TrinityX Documentation

Release 12.0

ClusterVision Solutions BV

September 07, 2021

List of Tables **iii**

1 Introduction **1**

2 Documentation **3**
2.1 Installation manual . . . 3
2.2 Administration manual . . . 26
2.3 User manual . . . 37
2.4 Docker documentation . . . 40

Index **45**

Table 2.1: Global variables 13
Table 2.2: Global variables (cont'd) 15

CHAPTER 1

Introduction

Welcome to the TrinityX documentation!

TrinityX is the new generation of ClusterVision's open-source HPC platform. It is designed from the ground up to provide all services required in a modern HPC system, and to allow full customization of the installation. Also it includes optional modules for specific needs, such as an OpenStack cloud and Docker on the compute nodes.

A standard TrinityX installation includes many software components, such as Slurm, MariaDB, OpenLDAP, or even CentOS, the operating system on which everything runs. All of those pieces of software are documented separately and their documentation can be accessed freely on the internet. In order to avoid duplicating existing documentation, the TrinityX documentation will only include chapters about external software when required, for example to describe a non-standard setup. For everything else, please refer to the original documentation of each piece of software.

CHAPTER 2

Documentation

Note Each TrinityX installation is adapted to the specific requirements of the customer, and the list of installed software varies.

The documentation is split into various chapters, each of which dealing with a specific topic. They are grouped into three manuals:

- the *Installation manual* on how to deploy TrinityX on your system;
- the *Administration manual* for system administrators managing a production TrinityX system;
- and the *User manual* for cluster users.

Although each manual is destined to a specific audience, there may be hyperlinks from one manual to another when more advanced reading is suggested.

2.1 Installation manual

2.1.1 TrinityX pre-installation requirements

Hardware

TrinityX has very few hardware requirements, summarized in the following paragraphs.

Controllers

The machines that will be used as TrinityX controllers are expected to have a minimum of 2 Ethernet NIC:

- one NIC for the public interface, used to connect to the cluster;
- one NIC for the private (internal) interface, used for compute node provisioning and general cluster communications.

Note The out-of-band management network can be integrated on the internal network, or be accessible on it's own network. Depending on the design of your cluster, this could add another NIC.

Note A third NIC that serves as a dedicated heartbeat link between a pair of HA controllers would provide a redundant communication link which would go a long way in reducing the risk of encountering a split-brain situation.

Note When installing the controllers in a high availability configuration it is required to have an IPMI

interface properly configured on both controllers. This allows for IPMI based fencing functionality to be implemented for safer failovers.

The OpenStack optional module adds another network requirement:

- a third Ethernet NIC used for the traffic between OpenStack VMs.

The controllers must have enough disk space to install the base operating system, as well as all the packages required by TrinityX. For a simple installation this amounts to a few gigabytes only. Other components of Trinity will likely require much more space, namely:

- compute images;
- shared applications;
- user homes.

All of the above are located in specific directories under the root of the TrinityX installation, and can be hosted either on the controller's drives or on remote filesystems. Sufficient storage space must be provided in all cases.

Compute nodes

The machines that will be used as compute nodes are expected to have at least 1 Ethernet NIC:

- one NIC for the private (internal) interface, used for provisioning and general cluster communications.

Note This NIC could also be an InfiniBand interface (boot-over-IB).

The OpenStack optional module adds another network requirement:

- a second Ethernet NIC used for the traffic between OpenStack VMs.

The compute nodes can be provisioned with or without local storage. When not configured for local storage a ramdisk will be used to store the base image. In that case, make sure to take into account the space of the OS image (which depends on its exact configuration) in your memory calculations.

Software and configuration

Controllers

The TrinityX installer requires the operating system of the controllers to be installed already. As of TrinityX release 11, the supported OS versions are:

- CentOS 7 **Minimal**
- Scientific Linux 7 **Minimal**

It is important to install only the Minimal edition, as some of the packages that are installed with larger editions may conflict with what will be installed for TrinityX. Note that when installing from a non-Minimal edition, it is usually possible to select the Minimal setup at the package selection step.

The network configuration of the controllers must be done before installing TrinityX, and it must be correct. This includes:

- IP addresses and netmasks of all interfaces that will be used by TrinityX;

The timezone must also be set correctly before installation as it will be propagated to all subsequently created node images.

If the user homes or the TrinityX installation directory (part of or whole) are to be set up on remote or distributed volume(s) or filesystem(s), all relevant configuration must be done before installing TrinityX. If necessary, remember to disable the NFS and DRBD roles in the `controller.yml` playbook.

On the machine used to install the controllers, the EPEL repository must be enabled and the git and ansible packages must be present.:

```
# yum install epel-release
# yum install git ansible
```

Note The `prepare.sh` script includes these steps.

This machine is usually the controller in non HA configurations or the first of the two controllers in HA configurations.

Also, the following installer dependencies need to be available on that machine:

- OndrejHome.pcs-modules-2 from the ansible galaxy: `ansible-galaxy install OndrejHome.pcs-modules-2`
- luna-ansible:

```
# curl https://updates.clustervision.com/luna/1.2/centos/luna-1.2.repo >
/etc/yum.repos.d/luna-1.2.repo
# yum install luna-ansible
```

Lastly, since Ansible uses ssh to deploy the configuration, the machine running the installer (whether it be one of the controllers or a third one) should have passwordless access to the controller(s)-to-be. As such, care must be taken to put its key in `/root/.ssh/authorized_keys` on the controller(s)-to-be.

Compute nodes

The compute nodes must be configured to PXE-boot from the NIC connected to the internal network.

2.1.2 TrinityX HA design and implementation

Introduction

The TrinityX installation playbooks can set up either of the following configurations: - one controller as a standalone system - a pair of controllers as a High Availability (HA) pair with failover of services between the two

In the stand-alone setup (also called non-HA in the TrinityX documentation), the various services are set up intuitively. The configuration will be similar to what can be achieved by setting up the services by hand, and it should not present any surprise to an experienced systems administrator.

When the `ha` variable in `group_vars/all` is set to `true`, the TrinityX playbook i.e. `controller.yml` will set up an HA controller pair. Please note that the installation playbook will only need to run once on the controller selected to become primary, the other controller will also be set up in parallel.

This section will cover the design and implementation of the HA configuration in TrinityX.

Note In the following paragraphs we will reference the standard configuration. This is the base configuration that is set up by the TrinityX playbook.

Note This document assumes that the reader is already familiar with the storage configuration options set by `shared_fs_type` in the table tab_global_variables.

Corosync and fencing

Corosync is the group communication system used to keep track of the machines present in the cluster. It is used to detect node and communication failure, and passes that information up to Pacemaker

for further action.

Note The term *cluster* used in the context of Corosync means the group of machines known to Corosync, and for which failover is required. It does not mean the whole cluster with compute nodes, storage, etc. Typically a Corosync cluster will include the HA pair (or group of machines that can run the resources), and possibly one or more quorum devices used to detect node failure in order to avoid split brain scenarios.

The Corosync configuration in TrinityX is fairly basic:

- the Corosync cluster includes only the two controllers;
- the cluster is a generic cluster, not a 2-node only cluster;
- quorum is disabled;
- IPMI based fencing is configured in *Pacemaker*;

Note This is not an ideal configuration in itself, as it is difficult if not impossible to reliably handle a split brain situation with only two nodes; setting up a *quorum device* or a third node is recommended for that.

Pacemaker

The management of computing resources is done by Pacemaker (cluster resource manager). Pacemaker relies on Corosync to determine the availability of nodes within an HA cluster (same definition as for Corosync), and follows a set of rules and constraints to determine where to run sets of resources.

The TrinityX standard configuration makes the following assumptions:

- there are only two nodes on which the resources can run;
- the hardware of those nodes is identical (at least as far as the device names presented by the operating system);

The resources defined in Pacemaker are separated into two broad groups, which are called primary and secondary. The node on which the resources of the primary group are running at any given time has the primary role, the other one has the secondary role.

Note Although the terms are identical, the primary role and the primary installation (and their respective secondary counterparts) are different. The *primary installation* is simply the one that happens on the node chosen to have the *primary role*, and during which the configuration of the shared hardware resources is done. The *primary role* is an arbitrary name for the node that is currently running the essential services. After the installation is done, the node with the primary role might change, as a failover can happen and the nodes switch roles.

The resources defined by the TrinityX installer are grouped together in resource groups. Resource groups are:

- colocated: all resources in a given group run on the same node;
- serialized: the resources start in that specific order and stop in reverse order; any failure of a resource prevents the subsequent ones from running.

There are two core groups: `Trinity`, which defines the primary role, and `Trinity-secondary` which defines the secondary role. Two other groups attached to the core `Trinity` group: `Trinity-fs` and `trinity-stack`. One Master/Slave sets: `Trinity-drbd`. And lastly: a clone set `ibmon-clone` and fencing resources `fence-controllerX`.

The exact number of resources defined depends on the storage model chosen by the user.

Resources

The full list of resources that may be created for the TrinityX base HA configuration is the following:

```
01  Resource Group: Trinity
02      primary                         (ocf::pacemaker:Dummy)
03      trinity-ip                      (ocf::heartbeat:IPaddr2)

04  Resource Group: Trinity-secondary
05      secondary                       (ocf::pacemaker:Dummy)

06  Resource Group: Trinity-fs
07      fs (ocf::pacemaker:Dummy)
08      wait-for-device                 (ocf::heartbeat:Delay)
09      trinity-fs                      (ocf::heartbeat:Filesystem)
10      fs-ready                        (ocf::pacemaker:Dummy)

11  Resource Group: Trinity-stack
12      stack                           (ocf::pacemaker:Dummy)
13      named                           (systemd:named)
14      openldap                        (systemd:slapd)
15      mariadb                         (systemd:mariadb)
16      slurmdbd                        (systemd:slurmdbd)
17      slurmctld                       (systemd:slurmctld)
18      nginx                           (systemd:nginx)
19      mongod                          (systemd:mongod)
20      xinetd                          (systemd:xinetd)
21      dhcpd                           (systemd:dhcpd)
22      lweb                            (systemd:lweb)
23      ltorrent                        (systemd:ltorrent)
24      httpd                           (systemd:httpd)
25      snmptrapd                       (systemd:snmptrapd)
26      zabbix-server                   (systemd:zabbix-server)

27  Master/Slave Set: Trinity-drbd [DRBD]

28  Clone Set: ibmon-clone [ibmon]

29  fence-controller1                   (stonith:fence_ipmilan)
30  fence-controller2                   (stonith:fence_ipmilan)
```

Notes:

- The filesystem resources (#08, which is only a delay to make sure that the kernel has caught up with the new device, and #09, which mounts the underlying filesystem) only exist for use cases where a separate filesystem is created for the TrinityX directory tree: `dev` and `drbd`.
- The DRBD master-slave set (#27) is only created when the `drbd` use case is selected. Due to its architecture, DRBD can only be managed through a master-slave resource. That resource includes two instances, the master which will always run on a node, and a slave which will run if another node is available.
- The dummy resources are there for practical reasons. It's not possible to insert a new resource at the very beginning of a group, only at the end or after an existing resource in that group. The dummy resources (which do nothing at all) are there so that other resources can be inserted just after them. This is just as good as being the first one in the group.
- The dummy resource #10 serves as an anchor for resources that require the TrinityX directory tree. With the `dev` and `drbd` use cases, the corresponding shared filesystem resources will be inserted before that one. All resources inserted after this anchor will be able to use the directory tree, regardless of the storage use case.
- The resource group Trinity-stack (#11-26) has monitoring disabled so that a service failing in this group does not trigger a failover or any pacemaker operation.

