Hewlett Packard
Enterprise

HPE Cray EX Analytics Applications Guide (1.2.22) (S-8045)

Part Number: S-8045
Published: July 2022

HPE Cray EX Analytics Applications Guide

Contents

1 Copyright and Version **3**

2 Introduction **4**
2.1 About the HPE Cray EX Analytics Applications Guide . . . 4
2.1.1 Scope and Audience . . . 4
2.1.2 Typographic Conventions . . . 4
2.1.3 Trademarks . . . 4
2.2 About HPE Cray EX Analytics and AI . . . 4
2.2.1 Supported Analytics and AI Frameworks . . . 4
2.2.2 Prerequisites for Running Analytics and AI jobs . . . 5
2.3 Documentation Conventions . . . 5
2.3.1 Markdown Format . . . 5
2.3.2 File Formats . . . 5
2.3.3 Typographic Conventions . . . 5
2.3.4 Command Prompt Conventions . . . 5
2.4 Analytics Release Notes . . . 7
2.4.1 New Features . . . 7
2.4.2 Known Issues . . . 7
2.4.3 Component Versions . . . 7

3 Install Analytics **7**
3.1 Install the Analytics Product Stream Update . . . 7
3.1.1 Prerequisites . . . 7
3.1.2 Procedure . . . 8
3.2 Upgrade the Analytics Product Stream . . . 9
3.2.1 Prerequisites . . . 9
3.2.2 Procedure . . . 9
3.3 Configure the Analytics Product Stream . . . 10
3.3.1 Prerequisites . . . 10
3.3.2 Pre-Procedure . . . 10
3.3.3 Perform NCN Personalization . . . 10
3.3.4 Use SAT to Perform Compute Node Operations . . . 11
3.3.5 Perform Compute Node Personalization . . . 11
3.3.6 Perform UAN Personalization . . . 12
3.3.7 Post-Procedure . . . 12

4 Analytics Admin Operations **13**
4.1 Install Singularity . . . 13
4.1.1 Prerequisites . . . 13
4.1.2 Procedure . . . 13
4.2 Create a Singularity container from an NVIDIA Tensorflow image . . . 20
4.2.1 Prerequisites . . . 20
4.2.2 Procedure . . . 20
4.3 Build and Test a Singularity Image with GPU and CPU Support . . . 23
4.3.1 Prerequisites . . . 23
4.3.2 Notes on The Procedures . . . 23

4.3.3 Build The Singularity Image . . . 23
4.3.4 Test The Singularity Image . . . 25
4.4 Build a Singularity Dask Image . . . 27
4.4.1 Prerequisites . . . 27
4.4.2 Notes on the Procedures . . . 27
4.4.3 Build the Singularity Image . . . 27
4.5 Build a Singularity Miniconda Image . . . 28
4.5.1 Prerequisites . . . 28
4.5.2 Notes on the Procedures . . . 28
4.5.3 Build the Singularity Image . . . 29
4.6 Build a Singularity Anaconda Image . . . 30
4.6.1 Prerequisites . . . 30
4.6.2 Notes on the Procedures . . . 30
4.6.3 Build the Singularity Image . . . 30
4.7 Build a Singularity Spark Image . . . 32
4.7.1 Prerequisites . . . 32
4.7.2 Notes on The Procedures . . . 32
4.7.3 Build the Singularity Image . . . 32

5 Analytics User Operations **33**
5.1 Spark MKL Tests . . . 33
5.1.1 Notes on the Tests . . . 34
5.1.2 Tests . . . 34
5.2 Pytorch MKL Tests . . . 34
5.2.1 Notes on the Tests . . . 34
5.2.2 Tests . . . 34
5.3 TensorFlow MKL Tests . . . 35
5.3.1 Notes on the Tests . . . 35
5.3.2 Tests . . . 35

6 Glossary **37**
6.0.1 Customer Access Network . . . 37
6.0.2 Compute Node (CN) . . . 37
6.0.3 Cray System Management (CSM) . . . 37
6.0.4 High Speed Network (HSN) . . . 37
6.0.5 Management Nodes . . . 37
6.0.6 Node Management Network . . . 37
6.0.7 Non-Compute Node (NCN) . . . 37
6.0.8 UAI . . . 38
6.0.9 UAN . . . 38
6.0.10 xname . . . 38

1 Copyright and Version

Analytics: 1.2.22-175

Doc git hash: 994ac7dc96c64495399412a2504f39378873d6b3

Generated: Tue Jun 28 2022

2 Introduction

2.1 About the HPE Cray EX Analytics Applications Guide

The following table shows the version history of this document.

Publication Title	Date
HPE Cray EX Analytics Applications Guide 1.2.22 S-8045	July 2022
HPE Cray EX Urika Analytic Applications Guide 1.4.1 S-8006	April 2021
HPE Cray EX Urika Analytic Applications Guide 1.4 S-8006	March 2021
Cray Shasta Urika Analytic Applications Guide 1.3 S-8006	August 2020
Cray Shasta Urika Analytic Applications Guide 1.2 S-8006	April 2020
Cray Shasta Urika Analytic Applications Guide 1.1 S-8006	January 2020

2.1.1 Scope and Audience

This publication is intended for data scientists and engineers who want to use open source analytics frameworks on the HPE Cray EX system. It assumes familiarity with standard Linux and the open source tools used on the system.

This publication does not include information about installing the base HPE Cray EX product. For details on installing the HPE Cray EX product, see the ***HPE Cray EX System Software Getting Started Guide S-8000*** and the Cray System Management (CSM) documentation.

For information about the User Access Service (UAS) and creating a User Access Instance (UAI), see the CSM documentation.

2.1.2 Typographic Conventions

- `Monospace`: Indicates program code, reserved words, library functions, command-line prompts, screen output, file/path names, and other software constructs.
- **`Monospaced Bold`**: Indicates commands that must be entered on a command line or in response to an interactive prompt.
- *`Oblique or Italics`*: Indicates user-supplied values in commands or syntax definitions.
- **Proportional Bold**: Indicates a **GUI Window**, **GUI element**, cascading menu (**Ctrl** > **Alt** > **Delete**), or key strokes (press **Enter**).
- \: A backslash at the end of a command line indicates a shell or command line continuation character (lines joined by a backslash are parsed as a single line).

2.1.3 Trademarks

2.2 About HPE Cray EX Analytics and AI

The HPE Cray EX system is capable of running open source analytics and AI components for performing big data and deep learning tasks. These components run as Singularity Containers. Documentation for building the container images and running them are provided. Documentation for installing specific third-party analytics and AI applications onto the HPE Cray EX system is not provided.

Analytics and AI applications are used through the User Access Service (UAS) and User Access Node (UAN).

2.2.1 Supported Analytics and AI Frameworks

- **Apache™ Spark™** is a general data processing framework that simplifies developing big data applications. It provides the means for executing batch, streaming, and interactive analytics jobs.
- **Dask Distributed** is a centrally managed, distributed, dynamic task scheduler. It coordinates several worker processes spread across multiple machines and the concurrent requests of several clients.
- **PyTorch™** is an open source optimized tensor library and deep learning framework for Python. It is designed to be deeply integrated into Python.
- **TensorFlow™** is a software library for dataflow programming across a range of tasks. It is a symbolic math library, which is also used for machine learning applications such as neural networks.

2.2.2 Prerequisites for Running Analytics and AI jobs

To run analytics and AI jobs on the HPE Cray EX system, users must be signed in to User Access Node (UAN), or there must be an active User Access Instance (UAI).

For instructions on creating a UAI, refer to the “Create a UAI” section in the CSM documentation.

2.3 Documentation Conventions

Several conventions have been used in the preparation of this documentation.

- Markdown Format
- File Formats
- Typographic Conventions
- Command Prompt Conventions which describe the context for user, host, directory, chroot environment, or container environment

2.3.1 Markdown Format

This documentation is in Markdown format. Although much of it can be viewed with any text editor, a richer experience will come from using a tool which can render the Markdown to show different font sizes, the use of bold and italics formatting, inclusion of diagrams and screen shots as image files, and to follow navigational links within a topic file and to other files.

There are many tools which can render the Markdown format to get these advantages. Any Internet search for Markdown tools will provide a long list of these tools. Some of the tools are better than others at displaying the images and allowing you to follow the navigational links.

2.3.2 File Formats

Some of the installation instructions require updating files in JSON, YAML, or TOML format. These files should be updated with care because some file formats do not accept tab characters for indentation of lines. Only space characters are supported. Refer to online documentation to learn more about the syntax of JSON, YAML, and TOML files.

2.3.3 Typographic Conventions

`This style` indicates program code, reserved words, library functions, command-line prompts, screen output, file/path names, and other software constructs.

(backslash) At the end of a command line, indicates the Linux shell line continuation character (lines joined by a backslash are parsed as a single line).

2.3.4 Command Prompt Conventions

2.3.4.1 Host name and account in command prompts

The host name in a command prompt indicates where the command must be run. The account that must run the command is also indicated in the prompt. - The root or super-user account always has the # character at the end of the prompt - Any non-root account is indicated with account@hostname>. A non-privileged account is referred to as user.

2.3.4.2 Node abbreviations

The following list contains abbreviations for nodes used below

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

Prompt	Description
ncn#	Run the command as root on any NCN, except an NCN which is functioning as an Application Node (AN), such as a UAN.
ncn-m#	Run the command as root on any NCN-M (NCN which is a Kubernetes master node).

Prompt	Description
ncn-m002#	Run the command as root on the specific NCN-M (NCN which is a Kubernetes master node) which has this hostname (ncn-m002).
ncn-w#	Run the command as root on any NCN-W (NCN which is a Kubernetes worker node).
ncn-w001#	Run the command as root on the specific NCN-W (NCN which is a Kubernetes master node) which has this hostname (ncn-w001).
ncn-s#	Run the command as root on any NCN-S (NCN which is a Utility Storage node).
ncn-s003#	Run the command as root on the specific NCN-S (NCN which is a Utility Storage node) which has this hostname (ncn-s003).
pit#	Run the command as root on the PIT node.
linux#	Run the command as root on a Linux host.
uan#	Run the command as root on any UAN.
uan01#	Run the command as root on hostname uan01.
user@uan>	Run the command as any non-root user on any UAN.
cn#	Run the command as root on any CN. Note that a CN will have a hostname of the form nid124356, that is “nid” and a six digit, zero padded number.
hostname#	Run the command as root on the specified hostname.
user@hostname>	Run the command as any non-root user son the specified hostname.

2.3.4.3 Command prompt inside chroot

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

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

2.3.4.4 Command prompt inside Kubernetes pod

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

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

2.3.4.5 Command prompt inside image customization session

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

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

2.3.4.6 Directory path in command prompt

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

Examples of prompts as they appear on the system:

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

Examples of prompts as they appear in this publication:

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

2.3.4.7 Command prompts for network switch configuration

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

Example of prompts as they appear in this publication:

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

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

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

2.4 Analytics Release Notes

These release notes list new features and known issues as of the latest release – Analytics 1.2.

