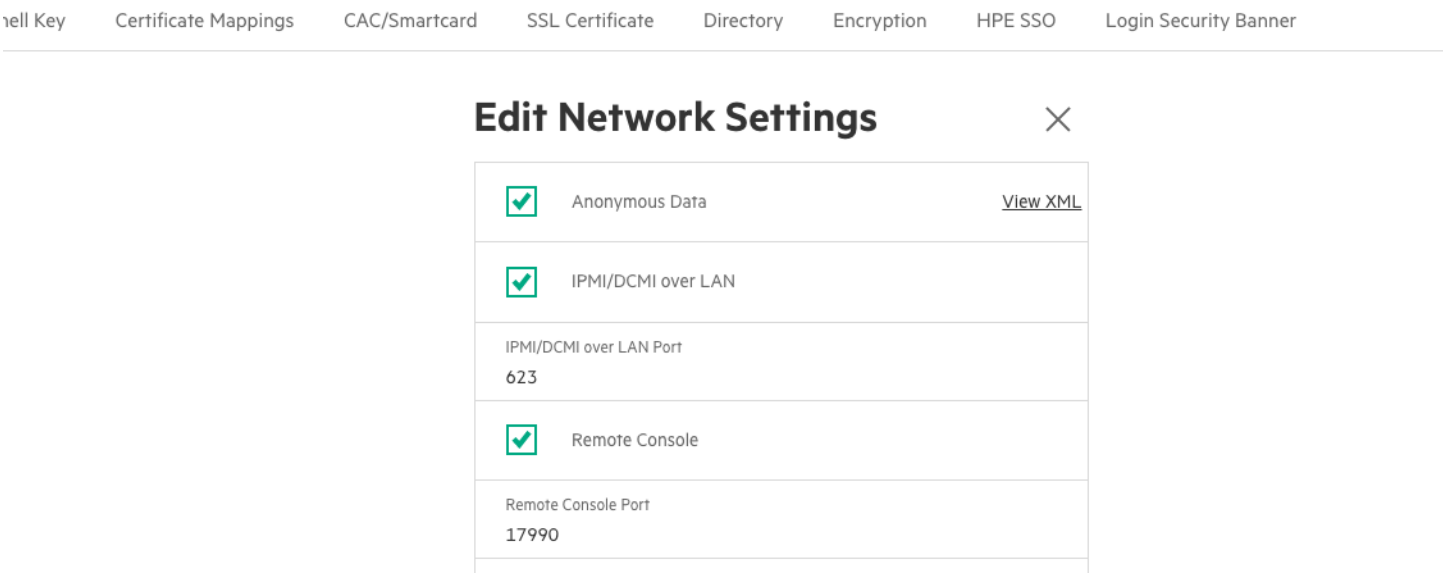
      In the following example, `uan01-mgmt` is the UAN and `shasta-ncn-m001` is the NCN the admin is logged into.

      ```
      # ssh -L 8443:uan01-mgmt:443 shasta-ncn-m001
      ```

c. Wait for SSH to establish the connection.

2. Open https://127.0.0.1:8443 in web browser on the NCN to access the BMC web GUI.
3. Log in to the web GUI using default credentials.
4. Click **Security** in the menu on the left side of the screen.
5. Click **Access Settings** in the menu at the top of the screen.
6. Click the pencil icon next to **Network** in the main window area.
7. Check the box next to **IPMI/DCMI over LAN**.
8. Ensure that the remove management settings match the following screenshot.

Figure 4. IPMI/DCMI configuration for HPE UANs



23.12 BOS Session Templates

A session template can be created by specifying parameters as part of the call to the Boot Orchestration Service (BOS). When calling BOS directly, JSON is passed as part of the call.

Session templates can be used to boot images that are customized with the Image Management Service (IMS). A session template has a collection of one or more boot set objects. A boot set defines a collection of nodes and the information about the boot artifacts and kernel parameters used to boot them. This information is written to the Boot Script Service (BSS) and sent to each node over the specified network, enabling these nodes to boot.

The Simple Storage Service (S3) is used to store the `manifest.json` file that is created by IMS. This file contains links to all of the boot artifacts. The following S3 parameters are used in a BOS session template:

- *`type`*: This is the type of storage used. Currently, the only allowable value is `s3`.

- *path*: This is the path to the `manifest.json` file in S3. The path will follow the `s3://<BUCKET_NAME>/<KEY_NAME>` format.
- *etag*: This entity tag helps identify the version of the `manifest.json` file. Currently not used but cannot be left blank.

The following is an example BOS session template:

```
{
 "cfs_url": "https://api-gw-service-nmn.local/vcs/cray/csm-config-management.git", <<-- Configuration manifest API
endpoint
 "enable_cfs": true, <<-- Invokes CFS
 "name": "session-template-example",
 "boot_sets": {
   "boot_set1": {
     "network": "nmn",
     "boot_ordinal": 1,
     "kernel_parameters": "console=ttyS0,115200 bad_page=panic crashkernel=360M hugepagelist=2m-2g intel_iommu=off
intel_pstate=disable iommu=pt ip=dhcp numa_interleave_omit=headless numa_zonelist_order=node oops=panic
pageblock_order=14 pcie_ports=native printk.synchronous=y rd.neednet=1 rd.retry=10 rd.shell k8s_gw=api-gw-service-
nmn.local quiet turbo_boost_limit=999",
     "rootfs_provider": "cpss3",
     "node_list": [   <<-- List of individual nodes
       "x3000c0s19b1n0"
     ],
     "etag": "foo", <<-- Used to identify the version of the manifest.json file
     "path": "s3://boot-images/e06530f1-fde2-4ca5-9148-7e84f4857d17/manifest_sans_boot_parameters.json", <<-- The path
to the manifest.json file in S3
     "rootfs_provider_passthrough": "66666666:dvs:api-gw-service-nmn.local:300:eth0",
     "type": "s3" <<-- Type of storage
   }
 },
```

When multiple boot sets are used in a session template, the *boot_ordinal* and *shutdown_ordinal* values indicate the order in which boot sets need to be acted upon. Boot sets sharing the same ordinal number will be addressed at the same time.

Each boot set needs its own set of S3 parameters (*path*, *type*, and optionally *etag*).

Specify Nodes in a BOS Session Template

There are three different ways to specify the nodes inside a boot set in a BOS session template. The node list, node groups, or node role groups values can be used. Each can be specified as a comma separated list.

Node list

The `"node_list"` value is a list of nodes identified by xnames.

For example:

```
"node_list": ["x3000c0s19b1n0", "x3000c0s19b1n1", "x3000c0s19b2n0"]
```

Node groups

The `"node_groups"` value is a list of groups defined by the Hardware State Manager (HSM). Each group may contain zero or more nodes. Groups can be arbitrarily defined.

For example:

```
"node_groups": ["green", "white", "pink"]
```

Node roles groups

The node role groups is a list of groups based on a node's designated role. Each node's role is specified in the HSM database. For example, to target all of the nodes with a "Compute" role, "Compute" would need to be specified in the `"node_role_groups"` value.

For example:

```
"node_roles_groups": ["Compute"]
```

The following roles are defined in the HSM database:

- Compute
- Service
- System
- Application
- Storage
- Management

23.13 Boot Orchestration Service (BOS)

The Boot Orchestration Service (BOS) is responsible for booting, configuring, and shutting down collections of nodes. This is accomplished using BOS components, such as boot orchestration session templates and sessions, as well as launching a Boot Orchestration Agent (BOA) that fulfills boot requests.

BOS users create a BOS session template via the REST API. A session template is a collection of metadata for a group of nodes and their desired boot artifacts and configuration. A BOS session can then be created by applying an action to a session template. The available actions are boot, reboot, shutdown, and configure. BOS will create a Kubernetes BOA job to apply an action. BOA coordinates with the underlying subsystems to complete the action requested. The session can be monitored to determine the status of the request.

BOS depends on each of the following services to complete its tasks:

- BOA - Handles any action type submitted to the BOS API. BOA jobs are created and launched by BOS.
- Boot Script Service (BSS) - Stores the configuration information that is used to boot each hardware component. Nodes consult BSS for their boot artifacts and boot parameters when nodes boot or reboot.
- Configuration Framework Service (CFS) - BOA launches CFS to apply configuration to the nodes in its boot sets (node personalization).
- Cray Advanced Platform Monitoring and Control (CAPMC) - Used to power on and off the nodes.
- Hardware State Manager (HSM) - Tracks the state of each node and what groups and roles nodes are included in.

Use the BOS Cray CLI Commands

BOS utilizes the Cray CLI commands. The latest API information can be found with the following command:

```
ncn-w001# cray bos list
[[results]]
major = "1"
minor = "0"
patch = "0"
[[results.links]]
href = "https://api-gw-service-nmn.local/apis/bos/v1"
rel = "self"
```

23.14 Compute Node Boot Sequence

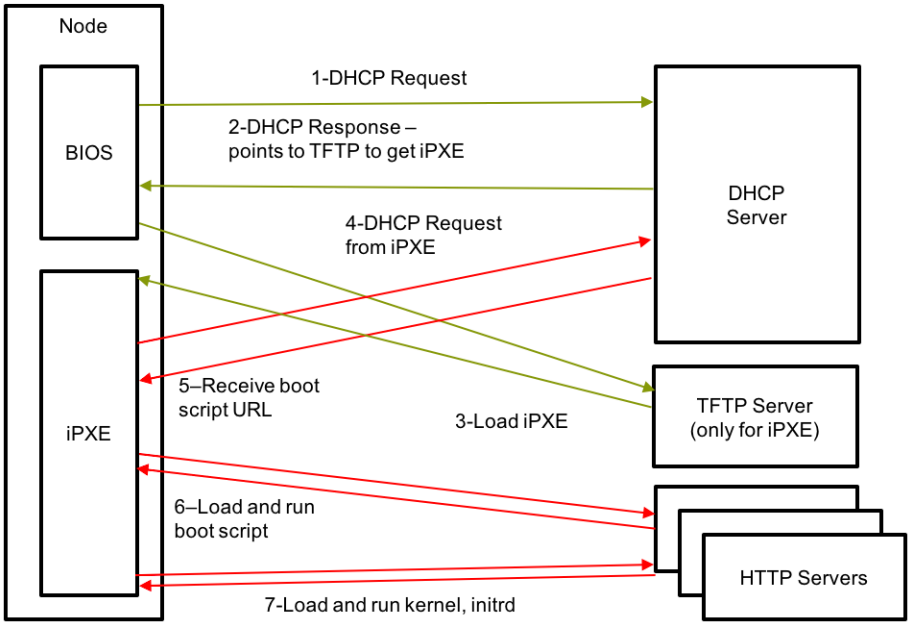
The following is a high-level overview of the boot sequence for compute nodes:

1. The compute node is powered on.
2. The BIOS issues a DHCP discover request.
3. DHCP responds with the following:
   a. `next-server`, which is the IP address of the TFTP server.
   b. The name of the file to download from the TFTP server.
4. The node's PXE sends a TFTP request to the TFTP server.
5. If the TFTP server has the requested file, it sends it to the node's PXE. In this case, the file name is `ipxe.efi`, which is a Cray-crafted iPXE binary that points at the Boot Script Service (BSS). The BSS will then serve up another iPXE boot script.
6. The `ipxe.efi` file downloads another `ipxe` boot script from BSS.

   This script provides information for downloading:

   - The location of `kernel`.
   - The location of `initrd`.
   - A string containing the kernel parameters.
7. The node attempts to download `kernel` and `initrd` boot artifacts. If successful, it will boot using these and the kernel parameters. Otherwise, it will retry to download these boot artifacts indefinitely.

Figure 5. Node Boot Flow



There may be times when certain issues may be encountered during the compute node boot up process. In order to resolve these issues, it is important to understand the underlying cause, symptoms, and stage at which the issue has occurred. The exact process and tools required to resolve the issue depends on this information.

23.15 Troubleshoot PXE Boot

About this task

The following page details how to troubleshoot various issues that arise when trying to PWE boot nodes in a HPE Cray EX system.

Procedure

REQUIRED PXE BOOT CONFIGURATION

1. To successfully PXE boot nodes, the following is required.
   - The IP helper-address must be configured on VLAN 1,2,4,7. This will be where the layer 3 gateway exists (spine or agg)
   - The virtual-IP/VSX/MAGP IP must be configured on VLAN 1,2,4,7.
   - There must be a static route pointing to the tftp server (Aruba Only).
   - M001 needs an active gateway on VLAN1 this can be identified from MTL.yaml generated from CSI.
   - M001 needs an IP helper-address on VLAN1 pointing to 10.92.100.222.

   The following is an example of a snippet of `MTL.yaml`.

```
name: network_hardware
  net-name: MTL
  vlan_id: 0
  comment: ""
  gateway: 10.1.0.1
```

ARUBA CONFIGURATION

2. Check the configuration for `interface vlan x`.

   This configuration will be the same on both switches (except the IP address). There is an `active-gateway` and `ip helper-address` configured.

```
sw-spine-002(config)# show run int vlan 1
interface vlan1
    vsx-sync active-gateways
    ip address 10.1.0.3/16
    active-gateway ip mac 12:01:00:00:01:00
    active-gateway ip 10.1.0.1
    ip mtu 9198
    ip helper-address 10.92.100.222
    exit

sw-spine-002(config)# show run int vlan 2
```

```
interface vlan2
    vsx-sync active-gateways
    ip address 10.252.0.3/17
    active-gateway ip mac 12:01:00:00:01:00
    active-gateway ip 10.252.0.1
    ip mtu 9198
    ip helper-address 10.92.100.222
    exit

sw-spine-002(config)# show run int vlan 4
interface vlan4
    vsx-sync active-gateways
    ip address 10.254.0.3/17
    active-gateway ip mac 12:01:00:00:01:00
    active-gateway ip 10.254.0.1
    ip mtu 9198
    ip helper-address 10.94.100.222
    exit

sw-spine-002(config)# show run int vlan 7
interface vlan7
    vsx-sync active-gateways
    ip address 10.103.13.3/24
    active-gateway ip mac 12:01:00:00:01:00
    active-gateway ip 10.103.13.111
    ip mtu 9198
    ip helper-address 10.92.100.222
    exit
```

If any of the configuration is missing, both switches must be updated.

```
sw-spine-002# conf t
sw-spine-002(config)# int vlan 1
sw-spine-002(config-if-vlan)# ip helper-address 10.92.100.222
sw-spine-002(config-if-vlan)# active-gateway ip mac 12:01:00:00:01:00
sw-spine-002(config-if-vlan)# active-gateway ip 10.1.0.1

sw-spine-002# conf t
sw-spine-002(config)# int vlan 2
sw-spine-002(config-if-vlan)# ip helper-address 10.92.100.222
sw-spine-002(config-if-vlan)# active-gateway ip mac 12:01:00:00:01:00
sw-spine-002(config-if-vlan)# active-gateway ip 10.252.0.1

sw-spine-002# conf t
sw-spine-002(config)# int vlan 4
sw-spine-002(config-if-vlan)# ip helper-address 10.94.100.222
sw-spine-002(config-if-vlan)# active-gateway ip mac 12:01:00:00:01:00

sw-spine-002# conf t
sw-spine-002(config)# int vlan 7
sw-spine-002(config-if-vlan)# ip helper-address 10.92.100.222
sw-spine-002(config-if-vlan)# active-gateway ip mac 12:01:00:00:01:00
sw-spine-002(config-if-vlan)# active-gateway ip xxxxxxx
sw-spine-002(config-if-vlan)# write mem
```

3. Verify the route to the TFTP server is in place.

   This is a static route to get to the TFTP server via a worker node. The worker node IP can be found in NMN.yaml from CSI generated data.

```
- ip_address: 10.252.1.9
    name: ncn-w001
    comment: x3000c0s4b0n0
    aliases:
```

```
sw-spine-002(config)# show ip route static

Displaying ipv4 routes selected for forwarding

'[x/y]' denotes [distance/metric]

0.0.0.0/0, vrf default
    via  10.103.15.209,  [1/0],  static
10.92.100.60/32, vrf default
    via  10.252.1.7,  [1/0],  static
```

The example above shows the route is `10.92.100.60/32 via 10.252.1.7` with `10.252.1.7` being the worker node.

If the static route is missing, it must be added.

```
sw-spine-001(config)# ip route 10.92.100.60/32 10.252.1.7
```

MELLANOX CONFIGURATION

4. Check the configuration for `interface vlan 1`.

   This configuration will be the same on both switches (except the IP address). There is an `magp` and `ip dhcp relay` configured.

```
sw-spine-001 [standalone: master] # show run int vlan 1
interface vlan 1
interface vlan 1 ip address 10.1.0.2/16 primary
interface vlan 1 ip dhcp relay instance 2 downstream
interface vlan 1 magp 1
interface vlan 1 magp 1 ip virtual-router address 10.1.0.1
interface vlan 1 magp 1 ip virtual-router mac-address 00:00:5E:00:01:01
```

   If the configuration is missing, both switches must be updated.

```
sw-spine-001 [standalone: master] # conf t
sw-spine-001 [standalone: master] (config) # interface vlan 1 magp 1
sw-spine-001 [standalone: master] (config interface vlan 1 magp 1) # ip virtual-
router address 10.1.0.1
sw-spine-001 [standalone: master] (config interface vlan 1 magp 1) # ip virtual-
router mac-address 00:00:5E:00:01:01
sw-spine-001 [standalone: master] # conf t
sw-spine-001 [standalone: master] (config) # ip dhcp relay instance 2 vrf
default
sw-spine-001 [standalone: master] (config) # ip dhcp relay instance 2 address
10.92.100.222
sw-spine-001 [standalone: master] (config) # interface vlan 2 ip dhcp relay
instance 2 downstream
```

5. Verify the VLAN 1 MAGP configuration.

```
sw-spine-001 [standalone: master] # show magp 1

MAGP 1:
  Interface vlan: 1
  Admin state   : Enabled
  State         : Master
  Virtual IP    : 10.1.0.1
  Virtual MAC   : 00:00:5E:00:01:01
```

6. Verify the DHCP relay configuration.

```
sw-spine-001 [standalone: master] (config) # show ip dhcp relay instance 2

VRF Name: default

DHCP Servers:
  10.92.100.222

DHCP relay agent options:
  always-on         : Disabled
  Information Option: Disabled
  UDP port          : 67
  Auto-helper       : Disabled

-----------------------------------------
Interface   Label             Mode
-----------------------------------------
vlan1       N/A               downstream
vlan2       N/A               downstream
vlan7       N/A               downstream
```

7. Verify that the route to the TFTP server and the route for the ingress gateway are available.

```
sw-spine-001 [standalone: master] # show ip route 10.92.100.60

Flags:
  F: Failed to install in H/W
  B: BFD protected (static route)
  i: BFD session initializing (static route)
  x: protecting BFD session failed (static route)
  c: consistent hashing
  p: partial programming in H/W

VRF Name default:

------------------------------------------------------------------------------
----------------------
  Destination       Mask              Flag     Gateway
Interface        Source     AD/M

------------------------------------------------------------------------------
----------------------
  default           0.0.0.0           c        10.101.15.161
eth1/12          static     1/1
  10.92.100.60      255.255.255.255   c        10.252.0.5
vlan2            bgp        200/0
                                     c        10.252.0.6
vlan2            bgp        200/0
                                     c        10.252.0.7
vlan2            bgp        200/0
```

```
sw-spine-001 [standalone: master] # show ip route 10.92.100.71

Flags:
  F: Failed to install in H/W
  B: BFD protected (static route)
  i: BFD session initializing (static route)
  x: protecting BFD session failed (static route)
  c: consistent hashing
  p: partial programming in H/W

VRF Name default:

------------------------------------------------------------------------------
----------------------
  Destination       Mask              Flag     Gateway
Interface        Source     AD/M

------------------------------------------------------------------------------
----------------------
  default           0.0.0.0           c        10.101.15.161
eth1/12          static     1/1
  10.92.100.71      255.255.255.255   c        10.252.0.5
vlan2            bgp        200/0
                                     c        10.252.0.6
vlan2            bgp        200/0
                                     c        10.252.0.7
vlan2            bgp        200/0
```

If the routes are missing, refer to *Configure BGP Neighbors on Management Switches* on page 146.

RESTART BSS

8. If the following error is observed, roll-out a restart of the BSS deployment from any other NCN (likely m002 if executing the m001 reboot).

```
https://api-gw-service-nmn.local/apis/bss/boot/v1/bootscript...X509 chain
0x6d35c548 added X509 0x6d360d68 "surtur.dev.cray.com"
X509 chain 0x6d35c548 added X509 0x6d3d62e0 "Platform CA - L1
(a0b073c8-5c9c-4f89-b8a2-a44adce3cbdf)"
X509 chain 0x6d35c548 added X509 0x6d3d6420 "Platform CA (a0b073c8-5c9c-4f89-
b8a2-a44adce3cbdf)"
EFITIME is 2021-02-26 21:55:04
HTTP 0x6d35da88 status 404 Not Found
```

```
ncn-m002# kubectl -n services rollout restart deployment cray-bss
deployment.apps/cray-bss restarted
```

Wait for the following command to return.

```
ncn-m002# # kubectl -n services rollout status deployment cray-bss
Waiting for deployment "cray-bss" rollout to finish: 1 out of 3 new replicas
have been updated...
Waiting for deployment "cray-bss" rollout to finish: 1 out of 3 new replicas
have been updated...
Waiting for deployment "cray-bss" rollout to finish: 1 out of 3 new replicas
have been updated...
Waiting for deployment "cray-bss" rollout to finish: 2 out of 3 new replicas
```

```
have been updated...
Waiting for deployment "cray-bss" rollout to finish: 2 out of 3 new replicas
have been updated...
Waiting for deployment "cray-bss" rollout to finish: 2 out of 3 new replicas
have been updated...
Waiting for deployment "cray-bss" rollout to finish: 1 old replicas are pending
termination...
Waiting for deployment "cray-bss" rollout to finish: 1 old replicas are pending
termination...
deployment "cray-bss" successfully rolled out
```

9. Reboot the NCN.

RESTART KEA

10. Get KEA pod.

```
ncn-w003:~ # kubectl get pods -n services | grep kea
cray-dhcp-kea-6bd8cfc9c5-m6bgw                                    3/3
Running     0          20h
```

11. Delete the pod.

```
ncn-w003:~ # kubectl delete pods -n services cray-dhcp-kea-6bd8cfc9c5-m6bgw
```

23.16 Troubleshoot COS Image Building Failure

About this task

There is a template macro which did not get expanded properly in the 1.4 RC4 COS image. The macro is in the URL to an RPM repository, and appears in two files: the recipe IMS uses to build the image, and the script in the image that initializes the RPM repositories.

The mistaken path in the image recipe looks like the following example.

```
<!-- Cray COS SLES15sp1 CN, Nexus repo -->
<repository type="rpm-md" alias="cos-%{_cos_version}-sle-15sp1-compute"
priority="2" imageinclude="true">
<source path="https://packages.local/repository/cos-%{_cos_version}-sle-15sp1-
compute/"/>
</repository>
```

The following procedure details how to download the recipe, modify it, and upload it in place of the current recipe.

Procedure

1. Find the recipe in cray-product-catalog. Search for the version "2.0.27".

```
ncn-m001# mkdir -p ims
ncn-m001# kubectl get cm cray-product-catalog -n services -o json | jq -r
```

```
".data | \"\(.cos)\""
2.0.27:
  configuration:
    clone_url: https://vcs.groot.dev.cray.com/vcs/cray/cos-config-management.git
    commit: 0b355a48804393af3e7de675710b00c62860d441
    import_branch: cray/cos/2.0.27
    import_date: 2021-03-10 19:35:25.299144
    ssh_url: git@vcs.groot.dev.cray.com:cray/cos-config-management.git
  images:
    cray-shasta-compute-sles15sp1.x86_64-1.4.66:
      id: ae957914-80f7-4613-b34e-44ead1d4e12b
  recipes:
    cray-shasta-compute-sles15sp1.x86_64-1.4.66:
      id: 31ef7800-1437-42c7-9deb-a164405be975
```

2. Retrieve the artifact for the recipe `cray-shasta-compute-sles15sp1.x86_64-1.4.66`.

```
ncn-m001# cray artifacts get ims recipes/31ef7800-1437-42c7-9deb-a164405be975/
recipe.tar.gz recipe.tar.gz
```

3. Extract the tar files.

```
ncn-m001# cd ims
ncn-m001:~/ims# tar xvf ../recipe.tar.gz
./
./config.xml
./root/
./root/root/
./root/root/bin/
./root/root/bin/zypper-addrepo.sh
./root/etc/
./root/etc/motd
./root/etc/sysconfig/
./root/etc/sysconfig/network/
./root/etc/sysconfig/network/ifcfg-nmn0
./root/etc/sysconfig/network/ifcfg-eth0
./root/etc/zypp/
./root/etc/zypp/zypp.conf
./root/usr/
./root/usr/lib/
./root/usr/lib/systemd/
./root/usr/lib/systemd/system/
./root/usr/lib/systemd/system/grub_config.service
./config.sh
./images.sh
```

```
ncn-m001:~/ims# grep -r cos_version
./config.xml:    <repository type="rpm-md" alias="cos-%{_cos_version}-sle-15sp1-
compute" priority="2" imageinclude="true">
./config.xml:        <source path="https://packages.local/repository/cos-%
{_cos_version}-sle-15sp1-compute/"/>
./root/root/bin/zypper-addrepo.sh:zypper addrepo --priority 2 https://
packages.local/repository/cos-%{_cos_version}-sle-15sp1-compute/ cos-%
{_cos_version}-sle-15sp1-compute
```

4. Edit `./config.xml` and `./root/root/bin/zypper-addrepo.sh` to replace the field "%{_cos_version}" with "2.0".

5. Repackage the recipe tar file.

```
ncn-m001:~/ims# tar cvf ../recipe.tar.gz  .
```

6. Remove the existing artifact and replace it with the new one.

```
ncn-m001:~/ims# cray artifacts delete ims recipes/31ef7800-1437-42c7-9deb-
a164405be975/recipe.tar.gz
ncn-m001:~/ims# cd ..
ncn-m001# cray artifacts create ims recipes/31ef7800-1437-42c7-9deb-
a164405be975/recipe.tar.gz recipe.tar.gz
artifact = "recipes/31ef7800-1437-42c7-9deb-a164405be975/recipe.tar.gz"
Key = "recipes/31ef7800-1437-42c7-9deb-a164405be975/recipe.tar.gz
```

Continue building COS compute node images, return to step *13.c* on page 91.

23.17 CSM Health Checks and Install Validation

The health and stability of the Cray System Management (CSM) product needs to be checked after the CSM installation is completed. Health checks can be run any time after the `install.sh` script completes to verify that CSM is behaving as expected, including the following scenarios:

- Before and after one of the NCNs reboot
- After the system is brought back up
- After an Emergency Power Off (EPO) event or system power cycle
- Anytime unexpected behavior is observed
- More information is needed for support tickets

The following health checks are available:

- *Platform Health Checks* on page 184
- *Network Health Checks* on page 188
- *Hardware Management Services (HMS) Health Checks* on page 190
- *Cray Management Services (CMS) Health Checks* on page 192
- *Automated Goss Testing Health Checks* on page 195
- *Boot CSM Barebones Image Health Check* on page 195
- *UAS and UAI Health Checks* on page 198

The health checks above should be tested in the order they are listed on this page. Errors in an earlier check may cause errors in later checks due to dependencies.

23.17.1 Platform Health Checks

Platform health check scripts do not verify results. Script output includes analysis that is needed to determine if each health check has passed or failed. All health checks are expected to pass.

Platform health checks can be run any time after the `install.sh` script completes to verify that Cray System Management (CSM) is behaving as expected, including the following scenarios:

- Before and after one of the NCNs reboot.

- After the system or a single node goes down unexpectedly.
- After the system is gracefully shut down and brought up.
- Anytime there is unexpected behavior on the system to get a baseline of data for the CSM services and components.
- More information is needed for support tickets.

Health Check scripts can be found and run on any worker or master node from any directory.

Platform `ncnHealthChecks`

The `ncnHealthChecks.sh` script is located in the following directory:

```
ncn# /opt/cray/platform-utils/ncnHealthChecks.sh
```

The `ncnHealthChecks` script reports the following health information for the user to verify:

- Determine and report the number and type of non-compute nodes (NCN) on the system.
- Verify the Kubernetes status for NCN master and worker nodes.

  Run the following command and report the results:

  ```
  ncn-m001# kubectl get nodes -o wide
  ```

  For more information, refer to "Kubernetes® Architecture" in the *HPE Cray EX System Administration Guide S-8001.*
- Verify the Ceph health status.

  Run the following command on first storage node and report results:

  ```
  ncn-s001# ceph -s
  ```

  Ensure that a *`HEALTH_OKAY`* status is returned. A *`HEALTH_WARN`* status may be okay at times if there are too few PGs per OSD and/or large omap objects.

  For more information on Ceph issues, see the "Utility Storage" section in the *HPE Cray EX System Administration Guide S-8001*.
- Check the health of the etcd clusters.

  Check for a healthy report for each etcd cluster member. For etcd health issues, see "Check the Health and Balance of etcd Clusters" in the *HPE Cray EX System Administration Guide S-8001*.
- Check the number of pods in each etcd cluster and verify that they are balanced across worker nodes.

  For etcd number and balance issues, refer to "Check the Health and Balance of etcd Clusters" in the *HPE Cray EX System Administration Guide S-8001*.
- Check if any alarms are set for every etcd cluster.

  For issues with etcd cluster alarms, see the "Check for and Clear etcd Cluster Alarms" procedure in the *HPE Cray EX System Administration Guide S-8001*.
- Check the health of each etcd cluster's database.

  For issues with etcd databases, see the "Check the Health and Balance of etcd Clusters" and "Clear Space in an etcd Cluster Database" sections in the *HPE Cray EX System Administration Guide S-8001*.

- List the automated etcd backups for the Boot Orchestration Service (BOS), Boot Script Service (BSS), Compute Rolling Upgrade Service (CRUS), Domain Name Service (DNS) and Firmware Action Service (FAS).

  For issues with automated etcd backups, see "Backups for etcd-operator Clusters" and "Restore an etcd Cluster from a Backup" in the *HPE Cray EX System Administration Guide S-8001*.
- Execute the uptime command on each NCN master, worker, and storage node, and then report the results.
- Verify the number of pods running on each NCN worker node.

  For any issues, refer to "Retrieve Cluster Health Information Using Kubernetes" in the *HPE Cray EX System Administration Guide S-8001.*
- List unhealthy pods or pods yet to reach the running state.

  Refer to the *HPE Cray EX System Administration Guide S-8001* for the service indicated.

Run the `ncnHealthChecks` script and analyze the output of each of the individual checks above.

Platform Get Xnames Check

The `getXnames.sh` script is located in the following directory:

```
ncn# /opt/cray/platform-utils/getXnames.sh
```

The `getXnames.sh` script reports a list of NCN hostnames, including their associated xnames and the `metal.no-wipe` setting.

A `metal.no-wipe` setting of one is expected, indicating that the data on the NCN will be preserved during a reboot. The `getXnames.sh` script requires that the CLI command is enabled on the NCN worker or master node that it is executed on. If the `metal.no-wipe` status for one or more NCNs is not returned, re-run the `getXnames.sh` script.

For more information, refer to "Check and Set the *`metal.no-wipe`* Setting on NCNs" in the *HPE Cray EX System Administration Guide S-8001*.

Platform Postgres Health Checks

The `ncnPostgresHealthChecks` script is located in the following directory:

```
ncn# /opt/cray/platform-utils/ncnPostgresHealthChecks.sh
```

For each Postgres cluster the `ncnPostgresHealthChecks` script determines the leader pod and then reports the status of all Postgres pods in the cluster and pertinent log entries.

Run the `ncnPostgresHealthChecks` script and verify leader for each cluster and status of cluster members.

For a particular Postgres cluster, expect output similar to the following:

```
--- patronictl, version 1.6.5, list for services leader pod cray-sls-postgres-0 ---
+ Cluster: cray-sls-postgres (6938772644984361037) ---+----+-----------+
|        Member        |    Host    |  Role  |  State  | TL | Lag in MB |
+----------------------+------------+--------+---------+----+-----------+
| cray-sls-postgres-0 | 10.47.0.35 | Leader | running |  1 |           |
| cray-sls-postgres-1 | 10.36.0.33 |        | running |  1 |         0 |
| cray-sls-postgres-2 | 10.44.0.42 |        | running |  1 |         0 |
+----------------------+------------+--------+---------+----+-----------+
```

Check the leader pod's log output for its status as the leader, such as the following:

```
INFO: no action. i am the leader with the lock
```

For example:

```
--- Logs for services "Leader Pod" cray-sls-postgres-0 ---
  ERROR: get_cluster
  INFO: establishing a new patroni connection to the postgres cluster
  INFO: initialized a new cluster
  INFO: Lock owner: cray-sls-postgres-0; I am cray-sls-postgres-0
  INFO: Lock owner: None; I am cray-sls-postgres-0
  INFO: no action. i am the leader with the lock
  INFO: No PostgreSQL configuration items changed, nothing to reload.
  INFO: postmaster pid=87
  INFO: running post_bootstrap
  INFO: trying to bootstrap a new cluster
```

Errors reported previous to the lock status, such as `ERROR: get_cluster`, can be ignored.

For any issues, see the "About Postgres" section in the *HPE Cray EX System Administration Guide S-8001*.

Border Gateway Protocol (BGP) Health Checks

Verify that all BGP sessions are in an `Established` state for the Mellanox and Aruba spine switches. Refer to "Check BGP Status and Reset Sessions" in the *HPE Cray EX System Administration Guide S-8001* to reset any sessions in an *`Idle`* state.

Determine the IP addresses of the switches:

```
ncn-m001# kubectl get cm config -n metallb-system -o yaml | head -12
apiVersion: v1
data:
  config: |
    peers:
    - peer-address: 10.252.0.2
      peer-asn: 65533
      my-asn: 65533
    - peer-address: 10.252.0.3
      peer-asn: 65533
      my-asn: 65533
    address-pools:
    - name: customer-access
```

Use the first peer-address (10.252.0.2 in this example) to SSH as the administrator to the first switch and note in the returned output if a Mellanox or Aruba switch is indicated. For example:

```
ncn-m001# ssh admin@10.252.0.2
```

- **Mellanox:** Look for `Mellanox Onyx Switch Management` or `Mellanox Switch` after logging in to the switch with SSH.
- **Aruba:** Look for `Please register your products now at: https://asp.arubanetworks.com` after logging in to the switch with SSH.

Mellanox: SSH to a Mellanox spine switch, verify BGP is enabled, and view the status of the BGP sessions.

```
sw-spine-001 [standalone: master] > enable
sw-spine-001 [standalone: master] # show protocols | include bgp
bgp:                    enabled
sw-spine-001 [standalone: master] # show ip bgp summary
VRF name : default
BGP router identifier : 10.252.0.2
local AS number : 65533
BGP table version : 50
Main routing table version: 50
IPV4 Prefixes : 68
IPV6 Prefixes : 0
L2VPN EVPN Prefixes : 0
------------------------------------------------------------------------------------------------------------
----------
```

```
Neighbor V AS MsgRcvd MsgSent TblVer InQ OutQ Up/Down State/
PfxRcd
-----------------------------------------------------------------------------------------------------------
----------
10.252.1.10 4 65533 3144 3564 50 0 0 1:01:50:41
ESTABLISHED/13
10.252.1.11 4 65533 3144 3569 50 0 0 1:01:50:40
ESTABLISHED/14
10.252.1.12 4 65533 3145 3576 50 0 0 1:01:50:41
ESTABLISHED/14
10.252.1.13 4 65533 3144 3568 50 0 0 1:01:50:41
ESTABLISHED/13
10.252.1.14 4 65533 3145 3572 50 0 0 1:01:50:41
ESTABLISHED/14
```

Repeat the above Mellanox commands for the second peer-address (10.252.0.3 in this example).

Aruba: SSH to an Aruba spine switch and view the status of the BGP sessions.

```
sw-spine-001# show bgp ipv4 unicast summary
VRF : default
BGP Summary
-----------
 Local AS               : 65533        BGP Router Identifier  : 10.252.0.4
 Peers                  : 7            Log Neighbor Changes   : No
 Cfg. Hold Time         : 180          Cfg. Keep Alive        : 60
 Confederation Id       : 0
 Neighbor        Remote-AS MsgRcvd MsgSent   Up/Down Time State        AdminStatus
 10.252.0.5      65533       19579   19588   20h:40m:30s  Established   Up
 10.252.1.7      65533       34137   39074   20h:41m:53s  Established   Up
 10.252.1.8      65533       34134   39036   20h:36m:44s  Established   Up
 10.252.1.9      65533       34104   39072   00m:01w:04d  Established   Up
 10.252.1.10     65533       34105   39029   00m:01w:04d  Established   Up
 10.252.1.11     65533       34099   39042   00m:01w:04d  Established   Up
 10.252.1.12     65533       34101   39012   00m:01w:04d  Established   Up
```

Repeat the above Aruba command for the second peer-address (10.252.0.3 in this example).

23.17.2 Network Health Checks

Run network health checks for the KEA, ExternalDNS, Spire, and Vault services.

Verify KEA has Active DHCP Leases

Right after an fresh install of the Cray System Management (CSM) product, it is important to verify that KEA is currently handing out DHCP leases on the system. The following commands can be ran on any of the non-compute node (NCN) master or worker nodes.

Retrieve an API Token:

```
ncn# export TOKEN=$(curl -s -S -d grant_type=client_credentials \
-d client_id=admin-client \
-d client_secret=`kubectl get secrets admin-client-auth \
-o jsonpath='{.data.client-secret}' | base64 -d` \
https://api-gw-service-nmn.local/keycloak/realms/shasta/protocol/openid-connect/token \
| jq -r '.access_token')
```

Retrieve all of the leases currently in KEA:

```
ncn# curl -H "Authorization: Bearer ${TOKEN}" -X POST -H "Content-Type: application/json" \
-d '{ "command": "lease4-get-all",  "service": [ "dhcp4" ] }' \
https://api_gw_service.local/apis/dhcp-kea | jq
```

If there is an non-zero amount of DHCP leases for air-cooled hardware returned, that is a good indication that KEA is working properly.

Verify Ability to Resolve External DNS

Perform the following health checks if unbound is configured to resolve outside hostnames. If unbound is not configured, then this check may be skipped.

Run the following on one of the master or worker nodes (not the PIT):

```
ncn# nslookup cray.com ; echo "Exit code is $?"
Server:         10.92.100.225
Address:        10.92.100.225#53

Non-authoritative answer:
Name:   cray.com
Address: 52.36.131.229

Exit code is 0
```

Verify that the command has the "Exit code is 0" message, reports no errors, and resolves the address in the returned output.

Verify Spire Agent is Running on Kubernetes NCNs

Run the following command on all Kubernetes NCNs (excluding the PIT):

```
ncn# goss -g /opt/cray/tests/install/ncn/tests/goss-spire-agent-service-running.yaml validate
```

Troubleshooting: Run a Kubernetes test to verify the spire-agent is enabled and running.

- The `spire-agent` service may fail to start on Kubernetes NCNs. Logging errors will be reported when running the `journalctl` command:
  - The "join token does not exist or has already been used" error is returned.
  - The last lines of the logs contain multiple lines of "systemd[1]: spire-agent.service: Start request repeated too quickly."

  To resolve the failure, delete the `request-ncn-join-token` daemonset pod running on the node to clear the issue. Log in to the impacted nodes and restart the service while waiting for the `spire-agent` systemctl service on the Kubernetes node to restart cleanly. The following recovery procedure can be run from any Kubernetes node in the cluster:

  1. Set `NODE` to the NCN which is experiencing the issue. In this example, `ncn-w002`.

     ```
     ncn# export NODE=ncn-w002
     ```

  2. Define the following function.

     ```
     ncn# function renewncnjoin() { for pod in $(kubectl get pods -n spire \
     |grep request-ncn-join-token | awk '{print $1}'); \
     do if kubectl describe -n spire pods $pod | grep -q \
     "Node:.*$1"; then echo "Restarting $pod running on $1"; \
     kubectl delete -n spire pod "$pod"; fi done }
     ```

  3. Run the function.

     ```
     ncn# renewncnjoin $NODE
     ```

- The `spire-agent` service may also fail if an NCN was powered off for too long and its tokens are expired. If the service fails, delete `/root/spire/agent_svid.der`, `/root/spire/bundle.der`, and `/root/spire/data/svid.key` off of the NCN before deleting the `request-ncn-join-token` daemonset pod.

Verify the Vault Cluster is Healthy

Run the following command on `ncn-m002`:

```
ncn-m002# goss -g /opt/cray/tests/install/ncn/tests/goss-k8s-vault-cluster-health.yaml validate
```

Check the output to verify no failures are reported:

```
Count: 2, Failed: 0, Skipped: 0
```

23.17.3 Hardware Management Services (HMS) Health Checks

Validate the health of the Hardware Management Services (HMS) components within the Cray System Management (CSM) product with the following health check scripts:

- `/opt/cray/tests/ncn-resources/hms/hms-test/hms_run_ct_smoke_tests_ncn-resources.sh`
- `/opt/cray/tests/ncn-resources/hms/hms-test/hms_run_ct_functional_tests_ncn-resources.sh`

Smoke Tests

The HMS smoke tests consist of bash scripts that check the status of HMS service pods in Kubernetes and verify HTTP status codes returned by the HMS service APIs. The `hms_run_ct_smoke_tests_ncn-resources.sh` wrapper script checks for executable files in the HMS smoke test directory for the non-compute node (NCN) and runs all tests found successively.

```
ncn-m002# /opt/cray/tests/ncn-resources/hms/hms-test/hms_run_ct_smoke_tests_ncn-resources.sh
    searching for HMS CT smoke tests...
    found 11 HMS CT smoke tests...
    running HMS CT smoke tests...
```

A summary of the test results is printed at the bottom of the output.

```
HMS smoke tests ran with 2/11 failures
exiting with status code: 1
```

The tests print the commands being executed while they are running. They also print the command output and status code if failures occur to help with debugging.

The following is an example of a pod status failure:

```
    running '/opt/cray/tests/ncn-smoke/hms/hms-reds/reds_smoke_test_ncn-smoke.sh'...
    Running reds_smoke_test...
    (11:40:33) Running '/opt/cray/tests/ncn-resources/hms/hms-test/hms_check_pod_status_ncn-
resources_remote-resources.sh cray-reds'...
    services         cray-reds-867c65879d-cr4mg                                    1/2
CrashLoopBackOff   266        24h

    Pod status: CrashLoopBackOff
    ERROR: '/opt/cray/tests/ncn-resources/hms/hms-test/hms_check_pod_status_ncn-resources_remote-
resources.sh cray-reds' failed with error code: 1
    FAIL: reds_smoke_test ran with failures
    cleaning up...
    '/opt/cray/tests/ncn-smoke/hms/hms-reds/reds_smoke_test_ncn-smoke.sh' exited with status code: 1
```

The following is an example of an API call failure:

```
    running '/opt/cray/tests/ncn-smoke/hms/hms-capmc/capmc_smoke_test_ncn-smoke.sh'...
    Running capmc_smoke_test...
    (11:40:27) Running '/opt/cray/tests/ncn-resources/hms/hms-test/hms_check_pod_status_ncn-
resources_remote-resources.sh cray-capmc'...
    (11:40:27) Running 'kubectl get secrets admin-client-auth -o jsonpath='{.data.client-secret}''...
    (11:40:27) Running 'curl -k -i -s -S -d grant_type=client_credentials -d client_id=admin-client -d
client_secret=${CLIENT_SECRET} https://api-gw-service-nmn.local/keycloak/realms/shasta/protocol/openid-
connect/token'...
    (11:40:28) Testing 'curl -k -i -s -S -o /tmp/capmc_smoke_test_out-${DATETIME}.${RAND}.curl$
{NUM}.tmp -X POST -d '{}' -H "Authorization: Bearer ${TOKEN}" https://api-gw-service-nmn.local/apis/
capmc/capmc/v1/get_power_cap_capabilities'...
    HTTP/2 503
    ERROR: 'curl -k -i -s -S -o /tmp/capmc_smoke_test_out-${DATETIME}.${RAND}.curl${NUM}.tmp -X POST -d
'{}' -H "Authorization: Bearer ${TOKEN}" https://api-gw-service-nmn.local/apis/capmc/capmc/v1/
```

```
get_power_cap_capabilities' did not return "200" or "204" status code as expected

    MAIN_ERRORS=1
    FAIL: capmc_smoke_test ran with failures
    cleaning up...
    '/opt/cray/tests/ncn-smoke/hms/hms-capmc/capmc_smoke_test_ncn-smoke.sh' exited with status code: 1
```

Functional Tests

The HMS functional tests consist of Tavern-based API tests for HMS services that are written in yaml and executed within *hms-pytest* containers on the NCNs that are spun up using podman. The functional tests are more rigorous than the smoke tests and verify the behavior of HMS service APIs in greater detail. The `hms_run_ct_functional_tests_ncn-resources.sh` wrapper script checks for executable files in the HMS functional test directory for the NCN and runs all tests found successively.

```
ncn-m002# /opt/cray/tests/ncn-resources/hms/hms-test/hms_run_ct_functional_tests_ncn-resources.sh
    searching for HMS CT functional tests...
    found 4 HMS CT functional tests...
    running HMS CT functional tests...
```

A summary of the test results is printed at the bottom of the output:

```
HMS functional tests ran with 1/4 failures
exiting with status code: 1
```

The tests print the commands being executed while running. They also print the command output and status code if failures occur in order to help with debugging.

The following is an example of a *hms-pytest* container spin-up failure, which may occur if the *hms-pytest* image is unavailable or missing from the local image registry on the NCN:

```
    (20:06:04) Running '/usr/bin/hms-pytest --tavern-global-cfg=/opt/cray/tests/ncn-functional/hms/hms-
bss/common.yaml /opt/cray/tests/ncn-functional/hms/hms-bss'...
    Trying to pull registry.local/cray/hms-pytest:1.1.1...
      manifest unknown: manifest unknown
    Error: unable to pull registry.local/cray/hms-pytest:1.1.1: Error initializing source docker://
registry.local/cray/hms-pytest:1.1.1: Error reading manifest 1.1.1 in registry.local/cray/hms-pytest:
manifest unknown: manifest unknown
    FAIL: bss_tavern_api_test ran with failures
    cleaning up...
```

A summary of the test suites executed and their results is printed for each HMS service tested. Period characters represent test cases that passed, and the letter F characters represent test cases that failed within each test suite.

Example *pytest* summary of Tavern test suites executed for a service:

```
    ============================= test session starts ==============================
    platform linux -- Python 3.8.5, pytest-6.1.2, py-1.10.0, pluggy-0.13.1
    rootdir: /opt/cray/tests/ncn-functional/hms/hms-smd, configfile: pytest.ini
    plugins: tap-3.2, tavern-1.12.2
    collected 38 items

    test_smd_component_endpoints_ncn-functional_remote-functional.tavern.yaml . [  2%]
    ......                                                                   [ 18%]
    test_smd_components_ncn-functional_remote-functional.tavern.yaml F.F.... [ 36%]
                                                                             [ 36%]
    test_smd_discovery_status_ncn-functional_remote-functional.tavern.yaml . [ 39%]
                                                                             [ 39%]
    test_smd_groups_ncn-functional_remote-functional.tavern.yaml .           [ 42%]
    test_smd_hardware_ncn-functional_remote-functional.tavern.yaml ....F     [ 55%]
    test_smd_memberships_ncn-functional_remote-functional.tavern.yaml .      [ 57%]
    test_smd_partitions_ncn-functional_remote-functional.tavern.yaml .       [ 60%]
    test_smd_redfish_endpoints_ncn-functional_remote-functional.tavern.yaml . [ 63%]
                                                                             [ 63%]
    test_smd_service_endpoints_ncn-functional_remote-functional.tavern.yaml . [ 65%]
    ...F........                                                             [ 97%]
    test_smd_state_change_notifications_ncn-functional_remote-functional.tavern.yaml . [100%]
```

When API test failures occur, output from Tavern is printed by *pytest* indicating the following:

- The *`Source test stage`* that was executing when the failure occurred, which is a portion of the source code for the failed test case.
- The *`Formatted stage`* that was executing when the failure occurred, which is a portion of the source code for the failed test case with its variables filled in with the values that were set at the time of the failure. This includes the request header, method, URL, and other options of the failed test case that are useful for attempting to reproduce the failure manually using the `curl` command.
- The specific *`Errors`* encountered when processing the API response that caused the failure. This is the first place to look when debugging API test failures.

Example 'Source test stage':

```
    Source test stage (line 179):
      - name: Ensure the boot script service can provide the bootscript for a given node
        request:
          url: "{base_url}/bss/boot/v1/bootscript?nid={nid}"
          method: GET
          headers:
            Authorization: "Bearer {access_token}"
          verify: !bool "{verify}"
        response:
          status_code: 200
```

Example 'Formatted stage':

```
    Formatted stage:
      name: Ensure the boot script service can provide the bootscript for a given node
      request:
        headers:
          Authorization: Bearer
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiS1dUIiwia2lkIiA6ICJXcXFhelBLNnNVSnpUV250bThmYWh3cGVLOGRjeTB4SjFpSmRWRGZaLV8wIn0.eyJqdGk
iOiI4ZGZkNTQ0YS1jNTY5LTQyNmUtYThiYy02NDg4MzgxOWUyOTAiLCJleHAiOjE2NDQ0MzczODEsIm5iZiI6MCwiaWF0IjoxNjEyOTAxMzgxLCJpc3MiOi
JodHRwczovL2FwaS1ndy1zZXJ2aWNlLW5tbi5sb2NhbC9rZXljbG9hay9yZWFsbXMvc2hhc3RhIiwiYXVkIjpbImdhdGVrZWVwZXIiLCJzaGFzdGEiLCJhY
2NvdW50Il0sInN1YiI6IjJjNDFiYjgwLTM2NGEtNGNkOS1hMGZkLTQyYzQ5ODRmMTM2ZSIsInR5cCI6IkJlYXJlciIsImF6cCI6ImFkbWluLWNsaWVudCIs
ImF1dGhfdGltZSI6MCwic2Vzc2lvbl9zdGF0ZSI6IjEzZGQ1YmY2LWQxNGMtNDcwZC05ZWI0LTQ5MDFmYzc3YWYwOSIsImFjciI6IjEiLCJyZWFsbV9hY2N
lc3MiOnsicm9sZXMiOlsib2ZmbGluZV9hY2Nlc3MiLCJ1bWFfYXV0aG9yaXphdGlvbiJdfSwicmVzb3VyY2VfYWNjZXNzIjp7InNoYXN0YSI6eyJyb2xlcy
I6WyJhZG1pbiJdfSwiYWNjb3VudCI6eyJyb2xlcyI6WyJtYW5hZ2UtYWNjb3VudCIsIm1hbmFnZS1hY2NvdW50LWxpbmtzIiwidmlldy1wcm9maWxlIl19f
Swic2NvcGUiOiJwcm9maWxlIGVtYWlsIiwiY2xpZW50SG9zdCI6IjEwLjMyLjAuMSIsImNsaWVudElkIjoiYWRtaW4tY2xpZW50IiwiZW1haWxfdmVyaWZp
ZWQiOmZhbHNlLCJwcmVmZXJyZWRfdXNlcm5hbWUiOiJzZXJ2aWNlLWFjY291bnQtYWRtaW4tY2xpZW50IiwiY2xpZW50QWRkcmVzcyI6IjEwLjMyLjAuMSJ
9.c37RtSOzcM_-6poyPecS7HP_t1hnqURkgjRcXTQj2S0IEZAkMyfeOVYRFr1gDlAFP-BnRPkf_X2_B7d63j9gI15M1gksYcv8PP0bZFX3PAOaBu-
hHGfIw2pDsNsJEA-L72Pb9nmcaPR1CnnVijwRFV-jAmGBJ_vv612mjR5nbI_YJUHDkgdzDfWbpWKQzuCxiJ8USxPD-
ASqx_pLecUzcihorb6PNngMaeisc2TqLTV8YRhSZYeL3cssEcXyTxRBe3zjPDawlPArjY2FUkEdzbtl-
Tq3D2Ulii44esOf4_ooGmUsOc9vrvYvM_JNPAVamv0-0709PRwwNjFl9nEd5g
        method: GET
        url: 'https://api-gw-service-nmn.local/apis/bss/boot/v1/bootscript?nid=None'
        verify: !bool 'False'
      response:
        status_code: 200
```

Example 'Errors':

```
    Errors:
    E   tavern.util.exceptions.TestFailError: Test 'Ensure the boot script service can provide the bootscript for a
given node' failed:
        - Status code was 400, expected 200:
            {"type": "about:blank", "title": "Bad Request", "detail": "Need a mac=, name=, or nid= parameter",
"status": 400}
```

23.17.4 Cray Management Services (CMS) Health Checks

In general, the Cray Management Services (CMS) tests contain up to three types of checks:

- Verifying that the service looks as expected in Kubernetes.
- Verifying that non-destructive API and CLI calls work as expected.