Constraints

The location and starting order of these resources is managed through Pacemaker constraints.

As mentioned earlier, groups have implicit constraints: they are both colocated and serialized. This allows for an intuitive understanding of what happens inside of each group.

A few additional constraints are defined to locate and order groups between themselves:

```
00  Location Constraints:
01    Resource: Trinity
        Constraint: location-Trinity Rule: score=-INFINITY Expression:
ethmonitor-ib0 ne 1
02    Resource: fence-controller1 Disabled on: controller1 (score:-INFINITY)
03    Resource: fence-controller2 Disabled on: controller2 (score:-INFINITY)

04  Ordering Constraints:
05    start Trinity then start Trinity-secondary (kind:Mandatory)
06    start Trinity then start Trinity-fs (kind:Mandatory)
07    start Trinity-fs then start Trinity-stack (kind:Mandatory)
08    start Trinity then start DRBD-master (kind:Mandatory)
09    start DRBD-master then start Trinity-fs (kind:Mandatory)
10    start Trinity-fs then start Trinity-secondary (kind:Mandatory)
11    promote DRBD-master then start wait-for-device (kind:Mandatory)

12  Colocation Constraints:
13    Trinity-secondary with Trinity (score:-INFINITY)
14    Trinity-fs with Trinity (score:INFINITY)
15    Trinity-stack with Trinity (score:INFINITY)
16    DRBD-master with Trinity (score:INFINITY) (rsc-role:Master)
(with-rsc-role:Started)
```

Notes:

- The two essential constraints, which are always present, are #05 and #13. #05 is a constraint which serializes the two groups. It means that `Trinity-secondary` will only start after `Trinity` has started successfully. As most, if not all, secondary resources depend on services that are started in the primary group, this is again the most intuitive strategy.
- #13 is a colocation constraint, which says that `Trinity-secondary` cannot run on the same node as `Trinity`, and that `Trinity` comes first. In other words: pick a node to run the primary, and if there is another one available, run the secondary on it, otherwise don't run the secondary. This is the rule that allows for failover of the primary resources, and makes sure that primary services are always up.
- #14-16 mean that the primary group serves as an anchor for all other services that must run on the primary controller.
- #11 is there to make sure that the device-related resources (`wait-for-device` and `trinity-fs`) only start after the promotion of the DRBD resource, which is to say, after it becomes master on the local node. This is needed due to the way Pacemaker starts resources and the difference between starting and promoting a resource.
- #02-03 ensure that fencing resources start on opposite nodes for fencing to function properly if the need for it arises.

Databases

In TrinityX HA installs, all databases (OpenLDAP, MariaDB and MongoDB) are managed by pacemaker and are part of the trinity-stack resource group. They all rely on the underlying DRBD replication to ensure that data is being constantly synchronized between the two controllers.

HA-pair management

A fully configured TrinityX HA cluster will automatically perform a failover upon a critical failure.

There are, however, a few scenarios that should be kept in mind when managing the cluster. These include:

- Bringing a failing secondary controller up;
- Bringing the cluster up from a cold state (a state in which both the primary and secondary controllers were down, as in case of a power failure); or,
- Recovering the new secondary node after a successful failover.

Upon a failure of the secondary node or a successful failover, the system administrators should be notified immediately. It will be necessary to either fix the issues on the secondary node in the first case, or to recover the new secondary node in the second case. Otherwise, if these failures remain unhandled, they will interfere with the proper execution of a failover in a case where the primary controller encounters another issue.

As such, the monitoring system should include checks to monitor the state of the HA cluster.

Note TrinityX does not configure pacemaker and corosync to start when a controller starts up. It is left at the discretion of the sysadmin to manually start it up using `pcs cluster start` on the newly booted controller.

Booting the controllers

When booting the cluster from a cold state (all nodes down), special care should be taken in choosing which node will serve as the primary controller.

When booting the cluster, the first resource group that comes up is `Trinity`, which includes the floating IP. Then, pacemaker will try to start `Trinity-drbd`. In cases where the node on which the resources are being started was the previous primary node (before the cold boot), the cluster will continue booting up successfully. If, however, this node had the secondary role before the cold boot, you may encounter a situation in which the node being promoted to the primary role may or may not have the latest state of the cluster. In particular, its DRBD state might be behind that of the node that pacemaker decided to load as secondary.

To avoid such a situation it is crucial that a sysadmin starts the cluster from the node that last had the primary role.

The sysadmin can proceed to boot the cluster by running the following command:

```
pcs cluster start --all
```

Maintenance

During the lifetime of the cluster a sysadmin might need to change configuration files, update packages, or restart services. Doing so may trigger a failover event, which could have negative consequences for the cluster. To avoid such behaviour and temporarily prevent pacemaker from interfering with the state of the cluster, it is essential to activate maintenance mode before applying any changes.

By entering maintenance mode, admins can take full control of the cluster to perform any required operations without worrying about the state of the cluster. Maintenance mode in pacemaker can be enabled by running the following command:

```
pcs property set maintenance-mode=true
```

Maintenance mode should be deactivated once maintenance is complete and the cluster is brought back to its previous state. Maintenance mode is disabled by running the following command:

```
pcs property set maintenance-mode=false
```

Conclusion

With a few carefully chosen resources and constraints, the TrinityX HA configuration reaches all of the design goals earlier specified:

- It is correct (barring bugs in the underlying software), as proven by repetitive testing of failover between controller nodes;
- It is generic, as it doesn't include resources that manage specific types of hardware, yet leaves room and includes documentation for the engineers to add those resources when deploying TrinityX;
- It is as simple and intuitive as possible, with very few constraints and clearly delimited primary and secondary roles. It is also extendable very easily, as there are few existing rules and constraints to be aware of.

When deploying a TrinityX HA pair, what is left for the engineer to do are the hardware-specific tasks:

- Add an external Corosync quorum device;
- If necessary in the `dev` storage use case, add a resource to assemble a RAID array and insert it before `wait-for-device` in the primary `Trinity` group.

2.1.3 TrinityX installation procedure

This document describes the installation of a TrinityX controller, as well as the creation and configuration of an image for the compute nodes of a TrinityX cluster.

Starting with Release 11, Ansible is fully integrated into TrinityX. This allows for a lot of flexibility when installing a cluster.

The requirements for all components of a TrinityX cluster are described in the *TrinityX pre-installation requirements*. It is assumed that the guidelines included in this document have been followed, and that the controller machines are ready for the TrinityX configuration.

Note While the procedure to create compute images must be run from the cluster controller (the primary controller when in HA mode), the installation procedure of the controllers themselves can be executed from any arbitrary machine (including your laptop).

Controller installation

The TrinityX configuration tool will install and configure all packages required to set up a working TrinityX controller.

First, make sure that all pre-requisites are present. This can be easily done by running the `prepare.sh` script.

cd ~/trinityX/site # bash prepare.sh

The configuration for a default controller installation is described in the file `controller.yml`, as well as the files located in the `group_vars/` subdirectory of the TrinityX tree, while the list of machines to which the configuration needs to be applied is described in the file called `hosts`.

Copy the `.yml.example` files to the `.yml` version.

pwd ~/trinityX/site/

ls hosts controller.yml group_vars/ hosts.example controller.yml

group_vars/: all.yml.example

These files can be edited to reflect the user's own installation choices. For a full list of configuration options supported by TrinityX, refer to *TrinityX configuration variables*.

Once the configuration files are ready, the `controller.yml` Ansible playbook can be run to apply the configuration to the controller(s):

```
# pwd
~/trinityX/site/
```

```
# ansible-playbook controller.yml
```

Note By default, high availability is enabled in the installer, so it expects to have access to two machines which will become the controllers. To install a non-HA version, you need to update the `ha` variable in `group_vars/all`. For more details on the high availability configuration you can consult *TrinityX HA design and implementation*.

If more verbose output is desired during the installation process, you can use `ansible-playbook`'s `-v` option. The verbosity level will increase according to the number of `v`. For further details about the use of Ansible, including its command line options, please consult the official Ansible documentation.

controller.yml playbook

When running this playbook for the first time, you will see initial warnings that some luna inventory could not be parsed. Luna is the cluster provisioning tool included in TrinityX. What these warnings mean is that *luna* could not be queried for a list of nodes and osimages. This is normal at this point of the installation, since luna has not been configured yet:

```
[WARNING]:  * Failed to parse /etc/ansible/hosts/luna with script plugin:
Inventory script
(/etc/ansible/hosts/luna) had an execution error: Traceback (most recent call
last):   File
"/etc/ansible/hosts/luna", line 3, in <module>     from luna_ansible.inventory
import LunaInventory
File "/usr/lib/python2.7/site-packages/luna_ansible/inventory.py", line 15, in
<module>     raise
AnsibleError("luna is not installed") ansible.errors.AnsibleError: luna is not
installed

[...]
```

The rest of the output would be a list of all the tasks that Ansible is running on controller(s):

```
[...]

TASK [trinity/init : Update the trix_ctrl_* variables in case of non-HA setup]
*************************
ok: [controller]

TASK [trinity/init : Toggle selinux state]
*********************************************************
 [WARNING]: SELinux state temporarily changed from 'enforcing' to 'permissive'.
State change will take
effect next reboot.

changed: [controller]

[...]

TASK [trinity/repos : Ensure "/trinity/repos" exists]
************************************************
changed: [controller]

[...]
```

Then at the end, if everything was successful, you will be able to see a summary of all the actions that Ansible has performed, including how many changes and how many failures:

```
PLAY RECAP
*********************************************************************************************
controller                 : ok=270  changed=197  unreachable=0    failed=0
```

Keep in mind that if some of the tasks fail during the installation, Ansible won't stop until it finishes running all the other tasks. If this happens, Ansible can be used to only re-apply the failing task, the full role containing it, or the entire playbook, after the cause of the failure has been fixed.

What are the passwords?

By default, the TrinityX installer will generate random passwords for all services that require one. You can find all of the generated passwords on the controller(s) at */etc/trinity/passwords/* where every password lives in its own file that's named after the service that uses it.