2.4.1 New Features

- Instructions to create TensorFlow and PyTorch images with Horovod support using Singularity. These images target CPUs and GPUs and are tested with CUDA 11.0.
- Instructions to create Miniconda and Anaconda images using Singularity.
- Instructions to create Spark, Pyspark and SparkR images using Singularity.
- Instructions to create Dask images and one way to start a Dask Cluster for distributed programming.
- Chapel version 1.26.0.

2.4.2 Known Issues

None.

2.4.3 Component Versions

Component	Current Version
Chapel	1.26.0

3 Install Analytics

3.1 Install the Analytics Product Stream Update

Describes the steps needed to install the Analytics Product Stream update.

3.1.1 Prerequisites

- Cray System Management (CSM) is installed and verified.
- UAN
- WLM

- PE
- gitea, cray-product-catalog, cray-cfs-api, cray-cfs-operator, and cray-ims are running.

3.1.2 Procedure

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

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

2. Copy the release distribution gzipped tar file, e.g. analytics-1.2.22.tar.gz, to ncn-m001.

3. Extract the Analytics release distribution.

   ```
   ncn-m001# tar -zxvf analytics-1.2.22.tar.gz
   ```

4. Change directories to the extracted directory.

   ```
   ncn-m001# cd analytics-1.2.22
   ```

5. Verify the prerequisites.

   Cray System Management (CSM) has been installed and verified including the following Helm Charts:

   - gitea
   - cray-product-catalog
   - cray-cfs-api
   - cray-cfs-operator
   - cray-ims
   - cray-uan
   - WLM (slurm or pbs)

   Verify the installation of these components with the following.

   ```
   ncn-m001# helm ls -n services | grep -E \
   'gitea|cray-cfs-operator|cray-cfs-api|cray-ims|cray-product-catalog|uan|slurm|pbs'
   ```

   Cray PE

   - If Cray PE will be installed on the system, ensure Cray PE is installed and configured before installing Analytics.

6. Run the installer, `install.sh`.

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

7. Run the `post-install.sh` script, to configure Analytics on the system (approximately 5 to 10 minutes).

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

8. Exit the typescript file started at the beginning of this procedure.

   ```
   ncn-m001# exit
   ```

At this point, the Analytics software has been installed:

- Analytics configuration content for this release has been uploaded to VCS.
- Analytics image for this release has been uploaded to IMS.
- Analytics information for this release has been uploaded to the CSM product catalog.
- Analytics content for this release has been uploaded to Nexus repositories.
- Analytics microservices have been upgraded if new versions were available.

The remainder of this document describes how to:

- Upgrade Analytics software, for HPE Cray EX systems that already have a version of Analytics installed
- Configure Analytics host software on NCNs via NCN Personalization
- Personalize the compute nodes with Analytics content
- Personalize the UAN nodes with Analytics content

If other HPE Cray EX software products are being installed in conjunction with Analytics, refer to the ***HPE Cray EX System Software Getting Started Guide*** to determine what step to perform next.

If other HPE Cray EX software products are not being installed at this time, continue to **Configure the Analytics Product Stream**.

3.2 Upgrade the Analytics Product Stream

Describes the steps needed to Upgrade the Analytics Product Stream update.

3.2.1 Prerequisites

- Cray System Management (CSM) is installed and verified.
- UAN
- WLM
- PE
- gitea, cray-product-catalog, cray-cfs-api, cray-cfs-operator, and cray-ims are running.

3.2.2 Procedure

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

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

2. Copy the release distribution gzipped tar file, e.g. analytics-1.2.22.tar.gz, to ncn-m001.

3. Extract the Analytics release distribution.

   ```
   ncn-m001# tar -zxvf analytics-1.2.22.tar.gz
   ```

4. Change directories to the extracted directory.

   ```
   ncn-m001# cd analytics-1.2.22
   ```

5. Verify the prerequisites.

   Cray System Management (CSM) has been installed and verified including the following Helm Charts:

   - gitea
   - cray-product-catalog
   - cray-cfs-api
   - cray-cfs-operator
   - cray-ims
   - cray-uan
   - WLM (slurm or pbs)

   Verify the installation of these components with the following.

   ```
   ncn-m001# helm ls -n services | grep -E \
   'gitea|cray-cfs-operator|cray-cfs-api|cray-ims|cray-product-catalog|uan|slurm|pbs'
   ```

   Cray PE

   - If Cray PE will be installed on the system, ensure Cray PE is installed and configured before installing Analytics.

6. Back up the existing Analytics configuration, by running `pre-upgrade.sh`

   ```
   ncn-m001# ./pre-upgrade.sh
   ```

7. Perform the upgrade, `upgrade.sh`.

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

8. Run the `post-upgrade.sh` script, to configure Analytics on the system (approximately 5 to 10 minutes).

   ```
   ncn-m001# ./post-upgrade.sh
   ```

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

```
ncn-m001# exit
```

At this point, Analytics software has been upgraded to version 1.2:

- Analytics configuration content for this release has been uploaded to VCS.
- Analytics image for this release has been uploaded to IMS.
- Analytics information for this release has been uploaded to the CSM product catalog.
- Analytics content for this release has been uploaded to Nexus repositories.
- Analytics microservices have been upgraded if new versions were available.

The remainder of this document describes how to:

- Configure Analytics host software on NCNs via NCN Personalization
- Personalize the compute nodes with Analytics content
- Personalize the UAN nodes with Analytics content

If other HPE Cray EX software products are being upgraded in conjunction with Analytics, refer to the ***HPE Cray EX System Software Getting Started Guide*** to determine what step to perform next.

If other HPE Cray EX software products are not being installed at this time, continue to **Configure the Analytics Product Stream**.

3.3 Configure the Analytics Product Stream

Describes the steps needed to configure the Analytics Product Stream for HPE Cray EX Raspberry

3.3.1 Prerequisites

- Cray System Management (CSM) is installed and verified.
- UAN
- WLM
- PE
- gitea, cray-product-catalog, cray-cfs-api, cray-cfs-operator, and cray-ims are running.

3.3.2 Pre-Procedure

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

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

2. Change directories to the previously extracted analytics product stream directory.

   ```
   ncn-m001# cd analytics-1.2.22
   ```

3. Verify Analytics has been Installed or Upgraded to version 1.2.22.

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

3.3.3 Perform NCN Personalization

Add an analytics layer to the existing NCN configuration.

1. Run the `deploy-ncns.sh` script.

   The script will create a new configuration file locally.

   NOTE: the argument `new.json` the following example is an example value. The script will use this value to build the filename for the new configuration file. If the existing configuration file name is `ncn-personalization`, the new file name will be `ncn-personalization-new.json`. The file will be created in the current working directory.

   ```
   ncn-m001# ./deploy-ncns.sh --active new.json
   ```

2. Update the existing configuration.

   Run `cat ncn-personalization-new.json` and review the output to verify that no mistakes were introduced to the new configuration file. NOTE: Ensure that the analytics layer is listed before a layer named `ncn-final` if that layer exists.

If there are no errors, run `cray cfs configurations update` to update the configuration.

```
ncn-m001# cat ncn-personalization-new.json    # verify integrity of new config
ncn-m001# cray cfs configurations update ncn-personalization --file ncn-personalization-new.json
```

At this point, the Analytics content has been configured on NCNs.

If other HPE Cray EX software products are being installed or upgraded in conjunction with Analytics, refer to the ***HPE Cray EX System Software Getting Started Guide*** to determine what step to perform next.

If other HPE Cray EX software products are not being installed at this time, continue to **Use SAT to Perform Compute Node Operations**.

3.3.4 Use SAT to Perform Compute Node Operations

This section provides detailed instructions on how to setup Analytics personalization on compute node images in the following section:

- Perform Compute Node Personalization

However, if System Admin Toolkit (SAT) version 2.2.16 or later is installed, the `sat bootprep` command can be used to perform the tasks outlined in section **Perform Compute Node Personalization** more quickly and with fewer operations. Use `sat showrev` to determine which version of SAT is installed on the system.

If the `sat bootprep` command is used, the content in the **Perform Compute Node Personalization** is mainly informational and does not have to be followed to perform these operations.

3.3.4.1 SAT Bootprep Details

The "SAT Bootprep" section of the ***HPE Cray EX System Admin Toolkit (SAT) Guide*** provides information on how to use `sat bootprep` to create CFS configurations, build images with IMS, and create BOS session templates. To include Analytics software and configuration data in these operations, ensure that the `sat bootprep` input file includes content similar to that described in the following subsections.

NOTE: The `sat bootprep` input file will contain content for additional HPE Cray EX software products and not only COS. The following examples focus on Analytics entries only.

3.3.4.1.1 Analytics Configuration Content

The installation procedure created a yaml input file to use with `sat bootprep`, located here: `/etc/cray-analytics.d/analytics-1.2.22-satl`

To ensure that Analytics configuration data is used by the compute image and compute nodes, the input file should contain a section similar to the following example:

```
configurations:
- name: analytics-config
  layers:
  - name: analytics-1.2.22-integration
    playbook: site.yml
    product:
      name: analytics
      version: 1.2.22
      branch: integration
```

At this point, `sat bootprep` will ensure that the appropriate compute images are properly configured and personalized with the Analytics information provided in the template.

If other HPE Cray EX software products are being installed in conjunction with Analytics, see the ***HPE Cray EX System Software Getting Started Guide*** to determine what step to perform next. If other HPE Cray EX software products are not being installed at this time, continue to the next sections of this document if you want to perform these operations without using `sat bootprep`.

3.3.5 Perform Compute Node Personalization

Add an analytics layer to the existing Compute Node Configuration.

1. Run the `deploy-computes.sh` script.

NOTE: the argument `new.json` the following example is an example value. The script will use this value to build the file-name for the new configuration file. If the existing configuration file name is `cos-1.2.3-compute`, the new file name will be `cos-1.2.3-compute-new.json`. The file will be created in the current working directory.

```
ncn-m001# ./deploy-computes.sh --active new.json
```

2. Update the existing configuration.

   Run `cat cos-1.2.3-compute-new.json` and review the output to verify that no mistakes were introduced to the new configuration file. NOTE: Ensure that the analytics layer is listed before a layer named `ncn-final` if that layer exists.

   If there are no errors, run `cray cfs configurations update` to update the configuration.

```
ncn-m001# cat cos-1.2.3-compute-new.json    # verify itegrity of new config
ncn-m001# cray cfs configurations update cos-1.2.3-compute --file cos-1.2.3-compute-new.json
```

3. Publish the Analytics software into newly created UAIs.

   Run the following commands.

   NOTE: compute, UAN, and NCN configurations can have the Analytics layer added at the end of the json file.

```
ncn-m001# kubectl get pods -A --sort-by=.metadata.creationTimestamp| \
    grep cfs    # verify CFS has completed deploying to NCN nodes
ncn-m001# /etc/cray-analytics.d/uas_setup_analytics.sh
```

At this point, the Analytics content has been configured on Compute Nodes.

If other HPE Cray EX software products are being installed or upgraded in conjunction with Analytics, refer to the ***HPE Cray EX System Software Getting Started Guide*** to determine what step to perform next.

If other HPE Cray EX software products are not being installed at this time, continue to **Perform UAN Personalization**.

3.3.6 Perform UAN Personalization

Add an analytics layer to the existing UAN configuration.