  In this case, non-destructive means they do not change the system state (nothing is created, deleted, or changed). By default, these checks only look at the return code (CLI) or status code (API) when determining if the call worked. Some tests do further verification of the response.
- Basic operations involving the service work as expected.

In some cases, these may involve creating things, but they are cleaned up before the test exits.

All tests include the Kubernetes health checks, most include the API and CLI calls, and a few include additional operational testing.

Usage

```
cmsdev test [-q | -v] <shortcut>
```

- The shortcut determines which component will be tested.
- The tool logs to `/opt/cray/tests/cmsdev.log`.
- The `-q` (quiet) and `-v` (verbose) flags can be used to decrease or increase the amount of information sent to the screen. The same amount of data is written to the log file in either case.

Interpreting Results

If the health tests pass, no further interpretation is required. In the case of failure, the underlying problem can be a variety of things. The tests indicate what they are attempting to do and what failed. Careful examination of the test log (or the test output, if run in verbose mode) should give insight into what exactly went wrong. This can be easier with a high level idea of what each test is doing.

The following sections give a brief description of each test.

Boot Orchestration Service (BOS):

- Verify the expected pods are seen in Kubernetes in the expected states.
- Make a call to every non-destructive BOS API endpoint and CLI command with the following exceptions:
  - A GET/describe is called on an individual BOS session or session template only if one already exists.

Configuration Framework Service (CFS):

- Verify the expected pods are seen in Kubernetes in the expected states.
- Make a call to every non-destructive CFS API endpoint and CLI command with the following exceptions:
  - A GET/describe is called on an individual CFS component, configuration, option, or session only if one already exists.

ConMan Console Manager:

- Verify the expected pods are seen in Kubernetes in the expected states.

  This check may fail because a container is in *`CrashLoopBackOff`*. If the system has no nodes configured, this is expected behavior because ConMan will have no consoles to manage.
- Verify the expected PVCs are seen in Kubernetes in the expected states.

Compute Rolling Upgrade Service (CRUS):

The CRUS service requires both munge and Slurm to be installed and configured. If they are not, the test skips some of its checks (as noted below).

- Verify the expected pods are seen in Kubernetes in the expected states.

  If munge and Slurm are present, the expected state is *`Succeeded`*. Otherwise, *`Pending`* or *`Running`* are also permitted.
- Make a call to every non-destructive CRUS API endpoint and CLI command with the following exceptions:

- These calls are only performed if munge and Slurm are present.
- A GET/describe is called on an individual CRUS session only if one already exists.

Image Management Service (IMS):

- Verify the expected pods are seen in Kubernetes in the expected states.
- Verify the expected PVCs are seen in Kubernetes in the expected states.
- Verify that the `ims` artifact bucket exists in S3.
- Make a call to every non-destructive CRUS API endpoint and CLI command with the following exceptions:
  - No GET/describes are made to specific IMS objects.
  - The responses to the GET/list requests are additionally validated to make sure the objects they return are in the expected formats.

iPXE and Trivial File Transfer Protocol (TFTP):

The iPXE and TFTP service tests are combined because they are typically used together. The iPXE image is generated by the iPXE service, and that image is retrieved using the TFTP service.

- Verify the expected pods are seen in Kubernetes in the expected states.
- Verify the expected PVCs are seen in Kubernetes in the expected states.
- Verify that the iPXE image exists and can be retrieved using TFTP.

  This is verified for both the cray-tftp and cray-tftp-hmn services. For each of those services, this is verified for both the cluster IP and external IP. A checksum is done to verify that the file transferred without errors.

Version Control Service (VCS):

- Verify the expected pods are seen in Kubernetes in the expected states.
- Verify the expected PVCs are seen in Kubernetes in the expected states.
- Verify there is a basic VCS workflow that attempts the following:
  - Create an organization in VCS using the API.
  - Retrieve that organization in VCS using the API.
  - Create a repository in that organization in VCS using the API.
  - Retrieve that repository in VCS using the API.
  - Clone that repository to the system running the test.
  - Copy a file into the repository.
  - Git add, commit, and push to VCS.
  - Delete the local copy of the repository.
  - Clone it again.
  - Verify that the previously-copied file exists and is unchanged.
  - Delete the local copy of the repository.
  - Delete the repository in VCS using the API.
  - Attempt to retrieve the repository in VCS using the API and verify that it now fails.
  - Attempt to clone the repository and verify that it now fails.

- Delete the organization in VCS using the API.
- Attempt to retrieve the organization in VCS using the API and verify that it now fails.

23.17.5 Automated Goss Testing Health Checks

Multiple Goss test suites are available on the GitHub Goss page:

- *https://github.com/aelsabbahy/goss*

These tests cover a variety of sub-systems. The two that may be run on the system are the general NCN test suite and the kubernetes test suite.

The general NCN test suite is run using the following command.

```
ncn# /opt/cray/tests/install/ncn/automated/ncn-run-time-checks
```

The Kubernetes test suite is run using the following command.

```
pit# /opt/cray/tests/install/ncn/automated/ncn-kubernetes-checks
```

Known Goss Test Issues

- Tests are only reliably run from the PIT node at this time.
- K8S Test: Kubernetes Query BSS Cloud-init for ca-certs
  - May fail immediately after platform install. Should pass after the TrustedCerts Operator has updated BSS (Global cloud-init meta) with CA certificates.
- K8S Test: Kubernetes Velero No Failed Backups
  - Due to a known issue with Velero, a backup may be attempted immediately upon the deployment of a backup schedule (for example, vault). It may be necessary to use the **`velero`** command to delete backups from a Kubernetes node to clear this situation.

23.17.6 Boot CSM Barebones Image Health Check

A pre-built node image is included with the Cray System Management (CSM) release. This Boot CSM Barebones Image can be used to validate that core CSM services are available and responding as expected. The CSM barebones image contains only the minimal set of RPMs and configuration required to boot an image and is not suitable for production use. To run production workloads, use an image from the Cray OS (COS) product, or a similar product.

Limitations

- The CSM Barebones image included with the EX System 1.4 release will not successfully complete beyond the dracut stage of the boot process. If the dracut stage is reached, the boot can be considered successful and shows the necessary CSM services needed to boot a node are up and available.
- In addition to the CSM Barebones image, the EX System 1.4 release also includes an IMS Recipe that can be used to build the CSM Barebones image. However, the CSM Barebones recipe currently requires RPMs that are not installed with the CSM product. The CSM Barebones recipe is built after the Cray OS (COS) product stream is installed on the system.

- The CLI must be used in order to complete these tasks. For additional details, see the "Initialize and Authorize the CLI" section.

Locate the CSM Barebones Image in IMS

Locate the CSM Barebones image and note the path to the image's manifest.json in S3.

```
ncn# cray ims images list --format json | jq '.[] \
| select(.name | contains("barebones"))'
{
  "created": "2021-01-14T03:15:55.146962+00:00",
  "id": "293b1e9c-2bc4-4225-b235-147d1d611eef",
  "link": {
    "etag": "6d04c3a4546888ee740d7149eaecea68",
    "path": "s3://boot-images/293b1e9c-2bc4-4225-b235-147d1d611eef/manifest.json",
    "type": "s3"
  },
  "name": "cray-shasta-csm-sles15sp1-barebones.x86_64-shasta-1.4"
}
```

Create a BOS Session Template for the CSM Barebones Image

The session template below can be copied and used as the basis for the BOS session template. As noted below, the S3 path for the manifest must match the S3 path shown in IMS.

Create a JSON file for the BOS session template:

```
ncn# vi sessiontemplate.json
 {
     "boot_sets": {
       "compute": {
         "boot_ordinal": 2,
         "etag": "6d04c3a4546888ee740d7149eaecea68",// <== Set to etag of the IMS Image
         "kernel_parameters": "console=ttyS0,115200 bad_page=panic crashkernel=340M hugepagelist=2m-2g
intel_iommu=off intel_pstate=disable iommu=pt ip=dhcp numa_interleave_omit=headless
numa_zonelist_order=node oops=panic pageblock_order=14 pcie_ports=native printk.synchronous=y
rd.neednet=1 rd.retry=10 rd.shell turbo_boost_limit=999 spire_join_token=${SPIRE_JOIN_TOKEN}",
         "network": "nmn",
         "node_roles_groups": [
           "Compute"
         ],
         "path": "s3://boot-images/293b1e9c-2bc4-4225-b235-147d1d611eef/manifest.json",// <==Path must
match the IMS Image Path
         "rootfs_provider": "cpss3",
         "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:nmn0",
         "type": "s3"
       }
     },
     "cfs": {
       "configuration": "cos-integ-config-1.4.0"
     },
     "enable_cfs": false,
     "name": "shasta-1.4-csm-bare-bones-image"
   }
```

Create the BOS session template using the file as the input

```
ncn# cray bos sessiontemplate create --file \
sessiontemplate.json --name shasta-1.4-csm-bare-bones-image
/sessionTemplate/shasta-1.4-csm-bare-bones-image
```

Find an Available Compute Node

```
ncn-m002# cray hsm state components list --role Compute --enabled true
...
[[Components]]
ID = "x3000c0s17b1n0"
Type = "Node"
State = "On"
```

```
Flag = "OK"
Enabled = true
Role = "Compute"
NID = 1
NetType = "Sling"
Arch = "X86"
Class = "River"

[[Components]]
ID = "x3000c0s17b2n0"
Type = "Node"
State = "On"
Flag = "OK"
Enabled = true
Role = "Compute"
NID = 2
NetType = "Sling"
Arch = "X86"
Class = "River"
```

Select a node and set the *XNAME* value to its ID. In the following example, x3000c0s17b2n0 is the ID.

```
ncn# export XNAME=x3000c0s17b2n0
```

Reboot the Node with a BOS Session Template

Create a BOS session to reboot the chosen node using the BOS session template that was created:

```
ncn# cray bos session create --template-uuid shasta-1.4-csm-bare-bones-image \
--operation reboot --limit $XNAME
limit = "x3000c0s17b2n0"
operation = "reboot"
templateUuid = "shasta-1.4-csm-bare-bones-image"
[[links]]
href = "/v1/session/8f2fc013-7817-4fe2-8e6f-c2136a5e3bd1"
jobId = "boa-8f2fc013-7817-4fe2-8e6f-c2136a5e3bd1"
rel = "session"
type = "GET"

[[links]]
href = "/v1/session/8f2fc013-7817-4fe2-8e6f-c2136a5e3bd1/status"
rel = "status"
type = "GET"
```

Verify Console Connections

Sometimes the compute nodes and uan are not up yet when cray-conman is initialized and will not be monitored yet. This is a good time to verify that all nodes are being monitored for console logging or re-initialze cray-conman if needed.

1. Identify the cray-conman pod.

   ```
   ncn# kubectl get pods -n services | grep "^cray-conman-"
   cray-conman-b69748645-qtfxj      3/3     Running            0          16m
   ```

2. Set the *PODNAME* value.

   ```
   ncn# export PODNAME=cray-conman-b69748645-qtfxj
   ```

3. Log into the cray-conman container in the pod you identified in the previous step.

   ```
   ncn# kubectl exec -n services -it $PODNAME -c cray-conman -- bash
   cray-conman#
   ```

4. Check the existing connections.

   ```
   cray-conman# conman -q
   x9000c0s1b0n0
   x9000c0s20b0n0
   x9000c0s22b0n0
   ```

```
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

5. If the compute nodes and UANs are not included in the list of nodes being monitored, the conman process can be re-initialized by killing the conmand process.

   a. Identify the conmand process.

   ```
   cray-conman# ps -ax | grep conmand | grep -v grep
   13 ?        Sl     0:45 conmand -F -v -c /etc/conman.conf
   ```

   b. Set *CONPID* to the process ID from the previous command output.

   ```
   cray-conman# export CONPID=13
   ```

   c. Kill the process.

   ```
   cray-conman# kill $CONPID
   ```

These steps will regenerate the ConMan configuration file and restart the conmand process. Repeat the previous steps to verify that it now includes all nodes that are included in state manager.

Connect to the Node's Console and Observe the Boot

Run conman from inside the conman pod to access the console. The boot will fail, but should reach the dracut stage. If the dracut stage is reached, the boot can be considered successful and shows that the necessary CSM services needed to boot a node are up and available.

```
cray-conman# conman -j x9000c1s7b0n1
...
[    7.876909] dracut: FATAL: Don't know how to handle 'root=craycps-s3:s3://boot-images/e3ba09d7-
e3c2-4b80-9d86-0ee2c48c2214/rootfs:c77c0097bb6d488a5d1e4a2503969ac0-27:dvs:api-gw-service-nmn.local:
300:nmn0'
[    7.898169] dracut: Refusing to continue
[    7.952291] systemd-shutdow: 13 output lines suppressed due to ratelimiting
[    7.959842] systemd-shutdown[1]: Sending SIGTERM to remaining processes...
[    7.975211] systemd-journald[1022]: Received SIGTERM from PID 1 (systemd-shutdow).
[    7.982625] systemd-shutdown[1]: Sending SIGKILL to remaining processes...
[    7.999281] systemd-shutdown[1]: Unmounting file systems.
[    8.006767] systemd-shutdown[1]: Remounting '/' read-only with options ''.
[    8.013552] systemd-shutdown[1]: Remounting '/' read-only with options ''.
[    8.019715] systemd-shutdown[1]: All filesystems unmounted.
[    8.024697] systemd-shutdown[1]: Deactivating swaps.
[    8.029496] systemd-shutdown[1]: All swaps deactivated.
[    8.036504] systemd-shutdown[1]: Detaching loop devices.
[    8.043612] systemd-shutdown[1]: All loop devices detached.
[    8.059239] reboot: System halted
```

23.17.7 UAS and UAI Health Checks

Initialize and Authorize the CLI

The procedures below use the CLI as an authorized user and run on two separate node types. The first part runs on the LiveCD node while the second part runs on a non-LiveCD kubernetes master or worker node. When using the CLI on either node, the CLI configuration must be initialized and the user running the procedure must be authorized. This section describes how to initialize the CLI for use by a user and authorize the CLI as a user to run the procedures on any given node. The procedures will need to be repeated in both stages of the validation procedure.

Discontinue Use of the CRAY_CREDENTIALS Service Account Token

Installation procedures leading up to production mode on Shasta use the CLI with a Kubernetes managed service account normally used for internal operations. There is a procedure for extracting the OAUTH token for this service account and assigning it to the **`CRAY_CREDENTIALS`** environment variable to permit simple CLI operations. The UAS / UAI validation procedure runs as a post-installation procedure and requires an actual user with Linux credentials, not this service account. Prior to running any of the steps below you must unset the **`CRAY_CREDENTIALS`** environment variable.

```
ncn-m002# unset CRAY_CREDENTIALS
```

Initialize the CLI Configuration

The CLI needs to know what host to use to obtain authorization and what user is requesting authorization so it can obtain an OAUTH token to talk to the API Gateway. This is accomplished by initializing the CLI configuration. This example uses the **`vers`** username. In practice, **`vers`** and the response to the **`password:`** prompt should be replaced with the username and password of the administrator running the validation procedure.

To check whether the CLI needs initialization, run the following command.

```
ncn-m002# cray config describe
```

If the output appears as follows, the CLI requires initialization.

```
Usage: cray config describe [OPTIONS]

Error: No configuration exists. Run `cray init`
```

If the output appears more like the following, then the CLI is initialized and logged in as **`vers`**. If that is the incorrect username, authorize the correct username and password in the next section. If **`vers`** is the correct user, proceed to the validation procedure on that node.

If the CLI must be initialized again, use the following command and include the correct username, password, and the password response.

```
ncn-m002# cray init
Cray Hostname: api-gw-service-nmn.local
Username: vers
Password:
Success!

Initialization complete.
```

Authorize the Correct CLI User

If the CLI is initialized but authorized for a user different, run the following command and substitute the correct username and password.

```
ncn-m002# cray auth login
Username: vers
Password:
Success!
```

Troubleshoot CLI Initialization or Authorization Issues

If initialization or authorization fails in any of the preceding steps, there are several common causes.

- DNS failure looking up api-gw-service-nmn.local may be preventing the CLI from reaching the API Gateway and Keycloak for authorization
- Network connectivity issues with the NMN may be preventing the CLI from reaching the API Gateway and Keycloak for authorization
- Certificate mismatch or trust issues may be preventing a secure connection to the API Gateway
- Istio failures may be preventing traffic from reaching Keycloak
- Keycloak may not yet be set up to authorize you as a user

While resolving these issues is beyond the scope of this section, adding **`-vvvvv`** to the **`cray auth`** or **`cray init`** commands may offer clues as to why the initialization or authorization is failing.

Validate the Basic UAS Installation

This procedure and the following procedures run on separate nodes on the system and validate the basic UAS installation. Ensure this runs on the LiveCD node and that the CLI is authorized for the user.

```
ncn-m002# cray uas mgr-info list
service_name = "cray-uas-mgr"
version = "1.11.5"

ncn-m001-pit# cray uas list
results = []
```

This shows that UAS is installed and running version 1.11.5 and that no UAIs are running. If another user has been using the UAS, it is possible to see UAIs in the list. That is acceptable from a validation standpoint.

To verify that the pre-made UAI images are registered with UAS, run the following command.

```
ncn-m002# cray uas images list
default_image = "dtr.dev.cray.com/cray/cray-uai-sles15sp1:latest"
image_list = [ "dtr.dev.cray.com/cray/cray-uai-sles15sp1:latest",]
```

The output shows that the pre-made end-user UAI image, `cray/cray-uai-sles15sp1:latest`, is registered with UAS. This does not necessarily mean this image is installed in the container image registry, but it is configured for use. If other UAI images have been created and registered, they may also appear in the output.

Validate UAI Creation

This procedure:

- must run on a master or worker node (and not ncn-w001)
- must run on the Shasta system (or from an external host, but the procedure for that is not covered here)
- requires that the CLI be initialized and authorized as for the current user.

To verify that the user account can create a UAI, use the following command.

```
ncn-w003# cray uas create --publickey ~/.ssh/id_rsa.pub
uai_connect_string = "ssh vers@10.16.234.10"
uai_host = "ncn-w001"
uai_img = "registry.local/cray/cray-uai-sles15sp1:latest"
uai_ip = "10.16.234.10"
uai_msg = ""
uai_name = "uai-vers-a00fb46b"
uai_status = "Pending"
```

```
username = "vers"

[uai_portmap]
```

The UAI is now created and in the process of initializing and running.

The following can be repeated as needed to view the UAIs state. If the results appear like the following, the UAI is ready for use.

```
ncn-w003# cray uas list
[[results]]
uai_age = "0m"
uai_connect_string = "ssh vers@10.16.234.10"
uai_host = "ncn-w001"
uai_img = "registry.local/cray/cray-uai-sles15sp1:latest"
uai_ip = "10.16.234.10"
uai_msg = ""
uai_name = "uai-vers-a00fb46b"
uai_status = "Running: Ready"
username = "vers"
```

Log into the UAI (without a password) as follows:

```
ncn-w003# ssh vers@10.16.234.10
The authenticity of host '10.16.234.10 (10.16.234.10)' can't be established.
ECDSA key fingerprint is SHA256:BifA2Axg5O0Q9wqESkLqK4z/b9e1usiDUZ/puGIFiyk.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.16.234.10' (ECDSA) to the list of known hosts.
vers@uai-vers-a00fb46b-6889b666db-4dfvn:~> ps -afe
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 18:51 ?        00:00:00 /bin/bash /usr/bin/uai-ssh.sh
munge         36       1  0 18:51 ?        00:00:00 /usr/sbin/munged
root          54       1  0 18:51 ?        00:00:00 su vers -c /usr/sbin/sshd -e -
f /etc/uas/ssh/sshd_config -D
vers          55      54  0 18:51 ?        00:00:00 /usr/sbin/sshd -e -f /etc/uas/
ssh/sshd_config -D
vers          62      55  0 18:51 ?        00:00:00 sshd: vers [priv]
vers          67      62  0 18:51 ?        00:00:00 sshd: vers@pts/0
vers          68      67  0 18:51 pts/0    00:00:00 -bash
vers         120      68  0 18:52 pts/0    00:00:00 ps -afe
vers@uai-vers-a00fb46b-6889b666db-4dfvn:~> exit
logout
Connection to 10.16.234.10 closed.
```

Clean up the UAI and note that the UAI name used is the same as the name in the output from `cray uas create` above.

```
ncn-w003# cray uas delete --uai-list uai-vers-a00fb46b
results = [ "Successfully deleted uai-vers-a00fb46b",]
```

Troubleshoot UAS and UAI Operations Issues

Authorization Issues

If the user is not logged in as a valid Keycloak user or is inadvertently using the `CRAY_CREDENTIALS` environment variable (i.e. the variable is set if the user is logged in with the their username or another username), the output of running the **`cray uas list`** command will produce output like the following.

```
ncn-w003# cray uas list
Usage: cray uas list [OPTIONS]
Try 'cray uas list --help' for help.

Error: Bad Request: Token not valid for UAS. Attributes missing: ['gidNumber',
'loginShell', 'homeDirectory', 'uidNumber', 'name']
```

Fix this by logging in as a "real user" (a user with Linux credentials) and ensure that **CRAY_CREDENTIALS** is unset.

UAS Cannot Access Keycloak

If the output of the **cray uas list** command appears similar to the following, the wrong hostname to reach the API gateway may be in use. In that case, run the CLI initialization steps again.

```
ncn-w003# cray uas list
Usage: cray uas list [OPTIONS]
Try 'cray uas list --help' for help.

Error: Internal Server Error: An error was encountered while accessing Keycloak
```

There also may be a problem with the Istio service mesh inside of the Shasta system. Troubleshooting this is beyond the scope of this section, but viewing the UAS pod logs in Kubernetes may provide useful information.

There are typically two UAS pods. View logs from both pods to identify the specific failure. The logs have a very large number of **GET** events listed as part of the aliveness checking. The following shows an example of viewing UAS logs (the example shows only one UAS manage, normally there would be two).

```
ncn-w003# kubectl get po -n services | grep uas-mgr | grep -v etcd
cray-uas-mgr-6bbd584ccb-zg8vx            2/2     Running            0          12d
ncn-w003# kubectl logs -n services cray-uas-mgr-6bbd584ccb-zg8vx cray-uas-mgr |
grep -v 'GET ' | tail -25
2021-02-08 15:32:41,211 - uas_mgr - INFO - getting deployment uai-vers-87a0ff6e in
namespace user
2021-02-08 15:32:41,225 - uas_mgr - INFO - creating deployment uai-vers-87a0ff6e
in namespace user
2021-02-08 15:32:41,241 - uas_mgr - INFO - creating the UAI service uai-
vers-87a0ff6e-ssh
2021-02-08 15:32:41,241 - uas_mgr - INFO - getting service uai-vers-87a0ff6e-ssh
in namespace user
2021-02-08 15:32:41,252 - uas_mgr - INFO - creating service uai-vers-87a0ff6e-ssh
in namespace user
2021-02-08 15:32:41,267 - uas_mgr - INFO - getting pod info uai-vers-87a0ff6e
2021-02-08 15:32:41,360 - uas_mgr - INFO - No start time provided from pod
2021-02-08 15:32:41,361 - uas_mgr - INFO - getting service info for uai-
vers-87a0ff6e-ssh in namespace user
127.0.0.1 - - [08/Feb/2021 15:32:41] "POST /v1/uas?imagename=registry.local%2Fcray
%2Fno-image-registered%3Alatest HTTP/1.1" 200 -
2021-02-08 15:32:54,455 - uas_auth - INFO - UasAuth lookup complete for user vers
2021-02-08 15:32:54,455 - uas_mgr - INFO - UAS request for: vers
2021-02-08 15:32:54,455 - uas_mgr - INFO - listing deployments matching: host
None, labels uas=managed,user=vers
2021-02-08 15:32:54,484 - uas_mgr - INFO - getting pod info uai-vers-87a0ff6e
2021-02-08 15:32:54,596 - uas_mgr - INFO - getting service info for uai-
vers-87a0ff6e-ssh in namespace user
2021-02-08 15:40:25,053 - uas_auth - INFO - UasAuth lookup complete for user vers
2021-02-08 15:40:25,054 - uas_mgr - INFO - UAS request for: vers
2021-02-08 15:40:25,054 - uas_mgr - INFO - listing deployments matching: host
None, labels uas=managed,user=vers
```

```
2021-02-08 15:40:25,085 - uas_mgr - INFO - getting pod info uai-vers-87a0ff6e
2021-02-08 15:40:25,212 - uas_mgr - INFO - getting service info for uai-
vers-87a0ff6e-ssh in namespace user
2021-02-08 15:40:51,210 - uas_auth - INFO - UasAuth lookup complete for user vers
2021-02-08 15:40:51,210 - uas_mgr - INFO - UAS request for: vers
2021-02-08 15:40:51,210 - uas_mgr - INFO - listing deployments matching: host
None, labels uas=managed,user=vers
2021-02-08 15:40:51,261 - uas_mgr - INFO - deleting service uai-vers-87a0ff6e-ssh
in namespace user
2021-02-08 15:40:51,291 - uas_mgr - INFO - delete deployment uai-vers-87a0ff6e in
namespace user
127.0.0.1 - - [08/Feb/2021 15:40:51] "DELETE /v1/uas?uai_list=uai-vers-87a0ff6e
HTTP/1.1" 200 -
```

UAI Images not in Registry

If output is similar to the following, the pre-made end-user UAI image is not in the user's local registry (or whatever registry it is being pulled from, see the **`uai_img`** value for details). Locate and the image and push / import it to the registry.

```
ncn-w003# cray uas list
[[results]]
uai_age = "0m"
uai_connect_string = "ssh vers@10.103.13.172"
uai_host = "ncn-w001"
uai_img = "registry.local/cray/cray-uai-sles15sp1:latest"
uai_ip = "10.103.13.172"
uai_msg = "ErrImagePull"
uai_name = "uai-vers-87a0ff6e"
uai_status = "Waiting"
username = "vers"
```

Missing Volumes and Other Container Startup Issues

Various packages install volumes in the UAS configuration. All of those volumes must also have the underlying resources available, sometimes on the host node where the UAI is running and sometimes from within Kubernetes. If the UAI gets stuck with a **`ContainerCreating uai_msg`** field for an extended time, this is a likely cause. UAIs run in the **`user`** Kubernetes namespace and are pods that can be examined using **`kubectl describe`**.

Run the following command to locate the pod.

```
ncn-w003# kubectl get po -n user | grep <uai-name>
```

Run the following command to investigate the problem.

```
ncn-w003# kubectl describe -n user <pod-name>
```

If volumes are missing, they will be in the **`Events:`** section of the output. Other problems may show up there as well. The names of the missing volumes or other issues should indicate what needs to be fixed to enable the UAI.

23.18 Set Default Passwords During NCN Image Customization

Prerequisites

The LiveCD is set up for re-squashing SquashFS images.

About this task

ROLE	System installer
OBJECTIVE	Use non-compute node (NCN) image customization to update the passwords for images. Repeat this process for the other NCN SquashFS file systems. This procedure uses Kubernetes as an example.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Open the image.

   In this example, the image name is `k8s-filesystem.squashfs`.

   ```
   pit# cd /var/www/ephemeral/data/k8s
   pit# unsquash k8s-filesystem.squashfs
   ```

2. Change to the image root.

   ```
   pit# chroot ./squashfs-root
   ```

3. Change the password.

   ```
   chroot-pit# passwd
   ```

4. Replace the SSH keys.

   Replace the default root public and private SSH keys with custom ones, or generate a new pair with `ssh-keygen(1)`.

   ```
   chroot-pit# cd root
   ```

5. Create a new SquashFS artifact.

   ```
   chroot-pit# /srv/cray/scripts/common/create-kis-artifacts.sh
   ```

6. Exit the chroot.

   ```
   chroot-pit# exit
   ```

7. Clean up the SquashFS creation.

   ```
   pit# umount /var/www/ephemeral/data/k8s/squasfs-root/squashfs
   ```

8. Repeat the preceding steps for the other image types being modified.

9. Set the boot links.

```
pit# set-sqfs-links.sh
```

The images will now have the new passwords for the next boot.

23.19 Verify CEPH CSI

About this task

The following procedure details how to verify CEPH CSI and troubleshoot if any components are missing.

Procedure

1. Verify all post ceph install tasks have run.

   Log in to ncn-s001 and check /etc/cray/ceph for completed task files.

```
ncn-s001# ls /etc/cray/ceph/
ceph_k8s_initialized  csi_initialized  installed  kubernetes_nodes.txt  tuned
```

2. Check to see if k8s ceph-csi prerequisites have been created.

   This can be run from any storage, master, or worker node.

```
pit# kubectl get cm
NAME               DATA   AGE
ceph-csi-config    1      3h50m
cephfs-csi-sc      1      3h50m
kube-csi-sc        1      3h50m
sma-csi-sc         1      3h50m
sts-rados-config   1      4h

pit# kubectl get secrets | grep csi
csi-cephfs-secret             Opaque                                4      3h51m
csi-kube-secret               Opaque                                2      3h51m
csi-sma-secret                Opaque                                2      3h51m
```

   Check the results of the system with the examples above.

TROUBLESHOOTING

3. If any components are missing, rerun the storage node cloud-init script.

   Log in to ncn-s001 and run the following example.

```
ncn-s001# ceph -s
```

   If it returns a connection error, then assume that ceph is not installed. Rerun the cloud-init script.

```
ncn-s001# ls /etc/cray/ceph
```

If any files are present, they will represent completed stages.

If the system has a running cluster, edit `/srv/cray/scripts/common/storage-ceph-cloudinit.sh` on ncn-s001 and comment out the following section:

```
#if [ -f "$ceph_installed_file" ]; then
#  echo "This ceph cluster has been initialized"
#else
#  echo "Installing ceph"
#  init
#  mark_initialized $ceph_installed_file
#fi
```

4. Run the `storage-ceph-cloudinit.sh` script.

```
ncn-s001# /srv/cray/scripts/common/storage-ceph-cloudinit.sh
       Configuring node auditing software
       Using generic auditing configuration
       This ceph cluster has been initialized
       This ceph cluster has already been tuned
       This ceph radosgw config and initial k8s integration already complete
       ceph-csi configuration has been already been completed
```

23.20 Manage Firmware Updates with FAS

The Firmware Action Service (FAS) communicates with many devices on the system. FAS can be used to update the firmware for all of the devices it communicates with at once, or specific devices can be targeted for a firmware update. Both methods are describe below. FAS also provides the ability to view the status of a firmware update or abort an update that has already started.

Update Firmware

To perform a firmware update, the `--command-overrideDryrun` parameter needs to be set to *`true`*.

> **IMPORTANT:** When the BMC is updated or rebooted after updating the Node0.BIOS and/or Node1.BIOS liquid-cooled nodes, the node BIOS version will not report the new version string until the nodes are powered back on. It is recommended that the Node0/1 BIOS be updated in a separate action, either before or after a BMC update and the nodes are powered back on after a BIOS update. The liquid-cooled nodes must be powered off for the BIOS to be updated.

The following steps can be used to do a firmware update on all hardware that communicates with FAS:

1. Create a JSON file for the FAS job.

   Ensure the *`overrideDryrun`* parameter is set to *`true`* in the file.

```
{
  "command": {
     "version": "latest",
     "tag": "default",
     "overrideDryrun": true,
     "restoreNotPossibleOverride": true,
     "timeLimit": 1000,
     "description": "full system update 20200623_0"
   }
}
```

2. Update the firmware on all devices that communicate with FAS:

```
ncn-w001# cray fas actions create CUSTOM_DEVICE_PARAMETERS.json
```

To perform an update for specific devices, the following steps can be used:

1. Create a JSON file with the specific device information to target when doing a firmware update.

   The command-line interface (CLI) does not support nested arrays at this time, so more complicated firmware updates require a custom JSON file. Ensure the *overrideDryrun* parameter is set to *true* in the file.

```
{
"stateComponentFilter": {
    "xnames": [
      "x9000c1s3b1"
    ]
  },
"inventoryHardwareFilter": {
    "manufacturer": "cray"
    },
"targetFilter": {
    "targets": [
      "Node1.BIOS",
      "Node0.BIOS"
    ]
  },
"command": {
    "version": "latest",
    "tag": "default",
    "overrideDryrun": true,
    "restoreNotPossibleOverride": true,
    "timeLimit": 1000,
    "description": "dryrun upgrade of x9000c1s3b1 Nodex.BIOS to WNC 1.1.2"
  }
}
```

2. Run a firmware update on the targeted devices.

```
ncn-w001# cray fas actions create CUSTOM_DEVICE_PARAMETERS.json
```

Abort a Firmware Update

Firmware updates can be stopped if required. This is useful given only one action can be run at a time. This is to protect hardware from multiple actions trying to modify it at the same time.

> **IMPORTANT:** If a Redfish update is already in progress, the abort will not stop that process on the device. It is likely the device will update. If the device needs to be manually power cycled (`needManualReboot`), it is possible that the device will `update`, but not actually apply the update until its next reboot. Admins must verify the state of the system after an abort. Only perform an abort if truly necessary. The best way to check the state of the system is to do a snapshot or do a dry-run of an update.

```
ncn-w001# cray fas actions instance delete actionID
```

View the Details of a FAS Action

There are several ways to get more information about a firmware update. An `actionID` and `operationID` are generated when an update or dry-run is created. These values can be used to learn more about what is happening on the system during an update.

To view counts of operations, what state they are in, the overall state of the action, and what parameters were used to create the action:

```
ncn-w001# cray fas actions list
[[actions]]
```

```
blockedBy = []
state = "completed"
actionID = "0a305f36-6d89-4cf8-b4a1-b9f199afaf3b"
startTime = "2020-06-23 15:43:42.939100799 +0000 UTC"
snapshotID = "00000000-0000-0000-0000-000000000000"
endTime = "2020-06-23 15:48:59.586748151 +0000 UTC"

[actions.command]
description = "upgrade of x9000c1s3b1 Nodex.BIOS to WNC 1.1.2"
tag = "default"
restoreNotPossibleOverride = true
timeLimit = 1000
version = "latest"
overrideDryrun = false
[actions.operationCounts]
noOperation = 0
succeeded = 2
verifying = 0
unknown = 0
configured = 0
initial = 0
failed = 0
noSolution = 0
aborted = 0
needsVerified = 0
total = 2
inProgress = 0
blocked = 0
[[actions]]
blockedBy = []
state = "completed"
actionID = "0b9300d6-8f06-4019-a8fa-7b3ff65e5aa8"
startTime = "2020-06-18 03:06:25.694573366 +0000 UTC"
snapshotID = "00000000-0000-0000-0000-000000000000"
endTime = "2020-06-18 03:11:06.806297546 +0000 UTC"

[actions.command]
description = "reissue upgrade of all cray nodeBMC BMCs to v1.3.8"
tag = "default"
restoreNotPossibleOverride = true
timeLimit = 1000
version = "latest"
overrideDryrun = false
[actions.operationCounts]
noOperation = 7
succeeded = 1
verifying = 0
unknown = 0
configured = 0
initial = 0
failed = 0
noSolution = 0
aborted = 0
needsVerified = 0
total = 8
inProgress = 0
blocked = 0
[[actions]]
blockedBy = []
state = "completed"
actionID = "21ff1da0-9520-4d11-92b9-10f8cff088a2"
startTime = "2020-06-23 17:00:51.256743994 +0000 UTC"
snapshotID = "00000000-0000-0000-0000-000000000000"
endTime = "2020-06-23 17:01:17.740695065 +0000 UTC"

...
```

To view what happened at the FAS action level:

```
ncn-w001# cray fas actions describe actionID --format json
{
  "parameters": {
    "stateComponentFilter": {
      "xnames": [
        "x9000c1s3b1"
```

```
      ]
    },
    "command": {
      "description": "restore snapshot nodebmc",
      "tag": "",
      "restoreNotPossibleOverride": true,
      "timeLimit": 1000,
      "version": "explicit",
      "overrideDryrun": false
    },
    "inventoryHardwareFilter": {},
    "imageFilter": {
      "imageID": "00000000-0000-0000-0000-000000000000"
    },
    "targetFilter": {}
  },
  "blockedBy": [],
  "state": "completed",
  "command": {
    "description": "restore snapshot nodebmc",
    "tag": "",
    "restoreNotPossibleOverride": true,
    "timeLimit": 1000,
    "version": "explicit",
    "overrideDryrun": false
  },
  "actionID": "f48aabf1-1616-49ae-9761-a11edb38684d",
  "startTime": "2020-06-24 14:19:14.017895702 +0000 UTC",
  "snapshotID": "00000000-0000-0000-0000-000000000000",
  "endTime": "2020-06-24 14:23:40.687948327 +0000 UTC",
  "operationSummary": {
    "succeeded": {
      "OperationsKeys": [

...
```

To view what happened at the FAS operation level:

```
ncn-w001# cray fas operations describe operationID actionID --format json
{
  "fromFirmwareVersion": "",
  "fromTag": "",
  "fromImageURL": "",
  "endTime": "2020-06-24 14:23:37.544814197 +0000 UTC",
  "actionID": "f48aabf1-1616-49ae-9761-a11edb38684d",
  "startTime": "2020-06-24 14:19:15.10128214 +0000 UTC",
  "fromSemanticFirmwareVersion": "",
  "toImageURL": "",
  "model": "WindomNodeCard_REV_D",
  "operationID": "24a5e5fb-5c4f-4848-bf4e-b071719c1850",
  "fromImageID": "00000000-0000-0000-0000-000000000000",
  "target": "BMC",
  "toImageID": "71c41a74-ab84-45b2-95bd-677f763af168",
  "toSemanticFirmwareVersion": "",
  "refreshTime": "2020-06-24 14:23:37.544824938 +0000 UTC",
  "blockedBy": [],
  "toTag": "",
  "state": "succeeded",
  "stateHelper": "unexpected change detected in firmware version. Expected nc.1.3.8-shasta-release.arm.
2020-06-15T22:57:31+00:00.b7f0725 got: nc.1.2.25-shasta-release.arm.2020-05-15T17:27:16+00:00.0cf7f51",
  "deviceType": "",
  "expirationTime": "",
  "manufacturer": "cray",
  "xname": "x9000c1s3b1",
  "toFirmwareVersion": ""
}
```

23.21 Cray System Management (CSM) Support

The installation of the Cray System Management (CSM) product requires knowledge of the various nodes and switches for the HPE Cray EX system. The procedures in this section should be referenced during the CSM install for additional information on system hardware, troubleshooting, and administrative tasks related to CSM.

23.21.1 Recover LiveCD

The *HPE Cray EX System Installation and Configuration Guide S-8000* covers the setup and installation of LiveCD. If any issues are encountered, the following actions can be taken to help troubleshoot problems with LiveCD.

Root Account

The root password is preserved within the copy-on-write (COW) partition at `cow:rw/etc/shadow`. This is the modified copy of the `/etc/shadow` file used by the operating system.

If an admin needs to reset or clear the password for `root`, they can mount their USB on another machine and remove this file from the COW partition. During then next boot from the USB, it will reinitialize to an empty password for `root`, and during the next login, it will require the password to be changed.

```
mypc# mount -L cow /mnt
mypc# sudo rm -f /mnt/rw/etc/shadow
mypc# umount -L cow
```

Basecamp

If the basecamp needs to be reset, use the following commands:

```
pit# systemctl stop basecamp
pit# podman rm basecamp
pit# podman rmi basecamp
pit# rm -f /var/www/ephemeral/configs/server.yaml
# Now basecamp will re-init.
pit# systemctl start basecamp
```

23.21.2 Re-install LiveCD

Prerequisites

LiveCD was installed using the procedure in the *HPE Cray EX System Installation and Configuration Guide S-8000*.

About this task

ROLE	System administrator
OBJECTIVE	Setup a re-install of LiveCD on a node using the previous configuration.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Backup to the data partition.

   a. Make a new directory for the backup.

   ```
   pit# mkdir -pv /var/www/ephemeral/backup
   ```

   b. Save the current directory.

   ```
   pit# pushd /var/www/ephemeral/backup
   ```

   c. Create a `tar` archive.

   ```
   pit# tar -czvf "dnsmasq-data-$(date  '+%Y-%m-%d_%H-%M-%S').tar.gz" /etc/dnsmasq.*
   pit# tar -czvf "network-data-$(date  '+%Y-%m-%d_%H-%M-%S').tar.gz" /etc/sysconfig/network/*
   ```

   d. Copy the backup files.

   ```
   pit# cp -pv /etc/hosts ./
   ```

   e. Change the current directory to the one stored with the `pushd` command.

   ```
   pit# popd
   ```

   f. Unmount the `/var/www/ephemeral` directory.

   ```
   pit# umount /var/www/ephemeral
   ```

2. Unplug the USB stick.

   The USB stick should now contain all the information already loaded, as well as the backups of the initialized files.

3. Plug the stick into a new machine, or make a backup on the booted NCN. Make a snapshot of the USB stick.

   ```
   mylinuxpc# mount /dev/disk/by-label/PITDATA /mnt
   mylinuxpc# tar -czvf --exclude *.squashfs \
   "install-data-$(date  '+%Y-%m-%d_%H-%M-%S').tar.gz" /mnt/
   mylinuxpc# umount /dev/disk/by-label/PITDATA
   ```

4. Follow the steps in the "Boot LiveCD" procedure in the *HPE Cray EX System Installation and Configuration Guide S-8000.*

   The new `tar.gz` file can be stored anywhere, and can be used to reinitialize the LiveCD.

5. Delete the existing content on the USB stick and create a new LiveCD on that same USB.

   Once the install-data partition is created, it can be remounted and can be used to restore the backup.

   ```
   mylinuxpc# mount /dev/disk/by-label/PITDATA /mnt
   mylinuxpc# tar -xzvf $(ls -ltR *.tar.gz | head -n 1)
   mylinuxpc# ls -R /mnt
   ```

   The tarball should have extracted everything into the install-data partition.

6. Retrieve the SquashFS artifacts.

   The artifacts can be retrieved at the following locations:

- `/mnt/var/www/ephemeral/k8s/`
- `/mnt/var/www/ephemeral/ceph/`

7. Attach the USB to a Cray NCN and reboot into the stick.

8. Restore the network configuration, dnsmasq, and ensure the pods are started.

   This should only be done after booting into the USB stick. Stop and inspect any of the commands that fail.

```
pit# tar -xzvf /var/www/ephemeral/backup/dnsmasq*.tar.gz
pit# tar -xzvf /var/www/ephemeral/backup/network*.tar.gz
pit# systemctl restart wicked wickedd-nanny
pit# systemctl restart dnsmasq
pit# systemctl start basecamp nexus
```

LiveCD is now re-installed with the previous configuration.

23.21.3 Rollback LiveCD

Prerequisites

LiveCD has been setup for the 1.4 release. Refer to the *HPE Cray EX System Installation and Configuration Guide S-8000* for more information.

About this task

ROLE	System administrator
OBJECTIVE	Rollback the LiveCD configuration to reflect the release 1.3 installation.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Reboot into the BIOS and change the boot order so that the USB drive is not first.

   a. Determine the boot order after rebooting into the BIOS.

   ```
   ncn-m001# efibootmgr
   ```

   b. Select a different device to boot first.

   For example:

   ```
   ncn-m001# efibootmgr -n 0002
   ```

2. Run the `enable-dns-conflict-hosts.yml` Ansible playbook after 1.3 is booted.

```
ncn-m001# ansible-playbook /opt/cray/crayctl/ansible_framework/main/enable-dns-conflict-hosts.yml \
-l ncn-w001
```

3. Start DHCP.

```
ncn-m001# systemctl start dhcpd
```

4. Power on the NCNs.

```
ncn-m001# for i in ncn-{m,s}00{1..3}-mgmt ncn-w00{2..3}-mgmt; \
do echo "------$i--------"; ipmitool -I lanplus -U $username \
-P $password -H $i chassis power on; done
```

5. Verify the status of the nodes until they are up again.

   Wait a couple minutes after powering on the nodes before running the following command.

```
ncn-m001# ansible ncn* -m ping
```

6. View the status of the nodes.

```
ncn-m001# kubectl get nodes
ncn-m001# kubectl get pods -A | grep Running | wc -l
ncn-m001# kubectl get pods -A | grep Completed | wc -l
ncn-m001# kubectl get pods -A | grep Crash | wc -l
ncn-m001# kubectl get pods -A | grep Image | wc -l
ncn-m001# ansible ncn-m* -m command -a 'ceph health'
```

If there are issues with any of the nodes, Ceph and Kubernetes can be fixed to help bring the nodes to a healthy state. Try restarting these services on the impacted nodes:

Restart non-responsive OSDs:

```
ncn-m001# for i in 0 3 6 9;do systemctl \
status ceph-osd@${i}.service | grep active;done
ncn-m001# for i in 0 3 6 9;do systemctl \
start ceph-osd@${i}.service | grep active;done
```

Restart the mgr and mds services:

```
ncn-m001# for i in mgr mds;do systemctl \
start ceph-${i}@ncn-s003.service | grep Active;done
ncn-m001# for i in mgr mds;do systemctl \
start ceph-${i}@ncn-s003.service | grep Active;done
ncn-m001# ceph status
```

23.21.4 Update the SHASTA-CFG Repository

About this task

ROLE	System admin, system installer
OBJECTIVE	The SHASTA-CFG repository contains relatively static, installation-centric artifacts, including: • Cluster-wide network configuration settings required by Helm Charts deployed by product stream Loftsman Manifests • Sealed Secrets • Sealed Secret Generate Blocks, which is a form of plain-text input that renders to a Sealed Secret

- Helm Chart value overrides that are merged into Loftsman Manifests by product stream installers

If the SHASTA-CFG repository exists from a previous release, update it based on the `shasta-cfg` directory included with that Cray System Management (CSM) release. If the SHASTA-CFG repository was previously updated **for the version of CSM being installed**, no further action is required.

```
linux# alias yq="/mnt/pitdata/${CSM_RELEASE}/shasta-cfg/utils/bin/$(uname \
| awk '{print tolower($0)}')/yq"
```

LIMITATIONS None.

NEW IN THIS RELEASE This procedure is new in release 1.4.

Procedure

CREATE OR UPDATE THE SITE-INIT DIRECTORY

1. Create the `/mnt/pitdata/prep/site-init` repository.

   - For administrators with an existing SHASTA-CFG repository:

     Clone the repository into `/mnt/pitdata/prep/site-init`.

     ```
     linux# /var/www/ephemeral/${CSM_VERSION}/shasta-cfg
     ```

   - For administrators without a SHASTA-CFG repository:

     Create the repository.

     ```
     linux# mkdir -p /mnt/pitdata/prep/site-init
     ```

2. Retain important credentials from a previous install.

   To retain `cray_reds_credentials`, `cray_meds_credentials`, or `cray_hms_rts_credentials` credentials already in `/mnt/pitdata/prep/site-init/customizations.yaml`, they **MUST BE REMOVED** from the `spec.kubernetes.tracked_sealed_secrets` list in the `/mnt/pitdata/${CSM_RELEASE}/shasta-cfg/customizations.yaml` file.

   a. View the currently tracked Sealed Secrets.

      These are the credentials that will be retained and regenerated.

      ```
      linux# yq read /mnt/pitdata/${CSM_RELEASE}/shasta-cfg/customizations.yaml \
      spec.kubernetes.tracked_sealed_secrets
      - cray_reds_credentials
      - cray_meds_credentials
      - cray_hms_rts_credentials
      ```

   b. Retain the credentials.

      ```
      linux# yq delete -i /mnt/pitdata/${CSM_RELEASE}/shasta-cfg/customizations.yaml \
      spec.kubernetes.tracked_sealed_secrets.cray_reds_credentials
      linux# yq delete -i /mnt/pitdata/${CSM_RELEASE}/shasta-cfg/customizations.yaml \
      spec.kubernetes.tracked_sealed_secrets.cray_meds_credentials
      linux# yq delete -i /mnt/pitdata/${CSM_RELEASE}/shasta-cfg/customizations.yaml \
      spec.kubernetes.tracked_sealed_secrets.cray_hms_rts_credentials
      ```

3. Initialize or update `/mnt/pitdata/prep/site-init` from CSM.

```
linux# /mnt/pitdata/${CSM_RELEASE}/shasta-cfg/meta/init.sh \
/mnt/pitdata/prep/site-init
```

UPDATE SYSTEM CUSTOMIZATIONS

4. Ensure the system-specific settings generated by Cray System Integrations (CSI) are merged into the `customizations.yaml` file.

   The system name environment variable `SYSTEM_NAME` must be set before running the following command.

   ```
   linux# yq merge -xP -i /mnt/pitdata/prep/site-init/customizations.yaml \
   <(yq prefix -P "/mnt/pitdata/prep/${SYSTEM_NAME}/customizations.yaml" spec)
   ```

   Set the cluster name:

   ```
   linux# yq write -i /mnt/pitdata/prep/site-init/customizations.yaml \
   spec.wlm.cluster_name "$SYSTEM_NAME"
   ```

5. Review the `spec.kubernetes.sealed_secrets` generate blocks for `cray_reds_credentials`, `cray_meds_credentials`, and `cray_hms_rts_credentials`.

   Replace the `Password` references with values appropriate for the system being used. If these secrets were retained, skip this step.

6. Customize the PKI Certificate Authority (CA) used by the platform.

   See **PLACEHOLDER: 055-CERTIFICATE-AUTHORITY.md** in the *HPE Cray EX System Installation and Configuration Guide S-8000*.

7. Federate with an upstream LDAP.

   a. Update the `cray-keycloak` Sealed Secret value by supplying a Base64 encoded Java KeyStore (JKS) that contains the CA certificate that signed the LDAP server's host key.

      The password for the JKS file must be *`password`*. Admins can use the `keytool` command and a PEM-encoded form of the certificate(s) to obtain this value. `keytool` may not be available on the PIT server.

      ```
      linux# keytool -importcert -trustcacerts -file myad-pub-cert.pem \
      -alias myad -keystore certs.jks -storepass password -noprompt
      linux# cat certs.jks | base64 > certs.jks.b64
      ```

   b. Encrypt the file into `customizations.yaml`.

      ```
      linux# cat <<EOF | yq w - 'data."certs.jks"' "$(<certs.jks.b64)" |
        yq r -j - | /mnt/pitdata/prep/site-init/utils/secrets-encrypt.sh |
        yq w -f - -i /mnt/pitdata/prep/site-init/customizations.yaml
      'spec.kubernetes.sealed_secrets.cray-keycloak'
      {
        "kind": "Secret",
        "apiVersion": "v1",
        "metadata": {
          "name": "keycloak-certs",
          "namespace": "services",
          "creationTimestamp": null
        },
        "data": {}
      }
      EOF
      ```

   c. Update the `keycloak_users_localize` Sealed Secret with the appropriate value for *`ldap_connection_url`*.

      Set the `ldap_connection_url`:

```
linux# yq write -i /mnt/pitdata/prep/site-init/customizations.yaml \
'spec.kubernetes.sealed_secrets.keycloak_users_localize.generate.data.
(args.name==ldap_connection_url).args.value' \
'ldaps://dcldap2.us.cray.com'
```

If successful, the `keycloak_users_localize` Sealed Secret will look similar to the following:

```
linux# yq read /mnt/pitdata/prep/site-init/customizations.yaml \
spec.kubernetes.sealed_secrets.keycloak_users_localize
generate:
    name: keycloak-users-localize
    data:
    - type: static
        args:
        name: ldap_connection_url
        value: ldaps://dcldap3.us.cray.com
```

d. Configure the `ldapSearchBase` and `localRoleAssignments` settings for the `cray-keycloak-users-localize` chart.

Set `ldapSearchBase`:

```
linux# yq write -i /mnt/pitdata/prep/site-init/customizations.yaml \
spec.kubernetes.services.cray-keycloak-users-localize.ldapSearchBase 'dc=dcldap,dc=dit'
```

Set `localRoleAssignments`:

```
linux# yq write -s - -i /mnt/pitdata/prep/site-init/customizations.yaml <<EOF
- command: update
  path: spec.kubernetes.services.cray-keycloak-users-localize.localRoleAssignments
  value:
  - {"group": "criemp", "role": "admin", "client": "shasta"}
  - {"group": "criemp", "role": "admin", "client": "cray"}
  - {"group": "craydev", "role": "admin", "client": "shasta"}
  - {"group": "craydev", "role": "admin", "client": "cray"}
  - {"group": "shasta_admins", "role": "admin", "client": "shasta"}
  - {"group": "shasta_admins", "role": "admin", "client": "cray"}
  - {"group": "shasta_users", "role": "user", "client": "shasta"}
  - {"group": "shasta_users", "role": "user", "client": "cray"}
EOF
```

On success, the `cray-keycloak-users-localize` values should look similar to the following:

```
linux# yq read /mnt/pitdata/prep/site-init/customizations.yaml \
spec.kubernetes.services.cray-keycloak-users-localize
sealedSecrets:
    - '{{ kubernetes.sealed_secrets.keycloak_users_localize | toYaml }}'
localRoleAssignments:
    - {"group": "criemp", "role": "admin", "client": "shasta"}
    - {"group": "criemp", "role": "admin", "client": "cray"}
    - {"group": "craydev", "role": "admin", "client": "shasta"}
    - {"group": "craydev", "role": "admin", "client": "cray"}
    - {"group": "shasta_admins", "role": "admin", "client": "shasta"}
    - {"group": "shasta_admins", "role": "admin", "client": "cray"}
    - {"group": "shasta_users", "role": "user", "client": "shasta"}
    - {"group": "shasta_users", "role": "user", "client": "cray"}
ldapSearchBase: dc=dcldap,dc=dit
```

8. Review the `/mnt/pitdata/prep/site-init/customizations.yaml` file and replace the remaining *~FIXME~* values with the appropriate settings.

9. Re-encrypt secrets in the `customizations.yaml` file.