Compute node image creation

The creation and configuration of an OS image for the compute nodes uses the same tool and a similar configuration file as for the controller. While the controller configuration applies its setting to the machine on which it runs, the image configuration does so in a directory that will contain the whole image of the compute node.

Note Building a new image isn't required for most system administration tasks. One of the images existing on your system can be cloned and modified. Creating a new image is only useful for an initial installation, or when desiring to start from a clean one. Another scenario might be a cluster where all configuration (creation, deletion, …) must be fully controlled by Ansible - in this case to create the image it is possible to copy `compute.yml` and update `image_name` variable to reflect the new image's name.

The setup of the default image is defined in the playbook `compute.yml`, which controls the creation of a new filesystem directory and applies the image configuration. The `compute.yml` file includes the `trinity-image-create.yml` and `trinity-image-setup.yml` playbooks as dependencies. These are playbooks that apply a standard Trinity image configuration.

In the vast majority of cases, changing the configuration of the default image is not required. It may be desired, however, to set up a custom root password, in which case the variable `image_password` can be set to the desired password.

Creating a new image is as simple as setting up the controller(s):

```
# ansible-playbook compute.yml
```

Note Any newly created image will reside in the directory defined by the configuration variable `trix_image` which points to */trinity/images/* by default.

After the configuration has completed, the node image is ready and integrated into the provisioning system. No further steps are required.

Updating images and nodes

It is worth pointing out that `compute.yml` or any copy thereof can be applied to both existing images and/or live nodes without issues. All that needs to be done is updating the list of hosts to which it applies.

By default `compute.yml` applies to the host *compute.osimages.luna* which means it only applies to the image called *compute*. It is, therefore, possible to apply the same playbook to all images, a compute node, or all nodes if so desired. To do so, the hosts definitions in both `trinity-image-setup.yml` and `compute.yml` will need to be updated to either of the following:

- "osimages.luna" which will cover all osimages defined in Luna.
- "nodes.luna" which will cover all nodes defined in Luna.
- "node001.nodes.luna" which will only cover node001 as is defined in Luna.

2.1.4 TrinityX configuration variables

Global variables

All global variables in the TrinityX installer are stored in the *site/group_vars/all* file. All variables have sane defaults that will work out of the box, but care must be taken to make sure the default IPs are actually the IPs used on the controllers.

What follows is a list of those variables together with their descriptions and default values.

Note Due to some roles being called as dependencies of others, their configuration variables have been put alongside the global ones. For their descriptions, please refer to the relevant role's variables table.

Table 2.1. Global variables

Variable	Value	Default	Description
local_install	Boolean	false	Do we need to use local DVD/USB-stick as a source of the all packages
trix_version	•	•	The TrinityX version number. This will be automatically set to the current release version.
project_id	String	000000	Project ID or string that'll show up in the default shell prompt on the controllers. A pure esthetical configuration option that gives to shell prompts of the format *000000 hh:mm:ss [root@hostname ~]#*
ha	Boolean	true	This option allows to choose whether to do a highly available setup on two controllers or a single controller setup. Set to 'False' to disable HA.
trix_domain	Hostname	'cluster'	A domain name to be assigned to the controller(s) and nodes on the internal network. This also serves as luna's default provisioning network name.
trix_ctrl1_hostname	Hostname	'controller1'	Default hostname for the controller in a single controller setup. In HA setups, this is the hostname of the first controller.
trix_ctrl2_hostname	Hostname	'controller2'	This option is ignored in a single controller setup. In HA setups, this is the hostname of the second controller.
trix_ctrl_hostname	Hostname	'controller'	This option is set by the installer to the value of *trix_ctrl1_hostname* in a single controller setup. In HA setups, this the controllers' floating hostname that will always resolve to the controller with the primary role.
trix_ctrl1_ip	IP address	'10.141.255.254'	Default IP address of the controller in a single controller setup. In HA setups, this is the IP address of the first controller.
trix_ctrl2_ip	IP address	'10.141.255.253'	This option is ignored in a single controller setup. In HA setups, this is the IP address of the second controller.

trix_ctrl_ip	IP address	'10.141.255.252'	This option is set by the installer to the value of *trix_ctrl1_ip* in a single controller setup. In HA setups, this is the controllers' floating IP address that will always point to the controller with the primary role.
trix_ctrl1_bmcip	IP address	'10.148.255.254'	Only useful in HA setups for fencing purposes. This is the IP address of the BMC on the first controller that will be used to enable IPMI LAN fencing.
trix_ctrl2_bmcip	IP address	'10.148.255.253'	Only useful in HA setups for fencing purposes. This is the IP address of the BMC on the second controller that will be used to enable IPMI LAN fencing.
trix_cluster_net	IP address	'10.141.0.0'	Default provisioning network used by luna to allocate IP addresses to provisioned nodes. This will be the luna network whose name is defined in *trix_domain*.
trix_cluster_netprefix	Subnet prefix	16	The subnet prefix of the provisioning network defined in *trix_cluster_net*.
trix_cluster_dhcp_start	IP address	'10.141.128.0'	The IP address that marks the start of the DHCP IP range used by the provisioning tool to PXE boot the nodes. This IP address must belong to the network defined in *trix_cluster_net*.
trix_cluster_dhcp_end	IP address	'10.141.140.0'	The IP address that marks the end of the DHCP IP range used by the provisioning tool to PXE boot the nodes. This IP address must belong to the network defined in *trix_cluster_net*.
trix_root	Path	/trinity	Path to which the standard TrinityX files and directories will be installed.
trix_images	Path	*trix_root*/images	The default path where compute node images will be stored.
trix_shared	Path	*trix_root*/shared	The default path where everything shared by the controllers to the nodes will be stored.
trix_local	Path	*trix_root*/local	The default path where configuration files specific to each of the controllers will be stored.
trix_home	Path	*trix_root*/home	The default path where the user home directories will be located.
trix_repos	Path	*trix_root*/repos	The default path where the local TrinityX rpm repository will be located.
enable_selinux	Boolean	false	Whether or not to enable SELinux throughout the cluster.
enable_slurm_pam	Boolean	true	Whether or not to enable Slurm PAM module by default. If enabled, sssd's ldap filters will be disabled on the compute nodes.
enable_docker	Boolean	false	Whether or not to install docker tools on the cluster

enable_heartbeat_link	Boolean	true	Whether or not to configure the secondary corosync heartbeat link between the controllers.
shared_fs_type	String	'drbd'	The type of shared storage used on the controllers in TrinityX. Currently the only type supported by the installer is 'drbd'. Other types are planned for future releases.
shared_fs_device	Path	/dev/vdb	A path to the device that will be used as backend for the default 'drbd' storage type.
additional_env_modules	List	[]	A user-defined list of environment modules to install in addition to the default one. See the table *environment-modules role*.

Table 2.2. Global variables (cont'd)

trinityx_local_reponame	**String**	**'trinityx-local'**	**Name of the local repository in the case of offline installation**
trinityx_local_repo_baseurl	URL	http://*trix_ctrl_ip*:*repos_port*/ repos/trinityx/	URL to be used to access local repository
luna_repo	URL	https://updates.clustervision.com/luna/1.2/centos/luna-1.2.repo	URL of the luna repository
trinity_repo	URL	https://updates.clustervision.com/ trinity/10.2/centos/trinity.repo	URL of the trinity repository
userspace_repo	URL	https://updates.clustervision.com/userspace/userspace-release.x86_64.rpm	URL of the repository of the userspace packages
elrepo_repo	URL	http://www.elrepo.org/elrepo-release-7.0-3.el7.elrepo.noarch.rpm	URL of elrepo repository

Role specific variables

Below is a list of options that each ansible role in TrinityX supports.

The default values for each variable are set in *site/controller.yml*. For the sake of simplicity, not all variables appear in that file. You can find those missing variables and their defaults in the ansible role itself, in defaults directory (*site/roles/trinity/*/defaults/main.yml*).

bind role

Variable	Value	Default	Description
bind_dns_forwarders	List	• '8.8.8.8' • '8.8.4.4'	A list of the default DNS forwarders to use on the controllers.
bind_dnssec_enable	Boolean	no	Whether to enable DNSSEC in Bind9 on the controllers or not.
bind_db_path	Path	*trix_local*/var/named	The default path where Bind9 will store is DNS database.

resolv_server	IP address	127.0.0.1	Default nameserver to use in /etc/resolv.conf
resolv_search_domains	String	cluster ipmi	Default search domains to use in /etc/resolv.conf

chrony role

Variable	Value	Default	Description
chrony_upstream_servers	List	• '0.centos.pool.ntp.org' • '1.centos.pool.ntp.org' • '2.centos.pool.ntp.org' • '3.centos.pool.ntp.org'	A list of upstream NTP servers that will be used by the controller(s) to keep time on the cluster synchronized.
chrony_allow_networks	List	[]	A list of networks that are allowed to query the controller(s) for time. An empty list is the same as allowing all networks.

drbd role

Variable	Value	Default	Description
drbd_ctrl1_ip	IP address	*trix_ctrl1_ip*	IP address of the first of controllers in an HA setup.
drbd_ctrl2_ip	IP address	*trix_ctrl2_ip*	IP address of the second of controllers in an HA setup.
drbd_ctrl1_device	Path	/dev/drbd1	The name that will be given to the block device node of the DRBD resource on the first controller in an HA setup.
drbd_ctrl2_device	Path	*drbd_ctrl1_device*	The name that will be given to the block device node of the DRBD resource on the second controller in an HA setup.
drbd_ctrl1_disk	Disk name	*shared_fs_device*	A path to the device that will be used as backend for the DRBD resource on the first controller in an HA setup.
drbd_ctrl2_disk	Disk name	*drbd_ctrl1_disk*	A path to the device that will be used as backend for the DRBD resource on the second controller in an HA setup.
drbd_shared_resource_name	String	'trinity_disk'	The name that will be given to the DRBD resource on the controllers in an HA setup.

environment-modules role

Variable	Value	Default	Description
envmodules_version	String	*current version*	The release name of the userspace packages to install.
envmodules_files_path	Path	*trix_shared*/modules	Path where files for all environment modules should be installed in TrinityX cluster.

envmodules_default_list	List	• gcc • gdb • hwloc • intel-runtime • iozone • likwid • osu-benchmarks • python2 • python3 *versions omitted*	List of modules to install by default.