1. Run the `deploy-uans.sh` script.

   NOTE: the argument `new.json` the following example is an example value. The script will use this value to build the file-name for the new configuration file. If the existing configuration file name is `uan-customization`, the new file name will be `uan-customization-new.json`. The file will be created in the current working directory.

```
ncn-m001# ./deploy-uans.sh --active new.json
```

   NOTE: On HPE EX Systems that do not have UAN(s), the previous command will simply inform the admin that no UANs were detected and no UAN configurations were found.

2. Update the existing configuration.

   Run `cat uan-customization-new.json` and review the output to verify that no mistakes were introduced to the new configuration file. NOTE: Ensure that the analytics layer is listed before a layer named `ncn-final` if that layer exists.

   If there are no errors, run `cray cfs configurations update` to update the configuration.

```
ncn-m001# cat uan-customization-new.json    # verfy itegrity of new config
ncn-m001# cray cfs configurations update uan-customization --file uan-customization-new.json
```

At this point, the Analytics content has been configured on UANs.

3.3.7 Post-Procedure

3. Stop the typescript.

```
ncn-m001# exit
```

At this point, the Analytics content has been fully installed or upgraded, and configured.

If other HPE Cray EX software products are being installed or upgraded in conjunction with Analytics, refer to the ***HPE Cray EX System Software Getting Started Guide*** to determine what step to perform next.

4 Analytics Admin Operations

4.1 Install Singularity

This sub-section describes how to download and install Singularity on a Shasta system. Singularity is a container platform that facilitates the creation of container images on one machine and their execution across cloud and HPC clusters. It is commonly used to execute AI and Machine Learning workloads on HPC cluster computers.

NOTE: For in-depth installation instructions, refer to Singularity's Installation Guide.

4.1.1 Prerequisites

- The Cray Command Line Interface (CLI) tool is initialized and configured on the system.
- The `kubectl` command is installed and available.

4.1.2 Procedure

1. Create a Singularity RPM file and upload it to the Nexus Package Manager.

    1. Install the pre-requisites.

        ```
        zypper addrepo https://download.opensuse.org/repositories/\
        openSUSE:Leap:15.2:Update/standard/openSUSE:Leap:15.2:Update.repo
        zypper addrepo https://download.opensuse.org/repositories/\
        devel:languages:go/openSUSE_Leap_15.2/devel:languages:go.repo
        zypper install -y go git libseccomp-devel rpm-build wget
        ```

    2. Download the tar file from the GitHub Singularity repository and create the RPM file.

        ```
        export SINGULARITY_VERSION=3.8.3 # update to the version needed
        wget https://github.com/sylabs/singularity/releases/\
        download/v${SINGULARITY_VERSION}/singularity-ce-${SINGULARITY_VERSION}.tar.gz
        rpmbuild -tb singularity-ce-${SINGULARITY_VERSION}.tar.gz
        ```

        NOTE: The environment variable `SINGULARITY_VERSION` is also used in subsequent steps. Adding it to the files like `.bashrc` will ensure its value remains constant even for multiple terminal sessions.

    3. Register the Singularity Repository with Nexus Package Manager.

        ```
        ncn-m001:~/admin $ vim nexus-blobstores.json
        {
            "name": "singularity",
            "path": "/nexus-data/blobs/singularity",
            "softQuota": null
        }
        ncn-m001:~/admin $ vim nexus-repositories.json
        {
            "name": "singularity-ce-3.8.3-compute",
            "online": true,
            "storage": {
                "blobStoreName": "singularity",
                "strictContentTypeValidation": false,
                "writePolicy": "ALLOW_ONCE"
            },
            "cleanup": null,
            "format": "raw",
            "type": "hosted"
        }
        ncn-m001:~/admin $ curl -sfkSL -X POST \
        https://packages.local/service/rest/beta/blobstores/file \
        ```

```
-H 'Content-Type:application/json' -d @nexus-blobstores.json
ncn-m001:~/admin $ curl -sfkSL -X POST \
https://packages.local/service/rest/beta/repositories/raw/hosted \
-H 'Content-Type:application/json' -d @nexus-repositories.json
```

NOTE: Update the version number in the `name` field of the `nexus-repositories.json` file to match the version of the Singularity downloaded.

4. Verify that the Singularity repository is registered in the Nexus Package Manager.

```
ncn-m001:~/admin $ curl -sS https://packages.local/service/rest/beta/repositories | \
jq '.[] | select(.name == "singularity-ce-3.8.3-compute")'
{
    "name": "singularity-ce-3.8.3-compute",
    "format": "raw",
    "url": "https://packages.local/repository/singularity-ce-3.8.3-compute",
    "online": true,
    "storage": {
        "blobStoreName": "singularity",
        "strictContentTypeValidation": false,
        "writePolicy": "ALLOW_ONCE"
    },
    "cleanup": null,
    "type": "hosted"
}
```

5. Create a Singularity repository and upload its contents to the Nexus Package Manager.

```
ncn-m001:~/admin $ mkdir -p rpms/singularity
ncn-m001:~/admin $ cd rpms/singularity
ncn-m001:~/admin/rpms/singularity $ cp \
/usr/src/packages/RPMS/x86_64/singularity-ce-${SINGULARITY_VERSION}-1.x86_64.rpm .
ncn-m001:~/admin/rpms/singularity $ cp \
/usr/src/packages/RPMS/x86_64/singularity-ce-debuginfo-${SINGULARITY_VERSION}-1.x86_64.rpm .
ncn-m001:~/admin $ cd ../..
ncn-m001:~/admin $ createrepo ./rpms/singularity
ncn-m001:~/admin $ ls -R rpms/singularity/
rpms/singularity/:
repodata  singularity-ce-3.8.3-1.x86_64.rpm  singularity-ce-debuginfo-3.8.3-1.x86_64.rpm

rpms/singularity/repodata:
0d2ae1bc79c2ff4c0893718c0267fb1d3c2858114edd7ce9693fccbf53e70a47-other.xml.gz
29e121106a332bbdde2b3201c7b8fc38d01e7f8bfde10a5c3b0eb2c3a6c234d0-other.sqlite.bz2
3112e51fbad191f7c7469636a286bcc139a20823f9f78211b63137cb25e3c6f8-primary.xml.gz
48f8f3b7fd6f11cf61fd0fe4473244412363985b0432ece1ab4b1b10912808c5-primary.sqlite.bz2
faa7f25d86ab301eb441f3fc7df79b6cb2e90032f650673faf4572a288db63b8-filelists.sqlite.bz2
ff6b71e3627c699e2affc200d1ff97bf8245d3f8504481b044fcd6e62113bb55-filelists.xml.gz
repomd.xml

ncn-m001:~/admin $ cd rpms/singularity/
ncn-m001:~/admin/rpms/singularity $ curl -v --upload-file \
singularity-ce-3.8.3-1.x86_64.rpm \
https://packages.local/repository/singularity-ce-3.8.3-compute/
ncn-m001:~/admin/rpms/singularity $ curl -v --upload-file \
singularity-ce-debuginfo-3.8.3-1.x86_64.rpm \
https://packages.local/repository/singularity-ce-3.8.3-compute/
ncn-m001:~/admin/rpms/singularity $ curl -v --upload-file repodata \
https://packages.local/repository/singularity-ce-3.8.3-compute/
ncn-m001:~/admin/rpms/singularity $ cd repodata/
ncn-m001:~/admin/rpms/singularity/repodata $ curl -v --upload-file \
```

```
0d2ae1bc79c2ff4c0893718c0267fb1d3c2858114edd7ce9693fccbf53e70a47-other.xml.gz \
https://packages.local/repository/singularity-ce-3.8.3-compute/repodata/
ncn-m001:~/admin/rpms/singularity/repodata $ curl -v --upload-file \
29e121106a332bbdde2b3201c7b8fc38d01e7f8bfde10a5c3b0eb2c3a6c234d0-other.sqlite.bz2 \
https://packages.local/repository/singularity-ce-3.8.3-compute/repodata/
ncn-m001:~/admin/rpms/singularity/repodata $ curl -v --upload-file \
3112e51fbad191f7c7469636a286bcc139a20823f9f78211b63137cb25e3c6f8-primary.xml.gz \
https://packages.local/repository/singularity-ce-3.8.3-compute/repodata/
ncn-m001:~/admin/rpms/singularity/repodata $ curl -v --upload-file \
48f8f3b7fd6f11cf61fd0fe4473244412363985b0432ece1ab4b1b10912808c5-primary.sqlite.bz2 \
https://packages.local/repository/singularity-ce-3.8.3-compute/repodata/
ncn-m001:~/admin/rpms/singularity/repodata $ curl -v --upload-file \
faa7f25d86ab301eb441f3fc7df79b6cb2e90032f650673faf4572a288db63b8-filelists.sqlite.bz2 \
https://packages.local/repository/singularity-ce-3.8.3-compute/repodata/
ncn-m001:~/admin/rpms/singularity/repodata $ curl -v --upload-file \
ff6b71e3627c699e2affc200d1ff97bf8245d3f8504481b044fcd6e62113bb55-filelists.xml.gz \
https://packages.local/repository/singularity-ce-3.8.3-compute/repodata/
ncn-m001:~/admin/rpms/singularity/repodata $ curl -v --upload-file repomd.xml \
https://packages.local/repository/singularity-ce-3.8.3-compute/repodata/
```

2. Install Singularity by customizing an image root.

   HPE recommends installing third-party software on a Shasta system by customizing an image root with the third-party software installed on it. This method ensures that administrators are able to re-image nodes with the customized image, and that the software will still be available after a node is rebooted.

   The process of customizing an image root involves utilizing multiple system services like Cray Command Line Interface (CLI), Version Control Service (VCS), Nexus Package Manager, Boot Script Service (BSS), Boot Orchestration Service (BOS), Image Management Service (IMS), and Configuration Framework Service (CFS). For in-depth details regarding each of these services, refer to the ***HPE Cray EX System Administration Toolkit (SAT) Guide***.

   The following steps will customize an image root with Singularity installed on it.

   1. Retrieve the IMS Image ID for Compute Nodes.

      1. Retrieve compute image details using Boot Script Service (BSS) and store the ROOTFS id in an environment variable.

         ```
         ncn-m001:~ $ cray bss bootparameters list --nids 1
         [[results]]
         hosts = [ "x3000c0s19b1n0",]
         params = "... spire_join_token=${SPIRE_JOIN_TOKEN} ..."
         kernel = "s3://boot-images/14dc7b54-ed29-41a5-9448-98c5c2ea7708/kernel"
         initrd = "s3://boot-images/14dc7b54-ed29-41a5-9448-98c5c2ea7708/initrd"

         [results.cloud-init.phone-home]
         pub_key_dsa = ""
         pub_key_rsa = ""
         pub_key_ecdsa = ""
         instance_id = ""
         hostname = ""
         fqdn = ""

         ncn-m001:~ $ export ROOTFS=14dc7b54-ed29-41a5-9448-98c5c2ea7708
         ```

      2. Retrieve image ID using Image Management Service (IMS) and the ROOTFS_ID saved in the previous step, and save the image ID in an environment variable.

