Hewlett Packard
Enterprise

HPE Cray Operating System Administration Guide: CSM on HPE Cray EX Systems (2.3.101) (S-8024)

Part Number: S-8024
Published: July 2022

HPE Cray Operating System Administration Guide: CSM on HPE Cray EX Systems

Contents

1 Copyright and Version

COS: 2.3.101-53

Doc git hash: 994ac7dc96c64495399412a2504f39378873d6b3

Generated: Wed Jul 13 2022

2 Content Projection Service

2.1 Content Projection Service (CPS)

The Content Projection Service (CPS) provides the root filesystem for compute nodes in conjunction with the Data Virtualization Service (DVS). Also, the Cray Programming Environment (PE) is provided as a separately-mounted filesystem to compute nodes and UAI/UAS nodes via CPS and DVS.

2.1.1 High-Level Administrator Workflow

As an administrator, CPS will likely be used in the following workflow:

1. Upload filesystem images to S3.

   See "Manage Artifacts with the Cray CLI" section in the CSM documentation for reference.

2. Pre-stage those images from S3 to CPS.

   See Add CPS Content.

3. Use BOS to configure compute nodes to use those images from CPS.

   See "Manage a BOS session" in the CSM documentation for reference.

4. Boot the compute nodes via BOS.

2.1.2 CPS Architecture

It can be helpful to have a high-level view of how CPS works. CPS is comprised of the following components that run under the control of Kubernetes on NCNs:

- CPS Broker: The broker provides the API service (https://api-gw-service-nmn.local/apis/v2/cps). Multiple instances of the `cray-cps` broker pod can exist. To enhance resiliency, each instance of the `cray-cps` pod will run on a separate node; two instances will not run on the same node. Broker pods communicate with the other CPS components via the CPS State Manager (etcd).
- CPS Content Manager (CM): The CM retrieves artifacts (file system images) from S3 to make them available to the CPS Projection Manager.
- CPS Projection Manager (PM): The PM makes artifacts available to other compute nodes via network file systems, such as DVS. One instance of CM and one instance of PM run in the `cray-cps-cm-pm` pod. There may be multiple instances of this pod, but each pod must run on a different node.
- CPS State Manager (etcd): The state manager enables the other CPS components to communicate about the current or desired state of the CPS service. There should be three copies of `cray-cps-etcd` running, each on a different NCN.

These CPS components interact with DVS, S3, and other HPE Cray EX components as shown in this diagram. The relationships between the CPS components described above are outlined in the following image.

The cpsmount.sh command asks the broker via HTTP requests to make a specific filesystem image available to a node with a particular network filesystem; the node then mounts that filesystem. CPS retrieves content objects from the Simple Storage Service (S3) repository and stores them on NCNs. CPS then serves the content from the NCN to other nodes over the network using DVS.

2.1.3 CPS Administrative Tasks

The following subsections are common administrative tasks for CPS. These tasks are essential for managing the CPS components mentioned above, and ensuring all components are in a healthy state.

- Scale CPS Broker Pods: Scaling the number of `cray-cps` pods is helpful for maintaining resiliency and load-balancing.
- Scale CPS cm-pm Pods: Scaling the number of `cm-pm` pods and controller where they run is also useful for resiliency and load-balancing when using CPS.
- Add CPS Content: Adding content to CPS is useful because it pre-stages it so the content will be downloaded by the `cray-cps-cm-pm` pods and will be ready when the first client tries to mount the new content.
- Remove CPS Content: Removing content downloaded by the `cray-cps-cm-pm` pods helps free up disk space on the nodes where those pods run.
- Troubleshoot CPS Issues: Checking the status of pods correlated to CPS components via Kubernetes is helpful for determining potential issues with CPS. More information about each troubleshooting operation is included in the troubleshooting section.

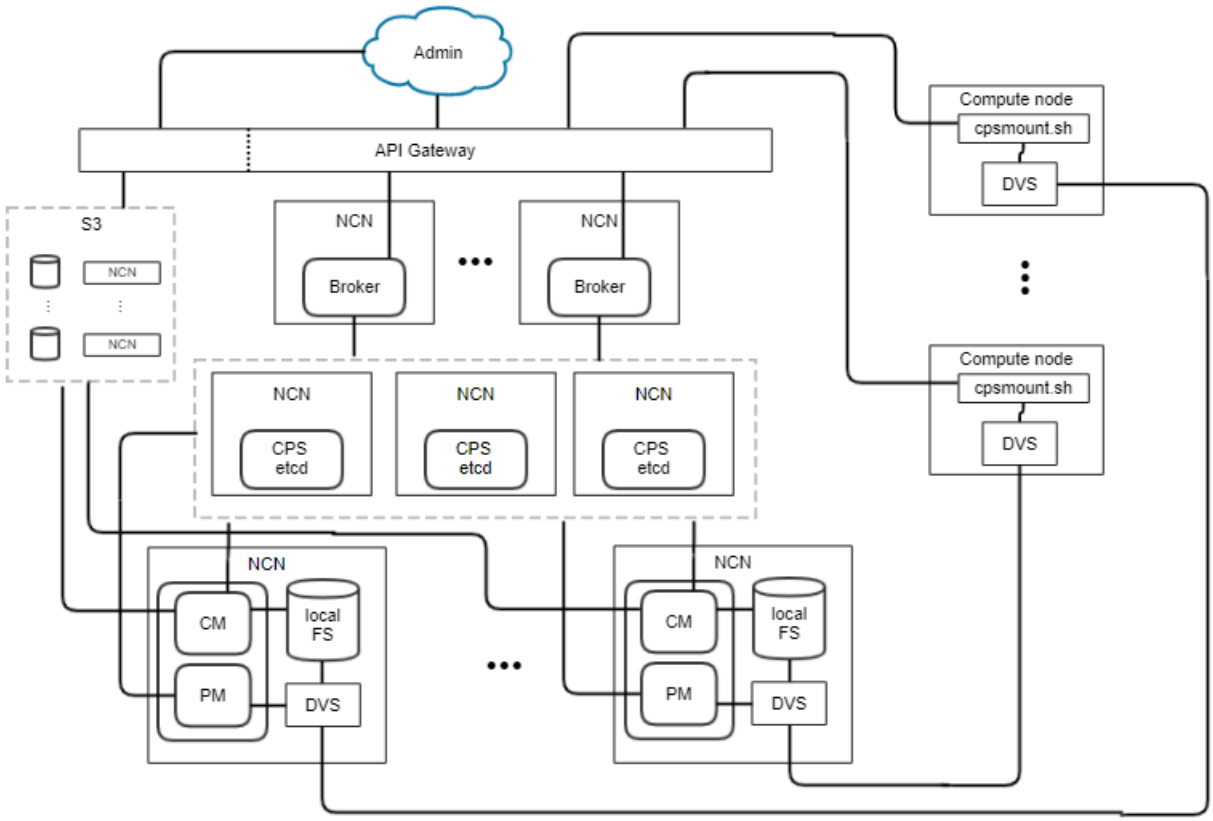


Figure 1: CPS Conceptual Diagram

2.1.4 CPS s3fs Migration

NOTE: The following actions are only necessary when performing an upgrade from a COS release prior to COS 2.3 to COS 2.3 or later. CPS will use s3fs without manual intervention if you are performing a fresh install or have already performed these actions as part of a previous upgrade.

Beginning with the COS 2.3 release, CPS uses s3fs to cache images on NCN worker nodes instead of creating NCN-local image copies in /var/lib/cps-local/. While upgrading to a COS release with support for s3fs, CPS will initially run in a mixed mode to temporarily provide NCN-local copies of images that were referenced prior to the upgrade. In this mixed s3fs and local copy mode, the CPS CM environment variable `CPS_USE_S3FS` is automatically set to `S3FS_LOCAL`. Completing the CPS s3fs upgrade requires all compute nodes and UANs be rebooted. After compute nodes and UANs have been rebooted, the CPS CM environment variable `CPS_USE_S3FS` should be manually updated from `S3FS_LOCAL` to `S3FS` as described below. After setting the value to `S3FS`, all the cray-cps-cm-pm pods will restart automatically, removing all NCN-local image copies.

The following subsections show how to configure `CPS_USE_S3FS` at the end of an upgrade to COS 2.3 and provide more in-depth details about the `CPS_USE_S3FS` environment variable.

2.1.4.1 Updating CPS_USE_S3FS at the end of an upgrade to COS 2.3

Perform this operation once all compute nodes and UANs have been rebooted as part of the upgrade to COS 2.3.

The following example shows the expected values you will see prior to modifying the values of `S3FS_LOCAL`.

```
ncn-m# kubectl edit ds/cray-cps-cm-pm -n services
...
      containers:
      - env:
        - name: CPS_USE_S3FS
          value: "S3FS_LOCAL"
        image: dtr.dev.cray.com/cray/cray-cps-cm:2.N.N
...
```

```
        name: cray-cps-cm
...
      - env:
        - name: CPS_USE_S3FS
          value: "yes"
        image: dtr.dev.cray.com/cray/cray-cps-pm:2.N.N
...
        name: cray-cps-pm
...
```

Change the `cray-cps-cm` environment variable `CPS_USE_S3FS` from `"S3FS_LOCAL"` to `"S3FS"` and exit the `kubectl` command. The following example shows the expected result once this step has completed:

```
ncn-m# kubectl edit ds/cray-cps-cm-pm -n services
...
      containers:
      - env:
        - name: CPS_USE_S3FS
          value: "S3FS"
        image: dtr.dev.cray.com/cray/cray-cps-cm:2.N.N
...
        name: cray-cps-cm
...
      - env:
        - name: CPS_USE_S3FS
          value: "yes"
        image: dtr.dev.cray.com/cray/cray-cps-pm:2.N.N
...
        name: cray-cps-pm
...
```

2.1.4.2 s3fs Environment Variable CPS_USE_S3FS

This section provides additional details on the `CPS_USE_S3FS` environment variable. No required steps are described in this section.

The `CPS_USE_S3FS` environment variable needs to be set in both CM (cray-cps-cm) and PM (cray-cps-pm) containers consistently. Updating or adding the `CPS_USE_S3FS` `env` value is done with this command:

```
ncn-m# kubectl edit ds/cray-cps-cm-pm -n services
```

The possible values for the `CPS_USE_S3FS` variable in the `cray-cps-cm` section are:

- `"NO_S3FS"`: NCN-local copies of images will be used. The `CPS_USE_S3FS` variable in the `cray-cps-pm` section must be set to `"no"`. This is the behavior used prior to COS 2.3.
- `"S3FS_LOCAL"`: s3fs as well as NCN-local copies of images will be used. The `CPS_USE_S3FS` variable in the `cray-cps-pm` section must be set to `"yes"`. This is the behavior used during the upgrade to COS 2.3.
- `"S3FS"`: Only s3fs will be used. The `CPS_USE_S3FS` variable in the `cray-cps-pm` section must be set to `"yes"`. This is the behavior used after the upgrade to COS 2.3 has completed.

The possible values for the `CPS_USE_S3FS` variable in the `cray-cps-pm` section are:

- `"no"`: Do not use s3fs; use NCN-local image copies instead. The `CPS_USE_S3FS` variable in the `cray-cps-cm` section must be set to `"NO_S3FS"`. This is the behavior used prior to COS 2.3.
- `"yes"`: Use s3fs. The `CPS_USE_S3FS` variable in the `cray-cps-cm` section must be set to `"S3FS_LOCAL"` or `"S3FS"`. This is the behavior used during the upgrade to COS 2.3 and after the upgrade to COS 2.3 has completed.

2.2 Scale CPS Broker Pods

The Content Projection Service (CPS) provides a Kubernetes pod for the `cray-cps-broker` RESTful API server container. The initial configuration for CPS provides two instances of this Kubernetes pod to provide resiliency and load-balancing. The number of `cray-cps` pods running for CPS can be adjusted. The following includes examples of how to increase or reduce the number of `cray-cps` pods created.

The cray-cps pod's Kubernetes deployment manifest is configured for PodAntiAffinity, which means there can't be more than one instance per NCN worker node. This limits the number of cray-cps pods to the number of NCN worker nodes. The scheduling and placement of pods are controlled by Kubernetes so CPS localization for the cps-broker can only control the number of pods. Use of other Kubernetes APIs not documented here can be used to prevent other NCNs from having cps-broker pods created on them.

To view the currently running cray-cps pods:

```
ncn-w001# kubectl -n services get po \
-l app.kubernetes.io/name=cray-cps
NAME                        READY   STATUS    RESTARTS   AGE
cray-cps-75cffc4b94-mwkz6   2/2     Running   225        7d15h
```

2.2.1 Increase the Number of cray-cps Pods

To increase the number of cray-cps pods by one:

```
ncn-w001# kubectl -n services scale --current-replicas=1 \
--replicas=2 deployment/cray-cps
deployment.extensions/cray-cps scaled
```

Confirm the number of pods has increased:

```
ncn-w001# kubectl -n services get po \
-l app.kubernetes.io/name=cray-cps
NAME                        READY   STATUS    RESTARTS   AGE
cray-cps-75cffc4b94-hll6n   2/2     Running   0          40s
cray-cps-75cffc4b94-mwkz6   2/2     Running   225        7d15h
```

2.2.2 Decrease the Number of cray-cps Pods

To decrease the number of cray-cps pods by one:

```
ncn-w001# kubectl -n services scale --current-replicas=2 \
--replicas=1 deployment/cray-cps
deployment.extensions/cray-cps scaled
```

Confirm the number of pods has decreased:

```
ncn-w001# kubectl -n services get po \
-l app.kubernetes.io/name=cray-cps
NAME                        READY   STATUS    RESTARTS   AGE
cray-cps-75cffc4b94-mwkz6   2/2     Running   225        7d15h
```

2.3 Scale CPS cm-pm Pods

CPS deployment command can be used for scaling the cray-cps-cm-pm pods up/down, as well as controlling where the pods run.

2.3.1 Retrieve CPS cm-pm Pod Information

The list command shows all of the worker nodes that are running cm-pm pods. This command is helpful for seeing the current scaling of cm-pm pods.

```
ncn-w001# cray cps deployment list
[[results]]
node = "ncn-w003"
podname = "cray-cps-cm-pm-dzz7b"
[[results.cpspods]]
name = "cray-cps-cm"
readiness = true
restartcount = 0
state = "running"

[[results.cpspods]]
```

```
name = "cray-cps-pm"
readiness = true
restartcount = 0
state = "running"

[[results.cpspods]]
name = "cray-cps-pm-helper"
readiness = true
restartcount = 0
state = "running"

[[results.cpspods]]
name = "istio-proxy"
readiness = true
restartcount = 0
state = "running"


[[results]]
node = "ncn-w002"
podname = "cray-cps-cm-pm-nh9dq"
[[results.cpspods]]
name = "cray-cps-cm"
readiness = true
restartcount = 0
state = "running"

...
```

The command can also be used to list CPS `cm-pm` pod information for specific worker nodes. In the example below, the information for worker nodes `ncn-w001` and `ncn-w002` are shown:

```
ncn-w001# cray cps deployment list --nodes "ncn-w001,ncn-w002"
[[results]]
node = "ncn-w002"
podname = "cray-cps-cm-pm-nh9dq"
[[results.cpspods]]
name = "cray-cps-cm"
readiness = true
restartcount = 0
state = "running"

[[results.cpspods]]
name = "cray-cps-pm"
readiness = true
restartcount = 0
state = "running"

[[results.cpspods]]
name = "cray-cps-pm-helper"
readiness = true
restartcount = 0
state = "running"

[[results.cpspods]]
name = "istio-proxy"
readiness = true
restartcount = 0
state = "running"
```

```
[[results]]
cpspods = []
node = "ncn-w001"
podname = "NA"
```

2.3.2 Add CPS cm-pm Pods

The update command is used to add cm-pm pods to additional nodes. Node lists can be used to select the worker nodes that the pods will run on. The following is an example of having CPS start cm-pm pods on ncn-w001 and ncn-w002.

```
ncn-w001# cray cps deployment update --nodes "ncn-w001,ncn-w002"
```

View the pods afterwards to verify the pods were added to the desired nodes:

```
ncn-w001# kubectl get pod -A |grep cps
services          cray-cps-69dc9986d5-h2skg       2/2     Running            0          27m
services          cray-cps-69dc9986d5-tkvt5       2/2     Running            0          26m
services          cray-cps-cm-pm-dzz7b            5/5     Running            0          26m
services          cray-cps-cm-pm-nh9dq            5/5     Running            0          25m
services          cray-cps-cm-pm-xjrtc            0/5     PodInitializing    0          7s
services          cray-cps-etcd-ltdcgmpclz        1/1     Running            0          20d
services          cray-cps-etcd-p6dsmbzpks        1/1     Running            0          20d
services          cray-cps-etcd-thvk5jshl2        1/1     Running            0          20d
services          cray-cps-wait-for-etcd-3-vxcdf  0/1     Completed          0          27m
```

There is also a –numpods parameter that can add additional cm-pm pods on any worker node that is not already running a cm-pm pod.

The number of cm-pm pods can't exceed the number of worker nodes on the system. For example, if two worker nodes are available, the –numpods parameter could be set as 2 to add two more cm-pm pods to the available worker nodes. If only one node is available, CPS will only start one cm-pm pod on that node, even if –numpods is set to 2.

```
ncn-w001# cray cps deployment update --numpods 2
```

View the pods afterwards to verify the number of pods was updated:

```
ncn-w001# kubectl get pod -A -o wide |grep cps
services cray-cps-69dc9986d5-h2skg   2/2  Running    0    33m   10.42.0.68    ncn-w001   <none>   <none>
services cray-cps-69dc9986d5-tkvt5   2/2  Running    0    32m   10.36.0.111   ncn-w003   <none>   <none>
services cray-cps-cm-pm-dzz7b        5/5  Running    0    32m   10.36.0.46    ncn-w003   <none>   <none>
services cray-cps-cm-pm-nh9dq        5/5  Running    0    31m   10.45.0.21    ncn-w002   <none>   <none>
services cray-cps-cm-pm-twf8v        0/5  Init:0/1   0    4s    <none>        ncn-w001   <none>   <none>
services cray-cps-etcd-ltdcgmpclz    1/1  Running    0    20d   10.45.0.68    ncn-w002   <none>   <none>
services cray-cps-etcd-p6dsmbzpks    1/1  Running    0    20d   10.42.0.38    ncn-w001   <none>   <none>
services cray-cps-etcd-thvk5jshl2    1/1  Running    0    20d   10.36.0.73    ncn-w003   <none>   <none>
services cray-cps-wait-for-etcd-3    0/1  Completed  0    33m   10.36.0.46    ncn-w003   <none>   <none>
```

It is important to be aware that if an incorrect node name is used when running an update command, the correct node names in the list will still be updated. For example, if ncn-w001 and ncn-w002 were being updated, and ncn-w002 was misspelled, ncn-w001 would still be updated.

```
ncn-w001# cray cps deployment update --nodes "ncn-w001,ncn-w002-typo"
Usage: cray cps deployment update [OPTIONS]
Try 'cray cps deployment update --help' for help.

Error: Not Found: node name: ['ncn-w002-typo'] not found
```

Check the pods to see that the correctly spelled worker node (ncn-w001 in this example) is still updated:

```
# kubectl get pod -A -o wide |grep cps|grep ncn-w001
services   cray-cps-69dc9986d5-h2skg     2/2  Running   0   37m    10.42.0.68   ncn-w001  <none>   <none>
services   cray-cps-cm-pm-8lkgc          5/5  Running   0   2m20s  10.42.0.61   ncn-w001  <none>   <none>
services   cray-cps-etcd-p6dsmbzpks      1/1  Running   0   20d    10.42.0.38   ncn-w001  <none>   <none>
```

2.3.3 Remove CPS cm-pm Pods

The delete command removes the cm-pm pod from any specified node. To remove the cm-pm pod from ncn-w001, the following command is used:

```
ncn-w001# cray cps deployment delete --nodes ncn-w001
```

View the pods afterwards to verify the pod was removed from the desired node:

```
ncn-w001# kubectl get pod -A -o wide | grep cps
services  cray-cps-744f7dc48f-s9zbk        2/2  Running     0  15d  10.45.0.59  ncn-w001  <none>  <none>
services  cray-cps-744f7dc48f-zsx4q        2/2  Running     0  15d  10.42.0.76  ncn-w002  <none>  <none>
services  cray-cps-cm-pm-54j9f             5/5  Running     4  14d  10.36.0.62  ncn-w003  <none>  <none>
services  cray-cps-cm-pm-dbprv             5/5  Running     5  14d  10.42.0.5   ncn-w002  <none>  <none>
services  cray-cps-cray-jobs-cray-cps-job  0/2  Completed   0  15d  10.42.0.57  ncn-w002  <none>  <none>
services  cray-cps-etcd-477wvnlg7j         1/1  Running     0  15d  10.45.0.34  ncn-w001  <none>  <none>
services  cray-cps-etcd-f7wsnlkvw6         1/1  Running     0  15d  10.36.0.65  ncn-w003  <none>  <none>
services  cray-cps-etcd-qggjq24ssw         1/1  Running     0  15d  10.42.0.84  ncn-w002  <none>  <none>
services  cray-cps-wait-for-etcd-4-rb7f8   0/1  Completed   0  15d  10.42.0.92  ncn-w002  <none>  <none>
```

2.4 Configure the Minimum Number of Functional CPS PM Servers

Previously, the cps-broker did not set the transport readyForMount to True unless all requested cm-pm pods were ready. The Data Virtualization Service (DVS) load balancing and fail-over features took advantage of all DVS servers as there was no method to discover servers after the file system was mounted. However, this was overly constraining on large systems where it was acceptable to mount DVS file systems using, for example, 18 of the 20 total DVS servers.

The minimum number of cps-pm containers required to be ready in order to set readyForMount to True is now configurable. The cps-broker now allows the minimum number of cps-pm instances that must be ready to be configurable.

The minimum number is specifiable as both an absolute number and a percentage of the total. The minimum number of PM servers used by cps-broker is calculated using the following container environment variables:

- CPS_PM_MIN_NR: 0
- CPS_PM_MIN_PCT: 50

Using these variables, the minimum PM servers are determined. In the following formula, N is the available PM replicas and M is the configured replicas. The readyForMount value is set as follows:

- If N == 0, False
- Else If N >= M, True
- Else N >= max(CPS_PM_MIN_NR,CPS_PM_MIN_PCT% of M)

This formula allows the required number of available replicas to be set to an absolute value, a percentage of the configured replicas, or the maximum of an absolute value and a percentage.

The following is an example where there are 3 CPS PM servers configured, and a request has arrived to mount an Simple Storage Service (S3) artifact. Using the defaults, once 2 of the 3 servers report ready, CPS Broker will inform the requester to proceed with accessing the requested contents.

```
ncn-w001# cray cps transports list \
--s3path s3://boot-images/dac98526-e4b7-4a9d-bd3b-917faa1e6f0a/manifest.json --transport dvs

ERROR = []
artifactID = "eebce16144882683b504797754e0f5ba"
[[transports]]
exportPath = "/var/lib/cps-local/eebce16144882683b504797754e0f5ba"
ready = 2                    <<-- 2 of 3 servers ready
readyForMount = true         <<-- Indicates transport is ready
servers = [ "10.252.0.4", "10.252.0.5",]
total = 3             <<-- 3 total servers
type = "dvs"
[[transports.status]]
detail = "eebce16144882683b504797754e0f5ba-dvs enabled"
```

```
replicaID = "10.252.0.4"
status = "ready"[[transports.status]]
detail = "eebce16144882683b504797754e0f5ba-dvs enabled"
replicaID = "10.252.0.5"
status = "ready"[[transports.status]]
detail = "waiting for content eebce16144882683b504797754e0f5ba: loading"
replicaID = "10.252.0.6"
status = "loading"
```

2.4.1 Impact of CPS PM readyForMount State on cpsmount.sh

The CPS utility script cpsmount.sh is used by the compute nodes and command line users to request that an artifact be provided for mounting on a compute node or NCN clients. Any request for content from CPS made by cpsmount.sh is a REST API call and results in the script polling CPS Broker for status of the request at a pre-determined polling rate.

Previously, if there was an issue with one of the PM servers loading content, the script would poll until it timed-out based on arguments passed to the script. The impact of the change to a minimum number of functional CPS PMs allows content to be accessed, even if not ideal loading conditions exists.

2.5 Set the Minimum Value for Functional CPS PM Servers

The `cray-cps` Kubernetes pod provides the control to set the minimum number of functional Content Projection Service (CPS) Projection Manager (PM) servers. The CPS_PM_MIN_NR and CPS_PM_MIN_PCT variables are Kubernetes container environment variables, so there is currently no way to persist changes made to the CPS microservice.

This procedure describes how to change the environment variable values with a running CPS deployment. Adjusting the minimum functional CPS PM servers allows admins to enhance load balancing.

2.5.1 Procedure

1. Find the currently running `cray-cps` pods to use for verification after changing the environment variables.

   ```
   ncn-w001# kubectl -n services get pods -l app.kubernetes.io/name=cray-cps
   NAME                        READY   STATUS    RESTARTS   AGE
   cray-cps-58c7cc5479-qfkl6   2/2     Running   0          5h20m
   cray-cps-58c7cc5479-zp5ss   2/2     Running   0          5h20m
   ```

2. Check the current settings for the `cray-cps` deployment environment variables.

   ```
   ncn-w001# kubectl -n services set env deployment/cray-cps \
   -c cray-cps-broker --list
   # Deployment cray-cps, container cray-cps-broker
   ETCD_HOST=cray-cps-etcd-client
   ETCD_PORT=2379
   ETCDCTL_API=3
   ETCD_SERVICE=cray-cps-etcd-client
   CPS_LOGGING_LEVEL=INFO
   CPS_PM_MIN_NR=0
   CPS_PM_MIN_PCT=50
   ```

3. Set the desired value for one of the environment variables in the previous step.

   Use one of the options below to update the environment variables:

   - Set the environment variable value directly.

     In this example, CPS_PM_MIN_NR is changing from 0 to 1.

     ```
     ncn-w001# kubectl -n services set env deployment/cray-cps \
     -c cray-cps-broker CPS_PM_MIN_NR=1
     deployment.extensions/cray-cps env updated
     ```

   - Use kubectl to edit the deployment.

```
ncn-w001# kubectl -n services edit deployment/cray-cps
deployment.extensions/cray-cps edited
```

4. Verify the environment variables have been updated.

```
ncn-w001# kubectl -n services set env deployment/cray-cps \
-c cray-cps-broker --list
# Deployment cray-cps, container cray-cps-broker
ETCD_HOST=cray-cps-etcd-client
ETCD_PORT=2379
ETCDCTL_API=3
ETCD_SERVICE=cray-cps-etcd-client
CPS_LOGGING_LEVEL=INFO
CPS_PM_MIN_NR=1
CPS_PM_MIN_PCT=50
```

5. Confirm the pods are running.

   The pods are automatically redeployed after updating the environment variables.

```
ncn-w001# kubectl -n services get pods | grep cps | \
grep -v cm-pm | grep -v etcd | grep -v jobs
services     cray-cps-58c7cc5479-qfkl6       2/2     Running           0     5h29m
services     cray-cps-58c7cc5479-zp5ss       2/2     Terminating       0     5h29m
services     cray-cps-768499789b-5dtxw       2/2     Running           0     43s
services     cray-cps-768499789b-r4989       0/2     Pending           0     6s
...
services     cray-cps-768499789b-5dtxw       2/2     Running           0     5m15s
services     cray-cps-768499789b-r4989       2/2     Running           0     4m38s
```

2.6 Add CPS Content

Add or load content to the Content Projection Service (CPS) that can be shared with NCNs and compute nodes. Content can be added to CPS with or without a transport. A transport can also be added to content already being managed by CPS.

Pre-stages content so that it is downloaded by the `cray-cps-cm-pm` pods and it is ready when the first client tries to mount the content.

2.6.1 Prerequisites

- The Cray command line interface (CLI) tool is initialized and configured on the system.

2.6.2 Limitations

Several commands in this procedure use the `--etag` parameter, which is currently accepted but not used.

2.6.3 Procedure

1. Add the desired content to CPS.

   There are a several different ways to add content to CPS described below.

   - Add content without a transport.

```
ncn-w001# cray cps contents create --s3path FULL_S3PATH --etag ETAG_ID
s3path = "s3://boot-images/08673352-fc26-4cc6-883a-f79e1ed3052b/rootfs"
etag = "90d7b9f298d1a638f5a80b3876691ccc-167"
transport = []
```

     For example:

```
ncn-w001# cray cps contents create --s3path s3://boot-images/
08673352-fc26-4cc6-883a-f79e1ed3052b/rootfs --etag 90d7b9f298d1a638f5a80b3876691ccc-167
```

   - Add content with a transport.

```
ncn-w001# cray cps contents create --s3path FULL_S3PATH \
--etag ETAG_ID --transport TRANSPORT_NAME
[source]
----
s3path = "s3://boot-images/08673352-fc26-4cc6-883a-f79e1ed3052b/rootfs"
etag = "90d7b9f298d1a638f5a80b3876691ccc-167"
transport = [ "dvs",]
----
```

For example:

```
ncn-w001# cray cps contents create --s3path s3://boot-images/
08673352-fc26-4cc6-883a-f79e1ed3052b/rootfs --etag
90d7b9f298d1a638f5a80b3876691ccc-167 --transport dvs
```

- Add a transport to existing content.

```
ncn-w001# cray cps transports create --s3path FULL_S3PATH \
--etag ETAG_ID --transport TRANSPORT_NAME
transport = [ "dvs",]
artifactID = "76df050e1fde782a58365504477a7af6"
```

For example:

```
ncn-w001# cray cps transports create --s3path s3://boot-images/
08673352-fc26-4cc6-883a-f79e1ed3052b/rootfs --etag
90d7b9f298d1a638f5a80b3876691ccc-167 --transport dvs
```

2. Confirm the new content was added to CPS.

   To list all CPS content:

```
ncn-w001# cray cps contents list
exportPath = "/var/lib/cps-local/76df050e1fde782a58365504477a7af6"
s3path = "s3://boot-images/e1079ab0-ae73-43ee-9b75-8f77d74001e0/rootfs"
ERROR = []
transports = [ "dvs",]
artifactID = "3c070d4f16dbd81e0c1870a751251880"
[[results.exportStatus]]
status = "ready"
type = "dvs"

[results.contentReplicas]
ready = 2
total = 2
[[results.contentReplicas.status]]
status = "ready"
replicaID = "10.252.1.5"
detail = "Artifact_id=3c070d4f16dbd81e0c1870a751251880 is ready"

[[results.contentReplicas.status]]
status = "ready"
replicaID = "10.252.1.6"
detail = "Artifact_id=3c070d4f16dbd81e0c1870a751251880 is ready"
```

   To list contents with a specific S3 path:

```
ncn-w001# cray cps contents list --s3path FULL_S3PATH --etag ETAG_ID
exportPath = "/var/lib/cps-local/76df050e1fde782a58365504477a7af6"
s3path = "s3://boot-images/08673352-fc26-4cc6-883a-f79e1ed3052b/rootfs"
ERROR = []
transports = [ "dvs",]
artifactID = "76df050e1fde782a58365504477a7af6"
```

```
[[exportStatus]]
status = "ready"
type = "dvs"

[contentReplicas]
ready = 2
total = 2
[[contentReplicas.status]]
status = "ready"
replicaID = "10.252.0.5"
detail = "Artifact_id=76df050e1fde782a58365504477a7af6 is ready"

[[contentReplicas.status]]
status = "ready"
replicaID = "10.252.0.6"
detail = "Artifact_id=76df050e1fde782a58365504477a7af6 is ready"
```

To list the transport attributes:

```
ncn-w001# cray cps transports list --s3path FULL_S3PATH \
--etag ETAG_ID --transport TRANSPORT_NAME
ERROR = []
artifactID = "76df050e1fde782a58365504477a7af6"
[[transports]]
exportPath = "/var/lib/cps-local/76df050e1fde782a58365504477a7af6"
readyForMount = true
servers = [ "10.252.0.5", "10.252.0.6",]
ready = 2
total = 2
type = "dvs"
[[transports.status]]
status = "ready"
replicaID = "10.252.0.5"
detail = "76df050e1fde782a58365504477a7af6-dvs enabled"

[[transports.status]]
status = "ready"
replicaID = "10.252.0.6"
detail = "76df050e1fde782a58365504477a7af6-dvs enabled"
```

2.7 Remove CPS Content

This procedure describes how to remove content from the Content Projection Service (CPS) that is no longer needed.

Following this procedure will remove content downloaded by the `cray-cps-cm-pm` pods and free up disk space on the nodes where those pods run.

2.7.1 Prerequisites

- The Cray command line interface (CLI) tool is initialized and configured on the system.

2.7.2 Procedure

1. List the content currently held by CPS.

   Listing the content is helpful for obtaining the S3 path and transport information that can be used to identify and remove specific content.

   ```
   ncn-w001# cray cps contents list
   exportPath = "/var/lib/cps-local/76df050e1fde782a58365504477a7af6"
   s3path = "s3://boot-images/08673352-fc26-4cc6-883a-f79e1ed3052b/rootfs" **<<-- Note this value**
   ```

```
ERROR = []
transports = [ "dvs",]
artifactID = "76df050e1fde782a58365504477a7af6"
[[exportStatus]]
status = "ready"
type = "dvs"[contentReplicas]
ready = 2
total = 2
[[contentReplicas.status]]
status = "ready"
replicaID = "10.252.0.5"
detail = "Artifact_id=76df050e1fde782a58365504477a7af6 is ready"[[contentReplicas.status]]
status = "ready"
replicaID = "10.252.0.6"
detail = "Artifact_id=76df050e1fde782a58365504477a7af6 is ready"
```

2. Remove a transport from existing content without removing the content.

   ```
   ncn-w001# cray cps transports delete --transport dvs --s3path S3_PATH
   ```

3. Remove specific content from CPS using the s3path for the content.

   Removing content from CPS will also remove all transports for that content.

   Warning: Do not remove content that is currently in use, as that may cause compute nodes using that content to crash.

   ```
   ncn-w001# cray cps contents delete --s3path S3_PATH
   ```

   For example:

   ```
   ncn-w001# cray cps contents delete --s3path s3://boot-images/
   08673352-fc26-4cc6-883a-f79e1ed3052b/rootfs
   ```

2.8 Clean up CPS Content

CPS contents should be periodically checked and old contents removed from CPS to avoid running out of disk space. CPS contents expects the source data (file object) to be in the S3 storage, but they might get deleted by IMS before the CPS contents are removed especially the old contents. To help identify which contents can be removed, CPS content clean up script is added. This helper script can list all the CPS contents and which ones are currently DVS mounted or not used. Optionally, the script can remove CPS contents that are not in use.

2.8.1 Prerequisites

- The CPS script /opt/cray/cps-utils/cps-cleanup/cleanup_cps.py has been installed.

2.8.2 Procedure

1. List all contents

   ```
   ncn-m001# cd /opt/cray/cps-utils/cps-cleanup
   ncn-m001# ./cleanup_cps.py
   ```

   NOTE: `cleanup_cps.py` defaults to scan all compute node. Optionally, it takes –xname option with list of comma separated xnames like, XNAME1,XNAME2,... to search only the listed compute nodes. List at least one or two nodes for different boot images that are currently used to boot compute nodes and UAN nodes as well).

2. Remove all unused contents by scanning all compute and UAN nodes

   ```
   ncn-m001# ./cleanup_cps.py --delete
   ```

   NOTE: To delete manually instead of delete all with the above command, use the following.

   ```
   ncn-m001# cray cps contents delete --s3path <a s3path from the unused list from list contents output>
   ```

2.9 Update CFS with Local Changes

Push the updated, local copy of the COS configuration management git repository upstream to VCS. This ensures that DVS configuration changes are saved in CFS which can then apply them to nodes and images.

- Prepare to Configure DVS with VCS
- Make changes in the local copy of the COS configuration management git repository.

To save locally made DVS configuration changes in CFS, HPE Cray EX administrators must:

1. Commit and push the changes.
2. Download and edit both the CN and NCN CFS configurations.
3. Upload the updated configurations to CFS.

2.9.1 Procedure

1. Commit the changes and push them into the working repository created in Step 3 of Prepare to Configure DVS with VCS.

   The following example assumes that this local repository is named `local_cos.1.4.0`.

   ```
   ncn-w# git commit -am 'Added COMMIT_COMMENT'
   ncn-w# git push --set-upstream origin local_cos.1.4.0
   ```

2. Obtain the most recent git commit hash for this branch, made in the previous step, and save it for later use.

   ```
   ncn-m001# git rev-parse --verify HEAD
   be2bd370ed246182ec765ca999d2fac4d355443a
   ncn-m001# export HEAD_COMMIT_HASH=be2bd370ed246182ec765ca999d2fac4d355443a
   ```

3. Navigate out of the cos-config-management directory and download the current NCN Personalization configuration.

   ```
   ncn-w# cd ..
   ncn-w# cray cfs configurations describe ncn-personalization --format json \> ncn-personalization.json
   ncn-w# cat ncn-personalization.json
   {
     "lastUpdated": "2020-12-16T16:34:19Z",
     "layers": [
       {
         "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
         "commit": "e4748060e1a609c155537b41afdf225243e568e6",
         "name": "cos-integration-1.4.0",
         "playbook": "ncn.yml"
       }
     ],
     "name": "ncn-personalization"
   }
   ```

4. Edit and update the JSON file.

   1. Remove the `lastUpdated` key.
   2. Remove the second `name` key (the one whose value `ncn-personalization`).
   3. Remove the comma after the `layers` array.
   4. Update the git commit hash with the one obtained from the “Obtain the most recent git commit hash...” step above.

   The JSON file should now resemble this one:

   ```
   ncn-w# echo ${HEAD_COMMIT_HASH}
   be2bd370ed246182ec765ca999d2fac4d355443a
   ncn-w# vi ncn-personalization.json
   . . .
   ncn-w# cat ncn-personalization.json
   {
     "layers": [
   ```

```
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
      "commit": "be2bd370ed246182ec765ca999d2fac4d355443a",
      "name": "cos-integration-1.4.0",
      "playbook": "ncn.yml"
    }
  ]
}
```

5. Upload the new configuration into CFS.

```
ncn-w# cray cfs configurations update ncn-personalization --file ncn-personalization.json --format json
{
  "lastUpdated": "2020-12-17T17:51:09Z",
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
      "commit": "be2bd370ed246182ec765ca999d2fac4d355443a",
      "name": "cos-integration-1.4.0",
      "playbook": "ncn.yml"
    }
  ],
  "name": "ncn-personalization"
}
```

There is no need to update the CFS components for the NCNs. Those components still refer to the `ncn-personalization` configuration by name.

6. Obtain the name of the current CFS configuration for the CNs by using the following methods. Skip this step if the name of the current CFS configuration for the CNs is already known.

   Each BOS session template has a section indicating which CFS configuration is used.

   - List all the BOS session templates using the following command. Then search for the BOS session template used by the CNs.

     ```
     ncn-w001# cray bos sessiontemplate list --format json
     ```

   - Print out the current BOS session template for the CNs. The template name is `cos-compute-2.0.17` in following command example. The lines indicating the current configuration for CNs are in bold in the following example output.

     ```
     ncn-w001# cray bos sessiontemplate describe --format json cos-compute-2.0.17
     {
       "boot_sets": {
         "compute": {
           "boot_ordinal": 2,
           "etag": "391bf701610d6e8f4c25781f7265cba9",
           "kernel_parameters": "ip=dhcp quiet spire_join_token=${SPIRE_JOIN_TOKEN}",
           "network": "nmn",
           "node_roles_groups": [
             "Compute"
           ],
           "path": "s3://boot-images/f7a03fa5-7197-4f02-b385-146b4ed4b032/manifest.json",
           "rootfs_provider": "cpss3",
           "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:nmn0",
           "type": "s3"
         }
       },
       **"cfs": {
         "configuration": "cos-config-2.0.17"**
       },
       "enable_cfs": true,
     ```

```
        "name": "cos-compute-2.0.17"
      }
```

7. Download the current CN configuration.

   Replace CN_CONFIG in the following command with name of the current CFS configuration for CNs that was obtained in the previous step. In the previous example, the CFS configuration name was cos-config-2.0.17.

   ```
   ncn-w# cray cfs configurations describe CN_CONFIG --format json \> CN_CONFIG.json
   ```

8. Make the same changes to the CN configuration JSON file that were made to the NCN file in Step 4.

9. Upload the new CN configuration into CFS.

   ```
   ncn-w# cray cfs configurations update CN_CONFIG --file CN_CONFIG.json --format json
   ```

Continue performing the procedure that referred to this one.

2.10 Troubleshoot CPS Issues

The following are common things to check when troubleshooting issues with the Content Projection Service (CPS). It is helpful to check the status of pods correlated to CPS components, and to view their logs via Kubernetes.

2.10.1 Confirm CPS Pods are Running

Check the status of the `cray-cps` pods to see if any of the CPS components are in an error state, or if the expected number of pods for each component are available.

Ensure that the following conditions are true:

- A minimum of two `cray-cps` pods (broker) are running.
- A minimum of two `cray-cps-cm-pm` pods are running.
- Three copies of `cps-etcd` exist, each running on a different node.

Confirm there is the expected number of `cray-cps`, `cray-cps-cm-pm`, and `cray-cps-etcd` pods present, and that they are all in a Running state:

```
ncn-w001# kubectl get pod -A -o wide | grep cray-cps
services  cray-cps-cm-pm-fhtxj  5/5  Running 0  22h  10.42.0.6   ncn-w001   <none>  XZ<none>
services  cray-cps-cm-pm-wj5g9  5/5  Running 0  22h  10.40.0.64  ncn-w002   <none>  <none>
services  cray-cps-cray-jobs-cray-cps-job-1-drlpl 0/2  Completed 0  36h  10.39.0.54 ncn-w003 <none> <none>
services  cray-cps-d97d9699d-gpj2f 2/2 Running 0 36h 10.42.0.81  ncn-w001   <none>  <none>
services  cray-cps-d97d9699d-gvt2x 2/2 Running 0 36h  10.39.0.55  ncn-w003  <none> <none>
services  cray-cps-etcd-2f5vqm9hzr 1/1 Running 0 36h  10.40.0.81  ncn-w002  <none> <none>
services  cray-cps-etcd-j8cjkf5slc 1/1 Running 0 36h  10.42.0.83  ncn-w001  <none> <none>
services  cray-cps-etcd-szv2qdbpc8 1/1 Running 0 36h  10.39.0.64  ncn-w003  <none> <none>
services  cray-cps-wait-for-etcd-1-mlq9t 0/1 Completed 0 36h 10.39.0.53 ncn-w003 <none><none>
```

If two `cray-cps-etcd` pods are running on the same node, then follow the instructions in “Rebalance Healthy etcd Clusters” in the CSM documentation to spread them across separate nodes.

2.10.2 Verify the CPS Broker is Ready

The CPS broker pods are needed to communicate with the other CPS components via the CPS State Manager (etcd).

Verify content for the `cray-cps` broker pods is ready by running the cps content list command.

```
ncn-w001# cray cps contents list |grep s3path
s3path = "s3://boot-images/15ab96a1-fefb-4d67-8292-023043aa1546/rootfs"
s3path = "s3://boot-images/87cabdc2-8919-49a9-b22b-f3d5265cdabf/rootfs"
s3path = "s3://boot-images/cray-l1diag-sles15-1.0.0.x86_64.squashfs"
s3path = "s3://boot-images/Analytics/Cray-Analytics.x86_64-base.squashfs"
```

Find the s3path for the desired content.

```
ncn-w001# cray cps contents list --s3path FULL_S3PATH
s3path = "s3://boot-images/87cabdc2-8919-49a9-b22b-f3d5265cdabf/rootfs"
exportPath = "/var/lib/cps-local/38eaef961bcd9eb49aaa1b89e41558c0"
ERROR = []
transports = [ "dvs",]
artifactID = "38eaef961bcd9eb49aaa1b89e41558c0"
[[exportStatus]]
status = "ready"
type = "dvs"

[contentReplicas]
ready = 2
total = 2
[[contentReplicas.status]]
status = "ready"
replicaID = "10.252.0.6"
detail = "Artifact_id=38eaef961bcd9eb49aaa1b89e41558c0 is ready"

[[contentReplicas.status]]
status = "ready"
replicaID = "10.252.0.7"
detail = "Artifact_id=38eaef961bcd9eb49aaa1b89e41558c0 is ready"
```

Check the `cray-cps-broker` logs and make sure no errors are in the logs.

```
ncn-w001# kubectl logs -n services -c cray-cps-broker CPS_POD_ID |grep -e ERR -e err
```

2.10.3 Verify the Content Manager (CM) is Ready

The CM is needed to retrieve artifacts (file system images) from S3 to make them available to the CPS Projection Manager. Check the CM logs to make sure no errors are in the logs related to etcd.

Confirm that reconciliation, indicated with INFO:root:reconcile_content_state, is finished without errors.

To view the full log:

```
ncn-w001# kubectl logs -n services -c cray-cps-cm CRAY-CPS-CM-PM-ID
CPS Content Manager 2.0.3
INFO:root:logging level set to: 20
INFO:root:using: etcd_lease_ttl=200 min_remaining_ttl=5
INFO:root:bucket_name: boot-images
INFO:root:s3_endpoint_url=https://rgw-vip.local
INFO:root:int_endpoint_url=http://cray-s3.ceph-rgw.svc.cluster.local
INFO:root:ext_endpoint_url=http://rgw.local:8080
INFO:root:use_ssl: False
INFO:root:Register handler for SIGTERM
INFO:root:reconcile_content_state running...
INFO:root:cleanup_workdir running...
...
INFO:root:reconcile_content_state: done
```

To check if there are errors in the logs:

```
ncn-w001# kubectl logs -n services -c cray-cps-cm CRAY-CPS-CM-PM-ID |grep -E "ERROR\|CRITICAL"
ERROR:root:stage_download_s3 572c7fc99a781bc1e85d4e459e3eca6a val=OrderedDict([('artifact_id',
'572c7fc99a781bc1e85d4e459e3eca6a'), ('etag', ''),
('s3_path', 's3://boot-images/PE/forge_license.dat')])
failed: An error occurred (NoSuchKey) when calling the GetObject operation: Unknown
...
```

2.10.4 Verify Transport is Ready

Check that the transport for content is ready by finding the transport status for a rootfs image used to boot a node.

```
ncn-w001# cray cps transports list --transport dvs --s3path FULL_S3PATH
ERROR = []
artifactID = "072d27aa4c09e321106726f79aca4651"
[[transports]]
exportPath = "/var/lib/cps-local/072d27aa4c09e321106726f79aca4651"
readyForMount = true
servers = [ "10.252.0.6", "10.252.0.7",]
ready = 2
total = 2
type = "dvs"
[[transports.status]]
status = "ready"
replicaID = "10.252.0.6"
detail = "072d27aa4c09e321106726f79aca4651-dvs enabled"

[[transports.status]]
status = "ready"
replicaID = "10.252.0.7"
detail = "072d27aa4c09e321106726f79aca4651-dvs enabled"
```

To quickly find out if a transport is usable, use grep to look for readyForMount:

```
ncn-w001# cray cps transports list --transport dvs --s3path FULL_S3PATH |grep readyForMount
readyForMount = true
```

2.10.5 Verify Projection Manager (PM) is Ready

The PM needs to be in a Ready state to make artifacts available to other compute nodes via network file systems. Check the PM logs to make sure no errors in the logs are connected to etcd.

Verify that reconciliation is finished without errors. At the end of reconciliation, — current_transports: ... will be displayed in the output. Once reconciliation starts, it should end, but the retry can continue. The important thing is that reconciliation starts at some point.