firewalld role

Variable	Value	Default	Description
firewalld_public_interfaces	List	['eth2']	A list of network interfaces that are considered to be public. i.e. used to access networks that are external to the cluster.
firewalld_trusted_interfaces	List	['eth0', 'eth1']	A list of network interfaces that are considered to be trusted. i.e. used to access networks that are interal to the cluster.
firewalld_public_tcp_ports	List	[443]	A list of TCP ports that will be allowd on the public interfaces defined in *firewalld_public_interfaces*
firewalld_public_udp_ports	List	[]	A list of UDP ports that will be allowd on the public interfaces defined in *firewalld_public_interfaces*
firewalld_masquerade_zone	firewall zone	public	Firewalld zone to enable masquerading on
firewalld_update_ifcfg_files	Boolean	true	Whether to update ifcfg-* files with zone information

luna role

Variable	Value	Default	Description
luna_user_id	User ID	880	The user ID of the luna user on the controller(s).
luna_group_id	Group ID	880	The group ID of the luna group on the controller(s).
luna	Dict		This the root of the object that describes how the cluster provisioning tool *luna* should be configured. It is a YAML dictionary. See the following variables for a description of all the attributes it supports.

luna.cluster	Dict		This sub-dictionary of the luna dict defines global luna options.
luna.cluster.frontend_address	IP address	*trix_ctrl_ip*	The IP address used by nodes during provisioning to query luna for configuration.
luna.cluster.path	Path	*trix_local*/luna	Path where all of luna's files will be stored on the controller(s).
luna.cluster.named_include_file	Path	*trix_local*/etc/named.luna.zones	Path where luna's Bind9 custom configuration will be located on the controller(s).
luna.cluster.named_zone_dir	Path	*trix_local*/var/lib/named	Path on the controller(s) where Bind9 will put DNS resolution files the networks managed by luna.
luna.dhcp	Dict		Sub-dict that defines luna's DHCP configuration used to PXE boot compute nodes.
luna.dhcp.conf_path	Path	*trix_local*/etc/dhcp	Path where generated DHCP configuration will be stored on the controller(s).
luna.dhcp.network	String	*trix_domain*	Name of network that will be used to provision compute nodes.
luna.dhcp.start_ip	IP address	*trix_cluster_dhcp_start*	The IP address that marks the start of the DHCP IP range used by luna to PXE boot the nodes.
luna.dhcp.end_ip	IP address	*trix_cluster_dhcp_end*	The IP address that marks the end of the DHCP IP range used by luna to PXE boot the nodes.

luna.networks	List of dict	See following	A list of dicts describing the networks that will be managed by luna. The dict that follows (which is also the first item of the luna.networks list) defines the attributes of the provisioning network.
luna.networks.0.name	String	*trix_domain*	The name that will be used for this network.
luna.networks.0.ip	IP address	*trix_cluster_net*	Network's address.
luna.networks.0.prefix	Number	*trix_cluster_netprefix*	Network's subnet prefix.
luna.networks.0.ns_ip	IP address	*trix_ctrl_ip*	IP address of the nameserver on this network. Usually this is the address of the controller(s) on this network.

mariadb role

Variable	Value	Default	Description
mariadb_db_path	Path	*trix_local*/var/lib/mysql	Path where MariaDB data folder will be located in a TrinityX cluster.

mongodb role

Variable	Value	Default	Description
mongo_db_path	Path	*trix_local*/var/lib/mongodb	Path where MongoDB data folder will be located in a TrinityX cluster.

nfs role

Variable	Value	Default	Description
nfs_rpccount	Number	256	Number of NFS server processes to be started on the controller(s).
nfs_enable_rdma	Boolean	false	Whether to enable NFS over RDMA by default or not. TCP will be used when this option if set to *false*.
nfs_export_shared	Boolean	true	If set to true, *trix_shared* directory will be exported to the compute nodes from the controller(s).
nfs_export_home	Boolean	true	If set to true, *trix_home* directory will be exported to the compute nodes from the controller(s).
nfs_exports_path	Path	*trix_local*/etc/exports.d	The path where to store NFS exports configuration on the controller(s).

obol role

Variable	Value	Default	Description
obol_conf_path	Path	/etc	Path where obol's configuration file will be stored on the controller(s).
users_home_path	Path	*trix_home*	Default home directory path to use for users created using obol.
ldap_host	FQDN	*trix_ctrl_hostname.trix_domain*	The FQDN of the ldap servers used to store ldap accounts on the cluster.

openldap role

Variable	Value	Default	Description
openldap_default_user	String	ldap	OpenLDAP default user name
openldap_default_group	String	ldap	OpenLDAP default group name
openldap_server_dir_path	Path	*trix_local*/var/lib/ldap	Path where OpenLDAPs databases will be stored on the controller(s).
openldap_server_conf_path	Path	*trix_local*/etc/openldap/slapd.d	Default path for the OpenLDAP configuration on the controller(s).
openldap_server_defaults_file	Path	/etc/sysconfig/slapd	Path where to put OpenLDAP's default command line options.
openldap_endpoints	String	'ldaps:/// ldapi:///'	Space separated list of endpoints that OpenLDAP will accept.
openldap_tls_cacrt	Path	*ssl_ca_cert*	Path of CA cert used to sign the controller(s) certificate(s).
openldap_tls_crt	Path	*ssl_cert_path*/*ansible_fqdn*.crt	Path of the controller(s) certificate(s).
openldap_tls_key	Path	*ssl_cert_path*/*ansible_fqdn*.key	Path of the controller(s) key(s).
openldap_schemas	List	• cosine • inetorgperson • rfc2307bis • autoinc	List of the schemas to be configured in OpenLDAP.

pacemaker role

Variable	Value	Default	Description
pacemaker_properties	Dict	no-quorum-policy: ignore	A list of pacemaker configuration options.
pacemaker_resource_defaults	List	• 'migration-threshold=1'	A list of pacemaker resource defaults.
fence_ipmilan_host_check	String	'static-list'	This option helps the stonith agent determine which machines are controlled by the fencing device.

fence_ipmilan_method	String	'cycle'	Method to fence (onoff or cycle)
fence_ipmilan_lanplus	String	'true'	Use Lanplus if True, don't otherwise.
fence_ipmilan_login	String	'user'	Username/Login (if required) to control power on IPMI device
fence_ipmilan_passwd	String	'password'	Password (if required) to control power on IPMI device

repos role

Variable	Value	Default	Description
repos	List		List of package repositories to install.
repos_port	Number	8080	Default port to listen on when serving the local package repository on the controller(s).

rsyslog role

Variable	Value	Default	Description
syslog_forwarding_rules	List of dicts		A list of log forwarding rules to use in rsyslog.d/ configuration files.
.0.name	String		Forwarding rule's name
.0.proto	String		Protocol to use for this rule. Can be TCP or UDP.
.0.port	Number		The port to which rsyslog will send logs that match the rule.
.0.host	String		The destination host.
.0.facility	String		Syslog facility name to use for logs sent through this rule.
.0.level	String		Syslog level to use for logs send through this rule.
syslog_listeners	List of dicts		A list of listeners to be configured in rsyslog.
syslog_listeners.0.name	String	default	Listener's name
syslog_listeners.0.proto	String	tcp	Listener's protocol. Can be TCP or UDP
syslog_listeners.0.port	Number	514	Listener's port.
syslog_file_template_rules	List of dicts		A list of template rules. See http://www.rsyslog.com/doc/master/configuration/templates.html for details.
syslog_file_template_rules	List of dicts		
.0.name	String	controllers	Template name
.0.type	String	string	Template type
.0.content	String	'/var/log/cluster-messages/%HOSTNAME%.messages'	Content of the template rule.
.0.field	String	'$fromhost-ip'	Template's field
.0.criteria	String	startswith	Templates's criteria

.0.rule	String	trix_cluster_net	The matching rule for the template.

The default for the syslog_file_template_rules.0.rule is '{{ trix_cluster_net.split(".")[:trix_cluster_net-prefix//8]|join(".") }}'

slurm role

Variable	Value	Default	Description
slurm_conf_path	String	*trix_shared*/etc/slurm	Path where slurm configuration files are stored.
slurm_spool_path	Path	*trix_local*/var/spool/slurm	Path for slurm's working data.
slurm_log_path	Path	/var/log/slurm	Location where to store slurm logs.
slurm_user_id	Number	891	slurm's user ID
slurm_group_id	Number	891	slurm's group ID
slurm_ctrl	Hostname	*trix_ctrl_hostname*	Hostname of the slurm controller
slurm_ctrl_ip	IP address	*trix_ctrl_ip*	IP address of the slurm controller
slurm_ctrl_list	Hostname list	*trix_ctrl1_hostname,trix_ctrl2_hostname*	Comma separated list of the machines that serve as slurm controller.
enable_slurm_pam	Boolean	true	Enable or disable slurm's PAM module that denies user access to nodes where they don't have a running job.
slurmdbd_sql_user	String	'slurm_accounting'	Name to use for slurmdbs's SQL user.
slurmdbd_sql_db	String	'slurm_accounting'	Name to use for slurmdbd's database.
munge_user_id	Number	892	munge's user ID
munge_group_id	Number	892	munge's group ID
munge_conf_path	Path	*trix_shared*/etc/munge	Path where munge's configuration files will be stored.

ssl-cert role

Variable	Value	Default	Description
ssl_cert_path	Path	*trix_local*/etc/ssl	Location where to store cluster certificates and keys.
ssl_cert_country	String	'NL'	CA certificate country attribute
ssl_cert_locality	String	'Amsterdam'	CA certificate locality attribute
ssl_cert_organization	String	'ClusterVision B.V.'	CA certificate organization attribute
ssl_cert_state	String	'Noord Holland'	CA certificate state attribute
ssl_cert_altname	FQDN	*trix_ctrl_hostname.trix_domain*	CA certificate alternative name attribute
ssl_cert_days	Number	3650	Number of controller's certificate validity days.
ssl_cert_owner	String	'root'	Default owner of the certificate files
ssl_cert_owner_id	Number	0	Default owner's id

ssl_cert_group	String	'ssl'	Default group owner of the certificate files
ssl_cert_group_id	Number	991	Default group owner's id

sssd role

Variable	Value	Default	Description
sss_allowed_groups	List	• admins	List of user groups that are allowed access on the controller(s).
sss_ldap_hosts	List	• *trix_ctrl_hostname.trix_domain*	List of hostnames that sssd can use for its ldap queries.
sss_filter_enabled	Boolean	false	Whether to use group based access filters on restrict access to compute nodes or not.