         ```
         ncn-m001:~ $ cray ims images list | grep -C 3 $ROOTFS
         type = "s3"
         [[results]]
         created = "2021-12-07T21:43:27.879303+00:00"
         id = "14dc7b54-ed29-41a5-9448-98c5c2ea7708"
         ```

```
name = "cos-2.1.70-recipe-slingshot-host-software-1.5_cfs_slurm-cos-1.1.1"

[results.link]
etag = "09ebbca016124bbcf5cf4bf7164b7ea7"
path = "s3://boot-images/14dc7b54-ed29-41a5-9448-98c5c2ea7708/manifest.json"
type = "s3"
[[results]]
created = "2021-12-08T00:25:44.466600+00:00"

ncn-m001:~ $ export IMS_IMAGE_ID=14dc7b54-ed29-41a5-9448-98c5c2ea7708
```

3. Verify that the image exists in IMS using the `IMS_IMAGE_ID` value, saved in the previous step.

   This image will be used for customizing in the subsequent steps.

   ```
   ncn-m001:~ $ cray ims images describe $IMS_IMAGE_ID
   created = "2021-12-07T21:43:27.879303+00:00"
   id = "14dc7b54-ed29-41a5-9448-98c5c2ea7708"
   name = "cos-2.1.70-recipe-slingshot-host-software-1.5_cfs_slurm-cos-1.1.1"

   [link]
   etag = "09ebbca016124bbcf5cf4bf7164b7ea7"
   path = "s3://boot-images/14dc7b54-ed29-41a5-9448-98c5c2ea7708/manifest.json"
   type = "s3"
   ```

2. Create a VCS repository for Singularity.

   1. Retrieve the password for the `crayuser` user on the Version Control Service (VCS), and store it in an environment variable.

      This environment variable will be used in the subsequent commands.

      ```
      ncn-m001:~/admin $ VCSPWD=$(kubectl get secret -n services vcs-user-credentials \
      --template={{.data.vcs_password}} | base64 --decode)
      ```

   2. Create an empty VCS repository for Singularity.

      This repository will be updated with relevant files in the subsequent steps.

      ```
      ncn-m001:~/admin $ curl -X POST \
      https://api-gw-service-nmn.local/vcs/api/v1/org/cray/repos \
      -d '{"name": "singularity"}' -u crayvcs:$VCSPWD \
      -H "Content-Type: application/json"
      ```

   3. Clone the Singularity repository.

      ```
      ncn-m001:~/admin $ git clone https://api-gw-service-nmn.local/vcs/cray/singularity.git
      ```

   4. Populate the Singularity repository with the following files.

      ```
      ncn-m001:~/admin $ cd singularity/
      ncn-m001:~/admin/singularity $ vim site.yml
      ncn-m001:~/admin/singularity $ cat site.yml
      ---
      # Set up singularity on computes
      - hosts: Compute
        roles:
        - singularity_repos
        - singularity_node

      ncn-m001:~/admin/singularity $ mkdir roles
      ncn-m001:~/admin/singularity $ cd roles/
      ncn-m001:~/admin/singularity/roles $ mkdir singularity_node
      ncn-m001:~/admin/singularity/roles $ mkdir singularity_repos
      ncn-m001:~/admin/singularity/roles $ cd singularity_node/
      ncn-m001:~/admin/singularity/roles/singularity_node $ mkdir defaults
      ```

```
ncn-m001:~/admin/singularity/roles/singularity_node $ mkdir tasks
ncn-m001:~/admin/singularity/roles/singularity_node $ cd defaults/
ncn-m001:~/admin/singularity/roles/singularity_node/defaults $ vim main.yml
ncn-m001:~/admin/singularity/roles/singularity_node/defaults $ cat main.yml
---
singularity_rpms:
  - singularity-ce
  - singularity-ce-debuginfo

ncn-m001:~/admin/singularity/roles/singularity_node/defaults $ cd ../tasks/
ncn-m001:~/admin/singularity/roles/singularity_node/tasks $ vim main.yml
ncn-m001:~/admin/singularity/roles/singularity_node/tasks $ cat main.yml
---
- name: Install/upgrade singularity with zypper
  zypper:
    name: "{{ singularity_rpms }}"
    state: latest
    disable_gpg_check: yes
  when: cray_cfs_image|default(false)|bool
  register: result
  until: result is not failed
  retries: 3
  delay: 5

ncn-m001:~/admin/singularity/roles/singularity_node/tasks $ cd ../../singularity_repos/
ncn-m001:~/admin/singularity/roles/singularity_repos $ ls
ncn-m001:~/admin/singularity/roles/singularity_repos $ mkdir defaults
ncn-m001:~/admin/singularity/roles/singularity_repos $ mkdir tasks
ncn-m001:~/admin/singularity/roles/singularity_repos $ cd defaults/
ncn-m001:~/admin/singularity/roles/singularity_repos/defaults $ vim main.yml
ncn-m001:~/admin/singularity/roles/singularity_repos/defaults $ cat main.yml
---
# Repositories containing singularity RPMs
singularity_repos:
  - https://packages.local/repository/singularity-ce-3.8.3-compute
# zypper repo priority, must be less than SLES repo priority (3 in COS 1.4.0)
singularity_repo_priority: 2

ncn-m001:~/admin/singularity/roles/singularity_repos/defaults $ cd ../tasks/
ncn-m001:~/admin/singularity/roles/singularity_repos/tasks $ vim main.yml
ncn-m001:~/admin/singularity/roles/singularity_repos/tasks $ cat main.yml
---
- name: Add singularity zypper repositories
  zypper_repository:
      repo: "{{ item }}"
      name: "{{ item | basename }}"
      priority: "{{ singularity_repo_priority }}"
      disable_gpg_check: yes
  with_items: "{{ singularity_repos }}"
  when: cray_cfs_image|default(false)|bool
```

5. Commit and push changes to the singularity repository.

```
ncn-m001:~/admin/singularity $ git add --all \
&& git commit -m "Initial singularity config" \
&& git push
```

6. Retrieve the git hash for the commit.

   This hash will be used in the subsequent steps.

```
ncn-m001:~/admin/singularity $ git rev-parse --verify HEAD
ef4c1ac8989c6192e7d27283bd5a2ebecd737821
```

3. Customize image root using CFS

    1. Create a Configuration Framework Service (CFS) config file for Singularity as shown below.

        Use the commit hash retrieved in the previous step for the `commit` field.

        ```
        ncn-m001:~/admin $ vim cfs_singularity_config.json
        ncn-m001:~/admin $ cat cfs_singularity_config.json
        {
          "layers": [
            {
              "name": "singularity-3.8.3",
              "cloneUrl": "https://api-gw-service-nmn.local/vcs/cray/singularity.git",
              "playbook": "site.yml",
              "commit": "ef4c1ac8989c6192e7d27283bd5a2ebecd737821"
            }
          ]
        }
        ```

    2. Update the CFS config with the new Singularity config.

        ```
        ncn-m001:~/admin $ cray cfs configurations update uan-config-2.1.2 \
        --file cfs_singularity_config.json
        lastUpdated = "2021-12-13T21:51:24Z"
        name = "uan-config-2.1.2"
        [[layers]]
        cloneUrl = "https://api-gw-service-nmn.local/vcs/cray/singularity.git"
        commit = "ef4c1ac8989c6192e7d27283bd5a2ebecd737821"
        name = "singularity-3.8.3"
        playbook = "site.yml"
        ```

    3. Run an image customization session using the updated CFS config and the previously retrieved IMS image ID.

        ```
        ncn-m001:~ $ cray cfs sessions create --name cfs-singularity --configuration-name \
        uan-config-2.1.2 --target-definition image --target-group Compute \
        14dc7b54-ed29-41a5-9448-98c5c2ea7708 --format json
        {
          "ansible": {
            "config": "cfs-default-ansible-cfg",
            "limit": null,
            "verbosity": 0
          },
          "configuration": {
            "limit": "",
            "name": "uan-config-2.1.2"
          },
          "name": "cfs-singularity",
          "status": {
            "artifacts": [],
            "session": {
              "completionTime": null,
              "job": null,
              "startTime": "2022-01-07T08:47:13",
              "status": "pending",
              "succeeded": "none"
            }
          },
          "tags": {},
          "target": {
        ```

```
    "definition": "image",
    "groups": [
      {
        "members": [
          "14dc7b54-ed29-41a5-9448-98c5c2ea7708"
        ],
        "name": "Compute"
      }
    ]
  }
}
```

4. Retrieve the Image ID for the Customized Root Image

    1. Retrieve the Kubernetes pod name for the CFS session using the `kubectl` command, and save the pod name in an environment variable.

        ```
        ncn-m001:~ $ kubectl get pods --no-headers -o custom-columns=":metadata.name" \
        -n services -l cfsession=cfs-singularity
        cfs-07e95646-da6b-4dc5-8ec8-fb41bdd22483-58hhn
        ncn-m001:~ $ export CFS_POD_NAME=cfs-07e95646-da6b-4dc5-8ec8-fb41bdd22483-58hhn
        ```

    2. After the CFS session is complete, view the log for the `teardown` container using the `CFS_POD_NAME` saved in the previous step. Locate and save the image ID for the new image in an environment variable.

        ```
        ncn-m001:~ $ kubectl logs -n services ${CFS_POD_NAME} -c teardown | tail -1
        2022-01-07 09:55:41,258 - INFO - cray.cfs.teardown - Updating cfsession=cfs-singularity
        artifacts with image=14dc7b54-ed29-41a5-9448-98c5c2ea7708
        result=b1d71311-4b22-4e4d-ac87-1c222070a671
        ncn-m001:~ $ export SINGULARITY_IMS_IMAGE_ID=b1d71311-4b22-4e4d-ac87-1c222070a671
        ```

    3. Verify the new image with Singularity is available on IMS using the `SINGULARITY_IMS_IMAGE_ID` saved in the previous step.

        ```
        ncn-m001:~ $ cray ims images describe $SINGULARITY_IMS_IMAGE_ID
        created = "2022-01-07T09:55:29.440769+00:00"
        id = "b1d71311-4b22-4e4d-ac87-1c222070a671"
        name = "cos-2.1.70-recipe-slingshot-host-software-1.5_cfs_slurm-cos-1.1.1_cfs-singularity"

        [link]
        etag = "ca224b4afe89b1de361546dc09fa0ecc"
        path = "s3://boot-images/b1d71311-4b22-4e4d-ac87-1c222070a671/manifest.json"
        type = "s3"
        ```

5. Boot the Compute Nodes using the new Customized Image

    1. Create a new Boot Orchestration Service (BOS) session template file and update image details to match the newly created Image.