To view the full log:

```
ncn-w001# kubectl logs -n services -c cray-cps-pm CRAY-CPS-CM-PM-ID
CPS Projection Manager 2.0.3
INFO:root:logging level set to: 20
INFO:root:Waiting for DVS interfaces in /mnt/dvs-interfaces/INTERFACES ...
INFO:root:Waiting for DVS interfaces in /mnt/dvs-interfaces/INTERFACES ...
INFO:root:Waiting for DVS interfaces in /mnt/dvs-interfaces/INTERFACES ...
INFO:root:Waiting for DVS interfaces in /mnt/dvs-interfaces/INTERFACES ...
INFO:root:Waiting for DVS interfaces in /mnt/dvs-interfaces/INTERFACES ...
INFO:root:Waiting for DVS interfaces in /mnt/dvs-interfaces/INTERFACES ...
INFO:root:Waiting for DVS interfaces in /mnt/dvs-interfaces/INTERFACES ...
INFO:root:Waiting for DVS interfaces in /mnt/dvs-interfaces/INTERFACES ...
INFO:root:Waiting for DVS interfaces in /mnt/dvs-interfaces/INTERFACES ...
INFO:root:Waiting for DVS interfaces in /mnt/dvs-interfaces/INTERFACES ...
INFO:root:DVS interfaces ['10.252.0.6']
INFO:root:using: etcd_lease_ttl=200
INFO:root:host=10.252.0.6 path=/var/lib/cps-local interfaces=['10.252.0.6']
INFO:root:get_all_transports
INFO:root:{('e698dabb95baa82043ce48f3a4a8d224', 'dvs'), ('fb4d7f30dbe64b9d2a51e43253c06757', 'dvs'),
('3ac214ea0ecb73f37234731586d533b4', 'dvs'),
('5ef8c49101831b11aa89e93b2ecd37ff', 'dvs'),
('b98be56101c32dda1dfc13d7185ba171', 'dvs')}
INFO:root:get_all_transport_statuses
```

```
INFO:root: -- need_to_disable: set()
INFO:root: -- missing_transports: {('5ef8c49101831b11aa89e93b2ecd37ff', 'dvs'),
('e698dabb95baa82043ce48f3a4a8d224', 'dvs'),
('b98be56101c32dda1dfc13d7185ba171', 'dvs'),
('fb4d7f30dbe64b9d2a51e43253c06757', 'dvs'),
('3ac214ea0ecb73f37234731586d533b4', 'dvs')}
INFO:root:get_artifact_status - 5ef8c49101831b11aa89e93b2ecd37ff - 10.252.0.6
INFO:root:5ef8c49101831b11aa89e93b2ecd37ff-dvs enabled
INFO:root:put_transport_status - 5ef8c49101831b11aa89e93b2ecd37ff
- 10.252.0.6 - dvs - ready -
5ef8c49101831b11aa89e93b2ecd37ff-dvs enabled -
/var/lib/cps-local/5ef8c49101831b11aa89e93b2ecd37ff - ['10.252.0.6']
INFO:root:5ef8c49101831b11aa89e93b2ecd37ff: dvs
INFO:root:get_artifact_status - e698dabb95baa82043ce48f3a4a8d224
- 10.252.0.6
INFO:root:e698dabb95baa82043ce48f3a4a8d224-dvs enabled
INFO:root:put_transport_status - e698dabb95baa82043ce48f3a4a8d224
- 10.252.0.6 - dvs - ready - e698dabb95baa82043ce48f3a4a8d224-dvs enabled
- /var/lib/cps-local/e698dabb95baa82043ce48f3a4a8d224 - ['10.252.0.6']
INFO:root:e698dabb95baa82043ce48f3a4a8d224: dvs
...
INFO:root:get_all_transport_statuses
INFO:root: -- current_transports: {('e698dabb95baa82043ce48f3a4a8d224', 'dvs'),
('fb4d7f30dbe64b9d2a51e43253c06757', 'dvs'),
('3ac214ea0ecb73f37234731586d533b4', 'dvs'),
('5ef8c49101831b11aa89e93b2ecd37ff', 'dvs'),
('b98be56101c32dda1dfc13d7185ba171', 'dvs')}
```

To check if there are errors in the logs:

```
ncn-w001# kubectl logs -n services -c cray-cps-cm CRAY-CPS-CM-PM-ID |grep -E 'ERROR|CRITICAL'
```

2.10.6 Verify DVS is Loaded

Find the cray-cps-cm-pm pods and figure out which nodes are running the pods.

```
ncn-w001# kubectl get pod -A -o wide |grep cray-cps-cm-pm
services cray-cps-cm-pm-268fr 5/5 Running  0 8m28s 10.42.0.47    ncn-w002  <none>  <none>
services cray-cps-cm-pm-2qnb7 5/5 Running  0 9m18s 10.47.192.64  ncn-w003  <none>  <none>
```

Verify that DVS modules are loaded on `ncn-w002`.

```
ncn-w001# ssh ncn-w002 'lsmod |grep dvs'
dvs                   405504  0
dvsipc                159744  2 dvs
dvsipc_lnet            57344  1 dvsipc
dvsproc               131072  3 dvsipc,dvsipc_lnet,dvs
craytrace              20480  5 dvsipc,dvsproc,dvsipc_lnet,dvs
dvskatlas              20480  4 dvsipc,dvsproc,dvsipc_lnet,dvs
lnet                  626688  3 ksocklnd,dvsipc_lnet
libcfs                512000  3 ksocklnd,lnet,dvsipc_lnet
```

Verify that DVS modules are loaded on `ncn-w003`.

```
ncn-w001# ssh ncn-w003 'lsmod |grep dvs'
dvs                   405504  0
dvsipc                159744  2 dvs
dvsipc_lnet            57344  1 dvsipc
dvsproc               131072  3 dvsipc,dvsipc_lnet,dvs
craytrace              20480  5 dvsipc,dvsproc,dvsipc_lnet,dvs
dvskatlas              20480  4 dvsipc,dvsproc,dvsipc_lnet,dvs
```

```
lnet                   626688  3 ksocklnd,dvsipc_lnet
libcfs                 512000  3 ksocklnd,lnet,dvsipc_lnet
```

2.10.7 The `cpsmount.sh` Script Times Out

The `cpsmount.sh` script asks the broker via HTTP requests to make a specific file system image available to a node with a particular network file system; the node then mounts that file system. If the script times out, the node will be unable to mount the desired file system.

- Verify that S3 artifacts exist and the s3path is correct in the boot parameters.

  ```
  ncn-w001# cray cps contents list |grep s3path
  s3path = "s3://boot-images/15ab96a1-fefb-4d67-8292-023043aa1546/rootfs"
  s3path = "s3://boot-images/87cabdc2-8919-49a9-b22b-f3d5265cdabf/rootfs"
  s3path = "s3://boot-images/cray-l1diag-sles15-1.0.0.x86_64.squashfs"
  s3path = "s3://boot-images/Analytics/Cray-Analytics.x86_64-base.squashfs"
  ```

- Check the cps transports list command output and verify that the readyForMount = true.

  ```
  cps transports list --transport dvs --s3path FULL_S3PATH
  ERROR = []
  artifactID = "76df050e1fde782a58365504477a7af6"
  [[transports]]
  exportPath = "/var/lib/cps-local/76df050e1fde782a58365504477a7af6"
  readyForMount = true
  servers = [ "10.252.0.5", "10.252.0.6",]
  ready = 2
  total = 2
  type = "dvs"
  [[transports.status]]
  status = "ready"
  replicaID = "10.252.0.5"
  detail = "76df050e1fde782a58365504477a7af6-dvs enabled"

  [[transports.status]]
  status = "ready"
  replicaID = "10.252.0.6"
  detail = "76df050e1fde782a58365504477a7af6-dvs enabled"
  ```

  In order for readyForMount to become true, there must be more than the minimum replica percent of transports that to be ready. Check the `cm-pm` pod status to make sure all are up and running.

  ```
  ncn-w001# kubectl get pod -A |grep cray-cps
  services   cray-cps-cm-pm-7ht8b                    5/5     Running   0  4h46m
  services   cray-cps-cm-pm-lmt7s                    5/5     Running   0  3h38m
  services   cray-cps-cm-pm-x5vr8                    5/5     Running   0  4h46m
  services   cray-cps-cray-jobs-cray-cps-job-2-pfnbf 0/2     Completed 0  3h38m
  services   cray-cps-d97d9699d-f8rq4                2/2     Running   0  4h49m
  services   cray-cps-d97d9699d-n29p6                2/2     Running   0  4h49m
  services   cray-cps-etcd-dxb4bzl526                1/1     Running   0  4h48m
  services   cray-cps-etcd-p67c65rtr4                1/1     Running   0  4h49m
  services   cray-cps-etcd-r4gwmnvf28                1/1     Running   0  4h47m
  services   cray-cps-wait-for-etcd-2-75djv          0/1     Completed 0  3h38m
  ```

2.10.8 The `craycps-s3` Dracut Fails

The `craycps-s3` dracut is a script session that normally runs while a node is booting.

Look for the following issues in the log outputs when a boot fails:

- Check the network.
- Check that auth is setup correctly.
- Check the status of DVS.

- Check to see if the image s3path is correct.

3 Node Memory Dump Service

3.1 Dump a Compute Node with Node Memory Dump (NMD)

This section describes how to generate a remote node memory dump using nmd to help with debugging compute node crashes.

- The Boot Script Service (BSS) is running in one or more pods in the Kubernetes cluster.
- The `crashkernel=360M` boot parameter has been added to BSS for the compute node.
- The `kdump` service customized for the NMD is enabled in the compute node image. It is enabled while the image is getting customized via the Configuration Framework Service (CFS).
- Cray kdump helper (ckdump-helper) for NMD is enabled.
- A compute node has crashed or an admin has triggered a node crash.
- 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 CSM documentation for more information.
- **ROLE**

  System administrator

- **OBJECTIVE**

  This procedure describes how to generate a remote node memory dump using the `nmd` command to help debug compute node crashes. Note that before nmd can be used on a compute node that has crashed, the node's dump-capture kernel must be booted.

- **LIMITATIONS**

  Limitations inherent to kdump that are not imposed by HPE:

  - In order to reboot into dump-capture kernel, dump-capture kernel needs to be loaded into memory by the kdump service with nmd's support, while the node is initially booting. Therefore, the node will not be in a dumpable state if the node has not booted far enough and the kdump service has not yet started up.
  - Because kdump has to boot into a dump-capture kernel, it cannot do a live node dump.

The NMD service includes the following features:

- Provides concurrent dump capability.
- Controls automated kdump so that the dump is generated only for the requested nodes.
- Provides a configurable makedumpfile dump level option for the selected node at dump time, which contrasts with kdump, where this is a preconfigured option that cannot be changed when the node goes down.

  Use the dumplevel argument of the makedumpfile command to specify makedumpfile's dump level, which is 31 by default. Use 16 if it is required to retrieve user process core dump (user data pages) or none-private cache pages.

HPE recommends keeping nmd enabled. In cases where this service is not needed, rather than disabling the service, remove the `crashkernel=360M` kernel parameter from BSS for the node(s) in question. This will prevent `kdump` from running and will free up that reserved kernel memory to be used for system RAM.

The `cray nmd` command is used to manage compute node memory dumps on the system. For more information about the NMD Cray CLI commands, use the –help option with any of the `cray nmd` commands.

The expected states of a node memory dump are described below:

- **waiting**

  The node is crashed and the dump capture kernel is booted to run the kdump service.

- **dump**

  A node memory dump was requested and is in the process of getting dumped. While in this state, the progress percentage of the dump will increase.

- **done**

  The node memory dump is finished. This state occurs before the status data in NMD gets deleted.

- **error**

  The node memory dump failed. NMD will be in this state until the next dump is requested or the state gets removed with the cray nmd status delete command.

- **cancel**

  The node memory dump is cancelled via the cray nmd dumps delete –force XNAME command.

1. Ensure that the compute node is ready for nmd to be invoked.

   Choose one of the following, depending on whether this is an unexpected node crash or a node crash is to be triggered for testing or other purpose. In this procedure, replace *XNAME* with the xname of the node.

   - To determine the readiness of a node that has unexpectedly crashed, run the following command:

     ```
     ncn-w001# cray nmd status describe XNAME
     hbrate = 120
     progress = 0
     state = "waiting"
     timestamp = "2020-03-02T17:25:53.253230-06:00"
     heartbeat = "2020-03-02T23:25:53.268386+00:00"
     bosid = "NA"
     endpoint_url = "http://rgw.local:8080"
     ```

     Look at the state field in the returned output to determine the readiness of the node. Refer to the introductory text above for more information on the meaning of each state.

     The node memory dump is done when the kdump service finishes. The node dump status entry gets removed and starts showing the following message:

     ```
     ncn-w001# cray nmd status describe XNAME
     Usage: cray nmd status describe [OPTIONS] XNAME
     Try 'cray nmd status describe --help' for help.
     Error: Data not found: status_xname_get:
     xname=x3000c0s19b3n0 not found
     ```

   - To trigger a node crash and then determine its readiness for nmd, set up a serial-over-LAN connection to the node, trigger a crash, and then monitor its console log. For more information about setting up a serial-over-LAN connection to the node, refer to “Log in to a Node Using ConMan” in the HPE CSM documentation.

     Execute the following to trigger a node crash:

     ```
     x0c0s18b0n0# echo c >/proc/sysrq-trigger
     ```

     The following key sequences trigger a crash dump for non-responsive nodes:

     - Use either the ConMan or ipmitool key sequences for Air Cooled NCNs and compute nodes:

       ```
       conman: &B<action>
       ipmitool: ~~B<action>
       ```

       The variable in the commands above represents one of the many SysRq options.

     - Connect through the node controller (nC) serial console to trigger a crash dump for Liquid Cooled compute nodes:

       ```
       ctrl-a ctrl-b <action>
       ```

2. Request a node dump.

   ```
   ncn-w001# cray nmd dumps create --xname XNAME
   requestID = "bf6afadf-5cdf-11ea-b9d2-76b3ec213338"

   [info]
   created = "2020-03-02T23:44:24.714431+00:00"
   ```

```
image = "NA"
bosid = "NA"
state = "dump"
requestid = "bf6afadf-5cdf-11ea-b9d2-76b3ec213338"
objectkeys = [ "2020/03/02/23/44/24/
bf6afadf-5cdf-11ea-b9d2-76b3ec213338/x3000c0s7b0n0/
x3000c0s7b0n0-bf6afadf-5cdf-11ea-b9d2-76b3ec213338.dump-config", "2020/
03/02/23/44/24/bf6afadf-5cdf-11ea-b9d2-76b3ec213338/x3000c0s7b0n0/
x3000c0s7b0n0-bf6afadf-5cdf-11ea-b9d2-76b3ec213338.dump-info",]
xname = "x3000c0s7b0n0"
```

Note the value of requestID. The request ID can be retrieved from the request output. Use the returned request ID to retrieve the current dump state.

Troubleshooting information:

- Normally, the dump-capture boot process takes about 1-2 minutes after the node crashes. If the dump-capture kernel did not boot for some reason, or the dump-capture is not fully booted, the node may not be in a dumpable state. When this happens, the system will return an error similar to the following:

  ```
  ncn-w001# cray nmd dumps create --xname x0c0s21b0n0 --sysrestart halt
  Usage: cray nmd dumps create [OPTIONS]
  Try "cray nmd dumps create --help" for help.Error:

  Error: Dump State Error: dumps_post req_data=

  {'dumplevel': 31, 'requestid': '303f2217-5ce0-11ea-9b2f-76b3ec213338', 'sysrestart': 'halt',
  'xname': 'x3000c0s7b0n0'}
  failed: request ID is 25d39f21-5ce0-11ea-8119-76b3ec213338 and dump state is dump
  ```

  If this error is returned, make sure that the dump-capture kernel has fully booted.

- At times, the node may crash and go into a state that prevents it from booting into the dump-capture kernel. If this happens, check the console output to make sure that the node rebooted into the dump-capture kernel. For more information, refer to "Access Compute Node Logs" in the CSM documentation. The following output indicates that the node has successfully booted and the node memory dump is ready to run:

  ```
  API Gateway is https://api-gw-service-nmn.local
  xname is x3000c0s7b0n0
  kdump starting
  ```

3. Request a dump status using the request ID retrieved.

   In the following examples, it is assumed that the request ID is `413d8311-c6bf-4a7e-9e36-0364e660f7b3`.

   ```
   ncn-w001# cray nmd dumps describe REQUEST_ID
   requestID = "f74f6b54-5ce0-11ea-a861-76b3ec213338"

   [info]
   created = "2020-03-02T23:53:07.982937+00:00"
   image = "NA"
   bosid = "NA"
   state = "dump"
   requestid = "f74f6b54-5ce0-11ea-a861-76b3ec213338"
   objectkeys = [ "2020/03/02/23/53/07/
   f74f6b54-5ce0-11ea-a861-76b3ec213338/x3000c0s7b0n0/
   x3000c0s7b0n0-f74f6b54-5ce0-11ea-a861-76b3ec213338.dump-config", "2020/
   03/02/23/53/07/f74f6b54-5ce0-11ea-a861-76b3ec213338/x3000c0s7b0n0/
   x3000c0s7b0n0-f74f6b54-5ce0-11ea-a861-76b3ec213338.dump-info",]
   xname = "x3000c0s7b0n0"
   ```

4. Wait for the dump state to change from `dump` to `done`.

5. Verify that the state has changed to `done`.

```
ncn-w001# cray nmd dumps describe REQUEST_ID
requestID = "f74f6b54-5ce0-11ea-a861-76b3ec213338"

[info]
created = "2020-03-02T23:53:07.982937+00:00"
image = "NA"
bosid = "NA"
state = "done"
requestid = "f74f6b54-5ce0-11ea-a861-76b3ec213338"
objectkeys = [ "2020/03/02/23/53/07/
f74f6b54-5ce0-11ea-a861-76b3ec213338/x3000c0s7b0n0/
x3000c0s7b0n0-f74f6b54-5ce0-11ea-a861-76b3ec213338.dump-config", "2020/
03/02/23/53/07/f74f6b54-5ce0-11ea-a861-76b3ec213338/x3000c0s7b0n0/
x3000c0s7b0n0-f74f6b54-5ce0-11ea-a861-76b3ec213338.dump-info",]
xname = "x3000c0s7b0n0"
```

It can be seen that `state` is updated to `done`.

6. Collect the dump data using the System Dump Utility (SDU).

   The `--start_time` command option can be customized. For example, "-1 day", "-2 hours", or a date/time string can be used. For more information on the SDU command options, use the `sdu --help` command.

   ```
   ncn-m001# sdu --scenario triage --start_time DATE_OR_TIME_STRING --end_time DATE_OR_TIME_STRING \
   --plugin hcqd.dump.callback --plugin obtain.oauth.tokens
   ```

   See "Run a Triage Collection with SDU" in the ***HPE Cray EX System Dump Utility (SDU) Administration Guide*** for more information.

7. Remove the dump if needed.

   a. Retrieve the request ID of the dump.

      ```
      ncn-w001# cray nmd dumps list |grep requestID
      requestid = "7fd43a46-5743-11ea-9f7a-76b3ec213338"
      requestid = "3e72d39f-57b6-11ea-890d-76b3ec213338"
      requestid = "4c1c4da1-57b6-11ea-9823-76b3ec213338"
      requestid = "9b1e21b1-57bc-11ea-a3b5-76b3ec213338"
      requestid = "d6051c44-57c7-11ea-a1f2-76b3ec213338"
      requestid = "f7be74fb-57ce-11ea-a23f-76b3ec213338"
      requestid = "6c34a18a-57d2-11ea-99f7-76b3ec213338"
      requestid = "e73cb179-589b-11ea-a91b-76b3ec213338"
      requestid = "de9d039c-589c-11ea-a04c-76b3ec213338"
      requestid = "bf6afadf-5cdf-11ea-b9d2-76b3ec213338"
      ...
      ```

   b. Delete the dump, specifying the request ID.

      The delete command will list all of the objects deleted as a result of the dump being deleted. An ongoing dump can be canceled by appending the –force true option to the end of the delete command. An ongoing dump cancelled with this command will be deleted.

      ```
      ncn-w001# cray nmd dumps delete REQUEST_ID
      [[Deleted]]
      Key = "2020/03/02/23/53/07/f74f6b54-5ce0-11ea-a861-76b3ec213338/
      x3000c0s7b0n0/x3000c0s7b0n0-f74f6b54-5ce0-11ea-a861-76b3ec213338.
      dump-config"

      [[Deleted]]
      Key = "2020/03/02/23/53/07/f74f6b54-5ce0-11ea-a861-76b3ec213338/
      x3000c0s7b0n0/x3000c0s7b0n0-f74f6b54-5ce0-11ea-a861-76b3ec213338.
      dump-info"

      [ResponseMetadata]
      ```

```
HostId = ""
RetryAttempts = 0
HTTPStatusCode = 200
RequestId = "tx00000000000000004a188-005e5d9e3d-119e-default"

[ResponseMetadata.HTTPHeaders]
transfer-encoding = "chunked"
server = "envoy"
x-envoy-upstream-service-time = "2"
x-amz-request-id = "tx00000000000000004a188-005e5d9e3d-119e-default"
date = "Tue, 03 Mar 2020 00:01:00 GMT"
content-type = "application/xml"
```

If you encounter issues running this procedure, see Run a Manual ckdump on Compute Nodes section to run `ckdump` manually.

3.2 Run a Manual ckdump on Compute Nodes

Use the ckdump script to get data from a crashed node.

- The crashed node is running ckdump_helper version 2.7.0 or newer.
- **ROLE**

  System administrator

- **OBJECTIVE**

  If a crashed node is not recognized by NMD, but the node is crashed and the network is initialized, run ckdump manually to upload the node dump.

  The crashed node requires a working spire (JWT) token to communicate with NMD. However, sometimes the JWT token expires or is otherwise invalid. When the JWT token is not valid, the node cannot access NMD through the API gateway and cannot be dumped using NMD. When this happens, assuming that the network for the node is working, `ckdump-helper-script` can be used to set data manually with the serial console to capture a node dump. The dump taken with this manual procedure will show up when running `cray nmd dumps describe <RequestId>`, and the System Dump Utility (SDU) can collect the manual dump along with the other NMD dumps.

1. Verify that NMD cannot communicate with the crashed node.

   Even though the node has crashed, look for the `Error: Data not found...` response to verify NMD cannot communicate with the node.

   In the following command, replace `x0000c0s00b0nN` with the xname of the node being dumped.

   ```
   ncn-m001# cray nmd status describe x0000c0s00b0nN
   ...
   Error: Data not found: status_xname_get: xname=x0000c0s00b0nN not found
   ```

2. Verify that `ckdump-helper-script` is running on the node to be dumped, and terminate it.

   Connect to the node console. See the **Log in to a Node Using ConMan** section in the CSM documentation for details on how to access the node's Serial Over LAN (SOL) console.

   Check the `/var/log/conman/console.\<xname\>` log file from within the `cray-console-node-\<N\>` pod to ensure the dump capture boot is running kdump and that `ckdump-helper-script` is running. There should be `Starting save kernel crash dump...` and `Running /usr/sbin/ckdump-helper-script` output near the end of the log file.

   ```
   cray-console-node-0:/# cd /var/log/conman
   cray-console-node-0:/var/log/conman# grep -E "save kernel crash|ckdump-helper" \
     console.x0000c0s00b0nN | tail -n2
   2022-02-18 20:44:30          Starting save kernel crash dump...
   2022-02-18 20:44:30 Running /usr/sbin/ckdump-helper-script
   ```

   Connect to the node console.

   ```
   cray-console-node-0:/var/log/conman# conman -j x0000c0s00b0nN
   ```

Terminate ckdump-helper-script by typing ctrl + c in the ConMan console. This will send SIGINT to terminate ckdump-helper-script and will give the prompt back for running manual dump commands.

```
^C
<ConMan> Console x0000c0s00b0nN#
```

3. Determine how to invoke ckdump-helper-script.

   ckdump-helper-script can be invoked with either -s3-manual or -token-manual arguments.

   - -s3-manual uses s3 json input to invoke a manual dump. Use this option if xname validation is enabled for the NMD API request.
   - -token-manual uses JWT input and waits for NMD to trigger the dump. Use this option if xname validation is disabled for the NMD API request. The JWT token used can be obtained from a functional compute node.

   Execute the following command to determine if xname validation is enabled:

   ```
   ncn-m001# if kubectl get cm -n opa opa-policy-ingressgateway -o json | \
   jq -r '.data."policy.rego"' | grep -q 'parsed_spire_token.xname'
   then
     echo "xname validation is enabled"
   else
     echo "xname validation is disabled"
   fi
   ```

   The result will determine whether you follow the instructions in step 5 or in step 6.

4. Optionally run ckdump-helper-script with -s3-manual is xname validation is enabled.

   a. Copy the ckdump-helper-script help output displayed for -s3-manual invocations as shown in the following example.

      ```
      <ConMan> Console x0000c0s00b0nN# /usr/sbin/ckdump-helper-script -h
      /usr/sbin/ckdump-helper-script [-h] | [-s3-manual|-token-manual] [dump_level]
      ...
      For s3-manual, use the following or equivalent to generate json input data
      on NCN node:
      echo '{"AccessKey":"'$(kubectl get secrets -o yaml nmd-s3-credentials ...
      "RequestId":"'$(uuidgen)'"}'
      ...
      ```

   b. Paste the echo command from the previous step and run it on the NCN. The echo command generates a valid single line json string that will be used in the next step.

      ```
      ncn-m001# echo '{"AccessKey":"'$(kubectl get secrets -o yaml nmd-s3-credentials ...
      "RequestId":"'$(uuidgen)'"}'
      ```

   c. Run /usr/sbin/ckdump-helper-script -s3-manual in ConMan, then copy and paste the echo command output from the previous step at the Enter s3 json data: prompt. This starts the node memory dump.

      ```
      <ConMan> Console x0000c0s00b0nN# /usr/sbin/ckdump-helper-script -s3-manual
      Enter s3 json data:
      ```

5. Optionally run ckdump-helper-script with -token-manual if xname validation is disabled.

   a. Log into any functional compute node, run following commands to acquire a JWT token, and log out of the compute node.

      ```
      compute# export SPIRE_AGENT_PATH=/usr/bin/ckdump-spire-agent
      compute# /opt/cray/auth-utils/bin/get-auth-token
      ```

   b. Run /usr/sbin/ckdump-helper-script -token-manual in ConMan, then copy and paste the JWT token from the previous step at the Enter JWT: prompt.

      ```
      <ConMan> Console x0000c0s00b0nN# /usr/sbin/ckdump-helper-script -token-manual
      Enter JWT:
      ```

   c. Invoke NMD from the NCN to start the node memory dump. Replace x0000c0s00b0nN with the correct xname.

      ```
      ncn-m001# cray nmd dumps create --xname x0000c0s00b0nN
      ```

6. Query the dump status to ensure the above steps were successful.

   ```
   ncn-m001# cray nmd dumps describe <RequestId>
   ```

   If `-s3-manual` was used, substitute `<RequestId>` with the value used at the `Enter s3 json data:` prompt in the previous step. For `-token-manual`, use the request ID from the `cray nmd dumps create` command output from the step 6 above.

7. Exit the connection to the console with the **&.** command.

   Exit the ConMan pod.

   ```
   &.
   <ConMan> Console x0000c0s00b0nN# exit
   ncn-m001#
   ```

4 Data Virtualization Service

4.1 Introduction to DVS

How the Data Virtualization Service (DVS) distributes images and data to diskless client nodes external file systems in an HPE Cray EX system.

The Data Virtualization Service (DVS) is a distributed network service that projects file systems mounted on non-compute nodes (NCN) to other nodes within the HPE Cray EX system. Projecting is simply the process of making a file system available on nodes where it does not physically reside. DVS-specific configuration settings enable clients to access a file system projected by DVS servers. These clients include compute nodes, User Access Nodes (UANs), and other NCNs running User Access Instances (UAIs). Thus DVS, while not a file system, represents a software layer that provides scalable transport for file system services.

DVS is integrated with the Content Projection Service (CPS). CPS is a service that enables nodes without local disk to mount file systems that are created as image artifacts over a network. Together, DVS and CPS form a convenient infrastructure for mounting root file systems, the Cray Programming Environment (PE), and other sets of mostly static data for diskless compute nodes in an HPE Cray EX system. DVS serves as the transport that moves file system data between the CPS Projection Manager and the CPS clients. The CPS is a Kubernetes pod running on an NCN and CPS clients are usually CNs.

DVS, in turn, uses Lustre™ Networking (LNet) to communicate over the network. LNet configuration is done by the code that configures DVS.

4.1.1 DVS Use Cases

DVS and the Content Projection Service (CPS) play an important role in the HPE Cray EX system:

- During system boot, the compute node and UAN root file system can be mounted as a DVS mount by CPS. Refer to the CSM documentation for instructions on how to create a session template to boot nodes with CPS.
- After system boot, DVS is used by CPS to distribute the HPE Cray Programming Environment (PE) to compute nodes, UANs, and NCNs.
- DVS can also be used to project external file systems to compute nodes and UANs within the HPE Cray EX system. In this use case, DVS operates independently of CPS.

System administrators must enable NCN Personalization to configure DVS on NCNs. Enabling NCN Personalization is normally done as part of the HPE Cray EX software installation. See “Enable NCN Personalization” in the CSM documentation. After NCN Personalization is enabled and configured to apply the Cray Operating System (COS) product, no further configuration is necessary when DVS and CPS are used to project the root file system. CPS can also project the Cray PE if the Cray PE product is included in NCN Personalization.

DVS can project CPS content and an external file system on the same HPE Cray EX system at the same time.

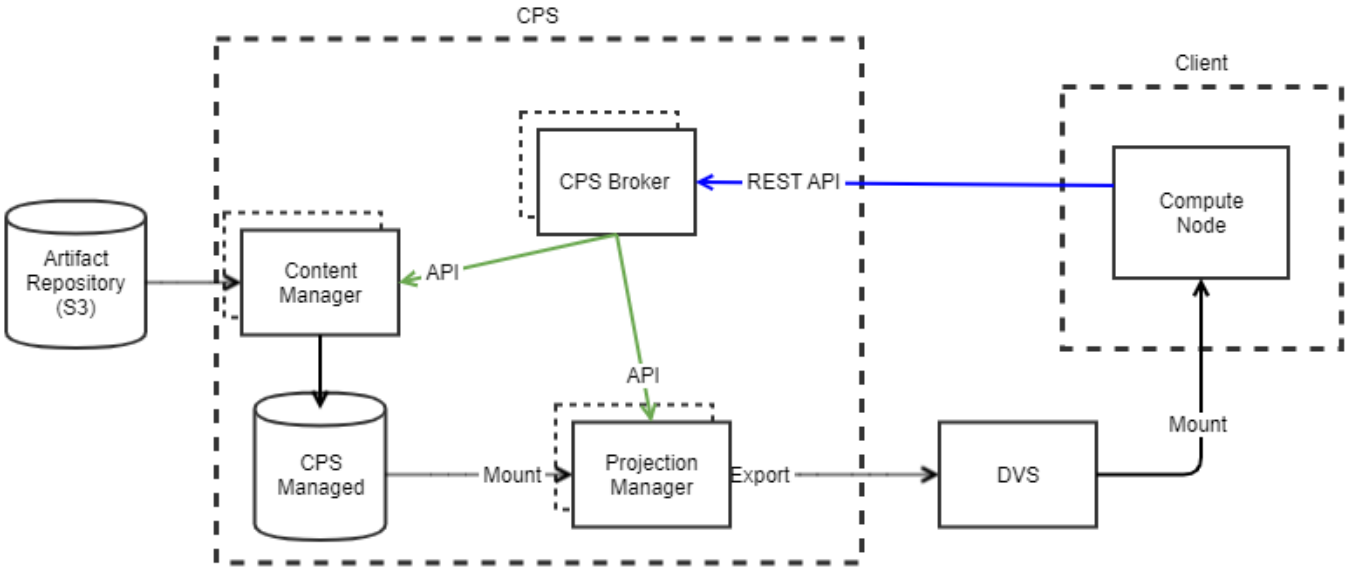
4.1.2 DVS Startup

DVS is loaded and started by NCN Personalization. DVS then works together with CPS to project static content (such as root file systems) to CNs and other DVS clients.

DVS can also project content independently of CPS. This “standalone” DVS operation is used for projecting read/write file systems or external file systems such as Lustre or IBM Spectrum Scale to HPE Cray EX nodes.

4.1.3 DVS Projection for CPS

The following figure illustrates how CPS uses both DVS and the artifact repository to distribute images to compute nodes. These images are stored as SquashFS images in the artifact repository. When a node needs a specific image, a process calls the `cpsmount.sh` script and provides an artifact identifier and a local directory mount point. The script then communicates with the CPS Broker. The Broker asks the CPS Content Manager to download the artifact from the repository and store it on the local disk of the NCN. The Broker then sends to the compute node the name of the CPS-CM-PM/DVS host to mount from. Finally, the `cpsmount.sh` script performs a DVS mount of directory containing the SquashFS image and mounts the image to access the content inside.



DVS needs further configuration only for performance tuning or if a user must project external file systems to nodes within the HPE Cray EX system. DVS configuration parameters enable system administrators to provide their users with client mounts. These parameters can then be tuned for high performance in a variety of use cases.

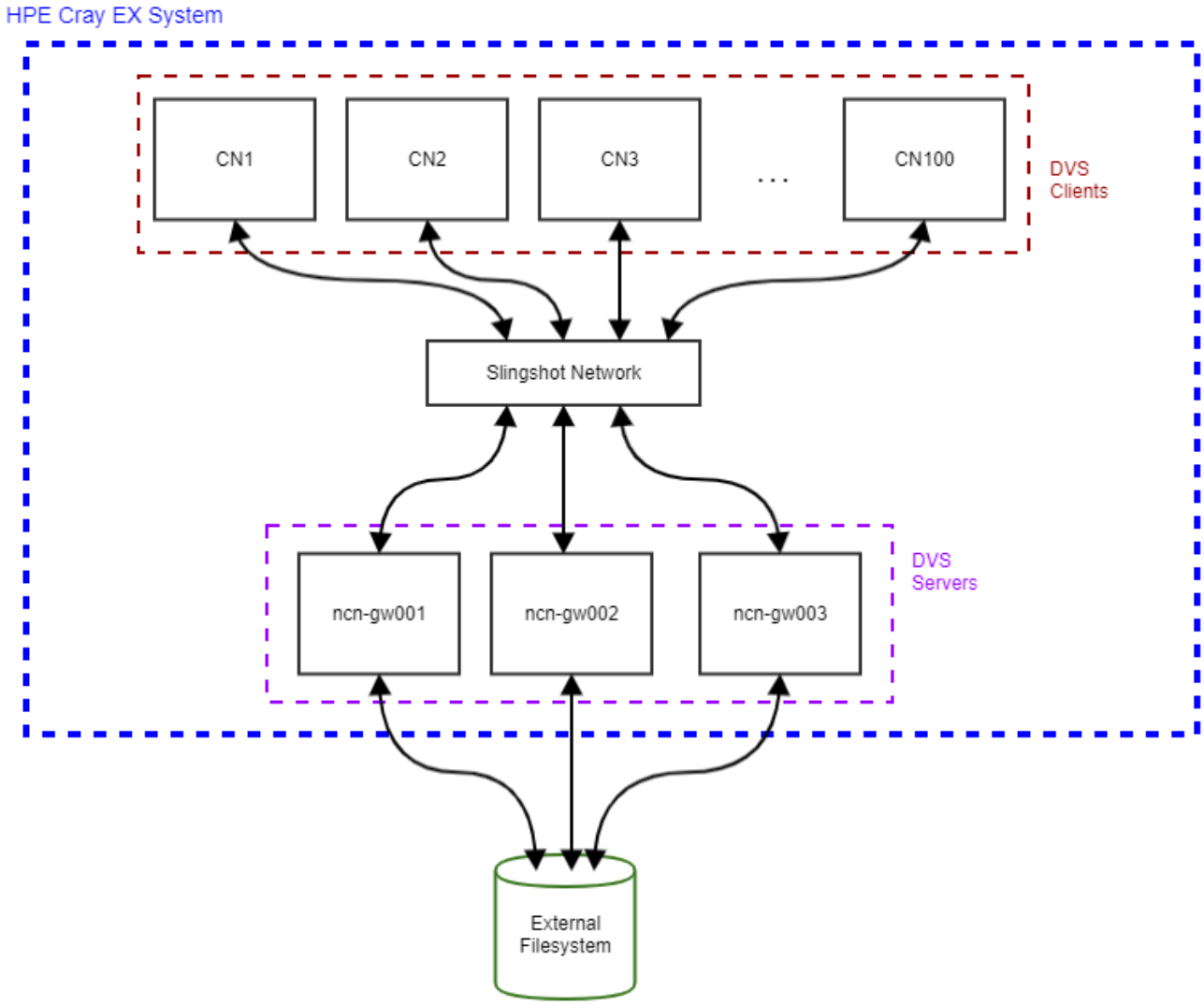
4.1.4 DVS Projection of External Filesystems

When projecting external file systems, DVS provides I/O performance and scalability to many nodes, far beyond the typical number of clients supported by a single NFS server. Operating system noise and impact on compute node memory resources are both minimized in the HPE DVS configuration. DVS uses the Linux-supplied virtual file system (VFS) interface to process file system access operations. This allows DVS to project any POSIX-compliant file system. When DVS is used to project external file systems, DVS operates as illustrated in the following figure. DVS can project file systems such as Spectrum Scale (formerly GPFS), NFS, and Lustre. Refer to Project an External Filesystem to Compute Nodes or User Access Nodes for instructions on how to project external file systems with DVS.

While CPS and DVS work together to project static, read-only content from the NCN Worker nodes, projection of external file systems is different. It's usually read-write data projected from NCN Gateway nodes. These gateway nodes aren't part of the Kubernetes cluster. They are a type of Application node, similar to UANs.

In a future COS release, Gateway nodes will have their own "gateway" sub-role, but for now they may have the "UAN" sub-role. All the Gateways should have their own BOS template because they'll have kernel command line arguments that are different from other Application nodes. They should also have their own branch in the UAN VCS repository. All the Gateways should be a member of the HSM group `dvs_gateways`.

In CSM 1.0, external NICs are not supported for K8s worker nodes (NCN-Ws). Therefore, user file system projection is not supported from worker nodes (NCN-Ws).



4.2 Procedure To Change DVS Network Transport

This procedure documents the steps to change the network transport that DVS uses to talk between nodes. DVS can be configured to use the Node Management Network (NMN) or the High Speed Network (HSN). It can also be configured to use any of a number of different Lustre Network Drivers (LNDs). DVS also supports both Industry Standard and HPE Slingshot NICs.

This procedure will cause service outage for all NCN-GWs (non-compute gateways), CNs, and UANs. Scheduling downtime is recommended when carrying out this procedure.

4.2.1 Prerequisites

1. This procedure assumes that all the nodes in the system are already set up to run correctly with DVS on a transport, whether it is the NMN or the HSN. It focuses on switching to another transport.
2. This procedure assumes familiarity with use of the Ansible group_vars directory for setting variables for different types of node groups.
3. There are no errors reported when the following command is run on one of the NCN workers running DVS.

   ```
   ncn-m001# scp ncn-w001:/opt/cray/dvs/default/sbin/dvs_validate_network /tmp
   ncn-m001# /tmp/dvs_validate_network
   ```

   Keep the following note in mind when running the `dvs_validate_network` script:

   - This script may take a long time to complete on a system with thousands of nodes running.

- The script produces a lot of output
- The end of the output contains a section for errors and warnings
- Errors will cause problems for DVS running on the NMN or HSN on the system.
- Warnings may or may not be problems.
- Warnings could result from underlying errors.
- For example, the most common warning is not being able to SSH into a node. That means the script can't check the IP addresses on the node. The IP addresses could be incorrect, or the node may be down.

4.2.2 Change DVS Network Transport on Worker Nodes (NCN-Ws)

To change the DVS network transport on worker nodes, the following tasks are required:

- Make changes to a few Ansible configuration files for DVS and LNet in the COS configuration.
- Update the CFS COS configuration layer for NCN personalization.
- Unload DVS and LNet services on workers that were running over the old transport.
- Reload DVS and LNet services on workers to run over the new transport.

Perform the steps below to accomplish these tasks.

1. Start a typescript to capture the commands and output from the deployment for troubleshooting if needed.

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

2. Clone the COS configuration management repository using the proper credentials. Checkout the branch currently used to hold COS configuration.

   The output of the first command provides the password required for the clone. The following example assumes the current COS configuration branch is "cos-integration".

   ```
   ncn-m001# kubectl get secret -n services vcs-user-credentials \
   --template={{.data.vcs_password}} | base64 --decode
   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':
   ncn-m001# cd cos-config-management/
   ncn-m001# git checkout cos-integration
   ```

3. Make the appropriate changes to the `group_vars` directory to configure the new network transport.

   DVS and LNet are configured by changing variables in files in the `group_vars` directory of the config management repositories. The relevant variables are:

   - **cray_lnet_mlnx**: This is a boolean variable that describes which set of Lustre RPMs to install. True means that the Industry Standard NIC version of the cray-lnet-client RPMs should be installed. False means that the HPE Slingshot NIC version should be installed.
   - **lnet_conf**: This is a variable that holds the data which will be installed into /etc/lnet.conf. It's the primary way of configure LNet. It specifies which Lustre Network Drivers (LNDs) and which network interfaces will be used.
   - **lnet_lnds**: This variable controls which LNet LNDs to load on Workers. It's a YAML list of LND names. It should contain the LNDs which are configured referenced in the lnet_conf variable for the Workers.
   - **lnet_modprobe_conf**: This variable contains the modprobe variables for the LNet modules.
   - **dvs_modprobe_conf**: This variable contains the modprobe variables for the DVS modules.
   - **dvs_node_map_conf**: This variable contains the JSON config that controls how the DVS node map is created. There should be only one instance of this variable for the whole system, since it needs to be consistent for all nodes.

   Some recipes for common network transports are below.

   These recipes just describe the values of the variables needed for DVS. If Lustre is involved, there may need to be additional values in the lnet-related variables. See the ***Configure Lustre*** chapters of this document for more information.

Also, care must be taken if your VCS has multiple lnet.yml or dvs.yml files. A variable in a group_vars directory other than "all" will override the value in "all". For example, the value of dvs_modprobe_conf in `group_vars/Management_Worker/dvs.yml` will override the value in `group_vars/all/dvs.yml`. If you have multiple files like this, make sure they're doing what you intend.

a. DVS over the Node Management Network (NMN) with ksocklnd

   The value of cray_lnet_mlnx does not matter for this recipe. However, it will make later migration easier if it matches your hardware type. Industry Standard NICs fit best with a "true" value and HPE Slingshot NICs fit best with a "false" value. If your system has a mix of the two, it's best to use a value of "false".

   The file `group_vars/all/lnet.yml` should look like what's below.

   ```
   cray_lnet_mlnx: true

   lnet_conf: |
     ip2nets:
       - net-spec: tcp99
         interfaces:
           0: nmn0

   lnet_modprobe_conf: |
     options lnet lnet_transaction_timeout=120

   lnet_lnds:
   - ksocklnd
   ```

   The file `group_vars/Management_Worker/lnet.yml` should look like what's below.

   ```
   lnet_conf: |
     ip2nets:
       - net-spec: tcp99
         interfaces:
           0: bond0.nmn0
   ```

   The file `group_vars/all/dvs.yml` should look like what's below.

   ```
   dvs_node_map_conf: |
     {
        "config" : [
           {
              "lnet" : "tcp99",
              "nid" : 0,
              "xname" : ""
           }
        ]
     }

   dvs_modprobe_conf: |
     options dvsipc_lnet lnd_name=tcp99
   ```

b. DVS over the High-Speed Network (HSN) with ksocklnd

   The value of cray_lnet_mlnx does not matter for this recipe. However, it will make later migration easier if it matches your hardware type. Industry Standard NICs fit best with a "true" value and HPE Slingshot NICs fit best with a "false" value. If your system has a mix of the two, it's best to use a value of "false".

   If your system has or will have Lustre clients which will also be using ksocklnd, make the "tcp99" network name match the same LNet network which Lustre will use (e.g. tcp, tcp1, etc) in the four spots below.

   The file `group_vars/all/lnet.yml` should look like what's below.

   ```
   cray_lnet_mlnx: true

   lnet_conf: |
   ```

```
  ip2nets:
    - net-spec: tcp99
      interfaces:
        0: hsn0
        1: hsn1
        2: hsn2
        3: hsn3

lnet_modprobe_conf: |
  options lnet lnet_transaction_timeout=120

lnet_lnds:
- ksocklnd
```

The file `group_vars/all/dvs.yml` should look like what's below.

```
dvs_node_map_conf: |
  {
     "config" : [
        {
           "lnet" : "tcp99",
           "nid" : 0,
           "xname" : "h0"
        }
     ]
  }

dvs_modprobe_conf: |
  options dvsipc_lnet lnd_name=tcp99
```

c. DVS over the High-Speed Network (HSN) with ko2iblnd

This is the preferred recipe to be used with Industry Standard (non-HPE Slingshot) NICs.

The value of cray_lnet_mlnx **does** matter for this recipe. It should be "true".

If your system has or will have Lustre clients which will also be using ko2iblnd, make the "o2ib" network name match the same LNet network which Lustre will use (e.g. o2ib1, o2ib2, etc) in the four spots below.

The file `group_vars/all/lnet.yml` should look like what's below.

```
cray_lnet_mlnx: true

lnet_conf: |
  ip2nets:
    - net-spec: o2ib
      interfaces:
        0: hsn0
        1: hsn1
        2: hsn2
        3: hsn3

lnet_modprobe_conf: |
  options lnet lnet_transaction_timeout=120
  options ko2iblnd map_on_demand=1

lnet_lnds:
- ko2iblnd
```

The file `group_vars/all/dvs.yml` should look like what's below.

```
dvs_node_map_conf: |
  {
```

```
    "config" : [
       {
          "lnet" : "o2ib",
          "nid" : 0,
          "xname" : "h0"
       }
    ]
  }

dvs_modprobe_conf: |
  options dvsipc_lnet lnd_name=o2ib
```

d. DVS over the High-Speed Network (HSN) with kkfilnd

In COS 2.2, there is prototype support for running DVS on top of HPE Slingshot NICs with kkfilnd.

This is the preferred recipe to be used with HPE Slingshot NICs.

The value of cray_lnet_mlnx **does** matter for this recipe. It should be “false”.

If your system has or will have Lustre clients which will also be using kkfilnd, make the “kfi” network name match the same LNet network which Lustre will use (e.g. kfi1, kfi2, etc) in the four spots below.

The file `group_vars/all/lnet.yml` should look like what’s below.

```
cray_lnet_mlnx: false

lnet_conf: |
  net:
    - net type: kfi
      local NI(s):
        - interfaces:
            0: cxi0
        - interfaces:
            0: cxi1

lnet_modprobe_conf: |
  options lnet lnet_transaction_timeout=120

lnet_lnds:
- kkfilnd
```

The file `group_vars/all/dvs.yml` should look like what’s below.

```
dvs_node_map_conf: |
  {
     "config" : [
        {
           "lnet" : "kfi",
           "nid" : 0,
           "xname" : "h0"
        }
     ]
  }

dvs_modprobe_conf: |
  options dvsipc_lnet lnd_name=kfi
```

Note: It’s important that each node has all of their cxiX interfaces configured. They should all be in the “lnet_conf” variable for each node. This makes sure that the LNet peer discovery code is able to allow nodes to talk to each other. If only some of the interfaces are configured, some nodes may have problems communicating via DVS.

This means that if your system has different classes of nodes with different numbers of HPE Slingshot NICs, you want to create

different VCS branches. Each branch should have a "lnet_conf" variable that lists the correct number of cxiX interfaces. For Worker nodes, each VCS branch should have its own ncn-personalization CFS configuration which is assigned to the Worker components. For CNs or UANs, each VCS branch should have its own CFS Configurations, rootfs images and BOS templates.

4. Commit and push the changes for the branch to the repository using the proper credentials. The output of the first command provides the password required for the push.

```
ncn-m001# kubectl get secret -n services vcs-user-credentials \
--template={{.data.vcs_password}} | base64 --decode
ncn-m001# git commit -am 'Added COMMIT_COMMENT'
ncn-m001# git push --set-upstream origin cos-integration
Username for 'https://api-gw-service-nmn.local': crayvcs
Password for 'https://crayvcs@api-gw-service-nmn.local':
```

5. Identify the commit hash for this branch, and save the commit hash for later use when updating the CFS COS configuration layer.

```
ncn-m001# git rev-parse --verify HEAD
<== commit hash output ==>
ncn-m001# export COS_CONFIG_COMMIT_HASH=<commit hash output>
```

6. Disable NCN personalization temporarily for each worker node, while the CFS configuration is being updated.

```
ncn-m001# cray cfs components update <NCN_XNAME> --enabled false
```

7. Update the COS config commit hash in the NCN personalization CFS configuration.

    1. Download the current ncn-personalization configuration.

```
ncn-m001# cray cfs configurations describe ncn-personalization \
--format json > ncn-personalization.json
ncn-m001# cat ncn-personalization.json
{
  "lastUpdated": "2021-04-02T15:10:36Z",
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
      "commit": "120a5d2f4acb42203c74ee8920f00c24253247b7",
      "name": "cos-integration",
      "playbook": "ncn.yml"
    },
   ...
  ],
  "name": "ncn-personalization"
}
```

    2. Edit and update the JSON file.

        1. Remove the lastUpdated key.
        2. Remove the second name key that contains the value "ncn-personalization".
        3. Remove the comma after the layers array.
        4. Update the git commit hash for the COS layer with the COS_CONFIG_COMMIT_HASH saved in Step 5, and leave other product stream layers untouched. The JSON file should now resemble the following:

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

```
      ...
    ]
  }
```

3. Upload the new configuration into CFS.

```
ncn-m001# cray cfs configurations update ncn-personalization \
--file ncn-personalization.json --format json
{
  "lastUpdated": "2021-05-19T21:30:09Z",
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
      "commit": "3d5114dd96201ca13463db5b69e192d28d404efc",
      "name": "cos-integration",
      "playbook": "ncn.yml"
    }
   ...
],
  "name": "ncn-personalization"
}
```

8. Copy the `dvs_reload_ncn` script from ncn-w001 to ncn-m001 for use on ncn-m001.

```
ncn-m001# scp ncn-w001:/opt/cray/dvs/default/sbin/dvs_reload_ncn /tmp
```

9. Run the following on each worker node that has local Lustre mounts, PE (Programming Environment) or Analytics contents deployed. This step ensures all local Lustre and DVS mounts on all workers are unmounted, before DVS and LNet services can be reloaded to run over the new transport. Since this deployment procedure precludes any DVS failover, this step can be done on multiple workers in parallel.