```
linux# /mnt/pitdata/prep/site-init/utils/secrets-reencrypt.sh \
/mnt/pitdata/prep/site-init/customizations.yaml /mnt/pitdata/prep/site-init/certs/sealed_secrets.key
\
/mnt/pitdata/prep/site-init/certs/sealed_secrets.crt
linux# /mnt/pitdata/prep/site-init/utils/secrets-seed-customizations.sh \
/mnt/pitdata/prep/site-init/customizations.yaml
Creating Sealed Secret keycloak-certs
Generating type static_b64...
Creating Sealed Secret keycloak-master-admin-auth
```

```
Generating type static...
Generating type static...
Generating type randstr...
Generating type static...
Creating Sealed Secret cray_reds_credentials
Generating type static...
Generating type static...
Creating Sealed Secret cray_meds_credentials
Generating type static...
Creating Sealed Secret cray_hms_rts_credentials
Generating type static...
Generating type static...
Creating Sealed Secret vcs-user-credentials
Generating type randstr...
Generating type static...
Creating Sealed Secret generated-platform-ca-1
Generating type platform_ca...
Creating Sealed Secret pals-config
Generating type zmq_curve...
Generating type zmq_curve...
Creating Sealed Secret munge-secret
Generating type randstr...
Creating Sealed Secret slurmdb-secret
Generating type static...
Generating type static...
Generating type randstr...
Generating type randstr...
Creating Sealed Secret keycloak-users-localize
Generating type static...
```

COMMIT SITE-INIT CHANGES

10. Add and commit changes to the `/mnt/pitdata/prep/site-init` files.

```
linux# cd /mnt/pitdata/prep/site-init
linux# git add -A
linux# git commit -m "Updated to $(/mnt/pitdata/${CSM_RELEASE}/lib/version.sh)"
```

If `/mnt/pitdata/prep/site-init` was cloned from an upstream repository, push the commits:

```
linux# git push
```

DECRYPT SEALED SECRETS FOR REVIEW

11. Decrypt or review previously encrypted Sealed Secretes with the `secrets-decrypt.sh` utility in the SHASTA-CFG repository.

Syntax:

```
secret-decrypt.sh SEALED-SECRET-NAME SEALED-SECRET-PRIVATE-KEY CUSTOMIZATIONS-
YAML
```

```
linux:/mnt/pitdata/prep/site-init# ./utils/secrets-decrypt.sh cray_meds_credentials \
./certs/sealed_secrets.key ./customizations.yaml | jq .data.vault_redfish_defaults \
| sed -e 's/"//g' | base64 -d; echo
{"Username": "root", "Password": "..."}
```

23.21.5 Set NCN Safeguards

About this task

ROLE System administrator

OBJECTIVE	Configure safeguards to prevent destructive behaviors on non-compute nodes (NCNs). If upgrading the system, run through these safeguards on a case-by-case basis: ● Whether or not Ceph should be preserved. ● Whether or not the RAIDs should be protected.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Set safeguards for Ceph OSDs.

   a. Edit the `/var/www/ephemeral/configs/data.json` file.

   ```
   ncn-m001# vi /var/www/ephemeral/configs/data.json
   ```

   Set the following options in the `data.json` file.

   ```
   {
     ..
     // Disables ceph wipe:
     "wipe-ceph-osds": "no"
     ..
   }
   ```

   ```
   {
     ..
     // Restores default behavior:
     "wipe-ceph-osds": "yes"
     ..
   }
   ```

   b. Toggle yes or no to the file.

   ```
   # set wipe-ceph-osds=no
   ncn-m001# sed -i 's/wipe-ceph-osds": "yes"/wipe-ceph-osds": \
   "no"/g' /var/www/ephemeral/configs/data.json

   # set wipe-ceph-osds=yes
   ncn-m001# sed -i 's/wipe-ceph-osds": "no"/wipe-ceph-osds": \
   "yes"/g' /var/www/ephemeral/configs/data.json
   ```

   c. Activate the new settings.

   ```
   pit# systemctl restart basecamp
   ```

2. Set safeguards for RAIDS, BOOTLOADERS, SquashFS, and OverlayFS.

   Edit `/var/www/boot/script.ipxe` and align the following options:

   - `rd.live.overlay.reset=0` will prevent any OverlayFS files from being cleared.
   - `metal.no-wipe=1` will guard against touching RAIDs, disks, and partitions.

23.21.6 Modify the NCNs

Prerequisites

The system is installed and the NCNs are booted.

About this task

ROLE	System administrator
OBJECTIVE	Update NCN settings after the system is up and running. When NCNs are booted, the metadata they use is stored in the `data.json` file and pulled in via `cloud-init`. If changes are made to the `data.json` file or a new image is needed, reboot the machine so it picks up the changes.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

This procedure uses an example of changing the NTP configuration on the system. An admin might want to add an upstream NTP server, instead of simply have the NCNs peer with themselves. Since an upstream NTP server is not defined by default, the change would be made in `data.json`.

Procedure

1. Insert the upstream NTP server into the `data.json` file.

   ```
   ncn-m001# sed -i 's/"upstream_ntp_server": "",/"upstream_ntp_server": \
   "cfntp-4-1.us.cray.com",/' /var/www/ephemeral/configs/data.json
   ```

2. Restart basecamp to pick up the changes.

   ```
   ncn-m001# systemctl restart basecamp
   ```

3. Reboot the nods and verify the server is running.

   Rebooting the nodes will pick up the changes to the `data.json` file.

23.21.7 Update Site Connections

Prerequisites

`ncn-w001` is up and accessible via the node management network (NMN) from `ncn-m001`.

About this task

ROLE	System administrator
OBJECTIVE	In release 1.4, the site connections that were previously connected to `ncn-w001` need to be moved to `ncn-m001`. Be aware that any number of NCN master nodes may have external connections.

LIMITATIONS None.

NEW IN THIS RELEASE This procedure is new in release 1.4.

Procedure

1. Make a request to DCHW to move the BMC/Host connections and attach a USB to `ncn-m001`.

   Ensure `ncn-w001` is up and accessible via the NMN from `ncn-m001`. The following changes are needed:

   - Swap `mn01` and `wn01` cabling. `mn01-j1` and `mn01-j3` need the external/site-links, and both `wn01-j1` and `wn01-j3` should be wired into the leaf switch.
   - Plug a USB stick (250GB or larger) into mn01. If one is already in `wn01`, please move it to `mn01`. Otherwise, new USBs should have been ordered.
   - Create DNS records for m001 and its BMC.
     - `<system-name>-ncn-m001`
     - `<system-name>-ncn-m001-mgmt`

     The old `ncn-w001` DNS entries should remain/stay to prevent interruptions, if at all possible.

2. Set the new host IP and default route.

   a. Navigate to the console of `ncn-m001` using the BMC address from the updates in the previous step.

   ```
   username=''
   password=''
   bmcaddr=''
   ```

   ```
   ncn-m001# ipmitool -I lanplus -U $username -P \
   $password -H $bmcaddr sol activate
   ```

   b. Set the new static em1 IP address in `/etc/sysconfig/network/ifcfg-em1`.

   Replace *`172.30.XX.XX`* with the *`em1`* IP for the system being used.

   ```
   ncn-m001# vi /etc/sysconfig/network/ifcfg-em1
   BOOTPROTO='static'
   STARTMODE='auto'
   ONBOOT='yes'
   IPADDR='172.30.XX.XX'
   NETMASK='255.255.240.0'
   ```

   c. Set the default route in `/etc/sysconfig/network/ifroute-em1`.

   ```
   ncn-m001# vi /etc/sysconfig/network/ifroute-em1
   default 172.30.48.1 255.255.240.0 em1
   ```

   d. Bring up em1.

   ```
   ncn-m001# wicked ifdown em1
   ncn-m001# wicked ifup em1
   ```

   e. Confirm that the IP address and default were set correctly.

   ```
   ncn-m001# ip addr show em1
   ncn-m001# ip route | grep default
   ```

f. Exit out of the `sol` console.

3. Set the `ncn-w001` BMC to DHCP.

   a. SSH from a laptop to `ncn-m001` via the new external connection.

   b. SSH from `ncn-m001` to `ncn-w001` over the NMN.

   c. Check to see if the *`IP Address Source`* value is set to Static or DHCP.

      ```
      ncn-w001# ipmitool lan print 1
      ```

      If the value is set to Static, run the following command:

      ```
      ncn-w001# ipmitool lan set 1 ipsrc dhcp
      ```

   d. Verify that the BMC is now set to DHCP and record if it has picked up an IP address.

      If an IP address isn't present, it will be picked up in a future step.

      ```
      ncn-w001# ipmitool lan print 1
      ```

4. Capture the bond0 MAC addresses and `ncn-w001` BMC MAC address.

   ```
   ncn-w001# ansible ncn -m shell -a "ip addr show bond0 \
   | grep ether" > /root/macs.txt
   ncn-w001# ipmitool lan print 1 | grep "MAC Address" \
   >> /root/macs.txt
   ```

5. Copy the file from the previous step to `ncn-m001`.

   ```
   ncn-w001# scp /root/macs.txt ncn-m001:/root
   ```

6. Log out of `ncn-w001`.

   `ncn-m001` should now be the node being used.

7. Shutdown all of the nodes, except for `ncn-m001`.

   Ensure all of the NCNs are shutdown before starting the 1.4 installation to avoid having multiple DHCP servers running. Kea is also serving the BMCs their addresses, so this should be done to all NCNs at approximately the same time before they lose their leases.

   Use `ipmitool` from `ncn-m001` to shutdown all of the NCNs other than `ncn-m001` and `ncn-w001`.

   ```
   username=''
   password=''
   ```

   ```
   ncn-m001# for i in m002 m003 w002 w003 s001 s002 s003;do ipmitool \
   -I lanplus -U $username -P $password -H \
   ncn-${i}-mgmt chassis power off;done
   ```

   If an IP address was found in step *3.d* on page 221, use `ipmitool` from `ncn-m001` to power off `ncn-w001`.

   If an IP address was not retrieved in step *3.d* on page 221 or the IP cannot be accessed, SSH to `ncn-w001` from `ncn-m001` and execute `shutdown -h now`. Make sure any important information is moved from `ncn-w001` because there will not be access to bring it back up until another DHCP server is brought up on the LiveCD.

Proceed to "Setup Booted LiveCD" in the *HPE Cray EX System Installation and Configuration Guide S-8000*.

23.21.8 Wipe Disks

This section describes how disks are wiped, and includes workarounds for wedged disks. Any process covered in this section will be covered by the installer.

CAUTION: Everything in this section should be considered DESTRUCTIVE.

After following these procedures, an NCN can be rebooted and redeployed.

Ideally, the basic wipe is enough, and should be tried first. Any type of disk wipe can be ran from Linux or an initramFS/initrd emergency shell. The following are potential use cases for wiping disks:

- Adding a node that isn't bare
- Adopting new disks that aren't bare
- Fresh-installing

Basic Wipe

A basic wipe includes wiping the disks and all of the RAIDS. These basic wipe instructions can be executed on all NCNs (master, worker and storage).

The following steps should be taken:

1. Print off the disk for verification.

   ```
   ncn-m001# ls -1 /dev/sd* /dev/disk/by-label/*
   ```

2. Wipe the disks and the RAIDS.

   ```
   ncn-m001# wipefs --all --force /dev/sd* /dev/disk/by-label/*
   ```

   The final `wipefs` command may fail if no labeled disks are found, which is not an indication of a larger problem.

Advanced Wipe

This section is specific to NCN storage nodes. An advanced wipe includes deleting the Ceph volumes, and then wiping the disks and RAIDS.

The following steps should be taken:

1. Delete the Ceph volumes.

   Make sure the OSDs (if any) are not running after running the first command.

   ```
   ncn-s001# systemctl stop ceph-osd.target
   ncn-s001# ls -1 /dev/sd* /dev/disk/by-label/*
   ncn-s001# vgremove -f --select 'vg_name=~ceph*'
   ```

2. Wipe the disks and the RAIDS.

   ```
   ncn-m001# wipefs --all --force /dev/sd* /dev/disk/by-label/*
   ```

Full Wipe

This section is specific to NCN storage nodes. A full wipe includes deleting the Ceph volumes, stopping the RAIDs, zeroing the disks, and then wiping the disks and RAIDS.

The following steps should be taken:

1. Delete the Ceph volumes.

   Make sure the OSDs (if any) are not running after running the first command.

   ```
   ncn-s001# systemctl stop ceph-osd.target
   ncn-s001# ls -1 /dev/sd* /dev/disk/by-label/*
   ncn-s001# vgremove -f --select 'vg_name=~ceph*'
   ```

2. Stop the RAIDs.

   ```
   ncn-m001# for md in /dev/md/*; do mdadm \
   -S $md || echo nope ; done
   ```

3. Wipe the disks and the RAIDS.

   ```
   ncn-m001# sgdisk --zap-all /dev/sd*
   ncn-m001# wipefs --all --force /dev/sd* /dev/disk/by-label/*
   ```

23.21.9 Software and Hardware Network Stack

Interfaces within the network stack can be reloaded or reset to fix wedged interfaces. NCNs have network device names that are set during the initial boot. The names vary based on the available hardware. For more information, refer to "NCN networking" in the *HPE Cray EX System Administration Guide S-8001*. Any process covered on this page will be covered by the installer.

There are a few daemons that makeup the SUSE network stack. The following are sorted by safest to touch relative to keeping an SSH connection up.

1. `wickedd.service`: The daemons handling each interface. Resetting this clears a stale configuration. The following command restarts Wicked daemons without reconfiguring the network interfaces.

   ```
   ncn-m001# systemctl restart wickedd
   ```

2. `wicked.service`: The overarching service for spawning daemons and manipulating interface configuration. Resetting this reloads daemons and configuration. The following command restarts Wicked, re-spawns daemons, and (re)configures the network.

   ```
   ncn-m001# systemctl restart wicked
   ```

3. `network.service`: Responsible for network configuration per interface. This does not reload Wicked. The following command restarts the network interface configuration, but leaves wicked daemons alone.

   ```
   ncn-m001# systemctl restart network
   ```

The following are potential use cases for resetting services:

- Interfaces not showing up
- IP Addresses not applying
- Member/children interfaces not being included

Command References

To check the interface status (up, down, or broken):

```
ncn-m001# wicked ifstatus
```

To show routing and the status for all devices:

```
ncn-m001# wicked ifstatus --verbose all
lo              up
      link:     #1, state up
      type:     loopback
      control:  persistent
      config:   compat:suse:/etc/sysconfig/network/ifcfg-lo,
                uuid: 6ad37e59-72d7-5988-9675-93b8df96d9f6
      leases:   ipv4 static granted
      leases:   ipv6 static granted
      addr:     ipv4 127.0.0.1/8 scope host label lo [static]
      addr:     ipv6 ::1/128 scope host [static]
      route:    ipv6 ::1/128 type unicast table main scope universe protocol kernel priority 256

em1             device-unconfigured
      link:     #2, state down, mtu 1500
      type:     ethernet, hwaddr a4:bf:01:48:1f:dc
      config:   none

em2             device-unconfigured
      link:     #3, state down, mtu 1500
      type:     ethernet, hwaddr a4:bf:01:48:1f:dd
      config:   none

mgmt0           enslaved
      link:     #4, state up, mtu 9000, master bond0
      type:     ethernet, hwaddr b8:59:9f:f9:1c:8e
      control:  none
      config:   compat:suse:/etc/sysconfig/network/ifcfg-mgmt0,
                uuid: 7175c041-ee2b-5ce2-a4d7-67fa6cb94a17

...
```

To print the real devices:

```
ncn-m001# wicked show --verbose all
```

To show the currently enabled network service (Wicked or Network Manager):

```
ncn-m001# systemctl show -p Id network.service
Id=wicked.service
```

23.21.1 Configure NTP on NCNs 0

Prerequisites

The NCNs are booted and running.

About this task

ROLE	System administrator
OBJECTIVE	The NCNs serve Network Time Protocol (NTP) at stratum 3, and all NCNs peer with each other. Currently, the LiveCD does not run NTP, but the other nodes do when they are

booted. NTP is currently allowed on the Node Management Network (NMN) and Hardware Management Network (HMN).

The NTP peers are set in the `data.json` file, which is normally created during an initial install. To configure NTP, edit this file, restart basecamp, and then reboot the nodes to apply the change.

If NTP is not configure in the `data.json` file, the NCNs will simply peer with themselves until an upstream NTP server is configured. The time on the NCNs may not match the current time at the site, but they will stay in sync with each other.

LIMITATIONS None.

NEW IN THIS RELEASE This procedure is new in release 1.4.

Procedure

1. Update the configuration on the NCNs to enable NTP.

   There are three different methods for configuration NTP, which are described below. The first option is the recommended method.

   - Edit `/etc/chrony.d/cray.conf` and restart chron on each node.

     ```
     ncn-w001# vi /etc/chrony.d/cray.conf
     ncn-w001# systemctl restart chronyd
     ```

   - Edit the `data.json` file, restart basecamp, and run the NTP script on each node.

     ```
     ncn-w001# vi data.json
     ncn-w001# systemctl restart basecamp
     ncn-w001# /srv/cray/scripts/metal/set-ntp-config.sh
     ```

   - Edit the `data.json` file, restart basecamp, and restart nodes so cloud-init runs on boot.

     ```
     ncn-w001# vi data.json
     ncn-w001# systemctl restart basecamp
     ```

     Cloud-init caches data, so there could be inconsistent results.

2. Verify NTP is configured correctly and troubleshoot any issues.

   The `chronyc` command can be used to gather information on the state of NTP.

   a. Check if a host is allowed to use NTP from HOST.

      ```
      ncn-w001# chronyc accheck 10.252.0.7
      208 Access allowed
      ```

   b. Check the system clock performance.

      ```
      ncn-s001# chronyc tracking
      Reference ID    : 0AFC0104 (ncn-s003)
      Stratum         : 4
      Ref time (UTC)  : Mon Nov 30 20:02:24 2020
      System time     : 0.000007622 seconds slow of NTP time
      Last offset     : -0.000014609 seconds
      RMS offset      : 0.000015776 seconds
      Frequency       : 6.773 ppm fast
      Residual freq   : -0.000 ppm
      Skew            : 0.008 ppm
      ```

```
Root delay      : 0.000075896 seconds
Root dispersion : 0.000484318 seconds
Update interval : 513.7 seconds
Leap status     : Normal
```

c. View information on drift and offset.

```
ncn-s001# chronyc sourcestats
210 Number of sources = 8
Name/IP Address            NP  NR  Span  Frequency  Freq Skew  Offset  Std Dev
==============================================================================
ncn-w001                    6   3   42m     -0.029      0.126  +4104ns    28us
ncn-w002                    6   6   42m     -0.028      0.030    +44us  7278ns
ncn-w003                   12   7   23m     -0.059      0.023    -35us  8359ns
ncn-s002                   36  17  213m     -0.001      0.010  +5794ns    54us
ncn-s003                   36  17  212m     -0.000      0.007   -178ns    40us
ncn-m001                    0   0     0     +0.000   2000.000     +0ns  4000ms
ncn-m002                   28  15  192m     -0.007      0.009  +9942ns    49us
ncn-m003                   24  15  197m     -0.005      0.009  +9442ns    46us
```

d. View the NTP servers, pools, and peers.

```
ncn-s001# chronyc sources
210 Number of sources = 8
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
=? ncn-w001                      4   9   377   435   +162us[ +164us] +/-  679us
=? ncn-w002                      4   9   377   505   +118us[ +120us] +/-  277us
=? ncn-w003                      4   7   377    82   +850ns[+2686ns] +/-  504us
=? ncn-s002                      4   9   377   542    -38us[  -36us] +/-  892us
=* ncn-s003                      3   9   377    19    +13us[  +15us] +/-  110us
=? ncn-m001                      0   9     0     -     +0ns[   +0ns] +/-    0ns
=? ncn-m002                      4   8   377   161    -47us[  -45us] +/-  408us
=? ncn-m003                      4   8   377   215    -11us[-9109ns] +/-  446us
```

e. View the log files at `/var/log/chrony/`.

```
ncn-w001# cd /var/log/chrony/
```

f. Force a time sync.

If the time is out of sync, force a sync of NTP.

If Kubernetes or other services are already up, they don't always react well if there is a large time jump. Ideally, this action should be made as the node is booting.

```
# chronyc burst 4/4
```

Wait about 15 seconds while NTP measurements are gathered and then manually adjust the clock.

```
# chronyc makestep
```

23.21.1 Set NCN Images Before Booting 1

About this task

ROLE	System administrator
OBJECTIVE	There are several types of NCN images, which each share a common base image. When booting NCNs, an admin will need to choose between stable (release) and unstable (pre-release/dev) images.

Each application image (Kubernetes and storage-ceph) inherit from the non-compute-common layer. Operationally, these are all that matter; the common layer, Kubernetes layer, Ceph layer, and any other new application images.

To boot an NCN, three artifacts are needed for each node type:

- Kubernetes SquashFS
  - `initrd-img-[RELEASE].xz`
  - `$version-[RELEASE].kernel`
  - `kubernetes-[RELEASE].squashfs`
- Ceph SquashFS
  - `initrd-img-[RELEASE].xz`
  - `$version-[RELEASE].kernel`
  - `storage-ceph-[RELEASE].squashfs`

LIMITATIONS None.

NEW IN THIS RELEASE This procedure is new in release 1.4.

Procedure

1. Retrieve the NCN artifacts.

   The following are the accurate version numbers for the kernel.

   ```
   stream=stable
   id_ceph=0.0.7
   id_k8s=0.0.8
   ```

   ```
   ncn-m001# wget --mirror -np -nH --cut-dirs=4 -A *.kernel,*initrd*,*.squashfs -nv \
   https://arti.dev.cray.com:443/artifactory/node-images-${stream}-local/shasta/storage-ceph/$id_ceph/
   ncn-m001# wget --mirror -np -nH --cut-dirs=4 -A *.kernel,*initrd*,*.squashfs -nv \
   https://arti.dev.cray.com:443/artifactory/node-images-${stream}-local/shasta/kubernetes/$id_k8s/
   ```

2. View the current ephemeral data payload for the LiveCD server.

   ```
   pit# ll /var/www
   total 8
   drwxr-xr-x 1 dnsmasq tftp 4096 Dec 17 21:20 boot
   drwxr-xr-x 7 root    root 4096 Dec  2 04:45 ephemeral

   pit# ll /var/www/ephemeral/data/*
   /var/www/ephemeral/data/ceph:
   total 4
   drwxr-xr-x 2 root root 4096 Dec 17 21:42 0.0.7

   /var/www/ephemeral/data/k8s:
   total 4
   drwxr-xr-x 2 root root 4096 Dec 17 21:26 0.0.8
   ```

3. Set up the booting repositories.

   ```
   pit# set-sqfs-links.sh
   Mismatching kernels! The discovered artifacts will deploy an undesirable stack.
   mkdir: created directory 'ncn-m001'
   /var/www/ncn-m001 /var/www
   'kernel' -> '../ephemeral/data/k8s/0.0.8/5.3.18-24.37-default-0.0.8.kernel'
   ```

```
'initrd.img.xz' -> '../ephemeral/data/k8s/0.0.8/initrd.img-0.0.8.xz'
'filesystem.squashfs' -> '../ephemeral/data/k8s/0.0.8/kubernetes-0.0.8.squashfs'
/var/www
mkdir: created directory 'ncn-m002'
/var/www/ncn-m002 /var/www
'kernel' -> '../ephemeral/data/k8s/0.0.8/5.3.18-24.37-default-0.0.8.kernel'
'initrd.img.xz' -> '../ephemeral/data/k8s/0.0.8/initrd.img-0.0.8.xz'
'filesystem.squashfs' -> '../ephemeral/data/k8s/0.0.8/kubernetes-0.0.8.squashfs'
/var/www
mkdir: created directory 'ncn-m003'
/var/www/ncn-m003 /var/www
'kernel' -> '../ephemeral/data/k8s/0.0.8/5.3.18-24.37-default-0.0.8.kernel'
'initrd.img.xz' -> '../ephemeral/data/k8s/0.0.8/initrd.img-0.0.8.xz'
'filesystem.squashfs' -> '../ephemeral/data/k8s/0.0.8/kubernetes-0.0.8.squashfs'
/var/www
mkdir: created directory 'ncn-w002'
/var/www/ncn-w002 /var/www
'kernel' -> '../ephemeral/data/k8s/0.0.8/5.3.18-24.37-default-0.0.8.kernel'
'initrd.img.xz' -> '../ephemeral/data/k8s/0.0.8/initrd.img-0.0.8.xz'
'filesystem.squashfs' -> '../ephemeral/data/k8s/0.0.8/kubernetes-0.0.8.squashfs'
/var/www
mkdir: created directory 'ncn-w003'
/var/www/ncn-w003 /var/www
'kernel' -> '../ephemeral/data/k8s/0.0.8/5.3.18-24.37-default-0.0.8.kernel'
'initrd.img.xz' -> '../ephemeral/data/k8s/0.0.8/initrd.img-0.0.8.xz'
'filesystem.squashfs' -> '../ephemeral/data/k8s/0.0.8/kubernetes-0.0.8.squashfs'
/var/www
mkdir: created directory 'ncn-s001'
/var/www/ncn-s001 /var/www
'kernel' -> '../ephemeral/data/ceph/0.0.7/5.3.18-24.37-default-0.0.7.kernel'
'initrd.img.xz' -> '../ephemeral/data/ceph/0.0.7/initrd.img-0.0.7.xz'
'filesystem.squashfs' -> '../ephemeral/data/ceph/0.0.7/storage-ceph-0.0.7.squashfs'
/var/www
mkdir: created directory 'ncn-s002'
/var/www/ncn-s002 /var/www
'kernel' -> '../ephemeral/data/ceph/0.0.7/5.3.18-24.37-default-0.0.7.kernel'
'initrd.img.xz' -> '../ephemeral/data/ceph/0.0.7/initrd.img-0.0.7.xz'
'filesystem.squashfs' -> '../ephemeral/data/ceph/0.0.7/storage-ceph-0.0.7.squashfs'
/var/www
mkdir: created directory 'ncn-s003'
/var/www/ncn-s003 /var/www
'kernel' -> '../ephemeral/data/ceph/0.0.7/5.3.18-24.37-default-0.0.7.kernel'
'initrd.img.xz' -> '../ephemeral/data/ceph/0.0.7/initrd.img-0.0.7.xz'
'filesystem.squashfs' -> '../ephemeral/data/ceph/0.0.7/storage-ceph-0.0.7.squashfs'
/var/www
```

4. View the currently set links.

```
pit# ll /var/www/ncn-*
boot:
total 1552
-rw-r--r-- 1 root    root 166634 Dec 17 13:21 graffiti.png
-rw-r--r-- 1 dnsmasq tftp 700480 Dec 17 13:25 ipxe.efi
-rw-r--r-- 1 dnsmasq tftp 700352 Dec 15 09:35 ipxe.efi.stable
-rw-r--r-- 1 root    root   6157 Dec 15 05:12 script.ipxe
-rw-r--r-- 1 root    root   6284 Dec 17 13:21 script.ipxe.rpmnew

ephemeral:
total 32
drwxr-xr-x 2 root root  4096 Dec  6 22:18 configs
drwxr-xr-x 4 root root  4096 Dec  7 04:29 data
drwx------ 2 root root 16384 Dec  2 04:25 lost+found
drwxr-xr-x 4 root root  4096 Dec  3 02:31 prep
drwxr-xr-x 2 root root  4096 Dec  2 04:45 static

ncn-m001:
total 4
lrwxrwxrwx 1 root root 53 Dec 26 06:11 filesystem.squashfs -> ../ephemeral/data/k8s/0.0.8/kubernetes-0.0.8.squashfs
lrwxrwxrwx 1 root root 47 Dec 26 06:11 initrd.img.xz -> ../ephemeral/data/k8s/0.0.8/initrd.img-0.0.8.xz
lrwxrwxrwx 1 root root 61 Dec 26 06:11 kernel -> ../ephemeral/data/k8s/0.0.8/5.3.18-24.37-default-0.0.8.kernel

ncn-m002:
total 4
lrwxrwxrwx 1 root root 53 Dec 26 06:11 filesystem.squashfs -> ../ephemeral/data/k8s/0.0.8/kubernetes-0.0.8.squashfs
lrwxrwxrwx 1 root root 47 Dec 26 06:11 initrd.img.xz -> ../ephemeral/data/k8s/0.0.8/initrd.img-0.0.8.xz
lrwxrwxrwx 1 root root 61 Dec 26 06:11 kernel -> ../ephemeral/data/k8s/0.0.8/5.3.18-24.37-default-0.0.8.kernel

ncn-m003:
total 4
```

```
lrwxrwxrwx 1 root root 53 Dec 26 06:11 filesystem.squashfs -> ../ephemeral/data/k8s/0.0.8/kubernetes-0.0.8.squashfs
lrwxrwxrwx 1 root root 47 Dec 26 06:11 initrd.img.xz -> ../ephemeral/data/k8s/0.0.8/initrd.img-0.0.8.xz
lrwxrwxrwx 1 root root 61 Dec 26 06:11 kernel -> ../ephemeral/data/k8s/0.0.8/5.3.18-24.37-default-0.0.8.kernel

ncn-s001:
total 4
lrwxrwxrwx 1 root root 56 Dec 26 06:11 filesystem.squashfs -> ../ephemeral/data/ceph/0.0.7/storage-
ceph-0.0.7.squashfs
lrwxrwxrwx 1 root root 48 Dec 26 06:11 initrd.img.xz -> ../ephemeral/data/ceph/0.0.7/initrd.img-0.0.7.xz
lrwxrwxrwx 1 root root 62 Dec 26 06:11 kernel -> ../ephemeral/data/ceph/0.0.7/5.3.18-24.37-default-0.0.7.kernel

ncn-s002:
total 4
lrwxrwxrwx 1 root root 56 Dec 26 06:11 filesystem.squashfs -> ../ephemeral/data/ceph/0.0.7/storage-
ceph-0.0.7.squashfs
lrwxrwxrwx 1 root root 48 Dec 26 06:11 initrd.img.xz -> ../ephemeral/data/ceph/0.0.7/initrd.img-0.0.7.xz
lrwxrwxrwx 1 root root 62 Dec 26 06:11 kernel -> ../ephemeral/data/ceph/0.0.7/5.3.18-24.37-default-0.0.7.kernel

ncn-s003:
total 4
lrwxrwxrwx 1 root root 56 Dec 26 06:11 filesystem.squashfs -> ../ephemeral/data/ceph/0.0.7/storage-
ceph-0.0.7.squashfs
lrwxrwxrwx 1 root root 48 Dec 26 06:11 initrd.img.xz -> ../ephemeral/data/ceph/0.0.7/initrd.img-0.0.7.xz
lrwxrwxrwx 1 root root 62 Dec 26 06:11 kernel -> ../ephemeral/data/ceph/0.0.7/5.3.18-24.37-default-0.0.7.kernel

ncn-w002:
total 4
lrwxrwxrwx 1 root root 53 Dec 26 06:11 filesystem.squashfs -> ../ephemeral/data/k8s/0.0.8/kubernetes-0.0.8.squashfs
lrwxrwxrwx 1 root root 47 Dec 26 06:11 initrd.img.xz -> ../ephemeral/data/k8s/0.0.8/initrd.img-0.0.8.xz
lrwxrwxrwx 1 root root 61 Dec 26 06:11 kernel -> ../ephemeral/data/k8s/0.0.8/5.3.18-24.37-default-0.0.8.kernel

ncn-w003:
total 4
lrwxrwxrwx 1 root root 53 Dec 26 06:11 filesystem.squashfs -> ../ephemeral/data/k8s/0.0.8/kubernetes-0.0.8.squashfs
lrwxrwxrwx 1 root root 47 Dec 26 06:11 initrd.img.xz -> ../ephemeral/data/k8s/0.0.8/initrd.img-0.0.8.xz
lrwxrwxrwx 1 root root 61 Dec 26 06:11 kernel -> ../ephemeral/data/k8s/0.0.8/5.3.18-24.37-default-0.0.8.kernel
```

23.21.1 NCN Boot Workflow
2

Determine if NCNs are Booted via Disk or PXE

There are two different methods for determining whether an NCN is booted using disk or PXE. The method to use will vary depending on the system environment. The two methods are described below:

- Run the `cat /proc/cmdline` command.

  ```
  ncn-w001# cat /proc/cmdline
  ```

  If the output starts with `kernel`, the node network booted. If it starts with `BOOT_IMAGE=(`, then it disk booted.

- Run the `efibootmgr` command.

  ```
  ncn-w001# efibootmgr
  ```

  View the returned entry for `BootCurrent` and determine if it lines up with a networking option or a `cray sd*)` option for disk boots.

Set BMCs to DHCP

If reinstalling a system, the BMCs for the NCNs may be set to static. Check `/var/lib/misc/dnsmasq.leases` to verify the symlinks are setup for the artifacts each node needs to boot.

If the BMCs are set to static, those artifacts will not get setup correctly. Set them back to DHCP by using the following command:

```
for h in $( grep mgmt /etc/dnsmasq.d/statics.conf | grep -v m001 | awk -F ','
'{print $2}' )
do
ipmitool -U username -I lanplus -H $h -P password lan set 1 ipsrc dhcp
done
```

Some BMCs require a cold reset to fully pick up the change:

```
for h in $( grep mgmt /etc/dnsmasq.d/statics.conf | grep -v m001 | awk -F ','
'{print $2}' )
do
ipmitool -U username -I lanplus -H $h -P password mc reset cold
done
```

NCN Boot Order

The NCN boot order is configured with the following tools:

- `ipmitool`: Used to set and edit boot order; it works better on some vendors based on their BMC implementation.
- `efibootmgr`: Speaks directly to systems UEFI and can only be ignored by new BIOS activity.

The following workflow is required to set the boot order:

1. Create the boot-bios-selector file(s) based on the manufacturer.

   Gigabyte Technology

   - Master nodes:

     ```
     ncn-m# efibootmgr | grep -iP '(pxe ipv?4.*adapter)' | tee /tmp/bbs1
     Boot0007* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:BE:8F:2E
     Boot0009* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:BE:8F:2F
     ncn-m# efibootmgr | grep cray | tee /tmp/bbs2
     Boot0000* cray (sda1)
     Boot0002* cray (sdb1)
     ```

   - Storage nodes:

     ```
     ncn-s# efibootmgr | grep -iP '(pxe ipv?4.*adapter)' | tee /tmp/bbs1
     Boot0007* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:C7:11:FA
     Boot0009* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:C7:11:FB
     ncn-s# efibootmgr | grep cray | tee /tmp/bbs2
     Boot0000* cray (sda1)
     Boot0002* cray (sdb1)
     ```

   - Worker nodes:

     ```
     ncn-w# efibootmgr | grep -iP '(pxe ipv?4.*adapter)' | tee /tmp/bbs1
     Boot0007* UEFI: PXE IP4 Mellanox Network Adapter - 98:03:9B:AA:88:30
     Boot0009* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:34:89:2A
     Boot000B* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:34:89:2B
     ncn-w# efibootmgr | grep cray | tee /tmp/bbs2
     Boot0000* cray (sda1)
     Boot0002* cray (sdb1)
     ```

   Hewlett-Packard Enterprise (HPE)

   - Master nodes:

     ```
     ncn-m# efibootmgr | grep -i 'port 1' | grep -i 'pxe ipv4' | tee /tmp/bbs1
     Boot0014* OCP Slot 10 Port 1 : Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HQCU-HC OCP3 Adapter -
     NIC - Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HQCU-HC OCP3 Adapter - PXE (PXE IPv4)
     Boot0018* Slot 1 Port 1 : Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HLCU-HC MD2 Adapter - NIC -
     Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HLCU-HC MD2 Adapter - PXE (PXE IPv4)
     ncn-m# efibootmgr | grep cray | tee /tmp/bbs2
     Boot0021* cray (sdb1)
     Boot0022* cray (sdc1)
     ```

- Storage nodes:

```
ncn-s# efibootmgr | grep -i 'port 1' | grep -i 'pxe ipv4' | tee /tmp/bbs1
Boot001C* OCP Slot 10 Port 1 : Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HQCU-HC OCP3 Adapter -
NIC - Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HQCU-HC OCP3 Adapter - PXE (PXE IPv4)
Boot001D* Slot 1 Port 1 : Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HLCU-HC MD2 Adapter - NIC -
Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HLCU-HC MD2 Adapter - PXE (PXE IPv4)
ncn-s# efibootmgr | grep cray | tee /tmp/bbs2
Boot0002* cray (sdg1)
Boot0020* cray (sdh1)
```

- Worker nodes:

```
ncn-w# efibootmgr | grep -i 'port 1' | grep -i 'pxe ipv4' | tee /tmp/bbs1
Boot0012* OCP Slot 10 Port 1 : Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HQCU-HC OCP3 Adapter -
NIC - Marvell FastLinQ 41000 Series - 2P 25GbE SFP28 QL41232HQCU-HC OCP3 Adapter - PXE (PXE IPv4)
ncn-w# efibootmgr | grep cray | tee /tmp/bbs2
Boot0017* cray (sdb1)
Boot0018* cray (sdc1)
```

Intel Corporation

- Master nodes:

```
ncn-m# efibootmgr | grep -i 'ipv4' | grep -iv 'baseboard' | tee /tmp/bbs1
Boot000E* UEFI IPv4: Network 00 at Riser 02 Slot 01
Boot0014* UEFI IPv4: Network 01 at Riser 02 Slot 01
ncn-m# efibootmgr | grep -i 'cray' | tee /tmp/bbs2
Boot0011* cray (sda1)
Boot0012* cray (sdb1)
```

- Storage nodes:

```
ncn-s# efibootmgr | grep -i 'ipv4' | grep -iv 'baseboard' | tee /tmp/bbs1
Boot000E* UEFI IPv4: Network 00 at Riser 02 Slot 01
Boot0012* UEFI IPv4: Network 01 at Riser 02 Slot 01
ncn-s# efibootmgr | grep -i 'cray' | tee /tmp/bbs2
Boot0014* cray (sda1)
Boot0015* cray (sdb1)
```

- Worker nodes:

```
ncn-w# efibootmgr | grep -i 'ipv4' | grep -iv 'baseboard' | tee /tmp/bbs1
Boot0008* UEFI IPv4: Network 00 at Riser 02 Slot 01
Boot000C* UEFI IPv4: Network 01 at Riser 02 Slot 01
ncn-w# efibootmgr | grep -i 'cray' | tee /tmp/bbs2
Boot0010* cray (sda1)
Boot0011* cray (sdb1)
```

2. Set the boot order.

   The following command works universally for each vendor and NCN type.

```
ncn-m# efibootmgr -o $(cat /tmp/bbs* | sed 's/^Boot//g' \
| awk '{print $1}' | tr -t '*' ',' | tr -d '\n' \
| sed 's/,$//') | grep -i bootorder
BootOrder: 000E,0014,0011,0012
```

Remove PXE Entries

The boot menu should be trimmed down to only contain relevant entries. Use the following commands to remove additional PXE entries.

1. Find the other PXE entries:

   - **Gigabyte Technology:**

     The following command will not remove on-board PXE IPv4 options.

```
ncn# efibootmgr | grep -ivP '(pxe ipv?4.*)' \
| grep -iP '(adapter|connection)' | tee /tmp/rbbs1
```

   - **HPE**:

```
ncn# efibootmgr | grep -i 'port 1' \
| grep -vi 'pxe ipv4' | tee /tmp/rbbs1
```

- **Intel Corporation**:

```
ncn# efibootmgr | grep -vi 'ipv4' \
| grep -i 'baseboard' | tee /tmp/rbbs1
```

2. Remove the entries.

```
ncn# cat /tmp/rbbs* | sed 's/^Boot//g' | awk '{print $1}' \
| tr -d '*' | xargs -t -i efibootmgr -b {} -B
```

Example of a master node with on-boards enabled:

```
ncn-m# efibootmgr
BootCurrent: 0009
Timeout: 2 seconds
BootOrder: 0004,0000,0007,0009,000B,000D,0012,0013,0002,0003,0001
Boot0000* cray (sda1)
Boot0001* UEFI: Built-in EFI Shell
Boot0002* UEFI OS
Boot0003* UEFI OS
Boot0004* cray (sdb1)
Boot0007* UEFI: PXE IP4 Intel(R) I350 Gigabit Network Connection
Boot0009* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:34:89:62
Boot000B* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:34:89:63
Boot000D* UEFI: PXE IP4 Intel(R) I350 Gigabit Network Connection
Boot0012* UEFI: PNY USB 3.1 FD PMAP
Boot0013* UEFI: PNY USB 3.1 FD PMAP, Partition 2
```

Example of a storage node with on-boards enabled:

```
ncn-s# efibootmgr
BootNext: 0005
BootCurrent: 0006
Timeout: 2 seconds
BootOrder: 0007,0009,0000,0002
Boot0000* cray (sda1)
Boot0001* UEFI: Built-in EFI Shell
Boot0002* cray (sdb1)
Boot0005* UEFI: PXE IP4 Intel(R) I350 Gigabit Network Connection
Boot0007* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:34:88:76
Boot0009* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:34:88:77
Boot000B* UEFI: PXE IP4 Intel(R) I350 Gigabit Network Connection
```

Example of a worker node with on-boards enabled:

```
ncn-w# efibootmgr
BootNext: 0005
BootCurrent: 0008
Timeout: 2 seconds
BootOrder: 0007,0009,000B,0000,0002
Boot0000* cray (sda1)
Boot0001* UEFI: Built-in EFI Shell
Boot0002* cray (sdb1)
Boot0005* UEFI: PXE IP4 Intel(R) I350 Gigabit Network Connection
Boot0007* UEFI: PXE IP4 Mellanox Network Adapter -98:03:9B:AA:88:30
Boot0009* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:34:89:2A
Boot000B* UEFI: PXE IP4 Mellanox Network Adapter - B8:59:9F:34:89:2B
Boot000D* UEFI: PXE IP4 Intel(R) I350 Gigabit Network Connection
```

Revert Changes

Reset the BIOS to revert changes to the boot order. Refer to vendor documentation for resetting the BIOS. Optionally, attempt to reset the BIOS with `ipmitool`:

```
# reset
ipmitool chassis bootdev none options=clear-cmos

# set boot order
ipmitool chassis bootdev pxe options=efiboot,persistent
```

```
# boot to BIOS for checkout
ipmitool chassis bootdev bios options=efiboot
```

`ipmitool` works against a machine remotely over TCP/IP. The `ipmitool` command requires the following arguments:

```
username=root
IPMI_PASSWORD=
ipmitool -I lanplus -U$username-E-H BMC_HOSTNAME
```

Locate a USB Stick

Some systems print out which device is the USB, while other systems, such as Gigabyte-based systems, do not. Parsing the output of `efibootmgr` can be helpful in determining which device is the USB stick.

Use tools such as `lsblk`, `blkid`, or kernel (`/proc`) as well if possible. For example, match up `ls -l /dev/disk/by-partuuid` with `efibootmgr -v`.

Print off the UEFI's boot selections:

```
ncn-m# efibootmgr
BootCurrent: 0015
Timeout: 1 seconds
BootOrder: 000E,000D,0011,0012,0007,0005,0006,0008,0009,0000,0001,0002,000A,000B,000C,0003,0004,000F,0010,0013,0014
Boot0000* Enter Setup
Boot0001  Boot Device List
Boot0002  Network Boot
Boot0003* Launch EFI Shell
Boot0004* UEFI HTTPv6: Network 00 at Riser 02 Slot 01
Boot0005* UEFI HTTPv6: Intel Network 00 at Baseboard
Boot0006* UEFI HTTPv4: Intel Network 00 at Baseboard
Boot0007* UEFI IPv4: Intel Network 00 at Baseboard
Boot0008* UEFI IPv6: Intel Network 00 at Baseboard
Boot0009* UEFI HTTPv6: Intel Network 01 at Baseboard
Boot000A* UEFI HTTPv4: Intel Network 01 at Baseboard
Boot000B* UEFI IPv4: Intel Network 01 at Baseboard
Boot000C* UEFI IPv6: Intel Network 01 at Baseboard
Boot000D* UEFI HTTPv4: Network 00 at Riser 02 Slot 01
Boot000E* UEFI IPv4: Network 00 at Riser 02 Slot 01
Boot000F* UEFI IPv6: Network 00 at Riser 02 Slot 01
Boot0010* UEFI HTTPv6: Network 01 at Riser 02 Slot 01
Boot0011* UEFI HTTPv4: Network 01 at Riser 02 Slot 01
Boot0012* UEFI IPv4: Network 01 at Riser 02 Slot 01
Boot0013* UEFI IPv6: Network 01 at Riser 02 Slot 01
Boot0014* UEFI Samsung Flash Drive 1100
Boot0015* UEFI Samsung Flash Drive 1100
Boot0018* UEFI SAMSUNG MZ7LH480HAHQ-00005 S45PNA0M838871
Boot1001* Enter Setup
```

In the example above, the device is 0014 or 0015. Assuming 0014 is used, this value can be adjusted on-the-fly in a POST call. Notice the lack of "Boot" in the ID number returned. Boot0014 is wanted in this case, so 0014 needs to be passed to `efibootmgr`:

```
# Verify the BootNext device is what was selected:
ncn-m# efibootmgr -n 0014
ncn-m# efibootmgr | grep -i bootnext
BootNext: 0014
```

Now the UEFI Samsung Flash Drive will boot next.

23.21.1 NCN Networking 3

Non-compute nodes (NCNs) and compute nodes have different network interfaces. This section focuses on the network interfaces for NCNs.

The following table includes information about the different NCN network interfaces:

Table 5. NCN Network Interfaces

Name	Type	MTU
`mgmt0`	Slot 1 on the SMNET card.	9000
`mgmt1`	Slot 2 on the SMNET card, or slot 1 on the 2nd SMNET card.	9000
`bond0`	LACP Link Aggregate of `mgmt0` and `mgmt1`, or `mgmt0` and `mgmt2` on dual-bonds (when `bond1` is present).	9000
`bond1`	LACP Link Agg. of `mgmt1` and `mgmt3`.	9000
`lan0`	Externally facing interface.	1500
`lan1`	Externally facing interface, or anything (unused).	1500
`hsn0`	High-speed network interface.	9000
`hsnN+1`	High-speed network interface.	9000
`vlan002`	Virtual LAN for managing nodes.	1500
`vlan004`	Virtual LAN for managing hardware.	1500
`vlan007`	Virtual LAN for the Customer Access Network (CAN).	1500

The network interfaces can be observed on a live NCN with the following command:

```
ncn-w001# ip link
```

Device Naming

The MAC based `udev` rules set the interfaces during initial boot in iPXE. When a node boots, iPXE will dump the PCI buses and sort network interfaces into three buckets:

- `mgmt`: internal/management network connection
- `hsn`: high-speed connection
- `lan`: external/site-connection

Vendor and Bus ID Identification

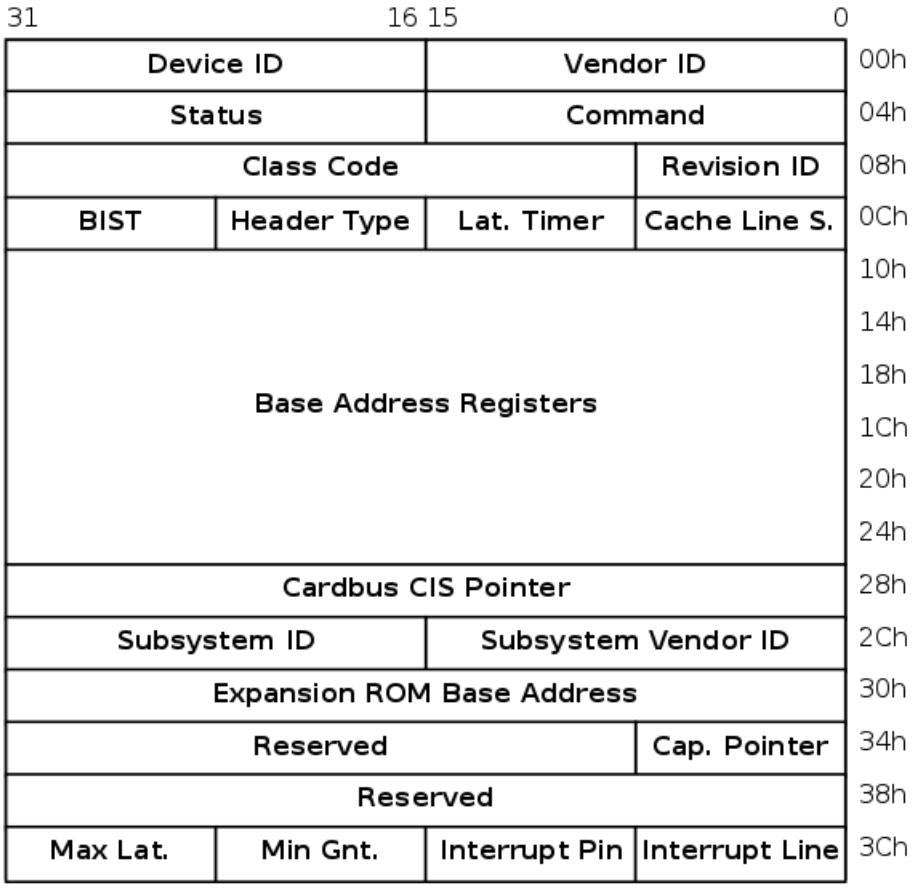
The initial boot of an NCN sets interface `udev` rules because it has no discovery method yet.

The following information is needed:

- PCI **Vendor** IDs for devices/cards to be used on the Management network.
- PCI **Device** IDs for the devices/cards to be used on the High-Speed Network.

The 16-bit Vendor ID is allocated by the Peripheral Component Interconnect Special Interest Group (PCI-SIG).

Figure 6. PCI Configuration Space



The figure above includes information belonging to the first 4 bytes of the PCI header, and the admin can obtain it using the `lspci` command, or another method for reading the PCI bus.

For example:

```
ncn-w001# lspci | grep -i ethernet
ncn-w001# lspci | grep c6:00.0
```

The device and vendor IDs are used in iPXE for bootstrapping the nodes, this allows generators to swap IDs out for certain systems until smarter logic can be added to cloud-init. The following table includes popular vendor and device IDs:

Table 6. Vendor and Device IDs

Vendor	Model	Device ID	Vendor ID
Intel Corporation	Ethernet Connection X722	37d2	8086
Intel Corporation	82576	1526	8086
Mellanox Technologies	ConnectX-4	1013	15b3
Mellanox Technologies	ConnectX-5	1017	15b3

Vendor	Model	Device ID	Vendor ID
Giga-Byte	Intel Corporation I350	1521	8086
QLogic Corporation	FastLinQ QL41000	8070	1077

23.21.1 NCN Mounts and File Systems 4

In general, there are three different kinds of disks:

Ephemeral	A disk that is not in the system's RAID.
RAID	A redundant array of independent disks (RAID).
Ceph	A disk for use as a Ceph OSD.

The table below represents all recognizable file system (FS) labels on any given NCN, varying slightly by node role.

Table 7. File System Overview

Kubernetes Master	Kubernetes Worker	Storage Ceph	FS Label	Partitions	Device	OverlayFS	Size on Disk
Yes	Yes	Yes	`BOOTRAID`	Not mounted	Two small disks in RAID-1	500 MiB	No
Yes	Yes	Yes	`SQFSRAID`	`/run/initramfs/live`	Two small disks in RAID-1	100 GiB	Yes
Yes	Yes	Yes	`ROOTRAID`	`/run/initramfs/overlayfs`	Two small disks in RAID-1	Max/ Remainder	Yes
No	Yes	No	`CONRUN`	`/run/containerd`	Ephemeral	75 GiB	No
No	Yes	No	`CONLIB`	`/var/lib/containerd`	Ephemeral	25%	Yes
Yes	No	No	`K8SETCD`	`/var/lib/etcd`	Ephemeral	32 GiB	No
No	No	No	`K8SKUBE`	`var/lib/kubelet`	Ephemeral	25%	Yes

OverlayFS and Persistence

There are a few overlays used for NCN image boots. These enable two critical functions:

- Changes to data and new data will persist between reboots
- RAM (memory) is freed because block-devices (SATA/PCIe) are used

The following is a list of the overlays used for NCN image boots:

ROOTRAID The persistent root OverlayFS that commits and saves all changes made to the running OS. It stands on a RAID-1 mirror.

CONLIB A persistent OverlayFS for containerd that commits and saves all new changes while allowing read-through to pre-existing (baked-in) data from the SquashFS.

K8SKUBE A persistent OverlayFS for kubelet that works exactly as the `CONLIB` OverlayFS.

The OverlayFS organization can be best viewed with these three commands:

- `lsblk`, `lsblk -f` - Shows how the RAIDs and disks are mounted
- `losetup -a` - Shows where the SquashFS is mounted
- `mount | grep ' / '` - Shows the overlay being layered atop the SquashFS

The `SQFSRAID` and `ROOTRAID` overlays contain the following:

- `/run/rootfsbase` is the SquashFS image itself
- `/run/initramfs/live` is the SquashFS's storage array, where one or more SquashFS can live
- `/run/initranfs/overlayfs` is the OverlayFS storage array, where the persistent directories live
- `/run/overlayfs` and `/run/ovlwork` are symlinks to `/run/initramfs/overlayfs/overlayfs-SQFSRAID-$(blkid -s UUID -o value /dev/disk/by-label/SQFSRAID)` and the neighboring work directory

Using the above bullets, one may be able to better understand the machine output below:

```
ncn-m001# mount | grep  ' / '
LiveOS_rootfs on / type overlay (rw,relatime,lowerdir=/run/rootfsbase,upperdir=/run/overlayfs,workdir=/run/ovlwork)
                                          ^^^R/O^SQUASHFS IMAGE^^^|^^^ R/W PERSISTENCE ^^^|^^^^^^INTERIM^^^^^^
                                          ^^^R/O^SQUASHFS IMAGE^^^|^^^ R/W PERSISTENCE ^^^|^^^^^^INTERIM^^^^^^
                                          ^^^R/O^SQUASHFS IMAGE^^^|^^^ R/W PERSISTENCE ^^^|^^^^^^INTERIM^^^^^^

ncn-m001# losetup -a
/dev/loop1: [0025]:56532 (/run/initramfs/thin-overlay/meta)
/dev/loop2: [0025]:49062 (/run/initramfs/thin-overlay/data)
/dev/loop0: [2431]:100 (/run/initramfs/live/LiveOS/ncn-m001.squashfs)
```

The thin overlay is the transient space the system uses behind the scenes to allow data to live in RAM as it's written to disk. Using thin overlays means the kernel will automatically clear free blocks.

The following is an example of a persistent system. This means that persistent capacity is there, but admins should beware of reset toggles on unfamiliar systems. There are toggles to reset overlays that are, by default, toggled `off`.