Compute specific variables

Global variables

Variable	Value	Default	Description
image_name	String	compute	The name of the OS image to create or to which to apply the playbook
image_password	String		The password to set up for the root user in the image. If empty, it will be randomly generated.

nfs-mounts role

Variable	Value	Default	Description
nfs_mounts	List of dicts	see below	A list of NFS mountpoints and their options
nfs_mounts.0.path	String	'/trinity/shared'	Path on the compute nodes where the NFS share will be mounted
nfs_mounts.0.remote	Path	controller:/trinity/shared	NFS share to mount
nfs_mounts.0.options	String	'defaults,nfsvers=4,ro,retrans=4'	Mount point options

2.1.5 Ansible Roles overview

- init: checks if primary controller, disables SElinux, removes NetworkManager
- local_repo: creates repository from DVD or USB with TrinityX DVD content on the controller
- repos: role to install repositories on the controller and images. Usually it is used as a dependency in other roles
- cv_support: installs remote assistance script, trinity health package, updates root's bashrc
- packages: installs all required diag packages required on a proper HPC head node. See roles/-packages/defaults/main.yml
- tunables: updates limits.conf and sysctl.conf
- hostname: sets hostname and /etc/hosts
- firewalld: installs, configures and enables firewalld
- chrony: installs, configures and enables chrony (NTP sync)
- rdma-centos: installs and enables RDMA
- ssh: configures passwordless SSH between controllers
- fail2ban: installs, configures and enables fail2ban

- pacemaker: installs and configures pacemaker and corosync for HA
- drbd: creates DRBD filesystem, adds resources to pacemaker
- trix-tree: just creates /trinity
- nfs: configures NFS export, adds resources to pacemaker
- environment-modules: prepares module environment and installs a default set of applications, see *environment-modules role*.
- ssl-cert: generates SSL certs for HTTPS and LDAPS
- openldap: installs OpenLDAP and configures schema
- obol: installs and configures obol (user management tool)
- sssd: installs, configures, and enables sssd
- mariadb: installs, configures MariaDB (MySQL), adds resources to pacemaker
- slurm: installs and configures Slurm, adds resources to pacemaker
- mongodb: installs and configures MongoDB, DB for luna
- bind: installs and configures bind/named, adds resources to pacemaker
- nginx: installs and configures nginx (web server for luna)
- luna: installs and configures luna
- rsyslog: installs and configures syslog
- docker-registry: installs and configures docker registry

2.1.6 Custom storage configurations for compute nodes

As you have seen in the luna documentation, by default, the compute nodes are provisioned in diskless mode:

```
# luna group show compute --partscript

mount -t tmpfs tmpfs /sysroot
```

What follows are examples of different partitioning schemes that could be used on the compute nodes.

Simple diskful setup

The following luna partitioning script will, on every reboot, delete and re-partition the `/dev/sda` disk according to the scheme:

- `/dev/sda1` as a boot partition
- `/dev/sda2` as a partition for the root file system

The script only makes use of Linux utilities that are available at provisioning time like `parted` and `mkfs.ext4`:

```
parted /dev/sda -s 'mklabel msdos'
parted /dev/sda -s 'rm 1; rm 2'
parted /dev/sda -s 'mkpart p ext2 1 256m'
parted /dev/sda -s 'mkpart p ext3 256m 100%'
parted /dev/sda -s 'set 1 boot on'

mkfs.ext2 /dev/sda1
mkfs.ext4 /dev/sda2

mount /dev/sda2 /sysroot
```

```
mkdir /sysroot/boot
mount /dev/sda1 /sysroot/boot
```

It may be necessary to format the compute disks using a different filesystem which is not included in the partitioning script run within the ramdisk. In such cases, update the `dracut` configuration in the osimage to include the desired filesystem support and utilities and then pack it up.

LVM based scheme

Sometimes, it might be useful to have an LVM based disk layout on the compute nodes, either for the root filesystem or for scratch storage.

The following is an example of a simple LVM partitioning scheme that uses the logical volume `storage/root` for the root filesystem on the compute nodes:

```
# Partition the sda disk
    parted /dev/sda -s 'mklabel msdos'
    parted /dev/sda -s 'rm 1; rm 2'
    parted /dev/sda -s 'mkpart p ext2 1 256m'
    parted /dev/sda -s 'mkpart p ext3 256m 100%'
    parted /dev/sda -s 'set 1 boot on'

# Update the LVM locking type which is set to read-only by default
sed -ie 's|\(\s*locking_type\)\s*=.*|\1 = 1|' /etc/lvm/lvm.conf

# Destroy any previously created PVs
lvm lvchange -a n storage
echo y | lvm pvremove /dev/sda2 -f -f

# Create PVs, VGs and LVs
lvm pvcreate /dev/sda2 -f
lvm vgcreate storage /dev/sda2 -f
lvm lvcreate storage -L 5g --name root

# Format boot and root disks
    mkfs.ext2 /dev/sda1
    mkfs.ext4 /dev/storage/root

    # Mount disks to their appropriate locations
    mkdir /sysroot/boot
    mount /dev/storage/root /sysroot
    mount /dev/sda1 /sysroot/boot
```

Do note that for the above to work, some changes need to be made to the osimage:

- `lvm` dracut module must be enabled in luna:

  ```
  # luna osimage change compute --dracutmodules 'luna,-i18n,-plymouth,lvm'
  ```

- `lvm2` must be installed in the osimage:

  ```
  # lchroot compute yum install lvm2
  ```

Note This example serves only as a sample LVM based configuration. Using the same methodology, more complex configurations can be achieved.

Software RAID1 scheme

2.2 Administration manual

2.2.1 User management in a TrinityX cluster

User management in TrinityX is handled by a utility called *obol*. Obol is a simple wrapper around LDAP commands to update the local LDAP directory installed by default. It supports both user and group management almost the same way those would be handled by the default Linux utilities.

Obol

Obol can manage users and groups on the local LDAP directory and it supports the following attributes for users:

Attribute	Description
`password`	User's password
`cn`	User's name (common name)
`sn`	User's surname
`givenName`	User's given name
`group`	The primary group the user belongs to
`uid`	The user's ID
`gid`	The primary group's ID
`mail`	Email address of the user
`phone`	Phone number
`shell`	Default shell
`groups`	A comma separated list of additional group names to add the user to.
`expire`	Number of days after which the account expires. If set to -1, the account will never expire

To create or modify a user, run:

```
# obol user add|modify ...
```

Note Please note that running obol commands requires root privileges.

Managing groups is similarly achieved using:

```
# obol group [command] ....
```

Where [command] can either be `add`, `show`, `delete`, `list`, `addusers`, or `delusers`

For the full list of the commands supported by Obol, run:

```
obol user -h
obol user [command] -h

obol group -h
obol group [command] -h
```

Cluster access

TrinityX supports both a group-based access control system and a SLURM PAM-based one where only users with running jobs can access the resources allocated to them.

When group-based access control (GBAC) is used, TrinityX will allow or deny access to the compute nodes based on the groups to which a user belongs. By default, TrinityX only allows access to users that belong to the group `admins`.

Choosing between the two described access modes is allowed by the `enable_slurm_pam` variable in trinityX/site/group_vars/all. If set to *true*, SLURM PAM will be used, otherwise, group-based filtering will be used.

Note If for some reason there is a need to update the list of groups that have access to the nodes or disable this control altogether, `sssd.conf` should be updated on each node. Change `ldap_access_filter` or `ldap_access_order` from `filter,expire` to `expire`.

Authentication backends

TrinityX installations come with an openldap directory by default. This directory is used for authentication across the cluster. In some cases, we might want to use a different directory entirely or another remote directory in combination with the local one.

Using a remote directory

In order to mask the local openldap directory and only use a remote one, an administrator needs to update sssd's configuration file across the cluster (i.e. on the controllers as well as on compute images).

What they will need to update is the two options *ldap_uri* and *ldap_search_base*:

- *ldap_uri* will need to point to the remote directory (e.g. ldaps://ldap.university.edu/)
- *ldap_search_base* will need to refer to the distinguished name of the directory (e.g. dc=university,dc=edu)

For the changes to take effect, sssd.service will need to be restarted on all the affected nodes (and compute nodes reprovisioned).

Using the local and a remote directory at the same time

In cases where an administrator prefers to use the local directory in combination with a remote one then one will need to update both sssd's and openldap's configurations.

The local openldap installation has a special proxy database *dn: dc=cluster* that can serve as an aggregator for multiple ldap directories. The local one, *dn: dc=local*, is already accounted for in the proxy with a *dn: dc=local,dc=cluster*. So the administrator only needs to add a reference to the remote directory:

```
ldapmodify -Y EXTERNAL -H ldapi:///

dn: olcMetaSub={0}uri,olcDatabase={2}meta,cn=config
changetype: add
objectClass: olcMetaTargetConfig
olcMetaSub: {0}uri
olcDbURI: "ldapi:///dc=remote,dc=cluster" ldap://ldap.university.edu
olcDbRewrite: {0}suffixmassage "dc=remote,dc=cluster" "dc=university,dc=edu"
```

Then, the sssd configuration must be updated to point to the proxy database:

- *ldap_search_base* has to refer to *dc=cluster*

For changes to take effect, sssd.service must be restarted on all affected nodes (and compute nodes reprovisioned).

2.2.2 Hints and tips for a Luna installation

Setting up a compute node image

Note This chapter assumes familiarity with the TrinityX configuration tool and the configuration files that it uses. If not, please refer to the general *Trinity X* first.

TrinityX includes configuration files to automatically create a basic compute node image. Upon delivery of a new cluster, this will have been completed by ClusterVision engineers and the image will be integrated into the provisioning system, in this case Luna.

Cloning an existing image

The easiest way to get a new image for modifications is to clone an existing one. For example:

```
# luna osimage list
+------------+----------------------------+-------------------------------+
| Name       | Path                       | Kernel version                |
+------------+----------------------------+-------------------------------+
| compute    | /trinity/images/compute    | 3.10.0-693.17.1.el7.x86_64    |
+------------+----------------------------+-------------------------------+

# luna osimage clone compute -t new-compute -p /trinity/images/new-compute
INFO:luna.osimage.compute:/trinity/images/compute => /trinity/images/new-compute

# luna osimage list
+----------------+--------------------------------+--------------------------------+
| Name           | Path                           | Kernel version
|
+----------------+--------------------------------+--------------------------------+
| compute        | /trinity/images/compute        | 3.10.0-693.17.1.el7.x86_64
|
| new-compute    | /trinity/images/new-compute    | 3.10.0-693.17.1.el7.x86_64
|
+----------------+--------------------------------+--------------------------------+
```

Creating a new image

In some cases it may be preferable to start from a fresh image. In that case, copy the existing playbook and change the name of the image (`image_name` variable and `hosts` target):

```
# cp compute.yml new-compute.yml
# sed -i -e 's/compute/new-compute/' new-compute.yml
```

In the event an image has been customized/more roles added, run `ansible-playbook` once finished:

```
# ansible-playbook new-compute.yml
```

To speed up creation time, it is possible to disable packing of the image. This is handy when testing changes:

```
# ansible-playbook --skip-tags=wrapup-images  new-compute.yml
```

Modifying images and packing them

There are two approaches to creating new images in TrinityX: manually and via Ansible. The same is true for modifying images.

In the manual approach, first clone the image with `luna osimage clone` and modify as a regular set of files or in chrooted environment.