   1. Unmount any Lustre mounts on the workers that have Lustre mounted by running this dvs_reload_ncn command. Then wait for the resulting Kubernetes job to complete.

```
ncn-m001# /tmp/dvs_reload_ncn -c ncn-personalization \
-p configure_fs_unload.yml \
<xname_of_ncn-w001 ... xname_of_ncn-wNNN>
```

   2. On each worker, unmount PE and Analytics contents. For all steps below, replace ncn-w001 with the appropriate worker name.

      1. Check to see if any UAIs are running on the worker.

```
ncn-m001# kubectl get pods -Ao wide | grep 'uai.*ncn-w001'
```

         If there are any, delete them with the following command and wait 30 seconds.

```
ncn-m001# cray uas uais delete
```

      2. Unmount PE contents on the worker.

```
ncn-m001# ssh ncn-w001 bash /etc/cray-pe.d/pe_cleanup.sh
```

         If any issues are encountered, refer to the troubleshooting section of the HPE Cray Programming Environment documentation for CSM on HPE Cray EX Systems.

      3. Unmount Analytics contents on the worker.

         The following commands are executed on the master NCN and retrieve the forcecleanup.sh script from the crayvcs repo on the master NCN and copy it to the worker NCN, then execute the forcecleanup.sh script on the worker NCN using remote shell.

```
ncn-m001# export git_pwd=$(kubectl get secret -n services vcs-user-credentials \
--template={{.data.vcs_password}} | base64 --decode)
ncn-m001# export git_url="api-gw-service-nmn.local/vcs/cray/analytics-config-management.git"
ncn-m001# git clone https://crayvcs:"${git_pwd}"@${git_url} >/dev/null 2>&1
ncn-m001# cd analytics-config-management
ncn-m001# git checkout integration
```

```
ncn-m001# scp roles/analyticsdeploy/files/forcecleanup.sh ncn-w001:/tmp
ncn-m001# ssh ncn-w001 'sh /tmp/forcecleanup.sh'
```

Note that sometimes it takes a few runs of the last command (ssh ncn-w001 'sh /tmp/forcecleanup.sh') to unmount the Analytics contents, i.e. until the following command no longer returns any output about Analytics mounts.

```
ncn-m001# ssh ncn-w001 'mount -t dvs | grep -i analytics'
```

3. At this point, the dvs.ko module should have a zero reference count. If the lustre.ko module is loaded, it should also have a zero reference count. If this is not the case, investigate why the unmounts did not complete correctly.

```
ncn-w# lsmod | grep -P '^(dvs|lustre) '
dvs                   425984  0
```

10. Reload DVS and LNet services to run over the new transport. This step can also be done on multiple workers in parallel.

1. If you did not change the value of the cray_lnet_mlnx variable, perform this step.

   Skip this step on Workers that show a non-zero (0) reference count on the dvs.ko or lustre.ko modules. Since they can't be unloaded correctly, the Worker will need to be rebooted in a later step.

   Reload DVS and LNet on all workers by running the following dvs_reload_ncn command. Then wait for the resulting Kubernetes job to complete.

   ```
   ncn-m001# /tmp/dvs_reload_ncn -c ncn-personalization \
   -p cray_dvs_reload.yml \
   <xname_of_ncn-w001 ... xname_of_ncn-wNNN>
   ```

2. If you did change the value of cray_lnet_mlnx, perform this step.

   Skip this step on Workers that show a non-zero (0) reference count on the dvs.ko or lustre.ko modules. Since they can't be unloaded correctly, the Worker will need to be rebooted in a later step.

   a. Unload DVS and LNet on all workers by running the following dvs_reload_ncn command. Then wait for the resulting Kubernetes job to complete.

      ```
      ncn-m001# /tmp/dvs_reload_ncn -c ncn-personalization \
      -p cray_dvs_unload.yml \
      <xname_of_ncn-w001 ... xname_of_ncn-wNNN>
      ```

   b. On all Workers, uninstall the DVS and Lustre RPMs. Then verify that the RPMs are no longer installed on the node.

      ```
      ncn-m001# ssh ncn-w001
      ncn-w001# rpm -qa | grep -P 'dvs|lustre' | xargs rpm -ev
      ...
      ncn-w001# rpm -qa | grep -P 'dvs|lustre'
      ncn-w001#
      ncn-w001# exit
      ```

   c. Install the new Lustre RPMs on all workers by running the following dvs_reload_ncn command. Then wait for the resulting Kubernetes job to complete.

      ```
      ncn-m001# /tmp/dvs_reload_ncn -c ncn-personalization \
      -p cray_lnet_install.yml \
      <xname_of_ncn-w001 ... xname_of_ncn-wNNN>
      ```

   d. Install the DVS RPMs on all workers by running the following dvs_reload_ncn command. Then wait for the resulting Kubernetes job to complete.

      ```
      ncn-m001# /tmp/dvs_reload_ncn -c ncn-personalization \
      -p cray_dvs_install.yml \
      <xname_of_ncn-w001 ... xname_of_ncn-wNNN>
      ```

   e. Reload DVS and LNet on all workers by running the following dvs_reload_ncn command. Then wait for the resulting Kubernetes job to complete.

```
ncn-m001# /tmp/dvs_reload_ncn -c ncn-personalization \
-p cray_dvs_load.yml \
<xname_of_ncn-w001 ... xname_of_ncn-wNNN>
```

3. Verify that DVS and LNet services are running over the new transport.

   Skip this step on Workers that show a non-zero (0) reference count on the dvs.ko or lustre.ko modules. Since they can't be unloaded correctly, the Worker will need to be rebooted in a later step.

   In our example below, the HSN uses ksocklnd, where DVS is supposed to use LNet network "tcp" on interface "hsn0" for NCN workers:

```
ncn-m001# pdsh -w ncn-w[001-NNN] lsmod | grep -P '\bdvs\b' | dshbak -c
----------------
ncn-w[001-003]
----------------
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
ncn-m001# pdsh -w ncn-w[001-NNN] lnetctl net show | grep -B 5 'hsn0' | dshbak -c
----------------
ncn-w001
----------------
    - net type: tcp
      local NI(s):
        - nid: 10.253.0.6@tcp
          status: up
          interfaces:
              0: hsn0
----------------
ncn-w002
----------------
    - net type: tcp
      local NI(s):
        - nid: 10.253.0.11@tcp
          status: up
          interfaces:
              0: hsn0
----------------
ncn-w003
----------------
    - net type: tcp
      local NI(s):
        - nid: 10.253.0.24@tcp
          status: up
          interfaces:
              0: hsn0
```

4. Restart the CPS cm-pm pods so they see the new network transport. Then wait until the pods are in the "running" state.

```
ncn-m001# kubectl delete pods -n services -l \
app.kubernetes.io/name=cm-pm
ncn-m001# kubectl get pods -A | grep cps-cm-pm
```

5. Re-enable and rerun NCN personalization for all workers.

```
ncn-m001# cray cfs components update --enabled true --state '[]' \
--error-count 0 <NCN_XNAME> --format json
```

6. To reload DVS and LNet services on the workers that previously had a non-zero reference count for the dvs.ko or lustre.ko

modules, reboot the worker node. Follow the instructions in the “Reboot NCNs” procedure in the CSM documentation, to reboot the worker.

7. Wait for ncn-personalization to be in the “configured” configuration status for each worker node.

   ```
   ncn-m001# cray cfs components describe --format json <NCN_XNAME>
   ```

8. Recheck that DVS and LNet services are running on all workers on the new transport.

   ```
   ncn-m001# pdsh -w ncn-w[001-NNN] lsmod | grep -P '\bdvs\b' | dshbak -c
   ncn-m001# pdsh -w ncn-w[001-NNN] lnetctl net show | grep -B 5 'hsn0' | dshbak -c
   ```

4.2.3 Change DVS Network Transport on Gateway Nodes (NCN-GWs)

A NCN-GW is a new type of node available starting with HPE Cray EX v1.5. The NCN-GW is solely used to project external file systems to the CNs and UANs. The NCN-GW is a special type of Application node that uses the same VCS repo from the UAN configuration management repository. Howerver, the NCN-GW has its own configuration branch and BOS session template. Refer to “Project an External Filesystem to Compute Nodes or User Access Nodes” section for more details on the gateway nodes.

Check whether there are any NCN-GWs configured on the sytem, using the example shown below. If there are, continuing with the procedure in this section. Else, move to the next section “Change DVS Network Transport on Compute Nodes (CNs)”.

```
ncn-m001# cray hsm groups describe --format json dvs_gateways
{
  "label": "dvs_gateways",
  "description": "DVS Gateways",
  "members": {
    "ids": [
      "x3000c0s27b0n0"
    ]
  }
}
```

To change the DVS network transport on NCN-GWs, the following tasks are required:

- Make changes to a few Ansible configuration files for DVS and LNet in the UAN configuration.
- Update the CFS UAN configuration layer for the NCN-GWs.
- Create new NCN-GW images with the updated CFS configuration.
- Update the BOS template used for booting NCN-GWs to use the new NCN-GW images.
- Reboot the NCN-GWs.

Perform the steps below to accomplish these tasks.

1. Clone the UAN configuration management repository. Checkout the branch currently used to hold NCN-GW configuration. The following example assumes that branch is “integration-gateways”.

   ```
   ncn-m001# git clone https://api-gw-service-nmn.local/vcs/cray/uan-config-management.git
   ncn-m001# cd uan-config-management/
   ncn-m001# git checkout integration-gateways
   ```

2. Make the same set of changes to configure the DVS network transport which you made during step 3 of the previous section in this document titled “Change DVS Network Transport on Worker Nodes (NCN-Ws)”.

3. Commit and push the changes for the branch to the repository using the proper credentials. Use the output of the first command as the password when prompted.

   ```
   ncn-m001# kubectl get secret -n services vcs-user-credentials \
   --template={{.data.vcs_password}} | base64 --decode
   ncn-m001# git commit -am 'Added COMMIT_COMMENT'
   ncn-m001# git push --set-upstream origin integration-gateways
   Username for 'https://api-gw-service-nmn.local': crayvcs
   Password for 'https://crayvcs@api-gw-service-nmn.local':
   ```

4. Identify the commit hash for this branch, and save the commit hash for later use when updating the CFS UAN configuration layer for the NCN-GWs.

```
ncn-m001# git rev-parse --verify HEAD
<== commit hash output ==>
```

5. Update any CFS configurations used by the NCN-GWs with the commit ID obtained in the previous step.
6. Create new NCN-GW boot images and update the NCN-GW BOS template. Refer to the "Project an External Filesystem to Compute Nodes or User Access Nodes" section, and UAN product stream documentation for details.
    1. When creating a CFS session to perform pre-boot image customization of the NCN-GW image, be sure to specify the CFS configuration used by the NCN-GWs from Step 5.
    2. When preparing the NCN-GW BOS template, be sure to set the "rootfs_provider_passthrough" for the correct networks. When booting over the NMN, the section before the last colon should just be "nmn0". When booting over the HSN, that section should be "hsn0,nmn0", as shown below.

        ```
        "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:hsn0,nmn0:0"
        ```
7. Reboot the NCN-GWs. Replace UUID in the following command with ID of the BOS session template that uses the new NCN-GW images.

    ```
    ncn-m001# cray bos session create --template-uuid UUID --operation reboot
    ```
8. Verify that DVS and LNet services are running over the correct transport. In our example below, the HSN uses ksocklnd, where DVS is supposed to use LNet network "tcp" on interface "hsn0" for the NCN-GWs:

    ```
    ncn-m001# ssh ncn-g001 -o StrictHostKeyChecking=no -i /tmp/supersecret 'lnetctl net show' | \
    grep -B5 'hsn0'
        - net type: tcp
          local NI(s):
            - nid: 10.253.0.27@tcp
              status: up
              interfaces:
                  0: hsn0
    ```

4.2.4 Change DVS Network Transport on Compute Nodes (CNs)

To change the DVS network transport on CNs, the following tasks are required:

- Update the CFS COS configuration layer for the CNs.
- Run an IMS Image Customization session with the updated CFS configuration to create new CN images.
- Update the BOS template used for booting CNs to use the new CN images.
- Reboot the CNs.

Perform the steps below to accomplish these tasks.

1. Update the COS config commit hash in the CFS configuration for the CNs to the same COS_CONFIG_COMMIT_HASH (Step 5 of the "Change DVS Network Transport on Worker Nodes (NCN-Ws)" section) used in the NCN personalization CFS configuration.
    1. Obtain the name of the current CFS configuration for the CNs by using the following methods. Skip this step if the name of the current CFS configuration for the CNs is already known.

        Each BOS session template has a section indicating which CFS configuration is used.

        1. List all the BOS session templates using the following command.

            ```
            ncn-m001# cray bos sessiontemplate list --format json
            ```
        2. Print out the current BOS session template for the CNs. The template name is "cos-sessiontemplate-2.X.38" in the following command example.

        The example output shows that the current CFS configuration for CNs is "cos-config-2.X.38". Most of the output of the following example command has been omitted for clarity

        ```
        ncn-m001# cray bos sessiontemplate describe --format json cos-sessiontemplate-2.X.38
        ```

        ```
        {
          "boot_sets": {
            "compute":
        ```

```
      . . .
    },
    "cfs": {
      "configuration": "cos-config-2.X.38"
    },
    "enable_cfs": true,
    "name": "cos-sessiontemplate-2.X.38"
  }
```

2. Download the current CN configuration. From now on for examples, “cos-config-2.X.38” is used as a current CN configuration to boot over NMN, and “cos-config-2.X.38-hsn” is a configuration for HSN.

   ```
   ncn-m001# cray cfs configurations describe cos-config-2.X.38 --format json \
   > cos-config-2.X.38-hsn.json
   ```

3. Make the same changes to the CN configuration JSON file “cos-config-2.X.38-hsn.json” that were made to the NCN file (Step 7.2 of the “Change DVS Network Transport on Worker Nodes (NCN-Ws)” section).

4. Upload the new CN configuration into CFS.

   ```
   ncn-m001# cray cfs configurations update cos-config-2.X.38-hsn \
   --file cos-config-2.X.38-hsn.json --format json
   ```

2. Run an IMS Image Customization session to create new CN images. Use the same CN configuration name in the previous step for booting over the new transport and the procedure documented in the “Create an Image Customization CFS Session” section of the CSM product stream documentation.

3. Update the BOS template used for booting CNs to use the new CN images. Refer to the “Boot Orchestration Service (BOS)” section of the CSM product stream documentation on how to update a BOS template. Be sure to set the “rootfs_provider_passthrough” for the correct networks. When booting over the NMN, the section before the last colon should just be “nmn0”. When booting over the HSN, that section should be “hsn0,nmn0”, as shown below.

   ```
   "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:hsn0,nmn0:0"
   ```

4. Reboot the CNs. Replace UUID in the following command with ID of the BOS session template that uses the new CN images.

   ```
   ncn-m001# cray bos session create --template-uuid UUID --operation reboot
   ```

5. Verify that DVS and LNet services are running over the new transport. In our example below, the HSN uses ksocklnd, where DVS is supposed to use LNet network “tcp” on interface “hsn0” for CNs:

   ```
   ncn-m001# PDSH_SSH_ARGS='-o StrictHostKeyChecking=no' pdsh -w nid[000001-000004]-nmn \
   lnetctl net show | grep -B5 'hsn0' | dshbak -c
   ----------------
   nid000001-nmn
   ----------------
       - net type: tcp
         local NI(s):
           - nid: 10.253.0.14@tcp
             status: up
             interfaces:
                 0: hsn0
   ----------------
   nid000003-nmn
   ----------------
       - net type: tcp
         local NI(s):
           - nid: 10.253.0.23@tcp
             status: up
             interfaces:
                 0: hsn0
   ----------------
   nid000004-nmn
   ----------------
   ```

```
    - net type: tcp
      local NI(s):
        - nid: 10.253.0.22@tcp
          status: up
          interfaces:
              0: hsn0
----------------
nid000002-nmn
----------------
    - net type: tcp
      local NI(s):
        - nid: 10.253.0.15@tcp
          status: up
          interfaces:
              0: hsn0
```

4.2.5 Change DVS Network Transport on User Access Nodes (UANs)

To change the DVS network transport on UANs, the following tasks are required:

- Make changes to a few Ansible configuration files for DVS and LNet in the UAN configuration.
- Update the CFS UAN configuration layer for the UANs.
- Create new UAN images with the updated CFS configuration.
- Update the BOS template used for booting UANs to use the new UAN images.
- Reboot the UANs.

Perform the steps below to accomplish these tasks.

1. Clone the UAN configuration management repository. Checkout the branch currently used to hold UAN configuration. The following example assumes that branch is “integration”.

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

2. Make the same set of changes to configure DVS for running over the new transport which you made during step 3 of the previous section in this document titled “Change DVS Network Transport on Worker Nodes (NCN-Ws)”.

3. Commit and push the changes for the branch to the repository using the proper credentials. Use the output of the first command as the password when prompted.

   ```
   ncn-m001# kubectl get secret -n services vcs-user-credentials \
   --template={{.data.vcs_password}} | base64 --decode
   ncn-m001# git commit -am 'Added COMMIT_COMMENT'
   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':
   ```

4. Identify the commit hash for this branch, and save the commit hash for later use when updating the CFS UAN configuration layer.

   ```
   ncn-m001# git rev-parse --verify HEAD
   <== commit hash output ==>
   ```

5. Update any CFS configurations used by the UANs with the commit ID obtained in the previous step.

6. Create new UAN boot images and update the UAN BOS template. Refer to the UAN product stream documentation for details.

   1. When creating a CFS session to perform pre-boot image customization of the UAN image, be sure to specify the CFS configuration used by the UANs for booting over the new transport from Step 5.

   2. When preparing the UAN BOS template, be sure to set the “rootfs_provider_passthrough” for the correct networks. When booting over the NMN, the section before the last colon should just be “nmn0”. When booting over the HSN, that section should be “hsn0,nmn0”, as shown below.

      ```
      "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:hsn0,nmn0:0"
      ```

7. Reboot the UANs. Replace UUID in the following command with ID of the BOS session template that uses the new UAN images.

   ```
   ncn-m001# cray bos session create --template-uuid UUID --operation reboot
   ```

8. Verify that DVS and LNet services are running over the new transport. In our example below, the HSN uses ksocklnd, where DVS is supposed to use LNet network "tcp" on interface "hsn0" for the UANs:

   ```
   ncn-m001# PDSH_SSH_ARGS='-o StrictHostKeyChecking=no -i /tmp/supersecret' pdsh -w uan[01-04]-nmn \
   lnetctl net show | grep -B5 'hsn0' | dshbak -c
   ----------------
   uan01-nmn
   ----------------
       - net type: tcp
         local NI(s):
           - nid: 10.253.0.27@tcp
             status: up
             interfaces:
                 0: hsn0
   ----------------
   uan03-nmn
   ----------------
       - net type: tcp
         local NI(s):
           - nid: 10.253.0.29@tcp
             status: up
             interfaces:
                 0: hsn0
   ----------------
   uan02-nmn
   ----------------
       - net type: tcp
         local NI(s):
           - nid: 10.253.0.26@tcp
             status: up
             interfaces:
                 0: hsn0
   ----------------
   uan04-nmn
   ----------------
       - net type: tcp
         local NI(s):
           - nid: 10.253.0.28@tcp
             status: up
             interfaces:
                 0: hsn0
   ```

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

   ```
   ncn-m001# exit
   ```

4.3 Tune and Optimize DVS

An overview of the different ways HPE Cray EX administrators can optimize the performance of DVS.

This section contains procedures administrators can use to increase the performance of DVS on HPE Cray EX systems. Consult the following table to find common DVS optimization procedures:

If:	Then try:
The I/O pattern of an application exhibits a "read many, write once" file I/O pattern	Enabling `ro_cache` for the DVS client mounts of that application. See When to Use Temporary, Client-Side, Per-File Read Caching

If:	Then try:
DVS client nodes will not write to a DVS mount	Configure Read-Only Client Mounts for Optimal Performance
Optimal I/O performance of a single application is wanted over uniform performance of all DVS clients	Universally Disable Fairness of Service
DVS client nodes will mount an external IBM Spectrum Scale (formerly GPFS) file system	Improve Performance and Scalability of Spectrum Scale (GPFS) Mounts

Gathering and comparing DVS performance statistics is useful for DVS tuning. See Collect and Analyze DVS Statistics for more details on how to collect this information.

4.3.1 Configure Read-Only Client Mounts for Optimal Performance

HPE Cray EX administrators can increase the I/O performance of read-only DVS client mounts by enabling `loadbalance` mode and by leaving the `attrcache_timeout` at the default of `14400`. The procedure for setting these options for external file systems differs from the procedure for internal file systems.

- Confirm that DVS client (compute) nodes will not write to the DVS-projected file system.
- Confirm that the data in the projected file system rarely changes, if ever.

When DVS client (compute) nodes require read-only access to a file system, and the data in the source file system never or infrequently changes, HPE Cray EX administrators can increase DVS I/O performance. This procedure describes how to configure client mounts with the `loadbalance` option enabled and an `attrcache_timeout` set to the default value of `14400` seconds (four hours).

This procedure applies to both external file systems (for example, Spectrum Scale, NFS) as well as artifacts mounted by the `cpsmount.sh` command. These options are automatically applied to file systems managed by CPS.

1. Determine if the file system to be mounted read-only is internal or external to the HPE Cray EX system.
2. Perform one of the following procedures:
    - If the read-only file system is external, follow the procedure in Project an External Filesystem to Compute Nodes or User Access Nodes and follow the guidance for a read-only file system.
    - If the read-only file system is internal, follow the procedure in Mount a Read-Only Filesystem Optimally with cpsmount.sh to mount the file system manually. File systems mounted using this method do not persist between reboots of either the DVS clients or servers.
    - Refer to the CSM documentation for instructions on projecting a persistent read-only file system to clients using CPS.

4.3.1.1 Mount a Read-Only Filesystem Optimally with `cpsmount.sh`

Mount non-CPS-managed read-only file systems in a way that optimizes DVS I/O performance. If DVS client nodes need read-only access to rarely or never changing data stored on an internal file system (for example, an image from the HPE Cray EX artifact repository), use this procedure.

- **OBJECTIVE**

  Quickly and manually use CPS to project a file system to a client.

- **LIMITATIONS**

  Client mounts created this way do not persist across reboots of the NCNs or CNs.

HPE Cray EX system administrators can configure client nodes to take advantage of the performance benefits available for DVS-projected read-only file systems. This procedure is for situations where such file systems:

- Are not the root file system images for compute nodes.
- Are not stored externally to the HPE Cray EX system. For external (such as IBM Spectrum Scale or NFS) read-only file systems, refer to Project an External Filesystem to Compute Nodes or User Access Nodes.

Mounting an image (artifact) from the artifact repository is a typical use case.

1. List all the S3 artifacts that are boot images and filter for the product type and S3 artifact key of the image that will be mounted by CPS.

   The following example returns the S3 artifact key for all boot images that contain the string PE.

   ```
   ncn-w001# cray artifacts list boot-images --format json | grep Key | grep PE
   "Key": "PE/CPE-base.x86_64-21.03.squashfs",
   ```

   The full S3 path for the artifact in the preceding example is `s3://boot-images/PE/CPE-base.x86_64-21.03.squashfs` because `PE/CPE-base.x86_64-21.03.squashfs` is the value of `"Key"`.

2. Mount the image using the `cpsmount` command. Replace *`S3_PATH`* in the following command with the S3 path obtained in the previous step, and *`MOUNT_POINT`* with the wanted location on the client node file system.

   ```
   ncn-w001# /opt/cray/cps-utils/bin/cpsmount.sh -a api-gw-service-nmn.local -t dvs S3_PATH MOUNT_POINT
   ```

4.3.2 Improve Performance and Scalability of Spectrum Scale (GPFS) Mounts

How to configure both read-only and read/write Spectrum Scale (formerly known as GPFS) DVS client mounts to improve performance and scalability.

4.3.2.1 Optimize DVS Read/Write Mounts of External IBM Spectrum Scale Filesystems

To improve the performance of readable and writable external IBM Spectrum Scale file systems projected by DVS, ensure that:

- The client mount option `blksize` is equal to, or a multiple of, the Spectrum Scale file system blocksize. Otherwise, degraded performance will result.
- If more than one external Spectrum Scale file system will be projected by DVS, and those file systems have different blocksizes, configure a separate DVS client mount (with an appropriate `blksize`) for each Spectrum Scale file system that has a different blocksize.
- All other guidance for optimizing DVS for file system performance in this section is followed as appropriate for the specific site and IBM Spectrum Scale file systems.

4.3.2.2 Optimize DVS Read-Only Mounts of External IBM Spectrum Scale Filesystems

When HPE Cray EX compute nodes require highly scalable read-only access to static (or rarely changing) data stored in an external IBM Spectrum Scale file system, such as shared libraries and input files, Hewlett Packard Enterprise recommends enabling loadbalance mode and specifying `attrcache_timeout=14400` in the mount options. Following the procedure in Project an External Filesystem to Compute Nodes or User Access Nodes to do so.

4.3.2.3 How to Handle Multiple IBM Spectrum Scale Mounts with Different Settings

Having separate mounts configured with different options has the advantage of providing maximum performance and scalability of access. However, having more than one mount has the disadvantage of requiring users, jobs, and applications to know which mount (path) to use when. This disadvantage could be mitigated by setting environment variables for each path (for example, setting `LD_LIBRARY_PATH` to use the read-only/read-write path).

4.3.3 Universally Disable Fairness of Service

Disable DVS fairness of service within an HPE Cray EX system by editing the NCN-GW settings for `dvs.conf` files within the configuration management repository.

Read DVS Fairness of Service to understand the benefits of leaving Fairness of Service enabled.

- **OBJECTIVE**

  Disable DVS fairness of service for all NCN-GWs and within an HPE Cray EX system.

DVS fairness of service is a server-only setting. Changing either `single_msg_queue` or the sixth field in `dvs_instance_info` on DVS clients has no effect.

1. Perform Prepare to Configure DVS with VCS.

   Disable fairness of service in dvs.conf

2. Inspect the value of `dvs_modprobe_conf` variable in all the files in the `group_vars` directory. Repeat the next two steps for all the instances of the variable for DVS Gateway nodes found in this directory.

   The default configuration is one `dvs_modprobe_conf` variable for all nodes, but site-specific needs may require a custom configuration and thus separate files for Workers, CNs, UANs, and NCN-GWs.

3. Open the file `dvs.yml` in an editor. For example:

   ```
   ncn-m001# vi group_vars/dvs_gateways/dvs.yml
   ```

4. Inspect the file for any lines containing `options dvsipc dvs_instance_info=`.

   Since the `dvs_instance_info` kernel module parameter values override any separately set parameters (such as `single_msg_queue`), it must be used to disable fairness of service if it is present.

5. Edit the `dvs.yml` file using either of the following methods:

   - If `options dvsipc dvs_instance_info` is present, then set the `single_msg_queue` field (the sixth number) to 1.
   - If `options dvsipc dvs_instance_info` is absent, then add the line `options dvsipc dvsipc_single_msg_queue=1`.

   For example:

   ```
   ncn-m001# cat group_vars/dvs_gateways/dvs.yml
   dvs_modprobe_conf: |
     options dvsipc dvsipc_single_msg_queue=1
   ```

6. Perform Update CFS with Local Changes

   Force HPE Cray EX system to immediately start using the updated configuration

7. Reboot the NCN Gateway nodes.

4.4 DVS Statistics

DVS can collect and report a large amount of detailed statistics about DVS performance and errors. HPE Cray EX administrators can use these statistics for system monitoring, performance tuning, and troubleshooting.

4.4.1 Collect and Analyze DVS Statistics

Aggregate and per-mount DVS statistics are available for client and server nodes. DVS users and admins can use this data for performance tuning and root-cause analysis of issues.

This procedure walks administrators through the process of obtaining and using DVS statistics.

1. Determine the DVS statistics that report the data of interest and decide if aggregate or per-mount statistics must be collected.

   Refer to DVS Statistics Collected for a comprehensive listing of the statistics DVS collects.

2. Determine the files which contain that statistics identified in the previous step and the specific nodes those files must be read from.

   Refer to Files That Contain DVS Statistics for guidance on selecting the correct `/sys/kernel/debug/` file on the correct node to obtain the information wanted.

3. Decide on an output format for the DVS statistics and determine the format and control options which produce that wanted format. Decide if these options should be set at boot time or after boot.

   See Control and Format Options for DVS Statistics for a list of all the control and format options available, descriptions for each of them, and how to set these options during boot or after a node is booted.

4. Set the control and format options as decided in the previous step.

5. Collect the statistics.

   - Manually read out the contents of appropriate stats file.
   - Use a script to read out one or more stats files. For per-mount statistics, this can be done by parsing the output of `mount -t dvs` and obtaining the absolute path to the right of `statsfile=`. That path is the stats file to read from for that mount. In future releases, HPE may change the order of lines and may deprecate (or remove as obsolete) individual statistics. Therefore, commands and scripts may sometimes require modification after system software upgrades.

6. Use Caveats for Interpreting DVS Statistics to help analyze and interpret the results.

   All output is readable (ASCII) and in a self-describing format.

4.5 Control and Format Options for DVS Statistics

DVS administrators can disable or enable DVS statistics collection, reset the statistics counters, and as well as control the output format that data is reported in.

4.5.0.1 Control and Format Options

Use the options and values described in the following table to control the collection and reporting of statistics.

- Multiple options can be specified, separated by commas, semicolons, spaces, or line-feeds.
- If incompatible options within the same category are specified (for example, `"verbose,brief"`), the last option specified will govern.
- If incompatible options from different categories are specified (for example, `"disable,json"`), the option in a higher-level category will take precedence. The order of precedence (decreasing) for option categories is: `control`, `format`, `flags`. For example, control options such as `disable` take precedence over format options such as `json`.
- Options not specified will continue to use their existing values.

Option Category	Option Value	Result
control	disable	Disables statistics collection and reporting.
control	enable	Enables statistics collection and reporting.
control	reset	Resets statistics.
format	help	Displays current information about the statistics values collected and reported.
format	flat	Displays statistics in flattened text format.
format	json	Displays statistics in JSON structured format.
flags	brief | verbose	`brief` omits statistics that have not been collected. `verbose` displays all statistics, even those that have not been collected. Applies to `flat` and `json` formats only.
flags	plain | pretty	`plain` uses a more machine-readable presentation. `pretty` uses a more human-readable presentation. Applies to `flat` and `json` formats only.

Option Category	Option Value	Result
flags	test | notest	`test` overrides data with a test pattern, which means the statistics data is replaced with fixed test data designed to be used for regression testing of the formatting alone. `notest` removes the test pattern override and restores normal operation.

These combinations of format and flag options are common:

- **flat, plain**

  Structured data is displayed as key/value pairs, one pair per line, key and value separated by a single space. The format is:

  `[container.[container.]...]key value[,value[,value...]]`

  Multiple comma-separated values represent an array of values.

- **flat, pretty**

  Same as flat, plain except that the single space is expanded to produce columns to enhance readability.

- **json, plain**

  Structured data is displayed in JSON format. It consists of a single, unnamed JSON object that contains key/value pairs. There are no spaces or line-feed characters to enhance readability.

- **json, pretty**

  Same as json, plain except that spaces and line-feeds are added to enhance readability.

The incompatible options `list flat,pretty,disable` resolves to `disable` and turns off stats.

4.5.0.2 How to use the DVS statistics control and format options

By default, DVS statistics are enabled, collected, and reported in a simple text format. Statistics can be controlled by:

- Specifying a module parameter at module load time
- Writing text values into the corresponding stats file (see Files That Contain DVS Statistics for a list of these)

Use the `dvsdebug_stat_defaults` parameter to enable and control DVS statistics.

- **dvsdebug_stat_defaults**

  Sets the DVS statistics reporting options to default values at module load time. Every new stats file will take these defaults, which can be changed later at any time for each stats file. Use this parameter to disable, enable, and format DVS statistics.

  - Default value: enable
  - To view: `cat /sys/module/dvsproc/parameters/dvsdebug_stat_defaults`
  - To change prior to boot, add this line to `/etc/modprobe.d/dvs-local.conf` and replace *`options_list`* with a comma-separated list of options:

    ```
    # Disable/enable and format DVS statistics
    options dvsdebug dvsdebug_stat_defaults="options_list"
    ```

    Note: See HPE Cray EX DVS Configuration Overview for more details.

- To change dynamically:

  This is root writable at `/sys/module/dvsproc/parameters/dvsdebug_stat_defaults`, but changes should be made only through the `/sys/kernel/debug/dvs/stats` interface, as shown in this example.

  ```
  hostname# echo disable > /sys/kernel/debug/dvs/stats
  hostname# echo enable > /sys/kernel/debug/dvs/stats
  hostname# echo reset > /sys/kernel/debug/dvs/stats
  hostname# echo json,pretty > /sys/kernel/debug/dvs/stats
  ```

4.5.1 Caveats for Interpreting DVS Statistics

DVS users and admins who collect and use DVS performance statistics must be aware of a few caveats when interpreting that data.

4.5.1.1 Statistics and Caching

DVS statistics measure I/O that passes through DVS. In general, all `read`, `aio_read`, `write`, and `aio_write` operations specified by the user application result in a DVS `read`, `aio_read`, `write`, or `aio_write` operation. DVS records the total bytes reported to the application for each read or write, and the maximum file offset accessed. These statistics are recorded regardless of whether the file system is cache-enabled.

If client-side caching is disabled, each read or write operation is forwarded to one or more servers. This results in a file system read or write on the server and a transfer of data between client and server. The accumulated totals in the statistics represent actual load on the DVS transport.

If client-side caching is enabled, the read or write operation is not forwarded to one or more servers. Instead, it is diverted to a Linux routine that attempts to satisfy the read or write using the Linux file system cache.

- If the data is already in the cache, Linux completes this operation entirely in memory. The accumulated totals in the statistics represent no load on the DVS transport.
- If the data is not already in the cache, Linux issues a page read or write operation to DVS, which results in that read or write sent to one or more servers. DVS statistics record the total bytes reported for this cache read or write.

The effectiveness of the cache is represented by the ratio of user read or write bytes to cache read or write bytes.

- **ratio > 1.0**

  Indicates that the cache is being used effectively, with more cache hits than misses.

- **ratio = 1.0**

  Represents something like sequential reads of a large file: reads force continual cache fills, but the reads then consume all the bytes in the cache before forcing a new cache fill.

- **ratio < 1.0**

  Indicates that cache is thrashing, such as would be caused by making small reads from many open files, such that only a tiny fraction of each cache page is ever consumed.

It is not feasible to determine whether any specific user application read or write “hit” or “missed” the cache, without modifying the Linux kernel.

4.5.1.2 Statistics and the `mmap()` Functions

The `mmap()` functions use the same mechanisms as the Linux cache: mmap can be thought of as a named, “private” Linux cache that the application sets up. The `mmap()` functions cannot be used unless the DVS file system is cache-enabled. As a result, any attempt to read from or write to the file is diverted to the same Linux routine as would be used for any caching. The Linux routine determines that this is a `mmap()` file and sends a page request to DVS, then satisfies the read or write from memory. As in the case of caching, DVS statistics track the reads and writes of the user application and the Linux page requests separately. As with caching, it cannot be determined whether a specific read or write “hit” or “missed” the mmap.

Because it is difficult to link a user application read or write with a corresponding page read or write, which is usually a many-to-one linkage, it is difficult to distinguish between a Linux file system cache operation and an `mmap()` cache operation.

4.5.1.3 Rates and Averaging

Performance rate measurements are simply counts that are automatically zeroed when sampled by reading the stats file. The resulting total is displayed, divided by the time since the last stats file read. Therefore it is possible to create real-time performance data with one-second resolution by reading the stats file every second. Because the DVS code normalizes the value by dividing by the time elapsed since the last sample, variations in the sampling rate should not be reflected in the averages. For example, if the actual data transfer rate is a steady 1GB/sec, a rate of 1GB/sec will be displayed every time the stats file is sampled, even if it is sampled manually at random intervals. If the actual data transfer rate is fluctuating, all the fluctuating rates will be averaged together (unweighted) over the entire sampling window, however long that may be.

4.5.1.4 Races

To keep DVS performance high, statistics reporting is not atomic: atomic values are sampled as each line of output is rendered, while DVS is running. As a result, counts that might be expected to have an exact mathematical relationship may not. For example, if all applications are reading in fixed-size blocks, one might expect an exact relationship between IOPS and bytes read. However, that relationship will not generally be exact, because values are sampled as the statistics output is rendered, and values rendered later in time will contain new counts not found in values rendered earlier.

4.5.2 Files That Contain DVS Statistics

DVS records data about its performance (both aggregate and per mount point) in several files in `/sys/kernel/debug/`. To collect and report DVS statistics, read from these files.

DVS provides both aggregate and per-mount statistics to enable performance and root-cause analysis. It reports statistics for client and server nodes in the following files. Each stats file is readable and writable.

- `/sys/kernel/debug/dvs/statistics/`

  Contains two files with statistics for the average timing of messages between the DVS client and the DVS server. Including:

  - Time spent queued on the server.
  - Time spent being processed by the server.
  - Time spent in the underlying file system
  - Time spent on the network and in network transport software
  - `client_message_timings`

    The `client_message_timings` file includes timing information for all requests and replies that the node sends or receives as a client. On servers, the `client_message_timings` file will likely be mostly zeroes.

  - `server_message_timings`

    The `server_message_timings` file includes timing information for all messages that a node receives as a server. On clients, the `server_message_timings` file will likely be mostly zeroes.

- `/sys/kernel/debug/dvs/stats`

  Contains aggregate statistics for the node. These reflect system operations that cannot be correlated to a specific DVS mount point and are therefore most interesting on DVS servers.

- `/sys/kernel/debug/dvs/mounts/NNN/stats`

  Contains per-mount statistics for the node, where *NNN* is a decimal integer value global to DVS on that node. Every invocation of the mount command creates the numbered mount directory and increments *NNN*. Every invocation of the umount command deletes the numbered mount directory but does not decrement *NNN*. The value of *NNN* appears in the `statsfile=/sys/kernel/debug/dvs/mounts/NNN/stats` parameter in the mount options for the mounted DVS file system. This can be obtained using the `mount -t dvs` command. More information about each of these mount points can be obtained by viewing the mount file that resides in the same directory (`/sys/kernel/debug/dvs/mounts/NNN/mount`).

- `/sys/kernel/debug/dvsipc/stats`

  Contains DVS interprocess communication (IPC) statistics, including bytes transferred and received, and NAK counts. It also contains message counts by type and size.

4.5.3 DVS Statistics Collected

Lists all the statistics collected by DVS.

The following table shows the full keys in flat format and describes whether the statistic is aggregate across all mounts (`/sys/kernel/debug/dvs/sta`
or per-mount (`/sys/kernel/debug/dvs/mounts/NNN/stats`), or both.

4.5.3.1 dvs statistics table

Key	AGG	MNT	Meaning
STATS.version	*	*	Current version of statistics. Anytime this number changes, the format or content of the statistics have changed. Because DVS uses a self-describing format (key/value or JSON), analysis code may be able to run with new versions without code changes; however, there is no guarantee.
STATS.flags	*	*	Current option flags.
RQ.req.req.ok	*	*	Count of requests sent by this host that resulted in successful response. See the table Valid req Values for valid values for the RQ keys.
RQ.req.req.err	*	*	Count of requests sent by this host that resulted in a failed send or a failure response.
RQ.req.reqp.ok	*	*	Count of requests received by this host that resulted in successful action on this host.
RQ.req.reqp.err	*	*	Count of requests received by this host that resulted in failed action on this host.
RQ.req.reqp.dur.prv	*	*	Duration (seconds) of the most recent request received and processed by this host.
RQ.req.reqp.dur.max	*	*	Maximum duration (seconds) of any requests received and processed by this host.
OP.fileop.ok	*	*	Count of successful file operations called by Linux on this host. See the table Valid fileop Values for valid values for the OP keys.
OP.fileop.err	*	*	Count of failed file operations called by Linux on this host.
OP.fileop.dur.prv	*	*	Duration (seconds) of the most recent file operation on this host.
OP.fileop.dur.max	*	*	Maximum duration (seconds) of any file operations on this host.
IPC.requests.ok, IPC.requests.err, IPC.async_requests.ok, IPC.async_requests.err, IPC.replies.ok, IPC.replies.err	*		These have the same meanings as the preceding RQ.req.req.ok and RQ.req.req.err keys, except for these differences: 1) IPC keys show aggregated messages across all mounts (RQ keys show individual messages) 2) IPC keys grouped by whether messages sent synchronously, asynchronously, or as a reply message.
PERF.user.read.min_len		*	Low-water mark of all nonzero-length user read operations. Includes both read() and aio_read() calls.
PERF.user.read.max_len		*	High-water mark of all nonzero-length user read operations.
PERF.user.read.max_off		*	High-water mark of all file byte offsets read by user calls.
PERF.user.read.total.iops		*	Accumulated I/O operation count of all user read operations since module load.
PERF.user.read.total.bytes		*	Accumulated byte transfer count of all user read operations since module load.

Key	AGG	MNT	Meaning
PERF.user.read.rate.iops		*	Accumulated I/O operation count of all user read operations since the last read of stats, divided by the number of seconds since the last read of stats.
PERF.user.read.rate.bytes		*	Accumulated byte transfer count of all user read operations since the last read of stats, divided by the number of seconds since the last read of stats.
PERF.user.write.min_len		*	Low-water mark of all nonzero-length user write operations. Includes both write() and aio_write() calls.
PERF.user.write.max_len		*	High-water mark of all nonzero-length user write operations.
PERF.user.write.max_off		*	High-water mark of all file byte offsets written by user calls.
PERF.user.write.total.iops		*	Accumulated I/O operation count of all user write operations since module load.
PERF.user.write.total.bytes		*	Accumulated byte transfer count of all user write operations since module load.
PERF.user.write.rate.iops		*	Accumulated I/O operation count of all user write operations since the last read of stats, divided by the number of seconds since the last read of stats.
PERF.user.write.rate.bytes		*	Accumulated byte transfer count of all user write operations since the last read of stats, divided by the number of seconds since the last read of stats.
PERF.cache.read.min_len		*	Low-water mark of all nonzero-length Linux cache read operations.
PERF.cache.read.max_len		*	High-water mark of all nonzero-length Linux cache read operations.
PERF.cache.read.max_off		*	High-water mark of all file byte offsets read by Linux cache calls.
PERF.cache.read.total.iops		*	Accumulated I/O operation count of all Linux cache read operations since module load.
PERF.cache.read.total.bytes		*	Accumulated byte transfer count of all Linux cache read operations since module load.
PERF.cache.read.rate.iops		*	Accumulated I/O operation count of all Linux cache read operations since last read of stats, divided by the number of seconds since the last read of stats.
PERF.cache.read.rate.bytes		*	Accumulated byte transfer count of all Linux cache read operations since last read of stats, divided by the number of seconds since the last read of stats.
PERF.cache.write.min_len		*	Low-water mark of all nonzero-length Linux cache write operations.
PERF.cache.write.max_len		*	High-water mark of all nonzero-length Linux cache write operations.
PERF.cache.write.max_off		*	High-water mark of all file byte offsets written by Linux cache calls.
PERF.cache.write.total.iops		*	Accumulated I/O operation count of all Linux cache write operations since module load.
PERF.cache.write.total.bytes		*	Accumulated byte transfer count of all Linux cache write operations since module load.
PERF.cache.write.rate.iops		*	Accumulated I/O operation count of all Linux cache write operations since the last read of stats, divided by the number of seconds since the last read of stats.
PERF.cache.write.rate.bytes		*	Accumulated byte transfer count of all Linux cache write operations since the last read of stats, divided by the number of seconds since the last read of stats.

Key	AGG	MNT	Meaning
PERF.legacy.inodes.created		*	Accumulated count of inodes created on this host (includes mirrored inodes that represent existing files on the server).
PERF.legacy.inodes.deleted		*	Accumulated count of inodes deleted on this host.
PERF.files.created		*	Accumulated count of regular files created on server file systems.
PERF.files.deleted		*	Accumulated count of regular files deleted on server file systems.
PERF.files.open		*	Current count of DVS files open.
PERF.symlinks.created		*	Accumulated count of symlinks created on server file systems.
PERF.symlinks.deleted		*	Accumulated count of symlinks deleted on server file systems.
PERF.directories.created		*	Accumulated count of directories created on server file systems.
PERF.directories.deleted		*	Accumulated count of directories deleted on server file systems.

4.5.3.2 DVS Messages

The following table shows the valid req values to be used in the RQ statistics, as described in the table dvs statistics table. They represent messages passed between the DVS client and server.

4.5.3.3 Valid req values table

req	Meaning
RQ_CLOSE	Close open file on the server.
RQ_CREATE	Create a regular file on the server.
RQ_FASYNC	Manage asynchronous notifications of a file descriptor on the server.
RQ_FLUSH	Close an instance of an open file on the server.
RQ_FSYNC	Flush a file to backend storage on the server.
RQ_GETATTR	Get file attributes on the server.
RQ_GETEOI	Return size of file on the server.
RQ_GETXATTR	Get file extended attributes on the server.
RQ_IOCTL	Perform a general ioctl on the server.
RQ_LINK	Create a link from one path to another on the server.
RQ_LISTXATTR	List extended attributes on the server.
RQ_LOCK	Lock a file on the server.
RQ_LOOKUP	Determine if a path exists on the server.
RQ_MKDIR	Create a directory on the server.
RQ_MKNOD	Create a special device file on the server.
RQ_OPEN	Open a file on the server.
RQ_PARALLEL_READ	Read from an open file on the server.
RQ_PARALLEL_WRITE	Write to an open file on the server.
RQ_PERMISSION	Determine if user has permission to an object on the server.
RQ_READDIR	Read the contents of a directory on the server.
RQ_READLINK	Read the contents of a symlink on the server.
RQ_READPAGES_RP	Client only - data for page cache file read has been returned.
RQ_READPAGES_RQ	Read from an open file on the server that resides in the client page cache.
RQ_REMOVEXATTR	Remove extended attributes on the server.
RQ_RENAME	Rename a file on the server.
RQ_RMDIR	Remove a directory on the server.
RQ_RO_CACHE_DISABLE	Disable the DVS ro_cache option for an open file on the server.
RQ_SETATTR	Set basic attributes of an object on the server.
RQ_SETXATTR	Set extended attributes of an object on the server.

req	Meaning
RQ_STATFS	Perform a file stat on the server.
RQ_SYMLINK	Create a symlink on the server.
RQ_SYNC_UPDATE	Return list of inodes that have been synced to backend storage on server recently.
RQ_TRUNCATE	Truncate an open file on the server.
RQ_UNLINK	Unlink (delete) a file object on the server.
RQ_VERIFYFS	Verify that a path exists on the server so that it can be mounted on a client.
RQ_WRITEPAGES_RP	Client only - data for page cache file write has been written.
RQ_WRITEPAGES_RQ	Write from an open file residing in the client page cache to the server.

4.5.3.4 Virtual File System Operations

The following list contains the valid fileop values to be used in IPC statistics, as described in the table dvs statistics table. They are virtual file system (VFS) operations requested by the Linux VFS framework, and they represent file system actions to be performed by the kernel on behalf of a user-space application. For more details on these VFS operations, refer to https://www.kernel.org/doc/html/latest/filesystems/vfs.html or to the corresponding page for the kernel version used on the CNs and NCNs.

For example, a call to `open()` in a user application results in a call to the open handler (fileop = open in the following list), requesting that the DVS file system open a file. VFS operations may operate locally (for example, out of local cache memory), or they may result in one or more request messages sent to the server. The `d_`, `f_`, and `l_` prefixes are vestigial references to the file system object on which the operation is performed. They will be removed in a future release.