```
ncn-m001# lsblk -f
NAME                FSTYPE            LABEL         UUID                                   FSAVAIL FSUSE%
MOUNTPOINT
loop0               squashfs                                                                   0
100% /run/rootfsbase
loop1
└─live-overlay-pool
loop2
└─live-overlay-pool
sda
├─sda1
├─sda2              linux_raid_member ncn-m001:SQFS d10e163c-169c-c23c-e6de-ddd79fd32ddb
│ └─md127           xfs               SQFSRAID      08b549af-0246-4ec2-ad26-76fb45db73b1   89.3G
4% /run/initramfs/live
```

```
└─sda3              linux_raid_member ncn-m001:ROOT 1bde5a2a-30b6-cd9b-fee0-702e9306760d
  └─md126           xfs               ROOTRAID      67d38d40-a323-48cc-a724-b7431f180c6d  272.5G
2% /run/initramfs/overlayfs
sdb
├─sdb1
├─sdb2              linux_raid_member ncn-m001:SQFS d10e163c-169c-c23c-e6de-ddd79fd32ddb
│ └─md127           xfs               SQFSRAID      08b549af-0246-4ec2-ad26-76fb45db73b1   89.3G
4% /run/initramfs/live
└─sdb3              linux_raid_member ncn-m001:ROOT 1bde5a2a-30b6-cd9b-fee0-702e9306760d
  └─md126           xfs               ROOTRAID      67d38d40-a323-48cc-a724-b7431f180c6d  272.5G
2% /run/initramfs/overlayfs
sdc
└─sdc1
```

Persistent Directories

Not all directories are persistent. Only the following directories are persistent by default:

- `etc`
- `home`
- `root`
- `srv`
- `tmp`
- `var`
- `/run/containerd`
- `/run/lib-containerd`
- `/run/lib-kubelet`

More directories can be added, but mileage varies. The initial set is managed by dracut, and when using a reset toggle, the above list is "reset/cleared". If more directories are added, they will be eradicated when enabling a reset toggle.

These are all provided through the Overlay from `/run/overlayfs`. The `/run/overlayfs` directory is a symbolic link to the real disk `/run/initramfs/overlayfs/*`.

```
ncn-m001# cd /run/overlayfs
ncn-m001# ls -l
total 0
drwxr-xr-x 8 root root 290 Oct 15 22:41 etc
drwxr-xr-x 3 root root  18 Oct 15 22:41 home
drwx------ 3 root root  39 Oct 13 16:53 root
drwxr-xr-x 3 root root  18 Oct  5 19:16 srv
drwxrwxrwt 2 root root  85 Oct 16 14:50 tmp
drwxr-xr-x 8 root root  76 Oct 13 16:52 var
```

Layering - Upperdir and Lowerdir

The file-system the user is working on is essentially two layered file-systems (overlays). Anything in the upper-layer takes precedence by default.

- The lower layer is the SquashFS image itself, read-only, which provides all that we need to run
- The upper layer is the OverlayFS, read-write, which does a bit-wise `xor` with the lower-layer

There are more advanced options for overlays, such as multiple lower-layers, copy-up (lower-layer precedence), and opaque (removing a directory in the upper layer hides it in the lower layer). Refer to *https://www.kernel.org/doc/html/latest/filesystems/overlayfs.html#inode-properties* for more information.

Layering Examples

The following examples show the `/root` directory. The upper-dir has the following files:

```
ncn-m001# ll /run/overlayfs/root/
total 4
-rw------- 1 root root 252 Nov  4 18:23 .bash_history
drwxr-x--- 4 root root  37 Nov  4 04:35 .kube
drwx------ 2 root root  29 Oct 21 21:57 .ssh
```

Then, in the SquashFS image (lower-dir), the following files are included:

```
ncn-m001# ll /run/rootfsbase/root/
total 1
-rw------- 1 root root   0 Oct 19 15:31 .bash_history
drwxr-xr-x 2 root root   3 May 25  2018 bin
drwx------ 3 root root  26 Oct 21 22:07 .cache
drwx------ 2 root root   3 May 25  2018 .gnupg
drwxr-xr-x 4 root root  57 Oct 19 15:23 inst-sys
drwxr-xr-x 2 root root  33 Oct 19 15:33 .kbd
drwxr-xr-x 5 root root  53 Oct 19 15:34 spire
drwx------ 2 root root  70 Oct 21 21:57 .ssh
-rw-r--r-- 1 root root 172 Oct 26 15:25 .wget-hsts
```

The `.bash_history` file in the lower-dir is `0` bytes, but it's `252` bytes in the upperdir. Additionally, the `.kube` dir exists in the upper, but not the lower. Looking at the `/root` directory helps explains these differences:

```
ncn-m001# ll /root
total 5
-rw------- 1 root root 252 Nov  4 18:23 .bash_history
drwxr-xr-x 2 root root   3 May 25  2018 bin
drwx------ 3 root root  26 Oct 21 22:07 .cache
drwx------ 2 root root   3 May 25  2018 .gnupg
drwxr-xr-x 4 root root  57 Oct 19 15:23 inst-sys
drwxr-xr-x 2 root root  33 Oct 19 15:33 .kbd
drwxr-x--- 4 root root  37 Nov  4 04:35 .kube
drwxr-xr-x 5 root root  53 Oct 19 15:34 spire
drwx------ 1 root root  29 Oct 21 21:57 .ssh
-rw-r--r-- 1 root root 172 Oct 26 15:25 .wget-hsts
```

The `.bash_history` now matches the upper-dir, and `.kube` now exists in both places. This indicates that any change done to `/root/` will persist through `/run/overlayfs/root` and will take precedence to the SquashFS image root.

OverlayFS Features

The features or toggles for OverlayFS are passable on the kernel command line, and they change the behavior of the OverlayFS. The OverlayFS provides a few reset toggles that are used to clear out the persistence directories without requiring a reinstall. However, these toggles do require rebooting.

Reset on Next Boot

The preferred way to reset persistent storage is to use the OverlayFS reset toggle. Modify the boot command line on the PXE server, adding the following:

```
# Reset the overlay on boot
rd.live.overlay.reset=1
```

Once reset, enable persistence again. Simply revert the change and the next reboot will persist.

```
# Cease resetting the overlayFS
rd.live.overlay.reset=0
```

Reset on Every Boot

There are two options that are used to reset on every boot:

- `rd.live.overlay.reset=1` will eradicate/recreate the overlay every reboot
- `rd.live.overlayr.readonly=1` will clear the overlay on every reboot

For long-term usage, `rd.live.overlay.readonly=1` should be added to the command line. The `reset=1` toggle is usually used to fix a side-ways overlay. To completely refresh and purge the overlay, use `rd.live.overlay.reset`.

```
# Authorize METAL to purge
metal.no-wipe=0 rd.live.overlay.reset=1
```

`metal.no-wipe=1` does not protect against `rd.live.overlay.reset`, and `metal.no-wipe` is not a feature of dmsquash-live.

Re-sizing the Persistent Overlay

The overlay can be resized to fit a variety of needs or use cases. The size is provided directly on the command line. Any value can be provided, but it must be in megabytes.

- Default Size: 300 GiB
- File System: XFS

If resetting the overlay on a deployed node, `rd.live.overlay.reset=1` needs to be set. It is recommended to set the size before deployment. There is a linkage between the metal-dracut module and the live-module that makes this inflexible.

```
# Use a 300 GiB overlayFS (default)
rd.live.overlay.size=307200

# Use a 1 TiB overlayFS
rd.live.overlay.size=1000000
```

Thin Overlay Feature

The persistent OverlayFS leverages newer, "thin" overlays that support discards and that will free blocks that are not claimed by the file system. This means that memory is free/released when the filesystem does not claim it anymore. Thin overlays can be disabled, and instead classic DM Snapshots can be used to manage the overlay. This will use more RAM. It is not recommended, since dmraid is not included in the initrd.

```
# Enable (default)
rd.live.overlay.thin=1

# Disable (not recommended; undesirable RAM waste)
rd.live.overlay.thin=0
```

23.21.1 NCN Packages 5

Each type of non-compute node (NCN) has packages of images stored on the node. Lists of these images are generated on running nodes. A list of images can be collected by running a `zypper` command on one of the storage, worker, or master nodes.

Node images for NCNs acting as Kubernetes master/worker nodes can be collected with the following command:

```
ncn-w002# zypper --disable-repositories se --installed-only \
| grep i+ | tr -d '|' | awk '{print $2}'
```

Node images for Ceph-based storage NCNs can be collected with the following command:

```
ncn-s002# zypper --disable-repositories se --installed-only \
| grep i+ | tr -d '|' | awk '{print $2}'
```

23.21.1 NCN Operating System Release 6

Non-compute nodes (NCNs) define their product's per image layer.

- Metal SquashFS are always SLE_HPC (SUSE High Performance Computing)
- Metal Ceph storage images are always SLE_HPC (SUSE High Performance Computing) with SES (SUSE Enterprise Storage)

The sles-release RPM is uninstalled for Metal, and instead, the sle_HPC-release RPM is installed. These both provide the same files, but differ for `os-release` and `/etc/product.d/baseproduct`. The ses-release RPM is installed on top of the sle_HPC-release RPM in the Ceph images.

The following example shows the two product files for a Ceph node. There is a metal node that is capable of high-performance computing and serving enterprise storage.

```
ncn-s001# ll /etc/products.d/
total 5
lrwxrwxrwx 1 root root   12 Jan  1 06:43 baseproduct -> SLE_HPC.prod
-rw-r--r-- 1 root root 1587 Oct 21 15:27 ses.prod
-rw-r--r-- 1 root root 2956 Jun 10  2020 SLE_HPC.prod
ncn-s001:~ # grep '<summary' /etc/products.d/*.prod
/etc/products.d/ses.prod:  <summary>SUSE Enterprise Storage 7</summary>
/etc/products.d/SLE_HPC.prod:  <summary>SUSE Linux Enterprise High Performance
Computing 15 SP2</summary>
```

Kubernetes nodes will report SLES HPC only, which is reflected in the `kubectl` output below:

```
pit# kubectl get nodes -o wide
NAME       STATUS   ROLES    AGE    VERSION   INTERNAL-IP   EXTERNAL-IP   OS-
IMAGE                                                KERNEL-VERSION
CONTAINER-RUNTIME
ncn-m002   Ready    master   128m   v1.18.6   10.252.1.14   <none>        SUSE
Linux Enterprise High Performance Computing 15 SP2   5.3.18-24.37-default
containerd://1.3.4
ncn-m003   Ready    master   127m   v1.18.6   10.252.1.13   <none>        SUSE
Linux Enterprise High Performance Computing 15 SP2   5.3.18-24.37-default
containerd://1.3.4
ncn-w001   Ready    <none>   90m    v1.18.6   10.252.1.12   <none>        SUSE
Linux Enterprise High Performance Computing 15 SP2   5.3.18-24.37-default
containerd://1.3.4
ncn-w002   Ready    <none>   88m    v1.18.6   10.252.1.11   <none>        SUSE
Linux Enterprise High Performance Computing 15 SP2   5.3.18-24.37-default
containerd://1.3.4
ncn-w003   Ready    <none>   82m    v1.18.6   10.252.1.10   <none>        SUSE
Linux Enterprise High Performance Computing 15 SP2   5.3.18-24.37-default
containerd://1.3.4
```

23.21.1 NCN and Management Node Locking 7

In release 1.4, the ability to ignore non-compute nodes (NCNs) is turned off by default. Management nodes and NCNs are also not locked by default. The administrator must lock the NCNs to prevent unwanted actions from affecting these nodes.

This section only covers using locks with the Hardware State Manager (HSM). For more information on ignoring nodes, refer to the following sections:

- Firmware Action Service (FAS): Refer to "Ignore Node within FAS" in the *HPE Cray EX Hardware Management Administration Guide S-8015*
- Cray Advanced Platform Monitoring and Control (CAPMC): Refer to "Ignore Node with CAPMC" in the *HPE Cray EX Hardware Management Administration Guide S-8015*

The following actions can be prevented when NCNs are locked.

- Firmware upgrades with FAS
- Power off operations with CAPMC
- Reset operations with CAPMC

Doing any of these actions by accident will shut down an NCN. If the NCN is a Kubernetes master or worker node, this can have serious negative effects on system operations. If a single node is taken down by mistake, it is possible that services will recover. If all NCNs are taken down, or all Kubernetes workers are taken down by mistake, the system must be restarted.

After critical nodes are locked, power/reset (CAPMC) or firmware (FAS) operations cannot affect the nodes unless they are unlocked. For example, any locked node that is included in a list of nodes to be reset will result in a failure.

When to Lock Nodes

To best protect system health, NCNs should be locked as early as possible in the install/upgrade cycle. The later in the process, the more risk there is of accidentally taking down a critical node. NCN locking must be done after Kubernetes is running and the HSM service is operational.

Check if Kubernetes and HSM are running with the following command:

```
linux# kubectl -n services get pods | grep smd
cray-smd-848bcc875c-6wqsh           2/2     Running    0          9d
cray-smd-848bcc875c-hznqj           2/2     Running    0          9d
cray-smd-848bcc875c-tp6gf           2/2     Running    0          6d22h
cray-smd-init-2tnnq                 0/2     Completed  0          9d
cray-smd-postgres-0                 2/2     Running    0          19d
cray-smd-postgres-1                 2/2     Running    0          6d21h
cray-smd-postgres-2                 2/2     Running    0          19d
cray-smd-wait-for-postgres-4-7c78j  0/3     Completed  0          9d
```

The *`cray-smd-xxx`* pods need to be in the *`Running`* state.

When to Unlock Nodes

Any time a Management node or NCN has to be power cycled, reset, or have its firmware updated, it will first need to be unlocked. After the operation is complete, the targeted nodes should once again be locked.

See below for instructions and examples.

Lock Management NCNs

Use the standard HSM CLI to perform locking.

To lock all nodes with the *Management* role:

```
linux# cray hsm locks lock create --role Management --processing-model rigid
Failure = []

[Counts]
Total = 8
Success = 8
Failure = 0

[Success]
ComponentIDs = [ "x3000c0s5b0n0", "x3000c0s4b0n0", "x3000c0s7b0n0", "x3000c0s6b0n0", "x3000c0s3b0n0",
"x3000c0s2b0n0", "x3000c0s9b0n0", "x3000c0s8b0n0",]
```

To lock single nodes or lists of specific nodes:

```
linux# cray hsm locks lock create --role Management \
--component-ids NODE_XNAME --processing-model rigid
Failure = []

[Counts]
Total = 1
Success = 1
Failure = 0

[Success]
ComponentIDs = [ "x3000c0s6b0n0",]
```

Unlock Management NCNs

The HSM CLI commands can also be used to unlock nodes.

To unlock all nodes with the *Management* role:

```
linux# cray hsm locks unlock create --role Management --processing-model rigid
Failure = []

[Counts]
Total = 8
Success = 8
Failure = 0

[Success]
ComponentIDs = [ "x3000c0s7b0n0", "x3000c0s6b0n0", "x3000c0s3b0n0", "x3000c0s2b0n0", "x3000c0s9b0n0",
"x3000c0s8b0n0", "x3000c0s5b0n0", "x3000c0s4b0n0",]
```

To unlock a single node or lists of nodes:

```
linux# cray hsm locks unlock create --role Management \
--component-ids x3000c0s6b0n0 --processing-model rigid
Failure = []

[Counts]
Total = 1
Success = 1
Failure = 0

[Success]
ComponentIDs = [ "x3000c0s6b0n0",]
```

23.21.1 Firmware Updates 8

Firmware and BIOS updates may be necessary before an install can start. During runtime, firmware is upgraded using the Firmware Action Service (FAS).

New systems, or systems upgrading from prior versions of the HPE Cray EX system, must meet the minimum requirements defined in the following sections:

- Node requirements: *Node Firmware* on page 244
- Network requirements: *Network Firmware* on page 246

Only network devices and non-compute nodes (NCNs) can upgrade firmware prior to a 1.4 install or upgrade. Other devices, such as compute nodes, provision upgrades from FAS.

FAS tracks and performs actions (upgrade, downgrade, restore and create snapshots) on system firmware. FAS is a runtime service deployed in Kubernetes. Fresh installs on bare-metal use FAS for upgrading compute node firmware.

LiveCD Availability for Bootstrap

The LiveCD serves firmware for bootstrapping devices and enables a CRAY install. Devices can use SCP or HTTP to retrieve firmware from the LiveCD USB stick, or from the remote ISO. Any interface of the LiveCD may be used in place of the following DNS names.

- http://pit/fw/
- http://pit.nmn/fw/
- http://pit.hmn/fw/
- http://pit.can/fw/
- `scp://<username>:<password>@pit/var/www/fw/`

23.21.18 Node Firmware .1

This sections describes the minimum specifications for nodes and their components, such as PCIe cards.

Servers

The following are supported servers by vendor, model, and version:

Table 8. Servers

Vendor	Model	Version
HPE	A41 DL325 Gen10	10/18/2019 2.30
HPE	A42 DL385 Gen10+	10/31/2020 1.38
HPE	A43 DL325 Gen10+	10/31/2020 1.38
Intel	S2600WFT	02.01.0012
Gigabyte	MZ32-AR0-00	12.84

HPE (iLO) Upgrades

There are no instructions yet, this is place-holder.

Intel Upgrades

There are no instructions yet, this is place-holder.

Gigabyte Upgrades

There are no instructions yet, this is place-holder.

PCIe Cards

The following are supported PCIe card vendors and models:

Table 9. PCIe Cards

Vendor	Model	PSID	Version
Marvell	QL41232HQCU-HC		08.50.78
Mellanox	MCX416A-BCAT	MT_2130111027	12.28.2006
Mellanox	MCX515A-CCAT	MT_0000000011	16.28.4000
Mellanox	MCX515A-CCAT	MT_0000000591	16.28.4000

The Mellanox firmware can be updated to minimum specification using `mlxfwmanager`. The `mlxfwmanager` command will fetch updates from online, a local file, or a local web server, such as `http://pit/`.

Marvell Upgrades

There are no instructions yet, this is place-holder.

Mellanox Upgrades

To start the Mellanox Software Tools (MST) and enable the tools for upgrades:

```
# mst status
Starting MST (Mellanox Software Tools) driver set
Loading MST PCI module - Success
Loading MST PCI configuration module - Success
Create devices
Unloading MST PCI module (unused) - Success
```

To print out the current firmware versions for all Mellanox cards:

```
# mlxfwmanager
Querying Mellanox devices firmware ...

Device #1:
----------

  Device Type:      ConnectX5
  Part Number:      MCX515A-CCA_Ax
  Description:      ConnectX-5 EN network interface card; 100GbE single-port QSFP28; PCIe3.0 x16; tall
bracket; ROHS R6
  PSID:             MT_0000000011
  PCI Device Name:  /dev/mst/mt4119_pciconf0
  Base GUID:        ec0d9a03007da71e
  Base MAC:         ec0d9a7da71e
  Versions:         Current        Available
```

```
     FW             16.26.4012     N/A
     PXE            3.5.0805       N/A
     UEFI           14.19.0017     N/A

  Status:           No matching image found

Device #2:
----------

  Device Type:      ConnectX4
  Part Number:      MCX416A-BCA_Ax
  Description:      ConnectX-4 EN network interface card; 40GbE dual-port QSFP28; PCIe3.0 x16; ROHS R6
  PSID:             MT_2130111027
  PCI Device Name:  /dev/mst/mt4115_pciconf0
  Base GUID:        506b4b030013982e
  Base MAC:         506b4b13982e
  Versions:         Current        Available
     FW             12.26.4012     N/A
     PXE            3.5.0805       N/A
     UEFI           14.19.0017     N/A

  Status:           No matching image found
```

If the LiveCD is reachable, firmware can be downloaded for local install:

```
# curl -O http://pit/fw/pcie/firmware.img
mlxfwmanager -u -i ./firmware.img
```

If external queries can be made by the node, it can update firmware from the internet:

```
# mlxfwmanager -u --online
```

23.21.18 Network Firmware .2

Management switch firmware needs to be updated prior to the install as well. Refer to the following table for more information about each type of management switch.

Table 10. Management Switches

Vendor	Model	Version
Aruba	6300	ArubaOS-CX_6400-6300_10.06.0010
Aruba	8320	ArubaOS-CX_8320_10.06.0010
Aruba	8325	ArubaOS-CX_8325_10.06.0010
Dell	S3048-ON	10.5.1.4
Dell	S4148F-ON	10.5.1.4
Dell	S4148T-ON	10.5.1.4
Mellanox	MSN2100	3.9.1014
Mellanox	MSN2700	3.9.1014

23.21.1 Collect BMC MAC Addresses 9

Prerequisites

- There is a configured switch with SSH access or un-configured with COM access (serial-over-lan/DB-9)
- A file is set up to record the collected BMC information

About this task

ROLE	System administrator
OBJECTIVE	Collect BMC MAC addresses from a racked HPE Cray EX system that has had its switches configured.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Establish an SSH or serial connection to the leaf switch.

   The following commands are examples and the IP addresses will vary. The 10.X.02 value may or may not match the setup on the system being used.

   To SSH over metal management:

   ```
   pit# ssh admin@10.1.0.2
   ```

   To SSH over node management:

   ```
   pit# ssh admin@10.252.0.2
   ```

   To SSH over hardware management:

   ```
   pit# ssh admin@10.254.0.2
   ```

   To establish a serial connection:

   ```
   pit# minicom -b 115200 -D /dev/tty.USB1
   ```

2. View the MAC addresses for the ports of the BMCs.

   If the ports exist on the same VLAN, dump the VLAN to view the MAC addresses. If the VLAN ID is known, use the following command:

   ```
   # DellOS 10
   sw-smn01# show mac address-table vlan 4 | except 1/1/52
   VlanId  Mac Address     Type        Interface
   4   00:1e:67:98:fe:2c   dynamic     ethernet1/1/11
   4   a4:bf:01:38:f0:b1   dynamic     ethernet1/1/27
   4   a4:bf:01:38:f1:44   dynamic     ethernet1/1/25
   4   a4:bf:01:48:1e:ac   dynamic     ethernet1/1/28
   4   a4:bf:01:48:1f:70   dynamic     ethernet1/1/31
   4   a4:bf:01:48:1f:e0   dynamic     ethernet1/1/26
   4   a4:bf:01:48:20:03   dynamic     ethernet1/1/30
   4   a4:bf:01:48:20:57   dynamic     ethernet1/1/29
   4   a4:bf:01:4d:d9:9a   dynamic     ethernet1/1/32
   ```

If only the interface and trunk is known:

```
# DellOS 10
sw-smn01# show mac address-table interface ethernet 1/1/32
VlanId  Mac Address     Type        Interface
4   a4:bf:01:4d:d9:9a   dynamic     ethernet1/1/32
```

To print everything:

```
# DellOS 10
sw-smn01# show mac address-table
VlanId  Mac Address     Type        Interface
4   a4:bf:01:4d:d9:9a   dynamic     ethernet1/1/32
....

# Onyx & Aruba
sw-smn01# show mac-address-table
```

3. Select the initial eight BMCs from the output returned in the previous step.

   Ignore the managers that have external BMC connections when selecting the eight BMCs.

```
Xname,Role,Subrole,BMC MAC,Bootstrap MAC,Bond0 MAC0,Bond0 MAC1
x3000c0s9b0n0,Management,Storage,94:40:c9:37:77:26,94:40:c9:5f:b5:de,94:40:c9:5f:b5:de,14:02:ec:da:b9:98
                                 ^^^^^^^^^^^^^^^^^
```

   The column heading must match that shown above for CSI to correctly parse it.

Proceed to "Collect NCN MAC Addresses" in the *HPE Cray EX System Administration Guide S-8001*.

23.21.2 Collect NCN MAC Addresses 0

Prerequisites

- LiveCD dnsmasq is configured for the bond0/metal network.

  The Node Management Network (NMN), Hardware Management Network (HMN), and Customer Access Network (CAN) are not relevant in this case.

- BMC MACs already collected.

  Refer to "Collect BMC MAC Addresses" in the *HPE Cray EX System Administration Guide S-8001*.

- LiveCD ConMan is configured for each BMC.

  Use the `conman -q` command to see the consoles.

About this task

ROLE	System administrator
OBJECTIVE	Collect the NCN MAC addresses from a racked HPE Cray EX system. The MAC addresses are needed for the `ncn_metadata.csv` file's BMC column. The easy way to do this leverages the NIC-dump provided by the metal-ipxe package. This procedure will run-through booting NCNs and collecting their MACs from the conman console logs.

	The alternative is to use serial cables (or SSH) to collect the MACs from the switch ARP tables. This can become exponentially difficult for large systems. If this is the only way, please proceed to the *Serial Consoles* section.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

iPXE Consoles

This procedure is faster for those with the LiveCD. It can be used to quickly boot-check nodes to dump network device information without an OS. This works by accessing the PCI configuration space.

1. Optional: Move the iPXE script.

   Moving the iPXE script will prevent network booting. However, if disk booting occurs it will prevent the nodes from continuing to boot and end in undesired states.

   ```
   pit# mv /var/www/boot/script.ipxe /var/www/boot/script.ipxe.bak
   ```

2. Verify the consoles are active.

   ```
   pit# conman -q
   ncn-m002-mgmt
   ncn-m003-mgmt
   ncn-s001-mgmt
   ncn-s002-mgmt
   ncn-s003-mgmt
   ncn-w001-mgmt
   ncn-w002-mgmt
   ncn-w003-mgmt
   ```

3. Set the nodes to PXE boot and restart them.

   ```
   export username=root
   export IPMI_PASSWORD=
   grep -oE "($mtoken|$stoken|$wtoken)" /etc/dnsmasq.d/statics.conf | xargs -t -i ipmitool -I lanplus -
   U $username -E -H {} chassis bootdev pxe options=efiboot,persistent
   grep -oE "($mtoken|$stoken|$wtoken)" /etc/dnsmasq.d/statics.conf | xargs -t -i ipmitool -I lanplus -
   U $username -E -H {} power off        if ipmitool -I lanplus -U $username -P $password -H $node
   power status =~ 'off' ; then
           ipmitool -I lanplus -U $username -P $password -H $node power on
       else
           ipmitool -I lanplus -U $username -P $password -H $node power reset
       fi
   done
   ```

4. Wait for the nodes to netboot.

   Follow the status of the nodes with `conman -j ncn-*id*-mgmt`. This takes less than 3 minutes. The speed depends on how quickly the nodes POST.

   ```
   pit# conman -j ncn-*id*-mgmt
   ```

5. Print off what has been found in the console logs.

   This snippet will omit duplicates from multiple boot attempts.

   ```
   for file in /var/log/conman/*; do
       echo $file
       grep -Eoh '(net[0-9] MAC .*)' $file | sort -u | grep PCI && echo -----
   done
   ```

6. Retrieve the two MACs to use for bond0, and the two more to use for bond1 based on the topology being used.

   This information can be found in the output of the previous step.

   Worker nodes also have the high-speed network cards. If the cards are known, filter their device IDs out from the above output using the following :

   ```
   unset did # clear it if it was used it.
   did=1017 # ConnectX-5 example.
   for file in /var/log/conman/*; do
       echo $file
       grep -Eoh '(net[0-9] MAC .*)' $file | sort -u | grep PCI | grep -Ev "$did" && echo -----
   done
   ```

   To filter out on-board NICs or site-link cards, omit their device IDs as well. Use the above snippet, but add the other IDs. This snippet prints out only mgmt MACs. The `did` is the HSN and on-board NICs that are being ignored.

   ```
   unset did # clear it if it was used it.
   did='(1017|8086|ffff)'
   for file in /var/log/conman/*; do
       echo $file
       grep -Eoh '(net[0-9] MAC .*)' $file | sort -u | grep PCI | grep -Ev "$did" && echo -----
   done
   ```

7. Examine the output from `grep`. Use the lowest value MAC address per PCIe card.

   The following example uses one PCIe card with two ports for a total of two ports per node.

   ```
   -----
   /var/log/conman/console.ncn-w003-mgmt
   net2 MAC b8:59:9f:d9:9e:2c PCI.DeviceID 1013 PCI.VendorID 15b3 <-bond0-mac0 (0x2c < 0x2d)
   net3 MAC b8:59:9f:d9:9e:2d PCI.DeviceID 1013 PCI.VendorID 15b3 <-bond0-mac1
   -----
   ```

   This example uses two PCIe cards with two ports each for a total of four ports per node.

   ```
   -----
   /var/log/conman/console.ncn-w006-mgmt
   net0 MAC 94:40:c9:5f:b5:df PCI.DeviceID 8070 PCI.VendorID 1077 <-bond0-mac0 (0x38 < 0x39)
   net1 MAC 94:40:c9:5f:b5:e0 PCI.DeviceID 8070 PCI.VendorID 1077 (future use)
   net2 MAC 14:02:ec:da:b9:98 PCI.DeviceID 8070 PCI.VendorID 1077 <-bond0-mac1 (0x61f0 < 0x7104)
   net3 MAC 14:02:ec:da:b9:99 PCI.DeviceID 8070 PCI.VendorID 1077 (future use)
   -----
   ```

   The examples above identified MAC0 and MAC1 of the bond as 14:02:ec:df:9c:38 and 94:40:c9:c1:61:f0 respectively.

   ```
   Xname,Role,Subrole,BMC MAC,Bootstrap MAC,Bond0 MAC0,Bond0 MAC1
   x3000c0s9b0n0,Management,Worker,94:40:c9:37:77:26,14:02:ec:df:9c:38,14:02:ec:df:9c:38,94:40:c9:c1:61:f0
                                                     ^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^
   ```

Serial Consoles

8. Proceed to the "Collect BMC MAC Addresses" procedure in the *HPE Cray EX System Administration Guide S-8001*.

Pick out the MACs for the BOND from each the spine01 and spine02 switch. A PCIe card with dual-heads may go to either spine switch, meaning MAC0 must be collected from spine-01. Please refer to the cabling diagram or the actual rack.

Follow the procedure on each spine switch that port1 and port2 of the bond is plugged into.

23.21.2 NCN Metadata over USB-Serial 1

It is recommended to use the Serial/COM ports on the spine and leaf switches in the event that network plumbing is lacking, down, or not configured for procuring devices. Admins will need to procure MAC addresses for the non-compute node (NCN) metadata files with the serial console. The BAUDRATE and `terminal` usage vary per manufacturer.

The following are the manufacturers with links to their documentation:

- *Aruba*
- *Dell*
- *Mellanox*

Setup and Connection

The `minicom`, `screen`, and `cu` utilities can be used to connect to the switch consoles. At this time, it is assumed a USB-DB-9 or USB-RJ-45 cable is connected between the switch and the NCN.

To use the `screen` utility:

```
# screen /dev/ttyUSB1
# screen /dev/ttyUSB1 115200
```

To use the `minicom` utility:

```
# minicom -b 9600 -D /dev/ttyUSB1
# minicom -b 115200 -D /dev/ttyUSB1
```

To use the `cu` utility:

```
# cu -l /dev/ttyUSB1 -s 115200
```

Debugging Connections

On Mellanox switches, if the console is not responding when opened, try holding **CTRL** + **R** (or **control** + **R** for macOS) to initiate a screen refresh. This should take 5-10 seconds.

If the USB TTY device is not showing up in `/dev/tty*`, follow `dmesg -w` and try reseating the USB cable by unplugging the end in the NCN and plugging it back in. Observe the `dmesg -w` output to see if it shows errors pertaining to USB. The cable may be bad or a reboot may be needed.

23.21.2 Netboot an NCN from a Spine Switch 2

Prerequisites

The system in use has on-board NICs.

About this task

ROLE	System administrator
OBJECTIVE	Migrate NCNs for PXE booting and boot them over the spine switches. The migration process will vary depending on the on-board NICs in use. Systems with on-board connections to their leaf switches and NCNs need to disable or remove that connection. NCNs use bond interfaces and spine switches for the Node Management Network (NMN), Hardware Management Network (HMN), and Customer Access Network (CAN). However, older systems might have a connection to their leaf switches and solely use it for PXE booting. This NIC is not used during runtime, and NCNs in this state should enable PXE within their PCIe devices' OpROMs and disable or remove this on-board connection.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Enable UEFI PXE mode.

   On any NCN using 0.0.10 Kubernetes, 0.0.8 Ceph, and anything built on ncn-0.0.21 or higher can run this to begin interacting with Mellanox cards.

   ```
   ncn-w001# mst start
   ```

   After running this command, `mst status` and other commands like `mlxfwmanager` or `mlxconfig` will work, and devices required for these commands will be created in `/dev/mst`.

   Troubleshooting: If recovering NCNs with an earlier image that doesn't include the Mellanox tools, obtain them directly through Mellanox with the following commands:

   ```
   linux# wget https://www.mellanox.com/downloads/MFT/mft-4.15.1-9-x86_64-rpm.tgz
   linux# tar -xzvf mft-4.15.1-9-x86_64-rpm.tgz
   linux# cd mft-4.15.1-9-x86_64-rpm/RPMS
   linux# rpm -ivh ./mft-4.15.1-9.x86_64.rpm
   linux# cd
   linux# mst start
   ```

2. Print the current UEFI and SR-IOV states.

   a. Print out the device name and current UEFI PXE state.

   ```
   # Print name and current state; on an NCN or on the liveCD.
   mst status
   for MST in $(ls /dev/mst/*); do
       mlxconfig -d ${MST} q | egrep "(Device|EXP_ROM|SRIOV_EN)"
   done
   ```

b. Enable and dump the UEFI PXE state.

The Mellanox will be configured for PXE booting after this step.

```
# Set UEFI to YES
for MST in $(ls /dev/mst/*); do
    echo ${MST}
    mlxconfig -d ${MST} -y set EXP_ROM_UEFI_x86_ENABLE=1
    mlxconfig -d ${MST} -y set EXP_ROM_PXE_ENABLE=1
    mlxconfig -d ${MST} -y set SRIOV_EN=0
    mlxconfig -d ${MST} q | egrep "EXP_ROM"
done
```

The following kernel modules for Qlogic are installed:

- qlgc-fastlinq-kmp-default
- qlgc-qla2xxx-kmp-default

3. Disable or remove on-board connections.

If this physical connection can be removed, this is the preferred method and can be done after enabling PXE on the PCIe cards.

If the connection can only be disabled, refer to the following substeps:

a. Connect to the leaf switch using serial or SSH connections.

Select one of the connection options below. The IP addresses and device names may vary in the commands below.

```
# SSH over METAL MANAGEMENT
pit# ssh admin@10.1.0.2

# SSH over NODE MANAGEMENT
pit# ssh admin@10.252.0.2

# SSH over HARDWARE MANAGEMENT
pit# ssh admin@10.254.0.2

# serial (device name will vary)
pit# minicom -b 115200 -D /dev/tty.USB1
```

b. Enter configuration mode.

```
$> configure terminal
(config)#>
```

c. Disable the NCN interfaces.

Check the SHCD for reference before continuing.

```
(config)#> interface range 1/1/2-1/1/10
(config)#> shutdown
(config)#> write memory
```

The NCN interfaces can be enabled again by using the `no shutdown` command instead of the `shutdown` command.

23.21.2 Construct the Switch Metadata File 3

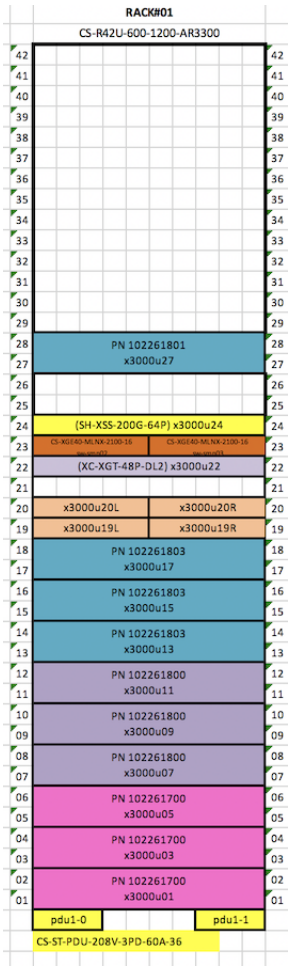
The `switch_metadata.csv` file contains all of the metadata for the various switches used in the HPE Cray EX system. This file needs to be manually created, and uses the following format:

```
Switch Xname,Type,Brand
x3000c0w18,Leaf,Dell
x3000c0h19s1,Spine,Mellanox
x3000c0h19s2,Spine,Mellanox
```

In order to construct this file, the SHCD is required to help map names in the `switch_metadata.csv` file. Use the SHCD to identify the switches in use. Look for the following indicators:

- The slot number(s) for the leaf switches (usually 48-port switches)
  - In the figure below, this is x3000u22
- The slot number(s) for the spine switches
  - In the figure below, this is x3000u23R and x3000u23L (two side-by-side switches)
  - Newer side-by-sides use slot numbers instead of R and L

Figure 7. SHCD Rack Example



Each spine/aggregation switch follows the `xXcChHsS` format:

- xX : where "X" is the river rack identification number (the figure above is "3000")
- cC : where "C" is the cabinet identification number (the figure above is "0")
- hH : where "H" is the slot number in the rack (height)
- sS : where "S" is the horizontal space number'

Each leaf switch follows the `xXcCwW` format:

- xX : where "X" is the river rack identification number (the figure above is "3000")
- cC : where "C" is the cabinet identification number (the figure above is "0")
- wW : where "W" is the slot number in the rack (height)

Each item in the file is either of type `Aggregate`, `CDU`, `Leaf`, or `Spine`. Each line in the file must denote the brand, which is either `Dell`, `Mellanox`, or `Aruba`.

To find the model, log into the switch and run one of the following commands:

- Dell: `show system`
- Mellanox: `show inventory`
- Aruba: `show system`

The following is an example file for Dell and Mellanox switches:

```
pit# cat example_switch_metadata.csv
Switch Xname,Type,Brand
x3000c0w38,Leaf,Dell
x3000c0w36,Leaf,Dell
x3000c0h33s1,Spine,Mellanox
x3000c0h33s2,Spine,Mellanox
```

The following is an example file for Aruba switches:

```
pit# cat example_switch_metadata.csv
Switch Xname,Type,Brand
x3000c0w14,Leaf,Aruba
x3000c0h12s1,Spine,Aruba
x3000c0h13s1,Spine,Aruba
```

23.21.2 Add UAN Aliases to SLS

4

Prerequisites

- The System Layout Service (SLS) is up and running and has been populated with data.
- Access to the API gateway `api-gw-service-nmn.local`.

About this task

ROLE	System administrator
OBJECTIVE	Manually add aliases to User Access Nodes (UANs) in SLS.

	Steps 3 and 4 of this procedure can be repeated for each UAN alias that needs to be added in SLS. This procedure is intended to be ran on any Kubernetes node that has access to the API gateway `api-gw-service-nmn.local`.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Authenticate with Keycloak to obtain an API token.

```
ncn-w001# export TOKEN=$(curl -k -s -S -d grant_type=client_credentials \
-d client_id=admin-client \
-d client_secret=`kubectl get secrets admin-client-auth -o jsonpath='{.data.client-secret}' | base64
-d` \
https://api-gw-service-nmn.local/keycloak/realms/shasta/protocol/openid-connect/token \
| jq -r '.access_token')
```

2. Find the xname of the UAN by searching through all Application nodes until it is found.

   The following command will return an array of Application nodes currently known in SLS.

```
ncn-w001# curl -s -k -H "Authorization: Bearer ${TOKEN}" \
"https://api_gw_service.local/apis/sls/v1/search/hardware?extra_properties.Role=Application" | jq
[
  {
    "Parent": "x3000c0s19b0",
    "Xname": "x3000c0s19b0n0",
    "Type": "comptype_node",
    "Class": "River",
    "TypeString": "Node",
    "LastUpdated": 1606332877,
    "LastUpdatedTime": "2020-11-25 19:34:37.183293 +0000 +0000",
    "ExtraProperties": {
      "Role": "Application",
      "SubRole": "UAN"
    }
  }
]
```

3. Update the UAN object in SLS by adding an aliases array with the UAN's hostname.

   Replace *UAN_XNAME* in the URL and JSON object with the UAN's xname. Replace *UAN_PARENT_XNAME* in the JSON object with the UAN's parent xname. Replace *UAN_ALIAS* in the Aliases array with the UAN's hostname. The LastUpdated and LastUpdatedTime fields are not required to be in the PUT payload.

```
ncn-w001# curl -X PUT -s -k -H "Authorization: Bearer ${TOKEN}" \
"https://api_gw_service.local/apis/sls/v1/hardware/UAN_XNAME" -d '
{
  "Parent": "UAN_PARENT_XNAME",
  "Xname": "UAN_XNAME",
  "Type": "comptype_node",
  "Class": "River",
  "TypeString": "Node",
  "ExtraProperties": {
    "Role": "Application",
    "SubRole": "UAN",
    "Aliases": ["UAN_ALIAS"]
  }
}'
```

   For example:

```
ncn-w001# curl -X PUT -s -k -H "Authorization: Bearer ${TOKEN}" "https://api_gw_service.local/apis/
sls/v1/hardware/x3000c0s19b0n0" -d '
 {
   "Parent": "x3000c0s19b0",
   "Xname": "x3000c0s19b0n0",
   "Type": "comptype_node",
   "Class": "River",
   "TypeString": "Node",
   "ExtraProperties": {
     "Role": "Application",
     "SubRole": "UAN",
     "Aliases": ["uan01"]
   }
 }'
{"Parent":"x3000c0s19b0","Xname":"x3000c0s19b0n0","Type":"comptype_node","Class":"River","TypeString"
:"Node","LastUpdated":1606332877,"LastUpdatedTime":"2020-11-25 19:34:37.183293 +0000
+0000","ExtraProperties":{"Aliases":["uan01"],"Role":"Application","SubRole":"UAN"}}
```

After a few minutes the -mgmt name should begin resolving. Communication with the BMC should be available via the alias `uan01-mgmt`.

4. Confirm that the BMC for the UAN is up and running at the address for the alias.

```
ncn-w001# ping -c 4 uan01-mgmt
PING uan01-mgmt (10.254.2.53) 56(84) bytes of data.
64 bytes from x3000c0s19b0 (10.254.2.53): icmp_seq=1 ttl=255 time=0.170 ms
64 bytes from x3000c0s19b0 (10.254.2.53): icmp_seq=2 ttl=255 time=0.228 ms
64 bytes from x3000c0s19b0 (10.254.2.53): icmp_seq=3 ttl=255 time=0.311 ms
64 bytes from x3000c0s19b0 (10.254.2.53): icmp_seq=4 ttl=255 time=0.240 ms

--- uan01-mgmt ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3061ms
rtt min/avg/max/mdev = 0.170/0.237/0.311/0.051 ms
```

When this node boots, the DHCP request of it's -nmn interface will cause the uan01 to be created and resolved.

23.21.2 Configure an Application Node 5

Prerequisites

- The `hmn_connections.json` file for the system being used is available.

About this task

ROLE	System administrator
OBJECTIVE	Construct the `application_node_config.yaml` file that controls how the `csi config init` command finds and treats Applications nodes discovered by the `hmn_connections.json` file when building the System Layout Service (SLS) input file.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

The following `hmn_connections.json` file contains four Application nodes. When the `csi config init` command is used without an `application_node_config.yaml` file, only the application node `uan01` will be included the generated SLS input file. The other three application nodes will be ignored as they have unknown prefixes and will not be present in the SLS Input file.

```
[
  {"Source":"uan01",     "SourceRack":"x3000", "SourceLocation":"u23", "DestinationRack":"x3000",
"DestinationLocation":"u13", "DestinationPort":"j37"},
  {"Source":"gateway01", "SourceRack":"x3113", "SourceLocation":"u23", "DestinationRack":"x3113",
"DestinationLocation":"u13", "DestinationPort":"j37"},
  {"Source":"vn02",      "SourceRack":"x3114", "SourceLocation":"u23", "DestinationRack":"x3114",
"DestinationLocation":"u13", "DestinationPort":"j37"},
  {"Source":"login02",   "SourceRack":"x3115", "SourceLocation":"u23", "DestinationRack":"x3115",
"DestinationLocation":"u13", "DestinationPort":"j37"}
]
```

This file is manually created and follows this format:

```
---
# Additional application node prefixes to match on the Source field in the hmn_connections.json file
# See step 1 for additional information
prefixes:
  - gateway
  - vn

# Additional HSM SubRole mappings
# If a prefix does not have an HSM SubRole defined, the application node will not have a SubRole.
# See step 2 for additional information
prefix_hsm_subroles:
  gateway: Gateway
  vn: Visualization

# Application Node aliases
# One or more aliases can be specified for an application node
# If an application does not have entry in this map, then it will not have any aliases defined in SLS
# See step 3 for additional information
aliases:
  x3113c0s23b0n0: ["gateway-01"]
  x3114c0s23b0n0: ["visualization-02", "vn-02"]
```

The following Application node configuration does not add any additional prefixes, Hardware State Manager (HSM) sub-roles, or aliases:

```
# Additional application node prefixes
prefixes: []

# HSM Subroles
prefix_hsm_subroles: {}

# Application Node alias
aliases: {}
```

The following is an example entry from the `hmn_connections.json` file. The source name is the `Source` field, and the name of the device that is being connected to the Hardware Management Network (HMN). From this source name, the `csi config init` command can infer the type of hardware that is connected to the HMN, such as a node, power distribution unit (PDU), high-speed network (HSN) switch, and more.

```
{
    "Source": "uan01",
    "SourceRack": "x3000",
    "SourceLocation": "u19",
    "DestinationRack": "x3000",
    "DestinationLocation": "u14",
    "DestinationPort": "j37"
}
```

Procedure

1. Add additional Application node prefixes.

The `prefixes` field is an array of strings that augments the list of source name prefixes that are treated as Application nodes. By default, `csi config init` only looks for Application nodes that have source names starting with `uan`, `gn`, and `ln`. If the system contains Application nodes that fall outside of those source name prefixes, additional prefixes must be added to the `application_node_config.yaml` file.

These additional prefixes will used in addition to the default prefixes.

To add an additional prefix, append a new string element to the `prefixes` array:

```
---
prefixes: # Additional application node prefixes
  - gateway
  - vn
  - login # New prefix. Match source names that start with "login", such as login02
```

2. Add HSM sub-roles for Application node prefixes.

   The `prefix_hsm_subroles` field must map the Application node prefix (string) to the applicable Hardware State Manager (HSM) sub-role (string) for the Application nodes.

   By default the `csi config init` command will use the following sub-roles for Application nodes:

Prefix	HSM Sub-role
`uan`	UAN
`ln`	UAN
`gn`	Gateway

   To add an additional HSM sub-role for a given prefix, add a new mapping under the `prefix_hsm_subroles` field. The key is the Application node prefix and the value is the HSM sub-role.

```
---
prefix_hsm_subroles:
  gateway: Gateway
  vn: Visualization
  login: UAN # Application nodes that have the non-default prefix "login" are assigned the HSM
SubRole "UAN"
```

3. Add Application node aliases.

   The `aliases` field is an map of xnames (strings) to an array of aliases (strings). By default, the `csi config init` command sets the `ExtraProperties.Alias` field for Application nodes in the SLS input file.

   Instead of manually adding the Application node alias as described after the system is installed, the Application node aliases can be included when the SLS input file is built.

   To add additional Application node aliases, add a new mapping under the `aliases` field. The key is the xname of the Application node, and the value is an array of aliases (strings).

```
---
aliases: # Application Node alias
  x3113c0s23b0n0: ["gateway-01"]
  x3114c0s23b0n0: ["visualization-02", "vn-02"]
  x3115c0s23b0n0: ["uan-02"] # Added alias for the application node with the xname x3115c0s23b0n0
```

23.21.2 Configure BGP Neighbors on Management Switches 6

Prerequisites

This procedure requires administrative privileges.

About this task

ROLE	System administrator
OBJECTIVE	Manually configure and verify Border Gateway Protocol (BGP) neighbors on the management switches. This procedure applies to both Mellanox and Aruba switches.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Log into the spine switches.

   ```
   ncn-m001# ssh admin@SWITCH_IP
   ```

2. Verify the BGP neighbors on the spine switches.

   The BGP neighbors will be the NCN worker node IP addresses on the Node Management Network (NMN). If the system is using Aruba, one of the neighbors will be the other spine switch. The following command should be run for Aruba switches:

   ```
   sw-spine-001# show bgp ipv4 unicast summary
   VRF : default
   BGP Summary
   -----------
    Local AS               : 65533        BGP Router Identifier  : 10.252.0.1
    Peers                  : 4            Log Neighbor Changes   : No
    Cfg. Hold Time         : 180          Cfg. Keep Alive        : 60

    Neighbor        Remote-AS MsgRcvd MsgSent   Up/Down Time State        AdminStatus
    10.252.0.3      65533       31457   31474   00m:02w:04d  Established   Up
    10.252.2.8      65533       54730   62906   00m:02w:04d  Established   Up
    10.252.2.9      65533       54732   62927   00m:02w:04d  Established   Up
    10.252.2.18     65533       54732   62911   00m:02w:04d  Established   Up
   ```

   Run the following command for Mellanox switches:

   ```
   sw-spine-001 [standalone: master] # show ip bgp summary
   VRF name                  : default
   BGP router identifier     : 10.252.0.1
   local AS number           : 65533
   BGP table version         : 308
   Main routing table version: 308
   IPV4 Prefixes             : 261
   IPV6 Prefixes             : 0
   L2VPN EVPN Prefixes       : 0

   ------------------------------------------------------------------------------------------------------------
   -------------
   Neighbor          V    AS           MsgRcvd   MsgSent   TblVer    InQ    OutQ   Up/Down       State/
   PfxRcd
   ------------------------------------------------------------------------------------------------------------
   -------------
   10.252.0.7        4    65533        37421     42948     308       0      0      12:23:16:07
   ```

```
ESTABLISHED/53
10.252.0.8        4    65533        37421     42920     308       0      0      12:23:16:07
ESTABLISHED/51
10.252.0.9        4    65533        37420     42962     308       0      0      12:23:16:07
ESTABLISHED/51
```

3. Correct any IP address issues discovered in the previous step.

If the BGP neighbors are not in the `ESTABLISHED` state, make sure the IP addresses are correct for the route-map and BGP configuration. If the IPs are incorrect, update the configuration to match the IPs. The configuration below will need to be edited.

a. Retrieve the NCN IPs located in */etc/dnsmasq.d* to correct any incorrect IPs.