Regarding the manual modification of an image's content: Luna has a tool called `lchroot`. It is a wrapper around the good old `chroot` tool, but with some additions and integration with Luna. In particular, it mounts /dev, /proc, /sys filesystems on start and unmounts on exit. Don't try to pack the image if an `lchroot` session is running. Packing the image at this point will pack the content of the mentioned service filesystems and this is probably not desired.

The tool can also mock the kernel version allowing to install software which requires particular release:

```
# uname -r
3.10.0-693.17.1.el7.x86_64
```

```
# lchroot new-compute
IMAGE PATH: /trinity/images/new-compute
chroot [root@new-compute /]$ uname -r
3.10.0-693.5.2.el7.x86_64
```

Before packing the image, it may be desirable to change kernel options or the kernel version if several are installed:

```
# luna osimage change new-compute --kernopts "console=ttyS1,115200n8 console=tty0
intel_pstate=disable"
```

To list installed kernels:

```
# luna osimage show new-compute --kernver
3.10.0-693.17.1.el7.x86_64 <=
3.10.0-693.5.2.el7.x86_64
```

And to change:

```
# luna osimage change new-compute --kernver 3.10.0-693.5.2.el7.x86_64

# luna osimage show new-compute --kernver
3.10.0-693.17.1.el7.x86_64
3.10.0-693.5.2.el7.x86_64 <=
```

After customization is done, it is important to pack the image so that the changes are available to the nodes. Packing is done by a creating a tarball, creating a torrent-file from it, and adding it to the Luna DB. Everything will be done automatically when `luna osimage pack` is executed:

```
# luna osimage pack new-compute
INFO:root:Creating tarball.
INFO:root:Done.
INFO:root:Creating torrent.
INFO:root:Done.
INFO:root:Copying kernel & packing inirtd.
INFO:root:Done.
```

In TrinityX's playbook, this task is done by the `wrapup-images` role:

```
TASK [trinity/wrapup-images : Pack the image] *********************************
changed: [new-compute.osimages.luna -> localhost]
```

Grabbing an image from a live node

This method is handy when some software requires hardware to be physically present on a node to run its installation procedure. After installation is complete, it is possible to sync files back to the image. Before doing so, it is worthwhile to inspect `--grab_exclude_list` and `--grab_filesystems` options in order to limit the amount of data to be synced. To check what needs to be synced, `--dry_run` can be specified:

```
# luna osimage grab new-compute --host node001 --dry_run
INFO:luna.osimage.new-compute:Fetching / from node001
INFO:luna.osimage.new-compute:Running command: /usr/bin/rsync -avxz -HAX -e
"/usr/bin/ssh -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null"
--progress --delete --exclude-from=/tmp/new-compute.excl_list.rsync.ybBx8D
--dry-run  root@node001:/ /trinity/images/new-compute/
<...snip...>
```

Networks in Luna

Networks in Luna have 3 main attributes: the name, the network itself, and the prefix:

```
# luna network show cluster
```

```
+----------------+--------------------+
| Parameter      | Value              |
+----------------+--------------------+
| name           | cluster            |
| NETWORK        | 10.141.0.0         |
| PREFIX         | 16                 |
| include        | -                  |
| ns_hostname    | controller         |
| ns_ip          | 10.141.255.252     |
| rev_include    | -                  |
| version        | 4                  |
| comment        |                    |
+----------------+--------------------+
```

The name is used as a domain for DNS. All IP addresses to be defined later in Luna will inherit their properties from the network definition. Networks in Luna automatically check for IP address uniqueness in order to avoid IP address conflicts. All occupied IP addresses can be listed:

```
# luna network show ipmi --reservedips | sort
10.149.0.1:node001
10.149.0.2:node002
10.149.0.3:node003
10.149.0.4:node004
10.149.200.1:switch01
10.149.250.1:pdu01
10.149.255.254:controller
```

Luna can manage DNS zones by itself. After running `luna cluster makedns`, a user will be able to resolve, for example, node001.ipmi and pdu01.ipmi hostnames. Luna will create reverse zones as well. If it is required to create additional records in DNS, like MX or SRV, `--include` and `--rev_include` options can be used.

Groups in Luna

A key concept in Luna is that of groups. Most (after osimage) of the customizations in Luna are performed here. A group is a homogeneous set of nodes. They usually have the same role within the cluster, with a similar hardware configuration, software set, and are connected to the same networks. Usually, they are logically grouped to the same queue (or partition) in the scheduling system. It is possible to specify the same osimage for several groups and perform additional customizations on install.

Creating a group requires the osimage to be specified. A group can't exist without an image or connection to a network. It is assumed that nodes need to be installed via the network, as we are using a network provisioning tool:

```
# luna group add --name new-compute-group --osimage new-compute --network cluster

# luna group show new-compute-group
+---------------+-------------------------------------------------+
| Parameter     | Value                                           |
+---------------+-------------------------------------------------+
| name          | new-compute-group                               |
| bmcsetup      | -                                               |
| domain        | [cluster]                                       |
| interfaces    | [BOOTIF]:[cluster]:10.141.0.0/16                |
| osimage       | [new-compute]                                   |
| partscript    | mount -t tmpfs tmpfs /sysroot                   |
| postscript    | cat << EOF >> /sysroot/etc/fstab                |
|               | tmpfs   /       tmpfs    defaults        0 0    |
|               | EOF                                             |
| prescript     |                                                 |
| torrent_if    | -                                               |
| comment       |                                                 |
+---------------+-------------------------------------------------+
```

In addition it is possible to specify a management (IPMI/BMC) network:

```
# luna group add --name new-compute-group --osimage new-compute --network cluster
--bmcnetwork ipmi --bmcsetup bmcconfig

# luna group show new-compute-group
+---------------+-------------------------------------------------+
| Parameter     | Value                                           |
+---------------+-------------------------------------------------+
| name          | new-compute-group                               |
| bmcsetup      | [bmcconfig]                                     |
| domain        | [cluster]                                       |
| interfaces    | [BMC]:   [ipmi]:10.149.0.0/16                   |
|               | [BOOTIF]:[cluster]:10.141.0.0/16                |
| osimage       | [new-compute]                                   |
| partscript    | mount -t tmpfs tmpfs /sysroot                   |
| postscript    | cat << EOF >> /sysroot/etc/fstab                |
|               | tmpfs   /       tmpfs    defaults        0 0    |
|               | EOF                                             |
| prescript     |                                                 |
| torrent_if    | -                                               |
| comment       |                                                 |
+---------------+-------------------------------------------------+
```

In this case, the IPMI configuration will be enforced on install, configuring the IP address and credentials for remote power management via `lpower`. This can be added, deleted, or changed later.

Please note the two interfaces BMC and BOOTIF on the example above.

- BMC reflects the IPMI interface of the node. Applied config can be found in the `ipmitool lan print` output on the node.
- BOOTIF is a synonym of the interface node connected to the network. Usually Luna operates with the actual names of interfaces, like eth0, em1, p2p1 or ib0. If BOOTIF is specified as the name, Luna tries to find the real name of the interface based on the MAC-address exposed by the node on boot.

To add nodes to the group simply run:

```
# luna node add --name node001 --group new-compute-group
```

Configuring interfaces

In simple cases, networking will just work. But sometimes a non-trivial configuration is necessary, in cases where bonding, bridging, or a VLAN config is required. This can be done with Luna.

First, it may be necessary to rename the interfaces:

```
# luna group change new-compute-group --interface BOOTIF --rename bond0
INFO:group.new-compute-group:No boot interface for nodes in the group configured.
DHCP will be used during provisioning.
```

And add two more interfaces:

```
# luna group change new-compute-group --interface eth0 --add
# luna group change new-compute-group --interface eth1 --add
```

Then, change the configuration of the interfaces, as one would configure `/etc/sysconfig/network-scripts/ifcfg-*` files. To do so, specify the `--edit` argument:

```
# luna group change new-compute-group --interface bond0 --edit
```

This will open an editor in which the configuration can be typed with regular `ifcfg-*` syntax. Optionally, the `--edit` flag accepts piping from STDIN:

```
# cat << EOF | luna group change new-compute-group --interface bond0 --edit
> TYPE=Bond
> BONDING_MASTER=yes
> BONDING_OPTS="mode=1"
> EOF

# cat << EOF | luna group change new-compute-group --interface eth0 --edit
> MASTER=bond0
> SLAVE=yes
> EOF

# cat << EOF | luna group change new-compute-group --interface eth1 --edit
> MASTER=bond0
> SLAVE=yes
> EOF
```

Please note that it is unnecessary to specify `NAME=` and `DEVICE=` for interfaces; `IPADDR=` and `PREFIX=` will be added automatically on a per-node basis.

Scripts in groups

Sometimes the installation procedure needs to be altered to perform some tasks before or after the osimage is deployed. Customization scripts come into play here. Each group has 3: prescript, partscript, and postscript.

- `prescript` is performed before any other task of the installation procedure. Can be handy if we need to insert a non-standard kernel module for later use or check some hardware status.
- `partscript` creates partitions and prepares filesystems to unpack the tarball. Dracut expects that all needed files will be located in `/sysroot` to perform switch_root to boot the actual OS up. We need to create filesystems and mount them under `/sysroot`. Also, partscript is a good place to check if the disk we are going to use for the OS is the proper one: check the size and hardware path of the disk.
- `postscript` is for finishing up installation: install bootloader on disk, perform some customization of the unpacked image, etc.

Some examples of the scripts can be found in `man luna`.

By default, every group is created with the default partscript where the osimage will be placed in memory. This is a so-called "diskless" configuration. Any file on the local filesystems will not be touched or altered. Changing the partscript from default to the following example will convert a node from diskless to diskful:

```
parted /dev/sda -s 'mklabel msdos'
parted /dev/sda -s 'rm 1; rm 2'
parted /dev/sda -s 'mkpart p ext2 1 256m'
parted /dev/sda -s 'mkpart p ext3 256m 100%'
parted /dev/sda -s 'set 1 boot on'
mkfs.ext2 /dev/sda1
mkfs.ext4 /dev/sda2
mount /dev/sda2 /sysroot
mkdir /sysroot/boot
mount /dev/sda1 /sysroot/boot
```

Please note that it is not necessary to change the osimage in order to make a node diskful. The same image can be used, but instead of mounting the ramdisk to `/sysroot`, /dev/sda2 is placed there.