4.5.3.4.1 Valid fileop values list

- `aio_read`
- `d_rmdir`
- `flock`
- `readdir`
- `aio_write`
- `d_setattr`
- `flush`
- `readpage`
- `d_create`
- `d_setxattr`
- `fsync`
- `readpages`
- `d_getattr`
- `d_symlink`
- `l_follow_link`
- `release`
- `d_getxattr`
- `d_unlink`
- `l_getattr`
- `show_options`
- `d_link`
- `direct_io`
- `l_put_link`
- `statfs`
- `d_listxattr`
- `evict_inode`
- `l_readlink`
- `unlocked_ioctl`
- `d_lookup`
- `f_getattr`
- `l_setattr`
- `write`
- `d_mkdir`
- `f_getxattr`

- `llseek`
- `write_begin`
- `d_mknod`
- `f_listxattr`
- `lock`
- `write_end`
- `d_permission`
- `f_removexattr`
- `mmap`
- `writepage`
- `d_removexattr`
- `f_setattr`
- `open`
- `writepages`
- `d_rename`
- `f_setxattr`
- `put_super`
- `d_revalidate`
- `fasync`
- `read`

4.6 Choose and Configure a DVS Mode

This procedure helps administrators of HPE Cray EX supercomputers select and configure the best DVS mode for projecting a new file system. The best mode for projecting a particular file system depends on:

- The amount of load distribution wanted.
- The performance and other requirements of the codes that use the data in that file system.
- The type and properties of the projected file system.

There is no need to configure the root file system for DVS client nodes (for example, compute nodes and User Application Nodes). CPS configures and handles those file systems (see Introduction to DVS).

1. Determine the requirements and properties of the file system to be projected.

   - Is POSIX read/write atomicity required?
   - Is the file system a cluster (that is, shared) file system?
   - Must the file system provide high I/O performance for reads, writes, or both?
   - Will DVS clients mount the file system read-only?
   - How large is the file system?

2. Determine if the file system must be projected by DVS separately from CPS. Skip this step if the DVS client nodes need to write to the new file system.

   CPS can be used only if:

   - The file system can be stored as a SquashFS file that is stored in S3.
   - An NCN Worker node can store the SquashFS file on its root disk.
   - DVS client nodes do not need to write to the file system.

3. Select the most appropriate DVS mode for the file system by using the following list and DVS Modes. Skip the rest of this procedure if CPS is chosen.

   - **Use Serial Mode** to project a non-cluster file system that requires POSIX read/write atomicity from a single DVS server node.
   - **Use Atomic Stripe Parallel Mode** to project from multiple DVS servers a ***non-cluster*** file system that requires both POSIX read/write atomicity and I/O parallelism.
   - **Use Cluster Parallel Mode** to project from multiple DVS servers a ***cluster*** (shared) file system, such as IBM Spectrum Scale, that requires both POSIX read/write atomicity and I/O parallelism.
   - **Stripe Parallel Mode** to project from multiple DVS servers a read/write file system that is not NFS and does not require POSIX read/write atomicity.
   - **Use CPS** to project a read-only file system with I/O parallelism and failover across multiple DVS servers. See Add CPS Content.

- **Use Loadbalance Mode** to project a read-only file system with I/O parallelism and failover across multiple DVS servers when CPS cannot be used.

NCN Gateway (NCN-GW) nodes must be used to project a file system with DVS separately from CPS. These nodes belong to the `Application_Gateway` node group.

4. Perform Prepare to Configure DVS with VCS.
5. **(Optional):** Configure the new external file system on the DVS servers. Skip this step if the new file system is not external to the HPE Cray EX system, or if the external file system cannot use the `configure_fs` role.

   Some external file systems, like IBM Spectrum Scale, cannot use the Ansible `configure_fs` role (and the `filesystems.yml` file that configures it) to mount on the NCN-GW nodes. However, NFS and other external file systems must use this method to mount.

   1. Open the `group_vars/Application_Gateway/filesystems.yml` file in an editor.
   2. Configure the new file system according the documentation for the external file system.
6. Update the DVS exports.

   Refer to Allowing Client Access to DVS Server File Systems
7. Configure the file system mount on the DVS clients.

   1. Check the `group_vars/Compute` directory in the COS VCS repository for any files that contain `filesystems:` sections similar to:

      ```
      filesystems:
           - src: /home
             mount_point: /home
             fstype: dvs
             opts: noauto,path=/home,nodename=x3000c0s25b0n0:x3000c0s23b0n0
             state: mounted
      ```

   2. If the previous step does not uncover any files with `filesystems:` sections, create a new YAML file in this directory. Otherwise, skip this step and use one of the files identified in the previous step.
   3. Edit the client mount file selected in the previous step to set the correct mount options in the `opts:`

      - Serial Mode: `nodename=DVS_SERVER_1,maxnodes=1,path=PATH`
      - Cluster Parallel Mode: `nodename=DVS_SERVER_1,DVS_SERVER_2 . . .,maxnodes=1,path=PATH`
      - Stripe Parallel Mode: `nodename=DVS_SERVER_1,DVS_SERVER_2 . . .,maxnodes=X,path=PATH`
      - Atomic Stripe Parallel Mode: `nodename=DVS_SERVER_1,DVS_SERVER_2 . . .,maxnodes=X,atomic,path=PATH`
      - Loadbalance Mode: `nodename=DVS_SERVER_1,DVS_SERVER_2 . . .,maxnodes=1,loadbalance,path=PATH`

      Where:

      - `DVS_SERVER_1` and `DVS_SERVER_2` are the xnames of two DVS servers
      - `X` is a value that is at least 2, but not greater than the number of nodes specified in the `nodename` list. All the modes, except for Serial Mode, support two or more DVS servers.
      - `. . .` are the xnames of any additional DVS servers.
      - `PATH` is the mount point on the clients for the projected file system. This is the same path that must be entered in the `mount_point` line.
   4. Add any other client mount options, following the guidance in DVS Mount Options.
   5. Enter the rest of the required client mount information in the YAML file.

      The "Enable DVS Failover and Failback" subsection of the ***Mangage and Customize DVS*** section contains two client configuration examples: one of a read-only Serial Mode file system and one of read-only file system projected using Loadbalance mode.

      - An example `filesystems` YAML section for Cluster Parallel Mode:

        ```
        filesystems:
         - src: /nfs_fs
           mount_point: /cluster_parallel_mode
           fstype: dvs
           opts: rw,noauto,path=/cluster_parallel_mode,nodename=DVS_SERVER_1:DVS_SERVER_2,maxnodes=1
           state: mounted
        ```

- A Stripe Parallel Mode example:

```
filesystems:
 - src: /gpfs_fs
   mount_point: /stripe_parallel_mode
   fstype: dvs
   opts: rw,noauto,path=/stripe_parallel_mode,nodename=DVS_SERVER_1:DVS_SERVER_2,maxnodes=2
   state: mounted
```

- An example Atomic Stripe Parallel Mode:

```
filesystems:
 - src: /gpfs_fs
   mount_point: /atomic_stripe_parallel_mode
   fstype: dvs
   opts: rw,noauto,path=/atomic_stripe_parallel_mode,nodename=DVS_SERVER_1:DVS_SERVER_2,
maxnodes=2,atomic
   state: mounted
```

8. Perform Update DVS Configuration with VCS

4.7 Manage and Customize DVS

A collection of procedures for managing DVS and enabling additional functionality. This topic does not include performance optimization.

4.7.1 Procedures to Manage and Optimize DVS Use

DVS is able to perform its core duty within an HPE Cray EX system without user configuration after installation. DVS can be modified, however, to serve site-specific needs. System administrators may also need to occasionally perform maintenance tasks. Use the following procedures, as needed, during operation or for further configuration:

Site-specific need:	Procedure:
Nodes on the HPE Cray EX system need access to an external file system (such as NFS or IBM Spectrum Scale)	Project an External Filesystem to Compute Nodes or User Access Nodes
Administrators must safely perform operations on a DVS-projected directory (or file system) without any possibility of interference by a DVS client. Examples include repairing a projected file system or taking a DVS server node out of service safely.	Quiesce a DVS-Projected File System
Allow DVS clients to access recent changes on a DVS-projected file system without waiting for the attribute cache timeout period to expire.	Force a Cache Revalidation on a DVS Mount Point

4.7.2 Allowing Client Access to DVS Server File Systems

Starting in COS 2.1, DVS switched from projecting all file system content on a node to requiring directories to be explicitly exported for projection. This prevents DVS nodes from accidentally projecting sensitive data to other nodes in the system. That could lead to compromised client nodes reading or altering sensitive data they shouldn't have access to.

Each node in the system is assigned a list of directory paths to export to other nodes in the system. Each path is exported in a Read-Write or Read-Only mode. A Read-Write directory path allows any client to read or write anything in that directory (or the directory's sub-directories), while a Read-Only directory only allows clients to read. Any directory which is not explicitly exported (or which is not a subdirectory of an explicitly exported directory) is unaccessable by any client. Read-Write and Read-only access applies to not only files, but also directories. Trying to create or remove a file or directory in a Read-Only-exported directory results in an EROFS error.

A node can have an empty exported directory list. In this case, the node doesn't act as a DVS server at all. This is usually the case for compute nodes.

It is allowed to export directories which are subdirectories of other directories. This is most often used to change the manner in which the subdirectory is exported. For example, directory `/foo` could be exported Read-Write while directory `/foo/bar` is exported Read-Only. A client would be able to change, read, and write files and directories in /foo and its subdirectories except for /foo/bar. /foo/bar and its subdirectories would be able to be read, but not changed.

Directories may also be exported with a "None" mode which specifically disallows access to a directory. This is useful in the case where the

admin wants to prevent access to a smaller part of a bigger exported directory. Using the above example, /foo/bar could be exported "None". In that case, /foo/bar would be unaccessable, while the rest of /foo was read and writeable. Any attempt to access /foo/bar would result in an EPERM error.

The exports for a node are controlled by changing the `dvs_exports_yml` variable for that node type. These variables are found in the system's VCS repo in the `group_vars` directory. In particular, they're kept in the `dvs.yml` files. For example, Gateway nodes can have their `dvs_exports_yml` variable changed by updating `group_vars/Application_Gateway/dvs.yml`.

As an example, the default `dvs_exports_yml` for Workers is seen below. The variable contains an `exports` dictionary entry which contains a list of exports. Each element in the list is a dictionary which two entries: `mode` and `path`. The path is the path of the directory to export and mode is one of `ro`, `rw`, or `none`. The default for workers is such that the directory that CPS needs to be projected is exported Read-Only and every other path on the Worker is unexported.

```yaml
dvs_exports_yml:
  exports:
  - mode: ro
    path: /var/lib/cps-local
```

The definitions for the other node types all default to empty lists, such as the one below. For Compute and UAN nodes, there isn't much need for changing the default, as these node groups should probably not ever project DVS.

```yaml
dvs_exports_yml:
  exports: []
```

Since the Application_Gateway node type is explicitly for projection of customer file systems, the `dvs_exports_yml` variable in that file should be customized for the correct paths for the site. For example:

```bash
ncn-m001# cat group_vars/Application_Gateway/dvs.yml
dvs_exports_yml:
  exports:
  - mode: rw
    path: /gpfs1
  - mode: rw
    path: /nfs2
```

The `dvs_exports_yml` variable is used to populate the `/etc/dvs_exports.yml` file on each node. On Workers, this file is created during the NCN Personalization process. For other nodes types, the file is added to the image during the Image Customization process. In either case, the contents of `/etc/dvs_exports.yml` file is loaded into the DVS kernel driver by a systemd unit file which calls the `dvs_exportfs` command line tool.

See the ***Project an External Filesystem to Compute Nodes or User Access Nodes*** section for more information on how to configure exports on a Gateway Node.

4.7.3 Prepare to Configure DVS with VCS

Perform this procedure to accomplish all the preparation steps for modifying the DVS configuration for CNs and NCNs. After completing this procedure, the administrator will be able to change the DVS configuration persistently by modifying the local copy of the Ansible files.

Perform this procedure as directed by the other procedures in this guide. All commands must be run from a worker NCN that has the cray administrative CLI installed and initialized.

1. Obtain the password for the `crayvcs` user.

   This password is required for the next step.

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

2. Create a local copy of the HPE Cray OS (COS) configuration management repository. Then cd into it.

   All DVS and LNet configuration for CNs and NCNs is done within HPE COS product.

   ```
   ncn-m# git clone https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git
   Cloning into 'cos-config-management'...
   remote: Enumerating objects: 421, done.
   remote: Counting objects: 100% (421/421), done.
   remote: Compressing objects: 100% (279/279), done.
   remote: Total 421 (delta 98), reused 283 (delta 41)
   Receiving objects: 100% (421/421), 81.65 KiB | 11.66 MiB/s, done.
   Resolving deltas: 100% (98/98), done.
   ncn-m# cd cos-config-management/
   ```

3. Obtain a list of the branches of the cos-config-management VCS repository on the HPE Cray EX system and determine which branch listed in the output is used to configure the system.

   ```
   ncn-m# git ls-remote https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git \
   refs/heads/cray/cos/*
   4f8e919ede7143929e26ee8d97e04c12572a2f90    refs/heads/cray/cos/1.4.0
   ccd964f8eeeb52ab8f895b480c5d1142c7bc0a8e    refs/heads/cray/cos/2.0.16-20210209170718-62cb853
   ccd964f8eeeb52ab8f895b480c5d1142c7bc0a8e    refs/heads/cray/cos/2.0.17
   ccd964f8eeeb52ab8f895b480c5d1142c7bc0a8e    refs/heads/cray/cos/2.0.18
   035ab5d400c22df123f9bd4962a5a8c1013a2703    refs/heads/cray/cos/2.1.13-20210212225706-6218fd1
   ```

4. Create a local branch that corresponds to the current COS release version used by the HPE Cray EX system.

   The following example assumes that the current COS release is 1.4.0. Use the branch name determined in the previous step. A local branch is required for the changes since the VCS does not allow updates to original remote branch in this release.

   ```
   ncn-m# git checkout cray/cos/1.4.0
   Branch 'cray/cos/1.4.0' set up to track remote branch 'cray/cos/1.4.0' from 'origin'.
   Switched to a new branch 'cray/cos/1.4.0'
   ncn-m# git checkout -b integration cray/cos/1.4.0
   Switched to a new branch 'integration'
   ```

5. Repeat Steps 2-4 to clone the UAN repository.

   Clone the repository at https://api-gw-service-nmn.local/vcs/cray/uan-config-management.git. Find the branch that represents the version of the UAN code you want to use, and create an integration branch.

6. Repeat Steps 2-4 to make a clone of the UAN repository to use for gateways.

   This time, use the command below to clone the repository. It will create directory called gateways-config-management that will contain configuration for gateways.

   ```
   ncn-m# git clone https://api-gw-service-nmn.local/vcs/cray/uan-config-management.git \
   gateways-config-management
   Cloning into 'gateways-config-management'...
   remote: Enumerating objects: 324, done.
   remote: Counting objects: 100% (324/324), done.
   remote: Compressing objects: 100% (200/200), done.
   remote: Total 324 (delta 43), reused 298 (delta 36)
   Receiving objects: 100% (324/324), 69.31 KiB | 23.10 MiB/s, done.
   Resolving deltas: 100% (43/43), done.
   ```

   Use the UAN branch you found branch you found in step 5 to create a branch called “integration-gateways”.

   ```
   ncn-m# git checkout cray/uan/2.0.1
   Branch 'cray/uan/2.0.1' set up to track remote branch 'cray/uan/2.0.1' from 'origin'.
   Switched to a new branch 'cray/uan/2.0.1'
   ncn-m# git checkout -b integration-gateways cray/uan/2.0.1
   Switched to a new branch 'integration-gateways'
   ```

Configure DVS as wanted by resuming the procedure that referred to this one.

4.7.4 Reload DVS on an NCN Worker

Reload DVS on one or more NCN Workers to restart DVS, if DVS or LNet parameters have changed or to resolve a DVS issue (for example, IP address changes for existing node map entries). Starting with HPE Cray EX release 1.4, restarting the `cps-cm-pm` pods will not reload DVS.

This procedure is to be performed successively on one worker at a time, so that the CNs and UANs can keep running by failing over to other workers that are still running and not being reloaded.

Note that this procedure only affects DVS for the CPS case. DVS for external file system projection is handled on NCN Gateway nodes.

To reload DVS, run the `dvs_reload_ncn` script. This script requires the name of the CFS configuration for the NCN Workers (`ncn-personalization`) and the xnames of the node or nodes which need DVS reloaded on them. The last line of output from this script is the name of Kubernetes job to watch to monitor the reload.

1. Unload DVS and LNet services if they are currently running on worker NCNs.

Refer to the ***Unload DVS and LNet services for the worker nodes*** step in the ***Perform NCN NCN Personalization*** procedure of the publication ***HPE Cray Operating System Installation Guide CSM on HPE Cray EX Systems***.

2. Run the `dvs_reload_ncn` script. Replace *`NCN_XNAME`* in the following command with the xname of the NCN.

   More than one xname can be given to the script, but the reference count checks for the dvs and lustre modules must be performed for each NCN.

   ```
   ncn-w001# /opt/cray/dvs/default/sbin/dvs_reload_ncn -c ncn-personalization NCN_XNAME
   Created/Updated CFS configuration "playbooks-cray-dvs-reload-yml".
   Started CFS session "playbooks-cray-dvs-reload-yml".

   The CFS session is now running:
   {
       "ansible": {
           "config": "cfs-default-ansible-cfg",
           "limit": null,
           "verbosity": 0
       },
       "configuration": {
           "limit": "",
           "name": "playbooks-cray-dvs-reload-yml"
       },
       "name": "playbooks-cray-dvs-reload-yml",
       "status": {
           "artifacts": [],
           "session": {
               "completionTime": null,
               "job": "cfs-8ec9a0d7-e4e9-41b6-9980-2e944328f5bd",
               "startTime": null,
               "status": "pending",
               "succeeded": "none"
           }
       },
       "tags": {},
       "target": {
           "definition": "spec",
           "groups": [
               {
                   "members": [
                       "x3000c0s7b0n0"
                   ],
                   "name": "dvs_nodes"
               }
           ]
   ```

```
    }
}

The K8S job is:
services     cfs-8ec9a0d7-e4e9-41b6-9980-2e944328f5bd-ndx5v     0/3     Init:0/2
0     0s     <none>     ncn-w001   <none>     <none>
```

3. **Optional:** Check the Kubernetes log to verify that the DVS reload successfully completes.

   ```
   ncn-w001# kubectl logs -n services cfs-8ec9a0d7-e4e9-41b6-9980-2e944328f5bd-ndx5v ansible-0 -f
   ```

4.7.5 Enable DVS Failover and Failback

Perform this procedure to enable failover and failback resiliency when projecting a file system with DVS independent of CPS. System administrators must update the configuration management repository, create new client images, then boot the clients with those new images.

Before performing this procedure, perform Prepare to Configure DVS with VCS

This procedure assumes:

- One NCN running standalone (non-CPS) DVS in Serial Mode is already projecting the file system to clients that will have failover and failback enabled.
- The necessary configuration for the new client amount exists in a `filesystems.yml` file somewhere within the `group_vars` directory of the relevant configuration management repositories. Refer to "Mount a New File System on a UAN" and "Set Up a Lustre File System with CFS" procedures in the CSM documentation for examples.

To enable failover and failback resiliency for a DVS-projected file system, HPE Cray EX administrators must:

1. Edit the Ansible playbooks that configure DVS on the clients.
2. Reboot the CNs and other DVS clients with the new images.

Content projected by CPS already has failover and failback enabled, since CPS uses DVS Loadbalance Mode with multiple `cps-cm-pm` pods by default. Therefore, this procedure only applies to content projected by DVS when DVS is operating independently of CPS ("standalone DVS").

To save time, enable failover and failback when initially configuring standalone DVS to project the content. In that scenario, only perform Step 3 of this procedure. All the other steps will be performed as part of configuring and starting standalone DVS projection.

1. Determine where in the `group_vars` directory the client mount configuration for the projected file system resides.

   The following example YAML excerpt is for a read-only file system. This configuration information may reside in either a `group_vars/`*`NODE_TYPE`*`/lustre.yml` file or in a separate `group_vars/Compute/filesystems.yml` file.

   ```
   - src: /example_projected_fs
     mount_point: /example_projected_fs_mount_point
     fstype: dvs
     opts: attrcache_timeout=14400,ro,
     path=/example_projected_fs_mount_point,
     nodename=x3000c0s25b0n0
     state: mounted
   ```

2. Open the client mount configuration file for editing.
3. Configure the client mount for failover and failback.
   1. Ensure that the xnames of at least two members of the `dvs_gateways` HSM group are specified for the nodename argument on the client mount.
   2. Specify one of the three parallel DVS modes.
   3. Add the failover option unless loadbalance mode is chosen.

   The NCN list for the nodename argument is colon-separated. The following excerpt is identical to the previous one except that the file system is projected in DVS Loadbalance Mode using two separate DVS server nodes. The unnecessary `ro` option is removed as `loadbalance` sets the mount as read-only.

```
- src: /example_projected_fs
  mount_point: /example_projected_fs_mount_point
  fstype: dvs
  opts: attrcache_timeout=14400,loadbalance,path=/
  example_projected_fs_mount_point,
  nodename=x3000c0s25b0n0:x3000c0s23b0n0
  state: mounted
```

Refer to DVS Modes and Choose a DVS Mode for guidance on choosing a parallel DVS mode.

4. Update the configuration used by the HPE Cray EX by performing Update CFS with Local Changes.
5. Reboot the DVS clients using the new BOS sessions.

   Replace *TEMPLATE_NAME* in the following command with the current BOS session templates used for CNs. Repeat this command as necessary for all applicable DVS client node types.

   ```
   ncn-w001# cray bos session create --template-uuid TEMPLATE_NAME --operation reboot
   ```

4.7.6 Project an External Filesystem to Compute Nodes or User Access Nodes

Perform this procedure to project either a read-only or read/write external file system to CNs and UANs with DVS.

Satisfy these prerequisites before performing this proceudre: - Verify that an external file system exists which can be physically attached to the HPE Cray EX system. - To project an IBM Spectrum Scale file system over DVS, first install the Spectrum Scale client software on the non-compute gateway nodes (NCN-GWs) according to the instructions at www.ibm.com.

External file systems can contain data such as user home directories and be NFS, IBM Spectrum Scale (formerly GPFS), Lustre, or others. HPE Cray EX Non-Compute Gateway Nodes (NCN-GWs) connect to the external file system and then use DVS to project it to the CNs and UANs. DVS can also project external file systems from one or more NCN Gateways to NCN Workers, such as an NCN Worker hosting a User Access Instance (UAI).

1. Configure the network interfaces on the NCN which will connect to the external file system.

   Refer to the site network documentation, this guide, and other HPE Cray EX documentation for details.

2. Perform Prepare to Configure DVS with VCS.

   Note that these instructions require changes to both the COS and UAN repositories.

   Make sure the UAN repository has a dedicated branch to support Ansible needed for the gateway nodes. These instructions assume the name of the branch is “integration-gateways”.

```
ncn-w# git checkout integration
Switched to branch 'integration'
Your branch is up to date with 'origin/integration'.
ncn-w# git checkout -b integration-gateways integration
Switched to a new branch 'integration-gateways'
```

Configure the external file system mount on both DVS servers and clients

3. Create a HSM group named dvs_gateways which contains the xnames of all the NCN Gateways which will serve the file system.

   Run this command to see if the “dvs_gateways” group already exists:

   ```
   cray hsm groups describe dvs_gateways
   ```

   If the group doesn’t exist, create it with this command:

   ```
   cray hsm groups create --label dvs_gateways --description "DVS Gateways"
   ```

   Add all the DVS Gateways to the group by running this command for each Gateway’s xname:

   ```
   cray hsm groups members create dvs_gateways --id XNAME
   ```

4. Configure the mount of the file system to be projected.

If the external file system in question requires mounting on the Gateway node via the `configure_fs` Ansible role during Node Personalization, follow the instructions in this step. An example of this type of mount might be a Lustre file system or NFS file system that requires IP routes to be added after boot during node personalization.

If the file system mount is accomplished by another means, you can skip this step. For example, if the file system is mounted by an `/etc/fstab` entry installed in the image during image customization, this step can be skipped.

Create a file in the UAN VCS repository which describes how to mount the external file system. Do this in the "integration-gateways" branch.

```
ncn-w# mkdir -p group_vars/dvs_gateways
ncn-w# vi group_vars/dvs_gateways/filesystems.yml
ncn-w# cat group_vars/dvs_gateways/filesystems.yml
filesystems:
- src: nfs.server.example.com:/home
  mount_point: /home
  fstype: nfs
  opts: defaults,noauto
  state: mounted
```

5. Update the `dvs_exports_yml` variable to allow access to the file system(s)

   Edit `group_vars/dvs_gateways/dvs.yml` and add the file systems to be projected to the `dvs_exports_yml` variable. You can set the mode to "ro" or "rw" depending on whether or not you want the file system to be exported Read-Only or Read-Write. See the ***Allowing Client Access to DVS Server File Systems*** section for more information.

   ```
   ncn-w# mkdir -p group_vars/dvs_gateways
   ncn-w# vi group_vars/dvs_gateways/dvs.yml
   ncn-w# cat group_vars/dvs_gateways/dvs.yml
   ...
   dvs_exports_yml:
     exports:
     - mode: rw
       path: /home
     - mode: rw
       path: /gpfs1
     - mode: ro
       path: /static-data
   ```

6. Perform Update CFS with Local Changes.

   This is for the UAN VCS repository, so no need to update the ncn-personalization CFS configuration. However, the CFS Configuration for the BOS template that boots the DVS Gateways needs to be updated.

7. Edit the Ansible files to mount the file system on CNs. Skip this step if the new file system will not be projected to CNs.

   1. Check the `group_vars/Compute` directory in the COS VCS repository for any files that contain `filesystems:` sections similar to the example in next sub-step.

   2. Add the client mount configuration for the external file system to either the `filesystems:` section of an existing file in the `group_vars/Compute` directory, if one was found in the previous sub-step, or a new YAML file within that directory.

      In the `opts` field, specify the xnames of the NCN-GWs (separated by colons) that will project the external file system as part of the `nodename` argument. The following example `filesystems:` section is for a CN mount of a remote `/home` directory, that is projected using the settings in Step 4.

      ```
      filesystems:
          - src: /home
            mount_point: /home
            fstype: dvs
            opts: noauto,path=/home,nodename=x3000c0s25b0n0:x3000c0s23b0n0
            state: mounted
      ```

This example mounts a remote read-only file system */READ_ONLY_FS* on the client at */READ_ONLY_FS_MOUNT_POINT*. Note the `attrcache_timeout=14400` and `loadbalance` mount options in the `opts` field, since this example is a read-only file system:

```
filesystems:
    - src:  /READ_ONLY_FS
      mount_point: /READ_ONLY_FS_MOUNT_POINT
      fstype: dvs
      opts: noauto,attrcache_timeout=14400,loadbalance,path=/READ_ONLY_FS_MOUNT_POINT,
      nodename=x3000c0s25b0n0:x3000c0s23b0n0
      state: mounted
```

Update the HPE Cray EX configuration with the changes

8. Perform Update CFS with Local Changes.
9. Edit the Ansible files to mount the file system on UANs. Skip this step if the new file system will not be projected to UANs.
    1. Perform "Create UAN Boot Images" in the UAN documentation, stopping before the step that calls for site-specific customizations.
    2. Check the `group_vars/Application` directory for any files that contain `filesystems:` sections similar to the example in Step 6.
    3. Add the client mount configuration for the external file system to either the `filesystems:` section of an existing file in `group_vars/Application`, if one was found in the previous sub-step, or a new YAML file within that directory.

       In the `opts` field, specify the xnames of the NCN-GWs (separated by colons) that will project the external file system as part of the `nodename` argument. The following example `filesystems:` section is for a UAN mount of a remote `/home` directory.

       ```
       filesystems:
           - src: /home
             mount_point: /home
             fstype: dvs
             opts: noauto,path=/home,nodename=x3000c0s25b0n0:x3000c0s23b0n0
             state: mounted
       ```

       This example mounts a remote read-only file system */READ_ONLY_FS_* ***on the client at*** `/READ_ONLY_FS_MOUNT_POINT_`. Note the `attrcache_timeout=14400` and `loadbalance` mount options in the `opts` field, since this example is a read-only file system:

       ```
       filesystems:
           - src:  /READ_ONLY_FS
             mount_point: /READ_ONLY_FS_MOUNT_POINT
             fstype: dvs
             opts: noauto, attrcache_timeout=14400,
             loadbalance,path=/READ_ONLY_FS_MOUNT_POINT,
             nodename=x3000c0s25b0n0:
             x3000c0s23b0n0
             state: mounted
       ```

    4. Resume the "Create UAN Boot Images" in the UAN documentation, beginning with the step after the site-specific customizations, but stopping before launching a CFS Image Customization session.

       This procedure uses Node Personalization, not Image Customization, to deploy the new configuration to the UANs. Therefore, there is no need to rebuild the UAN boot image.

Force the system to immediately use the new configuration

10. Update the BOS template for the NCN Gateways.

The NCN Gateways should have their own BOS template, separate from other Application nodes (such as UANs). Make sure the the template has `dvs_map=full` as part of one of the parameters in the `kernel_parameters` field. See the "BOS Session Templates" section of the HPE Cray EX CSM documentation.

11. Reboot the NCN Gateways to cause them to mount the external file system.

```
ncn-w001# cray bos session create --template-uuid UUID --operation reboot
```

12. Reboot the compute nodes, UANs, or both, depending on what types of nodes will be mounting the external file system.

    - Reboot the compute nodes with a custom image, replacing *UUID* with the wanted BOS template UUID:

      ```
      ncn-w001# cray bos session create --template-uuid UUID --operation reboot
      ```

    - Perform "Boot UANs" in the UAN documentation to reboot the UANs. Once the nodes finish rebooting, the DVS-projected external file system mount will persist across future CN, UAN, and NCN reboots.

13. If mounting DVS on the NCN Workers is needed, look at "Rerun NCN Personalization on an NCN" in the HPE Cray EX CSM documentation.

4.7.7 Quiesce a DVS-Projected File System

DVS quiesce is used to temporarily suspend traffic to a DVS-projected file system on any number of DVS servers. When a directory is quiesced, the DVS server completes any outstanding requests, but does not honor any new requests for that directory. When the quiesce is finished, administrators know it is safe to perform operations on the quiesced directory without any possibility of interference by a DVS client. DVS quiesce can be used when a file system needs to be repaired or to safely take a DVS server node out of service.

CAUTION: Because it may cause data corruption, do not use DVS quiesce to quiesce a directory on every DVS server.

An administrator must write into `/sys/fs/dvs/quiesce` to use DVS quiesce. Refer to the following use cases for more information:

- Quiesce a Single Directory on a Single DVS Server
- Quiesce All Directories on a DVS Server

4.7.7.1 How Quiesce Works: the User space Application View

User space applications have no visibility into any specific quiesce information. A quiesced directory will present in one of two ways:

- Entirely normally, if the directory is quiesced only on servers that the application is not using
- Useable but with degraded performance, if the application finds that its main server is quiesced and must query other servers

4.7.7.2 How Quiesce Works: the Server View

To provide the quiesce capability, DVS servers keep a list of quiesced directories and the current outstanding requests on each quiesced directory. When an admin requests that DVS quiesce a directory on a server, DVS does the following:

- Adds that directory to the list of quiesced directories on the server
- Iterates over all open files, closing any file that resides in the quiesced directory and setting a flag to indicate that the file was closed due to the directory being quiesced

When a DVS server receives a request from a client, DVS checks the request path against the list of quiesced directories. The comparison between the pathname in the request and the quiesced directory is a simple string compare to avoid any access of the underlying file system that has been quiesced. If DVS finds that the request is for a quiesced file system, it sends a reply indicating that the request could not be completed due to a quiesce and noting which directory is quiesced. If the client request is for a file that has been closed due to quiesce, the server returns a reply to the client indicating that the request could not be completed due to a quiesce.

When an admin unquiesces a directory on a DVS server, DVS simply removes that directory from the list of quiesced directories on the server and clears all quiesce-related flags for that directory.

4.7.7.3 How Quiesce Works: the Client View

When making a request of a server, a client may get back reply indicating that the request was for a file in a quiesced directory. The client then retries the operation on the next server in its server list. If it makes the request of every server in its server list and gets the same reply from each of them, then one of two things happens, depending on the type of request:

- **Pathname request**

  If the request is a pathname request (`lookup`, `stat`, file open, and so forth), then DVS reattempts the operation on a different server in a round-robin fashion until it finds a server that allows the operation to complete successfully.

- **Open file**

  If the request is for an open file (`read`, `write`, `lseek`, and so on), then DVS attempts the operation on a different server. If the file is not open on any other servers, DVS attempts to open on the file on a server in a round robin fashion until it gets a successful open. DVS will then attempt to perform the operation.

If a client receives a reply indicating a quiesced directory, the client adds that directory to a list of quiesced directories held on the DVS superblock. This is intended to reduce network traffic by avoiding requests that target quiesced directories. The list of quiesced directories on the client expires about every 60 seconds, thereby allowing clients to try those directories again in case one or more have been unquiesced during that time. This mechanism enables DVS to strike a balance between the timely unquiescing of a file system and a large reduction in network traffic and requests coming into the server. It also naturally staggers clients when they start to use a server.

4.7.8 Quiesce All Directories on a DVS Server

Remove a DVS server from service and let any outstanding requests complete first.

4.7.8.1 Prerequisites

- The administrator is logged into a DVS server
- Verify existing requests with List Outstanding DVS Client Requests

4.7.8.2 Procedure

1. Quiesce all directories on the DVS server.

   ```
   dvs1# echo quiesce / > /sys/fs/dvs/quiesce
   ```

   When the quiesce has completed, the shell prompt will return to user control. There is no command output.

2. Unquiesce all of the projected directories to allow traffic to the server to resume.

   ```
   dvs1# echo unquiesce / > /sys/fs/dvs/quiesce
   ```

4.7.9 Quiesce a Single Directory on a Single DVS Server

Quiesce a single directory on a DVS server. This is an unlikely use case, but it is an illustrative example. The example directory used in this procedure is `/gpfs/test/foo`.

4.7.9.1 Prerequisites

- The administrator is logged into a DVS server

4.7.9.2 Procedure

1. Quiesce the directory.

   ```
   dvs1# echo quiesce /gpfs/test/foo > /sys/fs/dvs/quiesce
   ```

   When the directory quiesce has completed, the shell prompt will return to user control. There is no command output.

2. When finished with the directory, unquiesce it.

   ```
   dvs1# echo unquiesce /gpfs/test/foo > /sys/fs/dvs/quiesce
   ```

3. Ensure that the directory is no longer on the quiesced list.

   ```
   dvs1# cat /sys/fs/dvs/quiesce
   ```

4.7.10 List Outstanding DVS Client Requests

Perform this procedure to list detailed information about any pending DVS client requests from client nodes in `/sys/kernel/debug/dvsipc/reques`

This file lists the following:

- The DVS server node
- The type of request
- The DVS file system path

- The `uid` of the user initiating the request
- The amount of time that the request has been waiting for a response

1. Print out the list of pending DVS requests.

   ```
   ncn-w002# cat /sys/kernel/debug/dvsipc/requests
   server: x3000c0s25b0n0 request: RQ_LOOKUP path: /cray_home user: 12795 time: 0.000 sec
   ```

4.7.11 Force a Cache Revalidation on a DVS Mount Point

Administrators can force a DVS cache revalidation at any time without having to wait for the attribute cache timeout to expire. If any content in a DVS-projected read-only file system changes (such as HPE Cray Programming Environment content), this procedure will enable DVS clients to see and use the new content immediately.

This procedure enables a system administrator with root privilege to force a cache revalidation at any time without having to wait for the timeout to expire—a process which can take up to four hours (14,400 seconds). This procedure is useful for shortening the time between when changes are made on a DVS-mounted file system (such as HPE Cray Programming Environment content) and when the new content can be used by the DVS client nodes.

1. Force a cache revalidation on a DVS client using one of the following methods:

   - To revalidate all cached attributes on a single DVS mount point, `echo 1` into the `drop_caches/sys` file of that mount point. The following example uses the second mount point on the client and uses the `cat` command first to confirm that is the wanted mount point. To specify a different mount point, replace the 2 with some other integer (0–n).

     ```
     cn# cat /sys/kernel/debug/dvs/mounts/2/mount
     cn# echo 1 > /sys/kernel/debug/dvs/mounts/2/drop_caches
     ```

   - To revalidate all attributes across all DVS mounts, `echo 1` into the universal DVS `drop_caches/sys` file. For example:

     ```
     cn# echo 1 > /sys/fs/dvs/drop_caches
     ```

2. **(Optional)** Clear out other caches on the DVS client to ensure that all data is revalidated.

   ```
   cn# echo 3 > /sys/fs/dvs/drop_caches
   ```

3. **(Optional)** Clear out other caches on the DVS ***server*** to ensure that all data is revalidated.

   If underlying file system is NFS, it is also likely that the same procedure will be required on the DVS servers to allow all the changes on the NFS file system to properly propagate out to the DVS clients. This is due to NFS caching behavior.

   ```
   ncn# echo 3 > /sys/fs/dvs/drop_caches
   ```

4.7.12 Change the Name of the DVS LNet Network

Avoid collisions between the LNet network used by DVS and the one used by an attached Lustre file system by changing the name of the DVS network.

- **PREREQUISITES**
  - Prepare to Configure DVS with VCS
- **OBJECTIVE**

  If the HPE Cray EX system is connected to a Lustre file system, use this procedure to change the name of the LNet network that DVS uses. Changing the name of the DVS LNet network will prevent conflicts with the LNet network used by Lustre.
- **LIMITATIONS**

  This requires downtime because the reboot step will cause those nodes to stop working until they are rebooted with the new image.

Perform Change LNet or DVS Parameters. While executing that procedure, update the files in the `cray_lnet_load` and `cray_dvs_load` roles as part of Step 2.

This must be done in the COS VCS repository for CNS and the UAN VCS repository for UANs. If the system has DVS Gateways (NCN-GWs), this must also be done in the “integration-gateways” branch of the UAN VCS repository.

1. Edit the files in `roles/cray_lnet_load/files/` so that `tcp99` is replaced by `tcpX` where `X` is an integer other than 99.
2. Edit the files in `roles/cray_dvs_load/files/` so that `tcp99` is replaced by `tcpXX` where `XX` is an integer other than 99.

4.7.13 Change LNet or DVS Parameters

To change the kernel module parameters used by DVS and LNet, clone the appropriate repository from VCS and modify the appropriate files. Then upload the updated repository, update CFS, reload DVS and LNet on NCNs, and finally reboot the NCN-GWs, CNs, and UANs.

- **OBJECTIVE**

  Perform this procedure to modify the kernel module parameters used by DVS or LNet on an HPE Cray EX system.

- **ADMIN IMPACT**

  This procedure must be performed to persistently modify any LNet and DVS kernel module parameters. This includes changing DVS modes, configuring modes for projecting new content, and DVS performance tuning.

1. Perform the procedure Prepare to Configure DVS with VCS to update the network name in HPE Cray EX system configuration.
2. Modify the VCS files that control the LNet or DVS configuration. Knowing which files to changes depends on what you're trying to accomplish. Often the section that refers you to this one will tell you what needs changing.
3. Perform the procedure Update CFS with Local Changes to update the network name in HPE Cray EX system configuration.
4. Run an IMS Image Customization sessions to create new images.

   New images need to be created for the CNs, UANs, and NCN-GWs (if they exist).

   Use the same configuration names used in the previous step and perform the procedure "Customize an Image Root Using IMS" in the HPE Cray EX CSM documentation.
5. Update the BOS template used for booting the nodes to use the new images.

   BOS templates need to be updated for the CNs, UANs, and NCN-GWs (if they exist). Repeat these steps for each template:

   - List all the BOS session templates using the following command. Then search for the BOS session template you want to update.

     ```
     ncn-w001# cray bos sessiontemplate list --format json
     ```

   - Download the current BOS session template into a file.

   Most of the output in the following example is omitted for clarity. The template name is `cos-compute-2.0.17` in following command example.

   ```
   ncn-w001# cray bos sessiontemplate describe --format json cos-compute-2.0.17 > \
   bos-template-cos-compute-2.0.17.json
   ncn-w001# cat bos-template-cos-compute-2.0.17.json
   {
     "boot_sets": {
       . . .
     },
     "cfs": {
       "configuration": "cos-config-2.0.17"
     },
     "enable_cfs": true,
     "name": "cos-compute-2.0.17"
   }
   ```

   - Update the "path" line of the template file with the new s3 URL you created in Step 4.

     ```
     ncn-w001# vi bos-template-cos-compute-2.0.17.json
     ```

   - Upload the new template.

     ```
     ncn-w001# cray bos sessiontemplate create --name cos-compute-2.0.17 --file \
     bos-template-cos-compute-2.0.17.json
     ```
6. Perform Reload DVS on an NCN Worker to reload LNet and DVS on the NCN Workers.
7. Reboot the NCN-GWs, CNs, and UANs (in that order).

   Replace *UUID* in the following command with ID of the BOS session template that uses the new NCN-GW images.

   ```
   ncn-w001#  cray bos session create --template-uuid UUID --operation reboot
   ```

4.7.14 Configure DVS Using `dvs.conf`

Use this procedure to apply different DVS configurations to different groups of nodes within an HPE Cray EX system.

Prerequisites - Refer to the CSM documentation for information on CFS, HSM, and VCS. - Perform Prepare to Configure DVS with VCS

The contents of each node's `/etc/modprobe.d/dvs.conf` file is generated from the Ansible `dvs_modprobe_conf` variable. You can set the variable for each type of node by setting it in `dvs.yml` file in the appropriate subdirectory of the `group_vars` directory in VCS. For example, you can set the `dvs.conf` file for the CNs by setting the `dvs_modprobe_conf` variable int the `group_vars/Compute/dvs.yml` file, like so:

```bash
ncn-m001# cat config-management/group_vars/Compute/dvs.yml
dvs_modprobe_conf: |
  options dvsipc_lnet lnd_name=tcp
  options dvsipc dvs_debug_mask=0xffffffff
```

The version of the `dvs_modprobe_conf` variable in the `group_vars/all` subdirectory applies to all the nodes unless there is a version in an applicable `group_vars/*` directory. If there is no definition of `dvs_modprobe_conf` in `group_vars`, the version in `roles/cray_dvs_load/defaults/main.yml` is used.

This procedure demonstrates how to use Ansible and Gitea to deploy different versions of the `dvs.conf` file to CNs and gateways. These two types of nodes can require different content in this file.

Perform Change LNet or DVS Parameters. While executing that procedure, make the changes below to the VCS repositories as part of Step 2.

This must be done in the COS VCS repository for CNS and the UAN VCS repository for UANs. If the system has DVS Gateways (NCN-GWs), this must also be done in the "integration-gateways" branch of the UAN VCS repository.

1. Navigate into the `group_vars/dvs_gateways` directory.

   ```
   ncn-m001# mkdir -p group_vars/dvs_gateways
   ncn-m001# cd group_vars/dvs_gateways
   ```

2. Edit the `dvs.yml` file to include the wanted DVS kernel module options the NCN-GWs. Each line must follow the syntax demonstrated in the following example as a guide. Replace `FOO` with a valid DVS kernel module parameter and `bar` with a valid value for that parameter.

   ```
   ncn-m001# vi dvs.yml
   ...
   ncn-m001# cat dvs.yml
   dvs_modprobe_conf: |
     options dvs FOO=bar
   ```

3. Repeat the previous two steps for all the node types in the system.

4. Add all the new files to the git index.

   "'bash ncn-m00# git add group_vars

5. Create the `dvs_gateways` host group. In this example, the host group only contains one node whose xname is x0c0s28b0.

   ```
   ncn-m001# cray hsm groups members create dvs_gateways --id x0c0s28b0
   ncn-m001# cray hsm groups create --label dvs_gateways --description "Gateways running DVS"
   ```

4.8 DVS Log Files

This section describes how to enable, disable, and reset DVS request logging and file system (FS) call logging. It also includes instructions on how to tune logging for optimized results.

4.8.1 DVS Request Logs

To aid in debugging problems that may arise, DVS request logging is enabled by default. To reduce the overhead to each client request, DVS logs only those requests that take more than a certain number of seconds to complete. For each request that exceeds the threshold, DVS

writes a single line to /sys/kernel/debug/dvs/request_log. That minimum threshold can be changed (default is 15 seconds) to see more or fewer requests in the log.

There are two ways to make changes to DVS request logging. When the system is running, an administrator can ssh to a node and make changes dynamically by writing values to certain /sys files. Alternatively, to set the behavior from the moment the DVS kernel modules load, administrators can set kernel module parameters prior to boot.

4.8.2 DVS File System (FS) Call Logs

DVS can log details about the calls that the DVS server node makes into the underlying file system. The log files are created in /sys/fs/dvs/fs_log and can provide the administrator with useful information, such as:

- The pid and name of the thread making the file system call
- The operation being executed (open, close, read, write, getattr, ...)
- The uid of the thread making the file system call, which DVS sets to the uid of the process on the DVS client
- The xname of the client node that sent the request to the server
- The path corresponding to the operation, if any

The file system operations logged are a larger set than the list of DVS requests. This is because it is sometimes necessary to make multiple and different file system calls to complete a single DVS request. For these secondary file system operations, the operation logged contains op[func] where op is the file system operation performed and func is the DVS function that contained the call. This lets the reader distinguish the primary from the secondary file system calls.

4.8.3 Disable or Re-enable DVS Logging

Disable or enable DVS request logging or FS call logging using kernel module parameters. When enabled, logging is useful for debugging problems with DVS.

Request and FS call logging are enabled by default.

Replace the PARAMETER_NAME value in the following commands with dvs_request_log_enabled for request logs, or dvs_fs_log_enabled for FS call logs.

1. Check if the logging type is enabled or disabled.

   To view the kernel module parameter in read-only mode:

   ```
   hostname# cat /sys/module/dvsproc/parameters/PARAMETER_NAME
   ```

2. Use the appropriate kernel module parameter to enable or disable DVS request or FS call logging.

   Refer to the Configure DVS Using dvs.conf section of the HPE Cray Operating System Administration Guide: CSM on HPE Cray EX Systems for instructions on how to add this change on all nodes.

   Use one of the following options to enable/disable DVS logging dynamically or prior to boot.

   - Edit the /etc/modprobe.d/dvs-local.conf file and add one of the following lines prior to boot:

     To disable DVS request logging:

     ```
     # Disable DVS request log
     options dvsproc PARAMETER_NAME=0
     ```

     To enable DVS request logging:

     ```
     # Enable DVS request log
     options dvsproc PARAMETER_NAME=1
     ```

   - Dynamically:

     Replace LOG_TYPE with request_log or fs_log depending on the type of log in use.

     0 disables the log:

     ```
     hostname# echo 0 > /sys/kernel/debug/dvs/LOG_TYPE
     ```

     1 enables the log:

     ```
     hostname# echo 1 > /sys/kernel/debug/dvs/LOG_TYPE
     ```

4.8.4 Reset the DVS Log

Reset the DVS log to remove logs that are no longer needed and to make it easier to search when debugging current issues.

Dynamically reset the DVS request log or FS call log with the following command:

Replace `LOG_TYPE` with `request_log` or `fs_log` depending on the type of log in use.

```
hostname# echo 2 > /sys/kernel/debug/dvs/LOG_TYPE
```

4.8.5 Read the DVS Log

Use the following command to view the DVS request log or FS call log:

Replace `LOG_TYPE` with `request_log` or `fs_log` depending on the type of log in use.

```
hostname# cat /sys/kernel/debug/dvs/LOG_TYPE
```

The following is example output from the DVS request log:

```
2022-1-31 16:22:01-UTC: pid=2302223 cmd=sadc path=/var/lib/cps-local/boot-images/
a4d33fad-0107-4c77-8f26-b0715b25ce7a type=dir req=RQ_STATFS count=1 node=x3000c0s9b0n0 time=60.064
2022-1-31 16:23:01-UTC: pid=2302223 cmd=sadc path=/var/
lib/cps-local/boot-images/PE type=dir req=RQ_STATFS count=1 node=x3000c0s9b0n0 time=60.016
2022-1-31 16:40:01-UTC: pid=2564090 cmd=sadc path=/var/lib/cps-local/boot-images/
a4d33fad-0107-4c77-8f26-b0715b25ce7a type=dir req=RQ_STATFS count=1 node=x3000c0s9b0n0 time=60.036
2022-1-31 16:41:01-UTC: pid=2564090 cmd=sadc path=/var/lib/cps-local/boot-images/PE type=dir req=RQ_STATFS
count=1 node=x3000c0s9b0n0 time=60.016
2022-2-5 11:52:14-UTC: pid=217823 cmd=sadc path=/var/lib/cps-local/boot-images/
a4d33fad-0107-4c77-8f26-b0715b25ce7a type=dir req=RQ_STATFS count=1 node=x3000c0s9b0n0 time=252.824

[...]
```

Depending on the log type, a combination of the following information will be included in each line of the log:

- date/time

  Date and time (in UTC) of request.