```
ncn-m001# grep w00 /etc/dnsmasq.d/statics.conf | grep nmn
host-record=ncn-w003,ncn-w003.nmn,10.252.1.13
host-record=ncn-w002,ncn-w002.nmn,10.252.1.14
host-record=ncn-w001,ncn-w001.nmn,10.252.1.15
```

b. Retrieve the Hardware Management Network (HMN) and Customer Access Network (CAN) IPs to correct the route-map configuration.

```
ncn-m001# grep ncn-w /etc/dnsmasq.d/statics.conf | \
egrep "NMN|HMN|CAN" | grep -v mgmt
dhcp-host=50:6b:4b:08:d0:4a,10.252.1.13,ncn-w003,20m # NMN
dhcp-host=50:6b:4b:08:d0:4a,10.254.1.20,ncn-w003,20m # HMN
dhcp-host=50:6b:4b:08:d0:4a,10.102.4.12,ncn-w003,20m # CAN
dhcp-host=98:03:9b:0f:39:4a,10.252.1.14,ncn-w002,20m # NMN
dhcp-host=98:03:9b:0f:39:4a,10.254.1.22,ncn-w002,20m # HMN
dhcp-host=98:03:9b:0f:39:4a,10.102.4.13,ncn-w002,20m # CAN
dhcp-host=98:03:9b:bb:a9:94,10.252.1.15,ncn-w001,20m # NMN
dhcp-host=98:03:9b:bb:a9:94,10.254.1.24,ncn-w001,20m # HMN
dhcp-host=98:03:9b:bb:a9:94,10.102.4.14,ncn-w001,20m # CAN
```

4. Delete the previous route-map and BGP configuration on Mellanox and Aruba switches.

The Aruba configuration requires the other peering switch to be set as a BGP neighbor. The Mellanox configuration does not require this. Delete the previous route-map and BGP configuration on both switches.

a. Delete the previous Aruba configuration.

```
spine01# conf t

spine01(config)# no router  bgp 65533
This will delete all BGP configurations on this device.
Continue (y/n)? y

spine01(config)# no route-map ncn-w003
spine01(config)# no route-map ncn-w002
spine01(config)# no route-map ncn-w001
```

The following is an example of the Aruba configuration:

```
route-map rm-ncn-w001 permit seq 10
     match ip address prefix-list pl-nmn
     set ip next-hop 10.252.2.8
route-map rm-ncn-w001 permit seq 20
     match ip address prefix-list pl-hmn
     set ip next-hop 10.254.2.27
route-map rm-ncn-w001 permit seq 30
     match ip address prefix-list pl-can
     set ip next-hop 10.103.10.10
route-map rm-ncn-w002 permit seq 10
     match ip address prefix-list pl-nmn
     set ip next-hop 10.252.2.9
route-map rm-ncn-w002 permit seq 20
     match ip address prefix-list pl-hmn
     set ip next-hop 10.254.2.25
route-map rm-ncn-w002 permit seq 30
     match ip address prefix-list pl-can
     set ip next-hop 10.103.10.9
route-map rm-ncn-w003 permit seq 10
     match ip address prefix-list pl-nmn
```

```
     set ip next-hop 10.252.2.18
route-map rm-ncn-w003 permit seq 20
     match ip address prefix-list pl-hmn
     set ip next-hop 10.254.2.26
route-map rm-ncn-w003 permit seq 30
     match ip address prefix-list pl-can
     set ip next-hop 10.103.10.11
!
router bgp 65533
    bgp router-id 10.252.0.1
    maximum-paths 8
    neighbor 10.252.0.3 remote-as 65533
    neighbor 10.252.2.8 remote-as 65533
    neighbor 10.252.2.9 remote-as 65533
    neighbor 10.252.2.18 remote-as 65533
    address-family ipv4 unicast
        neighbor 10.252.0.3 activate
        neighbor 10.252.2.8 activate
        neighbor 10.252.2.8 route-map ncn-w001 in
        neighbor 10.252.2.9 activate
        neighbor 10.252.2.9 route-map ncn-w002 in
        neighbor 10.252.2.18 activate
        neighbor 10.252.2.18 route-map ncn-w003 in
    exit-address-family
```

b. Delete the previous Mellanox configuration.

```
sw-spine-001 [standalone: master] #
sw-spine-001 [standalone: master] # conf t
sw-spine-001 [standalone: master] (config) # no router bgp 65533
sw-spine-001 [standalone: master] (config) # no route-map ncn-w001
sw-spine-001 [standalone: master] (config) # no route-map ncn-w002
sw-spine-001 [standalone: master] (config) # no route-map ncn-w003
```

The following is an example of the Mellanox configuration:

```
## Route-maps configuration
##
   route-map rm-ncn-w001 permit 10 match ip address pl-nmn
   route-map rm-ncn-w001 permit 10 set ip next-hop 10.252.0.7
   route-map rm-ncn-w001 permit 20 match ip address pl-hmn
   route-map rm-ncn-w001 permit 20 set ip next-hop 10.254.0.7
   route-map rm-ncn-w001 permit 30 match ip address pl-can
   route-map rm-ncn-w001 permit 30 set ip next-hop 10.103.8.7
   route-map rm-ncn-w002 permit 10 match ip address pl-nmn
   route-map rm-ncn-w002 permit 10 set ip next-hop 10.252.0.8
   route-map rm-ncn-w002 permit 20 match ip address pl-hmn
   route-map rm-ncn-w002 permit 20 set ip next-hop 10.254.0.8
   route-map rm-ncn-w002 permit 30 match ip address pl-can
   route-map rm-ncn-w002 permit 30 set ip next-hop 10.103.8.8
   route-map rm-ncn-w003 permit 10 match ip address pl-nmn
   route-map rm-ncn-w003 permit 10 set ip next-hop 10.252.0.9
   route-map rm-ncn-w003 permit 20 match ip address pl-hmn
   route-map rm-ncn-w003 permit 20 set ip next-hop 10.254.0.9
   route-map rm-ncn-w003 permit 30 match ip address pl-can
   route-map rm-ncn-w003 permit 30 set ip next-hop 10.103.8.9

##
## BGP configuration
##
   protocol bgp
   router bgp 65533 vrf default
   router bgp 65533 vrf default router-id 10.252.0.1 force
   router bgp 65533 vrf default maximum-paths ibgp 32
   router bgp 65533 vrf default neighbor 10.252.0.7 remote-as 65533
   router bgp 65533 vrf default neighbor 10.252.0.7 route-map ncn-w001
   router bgp 65533 vrf default neighbor 10.252.0.8 remote-as 65533
   router bgp 65533 vrf default neighbor 10.252.0.8 route-map ncn-w002
   router bgp 65533 vrf default neighbor 10.252.0.9 remote-as 65533
   router bgp 65533 vrf default neighbor 10.252.0.9 route-map ncn-w003
   router bgp 65533 vrf default neighbor 10.252.0.10 remote-as 65533
```

5. Restart the BGP process on the switches.

Once the IPs are updated for the route-maps and BGP neighbors, restart the BGP process on the switches. This may need to be done multiple times for all of the peers to come up.

a. Restart the BGP process for Mellanox switches.

```
# clear ip bgp all
```

b. Restart the BGP process for Aruba switches.

```
# clear bgp *
```

Troubleshooting: If the BGP peers are still not coming up you, check the `Metallb.yaml` configuration file for errors. The `Metallb.yaml` file should point to the NMN IPs of the configure switches.

```
ncn-m001# vi Metallb.yaml
---
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    peers:
    - peer-address: 10.252.0.2
      peer-asn: 65533
      my-asn: 65533
    - peer-address: 10.252.0.3
      peer-asn: 65533
      my-asn: 65533
    address-pools:
    - name: customer-access
      protocol: bgp
      addresses:
      - 10.102.9.112/28
    - name: customer-access-dynamic
      protocol: bgp
      addresses:
      - 10.102.9.128/25
    - name: hardware-management
      protocol: bgp
      addresses:
      - 10.94.100.0/24
    - name: node-management
      protocol: bgp
      addresses:
      - 10.92.100.0/24
```

If the management network has aggregation switches setup and peering is set up with them, the `Metallb.yaml` configuration file will need to be updated to reflect these IPs. The peer-address should be the IP of the switch that is doing BGP peering.

23.21.2 Passwordless SSH 7

Passwordless SSH keypairs for the Cray System Management (CSM) are created automatically and periodically maintained with a deployment, and then staged into Kubernetes secrets and configmaps unconditionally. Administrators can use these provided keys, provide their own keys, or use their own solution for authentication.

Master, worker, or storage non-compute nodes (NCNs) must have these keys applied to them through the Configuration Framework Service (CFS) in order for passwordless SSH to work. These can be applied to NCNs one time only through a single CFS session, or maintained more persistently for each environment by registering the configuration with CFS for each desired environment. This must be done electively in order for this to work.

Passwordless SSH setup contains two Ansible roles:

trust-csm-ssh-keys	Configures an environment to trust CSM keys. The public half of the key is added to the `authorized_keys` file.
passwordless-ssh	Pulls both of the public and private key portions into the local environment.

These roles are shared between products through the Version Control Service (VCS) and Gitea.

Downstream managed product environments, such as compute nodes and User Access Nodes (UANs), contain configuration references to the *trust-csm-ssh-keys* roles by default. During image customization, public portions of these keys are added to the environments exactly once. If an admin changes the automatically generated private or public key for these environments, the images must be reconfigured, or the product `site.yaml` needs to be reconfigured to allow for pushing subsequent updated public keys. Existing public keys injected into the authorized keys file are not automatically removed during reconfiguration.

Locations

Passwordless SSH Keys are generated using vault under the *csm* key. The private half of the key is published as a kubernetes secret:

```
ncn-m001# kubectl get secrets -n services csm-private-key \
-o jsonpath="{.data.value}" | base64 -d
 -----BEGIN EC PRIVATE KEY-----
 MIGkAgEBBDCax9yqFs4TlTR0pnI5rvk7FlKl4weWnQxfAGRtTGM5axygblJxLbdY
 RnRhym8t67qgBwYFK4EEACKhZANiAAQeVXJpIMVq2471w0q6zq62BXMy4nIrs+fd
 cTEeGPjEDpudChKCrdaMSriAe7W/xvb9tlVOFp+QWJn91CndpVcq632d9qyRoy/Z
 IpPkdiTkOQltdsSum2jUkWLKybou8Z4=
 -----END EC PRIVATE KEY-----
```

The public half of the key is stored in a Kubernetes configmap:

```
ncn-m001# kubectl get configmap -n services csm-public-key \
-o jsonpath="{.data.value}" | base64 -d
 ecdsa-sha2-
nistp384AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAAIbmlzdHAzODQAAABhBB5VcmkgxWrbjvXDSrrOrrYFczLiciuz5
91xMR4Y+MQOm50KEoKt1oxKuIB7tb/G9v22VU4Wn5BYmf3UKd2lVyrrfZ32rJGjL9kik
+R2JOQ5CW12xK6baNSRYsrJui7xng==
```

These commands can be run anywhere `kubectl` is configured to point at the cluster with proper authentication.

Scope

The goal of passwordless SSH is to enable an easy way for interactive passwordless SSH from and between CSM product environments to downstream managed product environments, without requiring each downstream product environment to create and apply individual changes to NCNs.

Passwordless SSH from managed nodes into management nodes is not intended or supported.

Local site security requirements may preclude use of passwordless SSH access between products or environments, so the private key is not applied anywhere by default. Users are advised to use the CSM product configuration repository (as uploaded during install to Gitea) to apply these changes to NCNs. See *Passwordless SSH Use Cases* on page 265 for more information.

Applying changes to NCN environments allows basic passwordless SSH to function for the lifetime of the affected file system. Most NCN environments are diskfully provisioned, so it is often only necessary to elect to do this once.

The public key is injected as a trusted source as an *authorized_key* for managed environments automatically (UAN, computes), even if the private half is never used. If this is not desirable, the role *trust-csm-public-keys* can be removed from the product `site.yaml` top level play.

Under no circumstances should the *passwordless-ssh* Ansible role be run against an image that is to be registered into IMS/S3 during image customization. Doing so would allow access to private credentials information to all nodes that trust the key by retrieving the image and extracting the saved private key.

CFS itself does not use the keys provided to initialize connections between nodes. Instead, CFS uses a time-limited vault signed public certificate along with its own key pair.

23.21.27 Passwordless SSH Use Cases .1

Administrators can manage the implementation of passwordless SSH keypairs for Cray System Management (CSM) by using the provided keys, providing their own keys, or using their own solution for authentication. This section outlines common use cases for working with passwordless SSH and keys on a HPE Cray EX system.

The following use cases are covered in this section:

- Provide Custom Keys
- Restore Keys to Initial Defaults
- Create a CFS Configuration for the Application of Passwordless SSH to NCNs
- Configure NCN Environments to Use Passwordless SSH
- Disable Passwordless SSH with CSM Configuration

Provide Custom Keys

Admins may elect to replace the provided keys with their own custom keys. This is best done before the impacted environments are configured or installed, because it can be difficult to know where all of the key portions are populated.

The csm-ssh-keys deployment will only push its configured keys when the upstream image is configured using the Configuration Framework Service (CFS) and Image Management Service (IMS) customization. The `trust-csm-ssh-keys` Ansible role does not remove existing entries that have been applied. This is true for both image customization and node personalization for CFS targets.

To replace the private key half:

```
ncn-m001# kubectl get secret -n services csm-private-key -o json | \
jq --arg value "$(cat ~/.ssh/id_rsa |base64)" \
'.data["value"]=$value' | | kubectl apply -f -
```

In this example, `~/.ssh/id_rsa` is a local file containing a private key in a format specified by the admin.

To replace the public key half:

```
ncn-m001# kubectl delete configmap -n services \
csm-public-key && cat ~/.ssh/id_rsa.pub | \
base64 > ./value && kubectl create configmap --from-file \
value csm-public-key --namespace services && rm ./value
```

In this example, `~/.ssh/id_rsa.pub` is a file containing the public key half that the admin intends to use for Cray System Management (CSM) and downstream products.

Restore Keys to Initial Defaults

The csm-ssh-keys deployment periodically checks the configmap and secret containing the key information. If these entries do not exist, it will overwrite them from the key generated with vault.

In this case, deleting the associated configmap and secrets will republish them.

Create a CFS Configuration for the Application of Passwordless SSH to NCNs

It may be necessary to create or update a CFS configuration entry to apply changes to the NCN environment. Typically, this needs to be done for newly installed HPE Cray EX product releases.

The following example creates a new CFS configuration with a single product configuration layer (*csm-1.40*). Multiple product configuration layers may be created to apply changes to a node at the same time. See the *Configuration Management* section for more information.

```
ncn-m001# RELEASE=1.4.0
ncn-m001# COMMIT='git ls-remote \
https://api-gw-service-nmn.local/vcs/cray/csm-configmanagement.git \
refs/heads/cray/csm/1.4.0 | awk '{print $1}''
ncn-m001# echo $COMMIT
ncn-m001# cat <<'EOF' > csm-config-$RELEASE.json
{
  "layers": [
    {
      "name": "csm-@RELEASE@",
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/csm-configmanagement.git",
      "playbook": "site.yaml",
       "commit": "@COMMIT@"
    }
  ]
}
EOF
ncn-m001# sed -i -e "s:@RELEASE@:$RELEASE:g" \
-e "s:@COMMIT@:$COMMIT@:g" csm-config-$RELEASE.json
ncn-m001# cray cfs configurations update csm-$RELEASE \
--file csm-config-$RELEASE.json
```

Configure NCN Environments to Use Passwordless SSH

A one time configuration action may be applied to an NCN environment to allow passwordless SSH into a node. Depending on the environment, root filesystems that are persistent may only need to be configured once, depending on the playbooks and associated roles registered in VCS.

In the following example, node x3000c0s7b0n0 is configured with the CSM playbook `site.yaml` file, which obtains the stored keys and applies them to the environment.

```
ncn-m001# cray cfs sessions create --ansible-limit x3000c0s7b0n0 \
--configuration-name csm-1.4.0 --name passwordless-ssh
```

Multiple NCNs may be configured in a managed and persistent way by using standard configuration management tooling. This can be done with or without the use of boot orchestration for these nodes, and with or without multiple layered configurations. However, passwordless SSH key information is not stored within the Version Control Service (VCS) itself, which means updates to the values stored within the Kubernetes secrets and configmaps for these keys are not automatically applied to the system on update.

Disable Passwordless SSH with CSM Configuration

If the CSM provided solution for passwordless SSH is undesirable, but the CSM product configuration contains other valid setup actions, the easiest course of actions is to modify the provided top level play to no longer run the associated passwordless SSH setup roles as part of CSM:

```
diff --git a/site.yaml b/site.yaml
    index 72a2ca7..a69e871 100644
    --- a/site.yaml
    +++ b/site.yaml
    @@ -9,5 +9,5 @@
         # Allow trust of CSM generated keys for elective passwordless SSH;
         # Enables both passwordless ssh from, and to, all nodes.
```

```
        - role: trust-csm-ssh-keys
-       - role: passwordless-ssh
+       # - role: passwordless-ssh
```

These changes can be applied to the CSM product in a custom branch using the documented procedures in the *Configuration Management* section using Gitea and VCS. Disabling this role will not revoke any authorized or trusted keys on affected nodes; it will prevent the application of any CSM specific keys.

23.22 Install the Management Network

The following is a high-level workflow of how to install the management network:

1. Prepare for installation:
   a. Access `ncn-m001`. The iLO should already be completed.

      Refer to "Boot LiveCD" in the *HPE Cray EX System Installation and Configuration Guide S-8000*.

   b. Install the baseline switch configuration.

      See *Management Network Base Configuration* on page 268.

   c. Update the firmware.

      See *Update the Management Network Firmware* on page 270.
2. Configure all switches via IPv6 connection from `ncn-m001`:
   a. Layer2 configuration:
      i. VLAN configuration.

         See *Configure the Management Network VLAN* on page 273.

      ii. Configure MLAG and VSX pairs.

         See *Configure the Management Network MLAG* on page 275.

      iii. Baseboard Management Controllers (BMCs), Chassis Memory Module (CMM), and Gateway Node port configuration.

         See *Configure the Management Network Access Ports* on page 278.

      iv. Set up switch uplink ports and inter-switch links (ISLs).

         See *Configure the Management Network Uplink* on page 279.

   b. Layer3 configuration:
      i. L3 interfaces for the network access control list (ACL).

         See *Configure the Management Network ACLs* on page 281.

      ii. Dynamic routing for OSPFv2 and BGP.

         See *Configure Layer3 Routing for the Management Network* on page 282.

      iii. SNMP configuration.

         See *Configure SNMP for the Management Network* on page 283.

   c. Create the Customer Access Network (CAN).

Management Network Changes in Release 1.4

The following network architecture changes have been made for the HPE Cray EX 1.4 release:

- New Aruba/HPE switches.
- ACL configuration is now supported.
- The site connection has moved from `ncn-w001` to `ncn-m001`.
- The IP-Helper now resides on the switches where the default gateway for the servers, such as BMCs and computes, is configured.
- The IP-Helper 10.92.100.222 is applied on vlan 1, 2, and 7.
- The IP-Helper 10.94.100.222 is applied on vlan 4.
- Added flow-control and MAGP changes.
- Removed BPDUfilter on the Dell access ports.
- Added BPDUguard to the Dell access ports.

For more information on changes for Dell and Mellanox switches, refer to *Upgrades for Dell and Mellanox Switches* on page 284.

23.22.1 Management Network Base Configuration

Prerequisites

- The administrator has console access to all of the switches.
- The SHCD is available.

About this task

ROLE	System administrator, system installer, network administrator
OBJECTIVE	Setup the base network configuration of the HPE Cray EX management network. The purpose of this configuration is to have an IPv6 underlay that allows admins to always be able to access the management switches. The base configuration is running OSPFv3 over IPv6 on VLAN 1, so the switches can dynamically form neighbours, which enables remote access to them.
ADMIN IMPACT	After the base configuration is applied, an admin will be able to access all management switches and apply the remaining configuration.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Apply the admin username and password.

```
switch# conf t
switch(config)# user admin group administrators password plaintext xxxxxxxx
```

2. Set the hostname of the switch.

   The name defined in the SHCD can be used for this step.

```
switch(config)# hostname sw-25g04
```

3. Enable each uplink or switch-to-switch link and configure them.

```
sw-25g04(config)# int 1/1/1
sw-25g04(config-if)# no routing
sw-25g04(config-if)# vlan trunk native 1
sw-25g04(config-if)# no shut
```

4. Add an IPv6 interface to VLAN 1 and start the OSPv3 process.

   A unique router-id is needed (IPv4 address) that will only be used for identifying the router; this is not a route-able address. This can be incremented by one for each switch. Use other IPs for router-IDs if desired.

```
sw-25g04(config)# router ospfv3 1
sw-25g04(config-ospfv3-1)# area 0
sw-25g04(config-ospfv3-1)# router-id 172.16.0.1
sw-25g04(config-ospfv3-1)# exit
```

5. Add VLAN 1 interface to OSPF area 0.

```
sw-25g04(config)# int vlan 1
sw-25g04(config-if-vlan)# ipv6 address autoconfig
sw-25g04(config-if-vlan)# ipv6 ospfv3 1 area 0
sw-25g04(config-if-vlan)# exit
```

6. Add a unique IPv6 Loopback address.

   This will be the address that is used to remotely connect. This can be incremented by one for every switch in use.

```
sw-25g04(config)# interface loopback 0
sw-25g04(config-loopback-if)# ipv6 address fd01::0/64
sw-25g04(config-loopback-if)# ipv6 ospfv3 1 area 0
```

7. Enable remote access through SSH, HTTPS, and API.

```
sw-25g04(config)# ssh server vrf default
sw-25g04(config)# ssh server vrf mgmt
sw-25g04(config)# https-server vrf default
sw-25g04(config)# https-server vrf mgmt
sw-25g04(config)# https-server rest access-mode read-write
```

8. Run a `show run` to view the configuration.

```
sw-25g04(config)# show run
Current configuration:
!
!Version ArubaOS-CX Virtual.10.05.0020
!export-password: default
hostname sw-25g04
user admin group administrators password ciphertext AQBapXDwaGq+GHwyLgj0Eu
led locator on
```

```
!
!
!
!
ssh server vrf default
ssh server vrf mgmt
vlan 1
interface mgmt
    no shutdown
    ip dhcp
interface 1/1/1
    no shutdown
    no routing
    vlan trunk native 1
    vlan trunk allowed all
interface loopback 0
    ipv6 address fd01::1/64
    ipv6 ospfv3 1 area 0.0.0.0
interface loopback 1
interface vlan 1
    ipv6 address autoconfig
    ipv6 ospfv3 1 area 0.0.0.0
!
!
!
!
!
router ospfv3 1
    router-id 192.168.100.1
    area 0.0.0.0
https-server vrf default
https-server vrf mgmt
```

9. View the OSPFv3 neighbors.

```
sw-25g04# show ipv6 ospfv3 neighbors
OSPFv3 Process ID 1 VRF default
================================

Total Number of Neighbors: 1

Neighbor ID      Priority  State             Interface
-----------------------------------------------------
192.168.100.2    1         FULL/BDR          vlan1
  Neighbor address fe80::800:901:8b4:e152
```

10. Connect to the neighbors via the IPv6 loopback that was set earlier in the procedure.

```
sw-25g03# ssh admin@fd01::1
```

23.22.2 Update the Management Network Firmware

Prerequisites

Access to the switches from the LiveCD or `ncn-m001`.

About this task

ROLE	System administrator, system installer
OBJECTIVE	Update the firmware for Aruba, Dell, and Mellanox management network switches.
LIMITATIONS	None.

Switch CLI prompts have been truncated in the examples.

Procedure

1. Confirm that the firmware is in `/var/www/fw/network` on the LiveCD.

   ```
   pit# ls -lh
   total 2.7G
   -rw-r--r-- 1 root root 614M Jan 15 18:57 ArubaOS-CX_6400-6300_10_05_0040.stable.swi
   -rw-r--r-- 1 root root 368M Jan 15 19:09 ArubaOS-CX_8320_10_05_0040.stable.swi
   -rw-r--r-- 1 root root 406M Jan 15 18:59 ArubaOS-CX_8325_10_05_0040.stable.swi
   -rw-r--r-- 1 root root 729M Aug 26 17:11 onyx-X86_64-3.9.1014.stable.img
   -rw-r--r-- 1 root root 577M Oct 28 11:45 OS10_Enterprise_10.5.1.4.stable.tar
   ```

   The table shows the required firmware versions for each type of switch for release 1.4.

   Table 11. Switch Firmware Versions

Vendor	Model	Version
Aruba	6300	ArubaOS-CX_6400-6300_10.06.0010
Aruba	8320	ArubaOS-CX_8320_10.06.0010
Aruba	8325	ArubaOS-CX_8325_10.06.0010
Dell	S3048-ON	10.5.1.4
Dell	S4148F-ON	10.5.1.4
Dell	S4148T-ON	10.5.1.4
Mellanox	MSN2100	3.9.1014
Mellanox	MSN2700	3.9.1014

   UPDATE ARUBA SWITCH FIRMWARE

2. SSH into the Aruba switch.

   ```
   ncn-m001# ssh sw-leaf-001
   sw-leaf-001#
   ```

3. Copy the firmware image to the switch primary flash destination.

   The IP address 10.252.1.12 is the liveCD.

   ```
   sw-leaf-001# copy \
   sftp://root@10.252.1.12//var/www/ephemeral/data/network_images/ArubaOS-
   CX_6400-6300_10_06_0010.stable.swi primary
   ```

4. Write the firmware.

   ```
   sw-leaf-001# write mem
   Copying configuration: [Success]
   ```

5. Wait for the upload to complete, then check the firmware image.

   ```
   sw-leaf-001# show image
   --------------------------------------------------------------------------------
   ArubaOS-CX Primary Image
   --------------------------------------------------------------------------------
   ```

```
Version : FL.10.06.0010
Size    : 643 MB
Date    : 2020-12-14 10:06:34 PST
SHA-256 : 78dc27c5e521e92560a182ca44dc04b60d222b9609129c93c1e329940e1e11f9
```

6. Boot the switch to the correct image.

```
sw-leaf-001# boot system primary
```

7. After the switch reboots, check the firmware image again.

```
sw-leaf-001# show version
------------------------------------------------------------------------------
ArubaOS-CX
(c) Copyright 2017-2020 Hewlett Packard Enterprise Development LP
------------------------------------------------------------------------------
Version      : FL.10.06.0010
Build Date   : 2020-09-29 07:44:16 PDT
Build ID     : ArubaOS-CX:FL.10.06.0010:3cbfcce60961:202009291304
Build SHA    : 3cbfcce609617b0cf84a6b941a2b36c43dfeb2cb
Active Image : primary

Service OS Version : FL.01.07.0002
BIOS Version       : FL.01.0002
```

UPDATE MELLANOX SWITCH FIRMWARE

8. SSH to the Mellanox switch and enable configuration changes.

```
ncn-m001# ssh sw-spine-001
sw-spine-001> enable
sw-spine-001# configure terminal
sw-spine-001(config)#
```

9. Retrieve the image from ncn-m001.

```
sw-spine-001# image fetch http://10.252.1.4/fw/network/onyx-X86_64-3.9.1014.stable.img
```

10. Install the image.

```
sw-spine-001# image install onyx-X86_64-3.9.1014.stable.img
```

11. Select the image as the `next` boot partition.

```
sw-spine-001# image boot next
```

12. Write memory and reload the image.

```
sw-spine-001# write memory
sw-spine-001# reload
```

13. After the switch has reloaded, verify that the image is installed.

```
sw-spine-001# show images
Installed images:
  Partition 1:
    version: X86_64 3.9.0300 2020-02-26 19:25:24 x86_64

  Partition 2:
    version: X86_64 3.9.1014 2020-08-05 18:06:58 x86_64

Last boot partition: 2
Next boot partition: 1

Images available to be installed:
  1:
```

```
    Image  : onyx-X86_64-3.9.1014.stable.img
    Version: X86_64 3.9.1014 2020-08-05 18:06:58 x86_64
```

UPDATE DELL SWITCH FIRMWARE

14. SSH to the Dell switch and enable configuration changes on the switch.

```
ncn-m001# ssh sw-leaf-001.mtl
sw-leaf-001> configure terminal
sw-leaf-001(config)#
```

15. Retrieve the image from ncn-m001.

```
sw-leaf-001# image install http://10.252.1.4/fw/network/OS10_Enterprise_10.5.1.4.stable.tar
```

16. Check the image upload status.

```
sw-leaf-001# show image status
Image Upgrade State:     download
================================================
File Transfer State:     download
------------------------------------------------
  State Detail:          In progress
  Task Start:            2021-02-08T21:24:14Z
  Task End:              0000-00-00T00:00:00Z
  Transfer Progress:     7 %
  Transfer Bytes:        40949640 bytes
  File Size:             604119040 bytes
  Transfer Rate:         869 kbps
```

17. After the image is uploads, write it to the switch reboot.

```
sw-leaf-001# write memory
sw-leaf-001.mtl# reload
```

18. Verify the firmware version.

```
sw-leaf-001# show version
Dell EMC Networking OS10 Enterprise
Copyright (c) 1999-2020 by Dell Inc. All Rights Reserved.
OS Version: 10.5.1.4
Build Version: 10.5.1.4.249
```

23.22.3 Configure the Management Network VLAN

Prerequisites

- The administrator has access to all of the switches.
- The SHCD is available.

About this task

ROLE	System administrator, system installer, network administrator
OBJECTIVE	Configure the VLANs that are required for liquid-cooled and air-cooled cabinets in the HPE Cray EX system.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Add all of the VLANS required by the HPE Cray EX system.

   Cray Site Init (CSI) generates the IP addresses used by the system. The following are only examples. Some switches will not need the Customer Access Network (CAN) VLAN7; most of the time this IP is only located on the Spine for external connectivity. The air-cooled cabinets will need the following VLANs:

   Table 12. Air-Cooled Cabinet VLANs

VLAN	Switch1 IP	Switch2 IP	Active Gateway	Purpose
2	10.252.0.2/17	10.252.0.3/17	10.252.0.1	Air-cooled node management
4	10.254.0.2/17	10.254.0.3/17	10.254.0.1	Air-cooled hardware management
7	TBD	TBD	TBD	Customer Access Network (CAN)
10	10.11.0.2/17	10.11.0.3/17	10.11.0.1	Storage (future release)

   The liquid-cooled cabinets will need the following VLANs, which are typically the CDU switches. The 2xxx and 3xxx VLANs are per cabinet, so with each additional cabinet, increment the VLAN by 1 and add a new /22 subnet.

   Table 13. Liquid-Cooled Cabinet VLANs

VLAN	Switch1 IP	Switch2 IP	Purpose
2	10.252.0.x/17	10.252.0.x/17	Air-cooled node management
4	10.254.0.x/17	10.254.0.x/17	Air-cooled hardware management
2000	10.100.0.2/22	10.100.0.3/22	Liquid-cooled node management
3000	10.104.0.2/22	10.104.0.3/22	Liquid-cooled hardware management

2. Add the networks to each of the VSX pairs.

```
sw-24g03(config)# vlan 2
sw-24g03(config-vlan-2)# name NMN
sw-24g03(config-vlan-2)# vlan 4
sw-24g03(config-vlan-4)# name HMN
sw-24g03(config-vlan-4)# vlan 7
sw-24g03(config-vlan-7)# name CAN
sw-24g03(config-vlan-7)# vlan 10
sw-24g03(config-vlan-10)# name SUN

sw-24g04(config)# vlan 2
sw-24g04(config-vlan-2)# name NMN
sw-24g04(config-vlan-2)# vlan 4
```

```
sw-24g04(config-vlan-4)# name HMN
sw-24g04(config-vlan-4)# vlan 7
sw-24g04(config-vlan-7)# name CAN
sw-24g04(config-vlan-7)# vlan 10
sw-24g04(config-vlan-10)# name SUN
```

3. Configure the VLAN interfaces on both switches participating in VSX.

   The IP-Helper is used to forward DCHP traffic from one network to a specified IP address.

```
sw-24g03(config)# int vlan 1
sw-24g03(config-if-vlan)# ip helper-address 10.92.100.222
sw-24g03(config-if-vlan)# int vlan 2
sw-24g03(config-if-vlan)# ip helper-address 10.92.100.222
sw-24g03(config-if-vlan)# int vlan 4
sw-24g03(config-if-vlan)# ip helper-address 10.94.100.222
sw-24g03(config-if-vlan)# int vlan 7
sw-24g03(config-if-vlan)# ip helper-address 10.92.100.222

sw-24g04(config)# int vlan 1
sw-24g04(config-if-vlan)# ip helper-address 10.92.100.222
sw-24g04(config-if-vlan)# int vlan 2
sw-24g04(config-if-vlan)# ip helper-address 10.92.100.222
sw-24g04(config-if-vlan)# int vlan 4
sw-24g04(config-if-vlan)# ip helper-address 10.94.100.222
sw-24g04(config-if-vlan)# int vlan 7
sw-24g04(config-if-vlan)# ip helper-address 10.92.100.222
```

   For CDU switches, the IP helpers will look like the following. Any 2xxx VLANs will have 10.92.100.222 as the IP helper address, and any 3xxx VLANs will have 10.94.100.222 as the IP helper address.

```
interface vlan 2000
    ip helper-address 10.92.100.222
interface vlan 3000
    ip helper-address 10.94.100.222
```

4. Add the networks to the switches connected to the BMCs.

```
sw-smn-001(config)# vlan 2
sw-smn-001(config-vlan-2)# name NMN
sw-smn-001(config-vlan-2)# vlan 4
sw-smn-001(config-vlan-4)# name HMN
sw-smn-001(config-vlan-4)# vlan 7
sw-smn-001(config-vlan-7)# name CAN
sw-smn-001(config-vlan-7)# vlan 10
sw-smn-001(config-vlan-10)# name SUN
```

5. Add VLAN interfaces to the VLANs that were just created.

   Specific addresses are provided by CSI.

```
sw-smn-001(config)# int vlan 1
sw-smn-001(config-if-vlan)# ip address 10.1.0.4/16
sw-smn-001(config-if-vlan)# int vlan 2
sw-smn-001(config-if-vlan)# ip address 10.252.0.4/17
sw-smn-001(config-if-vlan)# int vlan 4
sw-smn-001(config-if-vlan)# ip address 10.254.0.4/17
```

23.22.4 Configure the Management Network MLAG

Prerequisites

- Console Access to the switches participating in the multi-chassis link aggregation group (MLAG) configuration.
- Two switches running the same firmware version.
- Three cables connected between the switches: two for the Inter-Switch Link (ISL), and one for the Keepalive.

About this task

ROLE	System administrator, system installer, network administrator
OBJECTIVE	Set up a bonded configuration from the non-compute nodes (NCNs) to the management network.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Create the Keepalive vrf on both switches.

   The following configuration will need to be done on both switches participating in VSX/MLAG. If there is a unique configuration, it will be called out.

   ```
   sw-24g04(config)# vrf keepalive
   ```

2. Create the ISL lag on both switches.

   ```
   sw-24g04(config)# interface lag 99
   sw-24g04(config-lag-if)# no shutdown
   sw-24g04(config-lag-if)# description ISL link
   sw-24g04(config-lag-if)# no routing
   sw-24g04(config-lag-if)# vlan trunk native 1 tag
   sw-24g04(config-lag-if)# vlan trunk allowed all
   sw-24g04(config-lag-if)# lacp mode active
   ```

3. Set up the Keepalive link.

   This will require a unique IP address on both switches.

   ```
   sw-24g04(config)# int 1/1/3
   sw-24g04(config-if)# no shutdown
   sw-24g04(config-if)# mtu 9198
   sw-24g04(config-if)# vrf attach keepalive
   sw-24g04(config-if)# description keepalive
   sw-24g04(config-if)# ip address 192.168.255.0/31

   sw-24g03(config)# int 1/1/3
   sw-24g03(config-if)# no shutdown
   sw-24g03(config-if)# mtu 9198
   sw-24g03(config-if)# vrf attach keepalive
   sw-24g03(config-if)# description keepalive
   sw-24g03(config-if)# ip address 192.168.255.1/31
   ```

4. Add the ISL ports to the Link Aggregate Group (LAG).

   These are two of the ports connected between the switches.

```
sw-24g04(config)# int 1/1/1-1/1/2
sw-24g04(config-if-<1/1/1-1/1/2>)# no shutdown
sw-24g04(config-if-<1/1/1-1/1/2>)# mtu 9198
sw-24g04(config-if-<1/1/1-1/1/2>)# lag 99
```

5. Create the VSX instance and set up the keepalive link.

```
sw-24g03(config)# vsx
sw-24g03(config-vsx)# system-mac 02:01:00:00:01:00
sw-24g03(config-vsx)# inter-switch-link lag 99
sw-24g03(config-vsx)# role primary
sw-24g03(config-vsx)# keepalive peer 192.168.255.0 source 192.168.255.1 vrf keepalive
sw-24g03(config-vsx)# linkup-delay-timer 600
sw-24g03(config-vsx)# vsx-sync vsx-global

sw-24g04(config)# vsx
sw-24g04(config-vsx)# system-mac 02:01:00:00:01:00
sw-24g04(config-vsx)# inter-switch-link lag 99
sw-24g04(config-vsx)# role secondary
sw-24g04(config-vsx)# keepalive peer 192.168.255.1 source 192.168.255.0 vrf keepalive
sw-24g04(config-vsx)# linkup-delay-timer 600
sw-24g04(config-vsx)# vsx-sync vsx-global
```

6. Confirm there is now an established VSX session.

```
sw-24g04(config-if-vlan)# show vsx brief
ISL State                             : In-Sync
Device State                          : Sync-Primary
Keepalive State                       : Keepalive-Established
Device Role                           : secondary
Number of Multi-chassis LAG interfaces : 0
```

7. Create and set up high-availability VLAN interfaces.

This should be done for all VLANs. The VLANs in the 2xxx and 3xxx range will be for coolant distribution unit (CDU) switches only. Cray Site Init (CSI) generates the IP addresses used by the system. The following are examples only.

Table 14. VLAN Interfaces

VLAN	Switch1 IP	Switch2 IP	Active Gateway
2	10.252.0.2/17	10.252.0.3/17	10.252.0.1
4	10.254.0.2/17	10.254.0.3/17	10.254.0.1
7	TBD	TBD	TBD
10	10.11.0.2/17	10.11.0.3/17	10.11.0.1
2000	10.100.0.2/22	10.100.0.3/22	10.100.0.1/22
3000	10.104.0.2/22	10.104.0.3/22	10.104.0.1/22

```
sw-24g04(config)# vlan 2
sw-24g04(config-vlan-2)# interface vlan 2
sw-24g04(config-if-vlan)# vsx-sync active-gateways
sw-24g04(config-if-vlan)# ip mtu 9198
sw-24g04(config-if-vlan)# ip address 10.252.0.3/17
sw-24g04(config-if-vlan)# active-gateway ip mac 12:01:00:00:01:00
sw-24g04(config-if-vlan)# active-gateway ip 10.252.0.1

sw-24g03(config)# vlan 2
sw-24g03(config-vlan-2)# interface vlan 2
sw-24g03(config-if-vlan)# vsx-sync active-gateways
sw-24g03(config-if-vlan)# ip mtu 9198
sw-24g03(config-if-vlan)# ip address 10.252.0.2/17
```

```
sw-24g03(config-if-vlan)# active-gateway ip mac 12:01:00:00:01:00
sw-24g03(config-if-vlan)# active-gateway ip 10.252.0.1
```

8. Configure the bonds on ports connecting to the NCNs.

   This information can be found on the SHCD.

```
sw-24g04(config)# interface lag 1 multi-chassis
sw-24g04(config-lag-if)# no shutdown
sw-24g04(config-lag-if)# no routing
sw-24g04(config-lag-if)# vlan trunk native 1
sw-24g04(config-lag-if)# vlan trunk allowed 1-2,4,7,10
sw-24g04(config-lag-if)# lacp mode active
sw-24g04(config-lag-if)# lacp fallback

sw-24g04(config)# interface 1/1/1
sw-24g04(config)# no shutdown
sw-24g04(config)# mtu 9198
sw-24g04(config)# lag 1
```

23.22.5 Configure the Management Network Access Ports

Prerequisites

- The administrator has access to the switches.
- The SHCD is available.

About this task

ROLE	System administrator, system installer, network administrator
OBJECTIVE	Configure the access ports on the management network switches to enable access to the switches to transmit data to and from the specified VLANs.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Configure the edge ports.

   This information is included in the NMN/HMN/Mountain-TDS Management Tab in the SHCD. Typically, these are ports that are connected to iLOs (BMCs), gateway nodes, compute nodes, or chassis management module (CMM) switches.

```
sw-smn-001(config)# interface 1/1/28
sw-smn-001(config)# no shutdown
sw-smn-001(config)# mtu 9198
sw-smn-001(config)# description HMN
sw-smn-001(config)# no routing
sw-smn-001(config)# vlan access 4
```

2. Configure ports that go to the Node Management Network (NMN/VLAN2).

   Identify these ports by referencing the NMN tab on the SHCD.

```
sw-smn-001(config)# interface 1/1/6
sw-smn-001(config)# no shutdown
sw-smn-001(config)# mtu 9198
sw-smn-001(config)# description NMN
```

```
sw-smn-001(config)# no routing
sw-smn-001(config)# vlan access 2
```

3. Configure the User Access Node (UAN) ports.

UANs are have the same network connections as the previous HPE Cray EX system release. One connection will go to a NMN(VLAN2) access port, which is where the UAN will pxe boot and communicate with internal systems. See SHCD for UAN cabling. One Bond (two connections) will go to the MLAG/VSX pair of switches. This will be an access port for the CAN connection.

The following is the Aruba Configuration:

```
interface 1/1/16
    no shutdown
    mtu 9198
    no routing
    vlan trunk native 2
    vlan trunk allowed 2,7
    exit
```

The coolant distribution unit (CDU) switches have two cables connecting to each CMM. MC-LAG needs to be setup with the CDU switch pairs. The second CDU switch in the pair will have its port connected to the CMM shutdown. Redundancy is not yet available for the CMM.

a. Configure the first CDU.

```
sw-cdu-001(config)# int lag 2 multi-chassis
sw-cdu-001(config-lag-if)# vsx-sync vlans
sw-cdu-001(config-lag-if)# no shutdown
sw-cdu-001(config-lag-if)# description CMM_CAB_1000
sw-cdu-001(config-lag-if)# no routing
sw-cdu-001(config-lag-if)# vlan trunk native 2000
sw-cdu-001(config-lag-if)# vlan trunk allowed 2000,3000,4091
sw-cdu-001(config-lag-if)# lacp mode active
sw-cdu-001(config-lag-if)# lacp fallback
sw-cdu-001(config-lag-if)# exit

sw-cdu-001(config)# int 1/1/2
sw-cdu-001(config-if)# no shutdown
sw-cdu-001(config-if)# lag 2
sw-cdu-001(config-if)# exit
```

b. Configure the second CDU.

```
sw-cdu-002(config)# int lag 2 multi-chassis
sw-cdu-002(config-lag-if)# vsx-sync vlans
sw-cdu-002(config-lag-if)# shutdown
sw-cdu-002(config-lag-if)# description CMM_CAB_1000
sw-cdu-002(config-lag-if)# no routing
sw-cdu-002(config-lag-if)# vlan trunk native 2000
sw-cdu-002(config-lag-if)# vlan trunk allowed 2000,3000,4091
sw-cdu-002(config-lag-if)# lacp mode active
sw-cdu-002(config-lag-if)# lacp fallback
sw-cdu-002(config-lag-if)# exit

sw-cdu-002(config)# int 1/1/2
sw-cdu-002(config-if)# no shutdown
sw-cdu-002(config-if)# lag 2
sw-cdu-002(config-if)# exit
```

23.22.6 Configure the Management Network Uplink

Prerequisites

- The administrator has console access.
- Aruba VSX is configured on a pair of switches.

About this task

ROLE	System administrator, system installer, network administrator
OBJECTIVE	How to configure switch-to-switch connections or uplinks between switches. The example configuration in this procedure shows how to configure a Multi-chassis Link Aggregate Group (LAG) on a pair of Aruba VSX switches. These connections will go to other network switches.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Create the multi-chassis LAG on the first switch.

```
sw-24g03(config)# interface lag 100 multi-chassis
sw-24g03(config-lag-if)# no shutdown
sw-24g03(config-lag-if)# no routing
sw-24g03(config-lag-if)# vlan trunk native 1
sw-24g03(config-lag-if)# vlan trunk allowed all
sw-24g03(config-lag-if)# lacp mode active
```

2. Create the multi-chassis LAG on the second switch.

```
sw-24g04(config)# interface lag 100 multi-chassis
sw-24g04(config-lag-if)# no shutdown
sw-24g04(config-lag-if)# no routing
sw-24g04(config-lag-if)# vlan trunk native 1
sw-24g04(config-lag-if)# vlan trunk allowed all
sw-24g04(config-lag-if)# lacp mode active
```

3. Add ports to the LAG.

```
sw-24g03(config)# interface 1/1/48
sw-24g03(config-if)# no shutdown
sw-24g03(config-if)# mtu 9198
sw-24g03(config-if)# lag 100
sw-24g03(config-if)#

sw-24g04(config)# interface 1/1/48
sw-24g04(config-if)# no shutdown
sw-24g04(config-if)# mtu 9198
sw-24g04(config-if)# lag 100
sw-24g04(config-if)#
```

4. Configure the LAG on the access switch that is connected to the VSX pair.

```
sw-smn-001(config)# interface lag 1
sw-smn-001(config)# no shutdown
sw-smn-001(config)# no routing
sw-smn-001(config)# vlan trunk native 1
```

```
sw-smn-001(config)# vlan trunk allowed all
sw-smn-001(config)# lacp mode active

sw-smn-001(config)# interface 1/1/48
sw-smn-001(config-if)# no shutdown
sw-smn-001(config-if)# mtu 9198
sw-smn-001(config-if)# lag 100

sw-smn-001(config)# interface 1/1/49
sw-smn-001(config-if)# no shutdown
sw-smn-001(config-if)# mtu 9198
sw-smn-001(config-if)# lag 100
```

23.22.7 Configure the Management Network ACLs

Prerequisites

The administrator has access to the switches.

About this task

ROLE	System administrator, system installer, network administrator
OBJECTIVE	Create Access Control Lists (ACLs) and apply them to VLANs. ACLs are designed to block traffic from the Node Management Network (NMN) to and from the Hardware Management Network (HMN). These need to be set where the Layer3 interface is located, which will most likely be a VSX pair of switches. These ACLS are required on both switches in the pair.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Create the access list.

```
access-list ip nmn-hmn
    10 deny any 10.252.0.0/255.255.128.0 10.254.0.0/255.255.128.0
    20 deny any 10.252.0.0/255.255.128.0 10.104.0.0/255.252.0.0
    30 deny any 10.254.0.0/255.255.128.0 10.252.0.0/255.255.128.0
    40 deny any 10.254.0.0/255.255.128.0 10.100.0.0/255.252.0.0
    50 deny any 10.100.0.0/255.252.0.0 10.254.0.0/255.255.128.0
    60 deny any 10.100.0.0/255.252.0.0 10.104.0.0/255.252.0.0
    70 deny any 10.104.0.0/255.252.0.0 10.252.0.0/255.255.128.0
    80 deny any 10.104.0.0/255.252.0.0 10.100.0.0/255.252.0.0
    90 permit any any any
```

2. Apply the ACL to the VLANs.

   If using a coolant distribution unit (CDU) switch, apply the ACLs to the 2xxx and 3xxx VLANs.

```
sw-24g03(config)# vlan 2
sw-s24g03(config-vlan-2)# apply access-list ip nmn-hmn in
sw-s24g03(config-vlan-2)# apply access-list ip nmn-hmn out
sw-24g03(config)# vlan 4
```

```
sw-s24g03(config-vlan-4)# apply access-list ip nmn-hmn in
sw-s24g03(config-vlan-4)# apply access-list ip nmn-hmn out
```

23.22.8 Configure Layer3 Routing for the Management Network

Prerequisites

- The administrator has access to all of the switches.
- The SHCD is available.

About this task

ROLE	System administrator, system installer, network administrator
OBJECTIVE	Configure the Layer3 routing between the cooland distribution unit (CDU) switches and the Spine/Leaf switches.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

1. Start the Open Shortest Path First (OSPF) process.

   Give the switch a router-id, which is typically the Node Management Network (NMN) IP address. Border Gateway Protocol (BGP) will need to redistribute into OSPF to allow devices to communicate with Kubernetes.

   ```
   router ospf 1
       router-id 10.252.0.3
       redistribute bgp
       area 0.0.0.2
       area 0.0.0.4
   ```

2. Set the OSPF peering to happen over VLAN 2 and VLAN 4.

   ```
   interface vlan 2
       ip ospf 1 area 0.0.0.2
   interface vlan 4
       ip ospf 1 area 0.0.0.4
   ```

3. Change the BGP configuration on the switches to avoid routing loops.

   ```
   router bgp 65533
       distance bgp 85 70
   ```

4. Configure the CDU switch Layer3 configuration.

   ```
   router ospf 1
       router-id 10.252.0.6
       area 0.0.0.2
       area 0.0.0.4
   interface vlan 2
   ```

```
    ip ospf 1 area 0.0.0.2
interface vlan 4
    ip ospf 1 area 0.0.0.4
interface vlan 2000
    ip ospf 1 area 0.0.0.2
    ip ospf passive
interface vlan 3000
    ip ospf 1 area 0.0.0.4
    ip ospf passive
```

5. Verify the OSPF neighbors are on the CDU switches.

```
sw-cdu-002# show ip ospf neighbors
OSPF Process ID 1 VRF default
=============================

Total Number of Neighbors: 6

Neighbor ID      Priority  State             Nbr Address       Interface
-------------------------------------------------------------------------
10.252.0.2       1         FULL/DROther      10.252.0.2         vlan2

10.252.0.3       1         FULL/DROther      10.252.0.3         vlan2

10.252.0.5       1         FULL/BDR          10.252.0.5         vlan2

10.252.0.2       1         FULL/DROther      10.254.0.2         vlan4

10.252.0.3       1         FULL/DROther      10.254.0.3         vlan4

10.252.0.5       1         FULL/BDR          10.254.0.5         vlan4
```

The following route is needed for consistent PXE booting on Aruba switches. The second IP `10.252.1.10` will be an NCN worker node (`ncn-w001` in this example).

```
ip route 10.92.100.60/32 10.252.1.10
ip route 10.94.100.60/32 10.252.1.10
```

23.22.9 Configure SNMP for the Management Network

Prerequisites

The administrator has access to the switches.

About this task

ROLE	System administrator, system installer, network administrator
OBJECTIVE	Configure the Simple Network Management Protocol (SNMP) to enable hardware discovery on the HPE Cray EX system.
LIMITATIONS	None.
NEW IN THIS RELEASE	This procedure is new in release 1.4.

Procedure

Apply SNMP on all switches that are connected to the BMCs.

```
sw-smn-001(config)# snmp-server vrf default
sw-smn-001(config)# snmp-server system-contact "Contact Cray Global Technical Services (C.G.T.S.)"
sw-smn-001(config)# snmpv3 user testuser auth md5 auth-pass \
plaintext testpass1 priv des priv-pass plaintext text
```

23.22.1 Upgrades for Dell and Mellanox Switches 0

Dell and Mellanox switches have been upgraded for release 1.4 of the HPE Cray EX system.

IP Address Changes

Cray Site Init (CSI) generates the IP addresses for the switches on a 1.4 system. The IP addresses are located here `/var/www/ephemeral/prep/root/{system-name}/networks` directory on the LiveCD/ncn-m001. The following is a snippet of `NMN.yaml` file:

```
ip_reservations:
 - ip_address: 10.252.0.2
   name: sw-spine-001
   comment: x3000c0h41s1
   aliases: []
 - ip_address: 10.252.0.3
   name: sw-spine-002
   comment: x3000c0h41s2
   aliases: []
 - ip_address: 10.252.0.4
   name: sw-leaf-001
   comment: x3000c0w40
   aliases: []
```

Validate whether the switches match the `NMN.yaml`, `HMN.yaml`, and `CAN.yaml` files. If the IP addresses do not match, change the IP for the appropriate network. On most 1.3. x systems, the IP addresses will be as shown below, which will require them to be updated.