To make a node self-contained, bootloader should be added and fstab changed to communiate to systemd where to find `/`:

```
mount -o bind /proc /sysroot/proc
mount -o bind /dev /sysroot/dev
chroot /sysroot /bin/bash -c "/usr/sbin/grub2-mkconfig \
```

```
    -o /boot/grub2/grub.cfg; /usr/sbin/grub2-install /dev/sda"
chroot /sysroot /bin/bash -c \
    "echo '/dev/sda2 /     ext4 defaults 0 0' >> /etc/fstab"
chroot /sysroot /bin/bash -c \
    "echo '/dev/sda1 /boot ext4 defaults 0 0' >> /etc/fstab"
umount /sysroot/dev
umount /sysroot/proc
```

To edit the script, simply run:

```
# luna group change new-compute --partscript --edit
```

It will open the editor. In addition, it supports piping:

```
# cat compute-part.txt | luna group change compute --partscript --edit
```

Other configurable items in Luna

Switches must be configured for Luna to automatically discover nodes' MAC addresses. It is crucial to check if a switch provides information about learned MAC addresses vin SNMP:

```
# snmpwalk -On -c public -v 1 SWITCH_IP .1.3.6.1.2.1.17.7.1.2.2.1
```

It should list something like:

```
.1.3.6.1.2.1.17.7.1.2.2.1.2.1.24.102.218.96.27.201 = INTEGER: 210
```

The last 6 numbers is a MAC address in decimal format. See `man luna` for more information on how to decrypt it.

When Luna is able to get MAC addresses from switches, it will display them in `luna cluster listmacs`.

Other devices present as `otherdev` in Luna. This class of configurable items will fill DNS records. For example, it is handy to resolve PDUs' hostnames.

The last item worthy of mention is `bmcsetup`. It describes the IPMI/BMC settings for nodes: credentials and IPMI control channels.

Node management

As said, most of the tunables for nodes should be performed on a group level. However, several items need to be managed individually for each node. These are IP addresses, MAC address, and switch/-port pair.

The MAC address is considered a unique identifier of the node. If not configured manually, it will be acquired based on the switch and port configuration. Another way of setting up the MAC address is to choose node name from the list during boot. If the MAC address is not known for the node, it will be looping in the boot menu.

IP address for a node is always configured from the network defined in the corresponding group. IP is always assigned on the interface if the network is configured for this interface on the group level and Luna controls this rule.

It is possible to change the group for a node and Luna does its best to preserve configured IP addresses. It can be tricky as the set of interfaces on the destination group might be different from that of the source group.

Further individual settings for node are `--setupbmc` and `--service`. These are mostly relevant for debugging. The first allows disabling of attempts to configure BMC, as it is known this configuration might be flaky. `--service` tunable can be handy if an engineer needs to debug boot issues. Nodes in this mode will not try to run the install script, but will stay in the initrd stage, configure 2 consoles (Alt+F1, Alt+F2), and try to set up IP addresses and run ssh daemon. In addition, it can be used to inspect the hardware configuration of the node before setup and wiping of data on disks.

Another debug feature is a flag `luna node show --script` which accepts two options: `boot` and `install`.

- `--script boot` shows the exact boot options node will use to fetch and run kernel and initrd.
- `--script install` provides a way to inspect the script that will be used to install the node. Combined with `--service yes` it is a good way to catch mistakes like unpaired parentheses or quotes in pre/part/post scripts.

Debug hints

Sometimes a node refuses to boot and it is hard to say why. To address the issue, first check which step of the boot process gets stuck.

There are several boot steps:

- PXE/iPXE
- Luna boot menu
- Initrd
- Install procedure

First check the status of `node show` to get an idea of where the issue is. If this status is empty, most likely the node hangs somewhere before or in the boot menu.

For PXE/iPXE issues, the first suspect is usually the firewall. Then, check if the node is able to get an IP address from the DHCP range: check `/var/log/messages` on the controller, lease file, and DHCP range in `luna cluster show` and `/etc/dhcpd.conf`. Check if the node is able to download the `luna_undionly.kpxe` binary from the TFTP server using `tftp get`.

If a node is able to show the boot menu (blue one), but refuses to go further, check if the node has a proper MAC address configured. If the node has the switch/port configured, check `luna cluster listmacs` output to make sure Luna is able to acquire MAC addresses from the switches. Sometimes it takes several minutes to download all MAC addresses from all switches. Once this is done, check nginx logs in `/var/log/nginx`, `/var/log/luna/lweb_tornado.log`, and `--script boot` script. Then, check permissions and content in the `~luna/boot` folder. Be sure `osimage pack` has been run before trying to boot the node.

If the node is able to fetch the kernel and initrd (this will be visible in nginx logs), the next step in debugging is to be sure the kernel is able to boot. This usually has no issues; those which may arise are typically limited to general Lunux issues - incompatible hardware, for example.

At this step, access to the console can be gained by pressing Alt+F1 or Alt+F2. Check if the node is pingable and accessible via ssh.

If Luna is unable to configure IP addresses, please check that the nodes have interfaces visible in `ip a` output. It might be a driver issue in this case. To fix it, add drivers to dracut. This can be done in `/etc/dracut.conf.d` in the osimage (don't forget to repack after changes!). In `man dracut`, pay special attention to `dracutmodules+=`, `add_drivers+=` and `install_items+=`.

If the network is working but the node is unable to proceed with installation, check the nginx logs to be sure the node is trying to download the installation script. Check the output of `--script install` to see the script. Check `journalctl -xe` on the node and search for occurrences of `Luna`. Check the content of the `/luna` folder on the node. It should at least contain the `install.sh` script. Later, it will contain `*.torrent` file. The next step is to check the tarball in `/sysroot` on the node. It should exist and be the same size as in `~luna/torrents`. Inspect nginx logs for `announce` URLs. Pay attention to the `peer_id=` and `downloaded=` section. Records with `peer_id=lunalunalunalunaluna` are originating from the controller.

At this point, partscript should prepare `/sysroot`, i.e. format and mount disks or mount ramdisk. If some issues arise here, be sure the desired filesystem appears in `/proc/filesystems` on the node. Otherwise, use `filesystems+=` for dracut in the osimage (and pack again). Be sure there is enough space - 4G is absolute minimum. At some point during installation, the tarball itself and unpacked tarball will be present on the same filesystem, so a capacity of 2x the size of the osimage is required.

On this step, `/sysroot` should contain the same set of files as osimage configured for node. After `postscript`, the Luna dracut module is ready to exit and give control to systemd boot procedures. If

boot gets stuck, check that the filesystem was configured in the previous step. A common error is the failure to mount any filesystem to `/sysroot` and unpack content just in memory.

For more details about Luna boot internals, read `doc/hints-n-tips/boot-process.md` in Luna's repository.

2.2.3 Basic administrative tasks in Slurm

Daemons and files

SLURM functionality in TrinityX relies on 3 daemons: munged, slurmctld and slurmdbd running on the controller. Every compute node has slurmd running.

- `munged` is handling security communication between slurmd and slurmctld
- `slurmctld` main service doing heavy-lifting of a proper job scheduling
- `slurmdbd` SLURM accounting
- `slurmd` daemon running on compute nodes and spawning user executables

In addition, `slurmdbd` must have the mysql daemon running to store accounting data.

By default, the SLURM config consists of several files located in `/etc/slurm`, which is symlinked to `/trinity/shared/etc/slurm`.

- `slurm.conf` main config
- `slurm-nodes.conf` nodes' definitions
- `slurm-partitions.conf` definitions of queue
- `slurm-user.conf` customizations
- `slurmdbd.conf` configuration file of slurmdbd daemon

In addition, the `mongod` daemon relies on the corresponding config located at `/etc/mongod.conf`.

TrinityX has several `systemd` customizations located in /etc/systemd/system:

```
# ls /etc/systemd/system/{munge*,slurm*}
/etc/systemd/system/munge.service.d:
trinity.conf

/etc/systemd/system/slurmctld.service.d:
trinity.conf

/etc/systemd/system/slurmdbd.service.d:
trinity.conf
```

Log files can be found in `/var/log/slurm` on both controllers and compute nodes.

Commands

The most popular commands from an administrator perspective are usually `sinfo`, `squeue`, and `scontrol`.

- `sinfo` show status of the nodes and queues
- `squeue` list of jobs running on the cluster
- `scontrol` manage SLURM configuration and state

For `sinfo`, pay special attention to the "NODE STATE CODES" section in the man pages.

`scontrol` allows SLURM to be reconfigured on the fly. For example, we can drain (bring it offline in SLURM for maintenance purposes) and un-drain the node in the following way:

```
# sinfo
```

```
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
defq*        up   infinite      1   idle node001

# scontrol update node=node001 state=drain reason='Heavily broken'

# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
defq*        up   infinite      1  drain node001

# scontrol show node=node001
NodeName=node001 Arch=x86_64 CoresPerSocket=1
   CPUAlloc=0 CPUErr=0 CPUTot=1 CPULoad=0.13
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=node001 NodeHostName=node001 Port=0 Version=17.02
   OS=Linux RealMemory=100 AllocMem=0 FreeMem=4222 Sockets=1 Boards=1
   State=IDLE+DRAIN ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=defq
   BootTime=2018-03-09T16:01:07 SlurmdStartTime=2018-03-09T16:03:08
   CfgTRES=cpu=1,mem=100M
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
   Reason=Heavily broken [root@2018-03-09T16:04:43]
```

To make the node available for user jobs:

```
# scontrol update node=node001 state=undrain
```

With `scontrol`, it is possible to check the status of the running jobs:

```
# squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                 2      defq    sleep     root  R       0:07      1 node001

 # scontrol show job 2
 JobId=2 JobName=sleep
    UserId=root(0) GroupId=root(0) MCS_label=N/A
    Priority=1 Nice=0 Account=root QOS=normal
    JobState=RUNNING Reason=None Dependency=(null)
    Requeue=1 Restarts=0 BatchFlag=0 Reboot=0 ExitCode=0:0
    RunTime=00:00:13 TimeLimit=UNLIMITED TimeMin=N/A
    SubmitTime=2018-03-09T16:15:06 EligibleTime=2018-03-09T16:15:06
    StartTime=2018-03-09T16:15:06 EndTime=Unknown Deadline=N/A
    PreemptTime=None SuspendTime=None SecsPreSuspend=0
    Partition=defq AllocNode:Sid=controller1:18804
    ReqNodeList=(null) ExcNodeList=(null)
    NodeList=node001
    BatchHost=node001
    NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
    TRES=cpu=1,node=1
    Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
    MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
    Features=(null) DelayBoot=00:00:00
    Gres=(null) Reservation=(null)
    OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
    Command=sleep
    WorkDir=/trinity/shared/etc/slurm
    Power=
```

For more information about SLURM commands and slurm config please have a look at official documentation

2.3 User manual

2.3.1 Hints and tips for SLURM

Basic operations

Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. A good starting point to learn more about SLURM is the official site:

```
https://slurm.schedmd.com/quickstart.html
```

Paired with munge (for secure communication), SLURM provides scheduling facilities in TrinityX. It allows use of the cluster simultaneously by multiple users without affecting each others' jobs. The simplest way of getting access to a node in SLURM is to issue the following:

```
$ srun --nodelist=node001 hostname
node001.cluster
```

A list of all available nodes and partitions can be inspected in `sinfo` output.