- pid

  User process ID of the process initiating the request.

- cmd

  Command being executed by the user process ID.

- op

  The operation being executed (open, close, read, write, getattr, ...).

- path

  Path on the server node that is being referenced.

- type

  Type of path (file, directory, link, etc.).

- req

  Type of DVS request.

- count

  Number of servers the request was sent to. Some DVS client operations send a request to a single server, and others send a request to multiple servers. The former will log a line with count=1, and the latter will log a line with count=N, where N is the number of servers the request was sent to. When the request has been sent to multiple servers, the single line in the log is a summary of the requests.

- node

  Node is set to the node that the DVS request was sent to. If multiple nodes were targeted (i.e., the count field is > 1), then the value displayed is [multiple].

- time

  Time is set to the amount of time in seconds it took for the request to be sent to the server, execute, and for the reply to be received by the client. Granularity is milliseconds, so 0.000 is displayed for requests that take less than a millisecond.

4.8.6 Tune DVS Logging

Alter how DVS request logs and FS call logs are captured with kernel module parameters.

4.8.6.1 Buffer Size

The default buffer size for both log types is 16384 KB (16384 * 1024 bytes).

Replace the `PARAMETER_NAME` value in the following commands with `request_log_size_kb` for request logs, or `fs_log_size_kb` for FS call logs.

1. Determine the size (KB) of the log buffer.

   To determine the current buffer size, `cat` the file. For example:

   ```
   hostname# cat /sys/kernel/debug/dvs/PARAMETER_NAME
   16384
   ```

   Alternatively, the following command can be used to view in read-only mode:

   ```
   hostname# cat /sys/module/dvsproc/parameters/dvs_PARAMETER_NAME
   ```

2. Update the size of the request log with the appropriate kernel module parameter.

   Use one of the following options to update the size of the request log dynamically or prior to boot.

   - To change prior to boot, add these lines to `/etc/modprobe.d/dvs-local.conf`:

     ```
     # Set size (in kb) of the request log buffer
     options dvsproc dvs_PARAMETER_NAME=17000
     ```

   - To change dynamically:

     ```
     hostname# echo 17000 > /sys/kernel/debug/dvs/PARAMETER_NAME
     ```

4.8.6.2 Log Time

DVS only logs requests that take more than a certain number of seconds to complete. For each request that exceeds the threshold, DVS writes a single line to `/sys/kernel/debug/dvs/request_log` or `/sys/kernel/debug/dvs/fs_log`. Use this procedure to set the minimum threshold to see more or fewer requests in the log.

The default threshold for both log types is 15 seconds.

Replace the `PARAMETER_NAME` value in the following commands with `request_log_time_min_secs` for request logs, or `fs_log_time_min_secs` for FS call logs.

1. View the current threshold of time for data to be logged.

   ```
   cat /sys/module/dvsproc/parameters/dvs_PARAMETER_NAME
   ```

2. Update the time threshold with the appropriate kernel module parameter.

   Use one of the following options to update the time threshold dynamically or prior to boot.

   - To change prior to boot, add these lines to `/etc/modprobe.d/dvs-local.conf`:

     ```
     # Set threshold (in seconds) for time a DVS request
     # takes before logged. Requests taking fewer seconds
     # will not be logged.
     options dvsproc dvs_PARAMETER_NAME
     ```

- To change dynamically:

  ```
  hostname# echo 15 > /sys/kernel/debug/dvs/PARAMETER_NAME
  ```

4.9 Troubleshoot DVS

This topic contains several commands for gathering useful information for troubleshooting DVS.

In addition to the specific methods listed here, use DVS statistics for troubleshooting and root cause analysis. See Collect and Analyze DVS Statistics for more details.

4.9.1 Slow Compute Node (CN) Start-Up

DVS uses DNS to look up the node IP address while constructing the node map for DVS' operation. Therefore, the correct name servers must be used and included in the `/etc/resolv.conf` file of the node. Otherwise, compute node (CN) boots and reboots may experience problems or long delays.

4.9.2 Identify the Node Running the CPS-CM-PM Pod

Query Kubernetes to identify the worker node (NCN) which hosts the pod that works with DVS on NCNs and runs CPS. In the following example, that node is `ncn-w001`.

```
ncn-w001# kubectl get pods -n services -o wide | grep -i cps-cm-pm
cray-cps-cm-pm-442ph     5/5     Running     0     14h     10.40.0.69   ncn-w001   <none>     <none>
cray-cps-cm-pm-6pjrh     5/5     Running     0     14h     10.46.0.52   ncn-w002   <none>     <none>
cray-cps-cm-pm-qbww7     5/5     Running     0     14h     10.41.0.58   ncn-w003   <none>     <none>
```

4.9.3 Verify All Necessary DVS Kernel Modules for the Running Kernel Are Installed

Use the following command to confirm that all four DVS kernel modules are installed in the `/lib/modules/` directory for the running kernel.

The four DVS kernel modules are: - `dvsipc.ko` - `dvs.ko` - `dvskatlas.ko` - `dvsproc.ko` - `dvsipc_lnet.ko`

```
ncn-w001# find /lib/modules/`uname -r` -name dvs* -ls
```

4.9.4 Confirm All Necessary DVS Kernel Modules Are Loaded

After confirming that all the DVS kernel modules are installed, run the following command to verify that they and the `lnet` kernel module are loaded.

```
ncn-w001# lsmod | grep dvs
dvs                   405504  0
dvsipc                172032  2 dvs
dvsipc_lnet            57344  1 dvsipc
dvsproc               143360  3 dvsipc,dvsipc_lnet,dvs
craytrace              20480  5 dvsipc,dvsproc,dvsipc_lnet,dvs
dvskatlas              20480  4 dvsipc,dvsproc,dvsipc_lnet,dvs
lnet                  634880  3 ksocklnd,dvsipc_lnet
libcfs                512000  3 ksocklnd,lnet,dvsipc_lnet
```

4.9.5 Print Out the DVS Node Map

Print out the file DVS uses as the list of all nodes on the HPE Cray EX system. This list contains the mapping of xnames to IP addresses used by DVS to route traffic.

```
ncn-w001# cat /proc/fs/dvs/node_map
...
800024 x3000c0s10b0n0    10.252.0.12@tcp99
800032 x3000c0s12b0n0    10.252.0.4@tcp99
800040 x3000c0s14b0n0    10.252.0.5@tcp99
800048 x3000c0s16b0n0    10.252.0.6@tcp99
800064 x3000c0s20b0n0    10.252.0.8@tcp99
800072 x3000c0s22b0n0    10.252.0.9@tcp99
```

```
     8 x3000c0s24b1n0    10.252.0.28@tcp99
    16 x3000c0s24b2n0    10.252.0.33@tcp99
    24 x3000c0s24b3n0    10.252.0.27@tcp99
    32 x3000c0s24b4n0    10.252.0.26@tcp99
800008 x3000c0s6b0n0     10.252.0.10@tcp99
800016 x3000c0s8b0n0     10.252.0.11@tcp99
```

4.9.6 Search for DVS-Related Kernel Messages

Search for log messages from DVS in `/var/log/messages`.

```
ncn-w001# grep -Pi 'sdvs|dvs_orca_client' /var/log/messages
. . .
2020-08-19T00:27:23.976541+00:00 ncn-w001 kernel: [679557.849179] DVS: message size checks complete.
2020-08-19T00:27:24.352686+00:00 ncn-w001 systemd[1]: Started Cray DVS ORCA client.
2020-08-19T00:27:24.356257+00:00 ncn-w001 dvs_orca_client[4040305]: running
```

4.9.7 List All Running DVS Processes

Run the following command to list all running DVS processes:

```
ncn-w001# ps ax | grep -i dvs | grep -v 'grep'
```

4.9.8 Troubleshoot Node Map IP Change Issues

DVS can dynamically add new nodes into the node map, but does not process changes in existing node map entries. For example, if the IP address of a node changes, already configured nodes will not see that the new address of that node until DVS is reloaded.

If the compute node IP addresses change, the node will stop responding right after a `cpsmount.sh` command. The console look will look like:

```
dracut-initqueue[1817]: cps: All requested interfaces are UP, proceeding.
dracut-initqueue[1817]: LNet: loaded lnet module.
dracut-initqueue[1817]: LNet: lnet module configured.
dracut-initqueue[1817]: DVS: lnet setup completed.
dracut-initqueue[1817]: DVS: node map generated.
dracut-initqueue[1817]: DVS: loaded dvsproc module.
dracut-initqueue[1817]: DVS: loaded dvs module.
dracut-initqueue[1817]: mount is: /opt/cray/cps-utils/bin/cpsmount.sh -a api-gw-service-nmn.local -t
dvs -T 300 -i eth0 -e 457ca59e694fcd0110a42d9b764128de-176
s3://boot-images/21117f1e-f42e-411e-9f5a-d466cafc0c25/rootfs /tmp/cps
```

The NCN `dmesg` output will contain messages like:

```
2020-07-21T18:01:56.400796+00:00 ncn-w002 kernel: [101229.684364] DVS: merge_one#351: New node map entry
does not match the existing entry
2020-07-21T18:01:56.400818+00:00 ncn-w002 kernel: [101229.684365] DVS: merge_one#353: nid: 24 -> 24
2020-07-21T18:01:56.400818+00:00 ncn-w002 kernel: [101229.684365] DVS: merge_one#355: name: 'x3000c0s19b3n0'
-> 'x3000c0s19b3n0'
2020-07-21T18:01:56.400819+00:00 ncn-w002 kernel: [101229.684366] DVS: merge_one#357: address:
'10.252.0.29@tcp99' -> '10.252.0.31@tcp99'
2020-07-21T18:01:56.400819+00:00 ncn-w002 kernel: [101229.684366] DVS: merge_one#358: Ignoring.
```

If there are just a few nodes with changed addresses, fix this issue without causing systemwide DVS service disruption:

1. Perform Reload DVS on an NCN Worker on each worker NCN, one at a time, to restart DVS on those nodes. This script unloads the map on each worker node and reloads the new one. Wait for the DVS modules to unload and reload. Space out the restarts, so that the running compute nodes will fail over from the restarting node to a running one. The compute node will not ever suffer complete disruption.
2. Reboot the nonresponsive compute nodes. At this point, the NCNs and problem compute nodes will have matching values in their map and DVS will operate normally.

If that does not fix the problem, follow the instructions in the “Clear HSM Tables to Resolve DHCP Issues” in the CSM documentation.

4.10 DVS Reference Information

This section contains content that explains DVS behavior and details DVS configuration options.

4.10.1 DVS Modes

Some DVS mount option combinations are common enough to be characterized as “modes” of use. There are several parallel DVS modes and one serial mode.

4.10.1.1 Serial vs parallel DVS modes

A DVS ***mode*** is a name given to a combination of mount options which satisfies a common use case. These modes allow customers to tune DVS behavior as wanted to increase performance, ensure POSIX read/write atomicity, or both. There are two general types of DVS modes:

- **Serial:** One DVS node projects a file system to multiple client nodes. There is only one serial mode. See DVS Serial Mode.
- **Parallel:** multiple DVS nodes project a file system to client nodes. There are several ways to configure DVS servers to work in parallel to project a file system and thus multiple parallel modes. For more information on each of these parallel modes, see following sections:
  - DVS Loadbalance Mode
  - DVS Cluster Parallel Mode
  - DVS Stripe Parallel Mode
  - DVS Atomic Stripe Parallel Mode

4.10.1.2 A single projected file system can use multiple modes

The HPE Cray EX system administrator can set the DVS mount options for non-CPS file systems during initial system configuration and when the system is configured to project additional (for example, external) file systems over DVS. The administrator can make several DVS modes for the same file system available on the same client node. Administrators can accomplish this by creating multiple entries for the same underlying file system in the `configure_fs` Ansible role, with different mount options for each entry. This will cause the affected DVS clients to mount the same file system with different mount options on different mount points.

Non-root users cannot choose among DVS modes unless the system administrator has configured the system to make more than one mode available.

- **DVS Serial Mode**
  Serial mode is the simplest implementation of DVS and the only option if no cluster or shared file system available. In this mode, a single DVS server node handles all I/O traffic for the projected file system.
- **DVS Cluster Parallel Mode**
  DVS Cluster Parallel mode distributes I/O among multiple servers by assigning each projected file to a single server. All file data and metadata I/O for a given file is handled by the DVS server assigned to that file.
- **DVS Stripe Parallel Mode**
  DVS Stripe Parallel Mode can provide greater I/O performance than Cluster Parallel Mode. Stripe Parallel Mode, however, has restrictions that Cluster Parallel Mode does not.
- **DVS Atomic Stripe Parallel Mode**
  DVS Atomic Stripe Parallel Mode provides both parallel file data I/O and POSIX read/write atomicity compliance.
- **DVS Loadbalance Mode**
  DVS Loadbalance Mode combines caching with the load distribution. This mode can only be used on read-only file systems. All file systems projected by CPS use this mode.

4.10.1.3 DVS Serial Mode

4.10.1.3.1 How Serial Mode Operates

In this mode, a single DVS server node handles all I/O traffic from all clients (usually compute nodes) for the projected file system. DVS can project multiple file systems from the same server by using the `maxnodes=1` option for each client mount.

4.10.1.3.2 Advantages of Serial Mode

Serial Mode is the only DVS mode that can be used if there is no underlying cluster or shared file system available. This mode is the simplest implementation of DVS, since it requires only one DVS server node to project a file system.

This mode guarantees that all bytes associated with a read or write are not interleaved with bytes from other read or write operations. In other words, DVS Serial mode adheres to POSIX read/write atomicity rules.

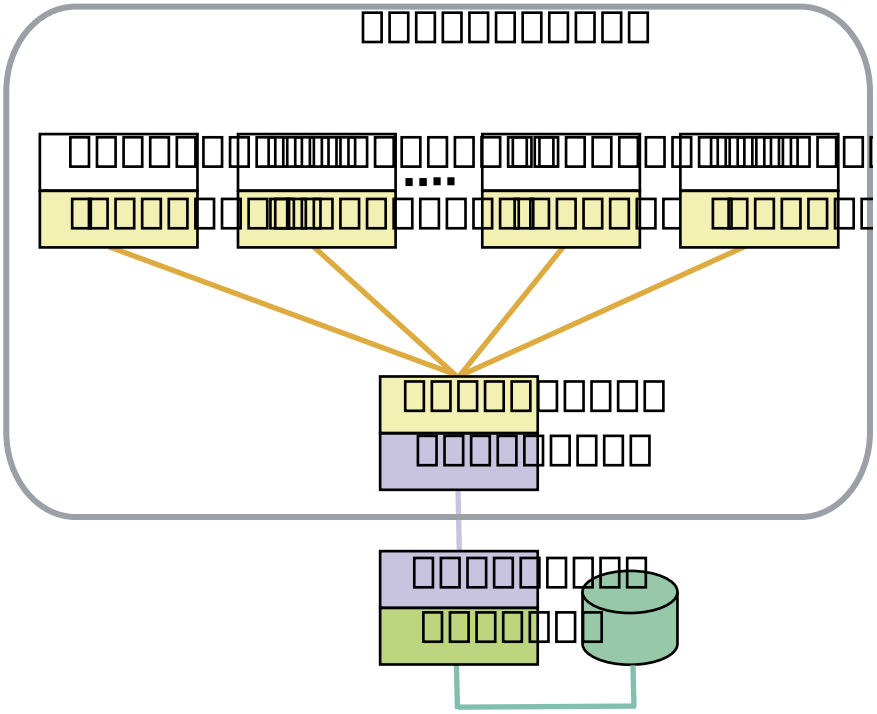


Figure 2: diagram showing a single DVS server using Serial Mode to project to clients

4.10.1.3.3 Limitations of Serial Mode

DVS Serial Mode does not provide any failover or failback resiliency, and I/O performance is limited to what a single DVS server node can provide.

4.10.1.4 DVS Cluster Parallel Mode

4.10.1.4.1 How Cluster Parallel Mode Operates

DVS Cluster Parallel mode distributes file I/O and metadata operations among multiple servers by assigning each projected file to a single server. All file data and metadata I/O for a given file is handled by the DVS server assigned to that file. I/O for a single file goes only to the chosen server. Therefore, each DVS client interacts with multiple servers to use the entire projected file system. This mode is often used for large file systems.

The following figure illustrates how Cluster Parallel Mode works. DVS is mounted on `/foo` on the DVS client, and three different files—`bar1`, `bar2`, and `bar3`—are handled by three different DVS servers (nodes), thus distributing the load. The server used to perform I/O or metadata operations on a file is chosen by an internal hash on the underlying file or directory inode number. Once a server has been selected for a file, Cluster Parallel Mode looks like Serial Mode: All I/O and metadata operations for that file from all clients use the selected server.

4.10.1.4.2 Advantages of Cluster Parallel Mode

DVS Cluster Parallel Mode adheres to POSIX read/write atomicity rules. That is, all bytes associated with a read or write are not interleaved with bytes from other read or write operations.

This mode improves performance while preventing any one server from becoming overloaded. Also, the way Cluster Parallel Mode distributes I/O prevents file system coherency thrash.

4.10.1.4.3 Limitations of Cluster Parallel Mode

Only a shared file system such as IBM Spectrum Scale (formerly “GPFS”) or Lustre can be projected through DVS Cluster Parallel Mode. Therefore, Cluster Parallel Mode cannot be used to project an NFS file system.

4.10.1.5 DVS Stripe Parallel Mode

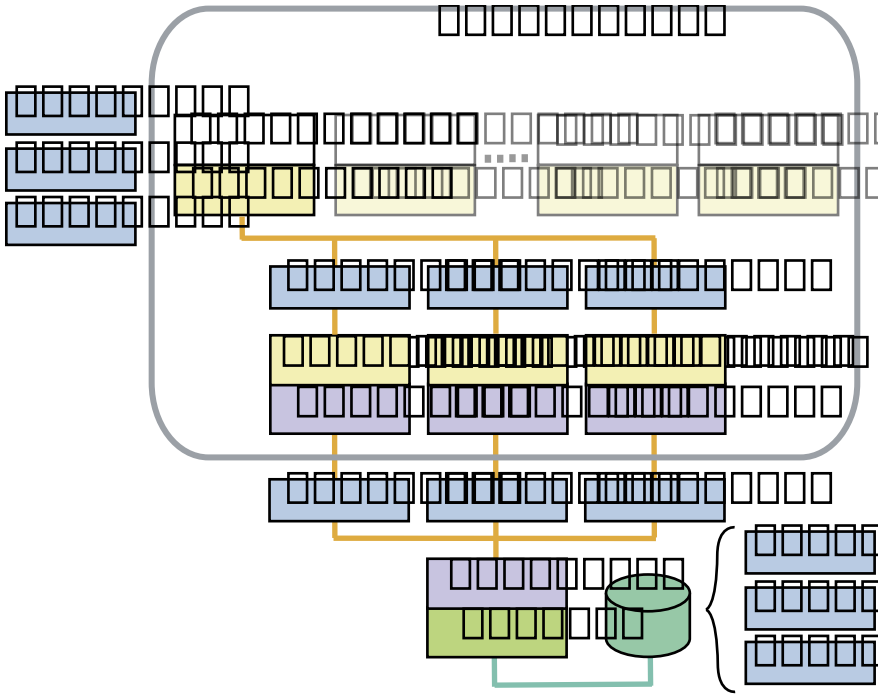


Figure 3: Diagram showing multiple DVS servers each projecting a different file from the same file system

4.10.1.5.1 How Stripe Parallel Mode Operates

Stripe Parallel Mode builds upon Cluster Parallel Mode to provide an extra level of I/O forwarding parallelization. Each DVS server (node) can serve all files. DVS servers are selected based on the file inode and offsets of data within the file relative to the DVS block size value (`blksize`).

For example, in the following figure, DVS is mounted to /foo on the DVS client. The I/O for three different blocks (or segments) of data within file `bar`—seg1, seg2, and seg3—is handled by three different DVS servers. Thus Stripe Parallel Mode distributes the load at a more granular level than that achieved by cluster parallel mode.

4.10.1.5.2 Advantages of Stripe Parallel Mode

Stripe Parallel Mode can provide greater aggregate I/O bandwidth than Cluster Parallel Mode when forwarding I/O from a coherent cluster file system. IBM Spectrum Scale has been tested extensively using this mode.

DVS routes all data I/O for each block of file data for the same file from all clients to the same server. Since only one DVS server reads or writes to a given block, file system coherency thrash is prevented. While file I/O is distributed at the block level, however, file metadata operations are distributed as in Cluster Parallel Mode: the metadata operations of a given file are always handled by the same DVS server.

4.10.1.5.3 Limitations

NFS cannot be used in Stripe Parallel Mode because NFS implements close-to-open cache consistency. Therefore, striping data across the NFS clients could compromise data integrity. Also, Stripe Parallel Mode **does not** adhere to POSIX read and write atomicity rules. Therefore applications which require such atomic reads and writes will have to implement their own file locking.

4.10.1.6 DVS Atomic Stripe Parallel Mode

DVS Atomic Stripe Parallel Mode provides both parallel file data I/O and POSIX read/write atomicity compliance.

4.10.1.6.1 How Atomic Stripe Parallel Mode operates

Atomic Stripe Parallel Mode uses the same method as Stripe Parallel Mode to select a server. Atomic Stripe Parallel Mode, however, uses only that selected server for the entire read or write operation. Thus, all I/O operations are atomic. Atomic Stripe Parallel Mode allows DVS clients to access different servers for subsequent I/O requests if they have different starting offsets within the file.

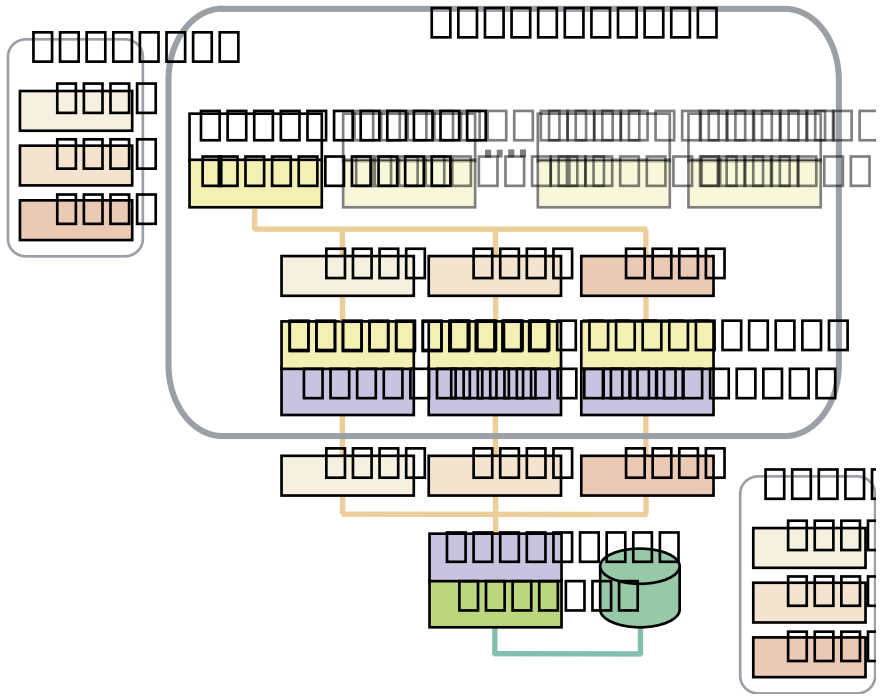


Figure 4: Diagram showing different file segments of a file and each one is projected from a different DVS server

Whereas Stripe Parallel Mode provides parallelism within a file at the granularity of the DVS block size, Atomic Stripe Parallel Mode provides I/O granularity based on the size of the application writes.

4.10.1.6.2 Advantages of Atomic Stripe Parallel Mode

DVS Atomic Stripe Parallel mode adheres to POSIX read and write atomicity rules while still allowing for possible parallel file data I/O. This mode allows applications which do not implement their own file locking to enjoy the performance benefit of parallel file data I/O.

4.10.1.6.3 Disadvantages of Atomic Stripe Parallel Mode

NFS cannot be projected using Atomic Stripe Parallel Mode.

4.10.1.7 DVS Loadbalance Mode

4.10.1.7.1 How Loadbalance Mode operates

The client nodes automatically select the server based on a DVS-internal node ID (NID) from the list of available server nodes.

DVS automatically enables the `cache` mount option in Loadbalance Mode because using cache on a read-only mount can improve performance. The first time a DVS client requests file data from a DVS server, the server retrieves that data. The client then stores it in the page cache. While the application is running, all future references to that data are local to the memory of the client, and DVS will not be involved at all. However, if the node runs low on memory, the Linux kernel may remove these pages. Then the client must fetch the data from the DVS server on the next reference to repopulate the page cache of the client.

CPS uses DVS Loadbalance Mode to project the root file system and the HPE Cray Programming Environment to compute nodes.

4.10.1.7.2 Advantages of Loadbalance Mode

DVS Loadbalance Mode combines fast, cached access to read-only file systems with load distribution across multiple DVS servers. Loadbalance Mode evenly distributes loads across servers.

4.10.1.7.3 Limitations of Loadbalance Mode

DVS Loadbalance Mode can only project read-only file systems.

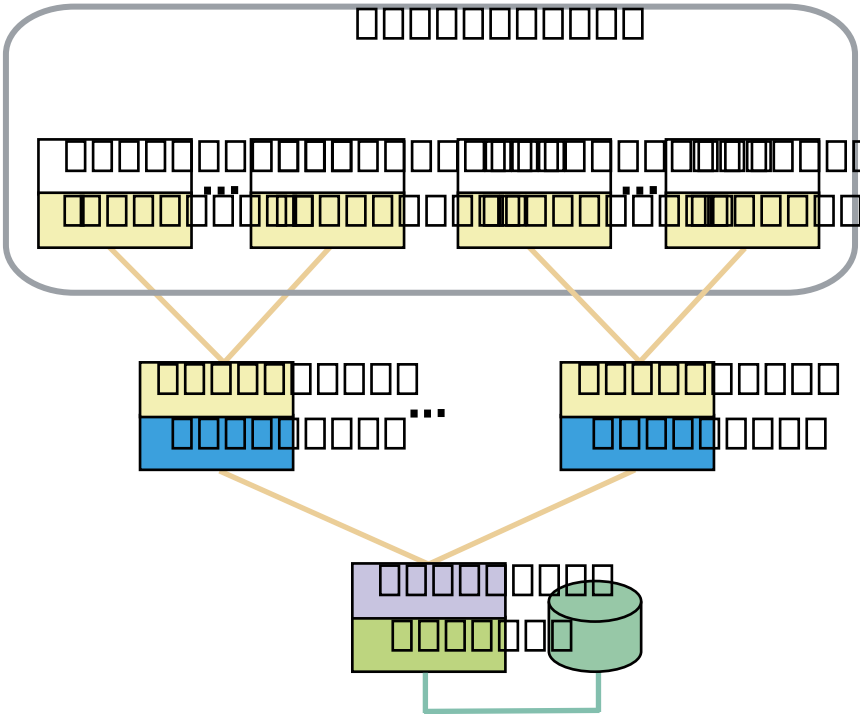


Figure 5: diagram showing multiple DVS servers projecting the same file system to different groups of clients

4.10.2 DVS Failover and Failback

DVS resiliency is provided by Kubernetes and ORCA. For cluster, stripe, and atomic stripe parallel modes, failover and failback must be enabled by the customer. Specific failover/failback behavior depends on the DVS mode used by the client.

4.10.2.1 How DVS provides resiliency

DVS clients use ORCA events to determine when server nodes have failed, when DVS has been unloaded from a server node, and when server nodes have been booted and DVS is reloaded. This ensures that all clients are informed of server failures and reboots in the same manner at the same time, which reduces the underlying file system coherency traffic associated with rerouting I/O operations away from downed servers and back to rebooted servers.

DVS supports failover and failback in an active-active manner for all modes except serial mode. Multiple `cps-cm-pm` pod instances, multiple active DVS server nodes, Kubernetes, and ORCA work together to provide resiliency for DVS.

4.10.2.2 How DVS handles failover

When a server fails, it is taken out of the list of servers to use for the client mount until it is rebooted. All open and new files use the remaining servers. Files not using the failed server are not affected.

DVS failover behavior is dependent on the DVS mode used:

- **Loadbalance Mode:** Any mount point using loadbalance mode automatically recalibrates the existing client-to-server routes to ensure that the clients are evenly distributed across the remaining servers. When failback occurs, this process is repeated.
- **Cluster Parallel Mode:** Any mount point using cluster parallel mode automatically redirects I/O to one of the remaining DVS servers for any file that previously routed to the now-down server. When failback occurs, files are rerouted to their original server.
- **Stripe Parallel or Atomic Stripe Parallel Modes:** Any mount point using stripe parallel mode or atomic stripe parallel mode automatically restripes I/O across the remaining DVS servers in an even manner. When failback occurs, files are restriped to their original pattern.

If all servers fail, I/O is retried as described by the retry client mount option. See DVS Mount Options for more information about the DVS retry mount option.

4.10.3 About the Close-to-Open Coherency Model

The only times the client cached copy of a file is guaranteed to match the copy stored by the DVS server are when the file is closed and when it is opened. This aspect of DVS read/write caching is important to understand so that data corruption issues can be avoided when write-back caching is enabled. Also, kernel page granularity affects coherency.

The DVS read/write cache capability provides a close-to-open coherency model, which means:

- File read operations are only guaranteed to see file data that was available on the server at the time the file was opened.
- Write data cached on the client is not guaranteed to be written back to, and thus visible on, the DVS server and backing file system storage until file close time.

This does not imply that server data newer than file open time cannot be read by the client or that some amount of client write data will not be written to the server prior to file close. Rather, that file open and close are the only times this can be **guaranteed**. To preserve the close-to-open coherency, any remaining 'dirty' (that is, not yet written back) cached client data for a file is written back to the DVS server at file close time, and any cached client data is invalidated at file open if the file on the server is newer than the cached data. This coherency model is similar to that provided by NFS.

4.10.3.1 How Granularity Affects Coherency

The kernel VFS interface provides caching with the granularity of a kernel page, which is 4 KB for the SLES15 kernel. This implies that 4 KB is the smallest amount of data that is stored in cache and that data is read from and written to the server in a minimum of 4 KB blocks. This page granularity affects coherency. Shared file access from different DVS clients is coherent only at page cache size boundaries. If two DVS clients attempt to write within the boundary of the same kernel page, coherency cannot be preserved, even if the two clients are writing nonoverlapping byte ranges within that page. This is because both clients would attempt to cache and write back the entire page, unable to see the data from the other client and ultimately conflicting with each other when writing back their cache pages. Cache access from separate clients can preserve coherency if they maintain the cache page size boundaries. If the separate clients each only write and cache distinct pages, then there will be no conflicts on the server when the pages are written back.

4.10.4 DVS Client-side Write-back Caching can Yield Performance Gains

How and under what circumstances DVS client-side write-back caching improves file I/O performance.

4.10.4.1 Limitations of Client-side Write-back Caching

Using the page cache may not provide a benefit for all applications. Applications that require large reads or writes may find that introducing the overhead of managing the page cache slows down I/O handling. Benefit can also depend on write access patterns: small, random writes may not perform as well as sequential writes since pages are aggregated for write-back. If random writes do not access sequential pages, then less-than-optimal-sized write-backs may have to be issued when a break in contiguous cache pages is encountered.

4.10.4.2 How Client-side Caching Increases Performance

With the advent of faster backing storage (for example, SSDs), the overhead of network operations has become an increasingly large portion of overall file system operation latency. DVS can cache both read and write data on a client node while preserving close-to-open coherency and without contributing to out-of-memory issues on compute nodes. Instead of using network communication for all read/write operations, DVS can aggregate those operations and reuse data already read by or written from a client. This can provide a substantial performance benefit for I/O patterns which typically bear the additional cost of network latency:

- Small reads and writes
- Reads following writes
- Multiple reads of the same data

4.10.4.3 How Client-side Caching Works

Client-side caching of both read and write data is implemented as a write-back type of cache. This type of cache allows write data to be targeted to local cache on the client, aggregating the data before it is later 'written back' to the backing file system storage on a DVS server, thus providing low latency and high throughput for write operations. Aggregation of write data allows the DVS client to wait until a larger amount of data must be written, and to then write all the data with a single network operation rather than a network operation for each write performed by an application. This enables an application to complete writes more quickly while reducing overall network traffic and the total utilization load on DVS servers. A write-back cache also enables caching of read data and reuse of previously read or written data in the cache.

DVS does not provide perfect cache coherency across disparate client nodes. Instead, it provides close-to-open coherency. See About the Close-to-Open Coherency Model for more information.

DVS write-back cache uses the standard Linux VFS page-cache mechanism and address-space operations interface, but adds its own triggers for write-back scheduling. The Linux kernel-provided functionality writes back cached data and reclaims cache pages as memory pressure increases or when a specified time limit is reached. DVS augments this by tracking what portions of the locally cached file have been modified by writes. When an optimal amount of contiguous data has been written to the cached copy, DVS immediately writes back the pages comprising the modified region of the file using a single DVS remote memory access (RMA) operation.

This provides a more steady and optimal flow of data to DVS servers, and it helps to avoid large amounts of cached data waiting to be written at file close time or when the cache timeout is reached. This DVS-provided functionality also helps to minimize the risk of large amounts of cached data contributing to out-of-memory issues on client nodes, because kernel memory management is able to reclaim the 'clean' (that is, already written back) pages immediately without having to attempt to start a write-back of the pages to the server.

4.10.5 DVS Fairness of Service

By default, DVS services incoming I/O requests in a way which prevents demanding users or clients from degrading the I/O performance of others. Administrators can disable DVS Fairness of Service if absolute application performance is wanted.

DVS creates user- or job-specific request queues for clients. Originally, DVS used one queue to handle requests in a First In, First Out (FIFO) fashion. Since all clients shared one queue, a demanding job could tax the server disproportionately and the other clients would have to wait until the requests of the demanding client are completed. DVS Fairness of Service solves that problem by creating a ***list*** of queues—one queue for each client and job. The list of queues is processed in a circular fashion. When a message thread is available, it fetches the first queue on the list, moves that queue to the end of the list, and processes the first message in that queue. This helps to distribute the workload and potentially helps contending applications perform better.

Fairness of Service is enabled by default in DVS, but administrators can disable it. Disabling Fairness of Service reverts DVS to using a single message queue for all incoming requests, causing all requests to be processed in the order they were received. For demanding applications which issue a disproportionately large number of requests, this could increase their performance as all their requests can be processed in order.

4.10.6 HPE Cray EX DVS Configuration Overview

In HPE Cray EX supercomputers, DVS is configured mainly by editing three configuration files with the configuration management Git version control system (VCS).

The Configuration Framework Service (CFS), configuration management Git repository, and component groups managed by the Hardware State Manager (HSM), are used to configure DVS. Refer to the CSM documentation for more information on HPE Cray EX configuration management, CFS, and component groups.

For both CNs and NCNs, LNet and DVS are configured through Ansible roles run under CFS. For compute nodes, the Image Customization step runs the Ansible roles against an image to create configuration files in the image. These are `/etc/lnet.conf`, `/etc/modprobe.d/lnet.conf`, and `/etc/modprobe.d/dvs.conf`. These files are kept in the Ansible roles inside the VCS repository. These files are used in the configuration procedures in this guide.

4.10.6.1 How CFS Applies DVS Configuration

The DVS configuration process (managed by CFS) is different for compute nodes (CNs) and non-compute nodes (NCNs).

An NCN is configured for DVS by a container in the CPS Content Manager and Projection Manager (CPS-CM-PM) pod run by Kubernetes. This container is responsible for configuring both LNet and DVS. It loads the modules for both services, configures LNet to work with the correct network interface, and builds the DVS node map. The DVS node map is a list of all the nodes that might run DVS. The DVS container uses the `/etc/lnet.conf`, `/etc/modprobe.d/lnet.conf`, and `/etc/modprobe.d/dvs.conf` files on the host NCN node to configure these services. If these files do not exist, the container will create them with the correct settings to run DVS. Also, these files will be overwritten if they do already exist.

Most kernel module parameters which affect DVS and LNet are typically changed by adding lines to a configuration file inside these `/etc/modprobe.d/` files residing on the non-compute nodes that run the DVS servers. Configuration changes to these files are made prior to booting the affected nodes, the changes take effect at boot, and the kernel module parameters changed will persist across boots. Refer to Change LNet or DVS Parameters for directions on how to change these parameters.

LNet on compute nodes is configured through the interface name passed into the `root=` kernel parameters while Image Customization is used to include the `/etc/modprobe.d/dvs.conf`, `/etc/modprobe.d/lnet.conf`, and `/etc/lnet.conf` files in the image root for the DVS module to read in. For more information on Image Customization, see the CSM documentation.

4.10.6.2 DVS Configuration Files

Brief descriptions and locations of the three main DVS configuration files in HPE Cray EX supercomputers.

4.10.6.2.1 `/etc/dvs_node_map.conf` and `/etc/modprobe.d/dvs.conf`

The `/etc/modprobe.d/dvs.conf` file configures the DVS kernel modules with different parameters and affects the way DVS works internally. In the configuration management VCS, the contents of this file is defined in the `dvs_modprobe_conf` variable. The variable's default value is in `config-management/roles/cray_dvs_load/defaults/main.yml`. The variable can be customized by defining it in the appropriate file in the `group_vars` directory hierarchy.

The `/etc/dvs_node_map.conf` file configures how the DVS node map is generated. The node map contains a list of all the xnames and LNet addresses which the DVS cluster will use. In the configuration management VCS, the contents of this file is defined in the `dvs_node_map_conf` variable. The variable's default value is in `config-management/roles/cray_dvs_load/defaults/main.yml`. The variable can be customized by defining it in the appropriate file in the `group_vars` directory hierarchy.

Sometimes, different sets of options must be specified on different nodes. For example, the options needed on CNs may be different from what are needed on NCNs. To accommodate such a scenario, users can use the `group_vars` directory to specify different values of the variables for different types of hosts. For example, the `/etc/modprobe.d/dvs.conf` file for Workers can be specified in `config-mamagement/group_vars/Management_Worker/dvs.yml` and the version for CNs can be specified in `config-management/group_vars/Compute/dvs.yml`. For instructions on how to apply different `dvs.conf` files to different nodes, see Configure DVS Using dvs.conf.

Note that the `/etc/dvs_node_map.conf` must be the same for all nodes in the system.

4.10.6.2.2 `/etc/modprobe.d/lnet.conf` and `/etc/lnet.conf`

The `/etc/modprobe.d/lnet.conf` file configures the LNet kernel modules with different parameters and affects the way LNet works. In the configuration management VCS, the contents of this file is defined in the `lnet_modprobe_conf` variable. The variable's default value is in `config-management/roles/cray_lnet_load/defaults/main.yml`. The variable can be customized by defining it in the appropriate file in the `group_vars` directory hierarchy.

The `/etc/lnet.conf` file contains more robust configuration options than the `/etc/modprobe.d/lnet.conf` file. In the configuration management VCS, the contents of this file is defined in the `lnet_conf` variable. The variable's default value is in `config-management/roles/cray_lnet_load/defaults/main.yml`. The variable can be customized by defining it in the appropriate file in the `group_vars` directory hierarchy.

Both of these files are modified and managed through the Git VCS repository and CFS, the same was as with `/etc/modprobe.d/dvs.conf`. For an example on how to apply different versions of these two LNet configuration files to different nodes, refer to the section "Set Up LNet Routers with CFS" in the CSM documentation.

4.10.7 When to Use Temporary, Client-Side, Per-File Read Caching

Applications which have a "read many, write once" I/O pattern can benefit greatly from the DVS `ro_cache` mount option.

Although DVS supports caching of both file reads and writes (see DVS Client-side Write-back Caching can Yield Performance Gains) the `ro_cache` mount option is still useful for situations where DVS client nodes perform several reads of a file before performing a single write (if any at all). The `ro_cache` mount option enables per-file, client-side, read caching until the first write operation on all files in mounted file system.

When the `ro_cache` mount option is used on a read/write file system, DVS enables caching for all reads on every file in that file system until a DVS client node opens one of the files with write access. When that first client attempts to write to the file, the DVS server notifies every client node to drop their locally cached version of the file. That file is never cached again and DVS handles all subsequent file I/O as it would on a normal read/write file system. All other files in that file system, however, will still be cached as long as every node opens them only for reads.

This caching mode enables codes which have a "read many, write once" I/O pattern to enjoy the benefits of client-side caching and coherency across multiple DVS client nodes at the same time for as long as possible.

4.10.8 DVS Environment Variables

By default, user environment variables allow client override of mount options specified during configuration.

By default, user environment variables allow client override of options specified during configuration and are evaluated whenever a file is opened by DVS. However, if the `nouserenv` option is included in the `options` setting of the `client_mount` configuration setting, then user environment variables are disabled for that client mount.

The following environment variables are for use in the default case:

Cray DVS User Environment Variables

Variable Name	Options	Purpose
`DVS_ATOMIC`	`on|off`	Overrides the `atomic` or `noatomic` mount options.
`DVS_BLOCKSIZE`	`n`	A nonzero number, `n` overrides the `blksize` mount option. **Note:** Unlike most other mount option and environment variable pairs, `DVS_BLOCKSIZE` and `blksize` have subtly different spellings.
`DVS_CACHE`	`on|off`	Overrides the `cache` or `nocache` mount options.Exercise caution if using this variable.
`DVS_CACHE_READ_SZ`	`n`	A positive integer, n overrides the `cache_read_sz` mount option.
`DVS_CLOSESYNC`	`on|off`	Overrides the `closesync` or `noclosesync` mount options. **Note:** Periodic sync functions similarly to the DVS `closesync` mount option, but it is more efficient and is enabled by default. Hewlett Packard Enterprise recommends not using `closesync` or this associated environment variable.
`DVS_DATASYNC`	`on|off`	Overrides the `datasync` or `nodatasync` mount options.**Note:** Setting `DVS_DATASYNC` to `on` can slow down an application considerably. The periodic sync feature, enabled by default, is a better way to synchronize data. See Periodic Sync Promotes Data and Application Resiliency for more information.
`DVS_DEFEROPENS`	`on|off`	Overrides the `deferopens` or `nodeferopens` mount options.
`DVS_KILLPROCESS`	`on|off`	Overrides the `killprocess` or `nokillprocess` mount options.

Variable Name	Options	Purpose
DVS_MAXNODES	n	A nonzero number, n overrides the maxnodes mount option. The specified value of maxnodes must be greater than zero and less than or equal to the number of server nodes specified on the mount, otherwise the variable has no effect.

4.10.9 DVS Configuration Settings

An overview of the three ways DVS users and administrators can configure DVS and enhance its performance.

Administrators and users can configure DVS (Data Virtualization Service) through:

- Writing values to the DVS-related /sys files on DVS clients and servers using the echo command.
- Adding mount options to the cpsmount.sh command (using the -o option) used by DVS clients to mount CPS-managed content other than compute node root file systems (for example, the HPE Cray Programming Environment).
- Setting DVS kernel module parameters and mount options by creating or editing the dvs.conf and lnet.conf files (both reside in the /etc/modprobe.d directory) distributed to DVS servers and clients through CFS. This method can only be used to change the mount options for all non-CPS DVS mounts.

Important: To prevent mount failure, only use settings that are compatible, in accordance with the instructions in this guide.

4.10.9.1 DVS Mount Options

DVS clients can use a long list of options when mounting a file system. This topic explains what they all do.

- loadbalance

  Used to specify loadbalance mode, which more evenly distributes loads across DVS servers, as all servers are used to access the same file. The server is chosen based on the compute node doing the access. This maximizes the scalability of both reads and meta operations like open(), stat(), and close() because all servers can be used for every file. Loadbalance mode is valid only for read-only mounts.

  - Default value: not enabled
  - Associated environment variable: none
  - Related settings/options: When loadbalance is enabled, the underlying DVS implementation automatically sets the readonly setting to true and sets these additional options: cache=1, failover=1, maxnodes=1, and hash_on_nid=1. Hewlett Packard Enterprise recommends setting attrcache_timeout=14400 as well to take advantage of the mount being read-only.

- attrcache_timeout

  Enables and sets client-side attribute caching, which can significantly increase performance, most notably in pathname lookup situations. File attributes and dentries for getattr requests, pathname lookups, and so forth, are read from DVS servers and cached on the DVS client for the number of seconds specified. For example, attrcache_timeout=5 caches attributes for five seconds. Subsequent lookups or getattr requests use the cached attributes until the timeout expires, at which point they are read and cached again on first reference. When attribute caching is disabled, DVS clients must send a lookup request to a DVS server for every level of a pathname, and repeat this for every pathname operation. When it is enabled, it sends a lookup request to a DVS server for every level of a pathname once per the number of specified seconds.

  Note: An administrator with root privilege can force a cache revalidation at any time, not just when the timeout has expired. See Force a Cache Revalidation on a DVS Mount Point .

  - Default value: 3 seconds.

    Important: This default is intended to maximize the safety of read/write mounts. This means that to enhance system performance for read-only mounts, configure this setting by entering a larger value (Hewlett Packard Enterprise recommends attrcache_timeout=14400). Large values (for example, 14400) for this mount option should not be used for read/write

file systems due to the risk of file system corruption. Not specifying this mount option will result in the underlying safe default being used.

This setting must be kept at a low value for read/write mounts to prevent application segfaults and other problems. The default setting allows the caching to be effective during the start-up of an application. DVS will detect a stale or deleted existing file on every open, so `attrcache_timeout` is needed primarily to prevent a previous failed access (due to ENOENT) from masking the subsequent creation of that file for too long.

- Associated environment variable: none

- **`atomic` / `noatomic`**

  `atomic` enables atomic stripe parallel mode. This ensures that stripe parallel requests adhere to POSIX read/write atomicity rules. DVS clients send each I/O request to a single DVS server to ensure that the bytes are not interleaved with other requests from DVS clients. The DVS server used to perform the read, write, or metadata operation is selected using an internal hash involving the underlying file or directory inode number and the offset of data into the file relative to the DVS block size.

  `noatomic` disables atomic stripe parallel mode. If there are multiple DVS servers and neither loadbalance nor cluster parallel mode is specified, DVS stripes I/O requests across multiple servers and does not necessarily adhere to POSIX read/write atomicity rules if file locking is not used.

  - Default value: `noatomic`
  - Associated environment variable: `DVS_ATOMIC`
  - Related settings/options: none

- **`blksize=n`**

  `blksize=n` sets the DVS block size to n bytes. Used in striping.

  - Default value: 524288 (512 KB)
  - Associated environment variable: `DVS_BLOCKSIZE`
  - Related settings/options: none

- **`cache` / `nocache`**

  `cache` enables client-side caching of both read and write data. The client node caches reads from the DVS server node, caches writes from user applications that are aggregated and later 'written back' to the backing file system storage on the DVS server node, and provides data to user applications from the page cache if possible, instead of performing a data transfer from the DVS server node. For more information, see DVS Client-side Write-back Caching can Yield Performance Gains. DVS is not a clustered file system; no coherency is maintained among multiple DVS client nodes reading and writing to the same file. If cache is enabled and data consistency is required, applications must take care to synchronize their accesses to the shared file.

  `nocache` disables client-side read/write caching.

  - Default value: `nocache`
  - Associated environment variable: `DVS_CACHE` (use with caution)
  - Related settings/options: When `loadbalance` is enabled, DVS automatically enables `cache`. If both `readonly` and `cache` options are specified, the client node will cache only read data (this is equivalent to disabling write caching). **Important:** If enabling read/write caching, read About the Close-to-Open Coherency Model to understand the implications and prevent data corruption.

- **`cache_read_sz`**

  `cache_read_sz` is a limit that can be specified to prevent reads or writes over this size from being cached in the Linux page cache.

  - Default value: 0
  - Associated environment variable: `DVS_CACHE_READ_SZ`
  - Related settings/options: If the `cache` mount option is not used, DVS ignores `cache_read_sz`.

- **`closesync` / `noclosesync`**

  `closesync` enables data synchronization on last close of a file. When a process performs the final close of a file descriptor, and forwards the close to the DVS server, the DVS server node waits until data has been written to the underlying media before indicating that the close has completed. Because DVS does not cache data on client nodes (unless the `cache` option is used) and has no replay capabilities, this ensures that data is not lost if a server node crashes after an application has exited.