        ```
        ncn-m001:~/admin $ vim bos-sessiontemplate-singularity-repo.json
        ncn-m001:~/admin $ cat bos-sessiontemplate-singularity-repo.json
        {
          "boot_sets": {
            "compute": {
              "boot_ordinal": 2,
              "kernel_parameters": "... ip=dhcp quiet spire_join_token=${SPIRE_JOIN_TOKEN}",
              "network": "nmn",
              "node_roles_groups": [
                "Compute"
              ],
              "path": "s3://boot-images/b1d71311-4b22-4e4d-ac87-1c222070a671/manifest.json",
              "rootfs_provider": "cpss3",
              "rootfs_provider_passthrough": "dvs:api-gw-service-nmn.local:300:nmn0",
        ```

```
        "etag": "ca224b4afe89b1de361546dc09fa0ecc",
        "type": "s3"
      }
    },
    "cfs": {
      "configuration": "cos-config-2.1.70"
    },
    "enable_cfs": true,
    "name": "bos-sessiontemplate-singularity-repo"
}
```

2. Create a new BOS session template.

   ```
   ncn-m001:~/admin $ cray bos sessiontemplate create \
   --name bos-sessiontemplate-singularity-repo \
   --file ./bos-sessiontemplate-singularity-repo.json
   /sessionTemplate/bos-sessiontemplate-singularity-repo
   ```

3. Create a BOS session to boot the compute nodes with the new customized image that has singularity installed in it.

   ```
   ncn-m001:~/admin $ cray bos session create \
   --template-uuid bos-sessiontemplate-singularity-repo \
   --operation boot
   ```

3. Verify the Singularity installation.

   The following methods are alternatives. Executing both is not necessary.

   - Print the installation location for Singularity with the `which` command.

     If Singularity is not installed, no path will be found.

     ```
     which singularity
     </path/to/binary> # if singularity is installed
     which: no singularity in ... # if singularity is not installed
     ```

   - Check the `--version` of the Singularity installation.

     ```
     singularity --version
     singularity version ...
     ```

4.2 Create a Singularity container from an NVIDIA Tensorflow image

This section describes how to create a Singularity container from an NVIDIA Tensorflow image.

4.2.1 Prerequisites

- This procedure requires that the user who is building the Singularity container has root privileges, as Singularity does not permit non-privileged users from creating containers.
- Internet access to pull down the Tensorflow2 image from the NVIDIA website.

4.2.2 Procedure

1. Log in to the UAN.
2. Determine the CUDA driver version on your CPU/GPU nodes.

   The version of the Tensorflow2 image that you will need depends on the CUDA driver version on your CPU/GPU nodes.

   Execute the commands shown below on a CPU/GPU node to obtain the CUDA driver version. In this example, the GrizzlyPeak SLURM partition contains the GPU nodes. Use the `salloc` options that are appropriate for your system. The GPU node in this example is running CUDA driver version 450.80.02.

   ```
   uan> salloc -N 1 --ntasks-per-node=1 -p GrizzlyPeak
   salloc: Granted job allocation 101113
   ```

```
uan> srun -n 1 nvidia-smi
Wed Oct 20 13:06:44 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.80.02    Driver Version: 450.80.02    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  A100-SXM4-40GB      On   | 00000000:03:00.0 Off |                    0 |
| N/A   26C    P0    51W / 400W |      0MiB / 40537MiB |      0%   E. Process |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   1  A100-SXM4-40GB      On   | 00000000:41:00.0 Off |                    0 |
| N/A   25C    P0    48W / 400W |      0MiB / 40537MiB |      0%   E. Process |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   2  A100-SXM4-40GB      On   | 00000000:82:00.0 Off |                    0 |
| N/A   26C    P0    48W / 400W |      0MiB / 40537MiB |      0%   E. Process |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   3  A100-SXM4-40GB      On   | 00000000:C1:00.0 Off |                    0 |
| N/A   25C    P0    49W / 400W |      0MiB / 40537MiB |      0%   E. Process |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+

uan> exit
exit
salloc: Relinquishing job allocation 101113
```

3. Use the NVIDIA support matrix to determine the Tensorflow2 image version that corresponds to your CUDA driver version.

   https://docs.nvidia.com/deeplearning/frameworks/support-matrix/index.html

4. Create a Singularity definition file for the Tensorflow2 image.

   This example uses the 21.07 image. Use the image version that is appropriate for your CPU/GPU nodes, as indicated in the previous steps.

   For systems with Internet access from which the NVIDIA image can be downloaded directly onto the UAN, create the Singularity definition file, as shown below.

   ```
   uan> cat nvidia-tensorflow2_21.07.def
   Bootstrap: docker
   From: nvcr.io/nvidia/tensorflow:21.07-tf2-py3
   ```

   For air-gapped systems (i.e., no Internet access) in which it's not possible to download the NVIDIA image directly onto the UAN, perform the following steps.

   1. On a computer with Internet access and Docker installed, pull the NVIDIA Tensorflow2 image.

      ```
      docker pull nvcr.io/nvidia/tensorflow:21.07-tf2-py3
      ```

   2. Save the image to a gzipped tar file.

```
docker save nvcr.io/nvidia/tensorflow:21.07-tf2-py3 | gzip > tensorflow_21.07-tf2-py3.tar.gz
```

3. Copy the image from your computer to the UAN on your system.

   ```
   scp tensorflow_21.07-tf2-py3.tar.gz username@host:/destination-directory
   ```

4. Create the Singularity definition file to bootstrap from the image contained in your gzipped tar file.

   Replace `path-to-image-dir` with the directory where the gzipped tar file resides.

   ```
   uan> cat nvidia-tensorflow2_21.07.def
   Bootstrap: docker-archive
   From: /path-to-image-dir/tensorflow_21.07-tf2-py3.tar.gz
   ```

5. Build the Singularity container from the definition file.

   This step requires root permissions.

   Place the Singularity container in a shared directory that is accessible from the compute nodes.

   Replace `path-to-containers-dir` with the appropriate shared containers directory for your system.

   ```
   uan> singularity build /lus/path-to-containers-dir/nvidia-tensorflow2_21.07.sif \
   nvidia-tensorflow2_21.07.def
   INFO:    Starting build...
   ...
   INFO:    Creating SIF file...
   INFO:    Build complete: nvidia-tensorflow2_21.07.sif
   ```

6. Verify that the Singularity container is readable by users.

   If the permissions are not correct, use the `chmod` command to change the permissions.

   ```
   uan> ls -l nvidia-tensorflow2_21.07.sif
   -rwxr-xr-x 1 root root 5604679680 Oct 25 10:04 nvidia-tensorflow2_21.07.sif
   ```

7. Verify that TensorFlow was built with CUDA (GPU) support.

   ```
   uan> salloc -N 1 --ntasks-per-node=1 -p GrizzlyPeak
   salloc: Granted job allocation 108151

   uan> srun --mpi=pmi2 -n 1 --ntasks-per-node=1 singularity exec --nv --bind /lus \
   /lus/<path-to-containers-dir>/nvidia-tensorflow2_21.07.sif \
   python -c '
   > import tensorflow as tf;
   > print("Tensorflow built with CUDA support =", tf.test.is_built_with_cuda())
   > '
   Tensorflow built with CUDA support = True

   uan> exit
   exit
   salloc: Relinquishing job allocation 108151
   ```

8. Download the `tensorflow2_mnist.py` test program.

   For `air-gapped` systems (i.e., no Internet access) in which it's not possible to download the test program directly, first download it onto a computer with Internet access and then copy it to the UAN.

   Place the test program in a shared directory that is accessible from the compute nodes.

   Replace `path-to-applications-dir` with the appropriate shared applications directory for your system.

   ```
   uan> wget https://github.com/horovod/horovod/blob/master/examples/tensorflow2/tensorflow2_mnist.py \
   -P /lus/path-to-applications-dir
   ```

   When executed, the `tensorflow2_mnist.py` test program will automatically download the `mnist.npz` data file it needs.

   For `air-gapped` systems, the `mnist.npz` data file cannot be automatically downloaded. To work around this issue, perform the following steps to use a local copy.

1. Download the `mnist.npz` data file onto a computer with Internet access.

   ```
   wget https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
   ```

2. Copy the `mnist.npz` data file to your UAN.

   ```
   scp mnist.npz username@host:/destination-directory
   ```

3. Edit the `tensorflow2_mnist.py` test program and make the following modification to the existing `mnist.load_data()` method to point to your `mnist.npz` data file.

   Replace `path-to-data-dir` with the appropriate shared data directory for your system.

   ```
   tf.keras.datasets.mnist.load_data(path='/lus/path-to-data-dir/mnist.npz')
   ```

9. Run the `tensorflow2_mnist.py` test program.

   This step does not need to be performed as a privileged user.

   In this example, the GrizzlyPeak SLURM partition contains the GPU nodes. Use the `salloc` options that are appropriate for your system. The "/lus" directory contains the users' home directories, which are accessible from the compute nodes.

   ```
   uan> salloc -N 2 --ntasks-per-node=4 -p GrizzlyPeak
   salloc: Granted job allocation 107475

   uan> srun --mpi=pmi2 -n 8 --ntasks-per-node=4 singularity exec --nv --bind /lus \
   /lus/<path-to-containers-dir>/nvidia-tensorflow2_21.07.sif \
   python /lus/path-to-applications-dir/tensorflow2_mnist.py
   ...
   Step #1220  Loss: 0.086334
   Step #1220  Loss: 0.066827
   Step #1230  Loss: 0.007979
   Step #1230  Loss: 0.059216
   Step #1240  Loss: 0.024316
   Step #1240  Loss: 0.034532
   hvd.local_rank(): 1, hvd.rank(): 1, hvd.size(): 8
   hvd.local_rank(): 2, hvd.rank(): 2, hvd.size(): 8
   hvd.local_rank(): 3, hvd.rank(): 3, hvd.size(): 8
   hvd.local_rank(): 2, hvd.rank(): 6, hvd.size(): 8
   hvd.local_rank(): 1, hvd.rank(): 5, hvd.size(): 8
   hvd.local_rank(): 3, hvd.rank(): 7, hvd.size(): 8

   uan> exit
   exit
   salloc: Relinquishing job allocation 107475
   ```

4.3 Build and Test a Singularity Image with GPU and CPU Support

This section describes how to build and test a Singularity image that has CPU and GPU support and includes PyTorch, OpenMPI, and Horovod. The Singularity image is built using an NVIDIA Docker image. Once built, it must be tested on CPU and GPU compute nodes.

4.3.1 Prerequisites

- Root permissions are required for creating the overlay image and for building the Singularity image.

4.3.2 Notes on The Procedures

- These procedures require read, write, and execute permissions for `nvidia-pytorch-21_09-py3.sif`.

4.3.3 Build The Singularity Image

A Singularity image may be built with an overlay, or with a Singularity definition file. The overlay method will create a writable Singularity image, while the method that uses a Singularity definition file will create a read-only image. Both options will create an image that is compatible with the test procedures that follow.

On EX systems, image building operations, including other install operations, such as `pip install`, require Internet access. After the Singularity image has been built, it must be made accessible to the GPU nodes. This can either be accomplished by copying it to the GPU nodes, or by placing it in a shared file system, such as Lustre.

Because the build options require Internet access, they cannot be executed on air-gapped systems. Built images must be copied onto air-gapped systems.

4.3.3.1 Option One: With An Overlay Embedded in A Singularity Image File (sif)

1. Create the overlay.

   ```
   dd if=/dev/zero of=overlay.img bs=1M count=500 && \
       mkfs.ext3 overlay.img
   ```

2. Build the Singularity image.

   ```
   singularity build nvidia-pytorch-21_09-py3.sif docker://nvcr.io/nvidia/pytorch:21.09-py3
   ```

3. Add the overlay to the SIF image.

   ```
   singularity sif add --datatype 4 --partfs 2 --parttype 4 --partarch 2 --groupid \
   1 nvidia-pytorch-21_09-py3.sif overlay.img
   ```

4. Install Horovod.

   Start the Singularity container on UAN.