```
spine-01 10.252.0.1
spine-02 10.252.0.3
leaf-01 10.252.0.2
```

Verify that these are correct by doing an SSH into the device and checking the hostname or the switch configuration repository. The following is for Mellanox IPs:

```
sw-spine-001 [rocket-mlag-domain: standby] > ena
sw-spine-001 [rocket-mlag-domain: standby] # conf t
sw-spine-001 [rocket-mlag-domain: standby] (config) # no protocol magp
sw-spine-001 [rocket-mlag-domain: standby] (config interface vlan 2) # no ip address 10.252.0.1/17
sw-spine-001 [rocket-mlag-domain: standby] (config interface vlan 2) # ip address 10.252.0.2/17
```

If MAGP is enabled, it will need to be disabled before deleting the current IP. This needs to be turned back on and configured in the MAGP section.

For Dell IP addresses:

```
sw-leaf-001# configure terminal
sw-leaf-001(config)# interface vlan 2
sw-leaf-001(conf-if-vl-2)# ip address 10.252.0.4/17
```

When changing these IPs, make sure the IP that is currently in use is not changed. Ensure the IP change is done for all VLANs (1, 2, 4, 7, 10) and to write memory to save changes.

Mellanox write memory:

```
sw-spine-001 [rocket-mlag-domain: standby] (config) # write memory
```

Dell Write memory:

```
sw-leaf-001# write memory
```

Dell

The following changes have been made to Dell switches:

- Removed spanning-tree bpdufilter.
- Added spanning-tree bpduguard.
- Removed the IP-Helper if moving to a different switch.

```
sw-leaf-001# configure terminal
sw-leaf-001(config)# interface vlan 2
sw-leaf-001(conf-if-vl-2)# no ip helper-address 10.92.100.222

sw-leaf-001(config)# interface vlan 4
sw-leaf-001(conf-if-vl-4)# no ip helper-address 10.94.100.222
```

Release 1.3 (old) configuration:

```
!v1.3 config
interface ethernet1/1/3
 no shutdown
 switchport mode trunk
 switchport access vlan 1
 switchport trunk allowed vlan 2,4,7,10
 mtu 9216
 flowcontrol receive on
 flowcontrol transmit off
 spanning-tree bpdufilter enable
 spanning-tree port type edge
```

Release 1.4 (new) configuration:

```
!v1.4 config
interface ethernet1/1/2
 no shutdown
 switchport mode trunk
 switchport access vlan 1
 switchport trunk allowed vlan 2,4,7,10
 mtu 9216
 flowcontrol receive on
 flowcontrol transmit off
 spanning-tree bpduguard enable
 spanning-tree port type edge
```

Release 1.3 to 1.4 changes:

```
!v1.3 to v1.4 changes
configure terminal
interface ethernet 1/1/x
no spanning-tree bpdufilter
spanning-tree bpduguard enable
exit
write memory
```

Dell coolant distribution unit (CDU) changes:

```
interface port-channel1
 description CMM_CAB_1000
 no shutdown
 switchport mode trunk
 switchport access vlan 2000
```

```
switchport trunk allowed vlan 3000,4091
mtu 9216
vlt-port-channel 1
spanning-tree bpduguard enable
```

Mellanox

Multi-Active Gateway Protocol (MAGP) should be set for every VLAN interface. See *https://community.mellanox.com/s/article/howto-configure-magp-on-mellanox-switches*.

- Spine01:

```
(config) # protocol mlag
(config) # interface port-channel 100
(config) # interface ethernet 1/14 channel-group 100 mode active
(config) # interface ethernet 1/13 channel-group 100 mode active
(config) # interface ethernet 1/13 dcb priority-flow-control mode on force
(config) # interface ethernet 1/14 dcb priority-flow-control mode on force
(config) # vlan 4000
(config) # interface vlan 4000
(config) # interface port-channel 100 ipl 1
(config) # interface port-channel 100 dcb priority-flow-control mode on force
(config interface vlan 4000) # ip address 192.168.255.254 255.255.255.252
(config interface vlan 4000) # ipl 1 peer-address 192.168.255.253
(config) # mlag system-mac 00:00:5E:00:01:5D
(config) # no mlag shutdown
```

- Spine02:

```
(config) # protocol mlag
(config) # interface port-channel 100
(config) # interface ethernet 1/14 channel-group 100 mode active
(config) # interface ethernet 1/13 channel-group 100 mode active
(config) # interface ethernet 1/13 dcb priority-flow-control mode on force
(config) # interface ethernet 1/14 dcb priority-flow-control mode on force
(config) # vlan 4000
(config) # interface vlan 4000
(config) # interface port-channel 100 ipl 1
(config) # interface port-channel 100 dcb priority-flow-control mode on force
(config interface vlan 4000) # ip address 192.168.255.253 255.255.255.252
(config interface vlan 4000) # ipl 1 peer-address 192.168.255.254
(config) # mlag system-mac 00:00:5E:00:01:5D
(config) # no mlag shutdown
```

The Multi-Chassis Link Aggregation Group (MLAG) needs to be configured on each Mellanox spine switch.

- Spine01:

```
(config) # int mlag-port-channel 1
(config interface mlag-port-channel 1) # mtu 9216 force
(config interface mlag-port-channel 1) # switchport mode hybrid
(config interface mlag-port-channel 1) # no shutdown
(config interface mlag-port-channel 1) # lacp-individual enable force
(config interface mlag-port-channel 1) # switchport hybrid allowed-vlan add 2
(config interface mlag-port-channel 1) # switchport hybrid allowed-vlan add 4
(config interface mlag-port-channel 1) # switchport hybrid allowed-vlan add 7
(config interface mlag-port-channel 1) # switchport hybrid allowed-vlan add 10
```

- Spine02:

```
(config) # int mlag-port-channel 1
(config interface mlag-port-channel 1) # mtu 9216 force
(config interface mlag-port-channel 1) # switchport mode hybrid
(config interface mlag-port-channel 1) # no shutdown
(config interface mlag-port-channel 1) # switchport hybrid allowed-vlan add 2
(config interface mlag-port-channel 1) # switchport hybrid allowed-vlan add 4
(config interface mlag-port-channel 1) # switchport hybrid allowed-vlan add 7
(config interface mlag-port-channel 1) # switchport hybrid allowed-vlan add 10
```

Ports need to be added once the MLAG is created:

```
(config) # interface ethernet 1/1
(config interface ethernet 1/1) # mlag-channel-group 1 mode active
```

```
(config interface ethernet 1/1) # interface ethernet 1/1 speed 40G force
(config interface ethernet 1/1) # interface ethernet 1/1 mtu 9216 force
```

Configuration with the recommended MLAG-VIP cable is meant to prevent "split brain," which is where both spines think they are the active gateway. It requires a RJ45 cable between the mgmt0 ports on both switches.

- Spine01:

```
no interface mgmt0 dhcp
   interface mgmt0 ip address 192.168.255.241 /29
no mlag shutdown
   mlag system-mac 00:00:5E:00:01:5D
mlag-vip rocket-mlag-domain ip 192.168.255.242 /29 force
```

- Spine02:

```
no interface mgmt0 dhcp
   interface mgmt0 ip address 192.168.255.243 /29
no mlag shutdown
   mlag system-mac 00:00:5E:00:01:5D
mlag-vip rocket-mlag-domain ip 192.168.255.242 /29 force
```

Verify mlag-vip:

```
sw-spine-001 [rocket-mlag-domain: master] # show mlag-vip
MLAG-VIP:
 MLAG group name: rocket-mlag-domain
 MLAG VIP address: 192.168.255.242/29
 Active nodes: 2

-------------------------------------------------------------------------------
Hostname                                VIP-State            IP Address
-------------------------------------------------------------------------------
sw-spine01                              master               192.168.255.241
sw-spine02                              standby              192.168.255.243
```

24 Slingshot Post-Installation Tasks

Slingshot post-installation tasks are run when the compute nodes are booted. These typically include:

- Verifying the fabric is up
- Configure NTP
- Configure Syslog Forwarding
- Telemetry
- LAG Configuration on Slingshot Switch with Controller

Refer to the Slingshot documentation included with HPE Cray EX 1.4 documentation for troubleshooting information. See the *Slingshot Documentation* topic for reference.

Verify the Fabric is Up

The compute nodes should be booted to verify that the fabric is up.

1. Access the Fabric Manager pod.

```
ncn-m001# kubectl get pods -A |grep fabric |awk '{print $2}'
slingshot-fabric-manager-5dc448779c-d8t72
```

```
ncn-m001# kubectl exec -it -n services slingshot-fabric-manager-5dc448779c-d8t72 -- /bin/bash
Defaulting container name to slingshot-fabric-manager.
Use 'kubectl describe pod/slingshot-fabric-manager-5dc448779c-d8t72 -n services' to see all of the
containers in this pod.
```

2. To run diagnostics on switches, fabric, and edge links, start STT.

```
slingshot-fabric-manager:# slingshot-topology-tool
```

   a. Show the switches.

```
(STT) show switches
Working with 'default' topology and 'default' filter profile.
Collecting data using 'check-switches' script.
Collecting data using 'dgrerrstat' script.
Point2Point file is not set for this profile !!!
Collecting data using 'dgrperfcheck' script.
Warning:perfcheck data not available.
+------------+----------+--------+------+------+------------+--------------+--------+----------+------+-----------
+-------+--------+-------+--------+-------------+
|   xname    |   type   | snum | gnum | edge_count | fabric_count | uptime | up_ports | flap | checkidle |
pktIn | pktOut | drops | dscrds | mcast_found |
+------------+----------+--------+------+------+------------+--------------+--------+----------+------+-----------
+-------+--------+-------+--------+-------------+
| x3000c0r21 | Columbia |  0   |  0   |     12     |      0       |  8:18  |  11/64   |  0   |    idle   |
44  |  484   |   0   |   44   |    False    |
+------------+----------+--------+------+------+------------+--------------+--------+----------+------+-----------
+-------+--------+-------+--------+-------------+
```

   b. Show the fabric switches.

```
(STT) show fabric

Working with 'default' topology and 'default' filter profile.
Collecting data using 'check-switches' script.
Collecting data using 'check-fabric' script.
```

```
Point2Point file is not set for this profile !!!
Collecting data using 'dgrlinkstat' script.
Collecting data using 'dgrperfcheck' script.
Warning:perfcheck data not available.
+-------+------+-----+----------+--------+-------+---------+-----+-----+---------+-------+-------+---------
+---------+
| xname | type | dst | hostname | status | ptype | subtype | mtu | mac | mactype | speed | media | mcast_a |
mcast_b |
+-------+------+-----+----------+--------+-------+---------+-----+-----+---------+-------+-------+---------
+---------+
+-------+------+-----+----------+--------+-------+---------+-----+-----+---------+-------+-------+---------
+---------+
```

c. Show edge switches.

```
(STT) show edge
``Working with ``'default'` `topology and ``'default'` `filter profile.``
+-----------------+------+------+----------+--------+----------+---------+------+-------------------
+-------------+---------+------------+---------+---------+``
|   xname   | ``type` `| dst | ``hostname` `| status | ptype  | subtype | mtu |    mac    |  mactype  | speed
|  media  | mcast_a | mcast_b |``
+-----------------+------+------+----------+--------+----------+---------+------+-------------------
+-------------+---------+------------+---------+---------+``
| x3000c0r21j4p0 | edge | None |     | True | Ethernet |  Edge | 9216 | 02:00:00:00:00:03 | algorithmic |
BJ_100G | electrical |  0  |  0  |``
| x3000c0r21j4p1 | edge | None |     | True | Ethernet |  Edge | 9216 | 02:00:00:00:00:02 | algorithmic |
BJ_100G | electrical |  0  |  0  |``
| x3000c0r21j8p0 | edge | None |     | True | Ethernet |  Edge | 9216 | 02:00:00:00:00:07 | algorithmic |
BJ_100G | electrical |  0  |  0  |``
| x3000c0r21j8p1 | edge | None |     | False | Ethernet |  Edge | 9216 | 02:00:00:00:00:06 | algorithmic |
BJ_100G | electrical |  0  |  0  |``
| x3000c0r21j10p1 | edge | None |     | True | Ethernet |  Edge | 9216 | 02:00:00:00:00:09 | algorithmic |
BJ_100G | electrical |  0  |  0  |``
| x3000c0r21j10p0 | edge | None |     | True | Ethernet |  Edge | 9216 | 02:00:00:00:00:08 | algorithmic |
BJ_100G | electrical |  0  |  0  |``
| x3000c0r21j14p0 | edge | None |     | True | Ethernet |  Edge | 9216 | 02:00:00:00:00:0d | algorithmic |
BJ_100G | electrical |  0  |  0  |``
| x3000c0r21j14p1 | edge | None |     | True | Ethernet |  Edge | 9216 | 02:00:00:00:00:0c | algorithmic |
BJ_100G | electrical |  0  |  0  |``
| x3000c0r21j6p1 | edge | None |     | True | Ethernet |  Edge | 9216 | 02:00:00:00:00:15 | algorithmic |
BJ_100G | electrical |  0  |  0  |``
| x3000c0r21j6p0 | edge | None |     | True | Ethernet |  Edge | 9216 | 02:00:00:00:00:14 | algorithmic |
BJ_100G | electrical |  0  |  0  |``
| x3000c0r21j12p0 | edge | None |     | True | Ethernet |  Edge | 9216 | 02:00:00:00:00:1b | algorithmic |
BJ_100G | electrical |  0  |  0  |``
| x3000c0r21j12p1 | edge | None |     | True | Ethernet |  Edge | 9216 | 02:00:00:00:00:1a | algorithmic |
BJ_100G | electrical |  0  |  0  |``
+-----------------+------+------+----------+--------+----------+---------+------+-------------------
+-------------+---------+------------+---------+---------+
```

d. Show cables. Use the pipe command to identify problems: `show cables | grep fabric | egrep 'Not|Wrong'`.

```
(STT) show cables
Working with ``'default'` `topology and ``'default'` `filter profile.
Collecting data using ``'check-switches'` `script.
Collecting data using ``'check-fabric'` `script.
Collecting data using ``'dgrlinkstat'` `script.
Error with SSH to x3000c0r15b0:
terminate called after throwing an instance of ``'std::invalid_argument'
 ``what(): stoul
+-----------------+-----------------+------------------+------------------+------+---------------
+-----------------------------------------+
|    srca    |    srcb    |    dsta    |    dstb    | type |   status  |        serial_ids        |
+-----------------+-----------------+------------------+------------------+------+---------------
+-----------------------------------------+
| x3000c0r15j14p1 | x3000c0r15j14p0 | x3000c0s4b0n0h1 | x3000c0s4b0n0h0 | edge | Not Connected |
``141100026``, ``141100026``, unknown, unknown |
| x3000c0r15j12p1 | x3000c0r15j12p0 | x3000c0s5b0n0h1 | x3000c0s5b0n0h0 | edge | Not Connected |
``0609199037``, ``0609199037``, unknown, unknown |
| x3000c0r15j10p1 | x3000c0r15j10p0 | x3000c0s6b0n0h1 | x3000c0s6b0n0h0 | edge | Not Connected |
``0616190087``, ``0616190087``, unknown, unknown |
| x3000c0r15j8p1 | x3000c0r15j8p0 | x3000c0s17b0n0h0 | x3000c0s17b0n0h1 | edge | Not Connected |
``1206189028``, ``1206189028``, unknown, unknown |
+-----------------+-----------------+------------------+------------------+------+---------------
+-----------------------------------------+
```

e. Run shell commands from the STT CLI using:

```
shell <shell command>
```

```
(STT) shell ping x1003c4r1b0
    PING x1003c4r1b0 (10.104.12.27) 56(84) bytes of data.
```

```
    64 bytes from x1003c4r1b0 (10.104.12.27): icmp_seq=1 ttl=62 time=0.239 ms
    64 bytes from x1003c4r1b0 (10.104.12.27): icmp_seq=2 ttl=62 time=0.207 ms
    64 bytes from x1003c4r1b0 (10.104.12.27): icmp_seq=3 ttl=62 time=0.233 ms
    64 bytes from x1003c4r1b0 (10.104.12.27): icmp_seq=4 ttl=62 time=0.222 ms
    ^C
    --- x1003c4r1b0 ping statistics ---
    4 packets transmitted, 4 received, 0% packet loss, time 3062ms
    rtt min/avg/max/mdev = 0.207/0.225/0.239/0.016 ms
```

f. Pipe STT commands just like shell commands:

```
\<diagnostic command> | \<shell commands> > \<output_file>
```

```
(STT) show cables | grep x1003c2r1j11 >>
cable_log_x1003c2r1j11
```

```
(STT) shell cat cable_log_x1003c2r1j11
|  x1003c2r1j11p1 |  x1003c2r1j11p0 |  x1003c3r7j18p1  |  x1003c3r7j18p0  |
fabric |   Connected   |   0211190120, 0211190120, 0211190120, 0211190120   |
```

Detailed information about STT is provided with the Slingshot documentation.

Configure NIC Ethernet Interfaces to Send Traffic on Edge Links

This is an optional step. Use the following procedure in case the compute nodes are not booted after the fabric is brought up.

1. Configure HSN IP addresses.

```
slingshot-fabric-manager# fmn_fabric_bringup -n
INFO: Configuring NIC HSN data to agent desiredStatus
```

2. Reboot compute nodes.
3. Login to a compute node.
4. Verify that compute node HSN interfaces have IP and MAC addresses.

```
nid000001# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000 \
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00 inet 127.0.0.1/8 scope host lo valid_lft \
forever preferred_lft forever inet6 ::1/128 scope host valid_lft forever preferred_lft forever

2: hsn0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP group default qlen 1000 link/
ether \
02:00:00:00:00:02 brd ff:ff:ff:ff:ff:ff inet 10.253.0.2/24 scope global hsn0 valid_lft forever
preferred_lft \
forever inet6 fe80::ff:fe00:2/64 scope link valid_lft forever preferred_lft forever

3: nmn0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000 link/
ether \
a4:bf:01:3e:fb:8a brd ff:ff:ff:ff:ff:ff inet 10.252.50.7/17 brd 10.252.127.255 scope global nmn0
valid_lft \
forever preferred_lft forever inet6 fe80::a6bf:1ff:fe3e:fb8a/64 scope link valid_lft forever
preferred_lft forever

4: net1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group default qlen 1000
link/ether \
a4:bf:01:3e:fb:8b brd ff:ff:ff:ff:ff:ff
```

Configure NTP

1. Log on to a Kubernetes worker or master node.
2. Find the Network Time Protocol (NTP) server.

```
ncn-m001# eval "ip -br a ls dev vlan004 | awk '{ print \$3 }' | sed 's|\(.*\)/.*|\1|'"

10.254.1.14
```

3. Access the Fabric Manager pod.

```
ncn-m001# kubectl get pods -A |grep fabric |awk '{print $2}'
slingshot-fabric-manager-5dc448779c-d8t72

ncn-m001:~# kubectl exec -it -n services slingshot-fabric-manager-5dc448779c-d8t72 -- /bin/bash
Defaulting container name to slingshot-fabric-manager.
Use 'kubectl describe pod/slingshot-fabric-manager-5dc448779c-d8t72 -n services' to see all of the
containers in this pod.
```

4. Set NTP server to all agents and send requests to the switches. Multiple NTP servers are not supported for this release.

```
slingshot-fabric-manager# curl  --request PATCH -H "Content-Type: application/json"
-d '{ "fabricPropertyMap":{ "NTPServer" : "10.254.1.14" } }'
http://127.0.0.1:8000/fabric/topology-policies/template-policy | jq
```

5. SSH to a switch and verify that NTP server IP is configured:

```
slingshot-fabric-manager# ssh x3000c0r21b0
root@x3000c0r21b0's password:

root@x3000c0r21b0:~# cat /nvram/startup-config
# BMC SystemConfig
!
interface management 0
  ipv6 address slaac
  no shutdown
  ip address dhcp
!
interface ethernet 0
  no shutdown
!
user-key root cVUVUXXX
!
hostname x3000c0r21b0
ntp 10.254.1.14
```

Configure Syslog Forwarding

1. Log on to a Kubernetes worker or master node.
2. Get the log forwarding IP address:

```
ncn-m001# kubectl -n sma get services | grep rsyslog-aggregator-hmn-udp | awk '{ print $4 }'
10.94.100.3
```

3. Access the Fabric Manager pod.

```
ncn-m001# kubectl get pods -A |grep fabric |awk '{print $2}'
slingshot-fabric-manager-5dc448779c-d8t72

ncn-m001# kubectl exec -it -n services slingshot-fabric-manager-5dc448779c-d8t72 -- /bin/bash
Defaulting container name to slingshot-fabric-manager.
Use 'kubectl describe pod/slingshot-fabric-manager-5dc448779c-d8t72 -n services' to see all of the
containers in this pod.
```

4. Configure log forwarding and set `SyslogServer` for all agents:

```
slingshot-fabric-manager# curl --request PATCH -H "Content-Type: application/json"
-d '{ "fabricPropertyMap":{ "SyslogServer" : "10.94.100.3" } }'
http://127.0.0.1:8000/fabric/topology-policies/template-policy |jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   912  100   854  100    58   833k  58000 --:--:-- --:--:-- --:--:--  890k
{
  "active": true,
  "healthMapLink": "/fabric/topology-maps/template-map",
  "routingEngineLink": "/fabric/routing-engines/dragonfly/template-routing",
  "deploymentEngineLink": "/fabric/deployment-engines/template-policy",
  "switchGroupLink": "/fabric/switch-groups/template-switches",
  "linkGroupLink": "/fabric/link-groups/template-links",
```

```
  "routingPropertyMap": {
    "numGroups": "1",
    "maxNumLocalSwitches": "2"
  },
  "fabricPropertyMap": {
    "SyslogServer": "10.94.100.3",
    "MAX_LAG_PORTS": "256",
    "NTPServer": "10.254.1.26",
    "MTU": "9216",
    "LLDP": "true"
  },
  "lagPropertyMap": {},
  "documentVersion": 8,
  "documentEpoch": 0,
  "documentKind": "com:services:fabric:TopologyPolicyState",
  "documentSelfLink": "/fabric/topology-policies/template-policy",
  "documentUpdateTimeMicros": 1613748098969003,
  "documentUpdateAction": "PATCH",
  "documentExpirationTimeMicros": 0,
  "documentOwner": "8475245d-3999-48a7-b9fc-8addc998a13e"
}
```

5. Verify that the logging server IP address is set on the switch:

```
slingshot-fabric-manager# ssh x3000c0r21b0
root@x3000c0r21b0's password:

root@x3000c0r21b0:~# cat /nvram/startup-config
# BMC SystemConfig
!
interface management 0
  ipv6 address slaac
  no shutdown
  ip address dhcp
!
interface ethernet 0
  no shutdown
!
user-key root c3NoLX
!
hostname x3000c0r21b0
ntp 10.254.0.4
logging 10.94.100.3:514
```

Set Telemetry Collector Endpoint

> **ATTENTION:** In release 1.4 it's possible for the slingshot-fabric-manager pod to exceed its storage quota. If this occurs, delete `FabricHost.8000.0.log` file and see the workaround below **Workaround to Modify Telemetry and Avoid Filling up Fabric Manager Container Disk Quota**.

The telemetry endpoint must be configured for switches to send metrics to the Fabric Manager. To configure the telemetry collector endpoint, run:

```
slingshot-fabric-manager# fmn_config_telemetry |jq
```

If the version of Fabric Manager scripts is lower than 0.1.135 (version can be obtained from `rpm -qi fmn-minimal |grep Version`), set the telemetry endpoint as follows:

```
read -ra ar <<(hostname --ip-address)
IP_HOST=${ar[ar[@]-1]}

FMN_BASE_URL=""
FMN_BASE_URL_SHASTA="https://api-gw-service-nmn.local/apis/fabric-manager"

[[ -f /root/.slingshotbuild ]] && FMN_BASE_URL=$FMN_BASE_URL_SHASTA
config=`jq -n --arg FMN_BASE_URL "$FMN_BASE_URL" \
'{
   collector: { value: ( $FMN_BASE_URL + "/telemetry-collector" ) }
 }
'`
```

```
curl -X PATCH                                          \
   -H "Content-Type: application/json"             \
   http://127.0.0.1:8000/switch-telemetry/settings \
   -d "${config}"
```

Modifying the Fabric Telemetry Configuration

The periodicity of fabric telemetry collection and the data being posted to telemetry is configurable.

Configurable	Description
Periodicity	Rate of data collection in seconds, range 0-300 (zero disables)
Collector	Telemetry collector address
Locality	Enable additional locality information (Dragonfly location, link type)
Statistics	Set of statistics to be collected (SNMP MIBs 1213, 2819, 2863, 3635, 4188)
Counters	Switch counters to be collected (ibuf, obuf, llr)

To query the Fabric Telemetry configuration, log on to the Fabric Manager pod and run the following:

```
slingshot-fabric-manager# curl -X GET http://127.0.0.1:8000/switch-telemetry/settings |jq
 {
 "collector": {
   "value": "172.29.68.22:8000/telemetry-collector"
 },
 "counters": {
   "values": []
 },
 "locality": {
   "enable": true
 },
 "periodicity": {
   "value": 1
 },
 "statistics": {
   "values": []
 }
 }
```

Both PATCH and PUT operations modify the fabric telemetry configuration.

An example of how to enable collection of RFC 3635 statistics is presented:

1. Create a file named `modifyStatistics.json`.

   ```
   slingshot-fabric-manager:/opt/slingshot# echo '
   {
     "statistics": {
       "values": [
         3635
       ]
     }
   }' > modifyStatistics.json
   ```

   You can verify by using this command:

   ```
   cat modifyStatistics.json |jq
   ```

2. Run cURL to apply the change:

```
slingshot-fabric-manager# curl -X PATCH http://127.0.0.1:8000/switch-telemetry/settings -d
@modifyStatistics.json -H "Content-Type: application/json" | jq ""
```

Workaround to Modify Telemetry and Avoid Filling up Fabric Manager Container Disk Quota

ATTENTION: In release 1.4 it's possible for the slingshot-fabric-manager pod to exceed its storage quota. Use this workaround to avoid exceeding the disk quota which sets periodicity to 10000 and statistics to "empty".

1. From the fabric manager pod, check the disk usage.

```
slingshot-fabric-manager#  du -h /opt/slingshot/data/slingshot/fabric-manager
```

2. Create a file named `fix_disk_quota.json` and add these settings:

```
{
      "collector": {
          "value": "10.46.0.24:8000/telemetry-collector"
  },
      "counters": {
          "values": []
  },
      "locality": {
    *     *"enable": true
  },
      "periodicity": {
          "value": 10000
  },
      "statistics": {
          "values": []
  }
}
```

3. Apply the change.

```
slingshot-fabric-manager# curl -X PATCH http://127.0.0.1:8000/switch-telemetry/settings \
-d @fix_disk_quota.json -H "Content-Type: application/json" | jq ""
```

4. In extreme case, disable telemetry by patching the collector address.

```
slingshot-fabric-manager# curl -X PATCH -k http://localhost:8000/switch-telemetry/settings \
-d'{"collector":{"value": "http://"' -H "Content-Type: application/json"}}
```

HMSCollector Forwarding

In version 1.4, Slingshot switches send telemetry packets to the Fabric Manager and are stored in a telemetry database by default. Use this procedure to forward packets to the HMSCollector.

To enable:

```
slingshot-fabric-manager# curl -X PATCH http://127.0.0.1:8000/fabric/topology-policies/template-policy
-d \ '{ "fabricPropertyMap": { "HMSCollector" : "istio-ingressgateway.hmnlb" } }' \
 -H "Content-Type: application/json" |jq
```

To disable:

```
slingshot-fabric-manager# curl -X PATCH http://127.0.0.1:8000/fabric/topology-policies/template-policy
-d \ '{ "fabricPropertyMap": { "HMSCollector" : "" } }' -H "Content-Type: application/json" |jq
```

Accessing Telemetry

When System Monitoring Framework (SMF) is present, telemetry is stored in a PostgreSQL (also known as Postgres) database. Accessing the data is typically done with tools like Grafana. The fabric telemetry data can also be accessed directly via SQL queries.

There are three tables for fabric telemetry in the Postgres database: fabric_view, fabric_crit_view, and fabric_perf_view. This section describes how to access the data in these tables using `psql`, the PostgreSQL interactive terminal.

1. From a worker node, find the name of the Kubernetes pods for the Postgres cluster:

```
ncn-w# kubectl -n sma get pods | grep postgres-cluster
sma-postgres-cluster-0                    1/1     Running             0          5d2h
sma-postgres-cluster-1                    1/1     Running             0          5d1h
```

2. Execute the bash command in the first pod to start a shell in the pod.

```
ncn-w# kubectl -n sma exec -it sma-postgres-cluster-0 /bin/bash
This container is managed by supervisord, when stopping/starting services use supervisorctl

Examples:

supervisorctl stop cron
supervisorctl restart patroni

Current status: (supervisorctl status)

cron                             RUNNING   pid 32, uptime 5 days, 1:35:35
patroni                          RUNNING   pid 33, uptime 5 days, 1:35:35
pgq                              RUNNING   pid 34, uptime 5 days, 1:35:35
root@craysma-postgres-cluster-0:/home/postgres#
```

3. Open a PostgreSQL interactive terminal against the pmdb database with the pmdbuser user.

```
sma-postgres-cluster-0# psql pmdb pmdbuser
psql (11.2 (Ubuntu 11.2-1.pgdg18.04+1))
Type "help" for help.

pmdb=>
```

4. View data from the various tables that contain fabric telemetry.

```
pmdb=> select * from pmdb.fabric_perf_view limit 5;
         timestamp          |   location   | sensor_type | parental_context | parental_index | physical_context |
index | physical_sub_context | device_specific_context | sub_index | value
---------------------------+--------------+-------------+------------------+----------------+------------------
+-------+----------------------+-------------------------+-----------+-------
 2020-11-03 17:27:18.942+00 | localhostl14 | Congestion  |                  |                | rxBW
|     1 |                      | global                  |        14 |  3283
 2020-11-03 17:27:18.942+00 | localhostl15 | Congestion  |                  |                | rxBW
|     1 |                      | global                  |        15 |  3284
 2020-11-03 17:27:18.943+00 | localhostl34 | Congestion  |                  |                | rxBW
|     1 |                      | local                   |        34 |  9339
 2020-11-03 17:27:18.943+00 | localhostl35 | Congestion  |                  |                | rxBW
|     1 |                      | local                   |        35 |  9339
 2020-11-03 17:27:18.943+00 | localhostl50 | Congestion  |                  |                | rxBW
|     1 |                      | local                   |        50 |  9339
(5 rows)
pmdb=> select * from pmdb.fabric_crit_view limit 5;
         timestamp          |   location   | sensor_type | parental_context | parental_index | physical_context |
index | physical_sub_context | device_specific_context | sub_index |  value
---------------------------+--------------+-------------+------------------+----------------+------------------
+-------+----------------------+-------------------------+-----------+---------
(0 rows)
pmdb=> select * from pmdb.fabric_view limit 5;
         timestamp          |   location   | sensor_type | parental_context | parental_index | physical_context  |
index | physical_sub_context | device_specific_context | sub_index | value
---------------------------+--------------+-------------+------------------+----------------+-------------------
+-------+----------------------+-------------------------+-----------+-------
 2020-11-03 17:19:12.968+00 | localhostl12 | LinkErrors  |                  |                | CableStateChange
|       |                      | edge                    |        12 |     1
 2020-11-03 17:19:12.968+00 | localhostl13 | LinkErrors  |                  |                | CableStateChange
|       |                      | edge                    |        13 |     1
 2020-11-03 17:19:12.968+00 | localhostl14 | LinkErrors  |                  |                | CableStateChange
|       |                      | global                  |        14 |     1
 2020-11-03 17:19:12.968+00 | localhostl15 | LinkErrors  |                  |                | CableStateChange
|       |                      | global                  |        15 |     1
 2020-11-03 17:19:12.968+00 | localhostl15 | LinkErrors  |                  |                | SerdesStateChange
|       |                      | global                  |        15 |     1
(5 rows)
```

Access Telemetry Data using OData query

After telemetry is configured to collect metrics, the metrics data is stored in the Fabric Manager datastore. The metrics data can be viewed through the Fabric Manager API with Open Data Protocol (OData) query and syntax.

Log on to a Fabric Manager pod and run these queries. To run these queries, you need to find the value of fmn-baseUrl in your environment.

```
ncn-m001# kubectl get services -A |grep fabric
services       slingshot-fabric-manager        ClusterIP      10.29.106.221   <none>
```

In this example, `fmn-baseUrl` is 10.29.106.221.

1. top - returns top numbers of metrics from the database. They are not guaranteed to be the latest metrics unless the query is combined with orderby timestamp in descending order.

```
slingshot-fabric-manager# curl -LG 'http://{fmn-baseUrl}/metrics?$top=1' |jq
{
    "totalCount": 1,
    "documentLinks": [
        "/metrics/e1ed54dc19deb6755b8ceb3cc4253"
    ],
    "documentCount": 1,
    "queryTimeMicros": 9000,
    "documentVersion": 0,
    "documentUpdateTimeMicros": 0,
    "documentExpirationTimeMicros": 0,
    "documentOwner": "5d4eec49-1478-4807-a874-7a7ff885ce71"
}
```

2. limit returns number of metrics per page. You can navigate to the next or previous page by click the "nextPageLink" or "prevPageLink".

```
slingshot-fabric-manager# curl -LG 'http://{fmn-baseUrl}/metrics?$limit=3'
{
    "totalCount": 3,
    "documentLinks": [
        "/metrics/e1ed54dc19deb6755b8ceb3cc4253",
        "/metrics/e1ed54dc19deb6755b8ceb3cc462e",
        "/metrics/e1ed54dc19deb6755b8ceb3cc4637"
    ],
    "documentCount": 3,
    "nextPageLink": "/metrics?path=1610576732165000&peer=5d4eec49-1478-4807-
a874-7a7ff885ce71",
    "queryTimeMicros": 5000,
    "documentVersion": 0,
    "documentUpdateTimeMicros": 0,
    "documentExpirationTimeMicros": 0,
    "documentOwner": "5d4eec49-1478-4807-a874-7a7ff885ce71"
}
```

3. count - returns the total number of metrics in a query result.

```
slingshot-fabric-manager# curl -LG 'http://{fmn-baseUrl}/metrics?$count=true'
{
    "totalCount": 12490,
    "documentLinks": [],
    "documentCount": 12490,
    "queryTimeMicros": 66999,
    "documentVersion": 0,
```

```
    "documentUpdateTimeMicros": 0,
    "documentExpirationTimeMicros": 0,
    "documentOwner": "5d4eec49-1478-4807-a874-7a7ff885ce71"
}
```

4. filter - returns a subset of metrics match criteria specified in the query option. This example uses "eq"(equal) logical operator.

```
slingshot-fabric-manager# curl -LG 'http://{fmn-baseUrl}/metrics?$filter=Location
%20eq%20%27x2006c0r39a0l1%27&$top=1'
{
    "totalCount": 1,
    "documentLinks": [
        "/metrics/e1ed54dc19deb6755b8ceb3cc4253"
    ],
    "documentCount": 1,
    "queryTimeMicros": 6000,
    "documentVersion": 0,
    "documentUpdateTimeMicros": 0,
    "documentExpirationTimeMicros": 0,
    "documentOwner": "5d4eec49-1478-4807-a874-7a7ff885ce71"
}
```

5. expand - returns a view of the metric data inline:

```
slingshot-fabric-manager# curl -LG 'http://{fmn-baseUrl}/metrics?$filter=Location
%20le%20%27x2006c0r39a0l1%27&$expand'
{
    "totalCount": 1,
    "documentLinks": [
        "/metrics/e1ed54dc19deb6755b8ceb3cc4253"
    ],
    "documents": {
        "/metrics/e1ed54dc19deb6755b8ceb3cc4253": {
            "Timestamp": "2019-01-17T19:08:06Z",
            "TimestampLong": 1547752086000,
            "Location": "x2006c0r39a0l1",
            "PhysicalContext": "Counters.txMulticastPkts",
            "ParentalIndex": 0,
            "Index": 0,
            "SubIndex": 0,
            "Value": "33303",
            "NumberValue": 33303.0,
            "documentVersion": 0,
            "documentEpoch": 0,
            "documentKind": "com:services:fabric:telemetry:models:Metric",
            "documentSelfLink": "/metrics/e1ed54dc19deb6755b8ceb3cc4253",
            "documentUpdateTimeMicros": 1610572802836001,
            "documentUpdateAction": "POST",
            "documentExpirationTimeMicros": 1610659202836000,
            "documentOwner": "5d4eec49-1478-4807-a874-7a7ff885ce71"
        }
    },
    "documentCount": 1,
    "queryTimeMicros": 2000,
    "documentVersion": 0,
    "documentUpdateTimeMicros": 0,
    "documentExpirationTimeMicros": 0,
    "documentOwner": "5d4eec49-1478-4807-a874-7a7ff885ce71"
}
```

6. select - returns a subset of properties specified in the query option:

```
slingshot-fabric-manager# curl -LG 'http://{fmn-baseUrl}/metrics?$filter=Location
%20le%20%27x2006c0r39a0l1%27&$select=Location,Value'
{
    "totalCount": 1,
    "documentLinks": [
        "/metrics/e1ed54dc19deb6755b8ceb3cc4253"
    ],
    "documents": {
        "/metrics/e1ed54dc19deb6755b8ceb3cc4253": {
            "Location": "x2006c0r39a0l1",
            "Value": "33303"
        }
    },
    "documentCount": 1,
    "queryTimeMicros": 1000,
    "documentVersion": 0,
    "documentUpdateTimeMicros": 0,
    "documentExpirationTimeMicros": 0,
    "documentOwner": "5d4eec49-1478-4807-a874-7a7ff885ce71"
}
```

7. orderby - returns a sorted view of metrics. Only the list of metrics in documentLinks are ordered. If $orderby is used with $expand, the expanded view of metrcis is not ordered.

```
slingshot-fabric-manager# curl -LG 'http://{fmn-baseUrl}/metrics?$filter=Location
%20le%20%27x2006c0r39a0l1%27&$orderby=TimestampLong%20desc&$orderbytype=LONG'
{
    "totalCount": 3,
    "documentLinks": [
        "/metrics/e1ed54dc19deb6755b8ceb3cc4a09",
        "/metrics/e1ed54dc19deb6755b8ceb3335da6",
        "/metrics/e1ed54dc19deb6755b8ceb3cc4dfc"
    ],
    "documentCount": 3,
    "queryTimeMicros": 4000,
    "documentVersion": 0,
    "documentUpdateTimeMicros": 0,
    "documentExpirationTimeMicros": 0,
    "documentOwner": "5d4eec49-1478-4807-a874-7a7ff885ce71"
}
```

Configure a LAG

To configure LAG, refer to *Configure a LAG between one Slingshot switch and one GigE switch* topic.

Slingshot documentation

The Slingshot documentation package includes PDF documents and is provided with HPE Cray EX 1.4 documentation.

- The *HPE Cray EX System Installation and Configuration Guide (1.4) S-8000* includes Slingshot software installation procedures for HPE Cray EX systems.
- The Slingshot documentation package includes all other documentation for Slingshot administration and troubleshooting.

24.1 Configure a LAG Between One Slingshot Switch and GigE Switch

Prerequisites

The example topology in this example shows various concepts and configuration methods for configuring a link aggregation group (LAG). The example uses one Slingshot switch (06) that has the following configuration:

- The Slingshot LAG number is 1
- The Slingshot HSN cable connector ports connected to LAG 1 are x1000c0r0j23p0 and x1000c0r0j17p0
- The Juniper Switch chassis MAC address for the Slingshot side LAG is c0:03:80:e8:4f:7e

About this task

This procedure configures a functional LAG between a Juniper QFX10002-60c switch and a single Slingshot switch. The most common use case for configuring a LAG for a Slingshot network is to connect Slingshot to a storage system. Link Aggregation Control Protocol (LACP) is not supported in this release.

Figure 8. Juniper Switch LAG Example

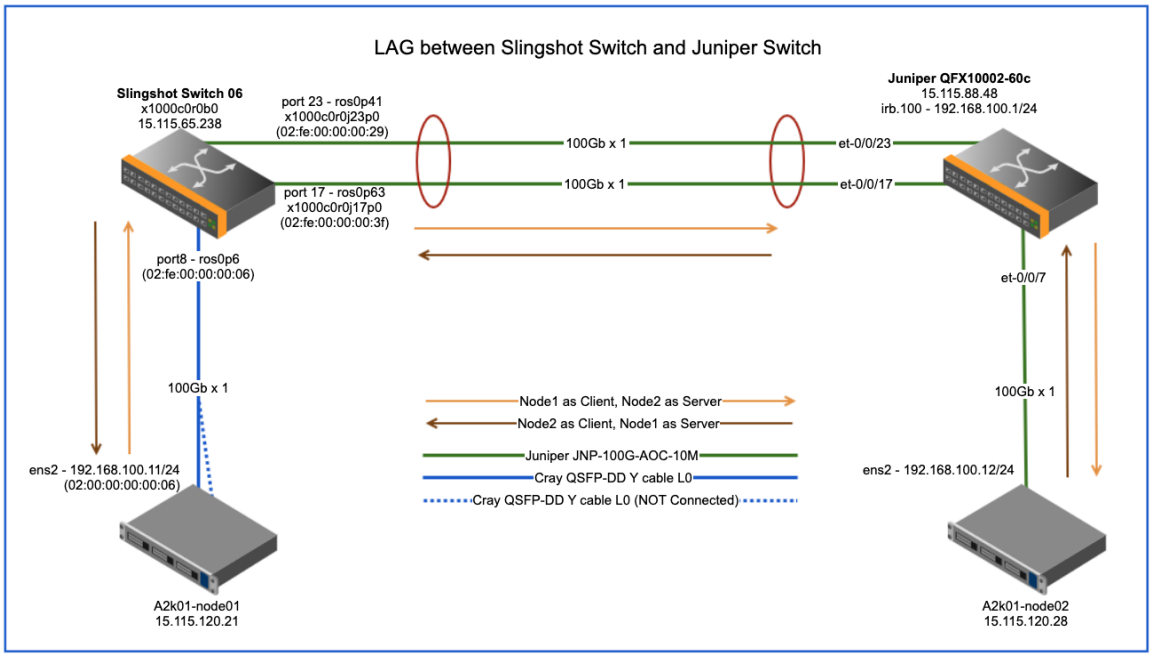


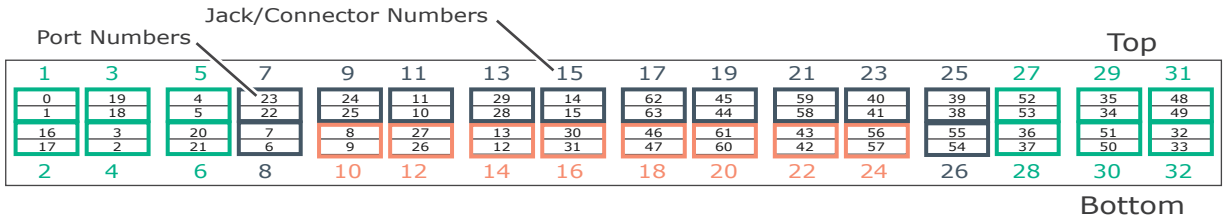
Figure 9. Juniper Switch Device and IP Information

Device	Hostname	Interface	IP
Juniper QFX10002-60c	JuniperSwitchZoo	mgmt	15.115.88.48
Slingshot Switch	SlingshotSwitch06	mgmt	15.115.65.238
Compute Node	A2k01-node01	iLo eno1	15.115.109.41 15.115.120.21
Compute Node	A2k01-node02	iLo eno1	15.115.109.42 15.115.120.28

Figure 10. Juniper Switch Interface Table

Hostname	Local Interface	Neighbor Host	Neighbor Interface	If connecting to/from Slingshot Switch				Static IP on Local Interface
				Local MAC	Neighbor MAC	Expected to change Local MAC?		
A2k01-node01	ens2	SlingshotSwitch06	port8 → ros0p6	04:09:73:e3:1a:6c	02:fe:00:00:00:06	Yes	02:00:00:00:00:06	192.168.100.11/24
A2k01-node02	ens2	JuniperSwitchZoo	et-0/0/7	N/A	N/A	No	N/A	192.168.100.12/24
SlingshotSwitch06	port8 → ros0p6	A2k01-node01	ens2	02:fe:00:00:00:06	04:09:73:e3:1a:6c	No	N/A	N/A
	port17 → ros0p63	JuniperSwitchZoo	et-0/0/17	02:fe:00:00:00:3f	c0:03:80:e8:4f:7e	No	N/A	N/A
	port23 → ros0p41	JuniperSwitchZoo	et-0/0/23	02:fe:00:00:00:29	c0:03:80:e8:4f:7e	No	N/A	N/A
JuniperSwitchZoo	et-0/0/7	A2k01-node02	ens2	N/A	N/A	No	N/A	
	et-0/0/17	SlingshotSwitch06	port17 → ros0p63	c0:03:80:e8:4f:7e	02:fe:00:00:00:3f	No	N/A	irb.100
	et-0/0/23	SlingshotSwitch06	port23 → ros0p41	c0:03:80:e8:4f:7e	02:fe:00:00:00:29	No	N/A	(192.168.100.1/24)

Figure 11. Slingshot ToR Switch Connector and Port Numbers



Procedure

1. Disable auto-negotiation on Slingshot LAG Ports.

The LAG ports require auto-negotiation to be disabled. The following steps modify port policies associated with the LAG port objects.

a. Create a port policy file named `portPolicyDisableAutoneg.json` and add the following lines to disable auto-negotiation as shown.

```
{
  "state": "ONLINE",
  "autoneg": false,
  "speed": "BJ_100G",
  "precode": "AUTO",
  "flowControl": {
    "rx": true,
    "tx": true
  },
```

```
  "mac": "02:00:00:00:00:00",
  "loopback": "NONE",
  "documentSelfLink": "/fabric/port-policies/edge-policy-disable-autoneg"
}
```

b. From the Fabric Manager pod, create the port policy.

```
slingshot-fabric-manager:# curl -X POST http://127.0.0.1:8000/fabric/port-policies \
-d @portPolicyDisableAutoneg.json -H "Content-Type: application/json" | jq
```

c. Create an edge port policy file named `portPolicyDisableAutoneg.json`, and add the following lines.

```
{
  "portPolicyLinks": [
    "/fabric/port-policies/edge-policy-disable-autoneg"
  ]
}
```

d. Apply the disable-autonegotiation edge port policy to the LAG ports.

Specify the xnames for the port objects for LAG ports used on the Slingshot switch (in this example, they are ports x1000c0r0j23p0 and x1000c0r0j17p0).

```
slingshot-fabric-manager# curl -X PATCH http://127.0.0.1:8000/fabric/ports/x1000c0r0j23p0 \
-d @edgePortDisableAutoneg.json -H "Content-Type: application/json"
```

```
slingshot-fabric-manager:# curl -X PATCH http://127.0.0.1:8000/fabric/ports/x1000c0r0j17p0 \
-d @edgePortDisableAutoneg.json -H "Content-Type: application/json"
```

2. Create a LAG properties file named `lagProperites.json` as shown.

```
{ "lagPropertyMap": {
    "1": {
      "portLinks": [
        "/fabric/ports/x1000c0r0j23p0",
        "/fabric/ports/x1000c0r0j17p0"
       ],
       "dmacs" : [
         "c0:03:80:e8:4f:7e"
       ]
     }
   }
}
```

3. Apply the LAG properties.

```
slingshot-fabric-manager# curl -X PATCH http://127.0.0.1:8000/fabric/topology-policies/template-
policy  \
-d @lagProperties.json -H "Content-Type: application/json"
```

4. Check LAG port operational state.

The fabric manager handles disruptions in LAG ports and removes LAG ports from switch tables when the ports go offline and adds them when ports go online. The set of active LAG ports (lagPortsActive) can be queried in the routing-engine data.

```
slingshot-fabric-manager#  curl -X GET
http://127.0.0.1:8000/fabric/routing-engines/dragonfly/template-routing
```

In the following example below, one of the ports in the LAG is offline.

```
{ "lagProperties": { "1": { "dmacs":
[
"74:83:ef:2a:e7:83",
"94:83:ef:2a:e7:83"
],
, "portLinks":
```

```
[
"/fabric/ports/x0c0r0j10p0",
"/fabric/ports/x0c0r0j3p1",
"/fabric/ports/x0c0r0j13p1",
"/fabric/ports/x0c0r0j11p1"
]
} }, "lagPortsActive": { "1": { "portLinks":
[
"/fabric/ports/x0c0r0j10p0",
"/fabric/ports/x0c0r0j13p1",
"/fabric/ports/x0c0r0j11p1"
]
} } }
```

5. Verify the LAG configuration on Slingshot (through agent query).

Agent LAG/FIB data - the agent data can be queried by the following call:

```
slingshot-fabric-manager# curl -X GET http://127.0.0.1:8000/fabric/agents/x1000c0r0b0
```

The desired/actual LAG/FIB entries should be identical. Note component names (xnames) have been converted to Agent Port IDs in the following example which identify port by <group, switch, port>. Missing ports means that they are down.

```
{ "actualStatus": { "lags": { 1: { "ports": } }, "fibEntries": { "c0:03:80:e8:4f:7e": 1
} } }
```

6. Configure the LAG on the Juniper switch.

a. Remove DHCP.

The first step is to remove DHCP from the interfaces that are being used.

- `et-0/0/7` is connects a compute node as a host.
- `et-0/0/17` and `et-0/0/23` are the links that are going to be part of the LAG. Delete DHCP because it must be a static configuration.

b. Log in to the switch and delete the interfaces.

```
root@JuniperSwitchZoo> et-0/0/7 unit 0 family inet delete interfaces
root@JuniperSwitchZoo> et-0/0/17 unit 0 family inet delete interfaces
root@JuniperSwitchZoo> et-0/0/23 unit 0 family inet
```

c. Create the VLAN.

```
root@JuniperSwitchZoo> set vlans Test100 vlan-id 100
```

d. Configure interface et-0/0/7 for the compute node.

```
root@JuniperSwitchZoo> set interfaces et-0/0/7 mtu 9000
root@JuniperSwitchZoo> set interfaces et-0/0/7 unit 0 family ethernet-switching interface-mode
access
root@JuniperSwitchZoo> set interfaces et-0/0/7 unit 0 family ethernet-switching vlan members
Test100
```

e. Configure Integrated Routing and Bridging (IRB) interfaces.

```
root@JuniperSwitchZoo> set interfaces irb unit 100 family inet address 192.168.100.1/24
root@JuniperSwitchZoo> set vlans Test100 l3-interface irb.100
```

f. Configure the number of aggregated Ethernet interfaces with LAG interface for the configuration. Set the device-count option to 5.

```
root@JuniperSwitchZoo> set chassis aggregated-devices ethernet device-count 5
```

g. Add port(s) to the aggregated Ethernet interface with LAG.

```
root@JuniperSwitchZoo> set interfaces et-0/0/17 ether-options 802.3ad ae0
root@JuniperSwitchZoo> set interfaces et-0/0/23 ether-options 802.3ad ae0
```

h. Configure family Ethernet switching for the aggregated Ethernet interface with LAG.

```
root@JuniperSwitchZoo> set interfaces ae0 unit 0 family ethernet-switching interface-mode access
root@JuniperSwitchZoo> set interfaces ae0 unit 0 family ethernet-switching vlan members Test100
```

i. Configure flow control on LAG interfaces.

```
root@JuniperSwitchZoo> set interfaces et-0/0/17 gigether-options flow-control
root@JuniperSwitchZoo> set interfaces et-0/0/23 gigether-options flow-control
```

j. Configure MTU to 9000.

```
root@JuniperSwitchZoo> set interfaces ae0 mtu 9000
```

7. Check interface status.