  `noclosesync` causes DVS to return a `close()` request immediately.

  - Default value: noclosesync
  - Associated environment variable: DVS_CLOSESYNC
  - Related settings/options: The closesync option is redundant with periodic sync, which is enabled by default. Because periodic sync is more efficient than closesync, Hewlett Packard Enterprise recommends letting periodic sync take care of data synchronization instead of using this mount option.

- **datasync / nodatasync**

  datasync enables data synchronization. The DVS server node waits until data has been written to the underlying media before indicating that the write has completed. Can significantly impact performance.

  nodatasync causes a DVS server node to return from a write request as soon as the user data has been written into the page cache on the server node.

  - Default value: nodatasync
  - Associated environment variable: DVS_DATASYNC
  - Related settings/options: none

- **deferopens / nodeferopens**

  deferopens defers DVS client open() requests to DVS servers for a given set of conditions. When a file is opened in stripe parallel mode or atomic stripe parallel mode, DVS clients send the open() request to a single DVS server only. More open() requests are sent as necessary when the DVS client performs a read or write to a different server for the first time. The deferopens option deviates from POSIX specifications. For example, if a file was removed after the initial open() succeeded but before deferred opens were initiated by a read or write operation to a new server, the read or write operation would fail with errno set to ENOENT because the open() was unable to open the file.

  nodeferopens disables the deferral of DVS client open() requests to DVS servers. When a file is open in stripe parallel mode or atomic stripe parallel mode, DVS clients send open() requests to all DVS servers denoted by nodename or nodefile.

  - Default value: nodeferopens
  - Associated environment variable: DVS_DEFEROPENS

- **distribute_create_ops / nodistribute_create_ops**

  distribute_create_ops causes DVS to change its hashing algorithm so that create and lookup requests are distributed across all the servers, as opposed to being distributed to a single server. This applies to create()s, mkdir()s, lookup()s, mknod()s, link()s, and symlink()s.

  nodistribute_create_ops causes DVS to use its normal algorithm of using just one target server.

  - Default value: nodistribute_create_ops
  - Associated environment variable: none
  - Related settings/options: none

- **failover / nofailover**

  failover enables failover and failback of DVS servers. If all servers fail, operations for the mount point behave as described by the retry option until at least one server is rebooted and has loaded DVS. If multiple DVS servers are listed for a client mount and one or more of the servers fails, operations for that mount continue by using the subset of servers still available. When the downed servers are rebooted and start DVS, any client mounts that had performed failover operations failback to once again include the servers as valid nodes for I/O forwarding operations.

  nofailover disables failover and failback of DVS servers. If one or more servers for a given client mount fail, operations for that mount behave as described by the retry or noretry option specified for the client mount.

  - Default value: failover
  - Associated environment variable: none
  - Related settings/options: When the failover option is enabled (occurs automatically when loadbalance is enabled), the noretry option cannot be enabled.

- **hash**

  Except in cases of advanced administrators or specific advice from DVS developers, do not use the hash mount option. The best course of action is to let DVS use its default value. The hash option has three possible values:

  - **fnv-1a**

    hash=fnv-1a offers the best overall performance with little variation due to differing numbers of servers.

  - **jenkins**

    hash=jenkins is the hash that DVS previously used. It is included in the unlikely case of edge-case pathological issues with the fnv-1a hash, but it has worse overall performance.

  - **modulo**

    hash=modulo does not do any hash at all, but rather takes the modulo of the seed that it is given. This option can potentially have high load-balancing characteristics, but is vulnerable to pathological cases such as file systems that only allocate even-numbered inodes or a prime number of servers.

  - Default value: fnv-1a
  - Associated environment variable: none
  - Related settings/options: none

- **hash_on_nid/nohash_on_nid**

  When the hash_on_nid mount option is used, DVS uses the nid of the client as the hash seed instead of using the file inode number. This effectively causes all request traffic for the compute node to go to a single server. This can help metadata operation performance by avoiding lock thrashing in the underlying file system when each process on a set of DVS clients is using a separate file. The nohash_on_nid option disables this behavior, causing DVS to use the file inode number as the hash seed.

  - Default value: nohash_on_nid
  - Associated environment variable: none
  - Related settings/options: When hash_on_nid is enabled, DVS automatically sets the hash option to modulo. When loadbalance is enabled, DVS automatically sets hash_on_nid=1.

- **killprocess / nokillprocess**

  killprocess enables killing processes that have one or more file descriptors with data that has not yet been written to the backing store. DVS provides this option to minimize the risk of silent data loss, such as when data still resides in the kernel or file system page cache on the DVS server after a write has completed.

  nokillprocess disables the killing of processes that have written data to a DVS server when a server fails. When a server fails, processes that have written data to the server are not killed. If a process continues to perform operations with an open file descriptor that had been used to write data to the server, the operations fail (with errno set to EHOSTDOWN). A new open of the file is allowed, and subsequent operations with the corresponding file descriptor function normally.

  - Default value: killprocess
  - Associated environment variable: DVS_KILLPROCESS
  - Related settings/options: With the periodic sync feature (enabled by default), DVS servers attempt to fsync dirty files to minimize the number of processes that are killed and will also fsync a dirty file's data when the file is closed. If periodic sync is disabled (not recommended), the killprocess option alone cannot fully guarantee prevention of silent data loss (though it is highly unlikely) because a close() does not guarantee that data has been transferred to the underlying media (see the closesync option).

- **magic**

  magic defines what the expected file system magic value for the projected file system on the DVS servers should be. When a DVS client attempts to mount the file system from a server, it verifies that the underlying file system has a magic value that matches the specified value. If not, the DVS client excludes that DVS server from the list of servers it uses for the mount point and prints a message to the system console. Once the configuration issue on the DVS server has been addressed and the client mounts the correct file system, DVS can be restarted on the server. All clients subsequently verify that the server is configured correctly and include the server for that mount point. Many file system magic values are defined in the /usr/include/linux/magic.h file. Commonly used magic values are:

  - 0x6969 for NFS
  - 0x47504653 for IBM Spectrum Scale (GPFS)
  - 0x9123683E for BTRFS
  - 0x01021994 for TMPFS

The default value is the underlying file system's magic value. There are no associated environment variables or related settings or options for `magic`.

- `maxnodes`

  `maxnodes` is used in configuring DVS modes.

  - Default value: number of nodes specified in node name list or contained in nodefile.
  - Associated environment variable: `DVS_MAXNODES`
  - Related settings/options: When `loadbalance` is enabled, DVS automatically sets `maxnodes=1`.

- **nodefile**

  nodefile is the file name of a file with a list of server nodes specified as xnames separated by a colon (:) and no spaces.

- **nodename**

  nodename is a list of server nodes specified as xnames separated by a colon (:) and no spaces.

- `retry` / `noretry`

  `retry` enables the retry option, which affects how a DVS client node behaves in the event of a DVS server node going down. If `retry` is specified, any user I/O request is retried until it succeeds, receives an error other than a "node down" indication, or receives a signal to interrupt the I/O operation.

  `noretry` disables retries of user I/O requests when the DVS server receiving the request is down.

  - Default value: `retry`
  - Associated environment variable: none
  - Related settings/options: When the `failover` option is enabled, the `noretry` option cannot be enabled.

- `ro_cache` / `no_ro_cache`

  `ro_cache` enables read-only caching for files on writable client mounts. Files opened with read-only permissions in `ro_cache` mode are treated as if they were on a DVS read-only cached client mount. If the file has any concurrent open that has write permissions, all instances of that file revert to the default `no_ro_cache` mode for the current and subsequent reads. For more information, see When to Use Temporary, Client-Side, Per-File Read Caching.

  `no_ro_cache` disables read-only caching for files on writable client mounts.

  - Default value: `no_ro_cache`
  - Associated environment variable: none
  - Related settings/options: none

- `userenv` / `nouserenv`

  `userenv` specifies that DVS must honor end-user environment variable overrides for DVS mount options.

  `nouserenv` allows the administrator to block end-user environment variable overrides for DVS mount options.

  - Default value: `userenv`
  - Associated environment variable: none
  - Related settings/options: none

- `clusterfs`/`noclusterfs`

  `clusterfs` specifies that the multiple DVS servers (if more than one are used) projecting the file system to be mounted are backed by a coherent cluster file system. As a result, DVS performs consistency checks on that file system, including verifying that the inode numbers are the same across all duplicated file systems. In previous versions of DVS, this option was enabled by default, but not available for users to enable or disable. Previous versions of DVS assumed that when multiple servers are used, they all mounted the same underlying file system, and thus the inode information for a given directory is the same, regardless of the specific server node used by a client.

  Starting with HPE Cray EX release 1.2, this assumption is no longer true. When DVS is run in conjunction with CPS and multiple servers are used, each of those servers is projecting multiple copies of the same content from separate NCN local file systems. This results in inode number mismatches between the servers for the same content, causing DVS to return errors when `clusterfs` is enabled (which it is by default).

noclusterfs was introduced in release 1.2 to relax the consistency checks that DVS performs on file system metadata, thus allowing multiple DVS servers to be used with CPS. As a result, load balancing and failover configurations are possible with DVS. When the noclusterfs mount option is specified, DVS enforces read-only access.

Note: Like clusterfs, the noclusterfs mount option assumes that the multiple DVS servers are projecting a coherent, parallel file system.

- Default value: clusterfs
- Associated environment variable: none
- Related settings/options: none

4.10.10 Periodic Sync Promotes Data and Application Resiliency

DVS periodic sync improves data and application resiliency and is more efficient than closesync.

DVS periodic sync improves data resiliency and application resiliency so that applications may continue executing in the event of a stalled filesystem or DVS server failure. Without periodic sync, such an event would result in DVS clients killing any processes with open files that were written through the failed server. Any data written through that server that was only in the server's page cache and not written to disk would be lost, and processes using the file would see data corruption.

Periodic sync works by periodically performing an fsync on individual files with written data on the DVS servers, to ensure that those files are written to disk. For each file, the DVS client tracks when a DVS server performs a file sync and when processes on DVS clients write to it, and then notifies the DVS server when fsync is needed. Periodic sync functions similarly to the DVS closesync mount option, but it is more efficient because it is aware of which files may have written data. Unlike closesync and datasync, periodic sync can sync data over time asynchronously so that the client does not need to wait for the sync operation to complete.

DVS periodic sync is effectively enabled by default because the sync_period_secs parameter, which affects the amount of time between syncs on the server, is non-zero by default. The only way to effectively disable periodic sync is to set that parameter to 0. HPE recommends keeping periodic sync enabled instead of using the closesync mount option.

5 Configure Overlay Preload

5.1 Configure Overlay Preload

Provides information on configuration, file lists, and workload optimization for the Cray Overlay Preload feature. The compute node root filesystem utilizes the Linux overlayfs architecture. This consists of a read-only lower layer that uses the Data Virtualization Service (DVS), and a read-write, RAM based upper layer. This architecture supports copying files from the lower layer to the upper layer to increase performance and support writes.

The Overlay Preload feature uses this copy operation to increase performance on frequently accessed files. A list of files is provided at boot, and they are all copied into local memory. All future references to those files are serviced by the local file system, rather than requiring remote data and/or metadata DVS operations. This improves system and application performance. However, the amount of memory available on the node is consequentially reduced by the cumulative size of all files copied into its memory.

The total amount of memory used by Overlay Preload can be configured by the system administrator to balance the performance and memory requirements of the system.

The system is shipped with a default list of files to be preloaded. This list is specific to the operating system release provided and the IO access recorded during system boot. The administrator can modify this list if desired.

5.1.1 Configuration Settings

Overlay Preload configuration management is supported via the standard Cray Configuration Framework Service (CFS). See "Configuration Management" in the CSM documentation for more information. The overlay-preload Ansible role is used to configure Overlay Preload.

The following variable is available in the overlay-preload Ansible role.

- **overlay-preload-size-limit**

  The size, in MB, that limits the amount of file data that is promoted to the overlay cache.

  A value of 0 indicates 'unlimited' file data.

5.1.2 File lists

The list of files to be preloaded at boot are located in the file /opt/cray/overlay-preload/config/dist/overlay-preload.filelist. The file format is simply a list of file paths, one per line. Wildcard values are supported.

The file list in the default boot image may be modified using standard configuration management techniques.

The file is read early in the boot process, and files will be processed in order. If there are constraints placed on total preload size, processing will stop once the limit is reached. In this case, files that are critical for preloading should be placed first.

5.1.3 Create Custom Loads for Specific Workloads

Sites may define their own file lists to optimize work for specific workloads. The Overlay Preload package ships with a script to aid in determining which files are accessed at boot time. It will analyze boot behavior and produce a list of files accessed.

Collect boot data by using the Image Management Service (IMS) to create a filesystem image with the cray-preload-strace service enabled. This can be done via the image customization or node personalization process.

The following is a general workflow for this process:

1. Enable the cray-preload-strace service.

   ```
   # systemctl enable cray-preload-strace
   ```

2. Boot a compute node with the new filesystem image.
3. Log into the compute node as root and kill any strace process.

   The strace log can be found at /cray-preload-strace.log.

4. Run the preload-strace-analyze.sh script with the strace log as input.

   ```
   # preload-strace-analyze.sh /cray-preload-strace.log
   ```

   The output will be a list of files, access counts, and sizes. The sum is included at the bottom. This can be used as the basis for creating or modifying an overlay file list.

5. Disable the cray-preload-strace service.

   ```
   # systemctl disable cray-preload-strace
   ```

5.1.4 The Overlay Preload Log File and Symlinks

Overlay Preload creates a log file on affected nodes at /var/log/cray/overlay-preload.log. The log file contains warnings for files that were not found, as well as the number and size of the files preloaded on the node.

Any symlinks included in a file list may not be copied from the lower layer to the node-local RAM file system, which might look confusing. For example, if a site's content list contains /etc/alternatives/unzip, which is a symlink to /usr/bin/unzip-plain. In this case, both the link and its target are present in lower layer, but neither of them appear in the node-local file system. This is expected and correct behavior. A site that is concerned about possible confusion for administrators can decide to exclude symlinks from file lists, or simply list the target of the symlink in a file list to ensure that it is present in the node-local file system.

6 Install Spectrum Scale

6.1 Install Spectrum Scale on a NCN-GW Node

Keep in mind the following three things before performing Create Spectrum Scale NCN Gateway Node Image:

1. **There are many 3rd-party dependencies that must be satisfied to successfully build and install the Spectrum Scale client software**. The guidance below suggests a procedure that is flexible enough for you to obtain and install all required dependencies. However, your available specific 3rd-party dependencies, and the sources for those 3rd-party dependencies, may vary from what is described. Please refer to documentation at www.ibm.com as needed.
2. **Per IBM documentation, Spectrum Scale servers and Spectrum Scale clients need to maintain password-less SSH access to each other across reboots.** For nodes that do not have local storage, the contents of an NCN-GW authorized keys file is not automatically preserved across a reboot. This means that, to re-establish the authorized keys for the external Spectrum Scale servers, the guidance below hard-codes those key values into the NCN-GW customized boot process. Therefore, if the SSH keys of the external Spectrum

Scale servers change, the customized boot process must be performed to re-establish appropriate authorized keys of those Spectrum Scale servers on the NCN-GW nodes that have Spectrum Scale clients. Customers are free to explore alternative approaches to address this issue.

3. **Nodes without persistent storage will not automatically rejoin the Spectrum Scale cluster after a reboot.** Nodes that do not have local or network-mounted storage, for storing dynamically updated Spectrum Scale data across reboots will lose cluster information after a reboot. This prevents that such nodes from automatically re-joining the Spectrum Scale cluster. The guidance below describes updating the customized boot process to establish and auto-run an ansible script to run the Spectrum Scale `mmstartup` and `mmsdrrestore` commands to recover and re-establish the necessary Spectrum Scale cluster membership on reboot. Customers are free to explore alternative approaches to address this issue.

6.1.1 Create Spectrum Scale NCN Gateway Node Image

PREREQUISITES

Identify an NCN Gateway (NCN-GW) node that will support a DVS server and Spectrum Scale client, that can be reserved for the entire duration of this procedure.

OBJECTIVE

This procedure guides HPE Cray EX edministrators through the phases of creating a Spectrum Scale-enabled NCN-GW image:

1. Installing prerequisite software
2. Copying the Spectrum Scale software to an NCN-GW node.
3. Extracting and installing the Spectrum Scale software on the NCN-GW node.
4. Building the Spectrum Scale client kernel module (portability layer) RPM
5. Installing the portability layer and IBM-provided RPMs into a base NCN-GW image to create a Spectrum Scale-enabled NCN-GW image.

LIMITATIONS

This procedure provides HPE Cray EX-specific instructions and guidance for manually installing Spectrum Scale on HPE Cray OS. These steps are not intended to replace the instructions and guidance provided on the IBM website (www.ibm.com).

PROCEDURE

The command examples in this procedure use `gw01` as the hostname of the NCN Gateway node.

Connect HPE Cray EX system to Spectrum Scale cluster

1. Connect the Spectrum Scale servers to the NCN-GW, following local administrative procedures.
2. Verify that password-less SSH can be successfully configured between Spectrum Scale servers and NCN-GW nodes.

Download the Spectrum Scale software to an NCN-GW node

3. Log into `ncn-m001` of the HPE Cray EX system as root through SSH.
4. From `ncn-m001` log into the NCN-GW node, for example `gw01`, of the HPE Cray EX system as root through SSH.
5. Create and navigate into a directory on that NCN-GW node to contain the Spectrum Scale installation software that will be downloaded in the next step.

   ```
   gw01# mkdir -p /tmp/GPFS5
   gw01# cd /tmp/GPFS5
   ```

6. Download and copy the install file of the Spectrum Scale Data Management release version 5.1.0.3 installation software for Linux from the IBM website to the `/tmp/GPFS5` directory.

   Only the 5.1.0.3 release version has been qualified for this HPE Cray OS release.

Extract and install the Spectrum Scale software

7. Run the self-extracting Spectrum Scale software archive.

   ```
   gw01# chmod +x Spectrum_Scale_Data_Management-*-x86_64-Linux-install
   gw01# ./Spectrum_Scale_Data_Management-*-x86_64-Linux-install
   ```

   Within 2 minutes, the self-extracting archive will print out several lines of output during the installation process. Then it will ask for acceptance of the Spectrum Scale license.

8. Enter 1 to accept the license agreement when prompted.

```
. . .
LICENSE INFORMATION

The Programs listed below are licensed under the following
License Information terms and conditions in addition to the
Program license terms previously agreed to by Client and
IBM. If Client does not have previously agreed to license
terms in effect for the Program, the International Program
License Agreement (Z125-3301-14) applies.

Program Name (Program Number):
IBM Spectrum Scale Data Management Edition 5.1.0.3 (5737-
F34)
IBM Spectrum Scale Data Management Edition 5.1.0.3 (5641-
DM1)

Press Enter to continue viewing the license agreement, or
enter "1" to accept the agreement, "2" to decline it, "3"
to print it, "4" to read non-IBM terms, or "99" to go back
to the previous screen.
'1'

License Agreement Terms accepted.
...
```

   The installer will then finish and display the message `Product packages successfully extracted to /usr/lpp/mmfs/5.1.0.3`.

9. Navigate to the directory containing the Spectrum Scale packages and install them.

```
gw01# cd /usr/lpp/mmfs/5.1.0.3/gpfs_rpms
gw01# rpm -ivh gpfs.base*.rpm gpfs.gpl*rpm gpfs.license*rpm gpfs.gskit*rpm \
gpfs.msg*rpm gpfs.compression*rpm gpfs.adv*rpm gpfs.crypto*rpm gpfs.docs*.rpm gpfs.afm*.rpm
warning: gpfs.base-5.1.0-3.x86_64.rpm: Header V3 RSA/SHA256 Signature, key ID efeb6eeb: NOKEY
Preparing...                          ################################# [100%]
Updating / installing...
1:gpfs.base-5.1.0-3                   ################################# [ 10%]
Created symlink /etc/systemd/system/multi-user.target.wants/mmautoload.service → 
/usr/lib/systemd/system/mmautoload.service.
Created symlink /etc/systemd/system/multi-user.target.wants/mmccrmonitor.service → 
/usr/lib/systemd/system/mmccrmonitor.service.
 2:gpfs.adv-5.1.0-3                   ################################# [ 20%]
 3:gpfs.license.dm-5.1.0-3            ################################# [ 30%]
 4:gpfs.gpl-5.1.0-3                   ################################# [ 40%]
 5:gpfs.compression-5.1.0-3           ################################# [ 50%]
 6:gpfs.crypto-5.1.0-3                ################################# [ 60%]
 7:gpfs.afm.cos-1.0.0-1               ################################# [ 70%]
 8:gpfs.docs-5.1.0-3                  ################################# [ 80%]
 9:gpfs.msg.en_US-5.1.0-3             ################################# [ 90%]
10:gpfs.gskit-8.0.55-12               ################################# [100%]
```

Build the portability layer RPM

10. Add the directory containing the Spectrum Scale programs to the PATH on the NCN-GW node.

```
gw01# export PATH=$PATH:/usr/lpp/mmfs/bin
```

    If this directory is not manually added, one or more Spectrum Scale scripts will fail.

11. Install the `rpm-build` package and its dependencies. Respond `yes` or `y` to all prompts during the process.

This utility must be installed to package the Spectrum Scale kernel module (portability layer) into an RPM.

```
gw01# zypper install rpm-build
Retrieving repository 'wlm-slurm-sle-15sp1-compute' metadata .............[done]
Building repository 'wlm-slurm-sle-15sp1-compute' cache ..................[done]
Loading repository data...
Reading installed packages...
Resolving package dependencies...

The following 5 NEW packages are going to be installed:
  bison-lang dwz gettext-tools rpm-build systemd-rpm-macros

The following recommended package was automatically selected:
  bison-lang

5 new packages to install.
Overall download size: 2.4 MiB. Already cached: 0 B. After the operation,
additional 8.6 MiB will be used.
Continue? [y/n/v/...? shows all options] (y): y
Retrieving package bison-lang-3.0.4-1.268.noarch
                                          (1/5), 107.9 KiB (551.3 KiB unpacked)
Retrieving: bison-lang-3.0.4-1.268.noarch.rpm ............................[done]
Retrieving package systemd-rpm-macros-5-7.9.1.noarch
                                          (2/5),  16.7 KiB ( 10.9 KiB unpacked)
Retrieving: systemd-rpm-macros-5-7.9.1.noarch.rpm ........................[done]
Retrieving package gettext-tools-0.19.8.1-4.11.1.x86_64
                                          (3/5),   2.1 MiB (  7.9 MiB unpacked)
Retrieving: gettext-tools-0.19.8.1-4.11.1.x86_64.rpm .....................[done]
Retrieving package dwz-0.12-1.483.x86_64   (4/5),  89.8 KiB (179.8 KiB unpacked)
Retrieving: dwz-0.12-1.483.x86_64.rpm ....................................[done]
Retrieving package rpm-build-4.14.1-10.19.8.x86_64
                                          (5/5),  30.0 KiB ( 24.9 KiB unpacked)
Retrieving: rpm-build-4.14.1-10.19.8.x86_64.rpm ..........................[done]

Checking for file conflicts: .............................................[done]
(1/5) Installing: bison-lang-3.0.4-1.268.noarch ..........................[done]
(2/5) Installing: systemd-rpm-macros-5-7.9.1.noarch ......................[done]
(3/5) Installing: gettext-tools-0.19.8.1-4.11.1.x86_64 ...................[done]
(4/5) Installing: dwz-0.12-1.483.x86_64 ..................................[done]
(5/5) Installing: rpm-build-4.14.1-10.19.8.x86_64 ........................[done]
```

12. If your NCN-GW node operating system is SLES 15 SP2, apply a patch to /usr/lpp/mmfs/src/gpl-linux/verdep.h so that mmbuildgpl will complete successfully.

Create a patch file with the following contents:

```
--- /usr/lpp/mmfs/src/gpl-linux/verdep.h        2021-09-21 16:27:32.721209906 -0400
+++ /tmp/verdep.h       2021-09-21 16:23:24.554452762 -0400
@@ -1864,7 +1864,8 @@
 #endif

 /* commit 021182e5 in kernel v4.8 enabled KASLR */
-#if defined(GPFS_ARCH_X86_64) && defined(CONFIG_RANDOMIZE_MEMORY)
+/* commit eedb92ab in kernel v4.17 changed __PAGE_OFFSET to support 5-level */
+#if defined(GPFS_ARCH_X86_64) && (defined(CONFIG_RANDOMIZE_MEMORY)
+| defined(CONFIG_DYNAMIC_MEMORY_LAYOUT))
 #define KC_X86_64_RANDOM_PAGE_OFFSET_BASE
 #endif
```

Apply the patch to the file /usr/lpp/mmfs/src/gpl-linux/verdep.h:

```
gw01# patch -u /usr/lpp/mmfs/src/gpl-linux/verdep.h -i <patch_filename>
```

13. Build the Spectrum Scale kernel module RPM on the NCN-GW node.

This module serves as the portability layer between the Linux kernel and the Spectrum Scale daemon from IBM.

```
gw01# /usr/lpp/mmfs/bin/mmbuildgpl --build-package
-----------------------------------------------------------
mmbuildgpl: Building GPL (5.1.0.3) module begins at Wed Mar 24 12:32:01 CDT 2021.
-----------------------------------------------------------
Verifying Kernel Header...
  kernel version = 41214197 (412140197078000, 4.12.14-197.78_9.1.58-cray_shasta_c, 4.12.14-197.78)
  module include dir = /lib/modules/4.12.14-197.78_9.1.58-cray_shasta_c/build/include
  module build dir   = /lib/modules/4.12.14-197.78_9.1.58-cray_shasta_c/build
  kernel source dir  = /usr/src/linux-4.12.14-197.78/include
  Found valid kernel header file under /lib/modules/4.12.14-197.78_9.1.58-cray_shasta_c/build/include
Getting Kernel Cipher mode...
  Will use blkcipher routines
Verifying Compiler...
  make is present at /usr/bin/make
  cpp is present at /usr/bin/cpp
  gcc is present at /usr/bin/gcc
  g++ is present at /usr/bin/g++
  ld is present at /usr/bin/ld
Verifying rpmbuild...
Verifying Additional System Headers...
  Verifying linux-glibc-devel is installed ...
    Command: /bin/rpm -q linux-glibc-devel
    The required package linux-glibc-devel is installed
make World ...
make InstallImages ...
make rpm ...
Wrote: /usr/src/packages/RPMS/x86_64/gpfs.gplbin-4.12.14-197.78_9.1.58-cray_shasta_c-5.1.0-3.x86_64.rpm
-----------------------------------------------------------
mmbuildgpl: Building GPL module completed successfully at Wed Mar 24 12:32:15 CDT 2021.
-----------------------------------------------------------
```

14. Confirm that the portability layer RPM was built.

```
gw01# cd /usr/src/packages/RPMS/x86_64
gw01# ls -l
total 1064
-rw-r--r-- 1 root root 1085920 Mar 24 12:32
gpfs.gplbin-4.12.14-197.78_9.1.58-cray_shasta_c-5.1.0-3.x86_64.rpm
```

Prepare to customize a base NCN-GW image

15. Copy the required Spectrum Scale software RPMs and portability layer RPM to a working directory.

```
gw01# cd /usr/lpp/mmfs/5.1.0.3/gpfs_rpms
gw01# cp gpfs.base*.rpm gpfs.gpl*rpm gpfs.license*rpm gpfs.gskit*rpm gpfs.msg*rpm \
gpfs.compression*rpm gpfs.adv*rpm gpfs.crypto*rpm gpfs.afm*rpm gpfs.docs*rpm \
/usr/src/packages/RPMS/x86_64
```

16. Login to `ncn-w001` of the HPE Cray EX system as `root` through SSH.

17. Make a directory to hold the Spectrum Scale RPMs, and copy the RPMs from the NCN-GW node into that directory.

```
ncn-w001# mkdir /tmp/GPFS5
ncn-w001# scp root@gw01:/usr/src/packages/RPMS/x86_64/gpfs.*.rpm /tmp/GPFS5
```

18. Follow IBM instructions to add your Spectrum Scale gateway node as a client in your Spectrum Scale cluster, and verify proper functionality. The process requires steps similar to the following:

    1. Login to a Spectrum Scale server node to obtain the Admin node name and Daemon node name for all Spectrum Scale server nodes

```
hi1# mmlscluster

GPFS cluster information
========================
  GPFS cluster name:         hi1
  GPFS cluster id:           17825395649093381303
  GPFS UID domain:           hi1
  Remote shell command:      /usr/bin/ssh
  Remote file copy command:  /usr/bin/scp
  Repository type:           CCR

 Node  Daemon node name  IP address    Admin node name  Designation
-----------------------------------------------------------------
   1   venom1-gpfs       10.253.100.1  hi1              quorum-manager
   2   venom2-gpfs       10.253.100.2  hi2              quorum-manager

hi1#
```

2. Add the following line to the `/etc/hosts` file of the NCN-GW node, making one entry for each Spectrum Scale server node:

   ```
   HSN_IP_ADDRESS  ADMIN_NODE_NAME DAEMON_NODE_NAME
   ```

   Where:

   - *HSN_IP_ADDRESS* is the IP address of the node on the HPE Cray EX HSN.
   - *ADMIN_NODE_NAME* is the Spectrum Scale `Admin node name` of the node.
   - *DAEMON_NODE_NAME* is the Spectrum Scale `Daemon node name` of the node.

3. On the NCN-GW node, generate an RSA SSH keypair in `/root/.ssh`.

   ```
   gw01# cd /root
   gw01# ssh-keygen -t rsa
   ```

4. Copy the NCN-GW root user RSA SSH public key to every Spectrum Scale server node (*DAEMON_NODE_NAME*) in the cluster by running the following command for each Spectrum Scale server node:

   ```
   gw01# ssh-copy-id DAEMON_NODE_NAME
   ```

5. On each Spectrum Scale server node, add to the `/etc/hosts` file a line for each NCN-GW node:

   ```
   HSN_IP_ADDRESS  NCN_GW_HOSTNAME
   ```

   Where:

   - *HSN_IP_ADDRESS* is the IP address of the NCN-GW node on the HPE Cray EX HSN.
   - *GW_HOSTNAME* is the hostname of the NCN-GW node (`gw01`, for example).

6. Copy the Spectrum Scale server node root user RSA SSH public key to every NCN-GW node (*NCN_GW_HOSTNAME*) in the HPE Cray EX system by running the following command on each Spectrum Scale server node, repeating for each NCN-GW node:

   ```
   hi1# ssh-copy-id NCN_GW_HOSTNAME
   ```

19. Obtain an IMS public key `id`.

    1. Check for a key belonging to the `root` user. Run the following command and search the output for a key that includes `root` in the `name` value.

       ```
       ncn-w001# cray ims public-keys list
       ```

    2. Create an IMS public key for the `root` user if does not already exist.

       The `public_key` value in the following example has been truncated due to length.

       ```
       ncn-w001# cray ims public-keys create --name "root public key" --public-key ~/.ssh/id_rsa.pub
       public_key = "ssh-rsa
       AAAAB3NzaC1yc2EAAAADAQABAAABgQDL3BkF8Q4N7bnbLbcIZmRbcNtwdgYos9a2jxAJytnzrgVvN3gIcEftnwKOl8F. . ."
       id = "d8af25a4-9d29-4baa-9c66-880aaa84d6c9"
       ```

```
    name = "root public key"
    created = "2021-03-16T15:38:43.285684+00:00"
```

20. Save the `id` value of the root public key in an environment variable.

    The output of the previous command returned `id = "d8af25a4-9d29-4baa-9c66-880aaa84d6c9"` Therefore, the command for this step would be:

    ```
    ncn-w001# export IMS_PUBLIC_KEY_ID=d8af25a4-9d29-4baa-9c66-880aaa84d6c9
    ```

21. Determine the xname value of the NCN-GW node that has Spectrum Scale installed.

    ```
    ncn-w001# ssh root@gw01
    gw01# cat /etc/cray/xname
    x3000c0s27b0n0
    gw01# exit
    ```

22. Determine the specific kernel running on the NCN-GW node that has Spectrum Scale installed. Run `cray bss bootparameters list` with the `--name X` option, where `X` is the xname of the NCN-GW node that has Spectrum Scale installed (found in the previous step).

    ```
    ncn-w001# cray bss bootparameters list --name x3000c0s27b0n0
    [[results]]
    hosts = [ "x3000c0s27b0n0",]
    params = "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}
    root=craycps-s3:s3://boot-images/d237da41-e29e-4f4f-9b51-df9410777ae2/rootfs:
    ca6877a1700bd1ec73fb908360369a5b-204:dvs:api-gw-service-nmn.local:300:nmn0 nmd_data=url=s3://boot-
    images/d237da41-e29e-4f4f-9b51-df9410777ae2/rootfs,etag=ca6877a1700bd1ec73fb908360369a5b-204
    bos_session_id=b606a266-9659-47d3-b93d-9b1de1e26514"
    kernel = "s3://boot-images/d237da41-e29e-4f4f-9b51-df9410777ae2/kernel"
    initrd = "s3://boot-images/d237da41-e29e-4f4f-9b51-df9410777ae2/initrd"

    [results.cloud-init.phone-home]
    pub_key_dsa = ""
    pub_key_rsa = ""
    pub_key_ecdsa = ""
    instance_id = ""
    hostname = ""
    fqdn = ""
    ```

23. Obtain the `id` from the `kernel` value in the output of the previous step.

    This `id` is the hyphenated hexadecimal string in the S3 path between `/boot-images` and `/kernel`. In the command output in the previous step, the kernel `id` is `d237da41-e29e-4f4f-9b51-df9410777ae2`.

24. Verify that one of the `id` values in the output of the following command matches the `kernel` value obtained in the previous step.

    ```
    ncn-w001# cray ims images list
    ...
    [results.link]
    etag = "7425cdc2f4c58510c6fa5e25223082c0"
    path = "s3://boot-images/d70aa2e2-2189-4322-bbd7-589ab94613c3/manifest.json"
    type = "s3"
    [[results]]
    created = "2021-03-09T13:34:36.855312+00:00"
    id = "d237da41-e29e-4f4f-9b51-df9410777ae2"
    name = "cray-shasta-uan-cos-sles15sp1.x86_64-0.1.32_cfs_uan-config-2.0.0"
    ...
    ```

25. Save the `id` value of the IMS image to an environment variable.

```
ncn-w001# export IMS_IMAGE_ID=d237da41-e29e-4f4f-9b51-df9410777ae2
```

Customize base NCN-GW image with Spectrum Scale software

26. Create an IMS image customization session using the base NCN-GW image identified in the previous steps. Replace *GW_GPFS_CUSTOM_IMAGE* in the following command with a descriptive name that includes information about the base image that was customized.

```
ncn-w001# cray ims jobs create --job-type customize --image-root-archive-name GW_GPFS_CUSTOM_IMAGE \
--artifact-id $IMS_IMAGE_ID --public-key-id $IMS_PUBLIC_KEY_ID
image_root_archive_name = "gw_gpfs_custom_image"
status = "creating"
enable_debug = false
kubernetes_job = "cray-ims-c4679389-a8d2-45d7-b983-028b388618d5-customize"
kubernetes_configmap = "cray-ims-c4679389-a8d2-45d7-b983-028b388618d5-configmap"
created = "2021-03-24T20:46:33.353803+00:00"
build_env_size = 10
kernel_file_name = "vmlinuz"
kubernetes_namespace = "ims"
public_key_id = "d8af25a4-9d29-4baa-9c66-880aaa84d6c9"
job_type = "customize"
initrd_file_name = "initrd"
id = "c4679389-a8d2-45d7-b983-028b388618d5"
artifact_id = "d237da41-e29e-4f4f-9b51-df9410777ae2"
kubernetes_service = "cray-ims-c4679389-a8d2-45d7-b983-028b388618d5-service"
[[ssh_containers]]
status = "pending"
jail = false
name = "customize"

[ssh_containers.connection_info.customer_access]
port = 22
host = "c4679389-a8d2-45d7-b983-028b388618d5.ims.groot.dev.cray.com"
[ssh_containers.connection_info."cluster.local"]
port = 22
host = "cray-ims-c4679389-a8d2-45d7-b983-028b388618d5-service.ims.svc.cluster.local"
```

The values for `kubernetes_job`, `id`, `port`, and `host` in the preceding example output will be saved in the next two steps.

27. Save the values for `host` and `port` under the `[ssh_containers.connection_info.customer_access]` section of the output of the previous step to environment variables.

```
ncn-w001# export IMS_SSH_HOST=c4679389-a8d2-45d7-b983-028b388618d5.ims.groot.dev.cray.com
ncn-w001# export IMS_SSH_PORT=22
```

28. Save the values for `kubernetes_job` and `id` to environment variables.

```
ncn-w001# export IMS_JOB_ID=c4679389-a8d2-45d7-b983-028b388618d5
ncn-w001# export IMS_KUBERNETES_JOB=cray-ims-c4679389-a8d2-45d7-b983-028b388618d5-customize
```

29. Verify that the NCN-GW image customization job has started successfully by running the following command once every minute for five minutes or until the output reports `SuccessfulCreate`.

```
ncn-w001# kubectl -n ims describe job $IMS_KUBERNETES_JOB
Name:           cray-ims-c4679389-a8d2-45d7-b983-028b388618d5-customize
...
Events:
Type    Reason            Age    From            Message
----    ------            ----   ----            -------
Normal  SuccessfulCreate  6m19s  job-controller  Created pod:
cray-ims-f5097693-5616-4495-9b5f-adfa1b4c49d9-customize-xxf74
```

The `Reason` column will read `SuccessfulCreate` and the text under the `Message` column will begin with `Created pod:`.

30. Save the full name of the pod to an environment variable. This name is given under the `Message` column in the output of the previous command.

    ```
    ncn-w001# export POD=cray-ims-f5097693-5616-4495-9b5f-adfa1b4c49d9-customize-xxf74
    ```

31. Confirm that the pod created by IMS is ready to be customized with the Spectrum Scale packages.

The pod is ready when the `status` value of the following command will is `waiting_on_user`. It may take several minutes to receive output.

```
ncn-w001# cray ims jobs describe $IMS_JOB_ID
artifact_id = "7642f972-a845-45f7-bfab-3c1e29d0a848"
build_env_size = 10
created = "2021-05-10T19:37:55.189446+00:00"
enable_debug = false
id = "f5097693-5616-4495-9b5f-adfa1b4c49d9"
image_root_archive_name = "gw_gpfs_custom_image"
initrd_file_name = "initrd"
job_type = "customize"
kernel_file_name = "vmlinuz"
kubernetes_configmap = "cray-ims-f5097693-5616-4495-9b5f-adfa1b4c49d9-configmap"
kubernetes_job = "cray-ims-f5097693-5616-4495-9b5f-adfa1b4c49d9-customize"
kubernetes_namespace = "ims"
kubernetes_service = "cray-ims-f5097693-5616-4495-9b5f-adfa1b4c49d9-service"
public_key_id = "0146386a-4eae-450b-9d59-04f9a5bf461a"
status = "waiting_on_user"
[[ssh_containers]]
jail = false
name = "customize"
status = "pending"

[ssh_containers.connection_info."cluster.local"]
host = "cray-ims-f5097693-5616-4495-9b5f-adfa1b4c49d9-service.ims.svc.cluster.local"
port = 22
[ssh_containers.connection_info.customer_access]
host = "f5097693-5616-4495-9b5f-adfa1b4c49d9.ims.baldar.dev.cray.com"
port = 22
```

The `status` value of the previous command will read `waiting_on_user`. It may take several minutes to receive output.

32. SSH into the pod that has the customized NCN-GW image mounted.

    ```
    ncn-w001# ssh -p $IMS_SSH_PORT root@$IMS_SSH_HOST
    [root@POD ~]#
    ```

33. Copy the Spectrum Scale install files into the mounted NCN-GW image root.

    ```
    [root@POD ~]# scp root@ncn-w001:/tmp/GPFS5/* /mnt/image/image-root/tmp
    ```

34. Copy all files from the Spectrum Scale gateway node directory `/root/.ssh` to the mounted NCN-GW image root directory `/mnt/image/image-root/root/.ssh`, including the generated ssh key files for the gateway node.

35. Copy the `/etc/hosts` file from the Spectrum Scale gateway node to the mounted NCN-GW image root.

36. Chroot into that image root.

    ```
    [root@POD ~]# chroot /mnt/image/image-root
    ```

37. Get the kernel release of the current pod.

    ```
    :/ # uname -r
    5.3.18-24.49-default
    ```

38. Create a symbolic link with that release name, but that points to the kernel release in the directory `/lib/modules`.

```
:/ # cd /lib/modules
:/ # ls
4.12.14-197.78_9.1.58-cray_shasta_c
:/ # ln -s 4.12.14-197.78_9.1.58-cray_shasta_c 5.3.18-24.49-default
```

39. Install Spectrum Scale RPMs, including the portability layer, on the NCN-GW node image.

```
:/ # cd /tmp
:/ # rpm -ivh ./gpfs*.rpm
warning: ./gpfs.adv-5.1.0-3.x86_64.rpm: Header V3 RSA/SHA256 Signature, key ID efeb6eeb: NOKEY
Preparing...                          ################################# [100%]
Updating / installing...
    1:gpfs.msg.en_US-5.1.0-3          ################################# [  9%]
    2:gpfs.gskit-8.0.55-12            ################################# [ 18%]
    3:gpfs.docs-5.1.0-3               ################################# [ 27%]
    4:gpfs.afm.cos-1.0.0-1            ################################# [ 36%]
Error, do this: mount -t proc proc /proc
    5:gpfs.base-5.1.0-3               ################################# [ 45%]
Failed to connect to bus: No such file or directory
Failed to connect to bus: No such file or directory
Failed to connect to bus: No such file or directory
Failed to connect to bus: No such file or directory
Failed to connect to bus: No such file or directory
Failed to connect to bus: No such file or directory
Failed to connect to bus: No such file or directory
Created symlink /etc/systemd/system/multi-user.target.wants/mmautoload.service ->
/usr/lib/systemd/system/mmautoload.service.
Created symlink /etc/systemd/system/multi-user.target.wants/mmccrmonitor.service ->
/usr/lib/systemd/system/mmccrmonitor.service.
    6:gpfs.adv-5.1.0-3                ################################# [ 55%]
    7:gpfs.license.dm-5.1.0-3         ################################# [ 64%]
    8:gpfs.compression-5.1.0-3        ################################# [ 73%]
    9:gpfs.crypto-5.1.0-3             ################################# [ 82%]
   10:gpfs.gpl-5.1.0-3                ################################# [ 91%]
   11:gpfs.gplbin-4.12.14-197.78_9.1.58################################# [100%]
```

40. Remove the kernel release symlink

```
:/ # rm /lib/modules/5.3.18-24.49-default
```

41. Delete the Spectrum Scale RPMs in both the NCN-GW image root mounted by the pod and on ncn-w001. Then finish the image customization session.

    1. Delete the Spectrum Scale RPMs in the `/tmp` directory of the pod.

       ```
       :/ # rm /tmp/gpfs*.rpm
       ```

    2. Escape the `chroot` environment of the customized NCN-GW image by pressing the **Ctrl** and **D** keys simultaneously.

    3. Notify IMS that the image customization session is complete.

       ```
       [root@POD /]# touch /mnt/image/complete
       Connection to c4679389-a8d2-45d7-b983-028b388618d5.ims.groot.dev.cray.com closed.
       ```

    4. Delete the copy of the Spectrum Scale RPMs on ncn-w001.

       ```
       ncn-w001# rm -rf /tmp/GPFS5
       ```

42. Verify that all the customized image artifacts are properly uploaded to S3 and registered with IMS. Perform this check in the original ncn-w001 terminal where the `IMS_` and `POD` environment variables were exported.

    A successful image customization process produces a customized root file system (in SquashFS format), kernel, and initrd. Each of these will have `etag`, `path`, `md5`, and `type` values. These values for each artifact are recorded in the pod logs. The following command example output is a sample of the full output and has been shortened for clarity.

```
ncn-w001# kubectl -n ims logs -f $POD -c buildenv-sidecar
Copying SMS CA Public Certificate to target image root
Running user shell for customize action
Image customization build environment is ready.
Use the following command to signal that image customization is complete in this container:
EXIT_COMMAND='touch /mnt/image/complete'
. . .
Waiting for SSH container to set /mnt/image/exiting flag
SSH Shell is exiting; tearing down build environment
Removing /mnt/image/exiting
+ source /scripts/helper.sh
+ IMS_PYTHON_HELPER_TIMEOUT=720
+ IMAGE_ROOT_PARENT=/mnt/image
+ IMAGE_ROOT_DIR=/mnt/image/image-root
+ KERNEL_FILENAME=vmlinuz
+ INITRD_FILENAME=initrd
+ IMAGE_ROOT_ARCHIVE_NAME=gw_gpfs_custom_image
+ set_job_status packaging_artifacts
+ local 'status=packaging_artifacts'
+ python3 -m ims_python_helper image set_job_status 2297a3fa-7149-4f2e-9eb9-fd982ff4fa44
packaging_artifacts
. . .
Exportable Squashfs 4.0 filesystem, gzip compressed, data block size 131072
        compressed data, compressed metadata, compressed fragments,
        compressed xattrs, compressed ids
        duplicates are removed
Filesystem size 1713931.36 Kbytes (1673.76 Mbytes)
        39.52% of uncompressed filesystem size (4337162.57 Kbytes)
Inode table size 946734 bytes (924.54 Kbytes)
        29.75% of uncompressed inode table size (3181815 bytes)
Directory table size 901553 bytes (880.42 Kbytes)
        37.81% of uncompressed directory table size (2384193 bytes)
. . .
{
    "ims_image_artifacts": [
        {
            "link": {
                "etag": "f584b84d382fee290c536831242fbe78-210",
                "path": "s3://boot-images/f2a2a5dd-b82f-4e30-955e-4ce8849bf0dd/rootfs",
                "type": "s3"
            },
            "md5": "af20afa34ab772babd412bcb92f9b3a4",
            "type": "application/vnd.cray.image.rootfs.squashfs"
        },
        {
            "link": {
                "etag": "175f0c1363c9e3a4840b08570a923bc5",
                "path": "s3://boot-images/f2a2a5dd-b82f-4e30-955e-4ce8849bf0dd/kernel",
                "type": "s3"
            },
            "md5": "175f0c1363c9e3a4840b08570a923bc5",
            "type": "application/vnd.cray.image.kernel"
        },
        {
            "link": {
                "etag": "dcd689d7b529f8c0cd1eb2a4b249e866-5",
                "path": "s3://boot-images/f2a2a5dd-b82f-4e30-955e-4ce8849bf0dd/initrd",
                "type": "s3"
```

```
            },
            "md5": "e72106383bcab23d3b17534f80411722",
            "type": "application/vnd.cray.image.initrd"
        },
        {
            "link": {
                "etag": "58c951bed5de80a3d7470921ebadef90",
                "path": "s3://boot-images/f2a2a5dd-b82f-4e30-955e-4ce8849bf0dd/manifest.json",
                "type": "s3"
            },
            "md5": "58c951bed5de80a3d7470921ebadef90",
            "type": "application/json"
        }
    ],
    "ims_image_record": {
        "created": "2021-03-29T15:02:27.464387+00:00",
        "id": "f2a2a5dd-b82f-4e30-955e-4ce8849bf0dd",
        "link": {
            "etag": "58c951bed5de80a3d7470921ebadef90",
            "path": "s3://boot-images/f2a2a5dd-b82f-4e30-955e-4ce8849bf0dd/manifest.json",
            "type": "s3"
        },
        "name": "gw_gpfs_custom_image"
    },
```


43. Verify updated root image.

```
ncn-w001# cray ims jobs describe $IMS_JOB_ID
artifact_id = "d237da41-e29e-4f4f-9b51-df9410777ae2"
build_env_size = 10
created = "2021-03-29T14:21:24.154361+00:00"
enable_debug = false
id = "2297a3fa-7149-4f2e-9eb9-fd982ff4fa44"
image_root_archive_name = "gw_gpfs_custom_image"
initrd_file_name = "initrd"
job_type = "customize"
kernel_file_name = "vmlinuz"
kubernetes_configmap = "cray-ims-2297a3fa-7149-4f2e-9eb9-fd982ff4fa44-configmap"
kubernetes_job = "cray-ims-2297a3fa-7149-4f2e-9eb9-fd982ff4fa44-customize"
kubernetes_namespace = "ims"
kubernetes_service = "cray-ims-2297a3fa-7149-4f2e-9eb9-fd982ff4fa44-service"
public_key_id = "5aab6259-4f04-4681-bd5a-ab78ef6cf58f"
resultant_image_id = "f2a2a5dd-b82f-4e30-955e-4ce8849bf0dd"
status = "success"
[[ssh_containers]]
jail = false
name = "customize"
status = "pending"

[ssh_containers.connection_info."cluster.local"]
host = "cray-ims-2297a3fa-7149-4f2e-9eb9-fd982ff4fa44-service.ims.svc.cluster.local"
port = 22
[ssh_containers.connection_info.customer_access]
host = "2297a3fa-7149-4f2e-9eb9-fd982ff4fa44.ims.groot.dev.cray.com"
port = 22
```

44. Save the `resultant_image_id` value returned in the previous step to an environment variable.

```
ncn-w001# export IMS_RESULTANT_IMAGE_ID=f2a2a5dd-b82f-4e30-955e-4ce8849bf0dd
```

45. Verify that the new customized NCN-GW image record exists.

```
ncn-w001# cray ims images describe $IMS_RESULTANT_IMAGE_ID
created = "2021-03-29T15:02:27.464387+00:00"
id = "f2a2a5dd-b82f-4e30-955e-4ce8849bf0dd"
name = "gw_gpfs_custom_image"

[link]
etag = "58c951bed5de80a3d7470921ebadef90"
path = "s3://boot-images/f2a2a5dd-b82f-4e30-955e-4ce8849bf0dd/manifest.json"
type = "s3"
```

46. Clean up the image customization environment.

    1. Delete the IMS job record for the image customization session.

       ```
       ncn-w001# cray ims jobs delete $IMS_JOB_ID
       ```

    2. Confirm that the IMS job record no longer exists.

       The following command will not return anything if the IMS job record was deleted.