   ```
   singularity shell --nv --bind /lus/shasta/pytorch-gpu:/lus/shasta/pytorch-gpu \
   --writable nvidia-pytorch-21_09-py3.sif
   ```

   Set environment variables for Horovod and install Horovod using `pip install`.

   ```
   Singularity>
   HOROVOD_WITHOUT_TENSORFLOW=1
   HOROVOD_WITH_PYTORCH=1
   pip install --no-cache-dir horovod[pytorch]

   Collecting horovod[pytorch]
   Downloading horovod-0.23.0.tar.gz (3.4 MB)
   |                 | 3.4 MB 4.0 MB/s
   Collecting cloudpickle
   ......
   ```

   Exit the container. `bash  Singularity> exit  exit`

4.3.3.2 Option Two: With a Singularity Definition File

1. Setup the `.def` file.

   Copy the following code to a `.def` file. This example uses `nvidia-pytorch.def`.

   ```
   Bootstrap: docker
   From: nvcr.io/nvidia/pytorch:21.09-py3

   %post
   export PATH=/opt/conda/bin:$PATH
   HOROVOD_WITHOUT_TENSORFLOW=1
   HOROVOD_WITH_PYTORCH=1
   pip install --no-cache-dir horovod[pytorch]
   ```

2. Build the Singularity image.

   This requires root permissions.

   ```
   singularity build nvidia-pytorch-container-21_09-py3.sif nvidia-pytorch.def
   ```

4.3.4 Test The Singularity Image

Execute the following procedure on a UAN (CPU) node first, then switch to a GPU node and repeat the procedure from the beginning.

Some tests should be executed only on GPU nodes. These tests are marked with **GPU nodes only:** and should be skipped when testing on a CPU node.

4.3.4.1 Testing Procedure

1. Start the Singularity container.

   NOTE: if the Singularity image was built with a `.def` file, remove `--writable` from the following command.

   ```
   singularity shell --nv --bind /lus/shasta/pytorch-gpu:/lus/shasta/pytorch-gpu \
   --writable nvidia-pytorch-21_09-py3.sif
   ```

2. Download the mnist script.

   NOTE: this step may be skipped if the mnist script was previously downloaded.

   Ensure that the bind mount directory, `/lus/shasta/pytorch-gpu`, exists.

   Navigate to Horovod's github in a browser.

   Download the mnist script, `pytorch_mnist.py`, and place it into the bind mount directory

3. Verify Horovod is installed.

   1. Start a Python interpreter.

      ```
      Singularity> python
      Python 3.8.10 | packaged by conda-forge | (default, May 11 2021, 07:01:05)
      [GCC 9.3.0] on linux
      Type "help", "copyright", "credits" or "license" for more information.
      >>>
      ```

   2. Verify that Horovod is installed.

      ```
      >>> import torch
      >>> import horovod.torch as hvd
      >>> hvd.init()
      ```

   3. Quit the interpreter.

      ```
      >>> quit()
      Singularity>
      ```

4. **GPU nodes only:** verify that the Singularity image has GPU support.

   ```
   Singularity> nvidia-smi

   Wed Oct 20 01:46:59 2021
   +-----------------------------------------------------------------------------+
   | NVIDIA-SMI 450.80.02    Driver Version: 450.80.02    CUDA Version: 11.4     |
   |-------------------------------+----------------------+----------------------+
   | GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
   | Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
   |                               |                      |               MIG M. |
   |===============================+======================+======================|
   |   0  A100-SXM4-40GB      On   | 00000000:03:00.0 Off |                  Off |
   | N/A   26C    P0    46W / 400W |      0MiB / 40537MiB |      0%   E. Process |
   |                               |                      |             Disabled |
   +-------------------------------+----------------------+----------------------+
   |   1  A100-SXM4-40GB      On   | 00000000:41:00.0 Off |                  Off |
   | N/A   25C    P0    47W / 400W |      0MiB / 40537MiB |      0%   E. Process |
   |                               |                      |             Disabled |
   +-------------------------------+----------------------+----------------------+
   ```

```
|   2  A100-SXM4-40GB      On   | 00000000:82:00.0 Off |                    Off |
| N/A   26C    P0    49W / 400W |      0MiB / 40537MiB |      0%   E. Process |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
|   3  A100-SXM4-40GB      On   | 00000000:C1:00.0 Off |                    Off |
| N/A   25C    P0    48W / 400W |      0MiB / 40537MiB |      0%   E. Process |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

5. **GPU nodes only:** verify that PyTorch and Cuda are installed.

   1. Start a Python interpreter.

      ```
      Singularity> python
      Python 3.8.10 | packaged by conda-forge | (default, May 11 2021, 07:01:05)
      [GCC 9.3.0] on linux
      Type "help", "copyright", "credits" or "license" for more information.
      >>>
      ```

   2. Check for PyTorch and Cuda.

      ```
      >>> import torch
      >>> torch.__version__
      '1.10.0a0+3fd9dcf'
      >>> torch.cuda.is_available()
      True
      >>> torch.cuda.current_device()
      0
      >>> torch.cuda.device(0)
      <torch.cuda.device object at 0x7fb345e631f0>
      >>> torch.cuda.get_device_name(0)
      'A100-SXM4-40GB'
      ```

   3. Quit the interpreter.

      ```
      >>> quit()
      Singularity>
      ```

6. Verify that Pytorch uses Horovod.

   1. Run the mnist script, `pytorch_mnist.py`.

      On air-gapped systems, the script requires a local dataset. The path to that dataset must be passed as an argument upon launch. The dataset must be downloaded on a system with Internet access and then copied to the air-gapped system's mounted directory – in this example, `/lus/shasta/pytorch-gpu`.

      Execute the **air-gapped system command** on air-gapped systems. Otherwise, execute the **standard system command**.

      These examples both assume that the script is located in the user's working directory.

      Air-gapped system command. Replace `LOCATION_OF_MNIST_DATA` with the name of the directory that contains the mnist dataset.

      ```
      Singularity> python pytorch_mnist.py --epoch=1 --data-dir LOCATION_OF_MNIST_DATA
      ```

      Standard system command.

      ```
      Singularity> python pytorch_mnist.py --epoch=1
      ```

The following text is an example of what the mnist script will output during successful execution.

```
mnist.py:57: UserWarning: Implicit dimension choice for log_softmax has been
deprecated. Change the call to include dim=X as an argument.
return F.log_softmax(x)
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.319931
Train Epoch: 1 [640/60000 (1%)] Loss: 2.322116
Train Epoch: 1 [1280/60000 (2%)]    Loss: 2.313180
...
Train Epoch: 1 [58880/60000 (98%)]  Loss: 0.471603
Train Epoch: 1 [59520/60000 (99%)]  Loss: 0.314268
/opt/conda/lib/python3.8/site-packages/torch/nn/_reduction.py:42: UserWarning:
size_average and reduce args will be deprecated, please use reduction='sum' instead.
warnings.warn(warning.format(ret))
mnist.py:81: UserWarning: To copy construct from a tensor, it is recommended to
use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires_grad_(True),
rather than torch.tensor(sourceTensor).
tensor = torch.tensor(val)

Test set: Average loss: 0.2143, Accuracy: 93.65%
```

2. Check that OpenMPI is installed.

```
Singularity cge-ompi-v4.simg:~/cge/cge-ompi> mpirun --version
mpirun (Open MPI) 4.1.0

Report bugs to http://www.open-mpi.org/community/help/
```

7. Exit the container.

```
Singularity> exit
exit
```

4.4 Build a Singularity Dask Image

This section describes how to build a Singularity Dask image. A detailed discussion of the uses cases for Singularity Dask images is beyond the scope of this document.

4.4.1 Prerequisites

- EX systems require internet access for image building operations, including other install operations, such as `docker pull`.

4.4.2 Notes on the Procedures

- The Docker image used to build Singularity `dask` image can be found in the GitHub Container Registry (GHCR).
- The ports used in the example are the default port used by Dask Scheduler (8786) and Jupyter Server (8888).

4.4.3 Build the Singularity Image

A Singularity image may be built with `singularity build` command.

To equip an air-gapped system with a Singularity Spark image, follow the build procedure on a machine with internet access and then copy the resulting image to the air-gapped machine.

4.4.3.1 Build the Singularity Dask Image from the Command Line

```
singularity build dask.sif docker://ghcr.io/dask/dask
```

4.4.3.2 Build the Singularity Dask Notebook Image from the Command Line

```
singularity build dask-notebook.sif docker://ghcr.io/dask/dask-notebook
```

4.4.3.3 Starting a Dask Cluster for Distributed Computing

1. Start the Dask Scheduler.

   ```
   singularity exec dask.sif /usr/bin/prepare.sh dask-scheduler
   ```

2. Start the Dask Workers.

   ```
   singularity exec dask.sif /usr/bin/prepare.sh /usr/bin/prepare.sh \
   dask-worker localhost:8786
   ```

   For testing purposes, all Dask containers (Dask schedulers and Dask workers) may be run on the same node, and many workers may be run on the same node. However, in production, schedulers and workers should be run on separate nodes, and workers should not be run on nodes with other workers.

   To run a Dask container on a non-local node, replace `localhost` in the previous example with address of the node where the `dask-scheduler` is running.

   To run a second worker on a node that is already running a worker, use the same command that was used for the first worker, as in the following example:

   ```
   singularity exec dask.sif /usr/bin/prepare.sh /usr/bin/prepare.sh dask-worker \
   localhost:8786
   ```

3. Start a Jupyter Server.

   ```
   singularity exec dask-notebook.sif /usr/bin/prepare.sh
   ```

4. From the laptop run the following port forwarding command to access the Jupyter Notebook.

   ```
   ssh -L 8888:localhost:8888 <user@remote-host>
   ```

   The `remote-host` is the host where the Jupyter Server is running.

5. Open a browser window and copy a command similar to the following in the address bar. The `token` is the token displayed in the terminal window, when Jupyter Server is started.

   http://localhost:8888/lab?token=

6. In the browser window open a Python Notebook. Copy the following command and run it.

   ```
   from dask.distributed import Client
   client = Client("scheduler:8786")
   client
   ```

   In browser window we can see the Client, Scheduler and Worker Information. If there are several workers we should see the details of all workers.

4.5 Build a Singularity Miniconda Image

This section describes how to build Singularity Miniconda image. A detailed discussion of the uses cases for Miniconda images is beyond the scope of this document.

4.5.1 Prerequisites

- Root permissions are required for building the Singularity image from Singularity definition file.
- Execute permission is required for `miniconda-<version>.sif`.
- EX systems require internet access for image building operations, including other install operations, such as `docker pull`.

4.5.2 Notes on the Procedures

- The Docker image used to build Singularity Miniconda image can be found in Docker Hub.

4.5.3 Build the Singularity Image

Use the `singularity build` command to build a Singularity image.

To equip an air-gapped system with a Singularity Miniconda image, follow the build procedure on a machine with internet access, and then copy the resulting image to the air-gapped machine.

Options 2 and 3 demonstrate how to install additional packages to the base image using Jupyter Notebook as an example. The same procedures would work for other packages.

4.5.3.1 Option One: Command Line

Execute `singularity build miniconda-<version>.sif docker://conda/miniconda3:<version>`, substituting the Miniconda version. The following example shows the command with a version substitution.