```
root@JuniperSwitchZoo> show interfaces terse | grep et
pfh-0/0/0.16383         up    up   inet
pfh-0/0/0.16384         up    up   inet
et-0/0/7                up    up
et-0/0/7.0              up    up   eth-switch
et-0/0/17               up    up
et-0/0/17.0             up    up   aenet    --> ae0.0
et-0/0/23               up    up
et-0/0/23.0             up    up   aenet    --> ae0.0
ae0.0                   up    up   eth-switch
bme0.0                  up    up   inet     128.0.0.1/2
em0.0                   up    up   inet     15.115.88.48/18
em1.32768               up    up   inet     192.168.1.2/24
em2.0                   up    down inet
irb.100                 up    up   inet     192.168.100.1/24
jsrv.1                  up    up   inet     128.0.0.127/2
lo0.0                   up    up   inet
                                   inet6    fe80::c203:800f:fce8:43c0
lo0.16385               up    up   inet
```

8. Verify connectivity over LAG, ping node1 over LAG at 192.168.100.11

```
root@JuniperSwitchZoo> ping 192.168.100.11 count 5
PING 192.168.100.11 (192.168.100.11): 56 data bytes
64 bytes from 192.168.100.11: icmp_seq=0 ttl=64 time=1.288 ms
64 bytes from 192.168.100.11: icmp_seq=1 ttl=64 time=1.332 ms
64 bytes from 192.168.100.11: icmp_seq=2 ttl=64 time=1.266 ms
64 bytes from 192.168.100.11: icmp_seq=3 ttl=64 time=1.400 ms
64 bytes from 192.168.100.11: icmp_seq=4 ttl=64 time=1.341 ms

--- 192.168.100.11 ping statistics ---
5 packets transmitted, 5 packets received, +4 duplicates, 0% packet loss
round-trip min/avg/max/stddev = 1.266/1.337/1.431/0.050 ms
```

9. Verify aggregate Ethernet interface counters.

```
root@JuniperSwitchZoo> show interfaces ae0.0
  Logical interface ae0.0 (Index 79) (SNMP ifIndex 635)
    Flags: Up SNMP-Traps 0x24024000 Encapsulation: Ethernet-Bridge
    Statistics        Packets        pps         Bytes          bps
    Bundle:
        Input :     206438132          1 1508037204186          512
        Output:     212706309          0 1814088860845            0
    Adaptive Statistics:
        Adaptive Adjusts:          0
        Adaptive Scans  :          0
        Adaptive Updates:          0
    Protocol eth-switch, MTU: 9000
```

25 1.4.1 Patch Upgrade

The following procedures detail how to upgrade the HPE Cray EX system with 1.4.1 Patch Upgrade content. These procedures can only be run on systems with existing 1.4 software.

25.1 Merge Tar Archives for CSM and Analytics Patch Deployment

Analytics 1.0.12 and CSM 0.9.1-RC1 have been split into smaller 7.8Gb in some environments. These files must be recombined into the full tarball.

1. Run the following command to view the analytics tar archives.

```
ncn-m001# ls -lh | grep analytics
-rw-r--r--  1 root root 7.8G Apr  2 13:44 analytics-1.0.12.tar.gz-00
-rw-r--r--  1 root root 7.8G Apr  2 14:02 analytics-1.0.12.tar.gz-01
-rw-r--r--  1 root root 3.0G Apr  2 13:46 analytics-1.0.12.tar.gz-02
-rw-r--r--  1 root root   58 Apr  2 13:38 analytics-1.0.12.tar.gz.md5sum
-rw-r--r--  1 root root  120 Apr  2 13:46 analytics-1.0.12.tar.gz.sha256
-rw-r--r--  1 root root  384 Apr  2 13:54 analytics-1.0.12.tar.gz.sha256.sig
```

2. Merge the files.

```
ncn-m001# cat analytics-1.0.12.tar.gz-0* > analytics-1.0.12.tar.gz
```

3. Verify the content is correct.

```
ncn-m001# cat analytics-1.0.12.tar.gz.md5sum
8c99296c8dc174ee8afece12c21f75f6 analytics-1.0.12.tar.gz

ncn-m001# md5sum analytics-1.0.12.tar.gz
8c99296c8dc174ee8afece12c21f75f6 analytics-1.0.12.tar.gz
```

4. Run the following command to view the CSM tar archives.

```
ncn-m001# ls | grep csm-0.9.1-rc.1.tar.gz
csm-0.9.1-rc.1.tar.gz-00
csm-0.9.1-rc.1.tar.gz-01
csm-0.9.1-rc.1.tar.gz-02
csm-0.9.1-rc.1.tar.gz-03
csm-0.9.1-rc.1.tar.gz.md5sum
csm-0.9.1-rc.1.tar.gz.sha256
csm-0.9.1-rc.1.tar.gz.sha256.sig
```

5. Merge the files.

```
ncn-m001# cat csm-0.9.1-rc.1.tar.gz-0* > csm-0.9.1-rc.1.tar.gz
```

6. Verify the content is correct.

```
ncn-m001# md5sum csm-0.9.1-rc.1.tar.gz
f2240f4f81a94a346a1514de06c0be24  csm-0.9.1-rc.1.tar.gz

ncn-m001# cat csm-0.9.1-rc.1.tar.gz.md5sum
f2240f4f81a94a346a1514de06c0be24  csm-0.9.1-rc.1.tar.gz
```

7. Repeat this process with CSM, using the `csm-0.9.2.tar.gz-0*` files.

25.2 Upgrade CSM

Prerequisites

An existing system running 1.4 software

About this task

The following procedure details how to upgrade the CSM product stream for 1.4.1 software upgrades.

Procedure

1. Start a typescript to capture the commands and output from this upgrade.

```
ncn-m001# script -af product-csm-upgrade.$(date +%Y-%m-%d).txt
ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Extract CSM tarball.
3. Locate the README.md file in `docs/upgrade/0.9/README.md`.
4. Execute the instructions within the README.md file.
5. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

25.3 Install the System Admin Toolkit Product Stream Update

Prerequisites

- CSM is installed and verified.
- cray-product-catalog is running.

About this task

ROLE

- System administrator

OBJECTIVE Update the SAT (System Admin Toolkit) product stream.

Procedure

1. Start a typescript to capture the commands and output from this installation update.

   ```
   ncn-m001# script -af product-sat.$(date +%Y-%m-%d).txt
   ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
   ```

2. Copy the release distribution gzipped tar file, e.g. `sat-2.0.4.tar.gz`, to **`ncn-m001`**.

3. Unzip and extract the release distribution.

   ```
   ncn-m001# tar -xvzf sat-2.0.4.tar.gz
   ```

4. Change directory to the extracted release distribution directory.

   ```
   ncn-m001# cd sat-2.0.4
   ```

5. Run the installer, **install.sh**.

   ```
   ncn-m001# ./install.sh
   ```

6. Export the environment variable **`SAT_TAG`** to the value 3.5.0 to ensure that the **`sat`** and **`sat-man`** commands use the latest version of the **`cray/cray-sat`** container image.

   ```
   ncn-m001# export SAT_TAG=3.5.0
   ```

   This should be added to the `~/.bashrc` or `~/.bash_profile` file of each master node where **`sat`** commands are run.

   ```
   ncn-m001# pdsh -w ncn-m00[1-3] cat ~/.bashrc
   ncn-m001: source <(kubectl completion bash)
   ncn-m003: source <(kubectl completion bash)
   ncn-m002: source <(kubectl completion bash)
   ncn-m001 # pdsh -w ncn-m00[1-3] 'echo "export SAT_TAG=3.5.0" >> ~/.bashrc'
   ncn-m001 # pdsh -w ncn-m00[1-3] cat ~/.bashrc
   ncn-m001: source <(kubectl completion bash)
   ncn-m001: export SAT_TAG=3.5.0
   ncn-m003: source <(kubectl completion bash)
   ncn-m003: export SAT_TAG=3.5.0
   ncn-m002: source <(kubectl completion bash)
   ncn-m002: export SAT_TAG=3.5.0
   ```

   NOTE: If any of the master nodes reboot and perform a PXE boot, this change will be undone and will need to be reapplied on that node.

7. Verify that SAT is successfully installed by running the following command to confirm the expected version.

   ```
   ncn-m001# sat --version
   sat 3.5.0
   ```

   NOTE: The first time a **`sat`** command is run with this new version set on a master NCN, it displays output indicating the container image is downloading.

```
ncn-m001# sat --version
Trying to pull registry.local/cray/cray-sat:3.5.0...
Getting image source signatures
Copying blob 62694d7552cc skipped: already exists
Copying blob df20fa9351a1 skipped: already exists
Copying blob 3ab6766f6281 skipped: already exists
Copying blob e85fc7b6b56d done
Copying blob a77f87103e92 done
Copying blob ec64268fe629 done
Copying blob 07e514e84853 done
Copying blob 815a8c980572 done
Copying blob b663030eddc2 done
Copying blob 454709382513 done
Copying blob 0784bc5c53fd done
Copying blob 9271bf9dcea7 done
Copying blob 8eecb6a00f44 done
Copying blob ba5f1233ceee done
Copying blob 0297d65b5e58 done
Copying blob e1770111f399 done
Copying blob ba5f1233ceee done
Copying config 9355e546f4 done
Writing manifest to image destination
Storing signatures
sat 3.5.0
```

8. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

> **NOTE:** If ONLY a SAT product stream update is occuring, no other steps in the *HPE Cray EX System Installation and Configuration Guide S-8000* need to be executed to install the 2.0.4 version of the SAT product stream in the v1.4.1 patch. Specifically, it is NOT necessary for the admin to configure SAT authentication, s3 credentials, and run **`sat setrev`** again. If they are configured again, it will cause no harm.
>
> If CSM is also being updated, it is likely that the admin should also configure SAT authentication, s3 credentials, and run **`sat setrev`**.

==OPTIONAL==

The old version of the **`cray/cray-sat`** container image will still be present in the registry on the system, and if the environment variable **`SAT_TAG`** is not set to 3.5.0, it will default to using the old version. If desired, the admin may remove the older version of the **`cray/cray-sat`** container image in the nexus registry. It is **not** removed by default.

The cray-product-catalog will also show both versions of **`sat`** that are installed. This is viewed with the command **`sat showrev --products`** as shown in the following example.

```
ncn-m001# sat showrev --products
################################################################################
Product Revision Information
################################################################################
+--------------+-----------------+--------------------+-----------------------+
| product_name | product_version | images             | image_recipes         |
+--------------+-----------------+--------------------+-----------------------+
...
| sat          | 2.0.3           | -                  | -                     |
| sat          | 2.0.4           | -                  | -                     |
```

```
...
+--------------+-----------------+--------------------+-----------------------+
```

25.4 Upgrade System Monitoring Application (SMA)

Prerequisites

- CSM is installed and verified.
- gitea, cray-cfs-api, cray-cfs-operator, and cray-ims are running.

About this task

ROLE	• System administrator
OBJECTIVE	Install the System Monitoring Application (SMA) product stream which now has its own independent release distribution and installer.
LIMITATIONS	None
NEW IN THIS RELEASE	This entire procedure is new with this release.

Procedure

1. Start a typescript to capture the commands and output from this installation.

   ```
   ncn-m001# script -af product-sma.$(date +%Y-%m-%d).txt
   ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
   ```

2. Copy the release distribution gzipped tar file, e.g. `sma-1.4.19.tar.gz`, to **`ncn-m001`**.

3. Extract the SMA release distribution.

   ```
   ncn-m001# tar xvfz sma-1.4.19.tar.gz
   ```

4. Change directories to the extracted directory.

   ```
   ncn-m001# cd sma-1.4.19
   ```

5. Run the **`./configure_sma_pvc.sh script`** as shown in the following example to configure PVCs that are appropriately sized for the available storage. The script will generate and update **`customizations.yaml`** with the appropriate PVC sizes for Kafka, Postgres, and ElasticSearch.

   ```
   ncn-m001# ./configure_sma_pvc.sh
   report 4050606990
   Fri Feb 19 22:06:20 UTC 2021 - Raw storage = 46089092726784 bytes
   Fri Feb 19 22:06:20 UTC 2021 - SMF raw storage = 39492022960128 bytes
   Fri Feb 19 22:06:20 UTC 2021 - SMF quota bytes = 19746011480064 bytes
   Fri Feb 19 22:06:20 UTC 2021 - SMF quota GiB = 18389GiB
   Fri Feb 19 22:06:20 UTC 2021 - Configure ceph SMF pool quota to 18389GiB
   Fri Feb 19 22:06:20 UTC 2021 - Total available for kafka, postgres, and
   elasticsearch = 18279GiB
   ```

```
Fri Feb 19 22:06:20 UTC 2021 - Backup ./build/customizations.yaml to ./build/
customizations.yaml.bak
Fri Feb 19 22:06:20 UTC 2021 - Update ./build/customizations.yaml with new
values
Fri Feb 19 22:06:20 UTC 2021 - Changing Kafka PVC size from 65Gi to 1218GiB
Fri Feb 19 22:06:20 UTC 2021 - Changing Postgres PVC size from 1.6Ti to 3655GiB
Fri Feb 19 22:06:20 UTC 2021 - Changing ElasticSearch PVC size from 188Gi to
2611GiB
```

> **NOTE:** If the capacity has increased in the Ceph-based utility storage after the initial 1.4.0 installation, then the Kafka, Postgres, and ElasticSearch PVCs and SMF quota will increase in size.

The actual PVC sizes on the system may slightly differ from the defined values in **`customizations.yaml`**. If the latter is less than the former, the Helm chart will fail to deploy. Perform the following steps to verify (and update, if needed) the sizes defined in **`customizations.yaml`**.

6. Verify the Kafka PVC size.

   Query the actual PVC size and compare it to the value from the output in Step 5.

   ```
   ncn-m001# kubectl -n sma get pvc data-cluster-kafka-0 \
   --output jsonpath={.spec.resources.requests.storage}

   1218Gi
   ```

   Output from Step 5:

   ```
   Fri Feb 19 22:06:20 UTC 2021 - Changing Kafka PVC size from 65Gi to <NEW-
   KAFKA_SIZE>
   ```

   If <NEW_KAFKA_SIZE> from the Step 5 output is less than the actual size, replace the value for **`spec.kubernetes.services.sma-zk-kafka.kafkaPVCSize`** in **`./build/customizations.yaml`** with the actual PVC size.

7. Verify Postgres PVC size.

   Query the actual PVC size and compare it to the value from the output in Step 5.

   ```
   ncn-m001# kubectl -n sma get pvc pgdata-sma-postgres-cluster-0 \
   --output jsonpath={.spec.resources.requests.storage}

   3655Gi
   ```

   Output from Step 5:

   ```
   Fri Feb 19 22:06:20 UTC 2021 - Changing Postgres PVC size from 1.6Ti to
   <NEW_PG_SIZE>
   ```

   If <NEW_PG_SIZE> from the Step 5 output is less than the actual size, replace the value for **`spec.kubernetes.services.sma-postgres-cluster.volume.size`** in **`./build/customizations.yaml`** with the actual PVC size.

8. Verify ElasticSearch PVC size.

   Query the actual PVC size and compare it to the value from the output in Step 5.

   ```
   ncn-m001# kubectl -n sma get pvc elasticsearch-master-elasticsearch-master-0 \
   --output jsonpath={.spec.resources.requests.storage}

   2611Gi
   ```

Output from Step 5:

```
Fri Feb 19 22:06:20 UTC 2021 - Changing ElasticSearch PVC size from 188Gi to
<NEW_ES_SIZE>
```

If <NEW_ES_SIZE> from the Step 5 output is less than the actual size, replace the value for **`spec.kubernetes.services.sma-elasticsearch.volumeClaimTemplate.resources.requests.storage`** in **`./build/customizations.yaml`** with the actual PVC size.

9. Run the **`./install.sh`** script.

```
ncn-m001# ./install.sh
```

10. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

25.4.1 Manually Delete the Redfish Event Alarm Definition

Prerequisites

An upgrade from HPE Cray EX 1.4.0 to HPE Cray EX 1.4.x software has occurred and includes the SMA product stream.

About this task

ROLE

- System installer or System administrator

OBJECTIVE Manually delete the Redfish Event Alarm Definition. The total number of alarms count will grow excessively.

LIMITATIONS Manually deleting the Redfish Event Alarm Definition is required for an upgrade from HPE Cray EX 1.4.0 to HPE Cray EX 1.4.x. For a base install of **only** 1.4.1, the *Redfish Event Alarm* will not exist, so it will not be listed after running the commands in Step 2 to obtain the alarm definition ID (the monasca alarm-definition-list) and Step 3 to delete the alarm definition (the alarm-definition-delete).

Procedure

1. Get the **`sma-monasca-agent`** name.

```
ncn-m001# kubectl get pods -A |grep sma-monasca-agent
sma    sma-monasca-agent-0          2/2     Running         0       95m
sma    sma-monasca-agent-wm94f      2/2     Running         0       104m
sma    sma-monasca-agent-wzndq      2/2     Running         0       104m
sma    sma-monasca-agent-zq9pz      2/2     Running         0       104m
```

2. Get the alarm definition ID.

```
ncn-m001# kubectl -n sma exec -it sma-monasca-agent-wm94f -c collector -- sh \
-c 'monasca alarm-definition-list' |grep 'Redfish Event Alarm'
```

```
| Redfish Event Alarm | 0cab2858-7316-4ba2-8564-8c3159bd9e08 |
last(dmtf.redfish_event) > 0 | context   | True   |
```

3. Delete the alarm definition.

```
ncn-m001# kubectl -n sma exec -it sma-monasca-agent-wm94f -c collector -- sh \
-c 'monasca alarm-definition-delete 0cab2858-7316-4ba2-8564-8c3159bd9e08'
Successfully deleted alarm definition
```

4. Verify alarm deletion.

```
ncn-m001# kubectl -n sma exec -it sma-monasca-agent-wm94f -c collector -- sh \
-c 'monasca alarm-definition-list' |grep 'Redfish Event Alarm'
ncn-m001#
```

25.5 Upgrade and Configure the Cray Operating System (COS)

Prerequisites

- An existing EX system with 1.4 software installed.
- The Cray command line interface (CLI) tool is initialized and configured on the system. See "Configure the Cray Command Line Interface (CLI)" in the HPE Cray EX System Administration Guide S-8001 for more information.
- Cray system management has been installed and verified, including the following Helm Charts:
  - Gitea
  - cray-product-catalog
  - cray-cfs-api
  - cray-cfs-operator
  - cray-ims

    NOTE: The following command will list the helm charts of the required microservices:

```
ncn-m001# helm ls -n services | grep -E 'gitea|cray-cfs-operator|cray-cfs-api|cray-ims|cray-product-catalog'
cray-cfs-api            services    1    2020-10-19 15:00:12.189790185 -0500 CDT deployed    cray-cfs-
api-1.5.0-20201006133025+58367e7            0.14.1-20201006133025_58367e7
cray-cfs-operator       services    1    2020-10-19 15:00:10.411523529 -0500 CDT deployed    cray-cfs-
operator-1.8.0-20201006133513+dab1dab       1.8.0-20201006133513_dab1dab
cray-product-catalog    services    1    2020-10-22 14:10:11.321431248 -0500 CDT deployed    cray-product-
catalog-0.0.1-20201028184555+30b896c     0.0.1-20201028184555_30b896c
cray-ims                services    2    2020-10-30 16:21:25.560614562 -0500 CDT deployed    cray-
ims-2.4.1-20201030143835+0cb54eb                2.3.12-20201030143835_0cb54eb
gitea                   services    1    2020-10-23 14:25:55.544840236 -0500 CDT deployed
gitea-1.8.0-20201023142353+f21d3f0
```

About this task

The procedure for upgrading the HPE Cray Operating System (COS) from a Shasta v1.4 release to a Shasta v1.4.1 release is similar to the process used to install COS in Shasta v1.4. The install.sh script is used to install the new COS content onto the system and upgrade the COS microservices. The administrator must then perform operational tasks to build, customize, and boot a COS compute node image with the updated COS content. While the remainder of this section documents the upgrade procedure in detail, the following key points are of interest before an administrator begins the upgrade:

- COS product version 2.0.27 was delivered with Shasta v1.4.
- COS product version 2.0.30 is delivered with Shasta v1.4.1.
- Multiple versions of COS can be installed on a system at the same time. The administrator can decide which version to use (or use both).
    - The Content Projection Service (CPS) and Node Memory Dump (NMD) microservices are an exception. When a new version of a microservice is provided in a new release of COS, the microservice will automatically upgrade to the new version as part of the COS install process. This is done in a rolling fashion. Administrators and users should not notice a loss in capability while the microservice is updated. In the COS 2.0.30 release, both CPS, and NMD will be upgraded.
    - Any COS content installed on the NCN host operating system is the other exception. If an upgrade provides new COS software for the NCN host operating system, that software will be used when the NCN is booted to its new image. A new NCN image is not provided in the Shasta v1.4.1 release however.
- The administrator can configure the system to boot either version of COS on the compute nodes, or boot both versions on subsets of compute nodes at the same time.
- New COS configuration content will be uploaded to VCS with a new version number (2.0.30) to distinguish it from the COS configuration content previously delivered with Shasta v1.4 (2.0.27).
- A new COS image recipe will be uploaded to IMS with a new version number (2.0.30) to distinguish it from the COS image recipe previously delivered with Shasta v1.4 (2.0.27).
- The new version of COS (2.0.30) and its artifacts will be displayed in the product catalog alongside the existing version of COS (2.0.27) and its artifacts.
- The COS RPMs provided by the Shasta v1.4.1 release will be stored in new Nexus raw repositories cos-2.0.30-sle-15sp1-compute and cos-2.0.30-sle-15sp2. The existing Nexus group repositories, cos-2.0-sle-15sp1-compute, and cos-2.0-sle-15sp2, will be reconfigured to reference the newly installed COS Nexus raw repositories.
    - Prior to the upgrade, the Nexus group repositories referenced the cos-2.0.27-sle-15sp1-compute and cos-2.0.27-sle-15sp2 raw repositories.
    - When a COS image is built from the COS image recipe, it references the Nexus group repository cos-2.0-sle-15sp1-compute to determine which RPM content to include in the image. If the administrator wants to build a compute image for an older version of COS installed on the system (2.0.27), they must modify the COS 2.0.27 image recipe to reference the cos-2.0.27-sle-15sp1-compute Nexus raw repository, as the Nexus group repository now references the cos-2.0.30-sle-15sp1-compute.

Procedure

1. Start a typescript to capture the commands and output from this installation.

    ```
    ncn-m001# script -af product-cos.$(date +%Y-%m-%d).txt
    ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
    ```

2. Copy the release distribution compressed tar file to ncn-m001.

3. Unzip and extract the release distribution.

    ```
    ncn-m001# tar xzvf cos-2.0.30.tar.gz
    ```

4. Change to the extracted release distribution directory.

```
ncn-m001# cd cos-2.0.30
```

5. Run the `install.sh` script to begin installing COS.

```
ncn-m001# ./install.sh
```

6. Query Nexus repo and verify that repos were created for COS.

```
ncn-m001# curl -s -k https://packages.local/service/rest/v1/repositories \
| jq -r '.[] | select(.name | startswith("cos")) | .name'
...
cos-2.0.30-sle-15sp1-compute
cos-2.0.30-sle-15sp2
cos-2.0-sle-15sp1-compute
cos-2.0-sle-15sp2
...
```

There should be two items for 2.0.30 added to the Nexus repo, one for the compute node SLE 15 SP1 OS and one for the Non-compute node SLE 15 SP2 OS.

7. Verify COS configuration and image recipes components.

Check that COS pods have completed without error before checking the cray-product-catalog for the presence of the COS recipe.

```
ncn-m001# kubectl get jobs -A | egrep "NAME|cos"
NAMESPACE     NAME                                                 COMPLETIONS
DURATION   AGE
services      cos-config-import-2.0.30                             1/1
94s        17m
services      cos-image-sp1-recipe-import-2.0.30                   1/1
2m35s      12m
ncn-m001# kubectl get pods -A | egrep "NAME|cos"
NAMESPACE        NAME
READY   STATUS             RESTARTS   AGE
services         cos-config-import-2.0.30-dqx2j
0/3     Completed          0          17m
services         cos-image-sp1-recipe-import-2.0.30-pdq7t
0/3     Completed          0          12m
ncn-m001# kubectl get cm cray-product-catalog -n services -o json | jq -r .data.cos

2.0.30:
  configuration:
    clone_url: https://vcs.sif.dev.cray.com/vcs/cray/cos-config-management.git
    commit: 0c70558938105b7a2841ae458c9184fe05ca7d0a
    import_branch: cray/cos/2.0.30
    import_date: 2021-01-29 23:38:10.971626
    ssh_url: git@vcs.sif.dev.cray.com:cray/cos-config-management.git
  images:
    cray-shasta-compute-sles15sp1.x86_64-1.4.56:
      id: 88c5c6ed-a873-4328-b37d-9983b2e20e27
  recipes:
    cray-shasta-compute-sles15sp1.x86_64-1.4.56:
      id: f5e0ea59-5687-476c-95c5-85d41c93749c

ncn-m001:~ # export IMS_RECIPE_ID=f5e0ea59-5687-476c-95c5-85d41c93749c

ncn-m001:~ # cray ims recipes describe $IMS_RECIPE_ID --format json
{
  "created": "2021-01-29T23:38:56.014283+00:00",
  "id": "f5e0ea59-5687-476c-95c5-85d41c93749c",
  "link": {
    "etag": "fba10f7df4591d9a4f4ad92e7b5ceb40",
    "path": "s3://ims/recipes/f5e0ea59-5687-476c-95c5-85d41c93749c/recipe.tar.gz",
```

```
    "type": "s3"
  },
  "linux_distribution": "sles15",
  "name": "cray-shasta-compute-sles15sp1.x86_64-1.4.56",
  "recipe_type": "kiwi-ng"
}
```

8. Confirm that the DVS servers have loaded the DVS and LNet kernel modules.

```
ncn-m001# pdsh -w ncn-w[001-NNN] lsmod | grep -P '\bdvs\b' | dshbak -c
 ----------------
   ncn-w[001-NNN]
   ----------------
   dvs 425984 0
   dvsipc 188416 2 dvs
   dvsproc 151552 3 dvsipc,dvsipc_lnet,dvs
   craytrace 20480 5 dvsipc,dvsproc,dvsipc_lnet,dvs
   dvskatlas 24576 4 dvsipc,dvsproc,dvsipc_lnet,dvs

   If the modules are not listed for each worker node, refer to the
   HPE Cray EX v1.4 Installation Guide "Enable NCN Personalization" section.
```

9. Create a branch using the imported branch from the installation to customize COS.

The imported branch will be reported in the cray-product-catalog and can be used as a base branch. The imported branch from the installation should not be modified. It is recommended that a branch is created from the imported branch.

The following steps will create an integration branch.

a. Replace the clone_url hostname with api-gw-service-nmn.local.

```
ncn-m001# kubectl get cm cray-product-catalog -n services -o yaml \
| yq r - 'data.cos' | yq r - '"2.0.30"'
2.0.30:
  configuration:
    clone_url: https://vcs.rocket.dev.cray.com/vcs/cray/cos-config-management.git
    commit: 215eab2c316fb75662ace6aaade8b8c2ab2d08ee
    import_branch: cray/cos/2.0.30    <==  IMPORT_BRANCH
    import_date: 2021-02-21 23:01:16.100251
    ssh_url: git@vcs.rocket.dev.cray.com:cray/cos-config-management.git
  images:
    cray-shasta-compute-sles15sp1.x86_64-1.4.64:
      id: c8013386-6fe4-49f3-92ca-96d4f3be29a7
  recipes:
    cray-shasta-compute-sles15sp1.x86_64-1.4.64:
      id: 5149788d-4e5d-493d-b259-f56156a58b0d
```

b. Store the import branch for later use.

```
ncn-m001# export IMPORT_BRANCH=cray/cos/2.0.30
```

c. Obtain the credentials for the crayvcs user with the Kubernetes secret.

```
ncn-m001# kubectl get secret -n services vcs-user-credentials --
template={{.data.vcs_password}} | base64 --decode
<== password output ==>

git clone now requires username and password to clone a vcs branch
```

```
ncn-m001# git clone https://api-gw-service-nmn.local/vcs/cray/cos-config-
management.git
Username for 'https://api-gw-service-nmn.local': crayvcs
```

```
Password for 'https://crayvcs@api-gw-service-nmn.local': <<-- password from
preceeding output
```

```
ncn-m001# cd cos-config-management/
ncn-m001# git checkout $IMPORT_BRANCH && git pull
ncn-m001# git checkout -b integration && git merge $IMPORT_BRANCH
```

If the branch chosen to merge currently exists, run the following command.

```
ncn-m001# git checkout integration && git merge $IMPORT_BRANCH
```

10. Apply any customizations and modifications to the Ansible configuration, if required. COS configuration can be done later.

11. Push any changes for the branch to the repository using the proper credentials.

a. Obtain the credentials for the crayvcs user with the Kubernetes secret.

```
ncn-m001# - kubectl get secret -n services vcs-user-credentials --
template={{.data.vcs_password}} | base64 --decode
```

```
ncn-m001# git push --set-upstream origin integration
Username for 'https://api-gw-service-nmn.local': crayvcs
Password for 'https://crayvcs@api-gw-service-nmn.local': <<-- password from
preceeding output
```

b. Identify the commit hash for this branch.

This will be used later when creating the CFS configuration layer.

```
ncn-m001# git rev-parse --verify HEAD
<== commit hash output ==>
```

c. Store the commit hash for later use.

```
ncn-m001# export COS_CONFIG_COMMIT_HASH=<commit hash output>
```

12. Create a Configuration Framework Service (CFS) session configuration for COS.

a. Create a JSON file as input to the CFS configurations CLI command.

Do not attempt to cut and paste the following example. This will lead to an incorrectly formatted json syntax.

```
ncn-m001# cat cos-config-2.0.30.json
{
  "layers": [
    {
      "name": "cos-integration-2.0.30",
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
      "playbook": "site.yml",
      "commit": "<COS_CONFIG_COMMIT_HASH>"
    }
  ]
}
```

If other products have been installed on the system, add additional configuration layers to be applied during the CFS session. This configuration can be used for preboot image customization, as well as post-boot node personalization. The clone URL, commit, and top-level play should be run for all configuration layers.

b. Update the configuration using the new JSON file.

```
ncn-m001# cray cfs configurations update cos-config-2.0.30 --file ./cos-config-2.0.30.json \
--format json
{
  "lastUpdated": "2020-11-05T17:05:58Z",
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
      "commit": "5cd59022d5e4d953a4124cff73064a810cb7ed4f",
      "name": "cos-integration-2.0.30",
      "playbook": "site.yml"
    }
  ],
  "name": "cos-config-2.0.30"
}
```

c. Build COS compute image from image recipe.

Refer to *Troubleshoot COS Image Building Failure* on page 182 before building the COS compute image. The following step will fail unless the workaround is performed.

If the system already has an IMS public key, use that for the build. Check by running 'cray ims public-keys list'.

```
Create and export an SSH Key for IMS
====================================

ncn-m001 # cray ims public-keys create --name "my public key" --public-key \
~/.ssh/id_rsa.pub

[[results]]
created = "2020-11-18T17:53:28.163021+00:00"
id = "4a629135-9ab6-4164-9fa2-86bd018a4c61"
name = "my public key"
...

NOTE: If you already have an IMS public key, use that for the build.
      Check by running `cray ims public-keys list`

ncn-m001 # export IMS_PUBLIC_KEY_ID=4a629135-9ab6-4164-9fa2-86bd018a4c61
```

Verify IMS_RECIPE_ID has expected value.

```
ncn-m001 # echo $IMS_RECIPE_ID (set in step 3)
```

Create an IMS image from the recipe

```
ncn-m001 # cray ims jobs create --job-type create \
--image-root-archive-name sles15sp1-recipe-example-image \
--artifact-id $IMS_RECIPE_ID \
--public-key-id $IMS_PUBLIC_KEY_ID \
--enable-debug False


Note results of command, should have action "creating"


        - initrd_file_name = "initrd"
          status = "creating"
          artifact_id = "a89a1013-f1dd-4ef9-b0f8-ed2b1311c98a"
          enable_debug = false
          kubernetes_configmap = "cray-ims-52e2d6f6-977f-4b91-
b0fb-174dc7e195e2-configmap"
          kubernetes_service = "cray-ims-52e2d6f6-977f-4b91-
```

```
b0fb-174dc7e195e2-service"
          kubernetes_job = "cray-ims-52e2d6f6-977f-4b91-b0fb-174dc7e195e2-
create"
          job_type = "create"
          id = "52e2d6f6-977f-4b91-b0fb-174dc7e195e2"
          created = "2020-11-06T20:40:51.823126+00:00"
          kubernetes_namespace = "ims"
          public_key_id = "4a629135-9ab6-4164-9fa2-86bd018a4c61"
          kernel_file_name = "vmlinuz"
          build_env_size = 10
          image_root_archive_name = "skern-sles15sp1-image"


      - Store the job id and kubernetes_job values from the output the IMS
job create


        ...
        kubernetes_job = "cray-ims-52e2d6f6-977f-4b91-b0fb-174dc7e195e2-
create"
        job_type = "create"
        id = "52e2d6f6-977f-4b91-b0fb-174dc7e195e2"
        ...

ncn-m001# export IMS_JOB_ID=52e2d6f6-977f-4b91-b0fb-174dc7e195e2
ncn-m001# export IMS_KUBERNETES_JOB=cray-ims-52e2d6f6-977f-4b91-
b0fb-174dc7e195e2-create

      - Use kubectl to describe the image create job. We want to identify
the POD doing the IMS image customization build.

ncn-m001# kubectl -n ims describe job $IMS_KUBERNETES_JOB
           ...
           Events:
             Type     Reason                Age    From            Message
             ----     ------                ----   ----            -------
             Normal   SuccessfulCreate      9m41s  job-controller  Created
pod: cray-ims-52e2d6f6-977f-4b91-b0fb-174dc7e195e2-create-tnk92

ncn-m001# export POD=cray-ims-52e2d6f6-977f-4b91-b0fb-174dc7e195e2-create-
tnk92

      - Check the kubernetes job containers for progress. Each kubernetes
log check is a tail that will not end until the container has completed. We
want to pend on each container in order. Once they have all completed, we
will be able to obtain the customized image id we then use for CFS image
customizations.

        - kubectl -n ims logs -f $POD -c fetch-recipe
        - kubectl -n ims logs -f $POD -c wait-for-repos
        - kubectl -n ims logs -f $POD -c build-ca-rpm
        - kubectl -n ims logs -f $POD -c build-image
        - kubectl -n ims logs -f $POD -c buildenv-sidecar
```

Store the resultant ID listed in the buildenv-sidecar container's log. "cray ims jobs describe $IMS_JOB_ID" can be used to obtain the resultant_image_id.

```
ncn-m001# export RESULTANT_IMAGE_ID=e3ba09d7-e3c2-4b80-9d86-0ee2c48c2214
```

13. Create a CFS session to do preboot image customization using the CFS session configuration that was created earlier.

    a. Generate the password HASH for the root user. Replace 'PASSWORD' with a chosen root password.

    ```
    ncn-m001# openssl passwd -6 -salt $(< /dev/urandom tr -dc _A-Z-a-z-0-9 |
    head -c4) PASSWORD
    ```

    b. Obtain the HashiCorp Vault root token.

    ```
    ncn-m001# kubectl get secrets -n vault cray-vault-unseal-keys -o
    jsonpath='{.data.vault-root}' | base64 -d; echo
    ```

    c. Write the password HASH from 13a. The 'vault login' command will request a token. The token value is the output of 13b. The 'vault read secret/cos' is used to verify that the HASH was stored correctly. This password HASH will be written to the applicable nodes for the root user by CFS.

    It is important to enclose the HASH in single quotes to preserve any special characters.

    ```
    ncn-m001# kubectl exec -itn vault cray-vault-0 -- sh
    export VAULT_ADDR=http://cray-vault:8200
    vault login
    vault write secret/cos root_password='HASH'
    vault read secret/cos
    ```

    d. Run a CFS image customization session.

    ```
    ncn-m001# cray cfs sessions create --name cos-config-2.0.30 \
              --configuration-name cos-config-2.0.30 --target-definition image \
              --target-group Compute ${RESULTANT_IMAGE_ID} --format json
    ```

14. Poll the CFS sessions job for completion.

    ```
    ncn-m001# cray cfs sessions describe cos-config-2.0.30 --format json \
    | jq -r .status.session.status

    ncn-m001# cray cfs sessions describe cos-config-2.0.30 --format json \
    | jq -r .status.session.succeeded
    ```

15. Obtain image result_id.

    ```
    ncn-m001# cray cfs sessions describe cos-config-2.0.30 --format json \
    | jq -r .status.artifacts[].result_id
    5f3fb8a7-4189-4d04-b0c3-eec98fdc6440
    ```

16. Store the CFS result_id.

    ```
    ncn-m001# export IMAGE_ID=5f3fb8a7-4189-4d04-b0c3-eec98fdc6440
    ```

17. Check the manifest.json artifact created by CFS, the md5sum value is needed for the BOS session template that will be created later in this procedure.

    ```
    ncn-m001# cray artifacts describe boot-images ${IMAGE_ID}/manifest.json --
    format json
    {
              "artifact": {
                "AcceptRanges": "bytes",
                "LastModified": "2021-03-30T01:20:33+00:00",
                "ContentLength": 1147,
    ```

```
            "ETag": "\"151cd5effbda573d8d05b3429c150e62\"",
            "ContentType": "binary/octet-stream",
            "Metadata": {
              "md5sum": "151cd5effbda573d8d05b3429c150e62"
            }
          }
        }
```

18. Retrieve and save the ETAG_VALUE.

```
ncn-m001# export ETAG_VALUE=151cd5effbda573d8d05b3429c150e62
```

19. Verify cray-conman is connected to the nodes being booted.

The cray-conman service determines which nodes it should monitor by checking with the Hardware State Manager (HSM) service. It does this *once* when it starts. If HSM has not discovered some nodes when conman starts, then HSM is unaware of them and so is conman. Therefore, it is important to verify that conman is monitoring all nodes console logs. If it is not, reinitialize the cray-conman service and recheck the nodes it is monitoring. `conman -q` can be used to list the existing connections.

Use kubectl to exec into the running cray-conman pod, then check the existing connections.

```
cray-conman-b69748645-qtfxj:/ # conman -q
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

If the compute nodes and UANs are not included in the list of nodes being monitored, the conman process can be reinitialized by killing the conman process.

```
cray-conman-b69748645-qtfxj:/ # ps -ax | grep conmand
     13 ?        Sl     0:45 conmand -F -v -c /etc/conman.conf
  56704 pts/3    S+     0:00 grep conmand
cray-conman-b69748645-qtfxj:/ # kill 13
```

This will regenerate the conman configuration file and restart the conman process, and now include all nodes that are included in the state manager.

```
cray-conman-b69748645-qtfxj:/ # conman -q
x9000c1s7b0n1
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

20. Construct a boot session template using the xnames of the compute nodes, the customized image ID, and the CFS session configuration name.

a. Create a boot session template using the `ETAG_VALUE` and `IMAGE_ID` identified in the previous step.

Manually insert `ETAG_VALUE` and `IMAGE_ID` into the file. Create the file manually. **Do not attempt to cut and paste the following example. This will lead to an incorrectly formatted json syntax.**

```
ncn-m001# vi cos-sessiontemplate-2.0.30.json
{
  "boot_sets": {
    "compute": {
      "boot_ordinal": 2,
      "kernel_parameters": "console=ttyS0,115200 bad_page=panic crashkernel=340M hugepagelist=2m-2g
intel_iommu=off intel_pstate=disable iommu=pt ip=dhcp numa_interleave_omit=headless numa_zonelist_order=node
oops=panic pageblock_order=14 pcie_ports=native printk.synchronous=y rd.neednet=1 rd.retry=10 rd.shell
turbo_boost_limit=999 spire_join_token=${SPIRE_JOIN_TOKEN}",
      "network": "nmn",
      "node_roles_groups": [
        "Compute"
      ],
      "path": "s3://boot-images/<IMAGE_ID>/manifest.json",,
      "rootfs_provider": "cpss3",
      "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:nmn0",
      "etag": "<ETAG_VALUE>",
      "type": "s3"
    }
  },
  "cfs": {
    "configuration": "cos-config-2.0.30"
  },
  "enable_cfs": true,
  "name": "cos-sessiontemplate-2.0.30"
}
```

b. Register the session template file with the Boot Orchestration Service (BOS).

   For more information about BOS templates, refer to *BOS Session Templates* on page 173.

   Any name or string may be used for the `sesstiontemplate --name` in the following example.

```
ncn-m001# cray bos v1 sessiontemplate create --file cos-sessiontemplate-2.0.30.json \
--name cos-sessiontemplate-2.0.30
/sessionTemplate/cos-sessiontemplate-2.0.30
```

21. Create a BOS Session to boot the compute nodes.

```
ncn-m001# cray bos v1 session create --template-uuid \
 cos-sessiontemplate-2.0.30 --operation reboot
```

The first attempt to reboot the compute nodes will most likely fail. The compute node boot may stop responding and the compute node console will look similar to the following:

```
2021-03-19 01:32:41 dracut-initqueue[420]: DVS: node map generated.
2021-03-19 01:32:41 katlas: init_module: katlas loaded, currently disabled
2021-03-19 01:32:41
2021-03-19 01:32:41 DVS: Revision: kbuild Built: Mar 17 2021 @ 15:14:05 against
LNet 2.12.4
2021-03-19 01:32:41 DVS debugfs: Revision: kbuild Built: Mar 17 2021 @ 15:14:05
against LNet 2.12.4
2021-03-19 01:32:41 dracut-initqueue[420]: DVS: loadDVS: successfully added 10
new nodes into map.
2021-03-19 01:32:41 ed dvsproc module.
2021-03-19 01:32:41 DVS: message size checks complete.
2021-03-19 01:32:41 dracut-initqueuedvs_thread_generator: Watching pool DVS-
IPC_msg (id 0)
2021-03-19 01:32:41 [420]: DVS: loaded dvs module.
2021-03-19 01:32:41 dracut-initqueue[420]: mount is: /opt/cray/cps-utils/bin/
cpsmount.sh -a api-gw-service-nmn.local -t dvs -T 300 -i nmn0 -e
3116cf653e84d265cf8da94956f34d9e-181 s3://boot-images/763213c7-3d5f-4f2f-9d8a-
ac6086583f43/rootfs /tmp/cps
2021-03-19 01:32:41 dracut-initqueue[420]: 2021/03/19 01:31:01 cpsmount_helper
Version: 1.0.0
2021-03-19 01:32:47 dracut-initqueue[420]: 2021/03/19 01:31:07 Adding content:
s3://boot-images/763213c7-3d5f-4f2f-9d8a-ac6086583f43/rootfs
3116cf653e84d265cf8da94956f34d9e-181 dvs
2021-03-19 01:33:02 dracut-initqueue[420]: 2021/03/19 01:31:22 WARN:
readyForMount=false type=dvs ready=0 total=2
```

```
2021-03-19 01:33:18 dracut-initqueue[420]: 2021/03/19 01:31:38 WARN:
readyForMount=false type=dvs ready=0 total=2
2021-03-19 01:33:28 dracut-initqueue[420]: 2021/03/19 01:31:48 2 dvs servers
[10.252.1.7 10.252.1.8]
```

If this occurs, repeat the BOS command.

For more information regarding the compute boot process, refer to *Compute Node Boot Sequence* on page 175 and *Boot Orchestration Service (BOS)* on page 175.

22. SSH to a newly booted compute node.

```
ncn-m001 # ssh nid001000-nmn
Last login: Wed Mar 17 19:10:12 2021 from 10.252.1.12
nid001000:~ #
```

23. Log out of the compute node.

```
nid000001:~ # exit
```

24. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

25.6 Apply UAN Upgrade Patch

Prerequisites

Version 2.0.0 of the UAN product must be installed.

About this task

ROLE	System administrator, system installer
OBJECTIVE	Updates the UAN product version from 2.0.0 to 2.0.1
NEW IN THIS RELEASE	This procedure is new in this release.

In this procedure:

- *UAN_RELEASE*: refers to the new UAN release version (for example, 2.0.1)
- *UAN_DISTDIR*: refers to the directory containing the extracted UAN release distribution for the new UAN release.
- *PRODUCT_VERSION*: refers to the full name of the UAN product version that is currently installed.

Procedure

1. Start a typescript to capture the commands and output from the installation.

```
ncn-m001# script -af product-uan.$(date +%Y-%m-%d).txt
ncn-m001#  export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Download the new UAN distribution tarball to the ncn-m001 node.

3. Select an existing directory, or create a new directory, to contain the extracted release distribution for the new UAN release.

   The remaining steps in this procedure will refer to this directory as *`UAN_DISTDIR`*.

4. Extract the contents of the tarball of the new UAN release to `UAN_DISTDIR`. Use the `--no-same-owner` and `--no-same-permissions` options of the `tar` command when extracting a UAN release distribution as the `root` user.

   These options ensure that the extracted files:

   - Are owned by the `root` user.
   - Have permissions based on the current `umask` value.

5. List the current UAN versions in the product catalog.

   This step verifies that version 2.0.0 of the UAN product is installed on the system.

   ```
   ncn-m001# kubectl -n services get cm cray-product-catalog -o
   jsonpath='{.data.uan}' \
   | yq r -j - | jq -r 'keys[]' | sed '/-/!{s/$/_/}' | sort -V | sed 's/_$//'
   ```

6. Deploy the manifests of new UAN version.

   The following command will install a new product branch in VCS. In the next steps, the updated content in this new release branch will be merged into the VCS branch currently being used to configure UANs. Refer to "VCS Branching Strategy" in the publication *HPE Cray EX System Administration Guide (S-8001)*.

   ```
   ncn-m001# UAN_DISTDIR/install.sh
   ```

7. Generate the password HASH for the root user. Replace `PASSWORD` with the root password chosen to use.

   ```
   ncn-m001# openssl passwd -6 -salt $(< /dev/urandom tr -dc _A-Z-a-z-0-9 | head -
   c4) PASSWORD
   ```

8. Obtain the HashiCorp Vault root token.

   ```
   ncn-m001# kubectl get secrets -n vault cray-vault-unseal-keys -o
   jsonpath='{.data.vault-root}' | base64 -d; echo
   ```

9. Write the password HASH from step 7 to the HashiCorp Vault. The `vault login` command will request a token. That token value is the output of step 8 above. The `vault read secret/uan` is to verify the HASH was stored correctly. This password HASH will be written to the UAN for the root user by CFS.

   It is important to enclose the HASH in single quotes to preserve any special characters.

   ```
   ncn-m001# kubectl exec -itn vault cray-vault-0 -- sh
   export VAULT_ADDR=http://cray-vault:8200
   vault login
   vault write secret/uan root_password='<HASH>'
   vault read secret/uan
   ```

10. Obtain the URL of UAN configuration management repository in VCS (the Gitea service).

This URL is reported as the value of the `configuration.clone_url` key in the `cray-product-catalog` Kubernetes ConfigMap.

11. Obtain the `crayvcs` password.

```
ncn-m001# kubectl get secret -n services vcs-user-credentials \
--template={{.data.vcs_password}} | base64 --decode
```

12. Clone the UAN configuration management repository. Replace the hostname reported in the URL obtained in the previous step with `api-gw-service-nmm.local` when cloning the repository.

```
ncn-m001# git clone https://api-gw-service-nmn.local/vcs/cray/uan-config-
management.git
. . .
ncn-m001# cd uan-config-management && git checkout cray/uan/PRODUCT_VERSION &&
git pull
Branch 'cray/uan/PRODUCT_VERSION' set up to track remote branch 'cray/uan/
PRODUCT_VERSION' from 'origin'.
Already up to date.
```

13. Checkout the branch currently used to hold UAN configuration.

The following example assumes that branch is `integration`.

```
ncn-m001# git checkout integration
Switched to branch 'integration'
Your branch is up to date with 'origin/integration'.
```

14. Merge the new install branch to the current branch. Write a commit message when prompted.

```
ncn-m001# git merge cray/uan/PRODUCT_VERSION
```

15. Push the changes to VCS. Enter the `crayvcs` password when prompted.

```
ncn-m001# git push
```

16. Retrieve the commit ID from the merge and store it for later use.

```
ncn-m001# git rev-parse --verify HEAD
```

17. Update any CFS configurations used by the UANs with the commit ID from the previous step.

a. Download the JSON of the current UAN CFS configuration to a file.

This file will be named `uan-config-2.0.1.json` since it will be modified and then used for the updated UAN version.

```
ncn-m001#  cray cfs configurations describe uan-config-2.0.0 \
--format=json &>uan-config-2.0.1.json
```

b. Remove the unneeded lines from the JSON file.

The lines to remove are in bold in the following example. These must be removed before uploading the modified JSON file back into CFS to update the UAN configuration.

```
ncn-m001# cat uan-config-2.0.1.json
{
  "lastUpdated": "2021-03-27T02:32:10Z",
```

```
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/uan-config-
management.git",
      "commit": "aa5ce7d5975950ec02493d59efb89f6fc69d67f1",
      "name": "uan-integration-2.0.0",
      "playbook": "site.yml"
    },
  "name": "uan-config-2.0.1-full"
}
```

c. Replace the `commit` value in the JSON file with the commit ID obtained in Step 12.

   The value to replace is in bold in the following example. The `name` value after the `commit` line may also be updated to match the new UAN product version, if desired. This is not necessary as CFS does not use this value for the configuration name.

```
{
 "layers": [
 {
 "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/uan-
configmanagement.git",
 "commit": "aa5ce7d5975950ec02493d59efb89f6fc69d67f1",
 "name": "uan-integration-2.0.0",
 "playbook": "site.yml"
 }
 ]
}
```

d. Create a new UAN CFS configuration with the updated JSON file.

   The following example uses `uan-config-2.0.1` for the name of the new CFS configuration, to match the JSON file name.

```
ncn-m001# cray cfs configurations update uan-config-2.0.1 \
--file uan-config-2.0.1.json
```

e. Tell CFS to apply the new configuration to UANs by repeating the following command for each UAN. Replace *`UAN_XNAME`* in the command below with the name of a different UAN each time the command is run.

```
ncn-m001# cray cfs components update --desired-config uan-config-2.0.1 \
--enabled true --format json UAN_XNAME
```

18. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

Perform *Create UAN Boot Images* on page 112

25.7 Upgrade Slurm

Prerequisites

- Loftsman

- Helm
- CSM
- COS
- UAN

About this task

ROLE	• System installer/system administrator
OBJECTIVE	The following procedure details how to upgrade Slurm on a 1.4 system.
NEW IN THIS RELEASE	This entire procedure is new with this release.

Procedure

1. Start a typescript to capture the commands and output from this upgrade.

```
ncn-m001# script -af product-slurm.$(date +%Y-%m-%d).txt
ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Copy Slurm release tarball onto the system.

3. Run installation script.

```
ncn-m001# tar -xf wlm-slurm-0.1.3.tar.gz
ncn-m001cd wlm-slurm-0.1.3
ncn-m001./install.sh
```

4. Customize Slurm ansible content.

```
ncn-m001# git clone https://api-gw-service-nmn.local/vcs/cray/slurm-config-
management.git
ncn-m001# cd slurm-config-management
ncn-m001# git merge origin/cray/slurm/0.1.3
```

   If desired, make additional changes and commits.

   a. Get crayvcs password for pushing.

```
ncn-m001# kubectl get secret -n services vcs-user-credentials --
template={{.data.vcs_password}} | base64 --decode
```

   b. Push changes to VCS (username crayvcs).

```
ncn-m001# git push origin master
```

   c. Obtain the commit hash to use for the CFS configurations.

```
ncn-m001# git rev-parse --verify HEAD
```

5. Update COS CFS configuration.

   The commit value for the COS layer needs to updated to the value obtained in step 5.

```
ncn-m001# cray cfs configurations describe cos-config-1.4.0 --format=json \
&>cos-config-x.x.x.json
```

Edit cos-config-x.x.x.json, adding a layer for Slurm.

```
ncn-m001# vi cos-config-x.x.x.json
ncn-m001# cat cos-config-x.x.x.json
{
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-
management.git",
      "commit": "5fb065ffc32414f02543654211486216f058986b",
      "name": "cos-integration-1.4.0",
      "playbook": "site.yml"
    },
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/slurm-config-
management.git",
      "commit": "b0cb51391940bb71844bb32cdcda7d89c2a1e80a",
      "name": "slurm-integration-0.1.3",
      "playbook": "site.yml"
    }
  ]
}
ncn-m001# cray cfs configurations update cos-config-1.4.0 \
--file cos-config-x.x.x.json
```

6. Create a new compute image.

```
ncn-m001# cray cfs sessions create \
    --name slurm-cos-0 \
    --configuration-name cos-config-1.4.0 \
    --target-definition image \
    --target-group Compute <IMAGE ID>
```

Obtain the customized image ID for the next step.

```
ncn-m001# IMS_ID=$(cray cfs sessions describe slurm-cos-0 --format json | jq -
r .status.artifacts[].result_id)
```

Obtain the etag and path for the BOS template.

```
ncn-m001# cray ims images describe $IMS_ID
```

7. Update COS BOS template with the new compute node image.

```
ncn-m001# cray bos v1 sessiontemplate describe cos-sessiontemplate-1.4.0 \
--format=json &>cos-sessiontemplate-1.4.0.json
```

Edit /upload/cos-sessiontemplate-1.4.0.json, replacing etag and path.

```
ncn-m001# cray bos v1 sessiontemplate create --name cos-sessiontemplate-1.4.0 \
--file cos-sessiontemplate-1.4.0.json
```

8. Verify cray-conman is connected to compute and UAN nodes.

The cray-conman service determines which nodes it should monitor by checking with the Hardware State Manager (HSM) service. It does this *once* when it starts. If HSM has not discovered some nodes when

conman starts, then HSM is unaware of them and so is conman. Therefore, it is important to verify that conman is monitoring all nodes' console logs. If it is not, re-initialize the cray-conman service and recheck the nodes it is monitoring. `conman -q` can be used to list the existing connections.