       ```
       ncn-w001# cray ims jobs list | grep $IMS_JOB_ID
       ```

Copy and create a new configuration to recover the Spectrum Scale configuration and mount after node reboot

47. Create and apply the new `gpfs-mount` role in the Gateway branch in the `uan-config-management` VCS repository.

    1. Create a new file `roles/gpfs-mount/tasks/main.yml` with the following contents:

       ```
       - name: Wait for hostid
         shell: hostid
         register: result
         until: result.stdout is not regex("^0+$")
         retries: 360
         delay: 10

       - name: mmsdrrestore
         shell: /usr/lpp/mmfs/bin/mmsdrrestore -p hi1

       - name: mmstartup
         shell: /usr/lpp/mmfs/bin/mmstartup
       ```

    2. Apply the changes in the diff below to the `site.yml` file.

       ```
       ncn-m001# diff --git a/site.yml b/site.yml
       index e8fee04..90a4383 100644
       --- a/site.yml
       +++ b/site.yml
       @@ -66,6 +66,9 @@
          - role: rebuild-initrd
            when: cray_cfs_image | default(false) | bool

       +    - role: gpfs-mount
       +      when: not cray_cfs_image|default(false)|bool
       +
          tasks:

          - import_tasks: playbooks/nvidia_deploy.yml
       ```

48. Commit the updated Ansible roles to the branch.

    ```
    ncn-m001# git add roles/gpfs-mount
    ncn-m001# git commit -a
    ```

49. Obtain the VCS password.

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

50. Push the change to the repository. When prompted for username and password, specify the username “crayvcs”, with the password that was obtained in the previous step.

    ```
    ncn-m001# git push origin
    ```

51. Obtain the most recent git commit hash for this branch, made in the previous step, and save it for later use.

    ```
    ncn-m001# git rev-parse --verify HEAD
    be2bd370ed246182ec765ca999d2fac4d355443a
    ncn-m001# export HEAD_COMMIT_HASH=be2bd370ed246182ec765ca999d2fac4d355443a
    ```

Create a BOS session template to boot the NCN-GW node(s) with the Spectrum Scale-enabled image

52. Make a copy of the BOS session template that is currently used to boot the NCN-GW node(s).

    1. Search for the original BOS session template name that the NCN-GW node or nodes are running with.

       ```
       ncn-m001# cray bos sessiontemplate list --format json | less
       ```

    2. Search for the `path` value the contains the image ID from Step 25 and note the `name` value.

       ```
       {
       "boot_sets": {
           "uan": {
           "boot_ordinal": 2,
           "etag": "a23718205a36ce7432c0260b2aa1b681",
           "kernel_parameters": "ip=dhcp quiet spire_join_token=${SPIRE_JOIN_TOKEN}",
           "network": "nmn",
           "node_list": [
               "x3000c0s27b0n0"
           ],
           "path": "s3://boot-images/d237da41-e29e-4f4f-9b51-df9410777ae2/manifest.json",
           "rootfs_provider": "cpss3",
           "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:nmn0",
           "type": "s3"
           }
       },
       "cfs": {
           "configuration": "uan-config-2.0.0"
       },
       "enable_cfs": true,
       "name": "uan-sessiontemplate-2.0.0"
       },
       ```

    3. Copy the sessiontemplate that matches the `name` identified in the previous sub-step.

       ```
       ncn-w001# cray bos sessiontemplate describe uan-sessiontemplate-2.0.0 --format json > \
       create_session_template
       ```

       The contents of the `create_session_template` file will be similar to the following:

       ```
       {
           "boot_sets": {
               "uan": {
               "boot_ordinal": 2,
               "etag": "5851e6d2ce8c70f641244c7c534a20e1",
               "kernel_parameters": "ip=dhcp quiet spire_join_token=${SPIRE_JOIN_TOKEN}",
               "network": "nmn",
               "node_list": [
                   "x3000c0s27b0n0"
               ],
               "path": "s3://boot-images/d237da41-e29e-4f4f-9b51-df9410777ae2/manifest.json",
       ```

```
            "rootfs_provider": "cpss3",
            "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:nmn0",
            "type": "s3"
            }
        },
        "cfs": {
            "configuration": "uan-config-2.0.0"
        },
        "enable_cfs": true,
        "name": "uan-sessiontemplate-2.0.0"
    }
```

53. Copy the latest CFS configuration for UANs.

    ```
    root-m001# cray cfs configurations describe --format json uan-config-2.0.0 > \
    spectrum-scale-uan-config-2.0.0.json
    ```

54. Update the configuration with the git commit hash obtained from the “Obtain the most recent git commit hash...” step above. Remove `"lastUpdated"`, `"name"`, and the comma after `"layers"`.

    ```
    ncn-m001# echo ${HEAD_COMMIT_HASH}
    be2bd370ed246182ec765ca999d2fac4d355443a
    ncn-m001# vi spectrum-scale-uan-config-2.0.0.json
    ...
    ```

55. Update the CFS configuration with your new configuration.

    ```
    ncn-m001# cray cfs configurations update uan-config-2.0.0 --file spectrum-scale-uan-config-2.0.0.json
    ```

56. Open the BOS session template copy in an editor.

    ```
    ncn-w001# vi create_session_template
    ```

57. Modify the copied BOS session template so that it can boot the NCN-GW nodes with the Spectrum Scale image.

    1. Change the `name` value to one that is unique, descriptive, and indicates the base image that was modified to produce the Spectrum Scale image. A name of `gw-sessiontemplate-2.0.0-Spectrum_Scale_client` is a good name for the Spectrum Scale session template based on the prior version name.
    2. Change the `path` and `etag` values to match the values for the ‘manifest.json’ entry obtained in Step 42 above.
    3. Ensure that the `configuration` name matches the configuration used to boot the NCN-GW nodes. This is the same configuration that was updated in Step 54 above.
    4. Update the `node_list` to contain only the xnames for gateway nodes that will run as GPFS clients.
    5. If you have any gateway nodes that will not run as GPFS clients, update the default BOS session template that those nodes use, to exclude the GPFS client nodes from that `node_list`.

58. Upload the new Spectrum Scale session template to BOS. Replace *TEMPLATE_NAME* in the command below with the template name chosen in Step 57.1

    ```
    ncn-w001# cray bos sessiontemplate create --name TEMPLATE_NAME --file ./create_session_template
    ```

59. Verify that the new Spectrum Scale session template successfully uploaded to BOS. Replace *TEMPLATE_NAME* with the template name chosen in Step 57.1.

    ```
    ncn-w001# cray bos sessiontemplate describe TEMPLATE_NAME --format json
    ```

    All the values changed in Step 57 will appear in the command output.

6.1.2 Deploy and Test Spectrum Scale NCN-GW Image

PREREQUISITES

- Create Spectrum Scale NCN Gateway Node Image

OBJECTIVE

This procedure guides system administrators and installers through deploying Spectrum Scale software on all NCN-GW nodes in an HPE Cray EX supercomputer.

LIMITATIONS

This procedure provides HPE Cray EX-specific instructions for deploying Spectrum Scale software on HPE Cray EX NCN-GW nodes. These steps are not intended to replace the instructions and guidance provided on the IBM website (www.ibm.com).

PROCEDURE

Boot NCN-GW nodes with the Spectrum Scale-customized image

1. Reboot the NCN-GW nodes to reboot with the new Spectrum Scale image. Replace *SPECTRUM_SCALE_SESSIONTEMPLATE_NAME* in the following command with the session templatename chosen in Step 57.1 of Create Spectrum Scale NCN Gateway Node Image.

   ```
   ncn-w001# cray bos session create --template-uuid SPECTRUM_SCALE_SESSIONTEMPLATE_NAME \
   --operation reboot
   operation = "reboot"
   templateUuid = "Spectrum_Scale_image_name"
   [[links]]
   href = "2aa74bbb-b06d-48ba-b5dc-6f204cb1c9c5"
   type = "GET"
   rel = "session"
   jobId = "boa-2aa74bbb-b06d-48ba-b5dc-6f204cb1c9c5"
   ```

2. Verify that the BOS session is running.

   ```
   ncn-w001# cray bos session list
   results = [ "2aa74bbb-b06d-48ba-b5dc-6f204cb1c9c5", "1fd70750-2aa0-4f20-8113-cdb1c23f38cd",]
   ```

   The command output includes a string that matches the `jobId` value returned in the previous step.

3. Describe the BOS session to see the status of the job. In the following command use the ID portion of the `jobId` value returned in Step 1.

   ```
   ncn-w001# cray bos session describe 2aa74bbb-b06d-48ba-b5dc-6f204cb1c9c5
   bos_launch = "2020-02-07 23:06:45.802140"
   uans = "shutdown_finished"
   operation = "reboot"
   session_template_id = "Spectrum_Scale_sessiontemplate_name"
   boa_launch = "2020-02-07 23:06:52.287819"
   stage = "Done"
   ```

4. Obtain the full name of the BOA pod.

   ```
   ncn-w001# kubectl get pods -n services | grep boa-2aa74bbb-b06d-48ba-b5dc-6f204cb1c9c5
   boa-2aa74bbb-b06d-48ba-b5dc-6f204cb1c9c5-mnnck                  2/2     Running     0          2m44s
   ```

5. Use the full name of the BOA pod to observe its log until the pod finishes.

   ```
   ncn-w001# kubectl logs -n services -c boa -f boa-2aa74bbb-b06d-48ba-b5dc-6f204cb1c9c5-mnnck
   ...
   2021-06-15 20:37:31,952 - INFO    - cray.boa.smd.wait_for_nodes - Waiting 5 seconds for 1 node to be
   in state: Ready
   2021-06-15 20:37:37,010 - INFO    - cray.boa.smd.wait_for_nodes -
   Ready: [x3000c0s27b0n0]
   2021-06-15 20:37:37,037 - INFO    - cray.boa.agent - Waiting on completion of configuration...
   2021-06-15 20:37:37,105 - INFO    - cray.boa.cfsclient - 1 unconfigured nodes
   ...
   2021-06-15 20:37:52,315 - INFO    - cray.boa.cfsclient - 1 unconfigured nodes
   2021-06-15 20:42:42,056 - INFO    - cray.boa.cfsclient - 1 unconfigured nodes
   2021-06-15 20:42:57,354 - INFO    - cray.boa.agent - BOA Agent
   (Session f3eea6ba-9d3b-4523-9ab5-fa913bfa928e Boot Set: uan) finished.
   2021-06-15 20:42:57,398 - INFO    - cray.boa - BOA completed requested operation.
   ```

6. Watch the node console log during the reboot. Replace *NCN_GW_XNAME* in the following command with the xname of a NCN-GW node that is rebooting.

Administrators can run the following command with different NCN-GW xnames to monitor the reboot process one each of the NCN-GW nodes.

```
ncn-w001# kubectl exec -it -n services cray-conman-98994bdc-2gddk -- conman -j  NCN_GW_XNAME
```

7. Delete the BOS session after it finishes.

   ```
   ncn-w001# cray bos session delete 2aa74bbb-b06d-48ba-b5dc-6f204cb1c9c5
   ```

8. Verify that the Spectrum Scale client software is installed on all NCN-GW nodes. Replace *NCN_GW_SS_LIST* with a comma-separated list of the xnames of all the NCN-GW nodes that were rebooted with the Spectrum Scale-enabled image.

   ```
   ncn-w001# ALL_NCN_GWS=$(<list of NCN-GW Spectrum Scale client names>)
   ncn-w001# pdsh -w ${ALL_NCN_GWS} "rpm -qa gpfs.gplbin"
   gw01: gpfs.gplbin-4.12.14-197.40_9.1.36-cray_shasta_c-5.1.0-3
   gw02: gpfs.gplbin-4.12.14-197.40_9.1.36-cray_shasta_c-5.1.0-3
   ...
   ```

Configure and test Spectrum Scale environment

9. Configure the site Spectrum Scale client and server environment. Refer to site documentation and www.ibm.com for further guidance.
10. Verify that the Spectrum Scale client software is functional. Refer to site documentation and www.ibm.com for further guidance.
11. Configure DVS server on the NCN-GW node to project the GPFS filesystem. The example shown is for a one-time mount that will not exist after a reboot of either the DVS server or the DVS client on the compute node. See Project an External Filesystem to Compute Nodes or User Access Nodes for in-depth instructions on making mounts persistant across reboots. See the Tune and Optimize DVS section for mount options. This example utilizes mount options for a read only filesystem projection.

    ```
    gw01# mount -t dvs -o loadbalance,attrcache_timeout=14400,noclusterfs,ro,nodename=x3000c0s7b0n0:x3000c0s9
    ```

12. Configure the DVS client on the compute node to mount the DVS projected GPFS filesystem. The example shown is for a one-time mount that will not exist after a reboot of either the DVS server or the DVS client on the compute node. See Project an External Filesystem to Compute Nodes or User Access Nodes for more in-depth instructions on making mounts persistant across reboots.
    1. Create mount point directory for DVS projected GPFS filesystem.
       ```
       nid000003# mkdir /tmp/my-gpfs-path
       ```
    2. Mount the DVS projected GPFS filesystem. mount -t dvs -o path,server_list,options remote_gpfs_path local_path
       ```
       nid000003# mount -t dvs -o path=/tmp/my-gpfs-path,
       nodename=x3000c0s7b0n0:x3000c0s9b0n0:x3000c0s24b0n0,
       maxnodes=1,noclusterfs /tmp/remote-gpfs-path /tmp/my-gpfs-path
       ```

7 Configure Lustre

7.1 Mount a Lustre File System on NCNs

Edit several Ansible playbooks and configuration files to configure Lustre before mounting it on NCNs.

- Familiarity with Git workflows and the Configuration Framework Service (CFS).
- Determine the LNet network name used by the Lustre file system. If this name is the same as the default LNet network name used by DVS (tcp99), then perform Change the Name of the DVS LNet Network first before performing this procedure.
- The Cray command line interface (CLI) tool is initialized and configured on the system. See "Configure the Cray Command Line Interface (CLI)" in the CSM documentation for more information.

1. Obtain the password for the `crayvcs` user.

   The `crayvcs` username and password is required for the next step.

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

2. Clone the system Version Control Service (VCS) repository to a directory on the system and change to the config-management directory.

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

3. Obtain a list of the branches of the cos-config-management VCS repository on the HPE Cray EX system.

   Determine which branch listed in the output is currently used to configure the system.

   ```
   ncn-m001# git ls-remote https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git
   refs/heads/cray/cos/*
   4f8e919ede7143929e26ee8d97e04c12572a2f90 refs/heads/cray/cos/1.4.0
   ccd964f8eeeb52ab8f895b480c5d1142c7bc0a8e refs/heads/cray/cos/2.0.17
   035ab5d400c22df123f9bd4962a5a8c1013a2703 refs/heads/cray/cos/2.1.13-20210212225706-6218fd1
   . . .
   ```

4. Create a local branch that corresponds to the current COS release version used by the HPE Cray EX system.

   The following example assumes that the current COS release is 2.0.17. Use the branch name determined in Step 3. A local branch is required for the changes since VCS prevents updates to original remote branch.

   ```
   ncn-m001# git checkout cray/cos/2.0.17
   Branch 'cray/cos/2.0.17' set up to track remote branch 'cray/cos/2.0.17' from 'origin'.
   Switched to a new branch 'cray/cos/2.0.17'
   ncn-m001# git checkout -b local_cos.2.0.17 cray/cos/2.0.17
   Switched to a new branch 'local_cos.2.0.17'
   ```

5. Edit the `lnet_conf` variables to change the LNet configuration on the nodes to include the LNet network that Lustre will be using. If Lustre will be using a numbered network (e.g. tcp2, o2ib3, etc), you should use that network instead of plain “tcp” or “o2ib”.

   Add the appropriate stanza to the `group_vars/Management_Worker/lnet.yml` file. If the file doesn't exist, you may need to create it from the default `lnet_conf` in `group_vars/all/lnet.yml` or `roles/cray_lnet_load/defaults/main.yml` (in that order).

   In the following example stanza, a `tcp` network type using the `hsn0` network is added after the vi editing session.

   ```
   ncn-m001# cat group_vars/Management_Worker/lnet.yml
   lnet_conf: |
     ip2nets:
       - net-spec: tcp99
         interfaces:
           0: bond0.nmn0
   ...
   ncn-m001# vi group_vars/Management_Worker/lnet.yml
   . . .
   ncn-m001# cat group_vars/Management_Worker/lnet.yml
   lnet_conf: |
     ip2nets:
       - net-spec: tcp99
         interfaces:
           0: bond0.nmn0
       - net-spec: tcp
         interfaces:
           0: hsn0
           1: hsn1
           2: hsn2
           3: hsn3
   ...
   ```

   This is suitable for accessing lustre over the HSN using ksocklnd while DVS continues to operate over the NMN. If your lustre filesystem uses ko2iblnd, instead of adding the tcp stanza above, you would want to add a stanza for o2ib, as follows:

   ```
   ncn-m001# cat group_vars/Management_Worker/lnet.yml
   lnet_conf: |
     ip2nets:
       - net-spec: tcp99
         interfaces:
           0: bond0.nmn0
   ```

```
...
ncn-m001# vi group_vars/Management_Worker/lnet.yml
. . .
ncn-m001# cat group_vars/Management_Worker/lnet.yml
lnet_conf: |
  ip2nets:
    - net-spec: tcp99
      interfaces:
        0: bond0.nmn0
    - net-spec: o2ib
      interfaces:
        0: hsn0
        1: hsn1
        2: hsn2
        3: hsn3
...
```

Note: if you have configured DVS for HSN, you may not have the tcp99 stanza at all, and may already have the correct stanzas here for the HSN. At this time, we recommend using the same HSN configuration for lustre and DVS if DVS is going to run on the HSN. For more information on running DVS over the HSN, please see: DVS over HSN Deployment.

If you have an Arista switch, make the same changes to the file `roles/cray_lnet_load/files/lnet-router.conf`.

6. If you have an Arista switch, the NCNs must be able to talk to the external Lustre cluster through that switch attached to HSN network.
   Note: Skip this section (6) if you do not have an Arista switch, e.g. if your Lustre cluster is DirectConnect.

   The following content may already exist because you first performed this procedure on CNs.

   1. Edit the `main.yml` file in the `tasks` directory in the `lustre_config` role. In it, create an IP route to the NCNs in the new file so they can talk to the cluster.

      ```
      ncn-m001# vi roles/lustre_config/tasks/main.yml
      - name: Add route to Lustre FS
        command: ip route replace 10.10.0.0/16 via 10.253.254.250 dev hsn0
        when: not cray_cfs_image|default(false)|bool
      ```

   2. Add the files and directories with Git.

      ```
      ncn-m001# git add roles/lustre_config
      ```

7. Modify the ncn.yml playbook to include the `cray_lustre_load` role. In addition, include the `lustre_config` role if your Lustre system has an Arista switch.

   Make sure that new roles are run when LNet is loaded by adding them to the `ncn.yml` file. The stanza will look like the following if you have an Arista switch:

   ```
   ncn-m001# cat ncn.yml
   #!/usr/bin/env ansible-playbook
   # Copyright 2021 Hewlett Packard Enterprise Development LP
   ---
   - hosts: Management_Worker
     any_errors_fatal: false
     remote_user: root
     roles:
       - cray_lnet_install
       - cray_dvs_install
       - lustre_config
       - cray_lnet_load
       - cray_dvs_load
       - cray_lustre_load
       - configure_fs
   ```

   The file will have one less line if you do not have an Arista switch.

8. Edit the `filesystems` variable to configure the mount of the Lustre file system.

   Add a section to the to the `group_vars/Management_Worker/filesystems.yml` file. Make sure the `src`, `mount_point`, and `opts` fields are set for the file system being used. Multiple file systems can be used if wanted.

   ```
   ncn-m001# vi group_vars/Management_Worker/filesystems.yml
   ...
   ncn-m001# cat group_vars/Management_Worker/filesystems.yml
   filesystems:
   - src: "10.10.100.4@tcp:10.10.100.5@tcp:/cls01053"
     mount_point: /lus/cls01053
     fstype: lustre
     opts: rw,noauto,flock,lazystatfs
     state: mounted
   ```

   NOTE: This example is for a ksocklnd-based Lustre filesystem. Hence the `@tcp` suffixes to the IP addresses in the `- src` line. If you are using ko2iblnd, the suffixes should be `@o2ib`.

   Hewlett Packard Enterprise recommends including noauto in the `opts` variable. Without the noauto flag, Linux will try to mount the file system before all the correct initialization takes place. For example, a node may try to mount Lustre before the proper IP and LNet routes are set up and fail.

9. Commit the changes and push them into the remote branch identified in Step 3.

   ```
   ncn-m001# git commit -am 'Added Lustre file system info for NCN worker nodes'
   ncn-m001# git push --set-upstream origin local_cos.2.0.17
   ```

10. Obtain the most recent git commit hash for this branch, made in the previous step, and save it for later use.

    ```
    ncn-m001# git rev-parse --verify HEAD
    be2bd370ed246182ec765ca999d2fac4d355443a
    ncn-m001# export HEAD_COMMIT_HASH=be2bd370ed246182ec765ca999d2fac4d355443a
    ```

11. Navigate out of the cos-config-management directory and download the current NCN Personalization configuration.

    ```
    ncn-m001# cd ..
    ncn-m001# cray cfs configurations describe ncn-personalization --format json > ncn-personalization.json
    ncn-m001# cat ncn-personalization.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-2.0.17",
          "playbook": "ncn.yml"
        }
      ],
      "name": "ncn-personalization"
    }
    ```

12. Edit and update the JSON file.

    a. Remove the `lastUpdated` key.

    b. Remove the second `name` key (the one whose value is `ncn-personalization`).

    c. Remove the comma after the `layers` array.

    d. Update the git commit hash with the one obtained in the “Obtain the most recent git commit hash...” step above.

    The JSON file will resemble this one:

    ```
    ncn-m001# echo ${HEAD_COMMIT_HASH}
    be2bd370ed246182ec765ca999d2fac4d355443a
    ncn-m001# vi ncn-personalization.json
    ```

```
. . .
ncn-m001# cat ncn-personalization.json
{
  "layers": [
    {
      "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
      "commit": "be2bd370ed246182ec765ca999d2fac4d355443a",
      "name": "cos-integration-2.0.17",
      "playbook": "ncn.yml"
    }
  ]
}
```

13. Unload DVS and LNet services if they are currently running on worker NCNs.

Refer to the ***Unload DVS and LNet services for the worker nodes*** step in the ***Perform NCN NCN Personalization*** procedure of the publication ***HPE Cray Operating System Installation Guide CSM on HPE Cray EX Systems***.

14. Upload the new configuration into CFS.

    ```
    ncn-m001# cray cfs configurations update ncn-personalization --file ncn-personalization.json \
    --format json
    {
      "lastUpdated": "2020-12-17T17:51:09Z",
      "layers": [
        {
          "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
          "commit": "be2bd370ed246182ec765ca999d2fac4d355443a",
          "name": "cos-integration-2.0.17",
          "playbook": "ncn.yml"
        }
      ],
      "name": "ncn-personalization"
    }
    ```

    There is no need to update the CFS components for the NCNs. Those components still refer to the `ncn-personalization` configuration by name.

15. Execute the following command until the NCN Personalization session triggered by the previous step to complete.

The output of the following command displays `configured` for the configuration status when NCN Personalization session is complete.

```
ncn-m001# cray cfs components describe --format json W001_XNAME
```

7.2 Mount a Lustre File System on CNs and UANs

- **PREQUISITES**
- Familiarity with Git workflows and a Git client compatible with the Git server provided by the Configuration Framework Service (CFS).
- General Ansible familiarity with use of the group_vars directory for setting variables for different types of node groups.
- Determine the LNet network name used by the Lustre file system. If this name is the same as the default LNet network name used by DVS (tcp99), then perform Change the Name of the DVS LNet Network first before performing this procedure.
- Perform Mount a Lustre File System on NCNs
- **OBJECTIVE**

  Configure and enable a Lustre file system on compute nodes or User Access Nodes (UANs) using Git and Ansible.

  Important: When setting up LNet in this procedure, ensure that the correct network is being configured. The LNet network must match the network name that is configured on the Lustre server. It is important to ensure that there are no LNet network conflicts in environments that consist of multiple LNet network configurations. In addition, a Lustre file system requires port policies to be

configured on the Slingshot switch. Consult the Slingshot Operations Guide, Section 5.6 Port Policies, to determine the port policies need to be configured for the type of file system.

1. Obtain the password for the crayvcs user.

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

2. Create a local copy of the HPE Cray OS (COS) configuration management repository. Then cd into it.

   Lustre mount configuration for CNs, UANs, and NCNs is done within HPE COS product. The crayvcs username and password is required.

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

   . . .

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

3. Obtain a list of the branches of the cos-config-management VCS repository on the HPE Cray EX system. Then determine which branch is currently used to configure the system.

   ```
   ncn-m001# git ls-remote https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git \
   refs/heads/cray/cos/*
   4f8e919ede7143929e26ee8d97e04c12572a2f90    refs/heads/cray/cos/1.4.0
   ccd964f8eeeb52ab8f895b480c5d1142c7bc0a8e    refs/heads/cray/cos/2.0.17
   . . .
   ```

4. Create a local branch that corresponds to the current COS release version used by the HPE Cray EX system. Use the branch name determined in previous step.

   The following example assumes that the current COS release is 2.0.17. A local branch is required for the changes since VCS prevents updates to original remote branch in this release.

   ```
   ncn-m001# git checkout cray/cos/2.0.17
   Branch 'cray/cos/2.0.17' set up to track remote branch 'cray/cos/2.0.17' from 'origin'.
   Switched to a new branch 'cray/cos/2.0.17'
   ncn-m001# git checkout -b local_cos.2.0.17 cray/cos/2.0.17
   Switched to a new branch 'local_cos.2.0.17'
   ```

5. Edit the lnet_conf variables to change the LNet configuration on the nodes to include the LNet network that Lustre will be using. If Lustre will be using a numbered network (e.g. tcp2, o2ib3, etc), you should use that network instead of plain “tcp” or “o2ib”.

   Add the appropriate stanza to the group_vars/Compute/lnet.yml file. If configuration of Lustre on UANs is required, the group_vars/Application_UAN/lnet.yml file should also be similarly modified (in the UAN repo). If a file doesn't exist, you may need to create it from the default lnet_conf in group_vars/all/lnet.yml or roles/cray_lnet_load/defaults/main.yml (in that order).

   In the following example stanza, a tcp network type using the hsn0 network is added after the vi editing session.

   ```
   ncn-m001# cat group_vars/Compute/lnet.yml
   lnet_conf: |
     ip2nets:
       - net-spec: tcp99
         interfaces:
           0: nmn0
   ...
   ncn-m001# vi group_vars/Compute/lnet.yml
   . . .
   ncn-m001# cat group_vars/Compute/lnet.yml
   lnet_conf: |
     ip2nets:
       - net-spec: tcp99
         interfaces:
   ```

```
        0: nmn0
    - net-spec: tcp
      interfaces:
        0: hsn0
        1: hsn1
        2: hsn2
        3: hsn3
...
```

This is suitable for accessing lustre over the HSN using ksocklnd while DVS continues to operate over the NMN. If your lustre filesystem uses ko2iblnd, instead of adding the tcp stanza above, you would want to add a stanza for o2ib, as follows:

```
ncn-m001# cat group_vars/Compute/lnet.yml
lnet_conf: |
  ip2nets:
    - net-spec: tcp99
      interfaces:
        0: nmn0
...
ncn-m001# vi group_vars/Compute/lnet.yml
. . .
ncn-m001# cat group_vars/Compute/lnet.yml
lnet_conf: |
  ip2nets:
    - net-spec: tcp99
      interfaces:
        0: nmn0
    - net-spec: o2ib
      interfaces:
        0: hsn0
        1: hsn1
        2: hsn2
        3: hsn3
...
```

Note: If you have configured DVS for HSN, you may not have the tcp99 stanza at all, and may already have the correct stanzas here for the HSN. At this time, we recommend using the same HSN configuration for lustre and DVS if DVS is going to run on the HSN. For more information on running DVS over the HSN, please see: DVS over HSN Deployment.

6. Define the Lustre file system.

   Create a file containing the mount information for the Lustre file system in the appropriate subdirectory of the `group_vars` directory. The following example is for configuring compute nodes, so the subdirectory is `Compute`. To configure UANs, change `Compute` to `Application_UAN`.

   If using an external file system, refer to Project an External Filesystem to Compute Nodes or User Access Nodes for more information on other types of entries in the `filesystems` array.

   Ensure that the correct network name is used for the LNet network. See the **OBJECTIVE** section for more information.

   The mount point must be in a subdirectory and not a directory in the root directory.

   Important: The use of Ansible `filesystems` data in this file might cause conflict if other types of file systems are needed, which use the same `filesystems` name for the Ansible data. It is best practice to only use a file name `filesystems.yml` for this purpose. Avoid using the name `lustre.yml`.

   ```
   ncn-m001# vi group_vars/Compute/filesystems.yml
   ...
   ncn-m001# cat group_vars/Compute/filesystems.yml
   filesystems:
       - src: "10.10.100.3@tcp:10.10.100.4@tcp:/cls12345"
         mount_point: /lus/cls12345
         fstype: lustre
   ```

```
    opts: rw,noauto,flock,lazystatfs
    state: mounted
```

NOTE: This example is for a ksocklnd-based Lustre filesystem. Hence the `@tcp` suffixes to the IP addresses in the `- src` line. If you are using ko2iblnd, the suffixes should be `@o2ib`.

7. If you have an Arista switch, your nodes must be able to talk to the external Lustre cluster through that switch attached to HSN network. **Note:** Skip this section (10) if you do not have an Arista switch, e.g. if your Lustre cluster is DirectConnect.

   The following content may already exist because you first performed this procedure on NCNs.

   1. Edit the `main.yml` file in the `tasks` directory of the `lustre_config` role.

      The file is listed below for reference. The content of this file will be site-specific. The IP addresses in the "lineinfile" command must be changed.

      Ensure that the correct network name is used for the LNet network. See the **OBJECTIVE** section for more information.

      ```
      ncn-m001# mkdir -p roles/lustre_config/tasks
      ncn-m001# vi roles/lustre_config/tasks/main.yml
      - name: Add route to Lustre FS
        command: ip route replace 10.10.0.0/16 via 10.253.254.250 dev hsn0
        when: not cray_cfs_image|default(false)|bool
      ```

   2. Add changes from the working directory to the staging area.

      ```
      ncn-m001# git add roles/lustre_config
      ```

8. Determine if multi-rail Lustre is enabled.

   A multi-rail Lustre setup uses two interfaces on a single node to increase throughput or to connect to more than one network. For more information about multi-rail Lustre, refer to the Lustre documentation.

   a. View a `lnet.yml` file in the `group_vars/` subdirectory.

   b. Check for more than one interface configured under the `lnet_conf` variable.

   If there is more than one interface, the system uses a multi-rail Lustre configuration.

   c. Repeat the previous two substeps for `lnet.yml` files within the `group_vars/` directory.

9. Disable peer discovery unless multi-rail Lustre is used.

   LNet peer discovery is disabled by adding the line `options lnet  lnet_peer_discovery_disabled=1` to the end of the `lnet_modprobe_conf` variable in the appropriate `lnet.yml` files in the `group_vars` directory.

   ```
   ncn-m001# cat group_vars/all/lnet.yml
   ...
   lnet_modprobe_conf: |
     options lnet lnet_transaction_timeout=120
     options ko2iblnd map_on_demand=1
   ncn-m001# vi group_vars/all/lnet.yml
   ...
   ncn-m001# cat group_vars/all/lnet.yml
   ...
   lnet_modprobe_conf: |
     options lnet lnet_transaction_timeout=120
     options ko2iblnd map_on_demand=1
     options lnet lnet_peer_discovery_disabled=1
   ```

   Repeat this step for the other `lnet.yml` files in the `group_vars` directory, such as `Application_UAN/lnet.yml`.

10. Edit the cos-config-management/site.yml file to include the `lustre_config` and `lustre_tuning` roles for compute nodes, by removing the comment character ("#") from the corresponding lines. Do **not** include the `lustre_config` role if the Lustre cluster does not have an Arista switch

    That section of the file should look like this:

```
. . .
- { role: lustre_config, when: not cray_cfs_image }
- { role: configure_fs, when: not cray_cfs_image|default(false)|bool }
- { role: lustre_tuning, when: not cray_cfs_image|default(false)|bool }
. . .
```

11. Add the change from the working directory to the staging area.

    ```
    ncn-m001# git add group_vars/
    ncn-m001# git add roles/lustre_tuning
    ncn-m001# git add site.yml
    ```

12. Commit the changes and push them into the remote branch identified in the “Obtain a list of the branches...” step.

    ```
    ncn-m001# git commit -am 'Added Lustre file system info'
    ncn-m001# git push --set-upstream origin local_cos.2.0.17
    ```

13. Obtain the most recent git commit hash for this branch, made in the previous step, and save it for later use.

    ```
    ncn-m001# git rev-parse --verify HEAD
    be2bd370ed246182ec765ca999d2fac4d355443a
    ncn-m001# export HEAD_COMMIT_HASH=be2bd370ed246182ec765ca999d2fac4d355443a
    ```

14. Obtain the name of the current CFS configuration for the CNs. Skip this step if the name of the current CFS configuration for the CNs is already known.

    Each BOS session template has a section indicating which CFS configuration is used.

    1. List all the BOS session templates.

       ```
       ncn-m001# cray bos sessiontemplate list --format json
       ```

    2. Search for the BOS session template used by the CNs.

    3. Print out the current BOS session template for the CNs.

    In the following example, the template name is cos-compute-2.0.17 and the lines indicating the current configuration for CNs are in bold.

    ```
    ncn-m001# cray bos sessiontemplate describe --format json cos-compute-2.0.17
    {
      "boot_sets": {
        "compute": {
          "boot_ordinal": 2,
          "etag": "391bf701610d6e8f4c25781f7265cba9",
          "kernel_parameters": "ip=dhcp quiet spire_join_token=${SPIRE_JOIN_TOKEN}",
          "network": "nmn",
          "node_roles_groups": [
            "Compute"
          ],
          "path": "s3://boot-images/f7a03fa5-7197-4f02-b385-146b4ed4b032/manifest.json",
          "rootfs_provider": "cpss3",
          "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:nmn0",
          "type": "s3"
        }
      },
         "cfs": {
         "configuration": "cos-config-2.0.17"


      },
      "enable_cfs": true,
      "name": "cos-compute-2.0.17"
    }
    ```

15. Navigate out of the cos-config-management directory and download the current CN configuration.

    Replace CN_CONFIG in the following command with name of the current CFS configuration for CNs that was obtained in the previous step.

    ```
    ncn-m001# cd ..
    ncn-m001# cray cfs configurations describe CN_CONFIG --format json > CN_CONFIG.json
    ```

    In the previous example, the CFS configuration name was `cos-config-2.0.17`:

    ```
    ncn-m001# cray cfs configurations describe cos-config-2.0.17 --format json > cos-config-2.0.17.json
    ncn-m001# cat 2.0.17.json
    {
      "lastUpdated": "2021-03-08T16:51:24Z",
      "layers": [
        {
          "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
          "commit": "ccd964f8eeeb52ab8f895b480c5d1142c7bc0a8e",
          "name": "cos-integration-2.0.17",
          "playbook": "site.yml"
        }
      ],
      "name": "cos-config-2.0.17"
    }
    ```

16. Edit and update the CFS configuration JSON file.

    1. Remove the `lastUpdated` key.
    2. Remove the second `name` key. This key is the one whose value is `cos-config-2.0.17` in the preceding example.
    3. Remove the comma after the `layers` array.
    4. Update the git commit hash with the one obtained in "Obtain the most recent git commit hash..." step above.

    The JSON file will now resemble the following example:

    ```
    ncn-m001# echo ${HEAD_COMMIT_HASH}
    be2bd370ed246182ec765ca999d2fac4d355443a
    ncn-m001# vi cos-config-2.0.17.json
    ...
    ncn-m001# # cat cos-config-2.0.17.json
    {
      "layers": [
        {
          "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git",
          "commit": "be2bd370ed246182ec765ca999d2fac4d355443a",
          "name": "cos-integration-2.0.17",
          "playbook": "site.yml"
        }
      ]
    }
    ```

17. Upload the new CN configuration to CFS.

    ```
    ncn-m001# cray cfs configurations update CN_CONFIG --file CN_CONFIG.json --format json
    ```

18. Create new CN images if LNet peer discovery was disabled. Skip this step if LNet peer discovery was not disabled.

    1. Run an IMS Image Customization session to create new CN images. Use the CN configuration name from the previous step.

       Refer to procedure "Create an Image Customization CFS Session" in the CSM documentation for reference.

    2. Update the BOS template used for booting CNs to use the new CN images.

       Refer to section "Boot Orchestration Service (BOS)" in the CSM documentation for instructions on how to update a BOS template.

19. **Optional:** Create a Boot Orchestration Service (BOS) session template for CNs. Skip this step if the previous step was not performed.

    1. Copy the current compute node BOS session template into a new file. Replace CN_TEMPLATE with the existing compute node sessiontemplate name. A list of sessiontemplate names can be obtained with the **cray bos sessiontemplate list –format json** command.

    ```
    ncn-m001# cray bos sessiontemplate describe CN_TEMPLATE --format=json > bos-template-cn-lustre.json
    ```

    2. Edit the new `bos-template-cn-lustre.json` file, changing the etag, path, and wlm values as needed.

    The existing session template will be overwritten if the name value is not updated.

    ```
    ncn-m001# vi bos-template-cn-lustre.json
    ```

    3. Create the session template with BOS.

    ```
    ncn-m001# cray bos session create --file bos-template-cn-lustre.json \
    --name cos-compute-2.3.xx-lustre
    ```

20. Reboot the compute nodes using BOS.

    Replace CN_TEMPLATE with either the BOS session template updated in the previous step, or the existing compute node template. The new Lustre file system will be available when the compute nodes are rebooted.

    ```
    ncn-m001# cray bos session create --template-uuid CN_TEMPLATE \
    --operation reboot
    ```

21. Repeat this entire process for the UANs to configure them to mount the Lustre file system. Make the same configuration changes in the UAN git repository that were made in the COS repository.

    Use the UAN repository, which is at https://api-gw-service-nmn.local/vcs/cray/uan-config-management.git. Also, Ansible variables should be set in the `group_vars/Application_UAN` directory instead of a `group_vars/Compute` directory. Finally, add the modified `configure_fs` line and the new `lustre_config` and `lustre_tuning` lines to the `–hosts: Application_UAN` section of the `site.yml` file.

22. **Optional:** Confirm that peer discovery is disabled on the compute and user access nodes. Skip this step if LNet peer discovery was not disabled earlier.

    1. SSH to a rebooted compute node.

    ```
    ncn-m001# ssh nid000001-nmn
    ```

    2. Verify that peer discovery is disabled on that node.

    ```
    nid000001# lnetctl global show
    global:
     numa_range: 0
     max_intf: 200
     **discovery: 0**
     drop_asym_route: 0
     retry_count: 2
     transaction_timeout: 120
     health_sensitivity: 100
     recovery_interval: 1
     router_sensitivity: 100
    ```

    The output of this command will include discovery:0 when LNet peer discovery is disabled.

7.3 Set Up LNet Routers

- **OBJECTIVE**

  Configure one or more Application nodes as LNet Routers.

- **ROLE**

  System administrator

- **Requirements**

  - A system domain expert (typically an admin) with the following background knowledge:
    * Hardware in the system, including how to configure the Lustre cluster for communication with routers. **Note:** Cluster configuration in particular is outside the scope of this document. Please see https://wiki.lustre.org/LNet_Router_Config_Guide and other Lustre documentation for further details.
    * GIT branching model being used for maintaining customized configuration for the various node classes represented in both the COS and UAN repositories. See Prepare to Configure DVS with VCS for the COS repository (although configuring LNet routers does not require DVS changes, the preliminary configuration steps are the same for both work flows), and equivalent UAN documentation for the UAN repository.
    * How CFS sessions are used to tie configuration specifications made in VCS to node classes. See Update CFS with Local Changes for background on the COS repository, and equivalent CSM documentation for the UAN repository.
    * How to create boot images with IMS. See CSM documentation on image management for details.
    * BOS sessiontemplates and their intended usage on the system. See CSM documentation on boot orchestration for details.
  - Network interface prerequisites:
    * Network interfaces on the LNet router nodes are configured. Refer to the site network documentation and the "HPE Cray User Access Node (UAN) Software Administration Guide", "Configure Interfaces on UANs" for details.

- **Caveats**
  - This procedure is provided as guidance only. Hewlett Packard Enterprise has not thoroughly tested these steps and does not guarantee that they will work for all customers.

Perform this procedure to establish one or more Application nodes as LNet Routers.

1. Register your LNet routers in HSM:

   You should see the xnames of your LNet routers in the following query:

   ```
   ncn-m# cray hsm state components list --role Application --subrole LNETRouter --format json
   ```

   If you do not, run the following command for the <XNAME> of each missing LNet router:

   ```
   ncn-m# cray hsm state components role update --role Application --subrole LNETRouter <XNAME>
   ```

2. Clone the UAN configuration repository, and enter it.

   In the `git clone` command below, the username will be `crayvcs`, and the password will be the result of:

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

   ncn-m# git clone https://api-gw-service-nmn.local/vcs/cray/uan-config-management.git
   ncn-m# cd uan-config-management && git checkout integration
   ```

3. Configure LNet routing

   Create a file which describes the source and destination networks for LNet routers and enables routing.

   ```
   ncn-m# mkdir -p group_vars/Application_LNETRouter
   ncn-m# vi group_vars/Application_LNETRouter/lnet.yml
   ncn-m# cat group_vars/Application_LNETRouter/lnet.yml
   lnet_conf: |
     ip2nets:
       - net-spec: <LC_NETWORK_NAME>
         interfaces:
           0: <LC_INTERFACE_0>
           1: <LC_INTERFACE_1>
           ...
       - net-spec: <HSN_NETWORK_NAME>
         interfaces:
           0: <HSN_INTERFACE_0>
           1: <HSN_INTERFACE_1>
           ...
     routing:
       - enable: 1
   lnet_modprobe_conf: |
   ```

```
options lnet lnet_transaction_timeout=120
options ko2iblnd map_on_demand=1
```

where by LC we mean the network hosting the Lustre Cluster and by HSN we mean the network generally available to the rest of the HPC system.

The content in the lnet_modprobe_conf variable may be different for your system, depending on the characteristics of your <LC_NETWORK> and <HSN_NETWORK>. We mainly include this section to demonstrate that this is how modprobe options should be specified for your routers.

4. Commit your new content, and push the commit to the remote.

```
ncn-m# git commit -am 'Added COMMIT_COMMENT'
ncn-m# git push origin integration
```

5. Obtain the most recent git commit hash for this branch, and save it for later use.

```
ncn-m# git rev-parse --verify HEAD
be2bd370ed246182ec765ca999d2fac4d355443a
ncn-m# export HEAD_COMMIT_HASH=be2bd370ed246182ec765ca999d2fac4d355443a
```

6. To apply your source code changes, follow instructions in the "HPE Cray User Access Node (UAN) Software Administration Guide", "Create UAN Boot Images" to:

    1. Create or update a CFS configuration explicitly for the LNet Routers (covered in section: "CONFIGURE UAN IMAGES"). Use the value of HEAD_COMMIT_HASH to include your new code in the section of this CFS configuration where cloneUrl refers to the uan-config-management repository.
    2. Create boot images for these LNet routers using the CFS configuration just created (also covered in section: "CONFIGURE UAN IMAGES").
    3. Create a BOS sessiontemplate explicitly for the LNet routers (covered in section: "PREPARE UAN BOOT SESSION TEMPLATES").
    4. Reboot these LNet routers using the BOS sessiontemplate just created. (also covered in section: "PREPARE UAN BOOT SESSION TEMPLATES").

7. Validate that your LNet routers have booted with the expected configuration by running the following commands for each LNet router:

    1. Verify that local interfaces have been configured correctly:

```
ncn-m# ssh <LNET_ROUTER> lnetctl net show
net:
    - net type: lo
      local NI(s):
        - nid: 0@lo
          status: up
    - net type: <LC_NETWORK_NAME>
      local NI(s):
        - nid: <LC_ADDR_0>
          status: up
          interfaces:
              0: <LC_INTERFACE_0>
...
    - net type: <HSN_NETWORK_NAME>
      local NI(s):
        - nid: <HSN_ADDR_0>
          status: up
          interfaces:
              0: <HSN_INTERFACE_0>
...
```

    2. Verify that routing is enabled:

```
ncn-m# ssh <LNET_ROUTER> lnetctl routing show
routing:
    - cpt[0]:
```

```
            tiny:
                npages: 0
                nbuffers: 512
                credits: 512
                mincredits: 512
            small:
                npages: 1
                nbuffers: 4096
                credits: 4096
                mincredits: 4096
            large:
                npages: 256
                nbuffers: 256
                credits: 256
                mincredits: 256
        - enable: 1
    buffers:
        tiny: 512
        small: 4096
        large: 256
```

8. Capture LNet addresses for client configuration.

   Capture the <HSN_NETWORK> LNet addresses of each LNet router for use in client configuration by running the following command on each LNet router, and preserving the results (by e.g., appending to a file):

   ```
   ncn-m# ssh <LNET_ROUTER> lnetctl net show | grep nid: \
   > | grep <HSN_NETWORK_NAME> | tr -s ' ' | cut -d\  -f 4
   ```

9. Configure all desired client nodes in the system to route to your new LNet routers by updating their `lnet_conf` variable in VCS, then deploy the updates.

   All node types will need a client stanza in VCS equivalent to the following example based on the Compute node class:

   ```
   ncn-m# cat group_vars/Compute/lnet.yml
   lnet_conf: |
     ip2nets:
       - net-spec: <NETWORK_NAME>
         interfaces:
           0: <INTERFACE_0>
           1: <INTERFACE_1>
           ...
     route:
       - net: <NETWORK_NAME>
         gateway: <LNET_ADDR_0>
       - net: <NETWORK_NAME>
         gateway: <LNET_ADDR_1>
         ...
   ```

   where <NETWORK_NAME> here should be the same as the <HSN_NETWORK_NAME> of your router stanza(s), and a given <LNET_ADDR> should be one of the LNet addresses you recorded during the previous step: "Capture LNet addresses for client configuration".

   There are slight differences between where this client configuration content is placed and how it is deployed, depending on the client node class. Differences are highlighted below.

   Note: There may already be configuration content in one or more of the lnet.yml files discussed below. Be sure to add content rather than replace it if that is the case.

   Note: Where the repository for client content is cos-config-management rather than uan-config-management, the `git clone` url will be: `https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git`. Otherwise, the steps for generally managing the repositories (e.g. cloning; committing; pushing) are identical.

1. NCN-W clients:
    1. Modify the `lnet_conf` variable in the file `group_vars/Management_Worker/lnet.yml` controlled by the appropriate branch of the `cos-config-managment` repository to define a route utilizing your new LNet routers, based on the example client stanza above.
    2. Make the new content available on the remote and capture the new git hash (as previously demonstrated).
    3. Update the appropriate CFS configuration for NCN workers to indicate your new git hash in the section where `cloneUrl` refers to the `cos-config-management` repository.
    4. Perform Reload DVS on an NCN to reload LNet.
2. Other client node types (Compute; UAN; Gateway):

    To set up routing on the remaining client node types, you will need to add configuration content based on the example client stanza above, but then follow steps similar to those used when establishing LNet routers above to actualize your configuration changes on the nodes, including:

    - pushing the changes to the remote
    - modifying the appropriate CFS session to indicate the new code
    - creating new images based on the modified CFS session
    - modifying the appropriate BOS sessiontemplate to use the new images
    - and rebooting the nodes using the modified BOS sessiontemplate.

    The location of the configuration content in VCS varies by node type. Differences are highlighted below.

    1. Compute:

        Modify the `lnet_conf` variable in the file `group_vars/Compute/lnet.yml` controlled by the appropriate branch of the `cos-config-managment` repository to define a route utilizing your new LNet routers, based on the example client stanza above.

    2. UAN:

        Modify the `lnet_conf` variable in the file `group_vars/Application_UAN/lnet.yml` controlled by the appropriate branch of the `uan-config-management` repository to define a route utilizing your new LNet routers, based on the example client stanza above.

    3. Gateways

        Modify the `lnet_conf` variable in the file `group_vars/Application_Gateway/lnet.yml` controlled by the appropriate branch of the `uan-config-management` repository to define a route utilizing your new LNet routers, based on the example client stanza above.

8 Manage GPUs

8.1 Enable GPU Support

This section describes the process for installing and making GPU support available via a CPS bind image on the system using the most recent COS compute image without GPU support. This image is mounted over the network to provide GPU support on top of the typical COS compute image.

- **OBJECTIVE**

    As of cos-2.3 this section serves as a guide for advanced configurations and setup. It will help you make customizations and then enable GPU support on the system via a Content Projection Service (CPS) bind image.

    As of cos-2.3 GPUs are supported as a standard configuration of COS if a CPS bind image of the configured name exists and GPU hardware is detected.

    Also, if for some reason you are having trouble using the gpu-nexus-tool or it does not fit your needs this section also shows you up to set up Nexus repos manually. The Nexus repos need to be available for CPS bind image creation to install the appropriate GPU vendor software.

- **ROLE**

  System administrator, system installer

- **Requirements**

  - The Cray command line interface (CLI) tool is initialized and configured on the system. See “Configure the Cray Command Line Interface (CLI)” in the
  - In order to configure image customization you will need familiarity with Git version control. You will need to update, compare, and possibly merge branches for configuration changes.
  - Familiarity with Nexus repository management practices.
  - A system domain expert (typically an admin) with the following knowledge:
    * Hardware in the system.
    * Images available on the system
      · COS images, in particular the one you want to use to boot GPU nodes.
      · GPU CPS bind images. In particular when they were created and which COS image they were based on.
    * BOS sessiontemplates and their intended usage on the system.
      · Has a new sessiontemplate been created or has an existed one been updated recently with a new COS image?

8.2 GPU Support Setup: Download Software and Create Nexus Repositories

Before you begin be aware there is now a gpu-nexus-tool available on the worker nodes that will automate this effort. For more information run `gpu-nexus-tool -h` on a worker node. There is also an overview of the `gpu-nexus-tool` at GPU Nexus Tool.

NOTE: As of CSM 1.2 some curl commands will need to be authenticated. This is normally handled by the `gpu-nexus-tool`. In order to use the curl commands in this guide is it recommended that you read the “Manage Repositories with Nexus” section in CSM documentation. The method described below is based on the referenced docs.