```
singularity build miniconda-latest.sif docker://conda/miniconda3:latest
```

4.5.3.2 Option Two: Singularity Definition File

1. Setup the `.def` file.

   Copy the following code to a `.def` file. This example uses `miniconda.def`.

   ```
   Bootstrap: docker
   From: conda/miniconda3:latest

   %post
   export PATH=/opt/conda/bin:$PATH
   conda install jupyter -y --quiet && mkdir -p /opt/notebooks
   ```

2. Build the Singularity image.

   This requires root permissions.

   ```
   singularity build miniconda-latest.sif miniconda.def
   ```

3. Start the Jupyter Notebook.

   ```
   singularity exec miniconda-latest.sif jupyter notebook --notebook-dir=/opt/notebooks \
   --ip='*' --port=8888 --no-browser --allow-root
   ```

4. Verify the terminal output. It should be similar to the following.

   ```
   [W 15:17:53.466 NotebookApp] WARNING: The notebook server is listening on all
   IP addresses and not using encryption. This is not recommended.
   [I 15:17:53.470 NotebookApp] Serving notebooks from local directory: /opt/notebooks
   [I 15:17:53.470 NotebookApp] Jupyter Notebook 6.4.8 is running at:
   [I 15:17:53.470 NotebookApp]
   http://osprey.us.cray.com:8888/?token=6b2598d07b7bfe587f0a9112fca3c8d0294303d466cf7d01
   [I 15:17:53.470 NotebookApp]
   or http://127.0.0.1:8888/?token=6b2598d07b7bfe587f0a9112fca3c8d0294303d466cf7d01
   [I 15:17:53.470 NotebookApp] Use Control-C to stop this server and shut down
   all kernels (twice to skip confirmation).
   [C 15:17:53.478 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///home/users/mandalpa/.local/share/jupyter/runtime/nbserver-135967-open.html
    Or copy and paste one of these URLs:
        http://osprey.us.cray.com:8888/?token=6b2598d07b7bfe587f0a9112fca3c8d0294303d466cf7d01
     or http://127.0.0.1:8888/?token=6b2598d07b7bfe587f0a9112fca3c8d0294303d466cf7d01
   ```

4.5.3.3 Option Three: Singularity Definition File with a Run script

1. Setup the `.def` file.

   Copy the following code to a `.def` file. This example uses `miniconda.def`.

```
Bootstrap: docker
From: conda/miniconda3:latest

%post
export PATH=/opt/conda/bin:$PATH
conda install jupyter -y --quiet && mkdir -p /opt/notebooks

%runscript
jupyter notebook \
--notebook-dir=/opt/notebooks --ip='*' --port=8888 \
--no-browser --allow-root
```

2. Build the Singularity image.

   This requires root permissions.

   `singularity build miniconda-latest.sif miniconda.def`

3. Start the Jupyter Notebook.

   `singularity run miniconda-latest.sif`

4. Verify the terminal output. Compare it to the output in Option two.

4.6 Build a Singularity Anaconda Image

This section describes how to build Singularity Anaconda image. A detailed discussion of the uses cases for Anacondas image is beyond the scope of this document.

4.6.1 Prerequisites

- Root permissions are required for building the Singularity image from the Singularity definition file.
- EX systems require internet access for image building operations, including other install operations, such as `docker pull`.
- The following procedures require execute permissions for `anaconda-<version>.sif`.

4.6.2 Notes on the Procedures

- The Docker image used to build a Singularity Anaconda image can be found in Docker Hub.

4.6.3 Build the Singularity Image

Use the `singularity build` command to build a Singularity image.

To equip an air-gapped system with a Singularity Spark image, follow the build procedure on a machine with internet access, and then copy the resulting image to the air-gapped machine.

Options 2 and 3 demonstrate how to install additional packages to the base image using Jupyter Notebook as an example. The same procedures would work for other packages.

4.6.3.1 Option One: Command Line

Execute `singularity build anaconda-<version>.sif docker://apache/spark:<version>`, substituting the Anaconda version. The following example shows the command with a version substitution.

`singularity build anaconda-latest.sif docker://continuumio/anaconda3:latest`

4.6.3.2 Option Two: Singularity Definition File

1. Setup the `.def` file.

   Copy the following code to a `.def` file. This example uses `anaconda.def`.

```
Bootstrap: docker
From: continuumio/anaconda3:latest

%post
export PATH=/opt/conda/bin:$PATH
conda install jupyter -y --quiet && mkdir -p /opt/notebooks
```

2. Build the Singularity image.

   This requires root permissions.

   ```
   singularity build anaconda-latest.sif anaconda.def
   ```

3. Start the Jupyter Notebook.

   ```
   singularity exec anaconda-latest.sif jupyter notebook --notebook-dir=/opt/notebooks \
   --ip='*' --port=8888 --no-browser --allow-root
   ```

4. Verify the terminal output. It should be similar to the following.

   ```
   [I 14:25:35.291 NotebookApp] Writing notebook server cookie secret to
   /home/users/<username>/.local/share/jupyter/runtime/notebook_cookie_secret
   [W 14:25:35.635 NotebookApp] WARNING: The notebook server is listening on all
   IP addresses and not using encryption. This is not recommended.
   [W 2022-04-02 14:25:35.837 LabApp] 'notebook_dir' has moved from NotebookApp
   to ServerApp. This config will be passed to ServerApp. Be sure to update your
   config before our next release.
   [W 2022-04-02 14:25:35.837 LabApp] 'ip' has moved from NotebookApp to ServerApp.
   This config will be passed to ServerApp. Be sure to update your config before
   our next release.
   [W 2022-04-02 14:25:35.837 LabApp] 'port' has moved from NotebookApp to ServerApp.
   This config will be passed to ServerApp. Be sure to update your config before our
   next release.
   [W 2022-04-02 14:25:35.837 LabApp] 'port' has moved from NotebookApp to ServerApp.
   This config will be passed to ServerApp. Be sure to update your config before our
   next release.
   [W 2022-04-02 14:25:35.837 LabApp] 'allow_root' has moved from NotebookApp to
   ServerApp. This config will be passed to ServerApp. Be sure to update your
   config before our next release.
   [W 2022-04-02 14:25:35.837 LabApp] 'allow_root' has moved from NotebookApp
   to ServerApp. This config will be passed to ServerApp. Be sure to update your
   config before our next release.
   [I 2022-04-02 14:25:35.845 LabApp] JupyterLab extension loaded from
   /opt/conda/lib/python3.9/site-packages/jupyterlab
   [I 2022-04-02 14:25:35.845 LabApp] JupyterLab application directory is
   /opt/conda/share/jupyter/lab
   [I 14:25:35.851 NotebookApp] Serving notebooks from local directory: /opt/notebooks
   [I 14:25:35.851 NotebookApp] Jupyter Notebook 6.4.5 is running at:
   [I 14:25:35.851 NotebookApp]
   http://osprey.us.cray.com:8888/?token=aa183af5436c7767bacaea7d14879da63cd96e3e956eecf9
   [I 14:25:35.852 NotebookApp]
   or http://127.0.0.1:8888/?token=aa183af5436c7767bacaea7d14879da63cd96e3e956eecf9
   [I 14:25:35.852 NotebookApp] Use Control-C to stop this server and shut down
   all kernels (twice to skip confirmation).
   [C 14:25:35.859 NotebookApp]

   To access the notebook, open this file in a browser:
       file:///home/users/<user name>/.local/share/jupyter/runtime/nbserver-115794-open.html
   Or copy and paste one of these URLs:
       http://osprey.us.cray.com:8888/?token=aa183af5436c7767bacaea7d14879da63cd96e3e956eecf9
    or http://127.0.0.1:8888/?token=aa183af5436c7767bacaea7d14879da63cd96e3e956eecf9
   ```

```
^C[I 14:25:46.487 NotebookApp] interrupted
Serving notebooks from local directory: /opt/notebooks
0 active kernels
Jupyter Notebook 6.4.5 is running at:
http://osprey.us.cray.com:8888/?token=aa183af5436c7767bacaea7d14879da63cd96e3e956eecf9
or http://127.0.0.1:8888/?token=aa183af5436c7767bacaea7d14879da63cd96e3e956eecf9
```

4.6.3.3 Option Three: Singularity Definition File with Run script

1. Setup the `.def` file.

   ```
   Bootstrap: docker
   From: continuumio/anaconda3:latest

   %post
   export PATH=/opt/conda/bin:$PATH
   conda install jupyter -y --quiet && mkdir -p /opt/notebooks

   %runscript
   jupyter notebook \
   --notebook-dir=/opt/notebooks --ip='*' --port=8888 \
   --no-browser --allow-root
   ```

2. Build the Singularity image.

   *__Note:__ This requires root permissions.

   ```
   singularity build anaconda-latest.sif anaconda.def
   ```

3. Start the Jupyter Notebook.

   ```
   singularity run anaconda-latest.sif
   ```

4. Verify the terminal output. Compare it to the output in Option two.

4.7 Build a Singularity Spark Image

This section describes how to build a Singularity Spark image. For examples of advanced Spark image use, refer to the official Spark documentation.

4.7.1 Prerequisites

EX systems require internet access for image building operations, including other install operations, such as `docker pull`.

4.7.2 Notes on The Procedures

- The docker image used to build singularity `Spark`, `Pyspark`, `Spark-R` image can be found in Docker Hub.

4.7.3 Build the Singularity Image

Use the `singularity build` command to build a Singularity image.

To equip an air-gapped system with a Singularity Spark image, follow the build procedure on a machine with internet access, and then copy the resulting image to the air-gapped machine.

4.7.3.1 Build the Singularity Spark Image

Execute `singularity build spark-<version>.sif docker://apache/spark:<version>`, substituting the Spark version. The following example shows the command with a substituted version number.

```
singularity build spark-v3.2.1.sif docker://apache/spark:v3.2.1
```

Spark shell could be used as follows.

```
singularity shell spark-v3.21.sif
```

From inside the container execute the command.

```
spark-shell
```

This will start a spark-shell with a scala prompt where commands can be entered, to run spark programs in scala.

4.7.3.2 Build the Singularity PySpark Image

Execute `singularity build pyspark-<version>.sif docker://godatadriven/pyspark:<version>`, substituting the Pyspark version. The following example shows the command with a substituted version number.

```
singularity build pyspark-latest.sif docker://godatadriven/pyspark:latest
```

The following steps show how to start a `pyspark`shell:

1. Start a Singularity container.

   ```
   singularity shell pyspark-latest.sif
   ```

2. Start the `pyspark` shell.

   Execute the following command from inside the container:

   “‘bash pyspark

4.7.3.3 Build the Singularity SparkR Image

Execute `singularity build sparkr-<version>.sif docker://beniyama/sparkr-docker:<version>`, substituting the correct SparkR version. The following example shows the command with a substituted version number.