In this quick example, node001 was allocated, the ssh client connected to node001, and the `hostname` command was issued on node001. Another way of running commands is to allocate resources first and then execute srun:

```
$ sallocate --nodelist=node001,node002
$ srun hostname
node001.cluster
node002.cluster
```

During allocation, the status of nodes can be viewed in the `squeue` output:

```
$ squeue
JOBID PARTITION     NAME        USER ST       TIME  NODES NODELIST(REASON)
 3439      defq     bash   cvsupport  R       0:02      2 node[001-002]
```

In most cases, the output above means that nodes are exclusively 'owned' by a user and no other job or user within SLURM can use these nodes to compute their jobs. However, it might be not true if a SLURM cluster is configured in shared mode.

If SLURM is unable to allocate resources, it will queue the request to wait:

```
$ salloc --nodelist=node001,node002
salloc: Pending job allocation 3440
salloc: job 3440 queued and waiting for resources

$ squeue
JOBID PARTITION     NAME        USER ST       TIME  NODES NODELIST(REASON)
 3440      defq     bash   cvsupport PD       0:00      2 (Resources)
```

The `ST` column shows the status of job allocation. For example, `R` is for running and `PD` is for pending. Other codes can be found in `man squeue`.

Specifying the list of nodes for jobs is not good practice, as one must be sure the nodes are available. A better approach is to specify a partition to run and a quantity of nodes:

```
$ srun --partition=defq --nodes=2 hostname
node004.cluster
node005.cluster
```

Partitioning is a method of organizing nodes in cluster. Usually all nodes in a given partition are homogeneous, i.e. have the same hardware configuration, same software installed, and access to the same resources, like shared filesystems.

Using sbatch

`srun` and `salloc` commands are useful when running interactive jobs. For long-running tasks, `sbatch` comes into play. `sbatch` allows a user to submit a job file into the queuing system. A job file is usually the ordinary shell script file with directives for SLURM. Directives start with `#SBATCH` and are usually located in the beginning of the job file. A job file may be submitted without any directives and SLURM will apply some defaults: i.e. allocate single node, put the job to a default partition, etc. Usually it is worthwhile to change such behaviour. Here is an example of a basic script:

```
#!/bin/bash
#SBATCH --partition=defq
#SBATCH --nodes=2
hostname
```

To submit a job, set content above to the `test01.job` file and simply run:

```
$ sbatch test01.job
```

Please note that you might not have a defq partition configured in your cluster. Check the `sinfo` output.

After a job finishes, its output will appear in the home directory, titled `slurm-3443.out`, in which 3443 is a job number.

If the job failed for some reason, the file `slurm-3443.err` will be created. The first file - `.out` - contains STDOUT from job script and `.err` has STDERR content. The path and name of these files can be customized:

```
#SBATCH --output=/path/to/store/outputs/myjob-%J.out
#SBATCH --error=/path/to/store/outputs/myjob-%J.err
```

The job number will be substituted instead of the %J variable. For more variables, please have a look at `man sbatch`.

By default, a job assumes that the current working directory is a home dir of the user. It can be customized, specifying `--workdir=`:

```
#SBATCH --workdir=/new/home/dir/
```

In addition, you can specify the number of nodes, dependencies, starting time, and change many other tunables. All are described in `man sbatch`.

Variables in job scripts

During job execution, SLURM provides several environment variables. For logging purposes, it can be useful to tune a job to render those valuables in its output.:

```
#!/bin/bash
#SBATCH --partition=defq
#SBATCH --nodes=2

echo "Job is running on ${SLURM_JOB_NUM_NODES} nodes"
echo "Allocated nodes are: ${SLURM_JOB_NODELIST}"
```

The output will contain:

```
$ cat slurm-3444.out
Job is running on 2 nodes
Allocated nodes are: node[001-002]
```

In addition, more than 100 variables are available. For reference, please run `man sbatch`.

Srun and mpirun in job scripts

Usually, it is unnecessary to use srun in job scripts. Spawning multiple copies of a binary is usually performed by mpi library. To get an idea of how things are working in an sbatch context you can check on the following output:

```
#!/bin/bash
#SBATCH --partition=defq
#SBATCH --nodes=2

echo "======= hostname: ======="
hostname
echo "======= srun hostname: ======="
srun hostname
echo "======= mpirun hostname: ======="
module load openmpi/2.0.1
mpirun hostname
```

Please note that the `module load` line might differ in your environment.

The output will be similar to:

```
$ cat slurm-3447.out
======= hostname: =======
node001.cluster
======= srun hostname: =======
node001.cluster
node002.cluster
======= mpirun hostname: =======
node001.cluster
node001.cluster
node001.cluster
node001.cluster
node002.cluster
node002.cluster
node002.cluster
node002.cluster
```

The number of mpirun hostnames depends on the number of cores in the nodes.

Running MPI application. Example

To be concrete, let's take MPI Hello Word from MPI Tutorial and put it to mpi-hello.c:

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
        // Initialize the MPI environment
        MPI_Init(NULL, NULL);

        // Get the number of processes
        int world_size;
        MPI_Comm_size(MPI_COMM_WORLD, &world_size);

        // Get the rank of the process
        int world_rank;
        MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

        // Get the name of the processor
        char processor_name[MPI_MAX_PROCESSOR_NAME];
        int name_len;
        MPI_Get_processor_name(processor_name, &name_len);
```

```
        // Print off a hello world message
        printf("Hello world from processor %s, rank %d"
                    " out of %d processors\n",
                    processor_name, world_rank, world_size);

        // Finalize the MPI environment.
        MPI_Finalize();
}
```

Now, compile the application with one of the MPI versions installed on the cluster:

```
$ module load openmpi/2.0.1
$ mpicc -o mpi-hello.bin mpi-hello.c
```

Create job file:

```
#!/bin/bash
#SBATCH --partition=defq
#SBATCH --nodes=2

module load openmpi/2.0.1
mpirun mpi-hello.bin
```

And run it:

```
$ sbatch test03.job
```

In the output file, something like the following will appear:

```
Hello world from processor node001.cluster, rank 2 out of 4 processors
Hello world from processor node001.cluster, rank 1 out of 4 processors
Hello world from processor node001.cluster, rank 0 out of 4 processors
Hello world from processor node001.cluster, rank 3 out of 4 processors
Hello world from processor node002.cluster, rank 1 out of 4 processors
Hello world from processor node002.cluster, rank 3 out of 4 processors
Hello world from processor node002.cluster, rank 0 out of 4 processors
Hello world from processor node002.cluster, rank 2 out of 4 processors
```

You are done! You have now created and run your first MPI application on the HPC cluster.

2.4 Docker documentation

If the configuration with Docker is used, the following guide provides information on running jobs.

2.4.1 Running docker apps on a cluster

TrinityX comes with the ability to run arbitrary dockerized MPI jobs on a cluster.

Note Please consult the official docker documentation for more information on Docker and how it works.

The only requirement to be able to run such jobs is to have openssh-server running in the container and listening on the non-standard port 2222.

Keep in mind that the application to be run as an MPI job must be installed or pre-compiled in the container image. All of its dependencies need to be installed/pre-compiled as well.

Building a Docker image

To be able to run the dockerized MPI job you need first to provide it to the cluster as a Docker image.

To do so, two options are available:

Building on the controller

A TrinityX controller comes pre-installed with Docker and Docker-registry (assuming that the docker option was selected at install time). This makes it possible for an administrator to create a Docker image that can subsequently be run on the cluster.

It is worth mentioning here that regular users cannot issue Docker commands unless permitted by the admins.

With that cleared up, let's build a Docker image that we can then use to run an OSU MPI benchmark:

1. First, create an empty directory to serve as the workdir.
2. Create a special file called *Dockerfile* that will be used to build the Docker image; an example is provided below:

```
FROM centos:latest

RUN yum -y install epel-release && \
    yum -y install openssh-server openssh-clients\
                   environment-modules \
                   openmpi openmpi-devel \
                   libibverbs librdmacm \
                   make gcc-c++ && \
    yum clean all

RUN echo "module load mpi" > /etc/profile.d/openmpi.sh && \
    chmod +x /etc/profile.d/openmpi.sh

RUN curl -O
http://mvapich.cse.ohio-state.edu/download/mvapich/osu-micro-benchmarks-5.3.2.tar.gz
 && \
    tar xf osu-micro-benchmarks-5.3.2.tar.gz -C /root/

RUN bash -lc "cd /root/osu-micro-benchmarks-5.3.2; ./configure CC=mpicc; make;
 make install"

ENTRYPOINT ["/usr/sbin/sshd", "-p", "2222", "-D"]
```

As you can see, this Dockerfile satisfies the TrinityX requirements as it sets up openssh-server correctly. It also installs all of the dependencies required to run the OSU benchmarks.

1. We then can build our image using *docker build*:

```
# docker build -t <controller-hostname>:5000/osu .
```

Make sure to replace <controller-hostname> with the correct hostname.

1. Lastly, we need to publish our image so that the compute nodes can fetch it when required:

```
# docker push <controller-hostname>:5000/osu
```

Using a remote Docker registry

If you prefer to build your images elsewhere and store them on a Docker registry other than the one provided by TrinityX, then it will be necessary to update the compute images.

Since compute nodes will need to query a remote Docker registry for Docker images, this one needs to be declared in */etc/sysconfig/docker* in your compute images.

It can be updated using *lchroot*.

Job scripts

Now that the image is ready, we need to create a job script to run our dockerized OSU benchmarks.

A job script would need to include the following commands (besides the usual directives):

- A Docker image name:

```
export DOCKER_IMAGE="<docker_image_to_run>"
```

- An executable application:

```
export APPLICATION="<mpi_app>"
```

And optionally:

- A list of shared folders and mount points:

```
export
DOCKER_SHARES="/host/share-1:/docker/mnt-1;/host/share-2:/docker/mnt-2;"
```

- A working directory to which mpi-drun will switch before launching the MPI application:

```
export DOCKER_WORKDIR="</path/to/work/dir>"
```

- A set of options to pass on to the underlying MPI implementation:

```
export MPI_OPTS="<options>"
```

Then finally:

- The `mpi-drun` command

Following is an example that can be used to run our previous OSU image:

```
#!/bin/bash

#SBATCH --partition=defq
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --job-name="osu-docker"

export DOCKER_IMAGE="<controller-hostname>:5000/osu"
export DOCKER_WORKDIR="/usr/local/libexec/osu-micro-benchmarks/mpi/pt2pt"
export APPLICATION="osu_latency"
export MPI_OPTS="-np 2 -mca orte_base_help_aggregate 0"
mpi-drun
```

Then, as a user, you can submit the job using sbatch:

```
sbatch job.sh
```

C

controller.yml, 10

S

site.yml, 10

C

controller.yml, 10

S

site.yml, 10