Use kubectl to exec into the running cray-conman pod, then check the existing connections.

```
cray-conman-b69748645-qtfxj:/ # conman -q
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

If the compute nodes and UANs are not included in the list of nodes being monitored, the conman process can be re-initialized by killing the conmand process.

```
cray-conman-b69748645-qtfxj:/ # ps -ax | grep conmand
  13 ?        Sl     0:45 conmand -F -v -c /etc/conman.conf
  56704 pts/3    S+     0:00 grep conmand
cray-conman-b69748645-qtfxj:/ # kill 13
```

This will regenerate the conman configuration file and restart the conmand process, and now include all nodes that are included in the state manager.

```
cray-conman-b69748645-qtfxj:/ #
x9000c1s7b0n1
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

9. Reboot compute nodes with updated session template.

```
ncn-m001# cray bos v1 session create --template-uuid cos-sessiontemplate-1.4.0
\
--operation reboot
```

10. Update UAN CFS configuration.

```
ncn-m001# cray cfs configurations describe uan-config-2.0.0 --format=json &>uan-
config-2.0.0.json
```

Edit /upload/uan-config-2.0.0.json, adding a layer for Slurm.

```
{
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/uan-config-
management.git",
      "commit": "aa5ce7d5975950ec02493d59efb89f6fc69d67f1",
      "name": "uan-integration-2.0.0",
      "playbook": "site.yml"
    },
```

```
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/slurm-config-
management.git",
      "commit": "b0cb51391940bb71844bb32cdcda7d89c2a1e80a",
      "name": "slurm-integration-0.1.0",
      "playbook": "site.yml"
    }
  ]
}
ncn-m001# cray cfs configurations update uan-config-2.0.0 \
--file uan-config-2.0.0.json
```

The name and lastUpdated fields should be removed from the cos.config-x.x.x.json file since it generates an error when left in.

```
Error: Bad Request: Property is read-only - 'name'
```

11. Create a new UAN image.

```
ncn-m001# cray cfs sessions create \
    --name slurm-uan-0 \
    --configuration-name uan-config-2.0.0 \
    --target-definition image \
    --target-group Application <IMAGE ID>
```

Get the customized image ID for the next step.

```
ncn-m001# cray cfs sessions describe slurm-uan-0 --format json | jq -
r .status.artifacts[].result_id
```

12. Update UAN BOS template.

```
ncn-m001# cray bos v1 sessiontemplate describe uan-sessiontemplate-2.0.0 --
format=json &> uan-sessiontemplate-2.0.0.json
```

Edit /upload/uan-sessiontemplate-2.0.0.json, replacing etag and path.

```
ncn-m001# cray bos v1 sessiontemplate create --name uan-sessiontemplate-2.0.0 --
file /upload/uan-sessiontemplate-2.0.0.json
```

13. Reboot UANs with updated session template.

```
ncn-m001# cray bos v1 session create --template-uuid uan-sessiontemplate-2.0.0
--operation reboot
```

14. Validate installation.

a. Validate Kubernetes deployments.

   Check for a running pod in each of the following deployments.

   ```
   ncn-m001# kubectl get deployment -n user slurmdb
   ncn-m001# kubectl get deployment -n user slurmdbd
   ncn-m001# kubectl get deployment -n user slurmctld
   ```

b. Validate compute node content.

   A private ssh key for root can be obtained with "kubectl get secrets -n services csm-private-key -o jsonpath="{.data.value}" | base64 -d

Check that `slurmd` is running.

```
ncn-m001# ssh nid000001
```

```
nid000001# systemctl status slurmd
```

```
nid00001# exit
```

c. Validate UAN content.

Check compute node Slurm state.

```
ncn-m001# ssh uan01
```

```
uan01# sinfo
```

Run *`hostname`* on every available compute node.

```
uan01# srun -N <num computes> hostname
```

```
uan01# exit
```

15. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

25.8 Install the Analytics Product Stream Update

Prerequisites

- Cray System Management (CSM) is installed and verified.
- UAN
- WLM
- PE
- gitea, cray-product-catalog, cray-cfs-api, cray-cfs-operator, and cray-ims are running.

About this task

ROLE	• System administrator
OBJECTIVE	Install the Analytics product stream update.
LIMITATIONS	None
NEW IN THIS RELEASE	This entire procedure is new with this release.

Procedure

1. Start a typescript to capture the commands and output from this installation.

```
ncn-m001# script -af product-analytics.$(date +%Y-%m-%d).txt
ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Copy the release distribution gzipped tar file, e.g. analytics-1.0.12.tar.gz, to ncn-m001.

3. Extract the Analytics release distribution.

```
ncn-m001# tar -zxvf analytics-1.0.12.tar.gz
```

4. Change directories to the extracted directory.

```
ncn-m001# cd analytics-1.0.12
```

5. Verify the prerequisites.

   Cray System Management (CSM) has been installed and verified including the following Helm Charts.

   - gitea
   - cray-product-catalog
   - cray-cfs-api
   - cray-cfs-operator
   - cray-ims
   - cray-uan
   - WLM (slurm or pbs)

   Verify the installation of these components with the following.

```
ncn-m001# helm ls -n services | grep -E \
'gitea|cray-cfs-operator|cray-cfs-api|cray-ims|cray-product-\catalog|uan|slurm|pbs'
```

   Cray PE

   - If Cray PE will be installed on the system, ensure Cray PE is installed and configured before installing Analytics.

6. Delete the existing Analytics Install chart if it exists.

```
ncn-m001# helm delete -n services cray-analytics-install || echo 'Ignore errors'
```

7. Run the following command.

```
ncn-m001# sed -i '/^AIupgrade/i set +o pipefail' ./install.sh
```

8. Run the installer, **`install.sh`**.

```
ncn-m001# ./install.sh
```

9. Run the post-install script, to configure Analytics on the system (approximately 5 to 10 minutes).

```
ncn-m001# ./analytics-post-install.sh
```

10. Once the NCN configuration has been applied to the NCNs, run the Analytics UAS configure script to publish the Analytics software into newly created UAIs. Note that the Compute, UAN, and NCN configurations can have the Analytics layer added at the bottom.

```
ncn-m001# ./uas_setup_analytics.sh
```

11. Exit the typescript file started at the beginning of this procedure. Analytics is installed and configured.

```
ncn-m001# exit
```

26 1.4.2 Patch Upgrade

The following procedures detail how to upgrade the HPE Cray EX system with 1.4.2 Patch Upgrade content. These procedures can only be run on systems with existing 1.4 software.

26.1 Merge Tar Archives for CSM, SUSE, and Analytics 1.4.2 Patch Deployment

Analytics 1.0.14, SUSE Updates 21.16.0, and CSM 0.9.3 have been split into smaller 7.8Gb in some environments. These files must be recombined into the full tarball.

1. Run the following command to view the analytics tar archives.

```
ncn-m001# ls -lh | grep analytics
-rw-r--r--  1 root root 7.8G Apr  2 13:44 analytics-1.0.14.tar.gz-00
-rw-r--r--  1 root root 7.8G Apr  2 14:02 analytics-1.0.14.tar.gz-01
-rw-r--r--  1 root root 3.0G Apr  2 13:46 analytics-1.0.14.tar.gz-02
-rw-r--r--  1 root root   58 Apr  2 13:38 analytics-1.0.14.tar.gz.md5sum
-rw-r--r--  1 root root  120 Apr  2 13:46 analytics-1.0.14.tar.gz.sha256
-rw-r--r--  1 root root  384 Apr  2 13:54 analytics-1.0.14.tar.gz.sha256.sig
```

2. Merge the files.

```
ncn-m001# cat analytics-1.0.12.tar.gz-0* > analytics-1.0.14.tar.gz
```

3. Verify the content is correct.

```
ncn-m001# cat analytics-1.0.14.tar.gz.md5sum
8c99296c8dc174ee8afece12c21f75f6 analytics-1.0.12.tar.gz

ncn-m001# md5sum analytics-1.0.14.tar.gz
8c99296c8dc174ee8afece12c21f75f6 analytics-1.0.12.tar.gz
```

4. Run the following command to view the CSM tar archives.

```
ncn-m001# ls | grep csm-0.9.3-rc.1.tar.gz
csm-0.9.3-rc.1.tar.gz-00
csm-0.9.3-rc.1.tar.gz-01
csm-0.9.3-rc.1.tar.gz-02
csm-0.9.3-rc.1.tar.gz-03
csm-0.9.3-rc.1.tar.gz.md5sum
csm-0.9.3-rc.1.tar.gz.sha256
csm-0.9.3-rc.1.tar.gz.sha256.sig
```

5. Merge the files.

```
ncn-m001# cat csm-0.9.3-rc.1.tar.gz-0* > csm-0.9.3-rc.1.tar.gz
```

6. Verify the content is correct.

```
ncn-m001# md5sum csm-0.9.3-rc.1.tar.gz
f2240f4f81a94a346a1514de06c0be24  csm-0.9.3-rc.1.tar.gz

ncn-m001# cat csm-0.9.3-rc.1.tar.gz.md5sum
f2240f4f81a94a346a1514de06c0be24  csm-0.9.3-rc.1.tar.gz
```

7. View the SUSE Updates tar archieves.

```
ncn-m001# ls | grep SUSE-Updates-x86_64-21.16.0.tar.gz
SUSE-Updates-x86_64-21.16.0.tar.gz
SUSE-Updates-x86_64-21.16.0.tar.gz-00
SUSE-Updates-x86_64-21.16.0.tar.gz-01
SUSE-Updates-x86_64-21.16.0.tar.gz-02
SUSE-Updates-x86_64-21.16.0.tar.gz-03
SUSE-Updates-x86_64-21.16.0.tar.gz-04
SUSE-Updates-x86_64-21.16.0.tar.gz-05
SUSE-Updates-x86_64-21.16.0.tar.gz-06
SUSE-Updates-x86_64-21.16.0.tar.gz.md5sum
```

8. Merge the files.

```
ncn-m001# cat SUSE-Updates-x86_64-21.16.0.tar.gz-0* > SUSE-Updates-
x86_64-21.16.0.tar.gz
```

9. Verify content is correct.

```
ncn-m001# md5sum SUSE-Updates-x86_64-21.16.0.tar.gz
ad832b8e04743b5d81da717335a49ee1 SUSE-Updates-x86_64-21.16.0.tar.gz

ncn-m001# cat SUSE-Updates-x86_64-21.16.0.tar.gz**.md5sum
ad832b8e04743b5d81da717335a49ee1 SUSE-Updates-x86_64-21.16.0.tar.gz
```

26.2 Upgrade CSM 1.4.2

Prerequisites

An existing system running 1.4 software

About this task

The following procedure details how to upgrade the CSM product stream for 1.4.2 software upgrades.

Procedure

1. Start a typescript to capture the commands and output from this upgrade.

```
ncn-m001# script -af product-csm-upgrade.$(date +%Y-%m-%d).txt
ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
```

2. Extract CSM tarball.

3. Locate the README.md file in `docs/upgrade/0.9/README.md`.

4. Execute the instructions within the README.md file.
5. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

26.3 Upgrade System Dump Utility (SDU) Product Stream

Prerequisites

- CSM is installed and verified
- Cray-product-catalog is running

About this task

ROLE

- System installer/system administrator

LIMITATIONS

Piping SDU command output to the less or more commands causes formatting problems. This is a known issue when using podman exec with the `--tty` option. Refreshing the less or more will workaround this issue. Less and more have the following commands to refresh the screen using `CTRL-r`, `CTRL-l`, or `r`.

OBJECTIVE

The following procedure details how to install and upgrade the SDU product stream, which has its own independent release distribution and installer. The SDU release distribution is a gzipped tar file that includes the SDU, the installation script, and libraries required to install it.

System Dump Utility (SDU) - Is responsible for gathering all necessary information needed to debug a significant majority of problems. "Gathering" can be extended to imply:

- Collection of multiple data sources across either a stand-alone or distributed system in a scalable manner, with the ability to tailor collection targets and time bounds
- Step-wide analysis of collected data to make assertions about a running product or platform, such as indicating that specific hardware is suspect.
- Presentation of system collection data for operator or programmatic analysis, such as triage tools and customer support summary tools.
- Capable of both exporting a portable collection to a POSIX file system and uploading a collection from a customer system into the Metis call-home data lake via RDA

Remote Device Access (RDA) - HPE RDA (Remote Device Access) provides integrated remote connectivity for support automation, device telemetry and remote service delivery. RDA is integrated with the SDU product stream.

- HPE RDA Documentation: *https://midway.ext.hpe.com/home*

NEW IN THIS RELEASE

This entire procedure is new with this release.

Procedure

1. Start a typescript to capture the commands and output from this installation.

   SDU is intended to be started on only one non-compute management node (ncn-m00x). However, it is staged and ready to be started on all management nodes for fail over support. Typically, ncn-m001 is chosen as the node to run SDU and will be used in the following examples.

   ```
   ncn-m001# script -af product-sdu.$(date +%Y-%m-%d).txt
   ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
   ```

2. Copy the release distribution gzipped tar file (e.g cray-sdu-rda-1.0.11.tar.gz) to ncn-m001.

3. Unzip and extract the release distribution.

   ```
   ncn-m001 # tar xzvf cray-sdu-rda-1.0.11.tar.gz
   ```

4. Change to the extracted release distribution directory.

   ```
   ncn-m001 # cd cray-sdu-rda-1.0.11
   ```

5. Run the installer.

   ```
   ncn-m001 # ./install.sh
   ```

6. Add the `cray-sdu-rda` nexus repository to zypper.

   ```
   ncn-m001# zypper addrepo -Gf https://packages.local/repository/cray-sdu-rda
   cray-sdu-rda
   ```

   The following error message can be ignored, " Repository named '`cray-sdu-rda`' already exists. Please use another alias".

7. Refresh the local metadata cache.

   ```
   ncn-m001# zypper refresh -f cray-sdu-rda
   ```

8. Install the `cray-sdu-rda` RPM.

   ```
   ncn-m001# zypper install -y cray-sdu-rda
   ```

9. Enable the `cray-sdu-rda` systemd service.

   ```
   ncn-m001 # systemctl enable cray-sdu-rda
   ```

10. Restart the `cray-sdu-rda` systemd service.

    ```
    ncn-m001 # systemctl start cray-sdu-rda
    ```

11. Verify that SDU is ready for use.

    - Poll for ready.

      ```
      ncn-m001 # sdu is_service_ready
      ```

    - Wait (block) for ready.

```
ncn-m001 # sdu wait_for_service 30
```

Starting the `cray-sdu-rda` service is asynchronous (i.e. the command returns before the service is fully started). Downloading the podman image is part of starting the container and may take several minutes to come back as ready. The above commands help to handle this situation.

Alternatively, issue the following to block until the service/container is ready or fails. An optional timeout in seconds argument may be provided. The default timeout is 30 seconds.

```
ncn-m001 # sdu wait_for_service 30
```

The following error may occur:

```
Error: error inspecting object: no such container cray-sdu-rda
```

If the above error is encountered, re-run the command to wait for the container to complete building.

```
ncn-m001 # sdu wait_for_service 30
```

If the error message still occurs after 5 minutes, run the following command to gather additional information.

```
ncn-m001# journalctl -f -u cray-sdu-rda
```

12. Configure SDU with system specific information.

The sdu setup command will ask for information, such as system name, serial number, system type, system description, product number, company name, company site, and country code.

See "sdu setup --help" for all of the information which will be requested.

- Start an interactive installation session.

  ```
  ncn-m001 # sdu setup
  ```

- Alternatively, non-blocking setup is provided by command line arguments. In particular, the --batch option. Run the following for more information about options that can be passed with the --batch option. Refer to `sdu setup --help` at the command line for more info on what will need to be input.

  ```
  ncn-m001 # sdu bash
  ncn-m001-sdu: # sdu setup --help
  ```

13. Verify configuration file is good.

```
ncn-m001 # sdu --check_conf
[stdout] INFO Configuration file
"/etc/opt/cray/sdu/sdu.conf" and CLI Options Valid.
```

14. Verify that a bash session within the SDU container can be started.

Verify ncn-m001-sdu or equivalent command prompt prior to exiting the container.

```
ncn-m001 # sdu bash
```

Verify "ncn-m001-sdu" or equivalent comment prompt prior to exiting the container.

```
ncn-m001-sdu:# exit
```

15. Verify bash commands can be executed in the context of the SDU container.

```
ncn-m001 # sdu bash ls
```

16. Verify the SDU man page and setup help.

    a. Verify that the SDU man-page can be viewed from the NCN.

    ```
    ncn-m001 # man sdu
    ```

    b. Verify that the main SDU man-page and setup can be viewed within the container.

    ```
    ncn-m001 # sdu bash man sdu
    ncn-m001# sdu bash
    ncn-m001-sdu: # man sdu
    ncn-m001-sdu: # sdu setup --help
    ncn-m001-sdu: # exit
    ```

17. Finish the typescript file started at the beginning of this procedure.

    ```
    ncn-m001# exit
    ```

26.4 Install 1.4.2 OS Product Stream

Prerequisites

Task prerequisites.

- Nexus is installed and running
- SUSE-PTF-x86_64-21.16.0.tar.gz must be downloaded to [path to tarballs]/os/
- SUSE-Updates-x86_64-21.16.0.tar.gz must be downloaded to [path to tarballs]/os/

About this task

The following procedure details how to install SUSE operating system rpm repositories to Nexus on 1.4.2 systems. The following examples use ncn-m001 as a generic placeholder for the node that the following steps are to be run on. The SUSE tarballs will be untarred and the install.sh scripts run to get the contents uploaded to Nexus. Before beginning the procedure, ensure there is adequate disk space where tarballs are located. If required, the tar.gz files may be deleted after their extraction. About 120GB of disk space will be required to save all of the 1.4.2 tar.gz files and their extracted/uncompressed contents.

Procedure

1. Start a typescript to capture the commands and output from this installation.

   ```
   ncn-m001# script -af product-os.$(date +%Y-%m-%d).txt
   ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
   ```

2. Change directory to the location where the OS Distribution tarballs reside.

   ```
   ncn-m001# cd [path to tarballs]/os
   ```

3. List all files.

```
ncn-m001# ls -al
    total 57126716
    drwxr-xr-x 2 root root 86 Apr 27 18:41 .
    drwx------ 1 root root 4096 Apr 27 18:37 ..
    rw-rr- 1 root root 2815877912 Apr 19 19:54 SUSE-PTF-x86_64-21.16.0.tar.gz
    rw-rr- 1 root root 55681867955 Apr 19 19:56 SUSE-Updates-
x86_64-21.16.0.tar.gz
```

4. Unarchive and uncompress the file SUSE-PTF-x86_64-21.16.0.tar.gz.

```
ncn-m001# tar -zxf SUSE-PTF-x86_64-21.16.0.tar.gz
```

5. Change directory.

```
ncn-m001# cd SUSE-PTF-x86_64-21.16.0
```

6. Run the install script.

```
ncn-m001# ./install.sh
Install SUSE-PTF-x86_64-21.16.0
```

7. Change directory.

```
ncn-m001# cd [path to tarballs]/os
```

8. Unarchive and uncompress the file SUSE-PTF-x86_64-21.16.0.tar.gz.

```
ncn-m001# tar -zxf SUSE-Updates-x86_64-21.16.0.tar.gz
```

9. Change directory.

```
ncn-m001# cd SUSE-Updates-x86_64-21.16.0
```

10. Run the install script.

```
ncn-m001# ./install.sh
Install SUSE-Updates-x86_64-21.16.0
```

11. Change directory.

```
ncn-m001# cd [path to tarballs]/os
```

    There may also be a SUSE-isos-x86_64 tarball included in this directory, but it is not required to extract and install unless directed by HPE.

12. Exit the typescript file which was started at the beginning of the procedure.

```
ncn-m001# exit
```

26.5 Upgrade and Configure the Cray Operating System (COS) 1.4.2

Prerequisites

- An existing EX system with 1.4.1 software installed.
- The Cray command line interface (CLI) tool is initialized and configured on the system. See "Configure the Cray Command Line Interface (CLI)" in the HPE Cray EX System Administration Guide S-8001 for more information.
- Cray system management has been installed and verified, including the following Helm Charts:
  - Gitea
  - cray-product-catalog
  - cray-cfs-api
  - cray-cfs-operator
  - cray-ims

    NOTE: The following command will list the helm charts of the required microservices:

```
ncn-m001# helm ls -n services | grep -E 'gitea|cray-cfs-operator|cray-cfs-api|cray-ims|cray-product-catalog'
cray-cfs-api            services    1    2020-10-19 15:00:12.189790185 -0500 CDT deployed    cray-cfs-
api-1.5.0-20201006133025+58367e7            0.14.1-20201006133025_58367e7
cray-cfs-operator       services    1    2020-10-19 15:00:10.411523529 -0500 CDT deployed    cray-cfs-
operator-1.8.0-20201006133513+dab1dab       1.8.0-20201006133513_dab1dab
cray-product-catalog    services    1    2020-10-22 14:10:11.321431248 -0500 CDT deployed    cray-product-
catalog-0.0.1-20201028184555+30b896c    0.0.1-20201028184555_30b896c
cray-ims                services    2    2020-10-30 16:21:25.560614562 -0500 CDT deployed    cray-
ims-2.4.1-20201030143835+0cb54eb            2.3.12-20201030143835_0cb54eb
gitea                   services    1    2020-10-23 14:25:55.544840236 -0500 CDT deployed
gitea-1.8.0-20201023142353+f21d3f0
```

About this task

The procedure for upgrading the HPE Cray Operating System (COS) from a Shasta v1.4.1 release to a Shasta v1.4.2 release is similar to the process used to install COS in Shasta v1.4 and the process used to upgrade COS from a Shasta v1.4 release to a Shasta v1.4.1 release. The install.sh script is used to install the new COS content onto the system and upgrade the COS microservices. The administrator must then perform operational tasks to build, customize, and boot a COS compute node image with the updated COS content. While the remainder of this section documents the upgrade procedure in detail, the following key points are of interest before an administrator begins the upgrade:

- COS product version 2.0.27 was delivered with Shasta v1.4.
- COS product version 2.0.38 is delivered with Shasta v1.4.1.
- COS product version 2.0.38 is delivered with Shasta v1.4.2
- Multiple versions of COS can be installed on a system at the same time. The administrator can decide which version to use (or use both).
  - The Content Projection Service (CPS) and Node Memory Dump (NMD) microservices are an exception. When a new version of a microservice is provided in a new release of COS, the microservice will automatically upgrade to the new version as part of the COS install process. This is done in a rolling fashion. Administrators and users should not notice a loss in capability while the microservice is updated. In the COS 2.0.38 release, both CPS, and NMD will be upgraded.

  - Any COS content installed on the NCN host operating system is the other exception. If an upgrade provides new COS software for the NCN host operating system, that software will be used when the NCN is booted to its new image. A new NCN image is not provided in the Shasta v1.4.2 release however.
- The administrator can configure the system to boot either version of COS on the compute nodes, or boot both versions on subsets of compute nodes at the same time.
- New COS configuration content will be uploaded to VCS with a new version number (2.0.38) to distinguish it from the COS configuration content previously delivered with Shasta v1.4 (2.0.38).
- A new COS image recipe will be uploaded to IMS with a new version number (2.0.38) to distinguish it from the COS image recipe previously delivered with Shasta v1.4 (2.0.38).
- The new version of COS (2.0.38) and its artifacts will be displayed in the product catalog alongside the existing version of COS (2.0.38) and its artifacts.
- The COS RPMs provided by the Shasta v1.4.1 release will be stored in new Nexus raw repositories cos-2.0.38-sle-15sp1-compute and cos-2.0.38-sle-15sp2. The existing Nexus group repositories, cos-2.0-sle-15sp1-compute, and cos-2.0-sle-15sp2, will be reconfigured to reference the newly installed COS Nexus raw repositories.
  - Prior to the upgrade, the Nexus group repositories referenced the cos-2.0.38-sle-15sp1-compute and cos-2.0.38-sle-15sp2 raw repositories.
  - When a COS image is built from the COS image recipe, it references the Nexus group repository cos-2.0-sle-15sp1-compute to determine which RPM content to include in the image. If the administrator wants to build a compute image for an older version of COS installed on the system (2.0.38), they must modify the COS 2.0.38 image recipe to reference the cos-2.0.38-sle-15sp1-compute Nexus raw repository, as the Nexus group repository now references the cos-2.0.38-sle-15sp1-compute.

Procedure

1. Start a typescript to capture the commands and output from this installation.

   ```
   ncn-m001# script -af product-cos.$(date +%Y-%m-%d).txt
   ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
   ```

2. Copy the release distribution compressed tar file to ncn-m001.

3. Unzip and extract the release distribution.

   ```
   ncn-m001# tar xzvf cos-2.0.38.tar.gz
   ```

4. Change to the extracted release distribution directory.

   ```
   ncn-m001# cd cos-2.0.38
   ```

5. Run the `install.sh` script to begin installing COS.

   ```
   ncn-m001# ./install.sh
   ```

6. Query Nexus repo and verify that repos were created for COS.

   ```
   ncn-m001# curl -s -k https://packages.local/service/rest/v1/repositories \
   | jq -r '.[] | select(.name | startswith("cos")) | .name'
   ...
   ```

```
cos-2.0.38-sle-15sp1-compute
cos-2.0.38-sle-15sp2
cos-2.0-sle-15sp1-compute
cos-2.0-sle-15sp2
...
```

There should be two items for 2.0.38 added to the Nexus repo, one for the compute node SLE 15 SP1 OS and one for the Non-compute node SLE 15 SP2 OS.

7. Verify COS configuration and image recipes components.

Check that COS pods have completed without error before checking the cray-product-catalog for the presence of the COS recipe.

```
ncn-m001# kubectl get jobs -A | egrep "NAME|cos"
NAMESPACE     NAME                                                 COMPLETIONS
DURATION   AGE
services      cos-config-import-2.0.38                             1/1
94s        17m
services      cos-image-sp1-recipe-import-2.0.38                   1/1
2m35s      12m
ncn-m001# kubectl get pods -A | egrep "NAME|cos"
NAMESPACE        NAME
READY   STATUS             RESTARTS   AGE
services         cos-config-import-2.0.38-dqx2j
0/3     Completed          0          17m
services         cos-image-sp1-recipe-import-2.0.38-pdq7t
0/3     Completed          0          12m
ncn-m001# kubectl get cm cray-product-catalog -n services -o json | jq -r .data.cos

2.0.38:
  configuration:
    clone_url: https://vcs.sif.dev.cray.com/vcs/cray/cos-config-management.git
    commit: 0c70558938105b7a2841ae458c9184fe05ca7d0a
    import_branch: cray/cos/2.0.38
    import_date: 2021-01-29 23:38:10.971626
    ssh_url: git@vcs.sif.dev.cray.com:cray/cos-config-management.git
  images:
    cray-shasta-compute-sles15sp1.x86_64-1.4.56:
      id: 88c5c6ed-a873-4328-b37d-9983b2e20e27
  recipes:
    cray-shasta-compute-sles15sp1.x86_64-1.4.56:
      id: f5e0ea59-5687-476c-95c5-85d41c93749c

ncn-m001:~ # export IMS_RECIPE_ID=f5e0ea59-5687-476c-95c5-85d41c93749c

ncn-m001:~ # cray ims recipes describe $IMS_RECIPE_ID --format json
{
  "created": "2021-01-29T23:38:56.014283+00:00",
  "id": "f5e0ea59-5687-476c-95c5-85d41c93749c",
  "link": {
    "etag": "fba10f7df4591d9a4f4ad92e7b5ceb40",
    "path": "s3://ims/recipes/f5e0ea59-5687-476c-95c5-85d41c93749c/recipe.tar.gz",
    "type": "s3"
  },
  "linux_distribution": "sles15",
  "name": "cray-shasta-compute-sles15sp1.x86_64-1.4.56",
  "recipe_type": "kiwi-ng"
}
```

8. Confirm that the DVS servers have loaded the DVS and LNet kernel modules.

```
ncn-m001# pdsh -w ncn-w[001-NNN] lsmod | grep -P '\bdvs\b' | dshbak -c
 ----------------
   ncn-w[001-NNN]
```

```
----------------
dvs 425984 0
dvsipc 188416 2 dvs
dvsproc 151552 3 dvsipc,dvsipc_lnet,dvs
craytrace 20480 5 dvsipc,dvsproc,dvsipc_lnet,dvs
dvskatlas 24576 4 dvsipc,dvsproc,dvsipc_lnet,dvs

If the modules are not listed for each worker node, refer to the
HPE Cray EX v1.4 Installation Guide "Enable NCN Personalization" section.
```

9. Create a branch using the imported branch from the installation to customize COS.

The imported branch will be reported in the cray-product-catalog and can be used as a base branch. The imported branch from the installation should not be modified. It is recommended that a branch is created from the imported branch.

The following steps will create an integration branch.

a. Replace the clone_url hostname with api-gw-service-nmn.local.

```
ncn-m001# kubectl get cm cray-product-catalog -n services -o yaml \
| yq r - 'data.cos' | yq r - '"2.0.38"'
2.0.38:
  configuration:
    clone_url: https://vcs.rocket.dev.cray.com/vcs/cray/cos-config-management.git
    commit: 215eab2c316fb75662ace6aaade8b8c2ab2d08ee
    import_branch: cray/cos/2.0.38    <==  IMPORT_BRANCH
    import_date: 2021-02-21 23:01:16.100251
    ssh_url: git@vcs.rocket.dev.cray.com:cray/cos-config-management.git
  images:
    cray-shasta-compute-sles15sp1.x86_64-1.4.64:
      id: c8013386-6fe4-49f3-92ca-96d4f3be29a7
  recipes:
    cray-shasta-compute-sles15sp1.x86_64-1.4.64:
      id: 5149788d-4e5d-493d-b259-f56156a58b0d
```

b. Store the import branch for later use.

```
ncn-m001# export IMPORT_BRANCH=cray/cos/2.0.38
```

c. Obtain the credentials for the crayvcs user with the Kubernetes secret.

```
ncn-m001# kubectl get secret -n services vcs-user-credentials --
template={{.data.vcs_password}} | base64 --decode
<== password output ==>

git clone now requires username and password to clone a vcs branch
```

```
ncn-m001# git clone https://api-gw-service-nmn.local/vcs/cray/cos-config-
management.git
Username for 'https://api-gw-service-nmn.local': crayvcs
Password for 'https://crayvcs@api-gw-service-nmn.local': <<-- password from
preceeding output
```

```
ncn-m001# cd cos-config-management/
ncn-m001# git checkout $IMPORT_BRANCH && git pull
ncn-m001# git checkout -b integration && git merge $IMPORT_BRANCH
```

If the branch chosen to merge currently exists, run the following command.

```
ncn-m001# git checkout integration && git merge $IMPORT_BRANCH
```

10. Apply any customizations and modifications to the Ansible configuration, if required. COS configuration can be done later.

11. Push any changes for the branch to the repository using the proper credentials.

a. Obtain the credentials for the crayvcs user with the Kubernetes secret.

```
ncn-m001# - kubectl get secret -n services vcs-user-credentials --
template={{.data.vcs_password}} | base64 --decode
```

```
ncn-m001# git push --set-upstream origin integration
Username for 'https://api-gw-service-nmn.local': crayvcs
Password for 'https://crayvcs@api-gw-service-nmn.local': <<-- password from
preceeding output
```

b. Identify the commit hash for this branch.

This will be used later when creating the CFS configuration layer.

```
ncn-m001# git rev-parse --verify HEAD
<== commit hash output ==>
```

c. Store the commit hash for later use.

```
ncn-m001# export COS_CONFIG_COMMIT_HASH=<commit hash output>
```

12. Create a Configuration Framework Service (CFS) session configuration for COS.

a. Create a JSON file as input to the CFS configurations CLI command.

Do not attempt to cut and paste the following example. This will lead to an incorrectly formatted json syntax.

```
ncn-m001# cat cos-config-2.0.38.json
{
  "layers": [
    {
      "name": "cos-integration-2.0.38",
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
      "playbook": "site.yml",
      "commit": "<COS_CONFIG_COMMIT_HASH>"
    }
  ]
}
```

If other products have been installed on the system, add additional configuration layers to be applied during the CFS session. This configuration can be used for preboot image customization, as well as post-boot node personalization. The clone URL, commit, and top-level play should be run for all configuration layers.

b. Update the configuration using the new JSON file.

```
ncn-m001# cray cfs configurations update cos-config-2.0.38 --file ./cos-config-2.0.38.json \
--format json
{
  "lastUpdated": "2020-11-05T17:05:58Z",
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
      "commit": "5cd59022d5e4d953a4124cff73064a810cb7ed4f",
      "name": "cos-integration-2.0.38",
      "playbook": "site.yml"
    }
  ],
```

```
  "name": "cos-config-2.0.38"
}
```

c. Build COS compute image from image recipe.

Refer to *Troubleshoot COS Image Building Failure* on page 182 before building the COS compute image. The following step will fail unless the workaround is performed.

If the system already has an IMS public key, use that for the build. Check by running 'cray ims public-keys list'.

```
Create and export an SSH Key for IMS
====================================

ncn-m001 # cray ims public-keys create --name "my public key" --public-key \
~/.ssh/id_rsa.pub

[[results]]
created = "2020-11-18T17:53:28.163021+00:00"
id = "4a629135-9ab6-4164-9fa2-86bd018a4c61"
name = "my public key"
...

NOTE: If you already have an IMS public key, use that for the build.
      Check by running `cray ims public-keys list`

ncn-m001 # export IMS_PUBLIC_KEY_ID=4a629135-9ab6-4164-9fa2-86bd018a4c61
```

Verify IMS_RECIPE_ID has expected value.

```
ncn-m001 # echo $IMS_RECIPE_ID (set in step 3)
```

Create an IMS image from the recipe

```
ncn-m001 # cray ims jobs create --job-type create \
--image-root-archive-name sles15sp1-recipe-example-image \
--artifact-id $IMS_RECIPE_ID \
--public-key-id $IMS_PUBLIC_KEY_ID \
--enable-debug False


Note results of command, should have action "creating"


        - initrd_file_name = "initrd"
          status = "creating"
          artifact_id = "a89a1013-f1dd-4ef9-b0f8-ed2b1311c98a"
          enable_debug = false
          kubernetes_configmap = "cray-ims-52e2d6f6-977f-4b91-
b0fb-174dc7e195e2-configmap"
          kubernetes_service = "cray-ims-52e2d6f6-977f-4b91-
b0fb-174dc7e195e2-service"
          kubernetes_job = "cray-ims-52e2d6f6-977f-4b91-b0fb-174dc7e195e2-
create"
          job_type = "create"
          id = "52e2d6f6-977f-4b91-b0fb-174dc7e195e2"
          created = "2020-11-06T20:40:51.823126+00:00"
          kubernetes_namespace = "ims"
          public_key_id = "4a629135-9ab6-4164-9fa2-86bd018a4c61"
          kernel_file_name = "vmlinuz"
```

```
          build_env_size = 10
          image_root_archive_name = "skern-sles15sp1-image"


      - Store the job id and kubernetes_job values from the output the IMS
job create



        ...
        kubernetes_job = "cray-ims-52e2d6f6-977f-4b91-b0fb-174dc7e195e2-
create"
        job_type = "create"
        id = "52e2d6f6-977f-4b91-b0fb-174dc7e195e2"
        ...

ncn-m001# export IMS_JOB_ID=52e2d6f6-977f-4b91-b0fb-174dc7e195e2
ncn-m001# export IMS_KUBERNETES_JOB=cray-ims-52e2d6f6-977f-4b91-
b0fb-174dc7e195e2-create

      - Use kubectl to describe the image create job. We want to identify
the POD doing the IMS image customization build.

ncn-m001# kubectl -n ims describe job $IMS_KUBERNETES_JOB
           ...
           Events:
             Type     Reason                Age    From            Message
             ----     ------                ----   ----            -------
             Normal   SuccessfulCreate      9m41s  job-controller  Created
pod: cray-ims-52e2d6f6-977f-4b91-b0fb-174dc7e195e2-create-tnk92

ncn-m001# export POD=cray-ims-52e2d6f6-977f-4b91-b0fb-174dc7e195e2-create-
tnk92

      - Check the kubernetes job containers for progress. Each kubernetes
log check is a tail that will not end until the container has completed. We
want to pend on each container in order. Once they have all completed, we
will be able to obtain the customized image id we then use for CFS image
customizations.

        - kubectl -n ims logs -f $POD -c fetch-recipe
        - kubectl -n ims logs -f $POD -c wait-for-repos
        - kubectl -n ims logs -f $POD -c build-ca-rpm
        - kubectl -n ims logs -f $POD -c build-image
        - kubectl -n ims logs -f $POD -c buildenv-sidecar
```

Store the resultant ID listed in the buildenv-sidecar container's log. "cray ims jobs describe $IMS_JOB_ID" can be used to obtain the resultant_image_id.

```
ncn-m001# export RESULTANT_IMAGE_ID=e3ba09d7-e3c2-4b80-9d86-0ee2c48c2214
```

13. Create a CFS session to do preboot image customization using the CFS session configuration that was created earlier.

a. Generate the password HASH for the root user. Replace 'PASSWORD' with a chosen root password.

```
ncn-m001# openssl passwd -6 -salt $(< /dev/urandom tr -dc _A-Z-a-z-0-9 |
head -c4) PASSWORD
```

b. Obtain the HashiCorp Vault root token.

```
ncn-m001# kubectl get secrets -n vault cray-vault-unseal-keys -o
jsonpath='{.data.vault-root}' | base64 -d; echo
```

c. Write the password HASH from 13a. The 'vault login' command will request a token. The token value is the output of 13b. The 'vault read secret/cos' is used to verify that the HASH was stored correctly. This password HASH will be written to the applicable nodes for the root user by CFS.

   It is important to enclose the HASH in single quotes to preserve any special characters.

```
ncn-m001# kubectl exec -itn vault cray-vault-0 -- sh
export VAULT_ADDR=http://cray-vault:8200
vault login
vault write secret/cos root_password='HASH'
vault read secret/cos
```

d. Run a CFS image customization session.

```
ncn-m001# cray cfs sessions create --name cos-config-2.0.38 \
          --configuration-name cos-config-2.0.38 --target-definition image \
          --target-group Compute ${RESULTANT_IMAGE_ID} --format json
```

14. Poll the CFS sessions job for completion.

```
ncn-m001# cray cfs sessions describe cos-config-2.0.38 --format json \
| jq -r .status.session.status

ncn-m001# cray cfs sessions describe cos-config-2.0.38 --format json \
| jq -r .status.session.succeeded
```

15. Obtain image result_id.

```
ncn-m001# cray cfs sessions describe cos-config-2.0.38 --format json \
| jq -r .status.artifacts[].result_id
5f3fb8a7-4189-4d04-b0c3-eec98fdc6440
```

16. Store the CFS result_id.

```
ncn-m001# export IMAGE_ID=5f3fb8a7-4189-4d04-b0c3-eec98fdc6440
```

17. Check the manifest.json artifact created by CFS, the md5sum value is needed for the BOS session template that will be created later in this procedure.

```
ncn-m001# cray artifacts describe boot-images ${IMAGE_ID}/manifest.json --
format json
{
          "artifact": {
            "AcceptRanges": "bytes",
            "LastModified": "2021-03-30T01:20:33+00:00",
            "ContentLength": 1147,
            "ETag": "\"151cd5effbda573d8d05b3429c150e62\"",
            "ContentType": "binary/octet-stream",
            "Metadata": {
              "md5sum": "151cd5effbda573d8d05b3429c150e62"
            }
          }
        }
```

18. Retrieve and save the ETAG_VALUE.

```
ncn-m001# export ETAG_VALUE=151cd5effbda573d8d05b3429c150e62
```

19. Verify cray-conman is connected to the nodes being booted.

The cray-conman service determines which nodes it should monitor by checking with the Hardware State Manager (HSM) service. It does this *once* when it starts. If HSM has not discovered some nodes when conman starts, then HSM is unaware of them and so is conman. Therefore, it is important to verify that conman is monitoring all nodes console logs. If it is not, reinitialize the cray-conman service and recheck the nodes it is monitoring. `conman -q` can be used to list the existing connections.

Use kubectl to exec into the running cray-conman pod, then check the existing connections.

```
cray-conman-b69748645-qtfxj:/ # conman -q
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

If the compute nodes and UANs are not included in the list of nodes being monitored, the conman process can be reinitialized by killing the conman process.

```
cray-conman-b69748645-qtfxj:/ # ps -ax | grep conmand
     13 ?        Sl     0:45 conmand -F -v -c /etc/conman.conf
  56704 pts/3    S+     0:00 grep conmand
cray-conman-b69748645-qtfxj:/ # kill 13
```

This will regenerate the conman configuration file and restart the conman process, and now include all nodes that are included in the state manager.

```
cray-conman-b69748645-qtfxj:/ # conman -q
x9000c1s7b0n1
x9000c0s1b0n0
x9000c0s20b0n0
x9000c0s22b0n0
x9000c0s24b0n0
x9000c0s27b1n0
x9000c0s27b2n0
x9000c0s27b3n0
```

20. Construct a boot session template using the xnames of the compute nodes, the customized image ID, and the CFS session configuration name.

a. Create a boot session template using the `ETAG_VALUE` and `IMAGE_ID` identified in the previous step.

Manually insert `ETAG_VALUE` and `IMAGE_ID` into the file. Create the file manually. **Do not attempt to cut and paste the following example. This will lead to an incorrectly formatted json syntax.**

```
ncn-m001# vi cos-sessiontemplate-2.0.38.json
{
  "boot_sets": {
    "compute": {
      "boot_ordinal": 2,
      "kernel_parameters": "console=ttyS0,115200 bad_page=panic crashkernel=340M hugepagelist=2m-2g
intel_iommu=off intel_pstate=disable iommu=pt ip=dhcp numa_interleave_omit=headless numa_zonelist_order=node
oops=panic pageblock_order=14 pcie_ports=native printk.synchronous=y rd.neednet=1 rd.retry=10 rd.shell
turbo_boost_limit=999 spire_join_token=${SPIRE_JOIN_TOKEN}",
      "network": "nmn",
      "node_roles_groups": [
        "Compute"
      ],
      "path": "s3://boot-images/<IMAGE_ID>/manifest.json",,
```

```
      "rootfs_provider": "cpss3",
      "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:nmn0",
      "etag": "<ETAG_VALUE>",
      "type": "s3"
    }
  },
  "cfs": {
    "configuration": "cos-config-2.0.38"
  },
  "enable_cfs": true,
  "name": "cos-sessiontemplate-2.0.38"
}
```

b. Register the session template file with the Boot Orchestration Service (BOS).

   For more information about BOS templates, refer to *BOS Session Templates* on page 173.

   Any name or string may be used for the `sesstiontemplate --name` in the following example.

```
ncn-m001# cray bos v1 sessiontemplate create --file cos-sessiontemplate-2.0.38.json \
--name cos-sessiontemplate-2.0.38
/sessionTemplate/cos-sessiontemplate-2.0.38
```

21. Create a BOS Session to boot the compute nodes.

```
ncn-m001# cray bos v1 session create --template-uuid \
 cos-sessiontemplate-2.0.38 --operation reboot
```

The first attempt to reboot the compute nodes will most likely fail. The compute node boot may stop responding and the compute node console will look similar to the following:

```
2021-03-19 01:32:41 dracut-initqueue[420]: DVS: node map generated.
2021-03-19 01:32:41 katlas: init_module: katlas loaded, currently disabled
2021-03-19 01:32:41
2021-03-19 01:32:41 DVS: Revision: kbuild Built: Mar 17 2021 @ 15:14:05 against
LNet 2.12.4
2021-03-19 01:32:41 DVS debugfs: Revision: kbuild Built: Mar 17 2021 @ 15:14:05
against LNet 2.12.4
2021-03-19 01:32:41 dracut-initqueue[420]: DVS: loadDVS: successfully added 10
new nodes into map.
2021-03-19 01:32:41 ed dvsproc module.
2021-03-19 01:32:41 DVS: message size checks complete.
2021-03-19 01:32:41 dracut-initqueuedvs_thread_generator: Watching pool DVS-
IPC_msg (id 0)
2021-03-19 01:32:41 [420]: DVS: loaded dvs module.
2021-03-19 01:32:41 dracut-initqueue[420]: mount is: /opt/cray/cps-utils/bin/
cpsmount.sh -a api-gw-service-nmn.local -t dvs -T 300 -i nmn0 -e
3116cf653e84d265cf8da94956f34d9e-181 s3://boot-images/763213c7-3d5f-4f2f-9d8a-
ac6086583f43/rootfs /tmp/cps
2021-03-19 01:32:41 dracut-initqueue[420]: 2021/03/19 01:31:01 cpsmount_helper
Version: 1.0.0
2021-03-19 01:32:47 dracut-initqueue[420]: 2021/03/19 01:31:07 Adding content:
s3://boot-images/763213c7-3d5f-4f2f-9d8a-ac6086583f43/rootfs
3116cf653e84d265cf8da94956f34d9e-181 dvs
2021-03-19 01:33:02 dracut-initqueue[420]: 2021/03/19 01:31:22 WARN:
readyForMount=false type=dvs ready=0 total=2
2021-03-19 01:33:18 dracut-initqueue[420]: 2021/03/19 01:31:38 WARN:
readyForMount=false type=dvs ready=0 total=2
2021-03-19 01:33:28 dracut-initqueue[420]: 2021/03/19 01:31:48 2 dvs servers
[10.252.1.7 10.252.1.8]
```

If this occurs, repeat the BOS command.

For more information regarding the compute boot process, refer to *Compute Node Boot Sequence* on page 175 and *Boot Orchestration Service (BOS)* on page 175.

22. SSH to a newly booted compute node.

```
ncn-m001 # ssh nid001000-nmn
Last login: Wed Mar 17 19:10:12 2021 from 10.252.1.12
nid001000:~ #
```

23. Log out of the compute node.

```
nid000001:~ # exit
```

24. Finish the typescript file started at the beginning of this procedure.

```
ncn-m001# exit
```

26.6 Apply 1.4.2 Patch for UAN

Prerequisites

UAN 2.0.0 is installed on the HPE Cray EX supercomputer system.

About this task

Follow this procedure to upgrade HPE Cray EX systems running UAN 2.0.0 to UAN 2.0.1.

The deployment of the new manifests will install a new product branch in VCS. These updates need to be merged into the VCS branch being used to configure UANs. Refer to the section "VCS Branching Strategy" in the publication *HPE Cray EX System Administration Guide (S-8001)*. This procedure uses a branch named `integration` as the current branch holding the existing UAN configuration data.

Procedure

1. Extract the UAN release distribution tarball with `tar` as the root user and with the `--no-same-owner` and `--no-same-permissions` options.

   These options ensure that the extracted files are owned by root and have permissions based on the current `umask` value.

   The remaining steps in this procedure will refer to the directory of the extracted UAN release distribution as *`UAN_DISTDIR`*.

2. List the current UAN versions in the product catalog.

```
ncn-m001# kubectl -n services get cm cray-product-catalog -o
jsonpath='{.data.uan}' | \
yq r -j - | jq -r 'keys[]' | sed '/-/!{s/$/_/}' | sort -V | sed 's/_$//'
```

3. Deploy the manifests.

```
ncn-m001# UAN_DISTDIR/install.sh
```

   Update VCS Content

4. Clone the UAN configuration management repository. Replace the hostname with **`api-gw-service-nmm.local`** when cloning the repository.

   The repository is located in the VCS/Gitea service and the location is reported in the `cray-product-catalog` Kubernetes ConfigMap in the `configuration.clone_url` key.

   ```
   ncn-m001# git clone https://api-gw-service-nmn.local/vcs/cray/uan-config-
   management.git
   . . .
   ncn-m001# cd uan-config-management && git checkout cray/uan/2.0.0 && git pull
   Branch 'cray/uan/2.0.0' set up to track remote branch 'cray/uan/2.0.0' from
   'origin'.
   Already up to date.
   ```

5. Checkout the branch currently used to hold UAN configuration.

   The following example assumes `integration` is the branch name.

   ```
   ncn-m001# git checkout integration
   Switched to branch 'integration'
   Your branch is up to date with 'origin/integration'.
   ```

6. Merge new installed branch to the current branch. Write a commit message when prompted.

   ```
   ncn-m001# git merge cray/uan/2.0.1
   ```

7. Obtain the password for the `crayvcs` user and push the changes to VCS.

   ```
   ncn-m001# kubectl get secret -n services vcs-user-credentials \
                 --template={{.data.vcs_password}} | base64 --decode
   ncn-m001# git push
   ```

8. Retrieve the commit ID from the merge.

   ```
   ncn-m001# git rev-parse --verify HEAD
   ```

9. Update any CFS configurations used by the UANs with the commit ID obtained in the previous step.

26.7 1.4.2 Install the Analytics Product Stream Update

Prerequisites

- Cray System Management (CSM) is installed and verified.
- UAN
- WLM
- PE
- gitea, cray-product-catalog, cray-cfs-api, cray-cfs-operator, and cray-ims are running.

About this task

ROLE	• System administrator
OBJECTIVE	Install the Analytics product stream update.
LIMITATIONS	None
NEW IN THIS RELEASE	This entire procedure is new with this release.

Procedure

1. Start a typescript to capture the commands and output from this installation.

   ```
   ncn-m001# script -af product-analytics.$(date +%Y-%m-%d).txt
   ncn-m001# export PS1='\u@\H \D{%Y-%m-%d} \t \w # '
   ```

2. Copy the release distribution gzipped tar file, e.g. analytics-1.0.14.tar.gz, to ncn-m001.

3. Extract the Analytics release distribution.

   ```
   ncn-m001# tar -zxvf analytics-1.0.14.tar.gz
   ```

4. Change directories to the extracted directory.

   ```
   ncn-m001# cd analytics-1.0.14
   ```

5. Verify the prerequisites.

   Cray System Management (CSM) has been installed and verified including the following Helm Charts.

   - gitea
   - cray-product-catalog
   - cray-cfs-api
   - cray-cfs-operator
   - cray-ims
   - cray-uan
   - WLM (slurm or pbs)

   Verify the installation of these components with the following.

   ```
   ncn-m001# helm ls -n services | grep -E \
   'gitea|cray-cfs-operator|cray-cfs-api|cray-ims|cray-product-\catalog|uan|slurm|pbs'
   ```

   Cray PE

   - If Cray PE will be installed on the system, ensure Cray PE is installed and configured before installing Analytics.

6. Delete the existing Analytics Install chart if it exists.

   ```
   ncn-m001# helm delete -n services cray-analytics-install || echo 'Ignore errors'
   ```

7. Run the following command.

```
ncn-m001# sed -i '/^AIupgrade/i set +o pipefail' ./install.sh
```

8. Run the installer, **`install.sh`**.

   ```
   ncn-m001# ./install.sh
   ```

9. Run the post-install script, to configure Analytics on the system (approximately 5 to 10 minutes).

   ```
   ncn-m001# ./analytics-post-install.sh
   ```

10. Once the NCN configuration has been applied to the NCNs, run the Analytics UAS configure script to publish the Analytics software into newly created UAIs. Note that the Compute, UAN, and NCN configurations can have the Analytics layer added at the bottom.

    ```
    ncn-m001# ./uas_setup_analytics.sh
    ```

11. Exit the typescript file started at the beginning of this procedure. Analytics is installed and configured.

    ```
    ncn-m001# exit
    ```

26.8 Analytics - Rebuild Pre-existing ck8s Node Images After 1.4.2 Patch Install

Update and Rebuild Images Using a SLES Base.

It is recommended to rebuild any pre-existing ck8s compute node images after the 1.4.2 patch is applied. If pre-existing images are not rebuilt, the existing ck8s compute node images may be vulnerable.