```
singularity build sparkr-latest.sif docker://beniyama/sparkr-docker:latest
```

The following steps show how to start a SparkR user interface.

1. Start a Singularity container.

   ```
   singularity shell sparkr-latest.sif
   ```

2. Start an R Studio Server.

   Execute the following command to start he server, which will run on port 8787.

   ```
   /usr/lib/rstudio-server/bin/rserver --server-daemonize 0
   ```

To access the UI for the server that was started in the previous steps, execute the following steps.

1. On another machine, start a bash terminal and run the following command for ssh port forwarding.

   ```
   ssh -L 8787:localhost:8787 user@remote-host
   ```

   The user is the user that started the Singularity SparkR container on the remote host. Note that `remote-host` should be replaced by the fully qualified hostname or IP Address of the host where the container is running.

2. Access the server UI.

   After the port forwarding command completes successfully, open a browser window and enter the following url in the address bar: `http://localhost:8787`. This will start an `R-Studio` terminal inside the browser window. SparkR programs can be run from the terminal.

---

5 Analytics User Operations

5.1 Spark MKL Tests

These tests verify that Spark is configured to use the Math Kernel Library (MKL). If Spark does not use MKL, Spark’s operations may still execute successfully, but with the less effiecient, default math library.

These tests may be executed directly on a host machine or in a container image running on a host machine.

5.1.1 Notes on the Tests

- These tests do **NOT** explain how to configure a container or start an image.
- These tests must be executed in the `spark-shell`, and Spark's version should be `3.1.2` or higher.

5.1.2 Tests

- Verify that MKL is enabled for Spark.

  ```
  scala> import com.github.fommil.netlib.BLAS;
  scale> System.out.println(BLAS.getInstance().getClass().getName());
  ```

  If the output is `com.github.fommil.netlib.NativeSystemBLAS`, MKL is enabled. Otherwise, warnings will be printed:

  ```
  WARN BLAS: Failed to load implementation from:com.github.fommil.netlib.NativeSystemBLAS
  WARN BLAS: Failed to load implementation from:com.github.fommil.netlib.NativeRefBLAS
  ```

5.2 Pytorch MKL Tests

These tests verify that PyTorch is configured to use the Math Kernel Library (MKL). If PyTorch does not use MKL, PyTorch's operations may still execute successfully, but with the less effiecient, default math library.

These tests may be executed directly on a host machine or in a container image running on a host machine.

5.2.1 Notes on the Tests

- These tests do **NOT** explain how to configure a container or start an image.
- Code examples are written to be copied and pasted into a text file. However, these tests may be executed in the interactive Python interpreter by sequentially executing each line from their code examples.
- If PyTorch is not installed, importing it will throw an exception: `python ModuleNotFoundError: No module named 'torch'`

5.2.2 Tests

- Verify that PyTorch was built with `MKL`.

  ```
  import torch
  torch_config = torch.__config__.show().replace(","," ")
  print(torch_config.lower().find("intel(r) math kernel library version") != -1)
  ```

  If the output is `true`, PyTorch was built with `MKL`. If the output is `false`, check for a newer version of `MKL`:

  ```
  import torch
  torch_config = torch.__config__.show().replace(","," ")
  print(torch_config.lower().find("intel(r) oneapi math kernel") != -1)
  ```

  If the output is `true`, PyTorch was built with `MKL`

- Verify that PyTorch uses `MKL`.

  ```
  import os
  os.environ['MKL_VERBOSE'] = '1'
  import torch
  print(torch.inverse(torch.rand(4, 4)))
  ```

  If the output contains `MKL_VERBOSE Intel(R) MKL`, PyTorch called `MKL`.

- Verify that PyTorch was built with `MKL-DNN`.

  NOTE: It it possible that PyTorch was built with MKL, but not with MKL-DNN.

  ```
  import os
  os.environ['MKLDNN_VERBOSE'] = '1'
  import torch
  print(torch._C.has_mkldnn)
  ```

If the output is True, PyTorch was built with MKL-DNN.

- Verify that PyTorch uses MKL-DNN.

```
import os
os.environ['MKLDNN_VERBOSE'] = '1'
import torch
print(torch.randn(10).to_mkldnn())
```

If the output contains mkldnn_verbose,info,Intel MKL-DNN, or dnnl_verbose,info,oneDNN (a newer version of MKL), PyTorch called MKL-DNN.

- Verify that torch.utils uses MKL-DNN.

```
import os
os.environ['MKLDNN_VERBOSE'] = '1'
import torch
from torch.utils import mkldnn as mkldnn_utils
print(mkldnn_utils.to_mkldnn(torch.nn.Linear(1000, 2)))
```

If the output contains mkldnn_verbose,info,Intel MKL-DNN or dnnl_verbose,info,oneDNN (a newer version of MKL), PyTorch called MKL-DNN.

5.3 TensorFlow MKL Tests

These tests verify that TensorFlow is configured to use the Math Kernel Library (MKL). If TensorFlow does not use MKL, TensorFlow's operations may still execute successfully, but with the less effiecient, default math library.

These tests may be executed directly on a host machine or in a container image running on a host machine.

5.3.1 Notes on the Tests

- These tests do **NOT** explain how to configure a container or start an image.
- Code examples are written to be copied and pasted into a text file. However, these tests may be executed in the interactive Python interpreter by sequentially executing each line from their code examples.
- If TensorFlow is not installed, importing it will throw an exception: python ModuleNotFoundError: No module named 'tensorflow'
- These tests differ slightly for versions 1 and 2 of TensorFlow. Both versions are provided for each test.

5.3.2 Tests

5.3.2.1 Get the version of TensorFlow

```
import tensorflow as tf
print("TensorFlow Version: {}".format(tf.__version__))
```

5.3.2.2 Verify that MKL is enabled

- TensorFlow Version 1

```
from tensorflow.python import pywrap_tensorflow
print(pywrap_tensorflow.IsMklEnabled())
```

If the output is True, then MKL is enabled.

- TensorFlow Version 2

```
from tensorflow.python import _pywrap_util_port
print(_pywrap_util_port.IsMklEnabled())
```

If the output is True, then MKL is enabled.

5.3.2.3 Verify that TensorFlow uses MKL

- TensorFlow Version 1

```
import os
os.environ['MKL_VERBOSE'] = '1'
import tensorflow as tf
x = tf.random_normal([10, 10])
y = tf.random_normal([10, 10])
print(tf.matmul(x, y))

# Run the graph with full trace option
with tf.Session() as sess:
    run_options = tf.RunOptions(trace_level=tf.RunOptions.FULL_TRACE)
    run_metadata = tf.RunMetadata()
    sess.run(res, options=run_options, run_metadata=run_metadata)
```

  If the output contains `MKL_VERBOSE Intel(R) MKL`, then `MKL` was called.

- TensorFlow Version 2

```
import os
os.environ['MKL_VERBOSE'] = '1'
import tensorflow as tf
x = tf.random.normal([10, 10])
y = tf.random.normal([10, 10])
print(tf.matmul(x, y))
```

  If the output contains `MKL_VERBOSE Intel(R) MKL`, or `oneapi Deep Neural Network Library (oneDNN)` (a newer version of `MKL`), `MKL` was called.

5.3.2.4 Verify that MKL-DNN is enabled

To verify that `MKL-DNN` is enabled, navigate to the TensorFlow documentation and run the mnist example code. If the output contains `mkldnn_verbose,info,Intel MKL-DNN` or `dnnl_verbose,info,oneDNN`, then `MKL-DNN` was called.

NOTE: To ensure that the Shasta is able to access the mnist data, download it from the TensorFlow repo. Then, change the local mnist code to point to the downloaded dataset before executing the test.

6 Glossary

Glossary of terms used in HPE Cray EX documentation.

6.0.1 Customer Access Network

The Customer Access Network (CAN) provides access from outside the customer network to services, noncompute nodes (NCNs), and User Access Nodes (UANs) in the system. This allows for the following.

1. Clients outside of the system:
    - Log in to each of the NCNs and UANs.
    - Access web UIs within the system (e.g. Prometheus, Grafana, and more).
    - Access the Rest APIs within the system.
    - Access a DNS server within the system for resolution of names for the webUI and REST API services.
    - Run Cray CLI commands from outside the system.
    - Access the User Access Instances (UAI).
2. NCNs and UANs to access systems outside the cluster (e.g. LDAP, license servers, and more).
3. Services within the cluster to access systems outside the cluster.

These nodes and services need an IP address that routes to the customer's network in order to be accessed from outside the network.

6.0.2 Compute Node (CN)

The compute node (CN) is where high performance computing application are run. These have hostnames that are of the form "nidXXXXXX", that is, "nid" followed by six digits. where the XXXXXX is a six digit number starting with zero padding.

6.0.3 Cray System Management (CSM)

Cray System Management (CSM) refers to the product stream which provides the infrastructure to manage a Cray EX system using Kubernetes to manage the containerized workload of layered microservices with well-defined REST APIs which provide the ability to discover and control the hardware platform, manage configuration of the system, configure the network, boot nodes, gather log and telemetry data, connect API access and user level access to Identity Providers (IdPs), and provide a method for system administrators and end-users to access the Cray EX system.

6.0.4 High Speed Network (HSN)

The High Speed Network (HSN) in an HPE Cray EX system is based on the Slingshot switches.

6.0.5 Management Nodes

The management nodes are one grouping of NCNs. The management nodes include the master nodes with hostnames of the form of ncn-mXXX, the worker nodes with hostnames of the form ncn-wXXX, and utility storage nodes, with hostnames of the form ncn-sXXX, where the XXX is a three digit number starting with zero padding. The utility storage nodes provide Ceph storage for use by the management nodes. The master nodes provide Kubernetes master functions and have the etcd cluster which provides a datastore for Kubernetes. The worker nodes provide Kubernetes worker functions where most of the containerized workload is scheduled by Kubernetes.

6.0.6 Node Management Network

The node management network (NMN) communicates with motherboard PCH-style hosts, typically 10GbE Ethernet LAN-on-motherboard (LOM) interfaces. This network supports node boot protocols (DHCP/TFTP/HTTP), in-band telemetry and event exchange, and general access to management REST APIs.

6.0.7 Non-Compute Node (NCN)

Any node which is not a compute node may be called a Non-Compute Node (NCN). The NCNs include management nodes and application nodes.

6.0.8 UAI

The User Access Instance (UAI) is a lightweight, disposable platform that runs under Kubernetes orchestraion on worker nodes. The UAI provides a single user containerized environment for users on a Cray Ex system to develop, build, and execute their applications on the Cray EX compute node. See UAN for another way for users to gain access.

6.0.9 UAN

The User Access Node (UAN) is an NCN, but is really one of the special types of Application nodes. The UAN provides a traditional multi-user Linux environment for users on a Cray Ex system to develop, build, and execute their applications on the Cray EX compute node. See UAI for another way for users to gain access. Some sites refer to their UANs as Login nodes.

6.0.10 xname

Component names (xnames) identify the geolocation for hardware components in the HPE Cray EX system. Every component is uniquely identified by these component names. Some, like the system cabinet number or the CDU number, can be changed by site needs. There is no geolocation encoded within the cabinet number, such as an X-Y coordinate system to relate to the floor layout of the cabinets. Other component names refer to the location within a cabinet and go down to the port on a card or switch or the socket holding a processor or a memory DIMM location. Refer to “Component Names (xnames)” in the ***HPE Cray EX Hardware Management Administration Guide 1.5 S-8015***.