```
ncn-m001# export USERNAME=$(kubectl -n nexus get secret nexus-admin-credential \
--template {{.data.username}} | base64 -d)
ncn-m001# export PASSWORD=$(kubectl -n nexus get secret nexus-admin-credential \
--template {{.data.password}} | base64 -d)
```

1. This is a setup section that will affect all subsequent steps. Do section “a” for Nvidia support. Do section “b” for AMD support. If you need to support both you will need to follow the entire guide twice: once for Nvidia and once for AMD. Before proceeding you will need to set up a staging area to contain and isolate your work in progress.

   For the staging area you must use an absolute path (one that begins with /). You must use a different staging area each time this guide is followed. `ncn-m001` is the most common server to do the staging work on.

   ```
   ncn-m001# export STAGING_AREA=<dir>
   ```

   Make sure your staging area exists and change directory to it.

   ```
   ncn-m001# mkdir -p $STAGING_AREA;
   cd $STAGING_AREA;
   ```

   Now that you have a staging area proceed with setup by doing section “a” for Nvidia support or doing section “b” for AMD support.

   a. Nvidia setup.

      Download the HPC SDK toolkit and driver to an appropriate staging area. The directory used to download the content will be the STAGING_AREA so make sure to set that now.

      NOTE: For more information the download Nvidia web sites are noted below.

      - Download the drivers: https://www.nvidia.com/Download/driverResults.aspx/186238/en-us
      - Download the toolkit: https://developer.nvidia.com/nvidia-hpc-sdk-223-downloads

      1. Run the Download

         NOTE: The version specified below is the tested version for this release. We do not recommend using newer releases.

         ```
         SDK=https://developer.download.nvidia.com/hpc-sdk/sles/x86_64/;
         DR=https://us.download.nvidia.com/tesla/510.47.03/;
         DR=${DR}nvidia-driver-local-repo-sles15-510.47.03-1.0-1.x86_64.rpm;
         wget -P nvidia_sdk $(echo ${SDK}nvhpc-22-3-22.3-1.suse.x86_64.rpm \
         ```

```
${SDK}nvhpc-2022-22.3-1.suse.x86_64.rpm \
${SDK}nvhpc-22-3-cuda-multi-22.3-1.suse.x86_64.rpm)
wget -P nvidia_driver $DR
```

2. Unpack the container RPM locally. All Nvidia driver packages will then be available at the `var/nvidia-driver-local-repo-sles` path where the RPM was unpacked.

```
ncn-m001# cd nvidia_driver
rpm2cpio nvidia-driver-local-repo-sles15-510.47.03-1.0-1.x86_64.rpm \
| cpio -idmv
cd ../
```

3. Setup environment variables for subsequent steps. We do not have a driver repository for Nvidia and all content goes to one repository. The recommended name is "hpc-sdk-22.3". Using this value will reduce the amount of configuration changes necessary but any value is allowed.

```
ncn-m001# export NVIDIA_GPU_REPO=hpc-sdk-22.3;
export NVIDIA_GPU_DRIVER_REPO=nvidia-driver-510.47.03;
```

4. Create a Nexus Zypper repository on the Nexus system that will contain the content necessary for Nvidia SDK support.

   NOTE: The instructions here use the command line and Nexus API. Use of the Nexus web interface is not covered here, but it is an alternative option.

```
ncn-m001# curl -u $USERNAME:$PASSWORD \
https://packages.local/service/rest/beta/repositories/yum/hosted \
--header "Content-Type: application/json" --request POST --data-binary \
@- << EOF
{
  "name": "$NVIDIA_GPU_REPO",
  "online": true,
  "storage": {
    "blobStoreName": "default",
    "strictContentTypeValidation": true,
    "writePolicy": "ALLOW"
  },
  "cleanup": null,
  "yum": {
    "repodataDepth": 0,
    "deployPolicy": "STRICT"
  },
  "format": "yum",
  "type": "hosted"
}
EOF
```

5. Verify the new repository is available. The following command will show all Nexus repositories on the system and grep for the name used previously.

```
ncn-m001# REPOURL=https://packages.local/service/rest/v1/repositories
curl -u $USERNAME:$PASSWORD ${REPOURL} -s --header "Content-type: application/json" \
--http1.1 | grep $NVIDIA_GPU_REPO
```

6. Upload the Nvidia HPC SDK content and the Nvidia drivers content to the Nexus repository.

```
ncn-m001# cd $STAGING_AREA
ncn-m001# for i in $(find ./nvidia_sdk -name \*.rpm); do
curl -u $USERNAME:$PASSWORD -v --upload-file ./$i --max-time 600 \
https://packages.local/repository/$NVIDIA_GPU_REPO/;
done
```

7. Build the repository index.

```
ncn-m001# NEXUSRI=https://packages.local/service/rest/beta
NEXUSRI=${NEXUSRI}/repositories/$NVIDIA_GPU_REPO/rebuild-index
curl -u $USERNAME:$PASSWORD --request POST $NEXUSRI
```

8. Verify the new repository is available on the target system via standard zypper repository commands.

   a. Add the repositories with the zypper command.

   ```
   ncn-m001# NXSREPO=https://packages.local/repository/$NVIDIA_GPU_REPO
   zypper ar --no-gpgcheck $NXSREPO $NVIDIA_GPU_REPO
   ```

   b. List the defined repositories.

   ```
   ncn-m001# zypper lr -u | grep $NVIDIA_GPU_REPO
   ```

   c. Refresh the repository.

   ```
   ncn-m001# zypper ref
   ```

   d. Double check that all content is present. This must be ran from the staging directory where content was initially downloaded.

   ```
   ncn-m001# cd $STAGING_AREA
   ```

   NOTE: The diff below should be empty.

   ```
   ncn-m001# diff <(rpm -qp --nosignature --queryformat "%{NAME}\n" $(find ./nvidia_sdk |
                   grep 'rpm$') | sort) \
   <(zypper pa $NVIDIA_GPU_REPO | awk -F '|' '{ gsub(/ /, ""); print $3}' |
     awk 'NF' | grep -v "Name" | sort)
   ```

   e. Remove the repository from the NCN list so it does not accidentally interfere with any installs.

   ```
   ncn-m001# zypper rr $NVIDIA_GPU_REPO
   ```

9. Create a Nexus Zypper repository on the Nexus system that will contain the content necessary for Nvidia SDK support.

   NOTE: The instructions here use the command line and Nexus API. Use of the Nexus web interface is not covered here, but it is an alternative option.

   ```
   ncn-m001# curl -u $USERNAME:$PASSWORD \
   https://packages.local/service/rest/beta/repositories/yum/hosted \
   --header "Content-Type: application/json" --request POST --data-binary \
   @- << EOF
   {
     "name": "$NVIDIA_GPU_DRIVER_REPO",
     "online": true,
     "storage": {
       "blobStoreName": "default",
       "strictContentTypeValidation": true,
       "writePolicy": "ALLOW"
     },
     "cleanup": null,
     "yum": {
       "repodataDepth": 0,
       "deployPolicy": "STRICT"
     },
     "format": "yum",
     "type": "hosted"
   }
   EOF
   ```

10. Verify the new repository is available. The following command will show all Nexus repositories on the system and grep for the name used previously.

```
ncn-m001# REPOURL=https://packages.local/service/rest/v1/repositories
curl -u $USERNAME:$PASSWORD ${REPOURL} -s --header "Content-type: application/json" \
--http1.1 | grep $NVIDIA_GPU_DRIVER_REPO
```

11. Upload the Nvidia HPC SDK content and the Nvidia drivers content to the Nexus repository.

```
ncn-m001# cd $STAGING_AREA
ncn-m001# for i in $(find ./nvidia_driver -name \*.rpm); do
curl -u $USERNAME:$PASSWORD -v --upload-file ./$i --max-time 600 \
https://packages.local/repository/$NVIDIA_GPU_DRIVER_REPO/;
done
```

12. Build the repository index.

```
ncn-m001# NEXUSRI=https://packages.local/service/rest/beta
NEXUSRI=${NEXUSRI}/repositories/$NVIDIA_GPU_DRIVER_REPO/rebuild-index
curl -u $USERNAME:$PASSWORD --request POST $NEXUSRI
```

13. Verify the new repository is available on the target system via standard zypper repository commands.

    a. Add the repositories with the zypper command.

    ```
    ncn-m001# NXSREPO=https://packages.local/repository/$NVIDIA_GPU_DRIVER_REPO
    zypper ar --no-gpgcheck $NXSREPO $NVIDIA_GPU_DRIVER_REPO
    ```

    b. List the defined repositories.

    ```
    ncn-m001# zypper lr -u | grep $NVIDIA_GPU_DRIVER_REPO
    ```

    c. Refresh the repository.

    ```
    ncn-m001# zypper ref
    ```

    d. Double check that all content is present. This must be ran from the staging directory where content was initially downloaded.

    ```
    ncn-m001# cd $STAGING_AREA
    ```

    NOTE: The diff below should be empty.

    ```
    ncn-m001# diff <(rpm -qp --nosignature --queryformat "%{NAME}\n" $(find ./nvidia_driver |
                    grep 'rpm$') | sort) \
    <(zypper pa $NVIDIA_GPU_DRIVER_REPO | awk -F '|' '{ gsub(/ /, ""); print $3}' |
      awk 'NF' | grep -v "Name" | sort)
    ```

    e. Remove the repository from the NCN list so it does not accidentally interfere with any installs.

    ```
    ncn-m001# zypper rr $NVIDIA_GPU_DRIVER_REPO
    ```

b. AMD ROCm setup

1. Run the Download

   NOTE: The version specified below is the tested version for this release. We do not recommend using newer releases.

   ```
   ncn-m001# export RADEON_URL=https://repo.radeon.com
   ncn-m001# wget -r -l1 --no-parent -A.rpm $RADEON_URL/rocm/zyp/5.0.2/
   ncn-m001# wget -r -l1 --no-parent -A.rpm $RADEON_URL/amdgpu/21.50.2/sle/15/main/x86_64/
   ```

2. Setup environment variables for subsequent steps. Using the values below will reduce the amount of configuration changes necessary but any value is allowed if the corresponding configuration is also updated. Keep this step in mind if you need to stop and restart the procedure of setting up GPUs from a subsequent step.

   ```
   ncn-m001# export AMD_GPU_REPO=rocm-5.0.2;
   export AMD_GPU_DRIVER_REPO=amdgpu-21.50.2;
   export GPU_MAGIC_ENABLE=amd;
   ```

3. Create a Nexus Zypper repository on the Nexus system that will contain the content necessary for GPU support. The driver repository is covered starting with step 8.

```
ncn-m001# curl -u $USERNAME:$PASSWORD \
https://packages.local/service/rest/beta/repositories/yum/hosted \
--header "Content-Type: application/json" --request POST --data-binary \
@- << EOF
{
  "name": "$AMD_GPU_REPO",
  "online": true,
  "storage": {
    "blobStoreName": "default",
    "strictContentTypeValidation": true,
    "writePolicy": "ALLOW"
  },
  "cleanup": null,
  "yum": {
    "repodataDepth": 0,
    "deployPolicy": "STRICT"
  },
  "format": "yum",
  "type": "hosted"
}
EOF
```

4. Verify the new repository is available. The following command will show all Nexus repositories on the system and grep for the name used previously.

   ```
   ncn-m001# REPOURL=https://packages.local/service/rest/v1/repositories
   curl -u $USERNAME:$PASSWORD ${REPOURL} -s --header "Content-type: application/json" \
   --http1.1 | grep $AMD_GPU_REPO
   ```

5. Upload the AMD ROCm content to the Nexus repository.

   ```
   ncn-m001# cd $STAGING_AREA
   ncn-m001# for i in $(find repo.radeon.com/rocm/zyp/5.0.2 -name \*.rpm); do
   curl -u $USERNAME:$PASSWORD -v --upload-file ./$i --max-time 600 \
   https://packages.local/repository/$AMD_GPU_REPO/;
   done
   ```

6. Build the repository index.

   ```
   ncn-m001# NEXUSRI=https://packages.local/service/rest/beta
   NEXUSRI=${NEXUSRI}/repositories/$AMD_GPU_REPO/rebuild-index
   curl -u $USERNAME:$PASSWORD --request POST $NEXUSRI
   ```

7. Verify the new repository is available on the target system via standard zypper repository commands.

   a. Add the repositories with the zypper command.

      ```
      ncn-m001# NXSREPO=https://packages.local/repository/$AMD_GPU_REPO
      zypper ar --no-gpgcheck $NXSREPO $AMD_GPU_REPO
      ```

   b. List the defined repositories.

      ```
      ncn-m001# zypper lr -u | grep $AMD_GPU_REPO
      ```

   c. Refresh the repository.

      ```
      ncn-m001# zypper ref
      ```

   d. Double check that all content is present. This must be ran from the staging directory where content was initially downloaded.

      ```
      ncn-m001# cd $STAGING_AREA
      ```

      NOTE: The diff below should be empty.

```
ncn-m001# diff <(rpm -qp --nosignature --queryformat "%{NAME}\n" \
                $(find repo.radeon.com/rocm/zyp/5.0.2 | grep 'rpm$') | sort) \
<(zypper pa $AMD_GPU_REPO | awk -F '|' '{ gsub(/ /, ""); print $3}' |
  awk 'NF' | grep -v "Name" | sort)
```

e. Remove the repository from the NCN list so it does not accidentally interfere with any installs.

```
ncn-m001# zypper rr $AMD_GPU_REPO
```

8. Create a Nexus Zypper repository on the Nexus system that will contain the content necessary for GPU support. The driver repository is covered starting with step 8 and basically repeats steps 3 through 7.

   NOTE: The instructions here use the command line and Nexus API. Use of the Nexus web interface is not covered here, but it is an alternative option.

```
ncn-m001# curl -u $USERNAME:$PASSWORD \
https://packages.local/service/rest/beta/repositories/yum/hosted \
--header "Content-Type: application/json" --request POST --data-binary \
@- << EOF
{
  "name": "$AMD_GPU_DRIVER_REPO",
  "online": true,
  "storage": {
    "blobStoreName": "default",
    "strictContentTypeValidation": true,
    "writePolicy": "ALLOW"
  },
  "cleanup": null,
  "yum": {
    "repodataDepth": 0,
    "deployPolicy": "STRICT"
  },
  "format": "yum",
  "type": "hosted"
}
EOF
```

9. Verify the new repository is available. The following command will show all Nexus repositories on the system and grep for the name used previously.

```
ncn-m001# REPOURL=https://packages.local/service/rest/v1/repositories
curl -u $USERNAME:$PASSWORD ${REPOURL} -s --header  "Content-type: application/json" \
--http1.1 | grep $AMD_GPU_DRIVER_REPO
```

10. Upload the AMD driver content to the Nexus repository.

```
ncn-m001# cd $STAGING_AREA
ncn-m001# for i in $(find repo.radeon.com/amdgpu/21.50.2/sle/15/main/x86_64 -name \*.rpm); do
curl -u $USERNAME:$PASSWORD -v --upload-file ./$i --max-time 600 \
https://packages.local/repository/$AMD_GPU_DRIVER_REPO/;
done
```

11. Build the repository index.

```
ncn-m001# NEXUSRI=https://packages.local/service/rest/beta
NEXUSRI=${NEXUSRI}/repositories/$AMD_GPU_DRIVER_REPO/rebuild-index
curl -u $USERNAME:$PASSWORD --request POST $NEXUSRI
```

12. Verify the new repository is available on the target system via standard zypper repository commands.

    a. Add the repositories with the zypper command.

```
ncn-m001# NXSREPO=https://packages.local/repository/$AMD_GPU_DRIVER_REPO
zypper ar --no-gpgcheck $NXSREPO $AMD_GPU_DRIVER_REPO
```

b. List the defined repositories.

```
ncn-m001# zypper lr -u | grep $AMD_GPU_DRIVER_REPO
```

c. Refresh the repository.

```
ncn-m001# zypper ref
```

d. Double check that all content is present. This must be ran from the staging directory where content was initially downloaded.

```
ncn-m001# cd $STAGING_AREA
```

NOTE: The diff below should be empty.

```
ncn-m001# diff <(rpm -qp --nosignature --queryformat "%{NAME}\n" \
                 $(find repo.radeon.com/amdgpu/21.50.2/sle/15/main/x86_64 | grep 'rpm$') |
                   sort) \
<(zypper pa $AMD_GPU_DRIVER_REPO | awk -F '|' '{ gsub(/ /, ""); print $3}' |
  awk 'NF' | grep -v "Name" | sort)
```

e. Remove the repository from the NCN list so it does not accidentally interfere with any installs.

```
ncn-m001# zypper rr $AMD_GPU_DRIVER_REPO
```

8.3 GPU Support: Create a New CPS Bind Image

2. Use sat bootprep to create the CPS Bine Image.

```
sat bootprep run gpu_bootprep.yaml
```

Example gpu_bootprep.yaml file:

```
configurations:
  - name: gpu-cps-bind-image-create
    layers:
    - name: cos-config-management-for-gpu-2.3.70
      playbook: gpu_customize_playbook.yml
      product:
        name: cos
        version: 2.3.70
        branch: integration
images:
  - name: gpu-image
    description: >
      The Nvidia CPS image.
    ims:
      name: cray-shasta-compute-sles15sp3.x86_64-2.3.31
      is_recipe: false
    configuration: gpu-cps-bind-image-create
    configuration_group_names: [Compute]
```

Notes for defining your bootprep yaml:

1. To check which version of cos you are using use `sat showrev --products`. This is useful for the “product” section.
2. Typically we use integration but use whichever branch has the configuration you want to use.
3. You must use the name “gpu-image” unless you have changed the COS configuration.
4. For the “ims” section you can use the name or the id of the COS image to be customized.
5. The configuration name in the “configurations” section needs to match the configuration in the “images” section.

3. Optional: Verify the Ansible configuration plays ran successfully by checking the Ansible logs for the CFS worker pod.

a. Set the CFS_SESSION

The CFS session is displayed in the sat output from the previous step on a line like:

```
INFO: CFS session: sat-371bdd62-e0ee-4380-b214-ba69a3f1c09d      Image: gpu-image:
```

```
ncn-m001# export CFS_SESSION=sat-371bdd62-e0ee-4380-b214-ba69a3f1c09d
```

b. Retrieve the CFS job for use in later commands.

```
ncn-m001# export CFS_JOB_ID=$(cray cfs sessions describe $CFS_SESSION --format=json |
jq -r '.status.session.job')
```

c. Get the CFS worker pod with the CFS job ID returned in the previous step.

```
ncn-m001# export CFS_POD_ID=$(kubectl get pods -n services |
grep $CFS_JOB_ID | awk '{print $1}')
```

d. Access the log for the CFS pod found in the previous step.

Follow the Ansible log as it runs the jobs to install and configure the image. Watch for errors in the ansible plays. At the end of the run, check the "PLAY RECAP" output for any failed Ansible plays.

```
ncn-m001# kubectl logs -n services $CFS_POD_ID -f -c ansible-0
```

4. Optional: Check the contents of the CPS image. This is valuable if you're looking for a specific change to be there now or diagnosing a problem and know what to check.

```
ncn-m001# cd $STAGING_AREA
ncn-m001# cray artifacts get boot-images $RESULTANT_IMAGE_ID/rootfs ./rootfs
ncn-m001# mkdir -p mnt/img
ncn-m001# mount -t squashfs -o loop ./rootfs mnt/img
ncn-m001# chroot mnt/img
ncn-m001# rpm -qa | grep nvidia
```

8.4 GPU Support: Reboot/Reconfigure Compute Nodes for your System

5. Use the COS BOS sessiontemplate to reboot or reconfigure the GPU nodes.

Rebooting or reconfiguring the nodes will make the GPU content available.

First you will need to determine the COS_TEMPLATE_NAME and then set the –operation parameter to `Reboot` or `Configure`.

```
ncn-m001# export COS_TEMPLATE_NAME=<cos-template-name>
```

```
ncn-m001# export OPERATION=Reboot
```

or

```
ncn-m001# export OPERATION=Configure
```

```
ncn-m001# export BOS_SESSION_JOB_ID=$(cray bos session create --template-uuid
$GPU_TEMPLATE_NAME --operation $OPERATION --format=json | jq -r '.job')
```

6. Track the status of the Boot Orchestration Agent (BOA) pods for the BOS session.

a. Get the name of the BOA Kubernetes pod ID.

```
ncn-m001# export BOA_POD_ID=$(kubectl get pods -n services | grep
$BOS_SESSION_JOB_ID | awk '{print $1}')
```

b. Watch the log for the BOA Kubernetes pod.

The BOA log will run until the nodes have reported either a completed boot and configure, or just a configuration depending on what BOS operation was performed.

```
ncn-m001# kubectl logs -n services $BOA_POD_ID -f -c boa
```

c. Verify the configuration was successful by monitoring the CFS worker pod.

To find the CFS pods that corresponds with the BOS session:

```
ncn-m001# cray cfs sessions list
```

NOTE: There is a bos_session value in `.tags.bos_session` that will match your $BOS_SESSION_JOB_ID if you trim the prefix "boa-". There may not be a matching CFS pod if the job didn't get that far along.

If successful, the end of the CFS Ansible log will report a number of successful ansible plays and no failures. If there are issues, review the log output for the failing play.

```
ncn-m001# kubectl logs -n services cfs-ff2174c0-16d2-4af2-8d0b-ba5e242e1e7e -c ansible-0
```

7. Check the status of the BOS session job.

   A finished pod will report "complete" as true, have a stop time listed, and report the field in_progress as false. Grizzly Peak compute node boot and CFS post-boot personalization is now complete.

   ```
   ncn-m001# cray bos session describe ${BOS_SESSION_JOB_ID:4} --format json | jq
   ```

8. Clean up the BOS session when it is finished.

   ```
   ncn-m001# cray bos session delete ${BOS_SESSION_JOB_ID:4}
   ```

9. As follow up there should be a system convention or notes so this new template is used to boot these nodes in the future. This may need to be reported to other administrators. Additionally it a good practice to do a health check on at least one node which is described in the following sections:

   - GPU Health Check: Nvidia
   - GPU Health Check: AMD ROCm

8.5 GPU Health Check: Nvidia

1. SSH to a Nvidia GPU compute node.

   ```
   ncn-m001# ssh <nvidia_gpu_xname>
   ```

2. Verify the Nvidia drivers are loaded.

   a. Check for the Nvidia drivers.

      ```
      node# lsmod | grep nv
      ```

      The following drivers should be present:

      - nvidia_modeset
      - nvidia_drm
      - nvidia_uvm
      - nvidia
      - nv_peer_mem

   b. Confirm the gdrdrv module is loaded.

      The gdrdrv module is the driver for the gdrcopy package.

      ```
      node# lsmod | grep gdr
      ```

3. Verify the Nvidia GPUs are healthy.

   Verify all four GPUs are found and running. Verify the "Persistence Mode" setting is reporting as "Enabled".

   ```
   node# nvidia-smi -q | grep "Persistence Mode"
   ```

4. Run the gdrcopy sanity tests.

   ```
   node# which sanity
   node# sanity -v
   node# which copybw
   node# copybw
   ```

5. Verify the HPC SDK module file is available. Under /opt/cray/modulefiles or /opt/cray/pe/lmod/modulefiles/core/ you should see cudatoolkit.

   NOTE: You need the PE layer configured for this to work. See GPU Support: Create a New CPS Bind Image.

```
node# module avail
node# module load cudatoolkit
```

8.6 GPU Health Check: AMD ROCm

1. SSH to an AMD compute node.

   ```
   ncn-m001# ssh <amd_gpu_xname>
   ```

2. Verify the ROCm install on a booted node.

   ```
   lsmod | grep amd
   ```

   In the above output following modules should be shown as loaded: amdgpu, amdkcl, amd-sched, and amdttm.

   Run the following command and verify the GPUs are found:

   ```
   /opt/rocm/bin/rocm-smi
   ```

8.7 GPU Configuration

The GPU customization can be configured. This is available at https://api-gw-service-nmn.local/vcs/cray/cos-config-management.git for compute nodes and https://api-gw-service-nmn.local/vcs/cray/uan-config-management.git for UAN which needs to be cloned. This guide does not go into details on managing the git configuration repository it just explains the configuration options. Ansible playbooks are very flexible and allow for more configuration than described here. This is just what we consider to be standard and supported configuration. It is recommended to change the configuration in the file `<cos-config-management>/ansible/group_vars/Compute/gpu_info.yml` for compute nodes or `<uan-config-management>/ansible/group_vars/Application/gpu_info.yml` for UAN.

1. gpus

   This is the list of vendors for GPU support the to check. By default this is all the vendors that are currently supported. This can make the Ansible output more noisy than it needs to be so reducing the list to only the vendors needed can be helpful. If the vendor repository exists in nexus when a CPS image is created for GPU support then it will be used. Additionally if GPU hardware is detected on the machine the vendor content from the CPS image will be projected if it exists.

   The supported values for gpus at this time are `amd` and `nvidia`.

2. site_gpu_image_Name

   This is set to `gpu-image` by default which is the name of the CPS image used to deploy the GPU software. You may want to change this if you want to test a new CPS image but do not want to replace the existing one.

3. gpu_forced_project

   This is a boolean value. If set to true it will mount all available vendor content on the node. This isn't currently support for Compute nodes and should always be set to false. It is supported for UAN nodes and set to true by default so the content is available even if that node doesn't have a physical GPU.

4. nvidia_mod_conf_options

   This sets additional Nvidia options in /etc/modprobe.d/50-nvidia-default.conf during image customization when creating the CPS GPU image. The default setting provides `NVreg_RestrictProfilingToAdminUsers=0` which allows all users to access perf counter.

5. nvidia_gpu_compute_mode

   Thie sets the gpu compute mode used by `nvidia-smi -c`. The default mode set by the Ansible playbook is 3.

   ```
   0 = DEFAULT
   1 = EXCLUSIVE THREAD (deprecated)
   2 = PROHIBITED
   3 = EXCLUSIVE PROCESS
   ```

Example Compute Node Config:

```
gpus:
    - amd
    - nvidia
```

```
# Options to set in the nvidia module.conf file for nvidia module
nvidia_mod_conf_options:
    - "NVreg_RestrictProfilingToAdminUsers=0"
nvidia_gpu_compute_mode: 2
```

Example UAN Config:

```
gpus:
    - amd
    - nvidia

# Always project SDK on application node regardless of the presence of the hw
gpu_forced_project: True

# Options to set in the nvidia module.conf file for nvidia module
nvidia_mod_conf_options:
    - "NVreg_RestrictProfilingToAdminUsers=0"
```

8.8 GPU Nexus Tool

The `gpu-nexus-tool` is primarily for uploading vendor GPU content to Nexus for supporting GPUs on the system and is available on worker nodes. The GPU content needs to be available in Nexus before a CPS image can be created to project the content. It can also check if the content is correct. This functionality eases the burden on the admin to provide the content and ensure it is correct by reducing the number of steps that need to be done and reducing the chance for mistakes.

Commands, subcommands, and options.

- Commmand: `repo`

  The main functionality of the `gpu-nexus-tool`. Provides the three subcommands `upload`, `check`, `anddelete'.

  - Subcommand: `upload`

    The upload command first obtains the vendor content and then uploads it to Nexus.

    * Option: `--download-dir`, `-d`

      If you do not want to download the RPMs to the default location in `/tmp/gpu_content_upload` you can specify a directory with this command.

    * Option: `--upload-dir`, `-u`

      If you are providing your own download directory this will attempt an upload without downloading anything from the vendor from the location specified.

      When providing content you need to provide the RPMs in the directories `nvidia_sdk_content`, `nvidia_driver_content`, `amd_sdk_content`, and `amd_driver_content`.

    * Option: `--upload-missing`, `-m`

      If content is not uploaded correctly on the first attempt this option will ensure only missing content is attempted to be uploaded.

    * Option: `--timeout`, `-t`

      The default timeout is 600 seconds. A longer default can be helpful if your machine is uploading artifacts slower then usual. A shorter timeout can be useful if experiencing frequent timeouts not due to the size of the file because it will fail faster.

    * Option: `--no-check-hash`

      It is not recommended to use this option because it bypasses a content check against a pre-determined hash before uploading to Nexus. Since HPE does not control the content it is possible the content is legitimately changed so this is provided to avoid issues in the field.

    * Option (Required): `--vendor`, `-v`

      Specify AMD or Nvidia for the tool to determine which Nexus repos to setup and which content to download.

    * Option: `--show`, `-S`

      Show output which is just the underlying commands that are being run.

    * Option: `--verbose`, `-V`

      Verbose output which is all underlying commands and their output.

  - Subcommand: `check`

    Provided as a convenience. Installation issues can be quickly diagnosed and avoided by checking the content in Nexus is correct. It is also useful to make sure there was no errors or timeouts encountered during the upload to Nexus.

    * Option (Required): `--vendor`, `-v`

      Specify AMD or Nvidia for the tool to determine which repos to check.

    * Option: `--show`, `-S`

      Show output which is just the underlying commands that are being run.

    * Option: `--verbose`, `-V`

      Verbose output which is all underlying commands and their output.

  - Subcommand: `delete`

    Provided as a convenience. This is not typically used during normal workflows. If there's an issue with content in a repo or the repo itself we have found it is occasionally easiest to delete and recreate the repo from scratch.

    * Option (Required): `--vendor`, `-v`

      Specify AMD or Nvidia for the tool to determine which repos to delete.

    * Option: `--show`, `-S`

      Show output which is just the underlying commands that are being run.

    * Option: `--verbose`, `-V`

      Verbose output which is all underlying commands and their output.

- Command: `clean`

  This will delete the default `/tmp/gpu_content_upload` location used by the upload command.

  - Option: `--verbose`, `-V`

    Provided by default but currently this command does not trigger any verbose output.

  - Option: `--show`, `-S`

    Provided by default but currently this command does not trigger any show output.

8.9 Boot COS on HPE Apollo 6500 Gen10 Plus XL675d with Nvidia A100 GPU Tray

Due to the large number of devices on a fully populated XL675d Modular Accelerator Tray with Nvidia A100 GPUs and the complexities of determining appropriate PCI device address ranges across the various bridge devices, these nodes can experience PCI device startup failures that can prevent completion of early compute node boot. This is caused by the Linux kernel attempting to reallocate PCI bridge resources after BIOS completion. That reallocation can prevent correct initialization of devices on the accelerator tray.

To enable the correct boot of nodes with the XL675d GPU Tray, the kernel PCI bridge reallocation can be disabled. Disable reallocation by adding the following kernel parameter to the `kernel_parameters` variable of the COS compute node BOS session template:

```
pci=realloc=off
```

For example:

```
"kernel_parameters": "ip=dhcp quiet spire_join_token=${SPIRE_JOIN_TOKEN} pci=realloc=off"
```

See the "Boot COS" section in the ***HPE Cray Operating System installation Guide: CSM on HPE Cray EX Systems*** for more information.

8.10 Update Nvidia GPU Software without Rebooting

Use CPS and BOS to update the Nvidia GPU software on compute nodes and UANs.

- 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 CSM documentation for more information.
- Nvidia GPU support has been enabled. Refer to Enable GPU Support.
- **ROLE**

  System administrator, system installer

- **OBJECTIVE**

  Manually update Nvidia GPU software to a new version without rebooting the GPUs. This procedure can be used for compute nodes and User Access Nodes (UANs).

1. Remove the currently deployed cudatoolkit and drivers.

   This step is required only if the existing version needs to be removed. Otherwise, this step can be skipped.

   Refer to "Configuration Management" section in the HPE CSM documentation to create a node personalization CFS session using the Git branch created for node personalization of the older version CPS bind image during the install process.

   Instead of running the default site.yml file, the session should run the gpu_undeploy_playbook.yml playbook by using the –ansible-playbook option. The node inventory to run the CFS session on should be all the Grizzly Peak nodes. There should be a boot session template to boot Grizzly Peak nodes. Use the static inventory mentioned in section "Run a Session with Static Inventory", listing all Grizzly Peak under the Compute section of the hosts file of the configuration manifest.

   Important: If deploying new Nvidia software on a UAN, use a boot session template for a UAN instead of the Grizzly Peak nodes.

   1. Create a CFS configuration that will be used to create a CFS job.

      Create the undeploy_gpu.json file to hold data about the CFS configuration. The new configuration should reference the appropriate clone URL for either the Cray Operating System (COS) or UAN Version Control Service (VCS) config branch, the commit ID of the default Grizzly Peak VCS branch, and the gpu_undeploy_playbook.yml playbook. Note your configuration may differ from `cuda_configuration`.

      ```
      ncn-w001# CONFIG_TO_COPY=cuda_configuration
      ncn-w001# CONFIG_TO_CREATE=undeploy_gpu
      ncn-w001# TMP=$(cray cfs configurations describe $CONFIG_TO_COPY --format json | \
      jq '.layers[0].playbook = "gpu_undeploy_playbook.yml"' | jq 'del(.lastUpdated, .name)')
      ncn-w001# echo $TMP > undeploy_gpu.json
      ncn-w001# cray cfs configurations update $CONFIG_TO_CREATE --file undeploy_gpu.json
      ```

   2. Use the new configuration to un-deploy an existing version of Nvidia software. As an example we are undeploying a single node but the method is left up to the admin to decide based on the specific use case. For example, if you do undeploys often of all the GPU computes you may want to set up a session template like is done in the GPU Support section or if you are troubleshooting a single node the example below should suffice.

      ```
      ncn-w001# export CFS_SESSION_NAME=undeploy-nvidia-software
      ncn-w001# export XNAME=<GPU_XNAME>
      ncn-w001# cray cfs sessions create --name $CFS_SESSION_NAME \
      --configuration-name $CONFIG_TO_CREATE --target-definition spec \
      --target-group Compute $XNAME
      ```

2. Create a new Content Projection Service (CPS) bind image with a new version of Nvidia GPU software.

   Follow the process for manually installing and making Nvidia Cuda support available via a CPS bind image. Refer to the Enable GPU Support procedure and complete the following tasks:

   - GPU Support Setup: Download Software and Create Nexus Repositories
   - GPU Support: Create a New CPS Bind Image
   - GPU Support: Update the Configuration Manifest with the New Bind Image
   - GPU Support: Reboot/Reconfigure Compute Nodes for your System

9 Manage Power and Performance

10 Power Management

Power management of Cray EX systems.

10.1 Overview

HPE Cray System Management (CSM) software manages and controls power out-of-band through Redfish APIs. Note that power management features are “asynchronous,” in that the client must determine whether the component status has changed after a power management API call returns.

In-band power management features are not supported in v1.4.

HPE supports Slurm as a workload manager which reports job energy usage and records it in the ITDB for system accounting purposes.

10.2 HPE Cray EX Systems

Cabinet-level power/energy data from compute blades, switch blades, and chassis rectifiers is collected by each Chassis Management Module (CMM) and provided on system management network. This power telemetry can be monitored by the SMF. Cabinet-level power data is collected and forwarded to the management nodes. The management nodes store the telemetry in the power management database.

10.3 PM Counters

The blade-level and node-level accumulated energy telemetry is point-in-time power data. Blade accumulated energy data is collected out-of-band and is made available via workload managers. Users have access to the data in-band at the node-level via a special sysfs files in /sys/cray/pm_counters on the node.

10.4 HPE Cray EX Standard Rack Systems

Rack systems support 2 intelligent power distribution units (iPDUs). The power/energy telemetry, temperature, and humidity measurements (supported by optional probes), are accessible through the iPDU HTTP interface.

Node-level accumulated energy data is point-in-time power and accumulated energy data collected via Redfish through the server BMC.

11 Set the Turbo Boost Limit

Set turbo boost limit.

Turbo boost limiting is supported on the Intel® and AMD® processors. Because processors have a high degree of variability in the amount of turbo boost each processor can supply, limiting the amount of turbo boost can reduce performance variability and reduce power consumption.

Turbo boost can be limited by setting the `turbo_boost_limit` kernel parameter to one of these values:

- 0 - Disable turbo boost
- 999 - (default) No limit is applied.

The following values are not supported in COS v1.4:

- 100 - Limits turbo boost to 100 MHz
- 200 - Limits turbo boost to 200 MHz
- 300 - Limits turbo boost to 300 MHz
- 400 - Limits turbo boost to 400 MHz

The limit applies only when a high number of cores are active. On an N-core processor, the limit is in effect when the active core count is N, N-1, N-2, or N-3. For example, on a 12-core processor, the limit is in effect when 12, 11, 10, or 9 cores are active.

11.1 Set or Change the Turbo Boost Limit Parameter

Modify the Boot Orchestration Service (BOS) template for the node(s). This example below disables turbo boost. The default setting is 999 (no limit).

```
{
"boot_sets": {
"boot_set61": {
"boot_ordinal": "0",
"ims_image_id": "efdfe6fc-af3f-40f0-9053-dd1ad6c359d3",
"kernel_parameters": "console=tty0 console=ttyS0,115200n8 root=crayfs
imagename=/SLES15 selinux=0 rd.shell rd.net.timeout.carrier=40 rd.retry=40 ip=dhcp
rd.neednet=1 crashkernel=256M turbo_boost_limit=0",
"network": "nmn",
"node_groups": [
"group1",
"group2"
],
"node_list": [
"x0c0s28b0n0",
"node2",
"node3"
],
"node_roles_groups": [
"compute"
],
"rootfs_provider": "",
"rootfs_provider_passthrough": ""
},
},
"cfs_branch": "my-test-branch",
"cfs_url": "https://api-gw-service-nmn.local/vcs/cray/config-management.git",
"enable_cfs": "true",
"name": "st6",
"partition": "p1"
}
```


12 Enable Low Noise Mode

Some application workloads show improved performance when compute node operating system tasks (sources of "OS noise") are migrated to one or more system CPUs that are excluded from application use. Low Noise Mode configures Linux features to achieve this configuration.

Two variations of Low Noise Mode (LNM) are available: full and lightweight. In full LNM, the Linux kernel is booted with parameters instructing it to reduce noise by moving system activities to CPU 0 (and potentially additional CPUs as well) and user space is configured to move any overhead processes to CPU 0. In lightweight mode, the kernel parameters are not used, but the user space configuration is still done.

12.1 Kernel Parameters

In full LNM mode, the Linux kernel must be booted with parameters to direct noise overhead to CPU 0:

```
nohz_full=1-255
rcu_nocbs=1-255
rcu_nocb_poll
```

The CPU range has to be specified for the number of CPUs on the node.

In addition, two parameters are specified on the kernel command line to guide the behavior of the user space configuration:

```
lnm={ full, lightweight }
[lnm.cpu=0]
```

The `lnm` parameter must be set to either `full` or `lightweight`. The `lnm.cpu` parameter is optional and is set to what CPU(s) to use to handle system overhead (see below for the full syntax).

12.2 The LNM Configuration File

At system boot, systemd runs lnmctl to configure the user space environment for low noise. Lnmctl reads default configuration from /etc/lnm-default.json and optional site configuration from /etc/lnm.json. The format of the two files is identical, however the site configuration file is not required to have all of the sections. The items in the site configuration file will override the ones in the defaults. This way individual items can be overridden without requiring that an entire section, or the whole file, be duplicated.

The following is an example of a LNM configuration file:

```
{
    "Tunables": {
        "sysctl": {
            "vm.stat_interval": 120
        },
        "files": {
            "/sys/bus/workqueue/devices/writeback/cpumask": 1,
            "/sys/kernel/mm/transparent_hugepage/enabled": "never"
        }
    },
    "Processes": [],
    "IRQs": {},
    "CPU": 0
}
```

12.2.1 CPU Section

The CPU section specifies which processors are used for system overhead CPUs. The system overhead CPUs are the CPUs that run all of the kernel services and user space processes that are not part of an application. These processors are not used to run user applications.

This parameter can be overridden by the `lnm.cpu=` kernel parameter.

The CPU field can list a single CPU, a range of CPUs, or `MAX` for lnmctl to find the highest numbered CPU on the node and use it.

```
{
    "CPU": 0
}
```

```
{
    "CPU": "16-32"
}
```

```
{
    "CPU": "MAX"
}
```

The same syntax is used for the `lnm.cpu` kernel parameter:

```
lnm.cpu=128
```

```
lnm.cpu=253-255
```

```
lnm.cpu=MAX
```

12.2.2 Tunables Section

The tunables section lists system tuning configuration, which is set automatically by LNM using `sysctl` or writing to files under `/sys`.

This section has two sub-sections: sysctl and files. The sysctl sub-section lists settings to be set with the sysctl utility, and their values. The files sub-section lists files to write to (typically under `/sys`) and the values to write.

These values can be overridden in the site configuration file.

This example sets the cpumask to `CPUMASK`, which will be calculated from the CPUs listed in the CPU section. It also removes setting `/sys/kernel/mm/transparent_hugepage/enabled` by setting it to an empty string.

```
{
    "Tunables": {
        "files": { '/sys/bus/workqueue/devices/writeback/cpumask': "CPUMASK",
            "/sys/kernel/mm/transparent_hugepage/enabled": ""
        }
    }
}
```

This example sets the cpumask to `0xffff`. The value is quoted so it looks like a string to json. Hexadecimal values are not valid json, so this must be in quotes, or converted to decimal.

```
{
    "Tunables": {
        "files": { '/sys/bus/workqueue/devices/writeback/cpumask': "0xffff",
        }
    }
}
```

12.2.3 Processes Section

This section contains a list of process names that should not be migrated to the system CPU(s). By default, all processes are migrated, with the exception of processes such as per-CPU threads that cannot be migrated.

This example lists two sets of processes to exclude from migration. The first is any process with `watchdog` in it's name. The second is any process with the exact name `spire_agent`.

```
{
    "Processes": [".*watchdog.*", "spire_agent" ]
}
```

12.2.4 IRQs Section

This section lists what hardware Interrupt Requests (IRQs) should be directed to CPUs handling system overhead.

IRQs can be listed two different ways:

- Migrate IRQs to the system overhead CPUs, listed in the CPU section, by providing a list of the IRQ numbers:

```
{
    "IRQs": [ 0, 1, 2 ]
}
```

- Migrate IRQs to specific CPUs using IRQ to CPU map. IRQs are listed by number, and there are two options for where IRQs can be directed: either CPU 0 (`cpu_0`) or the highest numbered CPU on the node (`cpu_last`).

This example directs IRQ 0 to CPU 0 and IRQ 1 to the highest numbered CPU.

```
{
    "IRQs": {
        "0": "cpu_0",
        "1": "cpu_last"
    }
}
```

12.2.5 Interaction With Workload Manager Software

This section provided information about enabling and customizing Low Noise Mode in COS. Workload manager software also needs to be configured to work properly with compute nodes booted with LNM enabled. See the "HPE Cray Programming Environment Installation Guide: HPCM on HPE Cray EX and HPE Cray Supercomputer Systems (S-8012)" for further details.

13 User Access to Compute Node Power Data

HPE Cray EX Liquid Cooled AMD EPYC compute node power management data is available to users.

HPE Cray EX Liquid Cooled compute blade power management counters (pm_counters) enable users access to energy usage over time for billing and job profiling.

The blade-level and node-level accumulated energy telemetry is point-in-time power data. Blade accumulated energy data is collected out-of-band and is made available via workload managers. Users have access to the data in-band at the node-level via a special sysfs files in /sys/cray/pm_counters on the node.

Time-stamped energy data from each node can be captured for a specific job before, during, and after the job to generate a power profile about the job. This energy usage data can be used in conjunction with current energy costs to assign a monetary value to the job.

The node CPU vendor provides specific in-band and out-of-band interfaces for controlling power management. In-band interfaces are accessed from the node OS through special `sysfs` files in /sys/cray/pm_counters on the node. Out-of-band interfaces are accessed from a node BMC or Redfish API.

The combined power consumption of the CPU and the accelerator can never exceed this limit. Thus, power to either the CPU or the accelerator must be capped so it does not exceed the total amount of power available.

13.1 pm_counters

Access to compute node power and energy data is provided by a set of files located in /sys/cray/pm_counters/ on the node. All pm_counters are accompanied by a timestamp.

- `power`: Point-in-time power (Watts). When accelerators are present, includes `accel_power`. See the limitation on data collection from accelerators in the `accel_power` bullet point below.
- `energy`: Accumulated energy, in joules. When accelerators are present, includes `accel_energy`. See the limitation on data collection from accelerators in the `accel_energy` bullet point below.
- `cpu_power`: Point-in-time power (Watts) used by the CPU domain.
- `cpu_energy`: The total energy (Joules) used by the CPU domain.
- `cpu_temp`: Temperature reading (Celsius) of the CPU domain.
- `memory_power`: Point-in-time power (Watts) used by the memory domain.
- `memory_energy`: The total energy (Joules) used by the memory domain.
- `accel_energy`: Accumulated accelerator energy (Joules). The data is non-zero only when an accelerator is present on the node.
- `accel_power`: Accelerator point-in-time power (Watts). The data is non-zero only when an accelerator is present on the node.
- `generation`: A counter that increments each time a power cap value is changed.
- `startup`: Startup counter
- `freshness`: Free-running counter that increments at a rate of approximately 10Hz.
- `version`: Version number for power management counter support.
- `power_cap`: Current power cap limit in Watts; 0 indicates no capping. When accelerators are present, includes `accel_power_cap`.
- `raw_scan_hz`: The power management scanning rate for all data in pm_counters.